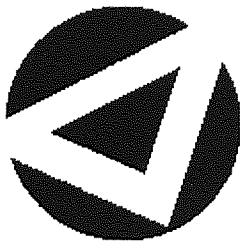
A Combined Approach for Hiding Partial Information in RSA

ALEXANDROS PAPANIKOLAOU

Doctor of Philosophy



ASTON UNIVERSITY

October 2006

ASTON UNIVERSITY

A Combined Approach for Hiding Partial Information in RSA

ALEXANDROS PAPANIKOLAOU

Doctor of Philosophy, 2006

Thesis Summary

Partial information leakage in deterministic public-key cryptosystems refers to a problem that arises when information about either the plaintext or the key is leaked in subtle ways. Quite a common case is where there are a small number of possible messages that may be sent. An attacker may be able crack the scheme simply by enumerating all the possible ciphertexts. Two methods are proposed for facing the partial information leakage problem in RSA that incorporate a random element into the encrypted message to increase the number of possible ciphertexts. The resulting scheme is, effectively, an RSA-like cryptosystem which exhibits probabilistic encryption.

The first method involves encrypting several similar messages with RSA and then using the Quadratic Residuosity Problem (QRP) to mark the intended one. In this way, an adversary who has correctly guessed two or more of the ciphertexts is still in doubt about which message is the intended one. The cryptographic strength of the combined system is equal to the computational difficulty of factorising a large integer; ideally, this should be infeasible.

The second scheme uses error-correcting codes for accommodating the random component. The plaintext is processed with an error-correcting code and deliberately corrupted before encryption. The introduced corruption lies within the error-correcting ability of the code, so as to enable the recovery of the original message. The random corruption offers a vast number of possible ciphertexts corresponding to a given plaintext, hence an attacker cannot deduce any useful information from it.

The proposed systems are compared to other cryptosystems sharing similar characteristics, in terms of execution time and ciphertext size, so as to determine their practical utility. Finally, parameters which determine the characteristics of the proposed schemes are also examined.

Keywords: RSA, partial information, quadratic residuosity, noise, public-key cryptography, error-correcting codes

To my parents, Nikos and Anna,
to my friend, Apostolis

Acknowledgements

I wish to thank Dr Song Y. Yan for inspiring me to carry out this research under his supervision, but I mostly thank Professor Ian T. Nabney who offered to supervise me in the final stage of this work and bring it to an end, when Dr Yan left Aston University. I also wish to thank Mr Bernard S. Doherty for giving very useful feedback on various aspects of my research topic.

At this point I would like to thank the Engineering and Physical Sciences Research Council (EPSRC) for granting me a research studentship, which covered the tuition fees of this course. I am also very grateful to the research administrator, Miss Vicky Bond, for being extremely helpful and for dealing with all the administrative issues that arose within this time period.

I am also very thankful to Professor David Saad and Professor David J.C. MacKay for answering my questions on the field of error-correcting codes, as well as Bassem Ammar for supplying some source code for simulations.

Last, but not least, I would like to thank all my friends and colleagues who made my 3-year stay as pleasant as possible and especially Steven Brockie, Lehel Csató, Wei Lee Woon, Yi Sun, Sterghios Moschos, Christos Mallios, Giorgos Chrysochoos, Vangelis Kozanitis, Elena Kakouri, Haris Charalampopoulos, Kostas Zygogiannis and Maria Ioannou.

Publications Related to the Thesis

[1] Alexandros Papanikolaou and Song Y. Yan. Prime Number Generation Based on Pocklington's Theorem. *International Journal of Computer Mathematics*, 79(10):1049–1056, 2002. Taylor & Francis.

[2] Alexandros Papanikolaou. A Combined Approach for Hiding Partial Information in RSA. In IADAT, editor, *IADAT-tcn2005 – International Conference on Telecommunications and Computer Networks*, Technological Advances in Telecommunications and Computer Networks, pages 192–196. IADAT, September 2005. ISBN: 84-933971-7-2.

[3] Alexandros Papanikolaou. Using LDPC Codes for Hiding Partial Information in RSA. Submitted to *IEEE Transactions on Information Theory*.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols / Abbreviations

$\in$	Exists in set
$GF(q)$	Galois Field with a finite number of elements q, where $q = p^m$ with p prime and m positive
$\gcd(a, b)$	The greatest common divisor of a and b
$\mathcal{O}(N)$	Order of complexity, measured in terms of arithmetic operations
$\mathcal{O}(\exp(x))$	Exponential complexity (also denoted by $\mathcal{O}(e^x)$)
$x \equiv y \pmod{n}$	x is congruent to y modulo n
$\left(\frac{a}{p}\right)$	Legendre symbol, where p is prime
$\left(\frac{a}{n}\right)$	Jacobi symbol, where n is composite
Q_n	Set of all quadratic residues of n
$\overline{Q}_n$	Set of all quadratic non-residues of n
$\tilde{Q}_n$	Set of all pseudosquares of n
J_n	Set of elements in $\mathbb{Z}_n^*$ with Jacobi symbol $\left(\frac{a}{n}\right) = 1$
$\lvert\mathcal{S}\rvert$	Cardinality of set $\mathcal{S}$
$a \mid n$	a divides n
$a \nmid n$	a does not divide n
$\phi(n)$	Euler's totient function
$\mathbb{Z}/n\mathbb{Z}$	Residue classes modulo n (also denoted by $\mathbb{Z}_n$)
$(\mathbb{Z}/n\mathbb{Z})^*$	The multiplicative group of the elements in $\mathbb{Z}/n\mathbb{Z}$ (also denoted by $\mathbb{Z}_n^*$)
$\wedge$	Bitwise AND

$\vee$	Bitwise OR
$\neg$	Bitwise NOT
$\oplus$	Bitwise Exclusive-OR
$\lll$	Left circular bit shift
$\forall$	For all
$\subset$	Subset
$\|n\|$	Size of n in bits (effectively, $\log_2 n$)
$\lceil x \rceil$	Ceiling: The least integer greater than or equal to x
$\lfloor x \rfloor$	Floor: The greatest integer less than or equal to x
$a \mid\mid b$	Concatenation of strings a and b
$P(A \mid B, C)$	The probability of A, given B and C
$a \leftarrow b$	Set a to the value of b
$\mathbb{1}[S]$	The truth function, which is 1 if the proposition S is true and 0 otherwise
$\propto$	Proportional to
A^T	The transpose matrix of A
$\mathbf{I}_K$	The $K \times K$ identity matrix
BSC	Binary Symmetric Channel
CRHF	Collision Resistant Hash Function
LDPC	Low-Density Parity-Check
MAC	Message Authentication Code
MDC	Modification Detection Code (or Manipulation Detection Code)
MGF	Mask Generation Function
MIC	Message Integrity Code
OAEP	Optimal Asymmetric Encryption Padding
OWHF	One-Way Hash Function
RFC	Request For Comments
RSAES-OAEP	RSA Encryption Standard –

	Optimal Asymmetric Encryption Padding
RSASSA-PSS	RSA Signature Scheme with Appendix – Probabilistic Signature Scheme
PKCS #1	Public-Key Cryptography Standards #1
DLP	Discrete Logarithm Problem
QRP	Quadratic Residuosity Problem
RNG	Random Number Generator
OR	Logical OR operation
XOR	Exclusive-OR operation (context dependent)

Chapter 1

Introduction

1.1 Motivation

The need for secure communications does not only emerge from security-critical cases, but also from the need for privacy when using various means of daily communication. In order to establish secure communication between two parties, the use of encryption is vital. Prior to transmission, the sensitive information is transformed into an incomprehensible form, so that only legitimate receivers can reverse the process and recover the original message.

The invention of *symmetric* (*secret-key*) cryptography successfully addressed the need for secure communication between two parties that shared the same secret key. Like all cryptographic algorithms it converts the secret message (*plaintext*) into an encrypted form (*ciphertext*), using the secret information provided by the key. The same key is needed to decrypt the ciphertext and recover the original plaintext. Thus it is necessary to ensure that both parties have the secret key without anyone else gaining access to it. An alternative way for communicating secretly is the use of *steganography*, that focuses on hiding the existence of the communication itself, rather than obscuring it (steganography is further discussed in Section 3.1).

The evolution of the Internet and the expansion of computer networks increased the

need for multi-party secure communications. Symmetric ciphers were no longer good enough, as they suffered from key management and key distribution problems. These were successfully overcome with the introduction of *asymmetric* (*public-key*) ciphers, where each communicating party possesses two keys: a *private key* known only to himself and a *public key* available to everyone. The sensitive information is encrypted using the receiver's public key and can only be decrypted by the corresponding private key. Furthermore, there is no way of deducing or computing the private key from the public one.

Asymmetric ciphers also introduced the concept of *digital signatures*, which can be used to ensure authentication, non-repudiation and message integrity. In order to create a digital signature the roles of the two keys are swapped. Hence, encryption is performed using the secret key and decryption using the public one. Upon successful decryption, the receiver is convinced that the other party is indeed the owner of the secret key.

The execution time of asymmetric ciphers is significantly greater than that of symmetric ciphers, as they require significantly larger keys to offer comparable cryptographic strength. This problem can be solved with the use of *hybrid systems*. That is, the message is encrypted with a symmetric cipher, using a *session key*. The latter is securely conveyed from the sender to the receiver using a public-key cipher (Schneier 1996). This technique is also known as *digital enveloping*.

Cryptographic algorithms can further be divided into *deterministic* and *probabilistic*. The former always convert a given plaintext to the same ciphertext. The latter can produce several ciphertexts that correspond to the same plaintext, if encryption is performed at different points in time.

As there is clearly a need for public-key ciphers, even for transmitting a session key securely, the use of a deterministic asymmetric cryptosystem may compromise security. This is due to partial information leakage, which can be exploited when there is a reasonably limited set of possible messages. Hence, a passive eavesdropper can make

guesses of the possible messages, compute the encryptions of these guesses, compare them to the intercepted ciphertext and in case of a match, the original message can be deduced. A preference for probabilistic public-key cryptography has therefore been established.

One other desirable property is that of small message expansion. Namely, the ciphertext corresponding to a given plaintext should be as short as possible. This is the main reason why it is sometimes preferred to convert a deterministic cipher into a probabilistic one using appropriate additions/modifications, rather than to use a probabilistic cipher in the first place. Moreover, well-studied algorithms are preferred to newer ones, as there is less chance that unforeseen security flaws will emerge in the future.

The security of most public-key cryptosystems is based on some computationally hard number-theoretic problem, such as large integer factorisation and the discrete logarithm problem. In both cases prime numbers are used, hence the generation of large primes has become vital for public-key cryptography. While finding the prime factors of a large integer is a computationally hard task, testing for primality is much easier. The preferred way for generating prime numbers is to generate a large integer of the desired size and then test its primality using one of the available tests. Such tests may be probabilistic, namely a number that has passed this test may actually prove to be a composite, although the probability of that to occur is negligible.

Alternatively, deterministic primality tests can be used that will guarantee the primality of a given integer. However, one of their main drawbacks is that they are significantly harder to implement than probabilistic ones. Furthermore, at some point they may require the factorisation of a number related to the one under question, which may prove to be equally hard to compute.

The contribution of error-correcting codes to cryptography is also significant. The role of an error-correcting code is to enable the complete recovery of the original data that may have been corrupted due to transmission through a noisy communication

channel. The data is expanded in a systematic way with extra information that supports complete recovery of the original data. Error-correcting codes can be classified in many categories according to their properties, but this is beyond the scope of this thesis. Low-density parity-check (LDPC) codes are used as an instance of error-correcting codes for the purpose of this work, although any error-correcting code would have been equally suitable for it. This choice is further supported by the availability of source code for implementing encoding and decoding with LDPC codes (MacKay & Neal 1995) that could be tailored to meet the requirements of this thesis. Furthermore, since they have good error-correcting properties, the resulting cryptosystem is expected to exhibit a good level of security. Some more information on LDPC codes is presented in Section 3.9.2.2.

At this point it is helpful to make a distinction regarding the terminology used throughout this thesis. Hence, the terms *encryption* and *decryption* refer to the corresponding cryptographic transformations, whereas the terms *encoding* and *decoding* relate to the operations of error-correcting codes (unless otherwise stated).

The first cryptosystem devised that utilised the properties of error-correcting codes was derived by McEliece (1978), who used Goppa codes for achieving his goal. In particular, permutation and scrambling matrices form the private key. They are then multiplied with the code's generator matrix to create the public key. A variant of this cryptosystem was proposed by Niederreiter (1986). It differs from the McEliece one in that the public key is formed by multiplying the parity-check matrix of a Goppa code with permutation and scrambling matrices (some more information on the Niederreiter cryptosystem can be found in Section 3.9.3).

Similar systems have been proposed more recently, such as the one by Kabashima, Murayama & Saad (2000). The main drawback of these *matrix-based* (also called *lattice-based*) cryptosystems is the very large size of the public key. Despite recent improvements that have reduced dramatically the public key size (Gabidulin, Ourivski, Ammar & Honary 2003), it is still quite large compared to other public-key algorithms

that offer comparable security. Nevertheless, since public keys are transmitted only once and are usually not changed frequently, matrix-based cryptosystems may still be of practical use.

The original implementation of RSA proposed by Rivest, Shamir & Adleman (1978) is a deterministic scheme. That is, if the same plaintext is encrypted with the same key, it will always yield the same ciphertext, which creates the *information leakage* problem. Shoup (1998) argued that the original RSA cryptosystem is leaking partial information and gave the following example as justification.

Suppose that Alice wants to send orders to her stock broker, Bob; Eve would like to know what the order is. Furthermore, Eve has a good reason to believe that the message, m, is one of just three possibilities:

- m_1 = 'Sell'
- m_2 = 'Buy'
- m_3 = 'Hold'

Eve can compute the encryptions c_1, c_2 and c_3 of these three messages, since Bob's encryption key is public. When Alice sends Bob the encryption of her message, say c_2, then Eve will know that the original message was m_2. This attack is also known as *forward search attack*. This problem faces any deterministic public-key cryptosystem when the number of messages is small enough so that all can be encrypted by an attacker.

RSA is probably the most popular and best studied asymmetric, deterministic cryptosystem. Its main advantages are the very low message expansion (by a factor of 1), as well as its ease of implementation. The superiority of probabilistic cryptosystems is widely accepted by the cryptographic community, since probabilistic schemes usually prevent partial information leakage. Hence, many attempts have been made to convert RSA into a probabilistic scheme, by inserting some randomness during the plaintext pre-processing phase, before encryption commences. However, it has not

yet been investigated whether randomisation could be performed after encryption has commenced. The process will involve blending several ciphertexts, as well as defining a suitable means for distinguishing the intended ciphertext from the rest.

Furthermore, since coding theory has made significant contributions to cryptography, the feasibility of exploiting error-correcting codes for hiding partial information will also be looked into. In particular, the ability of a codeword to withstand random transmission noise will be exploited, in an attempt to randomise the plaintext. The two techniques mentioned above should be evaluated in terms of both computational complexity and message expansion, in order to determine their practicality.

Finally, since the generation of prime numbers is vital for public-key cryptography, we shall look at a new method for generating primes using Pocklington's theorem and compare it with existing techniques.

1.2 Aims of the Thesis

1. To develop a method for transforming RSA into a probabilistic cryptosystem, where the random component (that is, the opaque blocks) is inserted after the encryption process.

2. To develop a method for transforming RSA into a probabilistic cryptosystem, where error-correcting codes are used for accommodating the random component.

   For each of the above two methods, it is also required:

   (a) To develop a framework for evaluating the performance of the end systems, as well as their security level.

   (b) To develop prototypes for experiments and testing, that will be able to withstand an adaptive chosen-ciphertext attack and compete the performance of existing schemes.

   A subsidiary aim is:

3. To evaluate the usefulness and practicality of Pocklington's theorem for generating large primes.

It is worth clarifying that throughout this thesis the term *performance* is associated with the ability to perform operations of encryption and decryption efficiently, by utilising low processing time. In spite of the fact that *message expansion* is usually related to the performance of a cryptosystem, it will be examined as a separate property. What is more, the term *security level* is associated with the ability of a cryptosystem to withstand all applicable types of cryptanalytic attacks.

The proposed methods should also be extensible to take advantage of the evolution of cryptographic technology and availability of computational power. That is, it should be feasible for the proposed techniques to be applied to a newly created, deterministic cryptosystem, provided that some requirements are met. Also, the continuous increase of both the magnitude and availability of computational power poses a need for larger keys in the near future. Both the information rate and the security level of the proposed systems should therefore not deteriorate if larger keys are used. On the contrary, an increase in these measures would be desirable.

More analytically, the first aim of the thesis requires a method for distinguishing the random component from the real ciphertext, so as to enable the successful retrieval of the original message. What also needs to be determined is both the character and the quantity of the introduced randomness. As far as quality is concerned, the relation of the random component to the true ciphertext should be determined. That is, the two could be contextually related or not. Regarding the quantity, what needs to be resolved is how large the random component inserted into a particular ciphertext should be. It is quite obvious that this will have a direct effect on both the security and the performance of the system, which pertains to the second aim of the thesis.

It is quite obvious that the more randomness is inserted in the ciphertext, the higher the achieved security level will be. Nevertheless, it will require more computational overhead, both in the encryption and the decryption phase, and will reduce the amount

of the plaintext that is effectively transmitted. A method is therefore required for indicating to the communicating party the provided level of security and the expected performance, based on the parameters they choose.

As far as the second aim is concerned, it should be determined whether specific characteristics of the error-correcting code exist (the codeword size, for instance) that can affect the system's performance and/or the security level. Should this prove to hold, a measure needs to be established for approximating the security level of the end system with respect to all these parameters.

For evaluating the security of a system two approaches can be followed: the theoretical and the practical one. The former involves producing formal mathematical proofs, which may be generalised in the future and serve as a guideline for future developers of similar systems. The latter one is more empirical; it involves examining the practical capabilities of an attacker, identifying the available attacks and then assess the security of the system against these attacks.

Since most public-key schemes base their security on well-known number-theoretic problems, in most cases it suffices to show that the systems are secure against an adaptive chosen-ciphertext attack, where the attacker is allowed to decrypt several (valid) ciphertexts and modify his choice based on the results he gets. What is more, since the attacker is assumed to have full knowledge of the cryptosystem in question, one should examine the available options he has, given the available information.

The requirement of the next aim is two-fold. The end systems should be secure against an adaptive chosen-ciphertext attack, since this is regarded as a primary requirement for public-key cryptosystems currently. The second part is about having a reasonable execution time, in relation to existing schemes that share similar characteristics.

The last aim deals with the checks that Pocklington's theorem needs to perform before reaching a conclusion regarding the primality of an integer. Any implementation difficulties should be identified and ways for overcoming them, if any, will be proposed.

Additionally, the execution speed of such an implementation needs to be evaluated, in order to determine the practicality of this technique.

Finally, a system with a Graphical User Interface would help in ensuring that all the required parameters are within the desired range and would offer some basic key management. The end system is expected to consist of several components that will interact among them through file I/O. What is more the keys required for the operations of encryption and decryption will be stored in different files. The use of a GUI will therefore shield against passing the wrong arguments in the various sub-components, thus leading to a much less error-prone program execution. Consequently, such a system would help with dissemination to the research community and would aid its further development.

1.3 Research Methodology

1.3.1 Literature Review

- Cryptography

  Some basic number-theoretic issues and complexity theory are investigated, to understand the foundations of public-key cryptography. Pocklington's theorem is studied, as it can aid the generation of prime numbers. The area of cryptography is then considered, with a focus on asymmetric cryptography. In particular, the RSA algorithm and the probabilistic encryption schemes of Rabin (1979), Goldwasser & Micali (1984) and Blum & Goldwasser (1985) are reviewed, since the first proposed scheme is related to these systems. Finally, digital signatures and hash functions are investigated because they can be used for detecting possible tampering of the ciphertext.

- Error-correcting codes

  The fundamental principles of error-correcting codes are also investigated, to

reveal their basic way of operation and the range of applications. Their contribution and interaction with cryptology is examined and characteristic cryptosystems from this domain are reviewed. More specifically, the McEliece (1978) cryptosystem is looked at, an improvement to it by Gabidulin et al. (2003) and a variant of this cryptosystem by Kabashima et al. (2000). Finally, a method proposed by Courtois, Finiasz & Sendrier (2001) for creating small digital signatures with the McEliece cryptosystem is also looked into.

1.3.2 Analysis and Investigation

Analysis focuses on determining the parameters and/or configurations that would shield the system against an adaptive chosen-ciphertext attack, as well as a means for measuring the required computational effort for launching (if applicable) a successful brute force attack. At the same time, the effect of these parameters on the performance of the system is examined, in an attempt to establish a good balance between security and performance. Information found in the literature was used for developing new methods, which are then evaluated by reproducing or even by extending published results.

1.3.3 Prototyping

Two prototypes were implemented for carrying out the proposed tests, using a combination of the C and C++ programming languages, due to the following reasons:

- The multiple precision integer library that was chosen for manipulating large integers (namely the keys) was written in C++ (Diommisfois 1988). That particular library was chosen because it offers *operator overloading*, a desirable feature.
- The software chosen for encoding and decoding data using LDPC error-correcting codes, as well as for simulating transmission through a noisy channel, was written in C (MacKay & Neal 1995).

The required interfacing of the heterogeneous components was achieved via file I/O. Furthermore, implementations of the Goldwasser–Micali probabilistic encryption scheme, the Rabin probabilistic cryptosystem and RSA were also developed, so as to provide the required data for comparison purposes.

1.3.4 System Evaluation

A theoretical approach was followed for testing the security of the two proposed systems. In the first one it is shown that it can easily be adjusted to prevent an adaptive chosen-ciphertext attack from taking place. Regarding the second system, the right choice of parameters can shield against the attack.

Experimental data was gathered by performing the operations of encryption and decryption with both systems, using specific inputs and a range of key sizes. These measurements involved both execution time and ciphertext size. Statistical tests were used to evaluate the quality of the obtained results in an attempt to justify the predictability of the systems' performance. They were then compared to implementations of RSA, the Goldwasser–Micali probabilistic encryption scheme and the Rabin probabilistic cryptosystem, so as to draw useful conclusions on the practicality of the proposed systems.

Graphical representations of the results were used to visualise the experimental data with respect to some parameters.

1.4 Novel Features of the Thesis

The novel aspects of this thesis are:

- A method for converting RSA into a probabilistic cryptosystem that
  - inserts the probabilistic component (opaque blocks) after the encryption process and uses the Quadratic Residuosity Problem for retrieving the original message (Chapter 4).

  - is also applicable to other deterministic public-key cryptosystems.
  - can easily prevent an adaptive chosen-ciphertext attack, when appropriately configured.
  - can be used for creating steganographic systems.

- A method for converting RSA into a probabilistic cryptosystem that
  - uses low-density parity check error-correcting codes for accommodating the random component, which is inserted before encryption takes place (Chapter 5).
  - is also applicable to other deterministic public-key cryptosystems.
  - offers security against an adaptive chosen-ciphertext attack, with an appropriate choice of its operational parameters.

- An assessment of the usability of Pocklington's theorem for generating true primes that can be used for forming the key of an asymmetric encryption scheme (Chapter 2).

1.5 Outline of the Thesis

The thesis is organised as follows: Chapter 2 presents the fundamentals of complexity theory, thus providing the necessary background required for understanding public-key cryptography and prime number generation techniques and an extensive study on the exploitation of Pocklington's theorem as a prime number generation technique. Chapter 3 presents the fundamentals of cryptography, focusing on asymmetric cryptography. Some cryptosystems are presented in detail, as well as hash functions and digital signatures. Cryptosystems based on error-correcting codes are also introduced, in addition to the relevant background. Chapter 4 describes the first proposed method, together with ways of evaluating it. Experimental data gathered after the developed prototype was run is also presented, together with interpretation of results and other findings.

In Chapter 5 the second method is presented, together with the experimental data, that was obtained by running the respective prototype. Finally, the conclusions and proposals for future research are presented in Chapter 6.

Chapter 2

Number-Theoretic Issues

2.1 Complexity Theory

2.1.1 Introduction

Complexity theory provides a methodology for analysing the relationship between number of steps of cryptographic techniques and algorithms and the size of inputs, thus providing a measure of their security. According to information theory, all cryptographic algorithms can be broken, with the exception of one-time pads. By applying complexity theory, the time required for breaking a specific algorithm can be estimated; cryptosystems which take a long time to decrypt can be regarded as being secure in practice.

2.1.2 Complexity of Algorithms

The complexity of an algorithm measures the number of computational steps needed to execute it. As Schneier (1996) mentions in his book, the computational complexity of an algorithm is often measured by two variables: T (for the time complexity) and S (for space complexity, or memory required). They are commonly expressed as functions of n, where n is the size of the input. The 'big O' notation is used, which denotes the

order of magnitude of the computational complexity. Hence, an algorithm has *constant* complexity, if its complexity is independent of n ($\mathcal{O}(1)$), *linear* if its time complexity is $\mathcal{O}(n)$, *quadratic* if its time complexity is $\mathcal{O}(n^2)$ and so on. All these measures are *polynomial*, thus having a complexity of $\mathcal{O}(n^m)$, where m is a constant. Algorithms whose complexities are $\mathcal{O}\left(t^{f(n)}\right)$, where t is a constant greater than 1 and $f(n)$ some polynomial function of n, are called *exponential*. The subset of exponential algorithms whose complexities are $\mathcal{O}\left(c^{f(n)}\right)$, where c a constant and $f(n)$ is more than constant but less than linear, is called *superpolynomial*.

2.1.3 Fast (Modular) Exponentiation

A great number of public-key cryptosystems, including the most popular ones, implement the operations of encryption and decryption via modular exponentiations. In order to compute $b^x \bmod n$ using the conventional method of *repeated multiplication*, would require $\mathcal{O}\left((x \log_2 n)^2\right)$ bit operations, which is far too much if large integers are involved. However, using a fast modular exponentiation technique, the number of the required bit operations can be reduced to $\mathcal{O}\left((\log_2 x \cdot \log_2 n)^2\right)$. Such techniques are usually referred to as *repeated squaring and multiplication methods*, named after the way they operate.

Algorithm 2.1 presents the required steps for performing a fast modular exponentiation. It should be noted that this algorithm can be applied without the modulo n computation in the calculation of c, thus leading to a fast exponentiation algorithm.

Algorithm 2.1 Computing $c \equiv b^x \bmod n$

1: Convert the exponent into its binary form: $x = x_k x_{k-1} \ldots x_0$.
2: Set $c \leftarrow 1$
3: **for** i from $k-1$ down to 0 **do**
4: $\quad c \leftarrow c^2 \bmod n$
5: $\quad$ **if** $x_i = 1$ **then**
6: $\quad\quad c \leftarrow cb \bmod n$
7: $\quad$ **end if**
8: **end for**
9: Output c and terminate the algorithm

The method described above is also known as *left-to-right binary exponentiation.* By modifying the algorithm slightly, so that the binary representation of the exponent is processed from right to left, it leads to the variant of the *right-to-left binary exponentiation.* Brickell, Gordon, McCurley & Wilson (1992) developed a more efficient algorithm, using pre-computed values to reduce the number of multiplications needed. It is claimed that the algorithm provides a substantial improvement over the level of performance, allowing to compute g^n for $n < N$ in time $\mathcal{O}(\log_2 N / \log_2 \log_2 N)$. What is more, they showed that the method can be parallelised, to compute powers in time $\mathcal{O}(\log_2 \log_2 N)$ with $\mathcal{O}(\log_2 N / \log_2 \log_2 N)$ processors.

2.2 Computationally Hard Problems

2.2.1 Integer Factorisation

2.2.1.1 Introduction

The Fundamental Theorem of Arithmetic states that *'every positive integer n greater than 1 can be written uniquely as the product of primes':*

$$n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k} = \prod_{i=1}^{k} p_i^{a_i}$$

where $p_1, p_2, \ldots, p_k$ are distinct primes and $a_1, a_2, \ldots, a_k$ are natural numbers. It also holds that $p_1 < p_2 < \ldots < p_k$. The above equation is called the prime factorisation of n.

The integer factorisation problem is to find a non-trivial factor f (not necessarily prime) of a composite integer n. Many algorithms exist that will factor a composite n. Especially if either the prime factors of n, or n itself are expected to be of a certain form, one can choose an algorithm that is expected to have the best performance. Nevertheless, large integer factorisation is believed to be computationally intractable (Knuth 1981). To date, neither a deterministic or a randomised polynomial-

time algorithm[1] has been found for integer factorisation, nor has it been proved that an efficient algorithm does not exist.

2.2.1.2 Classification of Factorisation Algorithms

Brent (1991) identifies two main efficiency classes of factorisation algorithms. These are distinguished by the problem parameter that characterises their running time:

1. The size of the number N to be factorised.

   The most efficient factorisation method belongs to this category and is the Number Field Sieve (NFS); GNFS (a generalised version of NFS) under plausible assumptions has the expected running time

   $$\mathcal{O}\left(\exp\left(c\sqrt[3]{\log_2 N}\sqrt[3]{(\log_2 \log_2 N)^2}\right)\right),$$

   where $c = (64/9)^{1/3}$ (Lenstra, Lenstra, Manasse & Pollard 1990, Pomerance 1996). For special classes of integers, that is, integers having a particular form (and therefore some particular properties), the value of c differs.

   Other methods belonging to this category are Lehmann's method (Lehman 1974), Shanks' SQUare FOrm Factorisation (SQUFOF), Shanks' class group method, the Continued FRACtion method (CFRAC) and the Multiple Polynomial Quadratic Sieve (MPQS).

2. The size of p (the factor found) of N (assuming that $p \leq \sqrt{N}$).

   Algorithms belonging to this class are Trial Division, Pollard's ρ-method (also called the *'rho' algorithm*) and Lenstra's Elliptic Curve Method (ECM).

2.2.2 The Discrete Logarithm Problem

The discrete logarithm problem (DLP) can be described as follows: Let $a, b, n \in \mathbb{N}$. Find $x \in \mathbb{N}$ such that $a^x \equiv b \pmod{n}$, if such x exists.

[1] A *randomised polynomial-time* algorithm is one which employs a degree of randomness as part of its logic and therefore has some probability of finding the factorisation in polynomial time.

It is believed to be a hard problem and harder than the integer factorisation problem. The best algorithm for solving the DLP is due to Gordon (1993) and has expected running time of $\mathcal{O}\left(\exp\left(c\sqrt[3]{\log_2 N}\sqrt[3]{(\log_2 \log_2 N)^2}\right)\right)$.

Yan (2000) identifies three different categories of algorithms for computing discrete logarithms:

1. Algorithms that work for arbitrary groups ($\mathbb{Z}_n^*$) and do not exploit any group-specific properties. Such examples are Shanks' baby-step giant-step method, Pollard's ρ-method (similar to Pollard's ρ-factorisation method) and the λ-method.

2. Algorithms that work well on finite groups for which the order of the groups has no large prime factors. The Silver–Pohlig–Hellman algorithm lies in this category.

3. Algorithms that exploit methods for representing group elements as products of elements from a relatively small set, like for instance, Adleman's index calculus algorithm and Gordon's NFS algorithm.

Both the large integer factorisation and the discrete logarithm problem are of great interest to cryptography, since they enable the creation of computationally secure cryptosystems, provided they are provably based on either of these problems. Hence, although such a cryptosystem can theoretically be broken, the required process is computationally intractable.

2.3 Prime Number Generation

2.3.1 Introduction

Public-key cryptosystems base their security on well-known number-theoretic problems, such as the factorisation of a given number n. The generation of prime numbers is therefore vital for such encryption schemes, as they are required for generating the keys. In what follows, primality testing and modern prime number generation techniques are presented, as they both aid the generation of public key parameters. A more detailed

description of Pocklington's Theorem is given, which can serve as a prime number generation technique.

2.3.2 Testing for Primality

Testing the primality of a number is not the same as finding its prime factors. However, for applications such as public-key encryption, knowledge of primality is sufficient. Quite a few primality tests have been devised, some better than others. They can be classified into two main categories: deterministic and probabilistic.

- Deterministic

  Deterministic methods provide a definite answer to the question whether a number n is prime or not. Unfortunately, at some point, they require a factorisation of a number related to n. This factorisation can prove to be as hard as factorising n itself.

  Such paradigms are the elliptic curve test[2], the APR test (Adleman, Pomerance & Rumely 1983) and the Lucas–Lehmer test for Mersenne primes (Lehmer 1981).

- Probabilistic

  Generally speaking, probabilistic methods are much easier to implement and their execution time is significantly less than deterministic ones. Once a number n has passed such a test, its probability of being prime is very high, nevertheless, its primality cannot be guaranteed. These methods make it feasible for large numbers to be tested for primality, although it is not possible to provide a certain answer.

  Tests that belong to this category are the APRCL test[3], the Solovay–Strassen test[4], the Lehmann test, the Lucas pseudoprimality test and the Rabin–Miller test[5].

[2] Although a deterministic test, its running time is probabilistic.
[3] There also exists a deterministic, but less practical version.
[4] Also known as *Euler's pseudoprimality test.*
[5] Also known as the *Strong pseudoprimality test.*

2.3.2.1 The Rabin–Miller Pseudoprimality Test

This is probably the most popular pseudoprimality test, due to its simplicity and its high hit rate. The algorithm for testing whether a number b is a witness[6] for n or not, is as follows:

Algorithm 2.2 The Rabin–Miller pseudoprimality test

Require: Let n be the random, odd number to be tested for primality and the *base* b, a random number such that $1 < b < n$. Let $r_k r_{k-1} \ldots r_0$ be the binary representation of $n - 1$.

1: Set $d \leftarrow 1$
2: **for** $i \leftarrow k$ down to 0 **do**
3: Set $x \leftarrow d$
4: Set $d \leftarrow (d * d) \bmod n$
5: **if** $d = 1$ and $x \neq 1$ and $x \neq n - 1$ **then**
6: **return true**
7: **end if**
8: **if** $r_i = 1$ **then**
9: $d \leftarrow (d * b) \bmod n$
10: **end if**
11: **if** $d \neq 1$ **then**
12: **return true**
13: **end if**
14: **end for**
15: **return false**

If the above algorithm returns true, then n is definitely not prime. If, however, it returns false, then n *may* be prime.

A positive integer n with $n - 1 = d2^j$ and d odd is called a *base-b strong probable prime*, if it passes this test for a particular base b. A base-b strong probable prime is called a *base-b strong pseudoprime*, if it is composite.

Regarding the effectiveness of this test, given that $n > 1$ is an odd composite integer, it passes the test for at most $(n - 1)/4$ bases b with $1 \leq b < n$. In his book, Rosen (1993) has a detailed proof of why this is so. Actually, the numbers are very pessimistic. In an article by Beauchemin, Brassard, Crépeau, Goutier & Pomerance (1988) it is claimed that for most random numbers, almost 99.9% of the possible values

[6] A *witness* is a number w which, as a result of its number-theoretic properties, guarantees either the compositeness or primality of a number n.

for b are witnesses; namely, they will cause n to fail the test.

2.3.3 Prime-Generating Sequences

One other way for generating prime numbers is the use of prime-generating sequences. Probably the most famous one has been proposed by Euler: $a_n = n^2 + n + 41$. This polynomial will produce an uninterrupted sequence of 80 primes for $n = -40, -39, \ldots, 39$ (Herkommer 1999). However, the sequence is symmetric and since each prime occurs twice, 40 primes are actually produced.

Another sequence is $c_i = (c_0)^{c_{i-1}} - 1$ (Herkommer 1999). Starting with $c_0 = 2$, it does produce a sequence of primes, but it is not known whether it produces just primes, because the numbers grow extremely rapidly:

$c_0 = 2$

$c_1 = 3$

$c_2 = 7$

$c_3 = 127$

$c_4 = 170,141,183,460,469,231,731,687,303,715,884,105,727$

$c_5 > 10^{51,217,599,719,369,681,879,879,723,386,331,576,246}$.

The number $c_4 = 2^{127} - 1$ is actually the 12th Mersenne prime, that was proved to be prime by Lucas in 1876. It is very unlikely that the primality of the number c_6, or the ones that follow it, could ever be determined (Herkommer 1999).

2.3.4 Modern Prime Number Generation Methods

It seems that most people are in favour of the following way of generating prime numbers:

1. Generate a random, odd number.

2. Test for primality with a method that is guaranteed to have a high hit rate.

Fairly quick methods for testing the primality of a given number n are the Solovay–Strassen method (Solovay & Strassen 1977), the Lehmann test (Lehmann 1982) and the Rabin–Miller test (Schneier 1996), although the latter seems to be the most widely used one.

However, this method is still not guaranteed to produce a true prime number, although the probability of producing a pseudoprime is very small. Perhaps a better approach would be the following (Yan 2000):

1. Generate an odd integer n.

2. Primality testing – Probabilistic method.

   Use a combination of the Rabin–Miller test and the Lucas pseudoprimality test.

3. Primality proving – Elliptic curve method (Goldwasser & Kilian 1999).

In this way, the last step ensures that the generated number is truly a prime. Nevertheless, this comes at a significant computational cost since it involves using factorisation algorithms, which have a slow execution time. Therefore, due to practicality reasons, this step is usually omitted.

Suppose that the method followed for generating a prime number has actually produced a pseudoprime n. Although the probability of such an occurrence is very low, it is still worth looking at the consequences that it could possibly have. For most public-key cryptosystems this would be jeopardising security. The risk of an encrypted message to be successfully cryptanalysed is 'inversely proportional' to the ease of factorising n. Some numbers are easier to factorise than others. For example, by using the Number Field Sieve (NFS), Fermat numbers are easier to factorise than hard numbers[7], since they all have the form $F_n = 2^{2^n} + 1$, which is exploited by the factorisation algorithms (Schneier 1996).

[7] A *hard number* is one that does not have any small factors and is not of a special form that allows it to be factored more easily.

2.3.5 Pocklington's Theorem

2.3.5.1 Introduction

When the generation of a true prime is an absolute requirement, a way of achieving this is by using Pocklington's theorem (Yan 2000). This states that:

Let p be an odd prime, k a natural number such that p does not divide k and $1 < k < 2(p+1)$, and let $n = 2kp + 1$. Then, the following conditions are equivalent:

1. *n is prime.*

2. *There exists a natural number a, $2 \leq a < n$, such that*

$$a^{kp} \equiv -1 \pmod{n} \tag{2.1}$$

$$\gcd(a^k + 1, n) = 1 \tag{2.2}$$

An algorithm derived from the above theorem is the following (Yan 2000):

1: Choose a prime p_1 with, for example, $d_1 = 5$ digits. Let $k_1 < 2(p_1 + 1)$ such that $p_2 = 2k_1p_1 + 1$ has $d_2 = 10$ digits, or $d_2 = d_1 - 1 = 9$ digits.

2: If there exists $a_1 < p_2$ satisfying the conditions $a_1^{k_1p_1} \equiv -1 \pmod{p_2}$ and $\gcd(a_1^{k_1} + 1, p_2) = 1$, by Pocklington's theorem p_2 is prime.

3: Repeat the same procedure starting from p_2 to obtain the primes $p_3, p_4, \ldots, p_n$.

In order to produce a prime with 100 digits, the process must be iterated five times, as shown in Table 2.1. In the last iteration, k_5 should be chosen so that $p_6 = 2k_5p_5 + 1$ has 100 digits.

2.3.5.2 Implementation

In what follows, the existence of a multiple-precision integer software library is assumed for handling large integers. Fast exponentiation and fast modular exponentiation tech-

Iteration	Number of Digits
1	$d_2 = 2 \cdot d_1 = 10$ digits
2	$d_3 = 2 \cdot d_2 = 20$ digits
3	$d_4 = 2 \cdot d_3 = 40$ digits
4	$d_5 = 2 \cdot d_4 = 80$ digits
5	$d_6 = 100$ digits

Table 2.1: Required iterations for generating a 100-digit prime.

niques will be used, since the powers are expected to be quite large (interested readers are referred to Yan (2000)).

Problems Faced

When the algorithm was initially used, a few digits would suffice for generating a secure key. Nowadays, more computational power is available at relatively low cost and more efficient factorisation algorithms have been discovered, thus creating a demand for larger numbers that will be harder to factorise under the current circumstances.

The only difficult part in the implementation stage is the evaluation of $\gcd(a_i^{k_i} + 1, p_{i+1})$, and more specifically, the calculation of $a_i^{k_i}$. Even if a_i is a 1-digit prime, the size of k_i is expected to be of approximately the same number of digits as p_{i-1}, since $p_i = 2k_ip_{i-1}$.

To give an approximate order of magnitude, the 37th Mersenne prime is $M_{37} = 2^{3021377} - 1$ and has 909526 digits (Yan 2000). Supposing that k_i was a 20-digit number, the calculation of $a_i^{k_i}$ would either be impossible, or would take far too long, even by using a fast exponentiation technique.

However, the above calculation would be feasible by using a fast modular exponentiation technique, provided that the condition can be re-written in an appropriate form. For example, proving that if

$$\gcd((a_i^{k_i} \bmod p_{i+1}) + 1, p_{i+1}) = 1 \tag{2.3}$$

then also

$$\gcd(a_i^{k_i} + 1, p_{i+1}) = 1, \tag{2.4}$$

would solve the problem, since $a_i^{k_i} \bmod p_{i+1}$ can be calculated very efficiently.

Another issue worth mentioning is the procedure followed for choosing a_i and k_i. For efficiency reasons, it is not worth calculating any power of a_i for large values of a_i. For that reason, in the calculation of $a_i^{k_i p_i}$ and $a_i^{k_i}$, a_i will take the values 3, 5 and 7, cyclically. Hence, once a k_i has been generated, the theorem's conditions will be tested for each one of the values of a_i. If they are not satisfied, a new k_i will be generated and the process repeated.

Choosing k_i by algebraic methods is impossible because of the amount of numbers that are to be tested. The use of random numbers is preferred, as it will speed up the execution.

Auxiliary Functions

The algorithm presented in Section 2.3.5.2 is an implementation of Pocklington's theorem for generating a 100-digit prime. It is written in pseudo-C++ notation. Moreover, for simplicity reasons, the existence of the following functions is assumed:

`SetNewNumOfDigits(d_old, d_new)`: Checks the number of digits of p_i (`d_old`) and sets the correct number of digits for p_{i+1} (`d_new`).

`DigitsOf(num, dig)`: Checks if `num` has `dig` number of digits.

`FastModExp(base, exp, m)`: Performs $\texttt{base}^{\texttt{exp}} \bmod \texttt{m}$, using a fast modular exponentiation technique.

`Gcd(num1, num2)`: Finds the greatest common divisor of `num1` and `num2`, using Euclid's algorithm (Yan 2000).

`NextBase(base)`: Examines the current value of `base` (effectively a_i) and returns the next one, cyclically.

`GenerateRandomK(old, new)`: Given an `old`-digit long p_i, it randomly generates a k, such that $p_{i+1} = 2kp_i + 1$ has `new` digits.

Algorithm

Algorithm 2.3 Generation of 100-digit long prime, based on Pocklington's theorem

```
1: for j = 1 to 5 do {5 iterations are needed for a 100-digit prime}
2:   k = 1 {Initialise}
3:   primeFound ← false
4:   while not primeFound do
5:     SetNewNumOfDigits(d_j, d_{j+1})
6:     while not primeFound do
7:       p_{j+1} ← 2kp_j + 1
8:       if DigitsOf(p_{j+1}, d_{j+1}) then
9:         a ← 3
10:        for i = 0 to 2 do {Since there are only 3 bases}
11:          r ← FastModExp(a, kp_j, p_{j+1})
12:          if r = p_{j+1} − 1 then {Check if r ≡ −1 (mod p_{j+1})}
13:            primeFound ← true
14:            print "Prime found:" p_j
15:            Halt the algorithm and exit
16:          end if
17:          a ← NextBase(a)
18:        end for
19:      end if
20:      k ← GenerateRandomK(d_j, d_{j+1})
21:    end while
22:  end while
23: end for
```

2.3.5.3 Performance

This algorithm trades certainty for speed. By generating large random numbers and then testing them for primality, one should be able to generate 100-digit prime numbers within less than a second. The algorithm was implemented in C++, using the *BigNum* for multiple precision integer representation (Diommisfois 1988). When run on an AMD Athlon 64 3000+ processor with 1 GB of RAM, the generation of a 100-digit prime ranged from 8 seconds to 6 minutes, with an average of 2 minutes in 100 runs. Each one of the required 5 intermediate steps (iteration) was therefore performed 100 times and its running time was probabilistic. Moreover, this probabilistic behaviour regarding the running time of each iteration is further supported by the statistical results presented in Table 2.2.

These results were obtained by checking only condition 2.1, since condition 2.2 could not be implemented (reasons for doing so are mentioned in Section 2.3.5.2). Moreover, the program was experimentally modified so as to generate a 160-digit prime and it took several hours of continuous execution, before it managed to come up with one.

Iteration	Average	Minimum	Maximum	Std. Deviation
1	89.4	< 1	360	86.7
2	330.7	< 1	2380	376.3
3	2724.3	40	12330	2749.4
4	61283.4	770	303060	61965.4
5	61490.2	430	322290	69685.5

Table 2.2: Statistical results on the running time (in ms) of each intermediate step of the generation of a 100-digit long prime.

To date, there is no formal proof that the first condition implies the second, or vice versa. However, given the fact that if p_i is not prime then p_{i+1} will not be prime either, together with the results obtained, implies that condition 2.1 is a strong condition for primality testing. Therefore, since condition 2.2 is difficult to implement, it may be possible to replace it with a primality-proving method (the Elliptic Curve Method (Goldwasser & Kilian 1999), for instance). The reason for doing so, is because once a number n has passed condition 2.1, it is almost definite that it is a prime, since each number is generated so as to be of a certain form that has a high probability of being a prime. Furthermore, this is reinforced by the experimental results.

However, the algorithm's performance in the second case raises an interesting point. Does the implementation of Pocklington's theorem have an 'expiration date'? If, for instance, two 160-digit primes are needed for generating a secure key in the near future, using this method is totally impractical, since it is a very time-consuming process.

2.3.5.4 Examples

The following two examples exhibit the generation of a prime, by showing the intermediate results and the corresponding execution times as well. Generally speaking, the larger the prime to be produced in an iteration, the longer the running time is.

However, since numbers are chosen randomly, it may be the case that the generation of p_{i+1} takes less time than the generation of p_i. One such case is exhibited in the second example, where the generation of p_6 was almost 12 times faster than the generation of p_5.

First Example

Starting from $p_1 = 97711$, it took 22828 ms to produce the 100-digit long prime:

$a_1 = 3$

$k_1 = 13508$

$p_2 = 2639760377$

Time taken: 31 ms

$a_2 = 3$

$k_2 = 4376213037$

$p_3 = 23104307552766869899$

Time taken: 422 ms

$a_3 = 5$

$k_3 = 51591698866807811712$

$p_4 = 2383980955576923374338765940087984914177$

Time taken: 1031 ms

$a_4 = 3$

$k_4 = 411448602566009485257565490113657061120 0$

$p_5 = 19617712654322101187254828413863180920017957733758864128311 2$_

$2622596506486996480 1$

Time taken: 5094 ms

$a_5 = 3$

$k_5 = 61063220874898063360$

$p_6 = 23958414417423064489579484917463691210337130399604664498102 8\_$
$27091974816818252180749363892525355827 21$

Time taken: 16250 ms

Second Example

Starting from $p_1 = 97711$, it took 78031 ms to produce the 100-digit long prime:

$a_1 = 3$

$k_1 = 20090$

$p_2 = 3926027981$

Time taken: 31 ms

$a_2 = 3$

$k_2 = 2049408165$

$p_3 = 16092067600559729731$

Time taken: 79 ms

$a_3 = 3$

$k_3 = 54210009287892640700$

$p_4 = 1744702268175478379206261938671781303401$

Time taken: 3968 ms

$a_4 = 5$

$k_4 = 69542819831361482627393975939383808597 76$

$p_5 = 24266303099019003515459454384918227201794077178772481180545 9\_$
85673495744585796353

Time taken: 68250 ms

$a_5 = 3$

$k_5 = 61131154630330700720$

$p_6 = 29668542541052075782372986345122253193484911206209031745029544525496521247532737691690466914209483 21$

Time taken: 5703 ms

2.4 The Quadratic Residuosity Problem

An integer a is said to be a *quadratic residue modulo* n if $\gcd(a, n) = 1$ and if there exists a solution x to the congruence $x^2 \equiv a \pmod{n}$. Hence, given integers a and n, the QRP is to decide whether a is a quadratic residue modulo n or not. If n is an odd prime, according to Euler's criterion, a is a quadratic residue of n if and only if $a^{(n-1)/2} \equiv 1 \pmod{n}$. However, in the case where n is an odd composite, a is a quadratic residue of n if and only if it is a quadratic residue modulo every prime p_i dividing n. Alternatively, the Legendre symbol $\left(\frac{a}{p_i}\right)$ can be evaluated, which is defined by:

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & \text{if } a \in Q_p \ (a \text{ is a quadratic residue modulo } p) \\ -1, & \text{if } a \in \overline{Q}_p \ (a \text{ is a quadratic non-residue modulo } p) \\ 0, & \text{if } a \mid n \ (a \text{ divides } n). \end{cases}$$

If the factorisation of n is unknown, then there is no efficient procedure known for solving the QRP, other than guessing the answer. If $n = pq$, then the probability of a correct guess is 0.5, since exactly half of the elements in $\mathbb{Z}_n$ are quadratic residues and the other half are quadratic non-residues. It is believed that the QRP is as difficult as the integer factorisation problem, although no proof is known (McCurley 1990). On the other hand, no other way has been invented up-to-date for solving the QRP. Due to the inability to solve the QRP without factorising, several researchers have proposed cryptosystems whose security is based on the difficulty of determining quadratic residuosity.

2.5 Concluding Remarks

In this chapter, the fundamentals of complexity theory were described, so as to understand better the principles of integer factorisation from a computational perspective. The difficulty of factorising a large integer n forms the foundation on which the security of most public-key cryptosystems is based. What is more, both the applications of prime numbers to asymmetric cryptosystems and the important role they play were presented. More specifically, methods for prime number generation and primality testing were mentioned. The Rabin–Miller strong primality testing algorithm was presented in detail, since it is the most widely used one and also because it was chosen to be used in the key generation phase of the proposed system of this thesis. A more extensive study was made on Pocklington's theorem. The reason for doing so was that this theorem, if implemented, is guaranteed to generate a true prime; however, its slow performance poses serious questions whether it can be used in the future, when larger primes will be required. Finally, the Quadratic Residuosity Problem was presented, since it relates to the integer factorisation problem and because it forms the basis of several cryptosystems.

Chapter 3

Cryptography

3.1 Introduction

Cryptography[1] is the practice of using encryption to conceal information. It is the study of *encrypting* (or *enciphering*) information by transforming it into apparently unintelligible forms, called *ciphertexts*. The algorithm of such a transformation process is called a *cipher*. The transformation process is performed by using some secret information, called the *key*. The reversal of this process and the extraction of the original message should not be feasible without knowledge of the key.

Perhaps the first example of applied cryptography dates back to 2000 BC, when the Egyptians used a hieroglyphic code for inscription in tombs. However, the first recognised cryptographic device was the *scytale*[2], devised by the Ancient Greeks. The first clear description of its use as a cryptographic device was by Apollonius of Rhodes, who lived in the middle of the 3rd century BC. A piece of long and narrow strap-like parchment was wound around a cylinder. The message was written along the length of the cylinder and, when the parchment was unreeled, the letters on it made no sense. The key in this case was the diameter of the cylinder.

[1] From the Greek *κρύπτω* (krýpto) and *γράφω* (grápho), which means *write secretly*; the definition is attributed to Thomas Browne in 1658, an English Physician and writer (Bauer 1997).

[2] From the Greek *σκυτάλη* (scytáli), meaning *baton* (like the ones used in relay races).

When two parties wish to communicate secretly, they may make use of secret-key cryptography, under the precondition that they share the same secret key. The operations of both encryption and decryption are performed with the use of this key. Alternatively, public-key cryptography can be used. In this case each party possesses a pair of keys: a private key known only to them and and a public one available to everyone. The plaintext is encrypted using the receiver's public key, whereas the ciphertext can only be decrypted by the respective secret key. Moreover, it is not possible in feasible time to deduce the secret key by studying the public key.

A different approach for establishing a secure communication is *steganography*[3], the purpose of which is to conceal the existence of a message. The message itself – unlike cryptography – does not necessarily undergo any actual transformations, although it may be encoded in a way that enables or simplifies its processing. The origins of steganography also go back to the time of Ancient Greece, where Histiaieus wished to inform his friends that it was time to begin a revolt against the Medes and the Persians. He shaved the head of one of his trusted slaves, tattooed the message on the head, waited until his hair grew back and dispatched him. His idea worked. The message got to his correspondents in Persia and the revolt succeeded.

Technological evolution gave birth to new steganographic techniques. Currently, they mainly involve manipulation of digital media files to embed secret information in them, without creating easily detectable anomalies that would give away the existence of the secret message. For instance, a digital image may have a decent amount of secret information embedded in it and still be viewable, without exhibiting any kind of distortion or malformation of the displayed picture. Furthermore, there also exist techniques that are used for (secret) digital watermarks with a wide area of applications, such as copyright protection.

In today's world, information transmission through networks is very insecure and open to many kinds of attack, as an eavesdropper could possibly monitor, or even tamper with the transmission. The best method for protection of such a data transfer

[3]From the Greek στεγανός (steganós) and γράφω (grápho), meaning *write covertly*.

is to encrypt the sensitive information prior to sending it. The main problem here is how to develop an uncrackable public-key encryption scheme. At present, almost all public-key encryption schemes have based their security on some intractable number-theoretic problem, such as integer factorisation and discrete logarithms, as there is little hope that (efficient) algorithms for solving them will be invented in the near future.

3.2 Symmetric (Secret-Key) Cryptosystems

Symmetric algorithms are those for which the encryption key can be calculated from the decryption key and vice versa. Schneier (1996) mentions that they may also be found under the terms *single-key algorithms*, or *one-key algorithms*, since they require that the sender and receiver agree on a key prior to commencing a secure communication. The key must remain secret for as long as the secret communication lasts.

They can further be divided into two categories:

- *Stream ciphers* are those that operate on one bit (sometimes byte) of plaintext at a time.

- *Block ciphers* are those that operate on the plaintext organised in groups of bits, or on groups of digits. The block size must be small enough so as to be workable and not too big so as to preclude cryptanalysis. According to Schneier (1996), a typical block size used to be 64 bits. The current standard, AES, uses a 128-bit block size, although the algorithm (Rijndale) is capable of handling larger block sizes (Daemen & Rijmen 2002).

One characteristic of these cryptosystems is their fairly high execution speed. Nevertheless, their main drawback is the key management and key distribution problem. That is, if n users wish to communicate secretly with each other, then the required number of secret keys is $n(n-1)/2$. If, for example, there were 20 users who wished to communicate secretly among them, then each one should have 190 keys! For larger values of n, the number of keys increases dramatically.

The key exchange problem can be solved by having a key centre (Buchmann 2000). In this framework, if Alice wants to send a message to Bob, she encrypts the message using her secret key and sends it to the key centre. The key centre, knowing all the keys, will decrypt the message, encrypt it with Bob's key and send it to Bob. In this way, the number of key exchanges for n users is reduced to n. However, the key centre gets to know all secret messages (thus not offering true secrecy) and it must store all n keys securely.

Since this research is oriented towards public-key cryptography, secret-key cryptosystems will not be discussed any further.

3.3 Asymmetric (Public-Key) Cryptosystems

In these algorithms, the key used for encryption is different from the one used for decryption. They were designed in order to address the key management and key distribution problem which is created by using symmetric algorithms.

At the moment, the majority of public-key cryptosystems are based on RSA, El-Gamal, and Rabin algorithms, or variants of them. There are algorithms which are most suited for encryption, others that are useful only for digital signatures and some that are equally suitable for both uses. The three algorithms mentioned above belong to the last category. The main disadvantage of public-key cryptosystems is that they encrypt and decrypt data much slower than symmetric (secret-key) algorithms.

Schneier (1996) suggests that the use of a hybrid cryptosystem solves this problem, as it uses a symmetric algorithm with a random session key for encrypting the message and a public-key algorithm for encrypting the random session key.

Elliptic-curve cryptosystems are continually gaining popularity and can be used for both encryption and digital signature schemes. This class of cryptosystems does not use fundamentally new algorithms; existing public-key algorithms have been adapted to use elliptic curves. According to Schneier (1996), their main advantages are that they are easy to construct, relatively simple to implement and that they provide faster

public-key cryptosystems with smaller key sizes (without reducing security).

3.4 Security of Cryptosystems

3.4.1 Cryptanalysis

When two or more people wish to communicate secretly, there are many more who would benefit from intercepting the communication and are thus motivated to do so by launching an *attack*. Cryptanalysis is the field of cryptology that deals with recovering the plaintext message from the ciphertext. Attacks applicable to both symmetric and asymmetric cryptosystems will be presented in this section, whereas the later sections will focus on asymmetric ones. Schneier (1996) identifies four general types of cryptanalytic attacks:

1. *Ciphertext-only attack*: The cryptanalyst has the ciphertext of several messages, all of which have been encrypted by using the same algorithm.

2. *Known-plaintext attack*: The cryptanalyst has access not only to the ciphertext of several messages, but also to the respective plaintexts.

3. *Chosen-plaintext attack*: The cryptanalyst can choose the plaintext that gets encrypted. In this way, they can get more information.

4. *Adaptive chosen-plaintext attack*: The cryptanalyst not only can choose the plaintext that gets encrypted, but can modify their choice based on the results of previous encryption. This is a very powerful attack that is mostly used on symmetric cryptosystems.

   Four other types of attacks are (Schneier 1996):

5. *Chosen-ciphertext attack*: The cryptanalyst can choose the ciphertext that gets decrypted and has access to the decrypted plaintext.

6. *Adaptive chosen-ciphertext attack* (also found under the terms *lunchtime attack* and *midnight attack*): This is a variant of the chosen-ciphertext attack, only that the cryptanalyst is also allowed to modify the ciphertexts that get decrypted based on the results they get. It is a very powerful attack applicable mostly to public-key schemes. As Shoup (2001) claims in his article,

   > "...it is generally agreed that the 'right' definition of security for a public-key encryption scheme is *security against adaptive chosen ciphertext attack*, as defined by Rackoff & Simon (1991). This notion of security is equivalent to other useful notions, such as the notion of *non-malleability*, defined by Dolev, Dwork & Naor (1991) and Dolev, Dwork & Naor (2000)".

7. *Chosen-key attack*: This attack does not mean that the cryptanalyst can choose the key! They do not know what the keys are; they only have some knowledge about the relationship between different keys. It is not a very practical attack.

8. *Rubber-hose cryptanalysis*: The cryptanalyst attempts to acquire the key by means of bribery (also known as *purchase-key attack*), blackmailing, intimidation, torturing.

   In a paper by Kocher (1996), another interesting kind of attack is mentioned:

9. *Timing attack*: By carefully measuring the amount of time required to perform private key operations, the cryptanalyst may be able to find fixed Diffie–Hellman exponents, factorise RSA keys and break other cryptosystems.

It is also worth noting that there may also be (and there are!) attacks that do not really belong to any of the categories mentioned above, but depend solely on the structure of a specific cryptosystem. This is quite common to number-theoretic cryptosystems, where special features of the scheme itself are exploited.

3.4.2 Notions of Security

As has already been mentioned, most public-key cryptosystems base their security on computationally complex number-theoretic problems. Although their security is

mostly based on the size of the key, different notions of security have been defined, each addressing different areas that may contain potential vulnerabilities. The security of a cryptosystem is therefore reinforced, if it is able to withstand the attack scenarios described in them.

At this point it would be useful to introduce the definition of a *passive adversary*: They are effectively an eavesdropper who simply obtains a copy of the transmitted information in a communication. They are neither blocking the communication, nor interfering with the transmitted information in any way. Any kind of interference makes them an *active adversary*.

3.4.2.1 Polynomial and Semantic Security

A scheme is said to be *polynomially secure* if no passive adversary can, in expected polynomial time, select two plaintext messages m_1 and m_2, and then correctly distinguish between encryptions of m_1 and m_2 with probability greater than 0.5 (Menezes, van Oorschot & Vastone 1997). In this case, the adversary is effectively performing a chosen plaintext attack.

Goldwasser & Micali (1984) defined *semantic security*, which is based on the assumption that an adversary should not be able to obtain any partial information about a message, given its encryption. For a passive adversary the above definition can guarantee secrecy. However, this is not the case if an adversary can mount an active attack. According to Menezes et al. (1997), this definition is equivalent to the definition of polynomial security mentioned above. Furthermore, it follows that a public-key encryption scheme is semantically secure if the ciphertext does not leak any partial information whatsoever about the plaintext that can be computed in expected polynomial time.

3.4.2.2 The Notion of Security Against the Adaptive Chosen-Ciphertext Attack

In order to deal with active attacks, Rackoff & Simon (1991) defined the notion of security against an adaptive chosen-ciphertext attack. For instance, if an adversary can inject messages into a network, these messages may be ciphertexts and the adversary may then be able to extract partial information through interaction with the parties in the network. The model they proposed for this attack is simply allowing the adversary to obtain decryptions of their choice. In other words, the adversary has access to a *decryption oracle*. The only restriction imposed is that the adversary cannot submit the target ciphertext itself; nevertheless, they are allowed to submit any other ciphertext, including some related to the target ciphertext. Hence, in order for a cryptosystem to be considered secure against this type of attack, it must be able to resist the attack scenario described above.

In most cases this attack is viewed as a certificational weakness. Attack scenarios involve gaining access to an unattended decryption machine. This is why these attacks may also be found under the terms *lunchtime attack* and *midnight attack* (namely, using one's computer while they are away for lunch, or gone to bed).

Since the attack set-up may seem odd in the sense that the attacker is allowed to submit arbitrary ciphertexts for decryption but not the one in question, it is worth clarifying that the attacker may have the ciphertext without the accompanying digital signature. They are therefore not able to produce a valid signature and assuming that the decryption oracle will not return anything decrypted back (because message integrity/authentication check will fail), they cannot get this ciphertext decrypted.

Furthermore, two other attack scenarios used in the literature are:

1. The attacker chooses two plaintexts m_1 and m_2. They get encrypted into c_1 and c_2 and given to them, but without knowing which ciphertext belongs to which plaintext. The attack is then launched and the attacker tries to determine which is which. If the cryptosystem is probabilistic, re-encrypting the plaintexts is

pointless.

2. The attacker possesses ciphertext c and requests the decryption of fc, where f is a multiplier. Based on the answer they get, they choose a different f and repeat the attack. They may end up with the plaintext, or they may manage to obtain the key.

3.4.2.3 Non-Malleability

In addition to the properties of security that were mentioned above, an equally important measure of security is the property of *non-malleability* (Dolev et al. 1991, Dolev et al. 2000). That is, given a ciphertext, it is computationally infeasible to generate a different ciphertext such that the respective plaintexts are related in a known manner. If a scheme is non-malleable, it is also semantically secure.

3.4.2.4 Plaintext-Aware Encryption

Bellare & Rogaway (1995) introduced the notion of *plaintext-aware encryption.* Namely, it is infeasible for an adversary to produce a valid ciphertext without knowledge of the corresponding plaintext. The scheme operates as follows: Let f be a k-bit to k-bit trapdoor one-way permutation. The message m is padded out with k_1 zeroes and it undergoes XOR operations from two random functions.

The original way they proposed for achieving this was by using two random functions. However, in practice, hash functions are used instead and thus the scheme cannot be proved plaintext-aware, because the random function assumption is no longer true. Nevertheless, it still seems to provide good security assurances (Menezes et al. 1997).

3.5 RSA

3.5.1 The Algorithm

This is probably the world's most popular and most extensively studied cryptosystem. It was named after the three inventors, Ron Rivest, Adi Shamir and Leonard Adleman (Rivest et al. 1978). Its security is based on the difficulty of factorising large integers.

It is worth noting that what is presented here is the simplest version of RSA. However, all modern implementations based on it use advanced techniques for processing the plaintext before it gets encrypted. In this way, certain vulnerabilities are addressed and thus the end system is a more secure one (see Section 3.5.2.1).

The keys are generated as follows:

Algorithm 3.1 Generation of RSA keys

1: Choose two random, large prime numbers, p and q, and compute the product $n = pq$ (for maximum security, p and q should be of approximately equal length).
2: Randomly choose the encryption key e, such that e and $\phi(n) = (p-1)(q-1)$ are relatively prime.
3: Using the Extended Euclidean algorithm compute the decryption key, d, such that

$$ed \equiv 1 \bmod \phi(n).$$

4: The public key is formed by the pair (e, n), whereas d is the private key. The two primes, p and q, are no longer needed and can be discarded, but never revealed.

Before encrypting a message, m, it must be divided into blocks, at least one bit less than the bit length of the modulus, n. For instance, if n is a 1024-bit long composite, then each message block should be at most 1023 bits long. This is to ensure that the numerical value of all blocks is less than n. Once this is done, the encryption formula

$$c_i \equiv m_i^e \bmod n$$

should be applied to each message block m_i.

In order to decrypt a message, c, the decryption formula

$$m_i \equiv c_i^d \bmod n$$

should be applied to each encrypted block c_i.

Regarding its complexity, the encryption operation requires $\mathcal{O}(k^2)$ operations, decryption requires $\mathcal{O}(k^3)$ operations and key generation takes $\mathcal{O}(k^4)$ operations, where k is the bit length of the RSA modulus. Its message expansion is $(|n|-1)/|n|$, which is sometimes referred to as being 1.

An interesting statement is made by Buchmann (2000), which is that RSA decryption can be hastened by using the Chinese Remainder Theorem (CRT). It is worth pointing out that the original RSA design discards p and q, as soon as e and d have been generated, but the two primes must be retained to implement this technique.

Supposing that Alice wants to decrypt the ciphertext c, with her decryption key, d, she computes:

$$m_p \equiv c^{d \bmod (p-1)} \bmod p,$$

$$m_q \equiv c^{d \bmod (q-1)} \bmod q.$$

Using the CRT she then computes an integer $m \in \{0, 1, \ldots, n-1\}$ such that

$$m \equiv m_p \bmod p,$$

$$m \equiv m_q \bmod q.$$

This m is the plaintext that was encrypted. To find m, she uses the extended Euclidean algorithm to find integers y_p and y_q, such that $y_p p + y_q q = 1$. Then,

$$m \equiv (m_p y_q q + m_q y_p p) \bmod n.$$

Let us assume that n and d are k-bit numbers. The prime factors, p and q, of n are $k/2$-bit numbers. The multiplication of two integers of binary length $\leq r$ takes time $\leq Cr^2$, where C is a constant. Similarly, the division with remainder of an integer of length $\leq r$ requires time $\leq Cr^2$. The computation of $m \equiv c^d \bmod n$ takes time $\leq 2Ck^3$. The computation of m_p and m_q requires time $Ck^3/2$. The time for the precomputation of $y_p p \bmod n$ and $y_q q \bmod n$ can be ignored, since it requires only one application of the extended Euclidean algorithm, which has quadratic running time.

The computation of $m \equiv (m_p y_p q + m_q y_q p) \bmod n$ requires only two multiplications and one addition modulo n, so decryption with the CRT is almost four times as fast as standard decryption.

3.5.2 Security of RSA

3.5.2.1 General

The security of RSA is based on the difficulty of factoring n. However, it should be noted that it is only a conjecture that breaking RSA is as hard as factorisation. Up to now no proof has been provided that factorising n is needed in order to calculate m from c and e. An alternative way would be by guessing $\phi(n) = (p-1)(q-1)$. Wu & Wang (1993) proved that this problem is not easier than factorisation. One other possibility is by determining d. An adversary can then follow a procedure for guessing a non-trivial factor of n (namely, p or q), which is expected to succeed within two trials (Menezes et al. 1997).

3.5.2.2 Common Modulus Attack

For reasons of efficiency, it is possible to generate different encryption and decryption exponents (e, d), while keeping the same modulus, n. Nevertheless, this poses a security threat, as the common modulus attack can be launched as follows (Simmons 1984):

Let m be the plaintext message and e_1, e_2 the two encryption keys. The two ciphertexts will be $c_1 \equiv m^{e_1} \bmod n$ and $c_2 \equiv m^{e_2} \bmod n$, respectively. Since e_1 and e_2 are relatively prime, the extended Euclidean algorithm can be used to find r and s such that $re_1 + se_2 = 1$. Since $e_1 > 0$ and $e_2 > 0$, one of either r or s has to be negative. Assuming that $r < 0$, using the extended Euclidean algorithm, c_1^{-1} can be calculated thus: $(c_1^{-1})^{-r} c_2^s \equiv m \bmod n$.

A variant of this attack also exists. Vanstone & Zuccherato (1995) proposed a method for generating RSA moduli having a predetermined set of bits. This was later proved to be insecure, after Coppersmith (1996*a*) showed that n can be factored in

polynomial time, if the high order $k/4$ bits of p are known.

3.5.2.3 Low Exponent Attack

It was believed in the past that the encryption exponent, e, can be chosen to be small, so as to enable faster encryption without compromising security (Schneier 1996). However, this belief was later proved to be false. As mentioned by Buchmann (2000), the *low exponent attack* can be used, which is a variant of the known-ciphertext attack. This attack works if the same message m is encrypted e times with encryption exponent e and e pairwise coprime RSA moduli $n_i, 1 \leq i \leq e$. The smaller the value of e, the more likely it is to happen. For instance, let $c_i \equiv m^e \bmod n_i$, $1 \leq i \leq e$ be the corresponding RSA ciphertexts. The attacker can then use the following algorithm:

Algorithm 3.2 The low exponent attack on RSA

1: Compute an integer c with $c \equiv c_i \bmod n_i$, $1 \leq i \leq e$ and $0 \leq c < \prod_{i=1}^{e} n_i$, using the Chinese Remainder Theorem.
2: Determine the message m as the eth root of c. Namely, find an integer m such that $m^e = c$.

Wiener (1990) proposed an attack which can recover the decryption exponent, d, if d is up to one quarter the size of n and e is less than n. Although this occurs rarely if e and d are chosen at random and never if e has a small value, the key generation process can be adjusted so as to produce a large d, thus avoiding this attack.

3.5.2.4 Malleability

This attack is based on the multiplicative property of modulo operations, that is $(m_1 m_2)^e \equiv m_1^e m_2^e \equiv c_1 c_2 \pmod{n}$. This attack was initially proposed by Dolev et al. (1991) and the example they presented is as follows. Assume that different users bid for an on-line auction by simply submitting a number (their bid for the item of interest). After the competitors' encrypted bids are intercepted, they can be multiplied by $f^{e_i} \pmod{n}$, where f is a number less than 1 and e_i the (public) encryption exponent of each other user. In this way, the cheating party can ensure that their com-

petitors' bids will be lower than theirs. For instance, if $f = 9/10$, then the competitors' bids will go down to 90 percent of their original value.

3.5.2.5 Other Attacks

Menezes et al. (1997) mention two more attacks on RSA:

- *Cycling attacks* and
- *Message concealing attacks.*

In the first case the objective is to find a positive integer k such that $c^{e^k} \equiv c \pmod{n}$ and then $c^{e^{k-1}} \equiv m \pmod{n}$. Essentially, this is a factorisation algorithm for n and because integer factorisation is assumed to be computationally intractable, cycling attacks are not considered to pose a threat to RSA.

The second attack focuses on determining whether the encryption scheme is unconcealed, that is $m^e \equiv m \pmod{n}$. The number of unconcealed messages is exactly $(1 + \gcd(e-1, p-1))(1 + \gcd(e-1, q-1))$. Since $e-1$, $p-1$ and $q-1$ are even, the number of unconcealed messages will always be at least 9. However, if p and q are random primes and if e is chosen at random (even if e is a small number, such as $e = 3$, $e = 2^{16} + 1 = 65537$) the proportion of such messages is, in general, negligibly small and they are therefore regarded as not being a threat for RSA.

3.5.3 The Partial Information Leakage Problem

As well as the direct attack on basic encryption algorithms described in Section 1.1, where the adversary knows (or guesses) the possible plaintext, 'background hints' are sometimes given away and this compromises security.

For instance, suppose that an on-line auction takes place via a network. Each bidder places only one offer and the item of interest goes to the one who will offer the most money. Valid offers are the ones submitted within the deadline (say 1 day). Obviously, it is of great importance for a bidder to know the offers of their competitors. Given

that the starting value is known, say $1,000,000, one can compute the encryptions of all the numbers up to $10,000,000 (assuming that this is a sensible upper limit in this case) in increments of $10,000 and compare them to the ones submitted by the other bidders. Given the current computational power available, this is a feasible task.

Another example is where customers request from their bank their PIN to be sent to them on-line. The bank will encrypt it using the customer's public key prior to sending it. Currently, the encryptions of all possible 4-digit PINs (or even 5-digit ones) can be computed in very little time, given the computational power available. After the comparison, the PIN will be unveiled.

There can also be cases where the set of possible messages is not that restricted. Nevertheless, an attacker is still able to obtain useful information. One such example is the case of a hybrid cryptosystem, where the message is actually the session key. Assuming that a deterministic public-key algorithm is used, some kind of padding will be required for making a (much smaller) session key suitable for encryption. Hence, the search space is limited by the length of the session key. According to the birthday paradox (Section B.2) it would be feasible to find several (encrypted) instances of the same session key. A cryptanalyst will then know that certain messages have been encrypted using the same key and they can use this information to their advantage. For instance, they may focus on breaking just one of them. Upon successful cryptanalysis they will also have access to the other messages at no extra computational cost.

3.5.4 Facing the Partial Information Leakage Problem

A very early attempt on hiding partial information was that of Blum & Goldwasser (1985). It is actually the most efficient probabilistic encryption method known so far, both in terms of speed and message expansion, and it is comparable to the RSA encryption scheme (Menezes et al. 1997). It is a semantically secure cryptosystem and bases its security on the integer factorisation problem. It exploits the Quadratic Residuosity Problem and operates by generating a pseudorandom bit sequence, which

is then XOR-ed with the message bit string. The resulting bit sequence together with the random seed (encrypted as a quadratic residue modulo a composite, n) is sent to the receiver, who uses their key to extract the random seed in order to reproduce the pseudorandom bit string and thus recover the plaintext. Assuming that t pseudorandom integers are needed for encrypting a message m, the ciphertext produced is longer than the plaintext only by a constant number of bits, namely the length in bits of the $(t+1)$th integer.

Nevertheless, this scheme has not received much attention, mainly because it is vulnerable to a chosen-ciphertext attack that recovers the private key from the public key (Menezes et al. 1997). The attack is as follows: Let y be the random seed, encrypted as a quadratic residue, and c be the ciphertext to be cryptanalysed. The attacker obtains m', which is the decryption of a chosen ciphertext, c'. The plaintext m corresponding to c is: $m = c' \oplus m' \oplus c$.

Bellare & Rogaway (1995) attempted to face the partial information leakage problem of RSA. They introduced a new encoding scheme for RSA, called Optimal Asymmetric Encryption Padding (OAEP), that uses a cryptographic hash function. A security flaw in OAEP was later pointed out by Shoup (2001), who proposed an improved version, OAEP+.

A more recent attempt is the Cramer–Shoup (Shoup 1998) cryptosystem, which is an extension of the ElGamal cryptosystem (ElGamal 1985*a*). It is a reasonably practical cryptosystem (Shoup 1998) and it can be proved that it is secure against a chosen-ciphertext attack.

Nowadays, almost every implementation of RSA takes into consideration the partial information leakage problem and actions are taken to prevent it. The structure of the RSA algorithm itself helps, as before a plaintext block can be encrypted (exponentiated) it must undergo some pre-processing. This processing can – and usually does – include some randomisation. Perhaps the simplest way of doing that is by *salting* the message. This involves generating a pseudorandom bit string of an appropriate length,

which is appended to the plaintext message prior to encryption; the pseudorandom bit string should be independently generated for each encryption. Nevertheless, this technique is still insufficient for security, as Coppersmith (1996*b*) showed how an adversary can recover the secret message, m, if certain conditions are met.

Based on what has been mentioned above, it has become apparent that it is of great importance to make RSA probabilistic and at the same time prevent it from leaking partial information. Of course, such a scheme should be provably secure, if it is to be of any use. The construction of two such techniques is among the main objectives of this thesis.

3.6 Other Public-Key Algorithms

3.6.1 The Goldwasser–Micali Cryptosystem

This cryptographic scheme was initially proposed in a paper by Goldwasser & Micali (1984), under the term *probabilistic encryption*. It is based on the Quadratic Residuosity Problem (QRP) and more specifically, on the difficulty of distinguishing squares from pseudosquares[4] for a given composite n; according to Yan (2000), the only known method so far is to factorise n. Since integer factorisation is – until now – computationally infeasible, solving the QRP problem is computationally infeasible.

An integer a is a quadratic residue modulo n, denoted by $a \in Q_n$, if $\gcd(a, n) = 1$ and there exists a solution x to the congruence $x^2 \equiv a \pmod{n}$; otherwise, a is a quadratic non-residue modulo n, denoted by $a \in \overline{Q}_n$. The only way to distinguish quadratic residues from quadratic non-residues modulo a prime n, is whether the Legendre symbol $\left(\frac{a}{n}\right)$ evaluates to 1 or -1, respectively. If n is composite, then $a \in Q_n \Rightarrow \left(\frac{a}{n}\right) = 1$, but $a \in Q_n \not\Leftarrow \left(\frac{a}{n}\right) = 1$. However, $a \in \overline{Q}_n \Leftarrow \left(\frac{a}{n}\right) = -1$.

Algorithms 3.3 to 3.5 present the details of this cryptosystem.

[4] For a complete definition and properties of pseudosquares, the reader is referred to Section A.2.2.3.

Algorithm 3.3 Key generation of the Goldwasser–Micali cryptosystem

1: Choose two large primes p and q, and compute $n = pq$.
2: Select $y \in \mathbb{Z}/n\mathbb{Z}$, such that $y \in \overline{Q}_n$ and $\left(\frac{y}{n}\right) = 1$. Hence, y is a pseudosquare modulo n.
3: The public key is the pair (n, y).

Algorithm 3.4 Encryption of the Goldwasser–Micali cryptosystem

1: Treat the message as a string of 1s and 0s, of length k.
2: **for** i from 1 to k **do**
3: Randomly choose $x_i \in (\mathbb{Z}/n\mathbb{Z})^*$.
4: Compute the ciphertext

$$c_i = \begin{cases} x_i^2 \bmod n, & \text{if } m_i = 0 \text{ (random square)} \\ yx_i^2 \bmod n, & \text{if } m_i = 1 \text{ (random pseudosquare)} \end{cases}$$

5: **end for**

Algorithm 3.5 Decryption of the Goldwasser–Micali cryptosystem

1: **for** i from 1 to k **do**
2: Evaluate the Legendre symbol $e_i' = \left(\frac{c_i}{p}\right)$ or $\left(e_i'' = \left(\frac{c_i}{q}\right)\right)$.
3: Compute

$$m_i = \begin{cases} 0, & \text{if } e_i' = 1 \text{ (or } e_i'' = 1) \\ 1, & \text{if } e_i' = -1 \text{ (or } e_i'' = -1) \end{cases}$$

4: **end for**
5: Using the derived string of 1s and 0s, obtain the original message.

One other point worth mentioning is that in the decryption phase mentioned above, it suffices to evaluate either $\left(\frac{c_i}{p}\right)$ or $\left(\frac{c_i}{q}\right)$. The proof is as follows[5]:

y is a pseudosquare, with $\left(\frac{y}{n}\right) = 1$, hence

$$\left(\frac{y}{p}\right) = \left(\frac{y}{q}\right) = -1. \tag{3.1}$$

x_i^2 is a random square, hence

$$\left(\frac{x_i^2}{p}\right) = \left(\frac{x_i^2}{q}\right) = 1. \tag{3.2}$$

After the encryption process, c_i will equal to either $x_i^2 \bmod n$, or to $x_i^2 y \bmod n$. In order to decrypt, in the second case, $\left(\frac{c_i}{p}\right)$ needs to be evaluated.

[5]For information on the properties of Legendre and Jacobi symbols, the reader is referred to Section A.2.2.

$$\left(\frac{c_i}{p}\right) = \begin{cases} \left(\frac{x_i^2 y}{p}\right) = \left(\frac{x_i^2}{p}\right)\left(\frac{y}{p}\right), \text{or} \\ \left(\frac{x_i^2}{p}\right) \end{cases} \tag{3.3}$$

However,

$$\left(\frac{c_i}{q}\right) = \begin{cases} \left(\frac{x_i^2 y}{q}\right) = \left(\frac{x_i^2}{q}\right)\left(\frac{y}{q}\right), \text{or} \\ \left(\frac{x_i^2}{q}\right) \end{cases} \tag{3.4}$$

From Equations (3.1) and (3.2) it follows that Equations (3.3) and (3.4) are equivalent.

An advantage of this cryptosystem is its probabilistic encryption. Namely, the same plaintext message can yield more than one ciphertext, thus making cryptanalysis harder. From the security point of view, it is provable that breaking the system is as hard as factorising the public key, n (Yan 2000). Finally, it can also function as a digital signature scheme by simply swapping the two keys.

A disadvantage of this cryptosystem is that it is only *semantically* secure. Thus, a ciphertext can trivially be altered by multiplying it with the pseudosquare, y, which will 'flip' the plaintext bit. A further drawback is that it has a high message expansion; the factor is $\log_2 n$, where n is the modulus (Menezes et al. 1997). Consequently, it suffers from low execution speed both in the encryption and decryption process. In particular, for a k-bit long modulus, encryption takes time $\mathcal{O}(k^2)$ to encrypt each bit and $\mathcal{O}(k^3)$ to produce one decrypted bit.

3.6.2 The Rabin Cryptosystem

The Rabin cryptosystem (or variants of it) is also based on the Quadratic Residuosity Problem (QRP), but unlike the Goldwasser–Micali scheme it is a deterministic scheme. It was the first *provably secure* public-key cryptosystem. In particular, it was proved that breaking the system is as hard as factorisation (Rabin 1979). Regarding its performance, Rabin encryption is an extremely fast operation, as it consists of a single modular squaring. Decryption is slower than encryption, but comparable to RSA decryption.

3.6.2.1 Algorithm

The algorithmic details of the Rabin cryptosystem are presented in Algorithms 3.6 to 3.8.

Algorithm 3.6 Key generation of the Rabin cryptosystem

1: Generate 2 large primes, p and q and calculate their product, n.
2: The public key is n and the private key is (p, q).

Algorithm 3.7 Encryption of the Rabin cryptosystem

1: Break the message m into k blocks, each representing an integer, $m_i \in [1, n-1]$, where $i = 1, 2, \ldots, k$.
2: Compute the ciphertext $c_i \equiv m_i^2 \bmod n$.

Algorithm 3.8 Decryption of the Rabin cryptosystem

1: **for** i from 1 to k **do**
2: Use the Extended Euclidean Algorithm to find the four square roots, m_{i_1}, m_{i_2}, m_{i_3}, m_{i_4} of $c_i \bmod n$.
3: The receiver somehow decides which one is the original block, m_i.
4: **end for**
5: Using all the m_i blocks, reconstruct the original message.

Step number 3 in the decryption phase above, is a drawback of the Rabin scheme. If the message consists of printable ASCII characters, it is most likely that three of the roots will contain control characters, or even characters from other character sets. In both cases, they will form meaningless strings, which can easily be rejected by either a human reader, or by computerised dictionary reference.

In their book, Menezes et al. (1997) mention a method for automating the decision problem mentioned in the paragraph above. The technique involves using pre-specified redundancy in each plaintext block. For instance, it could be arbitrarily chosen to replicate the last 64 bits of the message block. Then, with high probability, exactly one of the four square roots of a legitimate ciphertext will possess this redundancy. If none of these roots possesses this property, the ciphertext should be discarded as fraudulent. Furthermore, should the 64 redundant bits prove to be insufficient, the number can easily be increased to ensure the smooth operation of the cryptosystem.

The plain version of the Rabin cryptosystem is completely insecure against a chosen-ciphertext attack (Stinson 1995), which can be performed as follows: The attacker requests the decryption of $m^2 \bmod n$, where m is a random integer in $\mathbb{Z}_n^*$ chosen by them. The decryption oracle will respond with the plaintext y, which may not be equal to m (this does not cause any problems). Decryption requests are repeated for different m until a plaintext y is obtained such that $y \not\equiv \pm m \bmod n$. In this case, $\gcd(m-y, n)$ is one of the prime factors of n.

However, by having this redundancy in the message blocks, the system is no longer susceptible to this attack, at the expense of a lower information rate. If a message m having the required redundancy is encrypted into $c \equiv m^2 \bmod n$ and is then sent to a decryption oracle, the original message m will be returned, with high probability. However, if a message m not containing the required redundancy is encrypted, with high probability none of the four square roots of $c \equiv m^2 \bmod n$ will contain the required redundancy and hence the decryption oracle will not provide the adversary with any response. On the other hand, if the attacker submits a ciphertext containing the required redundancy (in its corresponding plaintext), the decryption oracle will respond with only the valid one of the four square roots. Namely, the adversary does not obtain any sort of useful information.

3.6.3 The ElGamal Encryption Scheme

This probabilistic cryptosystem was proposed by ElGamal (1985*a*). Its security is based on the intractability of the discrete logarithm problem and on the Diffie–Hellman problem (Section B.1). As mentioned by Menezes et al. (1997), the cryptosystem can be viewed as an instance of the Diffie–Hellman key exchange, so as to establish a session key ($g^{ak} \bmod p$) and then encrypt messages using this key.

3.6.3.1 Algorithm

The algorithmic details of the ElGamal cryptosystem are presented in Algorithms 3.9 to 3.11.

Algorithm 3.9 Key generation of the ElGamal cryptosystem

1: Generate a large, random prime, p, and select a generator $g \in \mathbb{Z}_p^*$.
2: Select a random integer a, $1 \leq a \leq p-2$ and compute $g^a \bmod p$.
3: The public key is the triad (p, g, g^a) and the private key is a.

Algorithm 3.10 Encryption of the ElGamal cryptosystem

1: Break the original message m into k blocks, each represented by an integer, $m_i \in [0, p-1]$, where $i = 1, 2, \ldots, k$.
2: **for** i from 1 to k **do**
3: Randomly select an integer ε_i, $1 \leq \varepsilon_i \leq p-2$.
4: Compute $\gamma_i \equiv g^{\varepsilon_i} \bmod p$ and $\delta_i \equiv m_i (g^a)^{\varepsilon_i} \bmod p$.
5: The corresponding ciphertext is $c_i = (\gamma_i, \delta_i)$.
6: **end for**

Algorithm 3.11 Decryption of the ElGamal cryptosystem

1: **for** i from 1 to k **do**
2: Compute $\gamma_i^{p-1-a} = \gamma_i^{-a} = g^{-a\varepsilon_i} \pmod p$.
3: The corresponding plaintext block is $m_i \equiv (\gamma_i^{-a})\delta_i \bmod p$.
4: **end for**
5: Using all the m_i blocks, reconstruct the original message.

As can be seen, encryption requires two modular exponentiations per plaintext block, namely the computation of γ and δ. Since they are both included in the ciphertext, the message is expanded by a factor of 2. Hence, this scheme is quite expensive, both in terms of execution time and ciphertext size.

Moreover, care should be taken not to use the same ε more than once (encryption phase, step 3), as this can lead to a chosen-plaintext attack being launched. Assuming that two messages, m_1 and m_2, have been encrypted using the same ε, it holds that $\delta_1/\delta_2 = m_1/m_2$. Hence, if one of the two messages is known, the other can easily be computed.

3.7 One-Way Hash Functions

3.7.1 Introduction

One-way hash functions (also found under the term *message digest*) are fundamental building blocks in cryptography. They are relatively easy to compute, but significantly harder to reverse. It can be visualised as smashing a plate into pieces; however, it is not easy to put all the tiny pieces together into a plate. In asymmetric cryptography they are usually used together with digital signatures, in order to provide both authentication and assurance of data integrity.

They operate by taking a variable-length input string (*pre-image*) and they convert it to a fixed-length (generally smaller) output string, called the *hash value* (this operation is sometimes also called *compression*[6]). One of the qualities they are anticipated to have is to be *collision-free* (or *collision-resistant*); that is, to be hard to generate two pre-images with the same hash value. Moreover, a collision-resistant hash function means that it can withstand the *birthday attack* (the mathematical details of this attack are presented in Section B.2). According to Schneier (1996), hash functions of 64 bits are too small to survive birthday attacks. Most practical one-way hash functions produce 128-bit hashes, although there are some exceptions, which produce 160-bit hashes. In what follows, some of the most popular one-way hash functions will be described.

Further, hash functions may be split into:

1. *Unkeyed hash functions*, that require only one input (the message).

2. *Keyed hash functions*, that require a secret key as a secondary input, in addition to the message.

A useful *functional classification* is proposed by Menezes et al. (1997):

[6]In the context of hash functions, the term *compression* refers to a non-reversible operation. Otherwise, it refers to a fully reversible operation, that enables the extraction of the original message by processing the compressed result.

1. *Modification Detection Codes (MDCs)*

   The goal of these unkeyed hash functions is to provide an integrity-checking mechanism. They may also be found under the terms *Manipulation Detection Codes* and *Message Integrity Codes (MICs)*. They can further be divided into:

   (a) *One-way hash functions*, where it is difficult to find an input that hashes to a pre-defined hash value.

   (b) *Collision-resistant hash functions*, where it is difficult to find any two inputs that hash to the same hash value.

2. *Message Authentication Codes (MACs)*

   These keyed hash functions offer message-integrity checking and also a means for validating the authenticity of the source, without the use of any other mechanisms.

Hash functions usually comprise *rounds*, each one consisting of *steps*. In each round the steps defined in it are repeated for a given number of times. For instance, an 80-step hash function may actually be formed by 4 rounds, each consisting of 20 steps.

3.7.2 Secure Hash Algorithm (SHA)

The National Institute of Standards and Technology (NIST), along with the NSA, designed the Secure Hash Algorithm for use in the Digital Signature Standard (Section 3.8.6.1). It accepts an input of less than 2^{64} bits in length and produces a 160-bit message digest. The SHA is based on principles similar to the ones used in MD4 (Section B.2.1) and is closely modelled on that algorithm.

The changes applied to MD4 that led to the creation of SHA are the addition of an expand transformation and the addition of the previous step's output into the next step for a better avalanche effect. One could say that SHA is MD4 with an additional expand transformation, an extra round and better avalanche effect. Due to its 160-bit hash value it is very resistant to brute-force attacks (including birthday attacks).

Nevertheless, Wang, Yin & Yu (2005) showed very recently that it is possible to find collisions for SHA-1 with complexity less than 2^{69} hash operations. It is worth noting that this is the first attack on the full 80-step SHA-1, with complexity less than the 2^{80} theoretical bound.

Such scientific developments were anticipated to occur at some point, hence the SHA-2 family has already been introduced. It is used in applications where security is an absolute requirement, such as governments and the military. It comprises of SHA-224, SHA-256, SHA-384 and SHA-512, where the number denotes the length of the hash value.

3.8 Digital Signatures

3.8.1 Introduction

Despite the fact that cryptography ensures secret communication among two or more parties, it is equally important to be able to verify the true sender of a message. In real life this is mainly achieved by hand-written signatures. Their digital counterpart, however, has the advantage that is based on some secret known only to the sender (their secret key) and on the content of the message being signed. Hence, in the case of a dispute regarding the signing of a document, an unbiased third party can very easily resolve the matter, without requiring access to the signer's secret information.

3.8.2 Classification of Digital Signatures

In the book by Menezes et al. (1997), digital signatures are classified into two main categories:

1. *Digital signatures with appendix* require the original message as input to the verification algorithm.

2. *Digital signatures with message recovery* do not require the original message. In

this case, the original message is recovered from the signature itself.

Any digital signature scheme with message recovery can easily be transformed into a digital signature scheme with appendix by simply hashing the plaintext and then signing the hash value.

Digital signatures associate messages from the *message space* with signatures in the *signature space*. Hence, they can further be divided into *randomised digital signature schemes*, when messages have more than one image in the signature space; otherwise, they are called *deterministic*.

The *redundancy function* is the 1-1 mapping from the *message space* to the *signing space*. The latter is the set of elements where the signing transformations are applied to and contains more than one copy of the original message. Both the redundancy function and its reverse are publicly known, but the choice of an appropriate redundancy function is crucial for the security of the signature algorithm. Although functions are usually used in digital signatures with message recovery, they may also apply to other schemes, such as the RSA digital signature scheme.

3.8.3 Attacks on Digital Signatures

3.8.3.1 Attack Criteria

Before dealing with the different kinds of attacks that can be launched, it is worth defining the *attack criteria*, or else, what it means to break a digital signature scheme. The following cases are identified by Menezes et al. (1997):

- *Total break*: An adversary is either able to compute the private key of the signer, or finds an efficient signing algorithm, functionally equivalent to the valid signing algorithm.

- *Selective forgery*: An adversary is able to create a valid signature for a particular message or class of messages chosen *a priori*. Creating the signature does not directly involve the legitimate signer.

- *Existential forgery*: An adversary is able to forge a signature for at least one message. The adversary has little or no control over which message's signature is obtained, and the legitimate signer may be involved in the deception.

3.8.3.2 Two Basic Attacks

Having identified the attack criteria, the following two basic attacks against public-key digital signature schemes are mentioned:

1. *Key-only attacks*: In these attacks, an adversary knows only the signer's public key.

2. *Message attacks*: Here, an adversary is able to examine signatures corresponding either to known or to chosen messages. Message attacks can be further subdivided into 3 classes:

   (a) *Known-message attack*: An adversary has signatures for a set of messages which are known to the adversary but not chosen by them.

   (b) *Chosen-message attack*: An adversary obtains valid signatures from a chosen list of messages before attempting to break the signature scheme. This attack is non-adaptive in the sense that messages are chosen before any signatures are seen. These attacks against signature schemes are analogous to chosen-ciphertext attacks against public-key encryption schemes.

   (c) *Adaptive chosen-message attack*: An adversary is allowed to use the signer as an oracle; the adversary may request signatures of messages which depend on the signer's public key and they may request signatures of messages which depend on previously obtained signatures or messages. These attacks are analogous to the adaptive chosen-ciphertext attacks against public-key cryptosystems.

3.8.4 The RSA Digital Signature Scheme

3.8.4.1 Background

The first method introduced was the RSA digital signature scheme, which is a deterministic scheme with message recovery. It is derived from the respective encryption scheme by simply reversing the roles of encryption and decryption[7]. Signing is performed with the secret key and verification with the public one. Since this thesis is closely related to RSA as an encryption scheme, it is worth looking at RSA as a digital signature scheme.

3.8.4.2 Algorithm

The key generation phase is identical to that of the RSA encryption scheme. In the signature generation phase the use of a redundancy function is included.

Algorithm 3.12 Key generation of the RSA digital signature scheme

1: Randomly generate two, large primes, p and q, of roughly the same size.
2: Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$.
3: Randomly select an integer e, $1 < e < \phi(n)$, relatively prime to $\phi(n)$; namely, $\gcd(\phi(n), e) = 1$.
4: Using the Extended Euclidean Algorithm generate the unique integer d, $1 < d < \phi(n)$, such that $ed \equiv 1 \pmod{\phi(n)}$.
5: The public key is the pair (e, d) and the private one is d.

Algorithm 3.13 Signature generation of the RSA digital signature scheme

1: Compute $\tilde{m} = R(m)$, an integer in the range $[0, n-1]$, where R is a redundancy function.
2: Compute $s \equiv \tilde{m}^d \bmod n$.
3: The signature for message m is s.

Algorithm 3.14 Signature verification of the RSA digital signature scheme

1: Retrieve the signer's (authentic) public key, (n, e).
2: Compute $\tilde{m} \equiv s^e \bmod n$.
3: Accept the signature only if $\tilde{m} \in \mathcal{M}_{\mathcal{R}}$, where $\mathcal{M}_{\mathcal{R}}$ is the image of R.
4: The original message is $m = R^{-1}(\tilde{m})$.

[7]Both the message space and the ciphertext space are in $\mathbb{Z}_n = [0, n-1]$, $n = pq$ and the encryption function is a bijection.

In their book, Menezes et al. (1997) provide the proof of why this scheme works. It is as follows: If s is a signature for a message, m, then $s \equiv \tilde{m}^d \bmod n$, where $\tilde{m} = R(m)$. Since $ed \equiv 1 \pmod{\phi(n)}$, $s^e \equiv \tilde{m}^{ed} \equiv \tilde{m} \pmod{n}$. Finally, $R^{-1}(\tilde{m}) = R^{-1}(R(m)) = m$.

3.8.4.3 Attacks on RSA Signatures

1. *Factorisation of n*: An adversary who has computed p and q can easily calculate $\phi(n)$ and, by using the extended Euclidean algorithm, they can compute the private key, d. The reason for choosing p and q to be of approximately the same size is to make the factorisation of n a computationally infeasible task.

2. *Multiplicative Property of RSA*: Both the RSA encryption scheme and the RSA digital signature scheme have the following multiplicative property:

   Let $s_1 \equiv m_1^d \pmod{n}$ and $s_2 \equiv m_2^d \pmod{n}$ be the signatures of messages m_1 and m_2, respectively. Then, $s \equiv s_1 s_2 \pmod{n} \equiv (m_1 m_2)^d \pmod{n}$. Hence, if $m = m_1 m_2$ has the proper redundancy (namely, $m \in \mathcal{M}_R$), then s is a valid signature for it. In order to block this attack, the redundancy function, R, should not be multiplicative. That is, $\forall a, b \in \mathcal{M}, R(ab) \neq R(a)R(b)$, where $\mathcal{M}$ is the message space. Nevertheless, this is still inadequate for security.

Finally, there are also attacks based on the existence of an insecure redundancy function. However, they will not be discussed in detail, since they are beyond the scope of this thesis.

3.8.5 The ElGamal Digital Signature Scheme

The ElGamal signature scheme (ElGamal 1985*a*, ElGamal 1985*b*) is a randomised signature mechanism. It generates digital signatures with appendix on binary messages of arbitrary length, and requires a hash function $h : \{0,1\}^* \rightarrow \mathbb{Z}_p$, where p is a large prime.

3.8.5.1 Algorithm

Algorithm 3.15 Key generation of the ElGamal digital signature scheme

1: Generate a large, random prime p and a generator g of the multiplicative group $\mathbb{Z}_p^*$.
1: Select a random integer a, $1 \leq a \leq p-2$.
2: Compute $y \equiv g^a \bmod p$.
3: The public key is the triad (p, g, y) and the private key is a.

Algorithm 3.16 Signature generation of the ElGamal digital signature scheme

1: Select a random secret integer k, $1 \leq k \leq p-2$, with $\gcd(k, p-1) = 1$.
2: Compute $r \equiv g^k \bmod p$.
3: Compute $k^{-1} \bmod (p-1)$.
4: Compute $s \equiv k^{-1}(h(m) - ar) \bmod (p-1)$.
5: The pair (r, s) is the signature for message m.

Algorithm 3.17 Signature verification of the ElGamal digital signature scheme

Let (r, s) be the signature for message m. In order to verify it, one should do the following:

1: Obtain the signer's (authentic) public key, (p, g, y).
2: Verify that $1 \leq r \leq p-1$; if not, reject the signature.
3: Compute $u_1 \equiv y^r r^s \bmod p$.
4: Compute $h(m)$ and $u_2 \equiv g^{h(m)} \bmod p$.
5: Accept the signature if and only if $u_1 = u_2$.

3.8.5.2 Variants of the ElGamal Scheme

Apart from the original scheme many variations exist, such as the *Schnorr Signature Scheme* (Schnorr 1991) and the *Digital Signature Algorithm* (Section 3.8.6). As mentioned by Menezes et al. (1997), the variants differ mainly in the signing equation. After suitable re-arrangements, the signing equation can be re-written as $u \equiv av + kw \bmod (p-1)$. Alterations of the basic algorithm are found mainly in the calculation of the parameters u, v and w.

3.8.6 The Digital Signature Algorithm (DSA)

3.8.6.1 Historical Background

In August of 1991, the U.S. National Institute of Standards and Technology (NIST) proposed the Digital Signature Algorithm, DSA (Kravitz 1993). The DSA has become a U.S. Federal Information Processing Standard (FIPS 186) called the *Digital Signature Standard* (DSS) and is the first digital signature standard recognised by any government. The algorithm is a variant of the ElGamal scheme and is a probabilistic digital signature scheme with appendix. The signature mechanism requires a hash function $h : \{0,1\}^* \rightarrow \mathbb{Z}_q$ for some integer q and the one used by DSS is the Secure Hash Algorithm (SHA), described in Section 3.7.2.

The security of DSA relies on two distinct but related discrete logarithm problems. One is the logarithm problem in $\mathbb{Z}_p^*$, where powerful index-calculus methods apply; the other one is the logarithm problem in the cyclic subgroup of order q, where the best current methods run in 'square-root' time (Menezes et al. 1997). Furthermore, FIPS 186 included both recommendations and restrictions regarding its security. Hence, since 1996 a modulus of at least 768 bits is recommended, but primes larger than 1024 bits are not permitted. The reason for doing so is for enabling security and intelligence agencies to decrypt messages of high importance to them.

Regarding its performance, as mentioned by Menezes et al. (1997), DSA has the advantage that the exponentiation can be pre-computed and need not be done at the time of the signature generation, something that is not feasible with any RSA-based signature scheme. Further savings can be made by performing the two exponentiations simultaneously.

3.8.6.2 Algorithm

Algorithm 3.18 Key generation of the DSA digital signature scheme

1: Select a prime number q such that $2^{159} < q < 2^{160}$.
2: Choose t so that $0 \leq t \leq 8$ and select a prime number p where $2^{511+64t} < p < 2^{512+64t}$, with the property that q divides $(p-1)$.
3: **repeat** {Select an appropriate generator b.}
4: Select $g \in \mathbb{Z}_p^*$ and compute $b \equiv g^{(p-1)/q} \pmod{p}$.
5: **until** $b \neq 1$
6: Select a random integer a, such that $1 \leq a \leq q-1$.
7: Compute $y \leftarrow b^a \pmod{p}$.
8: The public key is (p, q, b, y); the private key is a.

Algorithm 3.19 Signature generation of the DSA digital signature scheme

1: Select a random, secret integer k, $0 < k < q$.
2: Compute $r \leftarrow (b^k \bmod p) \bmod q$.
3: Compute $k^{-1} \bmod q$.
4: Compute $s \leftarrow k^{-1}(h(m) + ar) \bmod q$, where m is the message to be signed.
5: The signature is the pair (r, s).

Algorithm 3.20 Signature verification of the DSA digital signature scheme

To verify a signature (r, s) on a message m, one should do the following:

1: Obtain the signer's authentic public key (p, q, b, y).
2: Verify that $0 < r < q$ and $0 < s < q$; if not, reject the signature.
3: Compute $w \leftarrow s^{-1} \bmod q$ and $h(m)$.
4: Compute $u_1 \leftarrow (wh(m)) \bmod q$ and $u_2 \leftarrow rw \bmod q$.
5: Compute $v \leftarrow (b^{u_1} y^{u_2} \bmod p) \bmod q$.
6: Accept the signature if and only if $v = r$.

3.9 Error-Correcting Codes and Cryptography

3.9.1 Introduction

Any kind of electronic data transmission is subject to the noise of the transmission channel. This noise can modify the data being transmitted and hence there is a need for a technique that will enable the recovery of the original data. For this reason, error-correcting codes have been proposed, which expand the data in a systematic way with

extra information that supports perfect retrieval of the original data. At this point it would be useful to introduce some more aspects of error-correcting codes, before presenting cryptosystems that are based on them.

3.9.2 More on Error-Correcting Codes

3.9.2.1 The Binary Symmetric Channel

Several channel models have been proposed and used, in order to cover the need for simulating communications. One such model is the Binary Symmetric Channel (BSC), the simplest model of a channel with error. It is a *memoryless* channel; that is, the probability distribution of the output depends only on the input at that time and is conditionally independent of previous channel inputs or outputs. It transmits each bit correctly with probability $(1-p)$ and incorrectly with probability p. Namely,

$$P(t=0 \mid r=0) = P(t=1 \mid r=1) = 1-f$$

and

$$P(t=0 \mid r=1) = P(t=1 \mid r=0) = f,$$

where t and r are the transmitted and received bits, respectively. Despite its simplicity, it is suitable for the scope of the required simulations.

3.9.2.2 Low-Density Parity-Check Codes

Due to the vast variety of of error-correcting codes, this work will focus on low-density parity-check (LDPC) codes. Gallager codes will be used as a characteristic example of this code family. Apart from being quite popular and easily understood, they can communicate with lower decoder-error probability, higher information rate and with lower transmitted-signal-to-noise ratio. Furthermore, the decoding complexity of LDPC codes is not high and the encoding complexity can be made low (MacKay 2003).

LDPC codes are *block codes*, which means that they convert a sequence of K bits into a transmitted sequence of N bits, where $K < N$. They are defined by a sparse

parity-check matrix, H, of dimensionality $M \times N$. The number of non-zero elements per column or row defines the column or row *weight* respectively. In the case where every column has the same weight j and every row has the same weight k, then the code is a *regular* one. Otherwise, if either column weight or row weight are not constant, then the code is an *irregular* one. Based on this parity-check matrix a (N, K) error-correcting code can be created. That is, K source bits are packed into a codeword of N bits with error-correcting properties. The parity-check matrix H is required for both the generation of the code's generator matrix G (a boolean matrix of dimensionality $K \times N$), as well as for the decoding process. The $K \times N$ generator matrix G for the encoder can be specified by reducing H into a *systematic* form of $H_s = [P \mid \mathbf{I}_M]$. The corresponding generator matrix (also in systematic form) can then be constructed as $G = [\mathbf{I}_K \mid P^\mathsf{T}]$. It should also be noted that for a given H more than one G matrices may exist. Nevertheless, they all define the same code and the error-correcting capabilities do not depend on the choice of the generator matrix, but solely on H.

Since any LDPC code is defined by an $M \times N$ parity-check matrix, H, it can be represented by a graph consisting of N bit nodes and M check nodes. This graph is bipartite, namely bit nodes connect only to check nodes and vice versa. Check node f_i is connected to bit node c_j if and only if the element h_{ij} of H is a 1. Figure 3.1 presents one such example.

At this point it is worth presenting the computational complexity of LDPC codes. For a code that produces N-bit long codewords, the complexity for generating the G matrix is $\mathcal{O}(N^3)$ operations. Encoding the source message into codewords requires $\mathcal{O}(N^2)$ operations. Each decoded bit requires about $6jr/R$ operations, where j is the column weight of the G matrix, R is the code rate and r is the number of the required iterations during the decoding process (MacKay 2003).

Moreover, there is also the possibility of using LDPC codes over finite fields $GF(q)$, in order to achieve better error-correcting performance. As is mentioned by MacKay (2003), variables and check variables can be grouped together into metavariables and

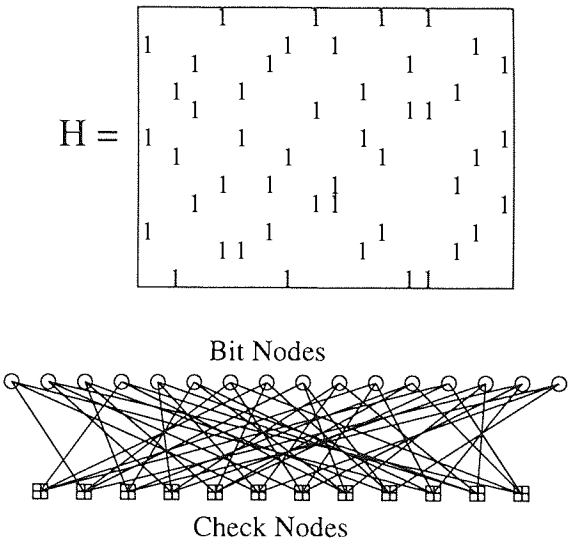


Figure 3.1: A low-density parity-check matrix (H) and the corresponding graph of a rate 1/4 LDPC code, with block length $N = 16$ and $M = 12$ constraints.

metachecks, respectively, consisting of several binary variables that will effectively represent elements of the $GF(q)$ field. In this way, a sparse graph can be constructed by connecting metavariables to metachecks, but will offer much more freedom in the way the variables and checks within are connected.

Any Galois Field can be used, hence $q = p^m$, where p a prime and m a positive integer. However, fields of the form $q = 2^m$ are usually preferred, due to the use of binary channels. In this way, one q-ary symbol of $GF(q)$ can be transmitted for every m uses of the channel. For instance, each element of $GF(2^3)$ can be represented by 3 binary bits.

The number of bits (say k) that the metavariables and metachecks will consist of is then chosen so that $2^k = q$, and a suitable mapping between the elements of the Galois Field and binary message symbols is created. One such mapping scheme is shown in Table 3.1. Low-density parity-check matrices can now be defined using elements of $GF(q)$, and the binary messages are translated into $GF(q)$. Thus, an $K \times N$ parity-check matrix over $GF(2^m)$ can be turned into a binary parity-check matrix of dimensionality $mK \times mN$.

In the decoding stage the decoder will effectively yield probabilities over conjunc-

Element	Polynomial	Bit Pattern	Matrix Pattern
0	0	00	00 00
1	1	01	10 01
A	x	10	11 10
B	x+1	11	01 11

Table 3.1: Example representations of the elements of $GF(4)$, as well as suitable binary mappings.

tions of binary variables. In the above example, these will represent the likelihoods of the 2^k alternative states of the k-bit long metavariables. The computational cost for decoding in $GF(q)$ is $q \log_2 q$, if the appropriate Fourier transform is used. As is mentioned by Davey & MacKay (1998), the resulting LDPC codes over $GF(4)$, $GF(8)$ and $GF(16)$ can significantly outperform the equivalent binary codes.

3.9.2.3 The Sum-Product Algorithm

The general version of the sum-product algorithm is a message-passing algorithm, in which simple messages are passed among simple processors, in order to solve a global problem. This algorithm works well on tree-like structures. A tailored version of the algorithm is applicable to the decoding process of LDPC codes, as they can be represented by a tree-like graph consisting of bit nodes and check nodes.

There are two main approaches to decoding:

- *Codeword decoding*: Infer which codeword t was transmitted given the received signal r. It can be described as $P(t) \propto \mathbb{1}[Ht = 0 \bmod 2]$.

- *Syndrome*[8] *decoding*: Find the most probable noise vector n, satisfying the equation $Hn = z$, where H is the parity-check matrix and z the syndrome vector. It can be described as $P(t) \propto P(n)\mathbb{1}[Hn = z]$.

[8] *Syndrome* is the pattern of violations of the parity bits.

MacKay (2003) prefers the syndrome decoding approach, because it possesses the advantage of not requiring the implementation of an encoder for a code in order to be able to simulate a decoding problem realistically.

What is actually needed is to compute the marginal posterior probabilities $P(x_n = 1 \mid z, H)$ for each n. Performing accurate computations is quite hard because of the many cycles that the graph contains. Nevertheless, the algorithm is applied as if there were no cycles, under the assumption that the introduced errors may be relatively small. Even with inaccurate probabilities the decoding process can be successful, as the most probable codeword is usually the correct one.

The recommended decoding procedure is to set $\hat{x}_n$ to 1 if the probability of that bit to be 1 is greater than 0.5 and check if $H\hat{x} = z$ is satisfied (for all bits). Should that be true, decoding is deemed successful and the process halts. Otherwise, if a maximum number of iterations has been reached, decoding has been unsuccessful.

3.9.3 Cryptosystems Based on Error-Correcting Codes

The idea of *deliberately* introducing random noise to a message and then using an error-correcting code to restore it – effectively creating a new cryptographic scheme – was initially introduced by McEliece (1978), who used *Goppa codes* in order to retrieve the original message. A variant of this system was proposed by Kabashima et al. (2000). A further improvement to the McEliece cryptosystem was the use of Reducible Rank Codes, which dramatically decreased the key size (Gabidulin et al. 2003). Both these attempts have created probabilistic cryptosystems that can successfully face the partial information leakage problem.

In order to form such a cryptosystem, the encoding matrix of an error-correcting code is used. By either multiplying it with permutation and scrambling matrices, or by adding corrupting matrices to it, the public key is formed. The main disadvantage of such matrix-based (also called *lattice-based*) cryptosystems is that they still use larger keys than the majority of public-key schemes. For instance, the security of the

McEliece cryptosystem depends on the values of three parameters. The security level in this case reflects the computational effort required for breaking the system. The recommended settings, which maximise the security level, produce a significantly large public key of approximately 2^{19} bits in length. In addition, the message expansion factor is about 1.6 (Menezes et al. 1997). Canteaut & Sendrier (1998) mention that breaking the McEliece cryptosystem with its original parameters requires $2^{64.2}$ binary operations. The same computational effort would be required to factorise an RSA modulus of approximately 124 bits (412 digits) in length, using the GNFS method (the related calculations are presented in Section F.1).

The public key of the McEliece cryptosystem is formed by manipulation of the code generator matrix. Niederreiter (1986) proposed a variant of this cryptosystem, where the public key is formed by manipulation of the parity-check matrix. From the security point of view, both cryptosystems are equivalent (Li, Deng & Wang 1994). However, as argued by Canteaut & Sendrier (1998), for given parameters, the Niederreiter cryptosystem has several advantages over its McEliece counterpart. Firstly, it allows a public key in a systematic form, without compromising security, whereas in the McEliece version this could be exploited to reveal part of the plaintext. Hence, in the Niederreiter scheme, the key is $(n-k)/n$ times smaller than that of McEliece. Furthermore, due to the systematic form of the public key matrix and the low-weight ciphertext vector, the Niederreiter scheme features a significantly reduced computational cost for the encryption operation. One other disadvantage of the McEliece cryptosystem is that it is easy to recover the plaintext, if it has been encrypted twice with the same public key. This attack is not feasible with the Niederreiter scheme, due its deterministic nature. Nevertheless, its property of being deterministic can be argued to be a disadvantage, under certain circumstances.

Courtois et al. (2001) showed how to produce short McEliece-based digital signatures that offer good security – about 2^{80} CPU operations for the attack proposed by Canteaut & Chabaud (1998). The main drawbacks are the long execution times

and the large public key size. In particular, signature generation time varies from a few tens of seconds to 1 minute. The signature can be shortened by omitting some information that it is possible to recover during the verification phase. However, this slows down the verification process significantly. More specifically, the verification time for a 132-bit signature is less than 1 μs, whereas, if the signature is shortened down to 86 bits, the required time is about 1 second. For the signatures mentioned above, a public key of approximately 1 MB was required. Therefore, although its security is claimed to be good enough, both the public key size and the signature time pose questions of practicality for larger keys.

3.10 Concluding Remarks

This chapter has presented the two main categories of cryptography, symmetric and asymmetric cryptosystems. The latter was presented in more detail, since the proposed cryptosystem is an asymmetric one. Several asymmetric cryptosystems were described in detail, as they relate directly to the proposed cryptosystem. Examples were given of how the flaw of 'plain-old' RSA can be exploited by adversaries/eavesdroppers, so as to successfully cryptanalyse the system and retrieve the plaintext. Moreover, other attempts made in the past for dealing with this problem were mentioned, as well as their effectiveness. Combining hash functions with digital signatures can increase the security of a system. For that reason, some of the most popular digital signature schemes and hash functions together with some possible attacks on them were presented extensively. The contribution of coding theory to cryptography is significant, since several cryptosystems have emerged from it. For that reason, characteristic examples of cryptosystems that combined error-correcting codes and cryptography were presented, as well as the required background.

The need for preventing cryptosystems from leaking partial information has been made apparent, since this vulnerability has repeatedly led to successful cryptanalysis of various schemes. What is more, partial information leakage may not only be due

to design flaws, but also due to insecure implementation. Hence, further analysis to identify any unforseen vulnerabilities is desirable, as well as ways for preventing them from being exploited.

In an attempt to convert RSA into a probabilistic cryptosystem most approaches randomise the plaintext during the required pre-processing stage, before encryption commences. What is more, the main primitives used for achieving this randomisation are usually hash functions. An alternative technique is proposed in this thesis, where the random component is inserted after the encryption process. In addition, an alternative way is proposed for randomising the plaintext in the pre-processing stage, where the core principle is the exploitation of the properties of error-correcting codes.

Chapter 4

Using the QRP for Hiding Partial Information

4.1 Introduction

This chapter describes a method for facing the information leakage problem in RSA, by combining it with the Quadratic Residuosity Problem.

This chapter is dedicated to the description and analysis of the proposed method for secure communications between two or more parties, through an untrusted network. We also test the strength of the end system and show that it executes within an acceptable time limit.

In what follows, the design objectives are specified, followed by a proposal for implementing a cryptographic scheme, fulfilling these objectives. A prototype built according to the specifications is compared to other cryptosystems sharing similar characteristics and based on the experimental data an evaluation of the system is performed.

4.2 Proposed System

4.2.1 Key Principles

In the proposed system, the principal idea is inserting 'opaque blocks' to the ciphertext *after* the encryption process. Considered at a lower level it is like *blending* two or more ciphertexts and then using the QRP so as to distinguish the intended ciphertext (and thus the corresponding message) from the rest. Alternatively, the plaintext message could randomly be mixed with the opaque blocks and then encrypted in one go, provided that a tagging scheme is included, which will allow the distinction between the intended message and the opaque blocks.

The key difference that distinguishes the proposed system from similar ones (as mentioned in Section 3.5.4) is that the plaintext does not undergo any modifications at all. The plaintext is encrypted as is and mixed with other blocks. The tagging scheme is based on quadratic residues.

What is presented in the sections that follow is a system that combines the QRP and RSA, in order to hide partial information. It may also seem that this technique can be applied to any deterministic public-key encryption scheme that requires some pre-processing of the plaintext before it gets encrypted. However, one should consider a couple of things before attempting such an implementation.

The first thing that should be considered is the expansion rate of the public-key cryptosystem which is to be combined with the QRP, as this would have a direct effect on the performance of the end system. For instance, the ElGamal cryptosystem has a message expansion factor of 2 (Menezes et al. 1997). Hence, the end system would produce more ciphertext after the encryption phase and would consequently take longer to execute during the decryption phase. Secondly, it would not be sensible to apply this technique to elliptic curve cryptosystems. This is due to the fact that the key that the elliptic curve cryptosystem would require is smaller than the one required for implementing the QRP technique. On the one hand, using a smaller key for the

QRP would not provide adequate security. On the other hand, using a large key for the elliptic curve cryptosystem would introduce unnecessary complexity and would not exploit the main advantage of elliptic curve cryptosystems, which is to provide sufficient security by using smaller keys than other public-key encryption schemes.

4.2.2 Insertion of the Opaque Blocks

Regarding the quality of the opaque blocks being inserted, there are two main options. The first one is to have totally randomly generated opaque blocks. That is, a random sequence of printable ASCII characters in the ciphertext, which obey the format of a valid ciphertext. This approach is not very effective, given that an attacker has full knowledge of how the system operates. They can therefore attack the scheme by trying plaintexts from a restricted set of possibilities and simply look for matches of their guesses. If a match is found, then the attacker has won, since they know that the remaining of the ciphertext is meaningless ASCII characters.

A much better approach would be to have one or more plaintexts, that belong to the same contextual space as the intended plaintext. Hence, the proposed scheme tries to put any potential attacker in a position such that they have more than one correct guess and no clues as to which one the intended message may be. In addition, since the intended message is tagged, it is also possible to use it as opaque blocks, provided that other opaque messages do exist in the ciphertext.

One other issue is the amount of the opaque blocks that should be inserted. It is self-explanatory that this will be a trade-off between the system's performance and the level of security that it will offer. In order to improve the system's efficiency, there are a couple of different approaches for determining the quantity of the opaque blocks that should be inserted. The simplest one is to have several plaintexts to serve as opaque blocks that are all encrypted. This method produces the most redundant ciphertext. The second approach is to produce opaque blocks only for blocks that contain context-related keywords. In this way, multiple occurrences of identical blocks will be avoided

and the amount of the inserted opaque blocks could be increased. Nevertheless, in order to implement a fully automated version of the second approach, a rather sophisticated system is required, capable of determining the context of a message and generating the opaque blocks accordingly. As this is an issue closer to artificial intelligence than to cryptography, it will not be discussed any further. Alternatively, the user could be asked for supplying additional opaque blocks, when setting up the system. Of course, this is quite a tedious task from the user's side.

Since the insertion of opaque blocks is expected to have a direct effect on the system's security and performance, this issue is further analysed, both in the security and in the performance sections (Section 4.3.3 and Section 4.3.7 respectively).

4.2.3 Algorithm

The algorithm for the proposed system[1] is as follows:

Let $M_i, i = 1, 2, \ldots, k_1$ be the *intended* message blocks and $F_j, j = 1, 2, \ldots, k_2$ be the *false* message blocks, namely the opaque blocks. Note that k_1 could equal k_2, although this is not an absolute requirement. Nevertheless, during the encryption process, *all* blocks must be consumed. It should also be clarified that a message may (and usually does) consist of more than one block. The number of additional messages that serve as opaque blocks, determines the opacity ratio (λ) and each message block is used only once.

4.2.3.1 Key Generation

Let $n = pq$, where p, q are large primes, and e, d be the RSA encryption (public) and decryption (secret) key respectively, such that

$$\gcd(e, (p-1)(q-1)) = 1$$

[1] In what follows, the proposed system is also referred to by its name, *Pythia*. Reasons for its naming can be found in Appendix C.

and

$$ed \equiv 1 \bmod (p-1)(q-1).$$

Algorithm 4.1 Pythia key generation

1: Select $y \in \mathbb{Z}/n\mathbb{Z}$, such that $y \in \overline{Q}_n$ and $\left(\frac{y}{n}\right) = 1$. Hence, y is a pseudosquare[2] modulo n.
2: The public key is the triad (n, y, e) and the secret key is the triad (p, q, d).

It should be noted that the number of elements in $\mathbb{Z}/n\mathbb{Z}$ that have multiplicative inverses is $\phi(n)$ and the number of pseudosquares is $\phi(n)/4$. Therefore, choosing a number y that satisfies the appropriate conditions can be done with a probability of 0.25. Once $y \in \mathbb{Z}/n\mathbb{Z}$ has been chosen, two evaluations of the Legendre symbol are required (one for p and one for q), in order to determine whether $\left(\frac{y}{n}\right) = 1$. If the latter holds, then y was a successful choice. Otherwise, a new value of y is selected.

4.2.3.2 Encryption

Algorithm 4.2 Pythia encryption

Require: $1 \leq v \leq k_1 + k_2$, $1 \leq i \leq k_1$, $1 \leq j \leq k_2$.
1: Set $v \leftarrow 1$, $i \leftarrow 1$, $j \leftarrow 1$.
2: **while** $i \leq k_1$ **do**
3: Generate a random number $r \in \{0, 1\}$ with equal probability.
4: **if** $r = 1$, OR (if $r = 0$ AND $j \leq k_2$) **then**
5: Randomly choose $x_v \in (\mathbb{Z}/n\mathbb{Z})^*$, so that $x_v \notin \{x_1, \ldots, x_{v-1}\}$ for $v \geq 2$. The ciphertext corresponding to the plaintext block is

$$c_v = (a_v, b_v) = \begin{cases} (x_v^2 \bmod n,\ M_i^e \bmod n), & \text{if } r = 1 \\ (yx_v^2 \bmod n,\ F_j^e \bmod n), & \text{if } r = 0 \end{cases}$$

6: **end if**
7: Increment v and i or j as appropriate (i.e. increment i if $r = 1$ and j if $r = 0$).
8: **end while**

One point worth clarifying regarding the encryption phase is the generation of the random number r. Mathematically, it should hold that $P(r = 0) = P(r = 1) = 0.5$. Therefore, even in the extreme case where a sequence of zeroes is produced and all the

[2]For a detailed description of the underlying background, as well as related definitions, the reader is referred to Section A.2.2.

opaque blocks have been consumed, the algorithm will keep looping until a sequence of ones is produced, long enough to consume all the intended message blocks.

4.2.3.3 Decryption

Let u be the number of blocks that were encrypted. It follows that $k_1+1 \leq u \leq k_1+k_2$, since for $k_1 = u$ the system is, effectively, plain RSA.

Algorithm 4.3 Pythia decryption

1: **for** i from 1 to u **do**
2: For a ciphertext block $c_i = (a_i, b_i)$, generated by the process mentioned above, evaluate the Legendre symbol $e'_i = \left(\frac{a_i}{p}\right)$ $\left(\text{or } e''_i = \left(\frac{a_i}{q}\right)\right)$.
3: The respective plaintext block is

$$m_i = \begin{cases} b_i^d \bmod n, & \text{if } e'_i = 1 \text{ (or } e''_i = 1) \\ \text{Empty}, & \text{if } e'_i = -1 \text{ (or } e''_i = -1) \end{cases}$$

4: **end for**
5: Use the m_i blocks to reconstruct the original message.

This particular ordering of the two types of ciphertext blocks was chosen (namely, one QRP-generated block followed by one RSA-generated block and so on), since it derives from the relationship between QRP-generated ciphertext and RSA-generated ciphertext: the former acts like a tag, indicating whether a particular RSA-generated ciphertext block belongs to the intended message, or is an opaque block. The QRP-generated ciphertext will always be evaluated, in order to determine whether the next ciphertext block (produced by RSA) belongs to the intended message, or not. If it proves to be an opaque block, then there is no need to waste computational effort on decrypting it; it can simply be discarded.

Additionally, it suffices to evaluate just one Legendre symbol and not both (reasons for doing so are mentioned in Section 3.6.1), as this increases the decryption speed.

4.3 Overview of the Proposed System

4.3.1 Miscellaneous Implementation Issues

Certain algorithms required for this system are available in the literature. In his book, Yan (2000) provides algorithms for implementing RSA and Quadratic Residuosity cryptosystems. Stallings (1999) provides a comprehensive algorithm for the Rabin–Miller probabilistic primality test and Herkommer (1999) provides an algorithm for evaluating the Legendre symbol (needed for Quadratic Residuosity-based cryptosystems).

The multiple precision integer library chosen was *BigNum* (Diommisfois 1988), which is written in C and C++. This particular library was chosen because it is compact, it provides all the required functions for manipulating large integers and because it offers *operator overloading*, a feature that makes programming easier. The language chosen for the main program is a combination of C and C++, since it offers fast program execution and efficient pointer manipulation.

The use of files has been preferred to standard output, because it is easier to interface with a Graphical User Interface. The input and output files are read by a Tcl/Tk script and they are then displayed in the interface editor window.

4.3.2 Measuring Success/Failure

The successful operation of the end system will be one criterion. It should be able to encrypt a message, safely recover the original plaintext from the ciphertext and detect any possible tampering that the ciphertext has undergone. The encryption speed of the system is another crucial factor, since user interaction is involved. An execution time of more than a few minutes will be impractical. Finally, the system's execution speed will be compared to similar cryptosystems (implemented using the same library), so as to get a relative performance measure.

In most of the literature studied for this work the three most popular methods that

authors used for comparing public-key cryptosystems were the required arithmetic operations, the size of the ciphertext with respect to the amount of plaintext being encrypted and the execution time needed for encrypting/decrypting a specific plaintext message. Hence, in order to measure the performance of the proposed scheme, the size of the ciphertext and the respective execution times will be used. These two factors will determine its efficiency and will thus allow us to compare it with other, similar schemes.

Furthermore, the proposed system will be compared to implementations of the Rabin, Goldwasser–Micali and plain RSA cryptosystems, in terms of ciphertext size and execution speed. The first two were chosen since they are based on quadratic residues, like the proposed scheme. The application will be implemented by making use of the same multiple precision integer library, in order to produce comparable results.

The importance of security against an adaptive chosen-ciphertext attack was also presented in the literature review. The proposed system should therefore be able to withstand such an attack before it can be claimed to be secure. In order to achieve this goal, techniques involving combining digital signatures and hash functions can be used. Nevertheless, we did not undertake any actual attempts to break the system, as it is a very complex, tedious and time-consuming process; future research could possibly deal with that issue.

4.3.3 Security

4.3.3.1 Overall System Security

In secret-key cryptography, the addition of a primitive building block to an existing cryptosystem can have degrading effects on the overall system's cryptographic strength. In the case of the proposed system, however, there is a fundamental difference: although RSA is combined with the Quadratic Residuosity Problem (QRP), the two schemes do not interact with each other; they may have some parameters in common, such as the

modulus, n and the two primes, p and q, yet they can still be regarded as two stand-alone cryptosystems. This is due to the fact that no data flows exist that go from one system to the other, or vice versa. Hence, the security of the compound system can be justified by examining independently the security of each of its components.

In his report, Shoup (1998) argues that the preferred approach of modern, mathematical cryptography is the *reductionist approach*, where one shows with mathematical rigour that any attack that can break the cryptosystem can be transformed into an efficient program to solve the underlying, well-studied problem (large integer factorisation, for example), which is widely believed to be very hard. This method will be followed in proving that the proposed system is secure.

To date, the only way of successfully cryptanalysing RSA is by factorising the modulus, n. This statement excludes implementations of RSA, which due to their poor choice of parameters (e and d, for instance) render RSA insecure. Similarly, for QRP-based cryptosystems, the only known method for distinguishing squares from pseudosquares is also by factorising the modulus n. It therefore follows that breaking the proposed system will be at least as hard as factorising the modulus n.

Finally, both the operating system and the key exchange protocols are assumed to be secure and any issues relating to them will not be dealt with.

4.3.3.2 Elementary Security Issues

In order to implement the proposed system as described in Section 4.2, there are a couple of obvious attacks that need to be addressed before it can offer reasonable security.

The first one, as mentioned in Section 3.6.1, is where the attacker multiplies the QRP-generated ciphertext with the pseudosquare, y. If the number of the opaque blocks equals that of the intended message, then the receiver will get the wrong message. Otherwise, the receiver will get a mixture of two or more messages, which will clearly indicate that the ciphertext has been altered.

The second attack is where the attacker re-arranges, either the RSA-generated blocks, or the QRP-generated ones. The result depends on the number of blocks that constitute each message, including the one(s) treated as opaque blocks. Hence, if each message consists of just one block, the receiver will get the wrong message. However, if each message consists of two or more blocks, the probability of getting them in an order such that it will not cause any inconsistencies, decreases.

One other subtle issue is to prevent having multiple occurrences of a QRP numbers in the ciphertext (although this is not very likely to occur in practice, given the large number of elements in $\mathbb{Z}/n\mathbb{Z}$). Such a case would give away relationships of certain parts of the ciphertext, but would also minimise slightly the cryptanalyst's work. This issue can easily be faced by comparing each newly-generated QRP number to the previous ones. Generation should be repeated in case of a match.

4.3.3.3 The Adaptive Chosen-Ciphertext Attack

In what follows, different ways of successfully performing such an attack will be identified, thus allowing us to determine what needs to be done in order to face this problem.

Launching The Attack

Let $\mathcal{M}$ be the set of all possible messages $m : \{0,1\}^{k_0}$ and let $\mu = |\mathcal{M}|$. Let $C : \{0,1\}^{k_0} \longrightarrow \{0,1\}^{k_1}$ be the RSA encryption function, with $k_0 < k_1$ (this precondition follows from the original RSA design). Finally, let s_1 be the intended message and $\delta \in [2, \mu]$, so that s_δ is a message that can be used as opaque blocks for the encryption process. For the sake of simplicity, let $\mathcal{S}$ be the set of all these messages, $\{s_\delta\}$.

Launching the attack is based on the following:

1. The keys are generated for the cryptosystem. The adversary obtains the public key, but not the private key.

2. The adversary has access to an encryption device.

3. They are allowed to submit an arbitrary number of plaintexts for encryption and have access to the respective ciphertexts. Moreover, they can modify their submissions based on the results they get. The only restriction applied is that the QRP numbers ($\in \mathbb{Z}/n\mathbb{Z}$) are chosen randomly by the device and no-one can interfere in this process. However, once generated, the QRP numbers can be copied and used for constructing other ciphertexts.

4. Furthermore, they have access to a decryption oracle, which will decrypt any ciphertext submitted to it.

5. Finally, they are allowed to submit an arbitrary number of ciphertexts for decryption and modify the ones they submits based on the results they get. It should be clarified that this set of ciphertexts excludes the one they are trying to acquire some information about.

The adversary has access to the public key e, and hence can compute $C(g_i)$ on their guesses, with $g_i \in \mathcal{M}$ and obtain the respective encryptions, g_i'. Once the encryptions and the appropriate comparisons have been performed, the adversary may have one or more matches with messages from $\mathcal{M}$. In the case where they have multiple matches, they have not gained anything, since the only way of determining which one is the intended message is by evaluating the accompanying QRP numbers to see whether they are squares or pseudosquares.

To date, the only way of performing that distinction is by evaluating the Legendre symbol $\left(\frac{a}{p}\right)$ or $\left(\frac{a}{q}\right)$, where a is the QRP number in question and p (or q) is one of the two prime factors of the modulus, n. The adversary is therefore facing the computationally intractable problem of factorising the large modulus, n, into its two prime factors (Yan 2000). Alternatively, the Jacobi symbol $\left(\frac{a}{n}\right)$ can be evaluated, which will give a correct answer with a probability of 0.5.

Simply combining RSA with the QRP does not provide any shielding at all against the adaptive chosen-ciphertext attack. The attacker can simply decrypt different ciphertexts that use the same QRP numbers. By examining the result, they will be able

to tell which ones mark the 'intended message' and which ones do not. Based on this new knowledge, the intercepted ciphertext can be decrypted. Consequently, the system must be re-designed in a way such that prohibits any alteration of the ciphertext and at the same time preserves the randomness component that has been introduced to it (namely, the QRP numbers).

Facing The Attack

All the possible ways mentioned above for attacking the proposed scheme involved some modification of the ciphertext. The system would therefore be sufficiently secure if it could prevent any alterations of the ciphertext, effectively preventing an adaptive chosen-ciphertext attack from taking place.

A hash function would help in solving the problem; the extended algorithm hashes entirely or partially the ciphertext and then encrypts the hash value using the sender's private key (effectively creating a digital signature). In this way, the receiver can easily verify the integrity of the ciphertext, as well as the sender. It should be noted that this is feasible because RSA can also serve as a digital signature scheme by swapping the exponents e and d during the operations of encryption and decryption. Hence, despite the fact that in Section 4.2 it was stated that any deterministic public-key cryptosystem may be used, only the ones that can also function as digital signature schemes can lead to secure end systems.

The operation of the decryption oracle should therefore be modified, so as to reject any invalid ciphertexts and produce no output. That is, the newly computed hash value must match the one already stored in it as a digital signature. Furthermore, the ciphertexts that are submitted to it do not necessarily have to have been encrypted with the encryption device, as long as they are valid. Hence, point number 4 in the attack setup above needs to be restated to cover the change in the behaviour of the decryption oracle:

- Furthermore, they have access to a decryption oracle, which will decrypt any

ciphertext submitted to it. However, should the integrity checks fail, the oracle must not respond.

It is worth mentioning that since the attacker may have a set of decrypted messages in their hands, they only need to distinguish the intended one from the rest; effectively, their attack focuses on determining which of the QRP numbers are squares and which are pseudosquares. In order to decide which parts of the ciphertext should be included in the hashing process, the following cases are identified, as well as a way for attacking the system in each one of them:

1. Either the QRP numbers that mark intended message (RSA) blocks, or the ones that mark opaque blocks, are hashed.

   – Changing one QRP number at a time before decrypting the ciphertext will either produce some plaintext or not, thus giving away which ones mark intended message blocks, by either producing some plaintext or not.

2. All the QRP numbers are hashed.

   – The adversary can still modify the RSA blocks and get the required information, based on which blocks got decrypted. For instance, each RSA block can be replaced (one at a time) with a given one whose decryption is known in advance. This will reveal what the associated QRP number is.

3. Either the intended message RSA blocks, or the opaque blocks, or all of them are hashed.

   – The QRP numbers can be modified, one at a time, and by examining the result, one can deduce what each QRP number is. In particular, if the decrypted text is lacking one block, then the RSA block following the modified QRP number belongs to the intended message.

The attacks mentioned above involved dealing with either the QRP numbers, or with the RSA blocks. However, there are a few other cases where combinations of the two can be used. They will now be presented, together with ways of attacking them.

4. The QRP numbers that mark intended message blocks are hashed, together with either the intended message blocks, or the opaque blocks, or both.

   – Changing one QRP number at a time before decrypting the ciphertext will either produce some plaintext or not, thus giving away which ones mark intended message blocks, by either producing or not some plaintext.

5. The QRP numbers that mark opaque blocks are hashed, together with either the intended message blocks, or the opaque blocks, or both.

   – The attack mentioned in the previous point can be performed.

6. The QRP numbers together with either the intended message blocks, or with the opaque blocks are hashed.

   – The RSA blocks can still get modified and the adversary will get the required information, based on which blocks got decrypted.

Hence, it has been shown that, apart from the case where the whole of the ciphertext gets hashed, all the remaining combinations (the cases mentioned above) are insecure. For an even higher level of security, the intended message can be included as an input to the hashing function. In this way, both the integrity and authenticity of the intended message can be verified at the receiving end.

4.3.4 Keys

As mentioned in Section 4.2.3, the public and private keys each essentially consists of three integers. The former comprises the triad (n, y, e) and the latter comprises the triad (p, q, d), respectively. The public and private key files, denoted by the extensions `.pub` and `.prv` respectively, contain these integers in the order described above. The operation of the system is based on the assumption that one can retrieve someone else's original public key securely. Namely, the existence of a secure key distribution protocol is assumed.

4.3.5 Encryption

For the encryption phase, both the intended plaintext blocks and the opaque blocks are encrypted using the RSA scheme and at the same time they are blended together in the ciphertext, using the QRP for distinguishing between the two, as mentioned in Section 4.2.

As can be seen in Figure 4.1, the encryption operation requires the generation of a random number, r, that determines whether an intended plaintext block $\mathrm{IPB}_1, \ldots, \mathrm{IPB}_i$, or an opaque block $\mathrm{NPB}_1, \ldots, \mathrm{NPB}_j$ will be encrypted. This process is repeated until all the intended plaintext blocks have been consumed (alternatively, it could be adjusted to consume all of both the intended plaintext and the opaque blocks), so as to produce $\mathrm{RSAN}_1, \ldots, \mathrm{RSAN}_u$, which effectively are RSA ciphertext. As soon as a RSAN block has been produced, it is appended to a randomly generated QRP number, QRPN. The latter will aid in determining whether the following-up block (RSAN) is part of the intended message or not. The sequence of alternating QRPN and RSAN blocks (essentially the so far produced ciphertext), together with all the intended message blocks are hashed. The hash value is appended to the already produced ciphertext in the form of a digital signature, thus forming the complete ciphertext.

4.3.6 Decryption

As depicted in Figure 4.2, the stored hash value is extracted from the digital signature and the remaining ciphertext is further processed. Each QRP-generated number (QRPN) is evaluated and, depending on the result, the next RSAN block is either decrypted and stored (thus retrieving the intended plaintext), or skipped. The retrieved intended plaintext, together with the received ciphertext (excluding the block carrying the digital signature) are hashed and a new hash value is obtained which is then compared to the extracted one. In case of a mismatch, the retrieved plaintext is rejected as the authenticity of the message has been compromised, otherwise it is deemed valid and authentic.

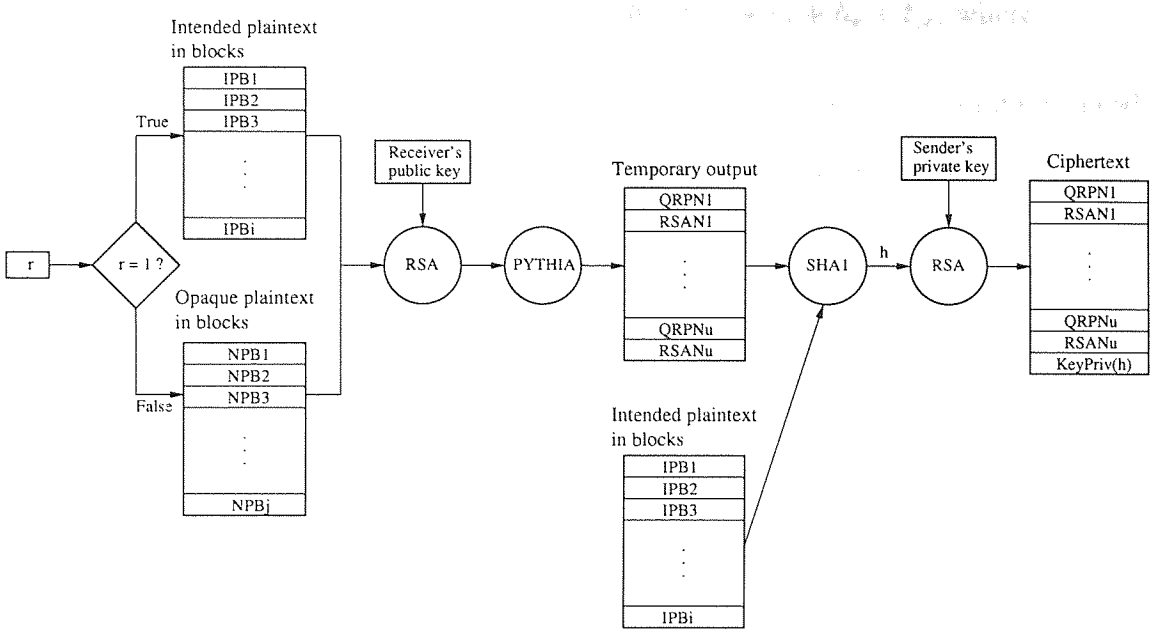


Figure 4.1: Pythia encryption.

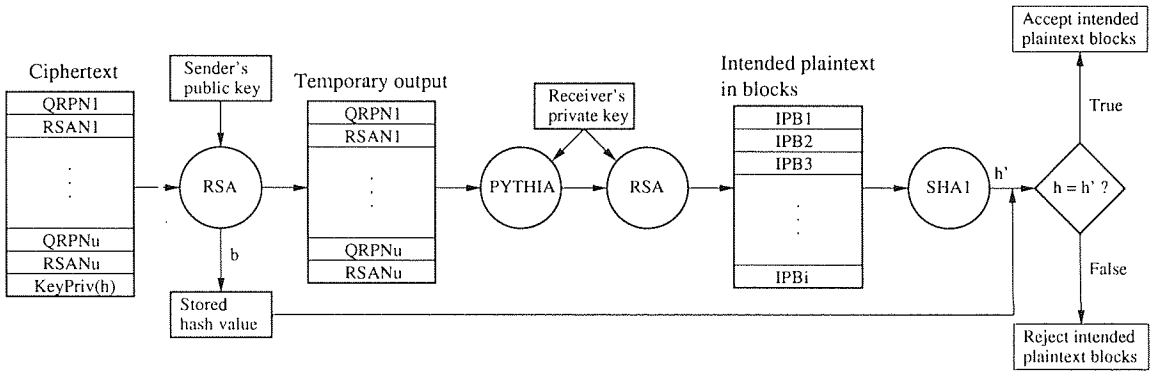


Figure 4.2: Pythia decryption.

4.3.7 Expected Performance

The following estimates are based on the assumption that the intended plaintext has a size of b blocks and that the size of the opaque blocks to be included is an integer multiple of b, namely kb, where $k \in \mathbb{N}^*$ (for $k = 0$, the system is effectively plain RSA).

4.3.7.1 Execution Time

Regarding the execution time, the estimation is relatively simple. If RSA encryption with a particular key (for one block) requires time t_e, then the proposed system is

expected to have an execution time of $T_e = (kb + b)t_e + c_e + h_e + t_{se}$, where:

- c_e is the extra time required for the generation of the QRP numbers, needed for distinguishing the intended message from the opaque blocks, as mentioned in Section 4.2.3.

- h_e is the time required to hash the intended plaintext blocks, together with the ciphertext. This time is usually short and should not have a significant effect on the overall end system's performance.

- t_{se} is the time required to encrypt the hash value and create the digital signature, that will be appended at the end of the ciphertext, as derived from the design, in Section 4.3.5.

Similarly, if RSA decryption with a particular key for one block requires time t_d, then the system is expected to have an execution time of $T_d = bt_d + c_d + h_d + t_{sd}$, where:

- c_d is the extra time required for the evaluation of *all* the QRP numbers, that classify the next RSA block (Section 4.3.6).

- h_d is the time required to compute the new hash value. It should be equal to h_e, if the ciphertext has not been tampered with.

- t_{sd} is the time required to decrypt the digital signature and extract the stored hash value. If the the two RSA exponents, e and d are of approximately the same size, it follows that t_{se} will be very close to t_{sd}.

It should be noted that not all ciphertext blocks are decrypted, as mentioned in Section 4.3.6.

As an overall comment, both c_e and c_d are random, due to the probabilistic nature of the scheme, and are proportional to the length of the QRP numbers generated, which varies; the upper and lower limits of these numbers are defined by the largest and the smallest number in $\mathbb{Z}/n\mathbb{Z}$, respectively.

From what was stated in this section, it follows that for encrypting a given plaintext, for a given opacity ratio λ, the proposed system is expected to take slightly longer than $k = \lambda + 1$ times as long as plain RSA (using the same key). Of course, this only holds if the inserted opaque blocks are an exact multiple of the plaintext size. On the other hand, decrypting the ciphertext is expected to take slightly longer than RSA, due to the overhead regarding the QRP numbers, as well as the digital signature.

4.3.7.2 Ciphertext Size

The proposed implementation takes advantage of RSA's ability to operate on blocks of ciphertext, thus increasing its execution speed. The QRP numbers' length may vary from $n - 1$ to as little as an 1-digit figure. Hence, it is reasonable to assume that their average length will be $|n|/2$, provided that the sample is large enough.

Assuming a modulus n of $|n|$ bits in length, the system produces

$$k\frac{|n|}{2} + k|n| \tag{4.1}$$

bits of ciphertext for every $|n| - 1$ bits of intended plaintext, for a opacity ratio λ, where $k = \lambda + 1$. The information rate then becomes:

$$\frac{|n| - 1}{k\frac{|n|}{2} + k|n|} = \frac{2(|n| - 1)}{3k|n|} = \frac{2}{3k} - \frac{2}{3k|n|} \tag{4.2}$$

which shows that the information rate of the proposed system increases for larger keys, although it is primarily inversely proportional to k.

4.4 Experimental Results

4.4.1 Methodology for Acquiring Results

The cryptosystem was implemented in C++ and run on a PC with an AMD Athlon 64 3000+ processor and 1 GB of RAM. The keys used were 140 to 200 decimal digits in length, in increments of 20 digits. For each key, three different ratios (λ) were used. For each of the 12 possible combinations, 100 runs of the program were performed.

It should be noted that although the same plaintext was used in each case (208 bytes), the size of the plaintext that actually got encrypted is not exactly the same in size for each key. This is due to the variation of block size as the key length increases. Hence, in some cases, the last block had to be padded out with more blank characters before it can be encrypted. This may affect the ciphertext size and the execution speed slightly, but not the encryption and decryption rates. Furthermore, the size of the inserted opaque blocks is an integer multiple of the plaintext size.

For the generation of the digital signature the SHA hash function (Section 3.7.2) has been chosen, since it produces a 160-bit hash value. Moreover, as the digital signature scheme will only be used for creating an end system that offers modest security, the use of RSA as a digital signature scheme will sufficiently fulfill these requirements.

For practical reasons, program execution was not done through the Graphical User Interface. A script in the UNIX `bash` shell scripting language was written instead. The execution times, as well as the size of the ciphertext were stored in files for later processing. Finally, the two RSA exponents, e and d, were chosen to be of approximately the same size, for reasons of increased security, as claimed by Yan (1999). The RSA implementations did not use the Chinese Remainder Theorem in their decryption operation, although it this would speed up decryption speed by almost a factor of 4 (Section 3.5.1). The main reason for doing so was the attempt to create a generic as possible implementation of RSA. Besides, a performance baseline would be achieved even without this optimisation and the effect of this optimisation could be estimated in a 'post hoc' manner anyway.

In order to get a relative measure of the proposed system's performance, both its execution speed and the ciphertext size have been compared to implementations of the Rabin encryption scheme, the Goldwasser–Micali cryptosystem and RSA, using the same library for multiple precision integer representation.

4.4.2 Results on Ciphertext Size

Regarding the ciphertext size, Figure 4.3 shows the size of the ciphertext for each cryptosystem with error bars marked at ± 2 standard deviations. The results of the Goldwasser–Micali scheme are presented on a separate graph, due to their larger scale.

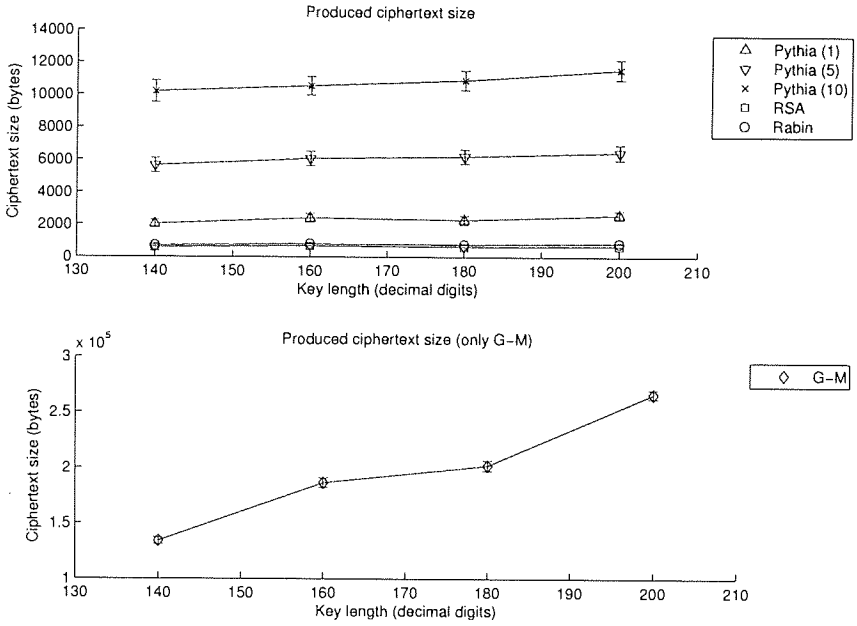


Figure 4.3: Ciphertext sizes for various key lengths.

In addition, Figure 4.4 shows the trend in the proposed system's information rate, for various key lengths and opacity ratios.

4.4.2.1 Evaluation of Results

In Section 4.3.7, a theoretical analysis was performed that attempted to predict the behaviour (and consequently the performance) of the proposed system. The analysis that follows will aid in determining how close to the theoretical predictions the obtained results are and will attempt to establish boundaries which will define the expected range of results for future executions of the proposed system.

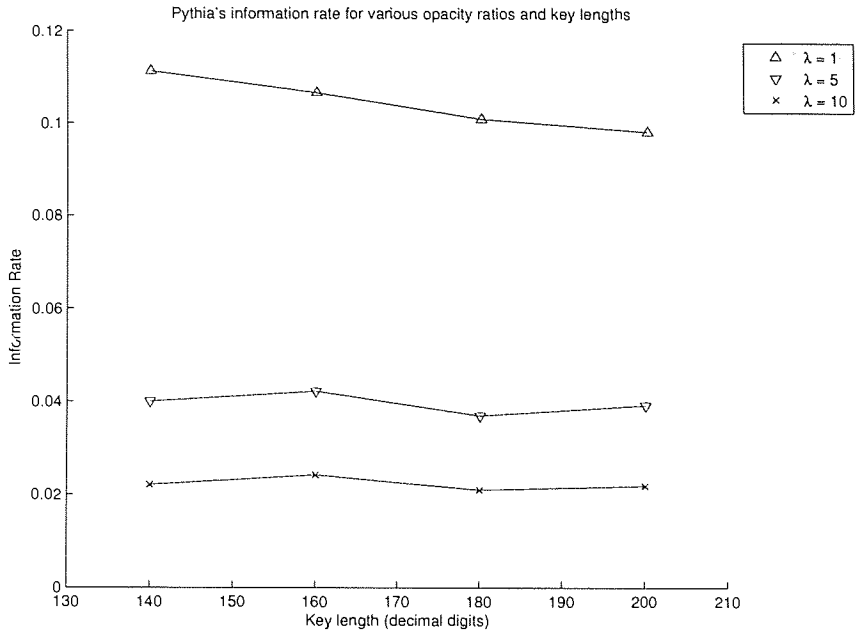


Figure 4.4: Information rate for various opacity ratios and key lengths, for Pythia.

Distribution of Results

In this section, an attempt will be made to classify the obtained results, so as to be able to perform various statistical tests and therefore draw conclusions about their quality. In Section D.1 the ciphertext distribution graphs for the proposed system are presented. The superimposed curve denotes the normal density fitted to the histogram. By examining the graphs, it seems that the data is almost normally distributed. Nevertheless, in order to establish how close the distribution of the data is to a normal distribution, a 'goodness-of-fit' test should be used. The most highly recommended tests are the Chi-square test, the Kolmogorov–Smirnov test and the Lilliefors test (Mendenhall, Beaver & Beaver 1999, SEMATECH n.d.).

Regarding the Chi-square test, the literature consulted recommended data sets of over 1000 values in order to have reliable results. Hence, it is not very suitable for the given data set (100 values).

The Kolmogorov–Smirnov test can compare a given data set against another one, whose CDF is known in advance. The Matlab `kstest` function performs the above test for a data set X for a standard normal distribution (namely, with mean 0 and variance

1).

The Lilliefors test is effectively a Kolmogorov–Smirnov test for normality when the mean and standard deviation of the hypothesised normal distribution are not known (namely, they are estimated from the sample data). It is therefore a suitable test for our case. The following hypotheses were employed for that test:

H_0: The data in X is normally distributed with a confidence level of 99%.

H_a: The data in X is not normally distributed with a confidence level of 99%.

The Lilliefors test showed that the null hypothesis could not be rejected. The results are therefore 'close enough' to a normal distribution and this property can be exploited for predicting the proposed system's performance.

Confidence Limits for the Mean

In this section an attempt will be made to determine the quality of the obtained results. In this way, the validity of Equation (4.2) can be estimated and consequently whether it can be trusted or not in the future.

An accurate approach suggested in the literature is by using the Student's t-test. Since the normality of the underlying distribution has already been established by the Lilliefors test and the obtained data does not have a known standard deviation (namely, it is calculated from the data set), Student's t-test can be used to provide the confidence limits for the mean (SEMATECH n.d.). By consulting the computed tables of the quantiles of the t-distribution, the expected mean for a 95% confidence interval ($a = 0.05$) will be

$$\overline{x} \pm \frac{z_{[1-a/2]}s}{\sqrt{n}} \Rightarrow \overline{x} \pm \frac{1.984s}{\sqrt{n}},$$

where s is the sample standard deviation and n the size of the sample. The results for different keys are presented in Table 4.1.

Opacity Ratio (λ)	Key Length	$\pm\epsilon$	$\pm\epsilon/\mu * 100\%$
1	140	25.113	1.242
	160	27.322	1.141
	180	27.549	1.219
	200	26.332	1.025
5	140	44.987	0.799
	160	44.777	0.740
	180	43.672	0.707
	200	46.790	0.727
10	140	66.238	0.650
	160	57.257	0.544
	180	59.590	0.548
	200	61.583	0.535

Table 4.1: Expected deviation of the predicted average (in bytes) with a 95% confidence level, according to the t-test, for various opacity ratios and key lengths (decimal digits).

Predictability of the Average Ciphertext

In an attempt to predict the performance of the proposed system, Equation (4.2) was constructed. By using this equation, the theoretical estimates can be calculated for each key. From the two major components of the ciphertext (namely, RSA-generated blocks and QRP-generated blocks) only the QRP-generated blocks can affect the size of the ciphertext, since their size is probabilistic. Therefore, any deviation from the expected ciphertext size average will be due to the fact that the average size of the QRP-generated ciphertext diverges from the predicted one. Such deviations have been observed for specific cases, as shown in Table D.1. A graphical presentation of the theoretical and the experimental results are shown in Figure 4.5.

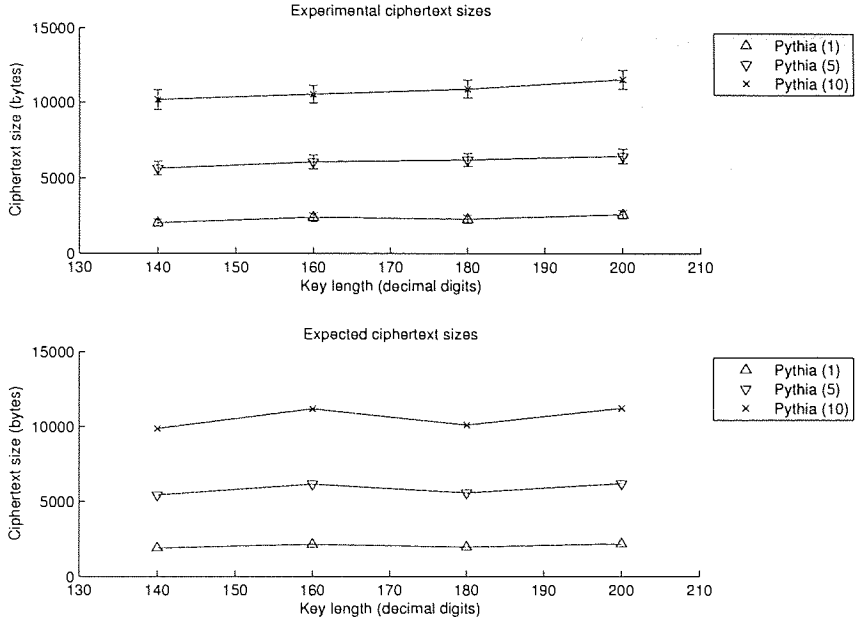


Figure 4.5: Ciphertext size for various opacity ratios and key lengths, for Pythia.

The theoretical estimate of the ciphertext size was quite close to the experimental one in the majority of the cases. The results for the opacity ratio of $\lambda = 1$ were, in general, significantly greater than expected, reaching an increase by 20% in the case of the 200-digit key.

4.4.3 Results on Execution Time

Figures 4.6 and 4.7 depict the execution times of the various cryptosystems. The results for the Goldwasser–Micali scheme are presented on a separate graph, due to their larger scale. As can be seen, the required computational power for Pythia encryption increases proportionally to the amount of the inserted opaque blocks. What is more, for a given opacity ratio, the demand for computational power increases for larger keys and this trend becomes more apparent as the quantity of the inserted opaque blocks increases.

It is worth pointing out that the encryption of the Rabin scheme surpasses the other cryptosystems by orders of magnitude (since this is not too obvious from the corresponding graph, the reader is referred to Table D.2 for the exact figures).

Surprisingly enough, the encryption speed of Pythia was higher than what is was

expected to be. As mentioned in Section 4.3.7, the required encryption time should be at least $k = \lambda + 1$ times larger than that of RSA for a given plaintext (in fact, it should be slightly larger due to the extra overhead for generating the QRP numbers and the digital signature). Nevertheless, as can be seen from Table 4.2, the encryption time was less than the expected value and this difference became more significant for larger ratios. The reason for that to occur is perhaps minor optimisations that were made by the compiler (although no optimisation options were explicitly specified at the compilation stage), or by the operating system during program execution.

Opacity Ratio (λ)	**Key Length**				**Expected**
	140	160	180	200	
1	1.9001	1.8933	1.8576	1.8572	> 2
5	5.1820	4.6998	4.9562	4.5210	> 6
10	9.3795	8.1458	8.6797	8.0241	> 11

Table 4.2: Ratios of Pythia encryption speed compared to RSA, for various keys (decimal digits).

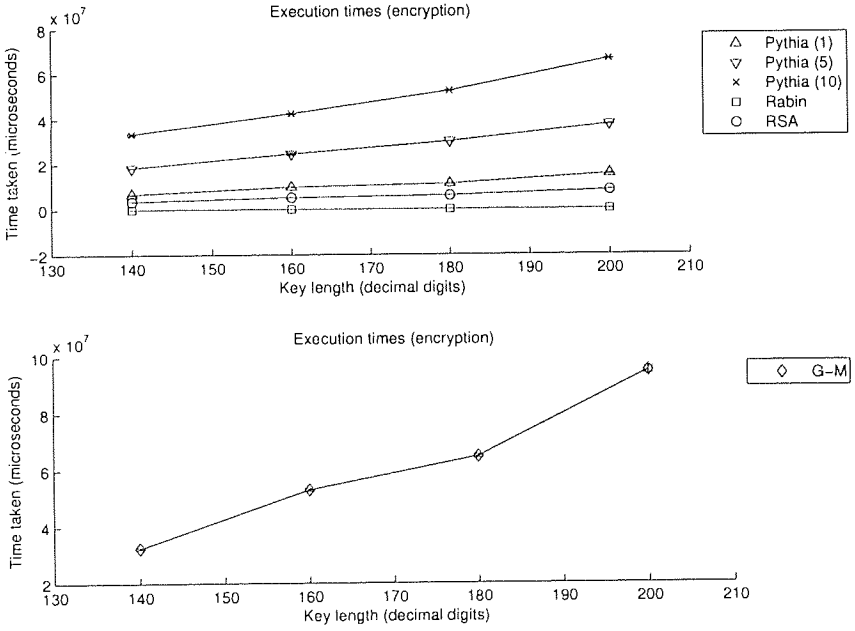


Figure 4.6: Time taken for encryption, for various cryptosystems and keys.

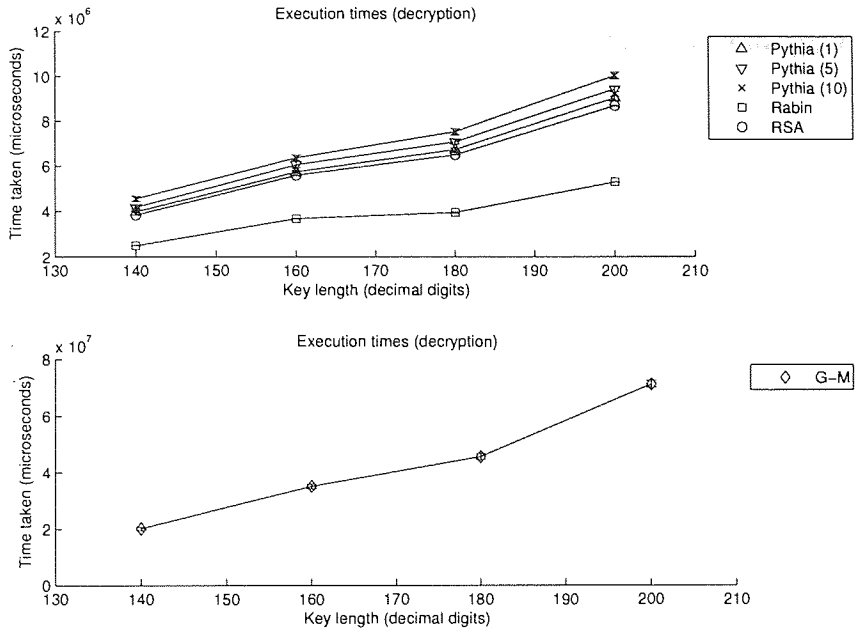


Figure 4.7: Time taken for decryption, for various cryptosystems and keys.

Finally, the encryption and decryption rates of the cryptosystems have been calculated for various key lengths and are shown in Tables D.2 and D.3. Another point worth clarifying is that the encryption rate of the proposed system actually represents the rate which *the intended plaintext* gets encrypted at. Otherwise, if the opaque blocks were included in the calculation, then the overall encryption rate would have been much higher and very close to that of plain RSA.

4.4.4 Implementation Problems

During both the development and the testing phase of the system, some problems were encountered, mainly of a technical nature. They are described in the sections below, so as to provide help for future developers using the same tools.

4.4.4.1 The BigNum Library

Although it is a very compact library that features operator overloading and provides all the necessary functions for manipulating large integers, its performance is questionable. For example, arithmetic operations on large integers were much slower compared to

two other such libraries (NTL by Victor Shoup and GNU MP Bignum). This could be either due to the programming language that *BigNum* (Diommisfois 1988) was written in, or because it was not implemented in an efficient way.

4.4.4.2 Tcl/Tk

Two main problems were faced while using the Tcl scripting language. The first one was with the `catch` keyword. More specifically, when a system call to a program is performed, `catch` will return 0 for normal program termination and 1 otherwise. However, there are cases where it is useful to know, not only whether the called program returned a non-zero exit code, but also *which* one. This was exactly the case in the developed program, which returned different non-zero exit codes to signal different error situations.

In order to overcome this problem, the called program stored its exit value to a file, which was read by the Tcl script; based on the value read, an appropriate action was taken.

During the testing phase, it was observed that system calls from a Tcl program invoking another executable file, lead to significantly slower execution of the called program. According to Welch (2000), unlike other UNIX shell `exec` commands, the Tcl `exec` does not replace the current process with the new one. Instead, the Tcl library forks first and executes the program as a child process. If the system call is performed through the Tcl interpreter (`wish`), it guarantees much slower execution.

4.4.5 Other Findings

The initial aim of this project was to combine RSA with the Quadratic Residuosity Problem (QRP), so as to face the partial information leakage problem of the original implementation of RSA. This approach, however, has a useful 'side-effect', which allows the creation of a steganographic system as follows:

The intended message together with the other messages (namely the opaque blocks)

can be transmitted unencrypted, using the QRP so as to distinguish the intended one from the rest. In steganography terms, the opaque blocks play the role of the *stego-medium* (or *cover*). Once the intended message is inserted into it, the *stego-text* is produced.

Nevertheless, this approach exhibits low security. A modification that could be made, in order to increase the degree of security, is to break down the messages into blocks. It is quite obvious that, the smaller the blocks get, the more secure the system becomes, but also the efficiency and the execution speed of the system decreases. If the block size gets down to the level of a single character, it can be regarded as a 'digital implementation' of the Girolameo Gardano Grille (Section E.3). This is also the threshold above which the system is still more efficient than the Goldwasser–Micali scheme (Section 3.6.1); if the block size gets down to bit level, it is meaningless to use the QRP (which itself can carry information for 1 bit) for marking another bit as a 1 or a 0, as this will lead to data redundancy.

Of course, the system should be designed in a way such that it prevents an adversary from tampering with the stego-text without being detected.

4.4.6 Concluding Remarks

This chapter has presented a method for combining RSA with the Quadratic Residuosity Problem, in order to face the partial information problem of 'plain-old' RSA. Some fairly obvious attacks were identified and actions that should be taken in order to face them were proposed. What is more, an extensive study of the adaptive chosen-ciphertext attack on the proposed system was made, as it generally agreed that it is the right definition of security for a public-key cryptographic scheme. Furthermore, the applicability of the technique to other public-key deterministic algorithms was examined and it was shown that only cryptosystems which can function as digital signature schemes can offer a high level of security. Comparisons with other cryptosystems sharing similar characteristics showed that it behaves within acceptable limits.

Finally, an additional application of the main idea for creating steganographic systems was discussed.

Chapter 5

Using Error-Correcting Codes for Hiding Partial Information

5.1 Introduction

This chapter presents a method for facing the information leakage problem in RSA, by exploiting the properties of error-correcting codes.

The strength of the proposed system is tested and a formula for estimating its security level is constructed. In addition, it is shown that it is a practical system featuring an execution time of almost as fast as RSA.

In what follows, the available options for implementing the proposed system are investigated and various recommendations are made. Some security issues are looked into and the specifications are adjusted accordingly. Finally, a prototype built according to the specifications is compared to the scheme presented in Chapter 4, based on the acquired experimental data.

5.2 Proposed Scheme

5.2.1 Key Principles

The plaintext undergoes some pre-processing before it gets encrypted with RSA, in a similar way to the schemes mentioned in Section 3.5.4. In particular, it is initially multiplied by an error-correcting code generator matrix. Hence, the resulting plaintext effectively consists of codewords that possess error-correcting qualities depending on the characteristics of the code used. They are then subjected to random bit flips (artificial noise), the number of which lies within the error-correcting abilities of that particular code. Finally, the 'corrupted' codewords are encrypted with RSA.

This scheme is expected to be in various ways superior to the cryptosystem proposed in Chapter 4. Firstly, for comparable cryptographic strength, it is expected to have a significantly faster execution as its message expansion takes smaller values, depending on the rate of the error-correcting code. More specifically, message expansion is inversely proportional to the code rate. Secondly, it should also offer considerably higher level of security, since the required computational effort for a brute-force attack is larger by orders of magnitude. One other advantage is that this scheme can be fully automated and thus lead to a complete system, as opposed to the considerable additional work that needs to be put into the other scheme, before it can form a totally automated system.

It should also be clarified that the proposed scheme is not a stand-alone cryptosystem, but rather a technique that focuses solely on hiding partial information when combined with RSA (or any similar deterministic public-key cryptosystem). Furthermore, despite the fact that the proposed scheme makes use of error-correcting codes, it does not form or lead to a matrix-based cryptosystem.

5.2.2 Choice of the Error-Correcting Code

The use of any error-correcting code would be appropriate for this system and the choice would affect the performance of the end system, depending on the code's characteristics. Nevertheless, one issue worth looking into is the codeword size. Using an (N, K) code with $N > K$ means that for every K bits of the source message, a codeword of N bits in length will have to be transmitted. Since this codeword will be the input for RSA encryption, ideally it should equal to the maximum possible block length that can be encrypted with RSA for a given key (this is equal to $|n| - 1$, where $|n|$ is the length in bits of the RSA modulus).

There are two approaches to achieving this requirement. The first is to carefully select the code and/or the RSA modulus length, so that the resulting codeword is $|n|-1$ bits in length. In this way, the information rate is maximised. The disadvantage, however, is that there is a dependency between the public key and the size of the parity-check matrix. Consequently, if the RSA key is changed in the future, a new parity-check matrix will be required.

The second approach is to use an error-correcting code with a smaller value of N, so that the result of the pre-processing consists of more than one codeword. The ideal condition that would give the maximum possible information rate is when $(|n|-1)$ mod $N \equiv 1$, otherwise, up to $N-1$ bits will remain unused. There are two advantages that emerge out of this approach. Firstly, the public key will be significantly smaller than in the previous case and secondly, the parity-check matrix can be re-used for larger keys. The balancing of criteria will be investigated in Section 5.4.

Moreover, it is worth emphasising that the rate of the error-correcting code will also have an effect on the overall information rate. For instance, for a given message, a rate 1/2 code will produce half as many codewords as a rate 1/4 code. Hence, in the second case, twice as many RSA blocks will be required to accommodate all the codewords, which will consequently lead to twice as much ciphertext and twice the execution time.

The error-correcting ability has a direct effect on the randomisation level of the

plaintext. A better error-correcting ability would allow more noise to be introduced, thus leading to a higher randomisation level. Consequently, the security level is expected to increase (this is further explained in Section 5.3.2). LDPC codes will be used as an instance of the wider class of error-correcting codes. It is worth pointing out that since LDPC codes are defined by a $M \times N$ sparse matrix, this can be exploited to our advantage. Under certain circumstances, given an appropriate encoding scheme, the sparse parity-check matrix may be converted into an alternative form that requires significantly less than MN bits for its representation. In this manner, the public key size will be significantly reduced.

Since the proposed scheme is aiming at the randomisation of the plaintext before it is encrypted with RSA, the use of interleaving could lead to an improved randomisation and thus to a higher security level. The purpose of interleaving is to evenly distribute along the codeword the extra bits that add error-correcting properties to it.

5.2.3 Algorithm

The algorithm for RSA-ECC[1] is as follows:

Let H be the $(M \times N)$ sparse parity-check matrix of the LDPC code able to correct t errors per codeword. Also assume the existence of a decoding algorithm suitable for handling the above code.

Let s be the original message. To enable processing, it needs to be divided into K-bit long blocks s_i, $i = 1, 2, \ldots, \beta$. It follows that each RSA block can accommodate a maximum of $w = \lfloor (|n| - 1)/N \rfloor$ codewords and therefore a total of $u = \lceil \beta/w \rceil$ RSA blocks will be required.

[1]Due to the combination of error-correcting codes with RSA, the proposed scheme has been given the name 'RSA-ECC'.

Key Generation

Let $n = pq$, where p, q are large primes, and e, d be the RSA encryption (public) and decryption (secret) key respectively, such that

$$\gcd(e, (p-1)(q-1)) = 1$$

and

$$ed \equiv 1 \bmod ((p-1)(q-1)).$$

The public key is the set (n, e, H, t) and the secret key is d.

Encryption

Using the parity-check matrix, H, create the $(K \times N)$ generator matrix for the encoder, G.

Algorithm 5.1 RSA-ECC encryption

Require: A function $f(b,t) : \{0,1\}^N \rightarrow \{0,1\}^N$, which randomly flips exactly t bits of the bit sequence b and returns the resulting bit sequence.

1: $c \leftarrow 0$ {Initialise the ciphertext}
2: **for** i from 1 to u **do**
3: $\quad \tilde{m}_i \leftarrow 0$
4: $\quad$ **for** j from 1 to w **do**
5: $\quad\quad \tilde{s}_k \leftarrow f(s_k G, t)$, where $k = (i-1) * w + j$
6: $\quad\quad$ The 'corrupted' plaintext becomes: $\tilde{m}_i \leftarrow \tilde{m}_i || \tilde{s}_k$
7: $\quad$ **end for**
8: $\quad$ Pad out $\tilde{m}_i$ with a sequence of r random bits, where $r \equiv (|n| - 1) \bmod N$
9: $\quad$ Ciphertext becomes: $c \leftarrow c || c_i$, where $c_i \equiv \tilde{m}_i^e \bmod n$
10: **end for**
11: **return** c

Thus, the ciphertext c consists of the blocks c_i produced with the algorithm above, where $i = 1, 2, \ldots, u$. It should also be noted that step number (8) above is effectively ignored if r evaluates to zero. Furthermore, if this step is implemented, it will add slightly to the security of the system, as explained in Section 5.3.3.2.

A schematic representation of RSA-ECC encryption is shown in Figure 5.1. The plaintext is encoded into codewords. It can then optionally be processed by an interleaver, as its omission will still allow the system to operate properly, although it

is expected to degrade the randomisation level (and thus affect indirectly the security level of the end system). The codewords are then subjected to random noise (up to the limit that the code is capable of correcting) and are finally encrypted with RSA, to form the final ciphertext.

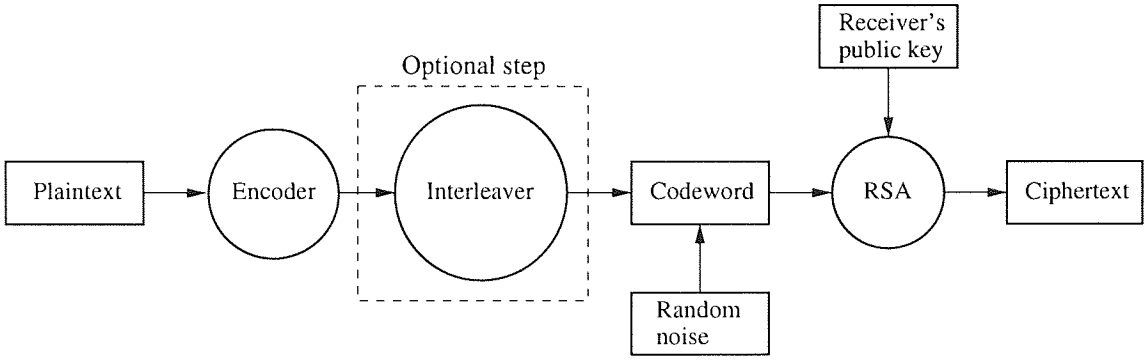


Figure 5.1: RSA-ECC encryption.

Decryption

Assuming that there are u RSA blocks to be decrypted, the procedure is as follows:

Algorithm 5.2 RSA-ECC decryption

Require: A function $g(b) : \{0,1\}^N \rightarrow (\{0,1\}^K, \mathbb{N})$ which is effectively a suitable decoding algorithm. It operates on a codeword, correcting up to t errors, and returns an ordered pair consisting of the decoded bit sequence and the number of errors corrected.

1: $s \leftarrow 0$ {Initialise the plaintext}
2: **for** i from 1 to u **do**
3: Decrypted block is: $\tilde{m}_i \leftarrow c_i^d \bmod n$
4: Strip the last r bits of $\tilde{m}_i$, where $r \equiv (|n| - 1) \bmod N$
5: Split $\tilde{m}_i$ into N-bit long blocks, $\tilde{s}_j$, where $j = 1, 2, \ldots, w$
6: **for** j from 1 to w **do**
7: $\hat{s}_j \leftarrow g(\tilde{s}_j)$
8: **if** decoding was successful and number of errors corrected $= t$ **then**
9: plaintext becomes: $s \leftarrow s||\hat{s}_j$
10: **else**
11: Abort process and report error
12: **end if**
13: **end for**
14: **end for**
15: **return** s

One point worth clarifying is that the algorithm will work equally well if function $f(b,t)$ flips fewer than t bits of the bit sequence b. In this case, step number (8) in the algorithm above should be modified accordingly. The reason for having $f(b,t)$ flip exactly t bits is because one would then expect to find and correct that many errors during the decoding process. Hence, any number of errors encountered other than t, poses questions about transmission noise as well as integrity.

Figure 5.2 is a schematic representation of the RSA-ECC decryption operation. The received ciphertext is initially decrypted with RSA. If interleaving was used in the encryption operation, the previously obtained output must be processed by a deinterleaver. The resulting codewords are input to the decoder that corrects the deliberately introduced errors and returns the original plaintext.

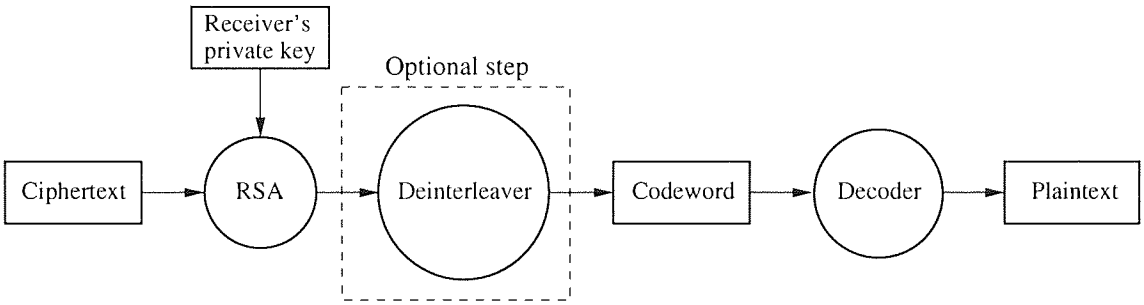


Figure 5.2: RSA-ECC decryption.

5.2.4 Applicability to Digital Signatures

The proposed scheme can also be applied to RSA digital signatures with appendix (Section 3.8.2). In brief, the original message is hashed and the hash value is processed before forming a digital signature. Once the signature has been created, it is appended to the end of the ciphertext. In this way, probabilistic signatures are created that also feature some tampering-detection mechanism.

The algorithm is very similar to the one described in Section 5.2.3. The only differences are the exchange of the two RSA exponents e and d, and also that the original message s is effectively the hash value of the plaintext.

5.3 Security Considerations

5.3.1 General

Since the proposed system results from combining RSA with error-correcting codes, there are several issues that need to be considered, as they may affect the security of the end system. Such issues are the quantity and the character of the inserted noise, as well as the ability to resist an adaptive chosen-ciphertext attack. The latter is widely accepted to be the strongest notion of security for public-key cryptosystems, and thus an extensive account will be given.

Moreover, both the operating system and the key exchange protocols are assumed to be secure and any issues relating to them will not be dealt with.

5.3.2 The Artificial Noise

There are two main approaches in inserting the artificial noise into the processed plaintext. The first one is to use a pseudo-random way of generation, or at least a way that has a pattern of some sort and the second one is to have a random way of generation that does not follow a particular pattern. The first approach has the advantage of being able to detect a possible manipulation of the ciphertext, should the received pattern of the inserted noise be different to the expected one. However, apart from requiring a way for safely conveying the seed, it is also susceptible to chosen-plaintext attacks, which will reveal the noise pattern and this information can be exploited by adversaries in future attacks. Ironically enough, in this way the technique proposed for hiding partial information is leaking some partial information itself. It is therefore recommended to use a good random number generator (with large period and good statistical properties) to avoid such mishaps.

One other important issue is that of the quantity of the inserted noise. The errors that are artificially introduced to the processed plaintext give to the end system probabilistic behaviour, which has a direct effect on its security. Consequently, the

parameters that affect this behaviour should take values that maximise security.

An attacker has access to the public key and therefore knows the maximum number of errors (t) the code can correct (nevertheless, the sender may have introduced less than t errors in the processed plaintext). Given that the parity-check matrix, H, is of dimensionality $M \times N$, the attacker can deduce the codeword size, N. Since t is relatively small compared to N, for security reasons, it will not be a reasonable choice to choose a value less than t in most cases. Even if this is not the case, trying all possible values for t is a very feasible computational task. It is therefore reasonable to assume that the attacker may have guessed correctly the number of bits that were flipped in a codeword (but not which ones). Their attack may then focus on determining either the flipped bits, or the non-flipped ones. Optimal security is therefore achieved when the flipped bits are exactly half of the total codeword bits. However, the maximum number of flipped bits will also depend on the particular error-correcting code that is used and more specifically on its ability to correct errors.

5.3.3 The Adaptive Chosen-Ciphertext Attack

5.3.3.1 Launching the Attack

Let $\mathcal{S} = \{0,1\}^K$ be the set of all possible source messages. Let $\mathcal{M} = \{0,1\}^N$ be the set of all possible encoded plaintext messages. The encoding function is $E : \{0,1\}^K \longrightarrow \{0,1\}^N$ and uses an (N, K) error-correcting code. Let $C : \{0,1\}^{k_0} \longrightarrow \{0,1\}^{k_1}$ be the RSA encryption function, with $k_0 < k_1$ and $k_1 \leq |n|$ (this precondition follows from the original RSA design, as mentioned in Section 3.5.1). It also holds that $N \leq k_0$. Finally, let $f : \{0,1\}^N \longrightarrow \{0,1\}^N$ be the function that introduces artificial noise into each codeword, by randomly flipping t bits.

An adversary launches an attack based on the following rules:

1. The keys are generated for the cryptosystem. The adversary obtains the public key, but not the private key.

2. They have access to an encryption device. The encryption device is also responsible for introducing the random artificial noise into the processed plaintext before encrypting it.

3. They are allowed to submit an arbitrary number of plaintexts for encryption and have access to the respective ciphertexts. Moreover, they can modify their submissions based on the results they get.

4. Furthermore, they have access to a decryption oracle, which will decrypt any ciphertext submitted to it. However, in case that the detected errors in each codeword are not equal to t, the ciphertext is deemed invalid and no output is produced.

5. Finally, they are allowed to submit an arbitrary number of ciphertexts for decryption and modify the ones they submit based on the results they get. It should be clarified that this set of ciphertexts excludes the one they are trying to acquire some information about.

An attack is deemed successful if either the adversary discovers which bits have been flipped, or if a constructed ciphertext is found to be valid, when submitted for decryption. The latter could be a case of an impersonation, or even an authenticity attack.

5.3.3.2 Facing the Attack

The choice of the error-correcting code shields against this attack. The security level of the end system is affected by whether the processed plaintext consists of just one codeword or several ones. In the second case, there may be $r \equiv b \bmod N$ unused bits, given an (N, K) error-correcting code and a block of b bits, where $b \geq N$ (in the case of an RSA modulus of $|n|$ bits in length, $b = |n| - 1$). These can be exploited so that they add slightly more to the security of the system, by offering an additional 2^r possible ciphertexts. It should be noted, however, that the adversary can calculate r

from the information contained in the public-key. Therefore, given that there are t bits randomly flipped per codeword and that there are $w = \lfloor b/N \rfloor$ codewords in each block, the number of possible valid combinations corresponding to a specific ciphertext is:

$$\begin{pmatrix} N \\ t \end{pmatrix}^{w} \cdot 2^{r} = \left(\frac{N!}{(N-t)!t!} \right)^{\lfloor b/N \rfloor} \cdot 2^{b \bmod N} \tag{5.1}$$

There are two main factors affecting the outcome of the above function. The first one is the codeword size, which can be chosen to be up to b. The second one is the maximum number of errors that can be detected and corrected in a codeword, which is not a fixed fraction of N but exhibits small fluctuations. Thus, some turning points may be observed for given values of N and t, similar to the ones shown on Figures 5.3 to 5.5.

The greater the result of Equation (5.1) for given values, the higher the security of the end system will be, as partial information is hidden by the randomness and the large possible data sets. The order of magnitude of these sets grows dramatically for larger values of N and t, as Equation (5.1) is exponential (the mathematical details are presented in Section F.2).

It is worth noting that error-correcting codes exhibit better error-correcting capabilities for larger codewords, usually of the order of a few tens of thousands bits. This number is significantly higher than that of an RSA modulus for offering sufficient security nowadays. Hence, provided that the plaintext is sufficiently large, a single codeword may span over several RSA blocks, thus achieving better error-correcting performance, which will have a positive effect on the security level of the end system. Referring to the notation in Equation (5.1), b will represent several RSA blocks and will therefore be a multiple of $|n| - 1$.

An adversary can make some guesses g_i, with $g_i \in \mathcal{S}$. They must then be processed to give $f(E(g_i))$, which are finally encrypted into $C(f(E(g_i)))$, since they have access to the public key. Due to the vast number of ciphertexts that can correspond to a given message, the adversary cannot know whether their guesses were right or not and

they therefore do not have any advantage in forward search types of attacks.

Additionally, due to the processing that the plaintext undergoes, followed by the encryption transformation, the end system provides security against attacks that exploit the multiplicative property of RSA. The attack can be described as follows: the adversary wishes to decrypt ciphertext c. They select a random integer $x \in (\mathbb{Z}/n\mathbb{Z})^*$ and computes $\tilde{c} \equiv cx^e \bmod n$. Decrypting $\tilde{c}$ will yield $\tilde{m} \equiv \tilde{c}^d \bmod n$ and in turn $m \equiv \tilde{m}x^{-1} \bmod n$. Since the processed plaintext, m, has to be a codeword with t erroneous bits, this will most probably not hold for $xm \bmod n$, where $x \in (\mathbb{Z}/n\mathbb{Z})^*$.

What is more, since the processed plaintext must not obey any particular format rules, it blocks attacks that exploit such features. For instance, Manger (2001) showed how to successfully perform an adaptive chosen-ciphertext attack that recovers the plaintext to PKCS #1 version 2.0, which was defined by RSA Optimal Asymmetric Encryption Padding (OAEP) and thus led to the definition of PKCS #1 version 2.1 (the reader is referred to Section B.3.1 for a detailed account on this attack).

Despite the fact that any modification of the ciphertext will most probably be detected during the decryption process, the attacker is still able to construct valid ciphertexts and submit them for decryption. This can easily be faced with the addition of a message integrity checking mechanism, which will effectively lead to a plaintext-aware encryption scheme. The simplest way to achieve this is by using a hash function and appending the result at the end of the ciphertext as a digital signature, thus exploiting the ability of RSA to function as a digital signature scheme.

Hashing just the plaintext in order to create such a digital signature may leak some partial information. Namely, if the same plaintext is processed and encrypted more than once, the respective ciphertexts will differ, but the digital signatures will be identical. A solution to this problem is including both the processed plaintext and the ciphertext as an input to the hashing process, which introduces some randomness into the digital signature and thus prevents partial information leakage. Alternatively, a probabilistic signature can be created, as described in Section 5.2.4.

5.4 Experimental Results

5.4.1 Methodology for Acquiring Results

A prototype of this cryptosystem was implemented in C/C++ and run on a PC with an AMD Athlon 64 3000+ processor and 1 GB of RAM. The software used for simulating encoding, transmission through a noisy channel and decoding of LDPC codes was developed by MacKay & Neal (1995). It is able to simulate Binary Symmetric Channel (BSC) transmissions and uses the sum-product decoding algorithm.

The keys used were 140 to 200 decimal digits in length, in increments of 20 digits and the gathered data is presented in Table 5.3. The program was run 20 times for each set of parameters, since significant variations were not observed. In particular, the standard deviation was used as a measure of the data spread (Table 5.4). Regarding the error-correcting code, there was a vast variety available to choose from. Regular Gallager codes with column weight 3, of different information rates were chosen, without the use of interleaving, so as to form a point of reference. Codes with better error-correcting ability for a given codeword length will have a positive effect on the security of the end system and vice versa. This quantity is fully predictable by evaluating Equation (5.1). One other point worth mentioning is that in some cases, the codeword length was slightly less than the maximum possible, due to code construction limitations.

5.4.2 Security Level of the System

The achieved security level of the system was measured by evaluating Equation (5.1). Two main paths were followed in the simulations to acquire the results. In the first one, the maximum possible codeword length was used. In the second approach, multiples of codewords from codes with different rates were used, so as to approximate the trend in the result during the transition from codes with small codeword lengths to larger ones.

Key Length	Code (N, K)	Error Bits	Result
140 (466 bits)	(462, 231)	23	1.37×10^{60}
160 (532 bits)	(528, 264)	27	5.96×10^{65}
180 (598 bits)	(594, 297)	30	1.91×10^{70}
200 (665 bits)	(660, 330)	33	2.08×10^{79}
140 (466 bits)	(465, 155)	60	5.01×10^{76}
160 (532 bits)	(531, 177)	66	3.89×10^{85}
180 (598 bits)	(597, 199)	77	4.61×10^{98}
200 (665 bits)	(663, 221)	85	3.44×10^{109}
140 (466 bits)	(464, 116)	67	3.32×10^{82}
160 (532 bits)	(528, 132)	82	7.92×10^{98}
180 (598 bits)	(596, 149)	93	2.21×10^{111}
200 (665 bits)	(664, 166)	112	5.94×10^{129}

Table 5.1: Number of possible combinations produced by evaluating Equation (5.1), using the maximum possible codeword length in each case, for different code rates. The maximum possible number of error bits has been obtained empirically.

Using the maximum possible codeword length, the results acquired using error-correcting codes of various rates are presented in Table 5.1. As it can be seen, the order of magnitude of the results is significantly large and tends to grow both for lower rate codes and larger codewords.

When smaller-sized codewords were used, generally speaking, there was an increase in the obtained results for smaller code rates. Despite the fact that the smallest codewords produced the lowest results for all rates, fluctuations were observed in the intermediate stages, as depicted in Figures 5.3 to 5.5. It can therefore be said that, although smaller-sized codewords are favoured, Equation (5.1) should be evaluated before a choice is made.

5.4.3 Execution Time

Regarding the execution time of the proposed system, this is mainly defined by the time required for RSA to encrypt or decrypt a block of information, as the extra overhead imposed by the processing of the error-correcting code is negligible, as can be seen from Tables 5.2 and 5.3. It should also be clarified that the execution time for the encoding process includes the required overhead for the insertion of random noise.

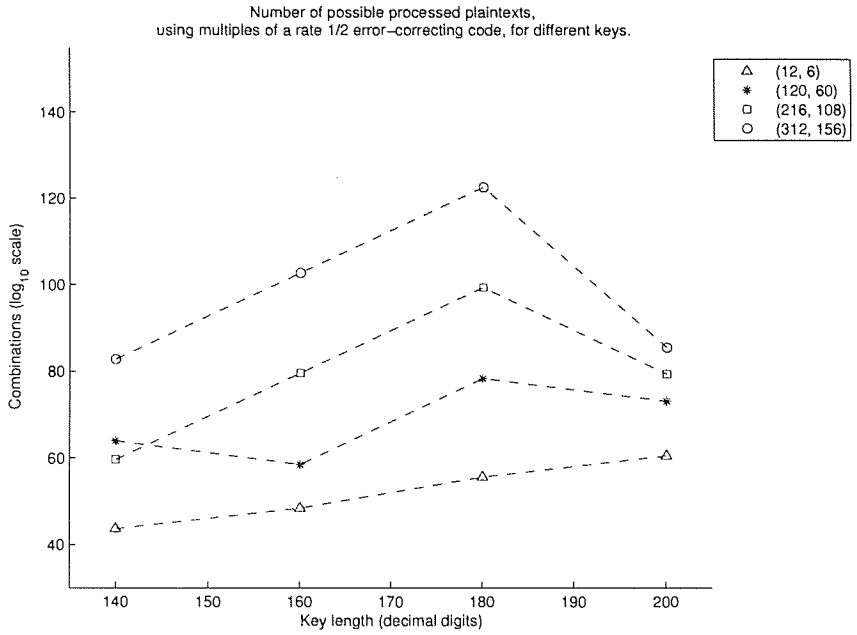


Figure 5.3: Number of possible processed plaintexts, for different keys, using a rate 1/2 error-correcting code, with various codeword sizes.

Key Length	Encryption	Decryption
140 (466 bits)	712300	766320
160 (532 bits)	1042420	1118620
180 (598 bits)	1510225	1622000
200 (665 bits)	2071375	2165100

Table 5.2: Average RSA execution times (in μs) per block, for different key lengths.

In order to compare the proposed system to an implementation of plain RSA or to other padding schemes, the code rate should be taken into consideration. For instance, if a rate 1/3 code is used, the overall execution time will be approximately 3 times longer than plain RSA, and the ciphertext produced will also be 3 times as much.

5.4.4 Comparison of RSA-ECC to Pythia

At this point it would be useful to compare the current scheme (RSA-ECC) to the one proposed in Section 4.2 (Pythia). Due to their structural differences, a special case has to be chosen so that the two systems have as many things as possible in common. One such case is where the two systems produce roughly the same ciphertext, namely Pythia using an opacity ratio of 1 and RSA-ECC using a rate 1/2 error-correcting code.

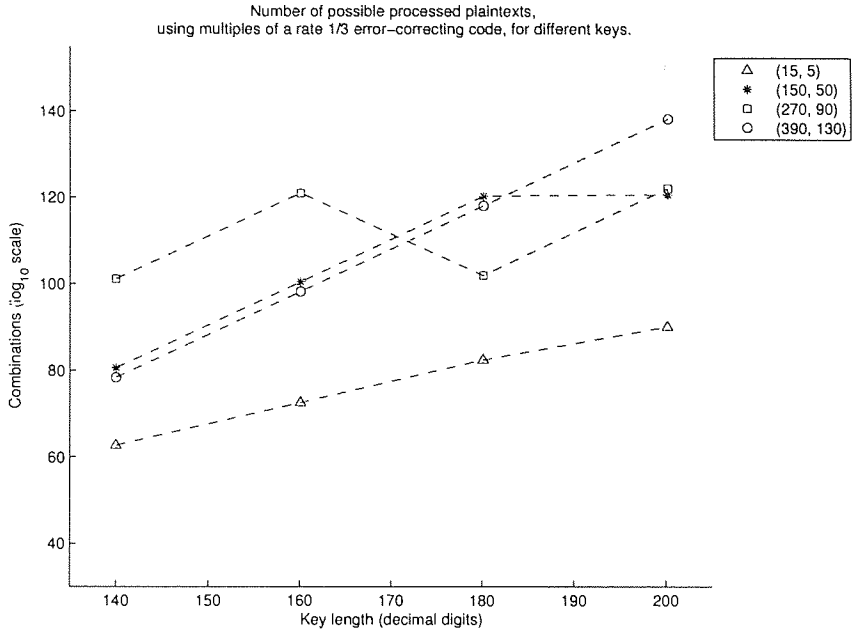


Figure 5.4: Number of possible processed plaintexts, for different keys, using a rate 1/3 error-correcting code, with various codeword sizes.

Key Length	Encoding			Decoding		
	Rate 1/2	Rate 1/3	Rate 1/4	Rate 1/2	Rate 1/3	Rate 1/4
140 (466 bits)	637	674	603	7675	6242	7915
160 (532 bits)	685	731	654	8770	7105	8953
180 (598 bits)	779	776	744	9495	10149	9746
200 (665 bits)	859	878	819	10173	11214	11655

Table 5.3: Average execution times (in μs) for encoding with an error-correcting code, per RSA block, for different key lengths and different code rates, using the maximum possible codeword length.

In both cases the ciphertext is approximately twice as much as the plaintext. As can be seen from Table 5.5, RSA-ECC has almost twice as fast encryption rate, whereas both schemes feature very similar decryption rates.

However, the corresponding security levels at these settings differ dramatically. In particular, assuming the worst-case scenario, the attacker will retrieve 2 plaintexts from the Pythia ciphertext, compared to between 10^{40} and 10^{120} for RSA-ECC, depending on the codeword size (see Table 5.1 and Figure 5.3). Thus in the Pythia case, the attacker only has to rule out one plaintext to be able to retrieve the correct plaintext, while the corresponding task is clearly impractical for RSA-ECC.

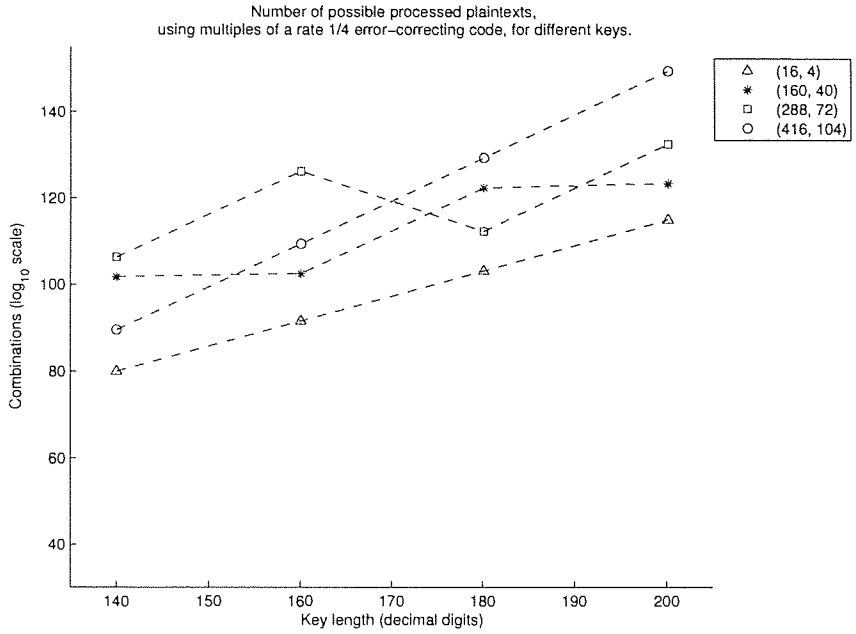


Figure 5.5: Number of possible processed plaintexts, for different keys, using a rate 1/4 error-correcting code, with various codeword sizes.

Key Length	σ **(Encoding)**			σ **(Decoding)**		
	Rate 1/2	Rate 1/3	Rate 1/4	Rate 1/2	Rate 1/3	Rate 1/4
140 (466 bits)	15.31	11.82	12.18	22.82	28.58	35.02
160 (532 bits)	13.57	9.45	11.88	28.65	21.88	24.25
180 (598 bits)	16.31	8.21	8.75	33.95	32.16	26.64
200 (665 bits)	10.89	9.67	11.37	19.97	25.40	27.43

Table 5.4: Standard deviation (in μs) of the execution times presented in Table 5.3.

5.4.5 Concluding Remarks

In this chapter, a new scheme was introduced for converting RSA into a probabilistic cryptographic scheme, using error-correcting codes. Some possible attacking scenarios were identified and ways for confronting or even preventing these attacks were proposed. Parameters affecting both the security and the performance of the end system were determined, as well as the applicability of this technique to digital signatures.

Cryptosystem	Encryption Rate				Decryption Rate			
	Key Length				Key Length			
	140	160	180	200	140	160	180	200
Pythia (1)	33.25	25.84	20.32	16.38	56.53	44.40	37.72	30.40
RSA-ECC (rate 1/2)	63.12	48.89	37.72	30.40	58.14	45.24	34.94	28.94

Table 5.5: Comparative table of encryption and decryption rates (in bytes/s) for Pythia (opacity ratio of 1) and RSA-ECC (using a rate 1/2 error-correcting code).

Chapter 6

Conclusions

The findings of the thesis are summarised in this chapter. They consist of evaluation, conclusions and observations, followed by suggestions for future research.

6.1 Aims of the Thesis

The aims of this thesis were:

1. To develop a method for transforming RSA into a probabilistic cryptosystem, where the random component (that is, the opaque blocks) is inserted after the encryption process.

   (a) Determine the effect of both the quality and the quantity of the inserted opaque blocks on the security of the system.

   (b) Investigate the applicability of the proposed method to other deterministic, public-key algorithms.

   (c) Attempt to predict the performance of the system by estimating the produced ciphertext for given parameters (key length, plaintext size, amount of inserted opaque blocks).

   (d) Develop a prototype software system for experiment and testing.

i. Ensure the system's ability to withstand an adaptive chosen-ciphertext attack.

ii. Compare to similar schemes and decide on its practicality.

iii. The proposed methods should also be extensible to take advantage of the evolution of cryptographic technology and availability of computational power.

2. To develop a method for transforming RSA into a probabilistic cryptosystem, where error-correcting codes are used for accommodating the random component.

   (a) Devise a means for estimating the security level of the system, with respect to certain parameters (key length, codeword length, error-correcting ability of the code).

   (b) Determine whether there are any parameters that could affect the choice of the error-correcting code.

   (c) Investigate the applicability of the proposed method to other deterministic, public-key algorithms.

   (d) Develop a prototype software system for experiment and testing.

      i. Ensure the system's ability to withstand an adaptive chosen-ciphertext attack.

      ii. Compare to similar schemes and decide on its practicality.

      iii. Allow for evolution in cryptographic technology and increase of cryptanalytic computational power.

3. To evaluate the usefulness and practicality of Pocklington's theorem for generating large primes.

6.2 Evaluation

6.2.1 Number-Theoretic Issues

The literature in the field of Number Theory revealed the fundamentals of public-key cryptography. The computational intractability of the integer factorisation problem and the discrete logarithm problem for large integers was made apparent, as well as the need for prime-generating techniques. There are two main approaches for achieving the latter: The first one is to randomly generate a large integer and then test it for primality, using a test that is guaranteed to have a high hit rate. The second way is to use a method that is guaranteed to generate a true prime.

For generating the primes that formed the keys of the proposed cryptosystem, the Rabin–Miller probabilistic pseudoprimality test was used, because it is easy to implement, it has a high hit rate and quite a fast execution. At the extreme case where it will fail to distinguish a prime from a pseudoprime (a Carmichael number, for instance), encrypting an example plaintext and then decrypting it to obtain the original message is a sufficient test. Should decryption fail to work properly, the keys must be rejected.

The other alternative that was considered was the use of Pocklington's theorem, which is a prime number generation algorithm and not a primality test. Due to implementation difficulties, only one of its two test conditions were implemented. Nevertheless, it proved to be a strong condition for ensuring the primality of the generated primes. However, when used for generating larger primes, its long running time left open questions regarding its practicality. This completed Aim 3 of the thesis.

6.2.2 Cryptography

The literature in the area of Cryptography was studied and the research focused on public-key cryptography, because the proposed scheme is a public-key one. In particular the RSA cryptosystem was studied in detail, to justify the need for a method that would

convert it into a probabilistic scheme. Cryptosystems sharing similar characteristics were studied in an attempt to identify common vulnerabilities.

Cryptanalytic attacks were also reviewed and emphasis was given on the adaptive chosen-ciphertext attack, guarding against which is the widely accepted definition of public-key security nowadays. The areas of digital signatures and hash functions provided information on other aspects of security, such as message integrity, authentication and non-repudiation, some of which were employed for fine-tuning the security level of the proposed systems.

The field of error-correcting codes was also looked into, since several cryptosystems have emerged out of this area. The fundamental properties of error-correcting codes were reviewed and emphasis was given on low-density parity-check codes, followed by a proposal of how they could be exploited for hiding partial information in RSA.

6.2.3 Proposed System Using the QRP

In Chapter 4, a proposed cryptosystem was outlined and described, exhibiting how the Quadratic Residuosity Problem could be combined with RSA in order to hide partial information. The main idea was to insert some random component in the ciphertext and use the QRP to distinguish between the opaque blocks and the intended message. A theoretical analysis showed that producing opaque blocks just for the parts of the ciphertext where context-related keywords exist the efficiency of the system is improved. Nevertheless, quite a lot of work needs to be put into it before this approach is fully automated. The security level of the end system was found to be proportional to the amount of the inserted opaque blocks. This completes Aim 1a.

An investigation on the applicability of this technique to other deterministic public-key cryptosystems showed that it is feasible, provided they meet certain criteria. Additionally, only the ones that can also function as digital signature schemes can reach a high level of security (Aim 1b).

Theoretical estimates of the system's performance were attempted in the case where

the number of the inserted opaque blocks is an integer multiple of the intended message. The information rate was successfully estimated with respect to the key size and the amount of the inserted opaque blocks. These estimates were later proved to be quite close to the experimental values (Aim 1c). Moreover, it was shown that the system's information ratio would improve for larger keys, although it primarily depended on the opacity ratio. One such implementation would therefore meet Aim 1(d)iii.

Once the system was theoretically proved to provide sufficient security against various attacks, an extensive study in shielding the system against the adaptive chosen-ciphertext attack took place. The attack scenario was carefully defined and solutions were provided in every case where the attacker could succeed. The prototype was built according to this design, thus meeting Aim 1(d)i.

Various tests were then performed in order to measure its performance and efficiency. The main measures found in the literature were the number of modular exponentiations, the message expansion and the execution time on a particular machine (or, equivalently, the encryption and decryption rates). All but the first were more appropriate for yielding practical results and were thus used. Generally speaking, the proposed system could not compete the implementations of plain RSA and the Rabin scheme. However, it was able to outperform the Goldwasser–Micali scheme, even for high opacity ratios (Aim 1(d)ii). Interestingly enough, the measured execution speed of the proposed system was slightly less than what was expected and no definitive reasons could be given to justify this phenomenon.

6.2.4 Proposed System Using Error-Correcting Codes

In Chapter 5 a technique was proposed where error-correcting codes were used for accommodating the random component prior to RSA encryption. The resulting cryptosystem was therefore a probabilistic version of RSA that hid partial information.

The security level of the system was found to be proportional to the number of possible combinations that the processed plaintext could be corrupted, before encryption

took place. All affecting parameters were considered and a formula was constructed for indicating the system's security level, which primarily depended on the error-correcting ability of the code. Aim 2a has therefore been met.

Any error-correcting code can safely be used for implementing the proposed technique and any evolution in the field of error-correcting codes can be utilised, so as to produce an even more secure version of the system. Furthermore, if the code can be described by a sparse parity-check matrix, this can be exploited to reduce the size of the public key (Aim 2b).

This technique was found to be applicable to any deterministic public-key cryptosystem, as well as to any deterministic digital signature scheme. Although the scheme can itself provide some shielding against the adaptive chosen-plaintext attack, using a message integrity mechanism increases the security level significantly. Consequently, this limits the application field to cryptosystems that can operate as digital signature schemes (Aim 2c).

The design followed for building the prototype provided security against the adaptive chosen-ciphertext attack, thus meeting Aim 2(d)i. Its performance depended on the extra imposed overhead for processing with the error-correcting code. The execution time of the former proved to be negligibly small compared to the time required for encrypting or decrypting a specific block. What is more, the code rate was directly related to the information rate of the end system and under certain circumstances it was identical to it.

Compared to the proposed system in Chapter 4, it proves to be superior, since for a message expansion of 2 its security level is higher by orders of magnitude. Furthermore, as the message expansion is directly related to the code rate, it is not expected to grow over 4. Although a practical comparison among matrix-based cryptosystems was not performed due to significant structural differences, the proposed system features much smaller key sizes compared to them. This concludes Aim 2(d)ii. Finally, should larger keys be needed in the near future for providing a sufficient security level,

this system does not require any additional configurations apart from a larger RSA key (Aim 2(d)iii).

6.3 Recommendations for Future Research

6.3.1 Proposed System Using the QRP

Perhaps one of the most important issues that would render the proposed system truly practical and useful is the automatic generation of the additional plaintexts that serve as opaque blocks. In the example tests performed for the purpose of this thesis, the opaque blocks were manually constructed, without really paying attention to the context of the generated opaque blocks, since it would not have any impact on the experimental results. Future research could investigate whether this process can be automated. This involves determining what a program would look for in a piece of text so as to establish its nature, the background that it belongs to and then generate several similar texts accordingly. This in turn raises other questions, such as whether the new texts will emerge from the original one by simply changing context-sensitive key-words.

When the experimental results were plotted against the expected ones (namely, the ones calculated from the formulae derived from the analysis of the system), some diversions were observed that were amplified as the key length increased. It was then conjectured that one of the possible reasons for this phenomenon was the size of the data set and/or the particular random number generator (RNG) that was used (the C `rand()` function). Repeating the experiment for a considerably larger data set, as well as with a different RNG, would help in determining the true cause of the observed fault in the predictions and perhaps aid the inclusion of correcting coefficients for obtaining more accurate predictions in the future.

6.3.2 Proposed System Using Error-Correcting Codes

For this system it was theoretically established that codes with better error-correcting performance would increase the security of the end system. In the developed prototype regular Gallager codes were used for accommodating the random component and establishing a performance baseline. This choice was justified by a good balancing between error-correcting performance and complexity. However, other codes with better error-correcting abilities exist, such as irregular Gallager codes. Future research could experiment with more codes, in order to establish the best one available. In addition, the exact effect of interleaving on the security level of the end system could be investigated.

6.3.3 Common Issues

For both systems a theoretical analysis was performed for ensuring their security against an adaptive chosen-ciphertext attack. Nevertheless, no actual attacks for breaking the systems were performed. It would be useful if future research dealt with this matter. In this way unforseen flaws may be discovered.

6.4 Conclusions

Overall, the research has achieved its aims, providing two methods for dealing with the partial information leakage problem of RSA and at the same time leading to a probabilistic cryptographic scheme which showed good performance, for the range within which the tests were performed.

Regarding the generation of prime numbers to serve as keys in asymmetric cryptosystems, it seems that the generation of a large integer and then testing it for primality is a much more efficient method than attempting to generate a true prime directly.

It was shown that is possible to transform a deterministic public-key algorithm into a probabilistic one by either inserting opaque blocks after the encryption process and

then use the Quadratic Residuosity Problem for distinguishing the intended message from the rest, or by using error-correcting codes to accommodate the random component before encryption commences. Although the latter approach was the best of the proposed ones, it was still inferior to the currently widely accepted standard that utilises Optimal Asymmetric Encryption Padding.

Last but not least, there is a rule that applies to every cryptographic scheme. That is, regardless of the amount of formal proofs which support its security, it should be made public and be subjected to extensive testing. Only in this way can any unforeseen flaws be detected, which will lead to either the creation of a revised version of the cryptosystem, or its rejection. The developed cryptosystems are no exceptions.

Bibliography

Adleman, L. M., Pomerance, C. & Rumely, R. S. (1983), 'On Distinguishing Prime Numbers from Composite Numbers', *Annals of Mathematics* **117**, 173–206.

Bauer, F. (1997), *Decrypted Secrets: Methods and Maxims of Cryptology*, Springer-Verlag.

Beauchemin, P., Brassard, G., Crépeau, C., Goutier, C. & Pomerance, C. (1988), 'The Generation of Random Numbers that are Probably Prime', *Journal of Cryptology* **1**(1), 53–64.

Bellare, M. & Rogaway, P. (1995), Optimal Asymmetric Encryption, *in* 'CRYPTO 94 – Advances in Cryptology', Vol. 950 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 92–111.

Bellare, M. & Rogaway, P. (1996), The Exact Security of Digital Signatures – How to Sign with RSA and Rabin, *in* 'Advances in Cryptology – EUROCRYPT '96', Vol. 1070 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 399–416.

Bellare, M. & Rogaway, P. (1998), 'PSS: Provably Secure Encoding Method for Digital Signatures'. Submissions to IEEE P1363a. Available on the Internet at: `http://grouper.ieee.org/groups/1363/`.

Blum, M. & Goldwasser, S. (1985), An efficient probabilistic public-key encryption scheme which hides all partial information, *in* 'CRYPTO 84 – Advances in Cryp-

tology', Vol. 196 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 289–299.

Brent, R. P. (1991), Primality Testing and Integer Factorization, *in* 'Proceedings of Australian Academy of Science, Annual General Meeting Symposium on the Role of Mathematics in Science', Canberra, pp. 14–26.

Brickell, E. F., Gordon, D. M., McCurley, K. S. & Wilson, D. B. (1992), Fast Exponentiation with Precomputation (Extended Abstract), *in* 'Advances in Cryptology – EUROCRYPT '92, Proceedings', Vol. 658 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 200–207.

Buchmann, J. A. (2000), *Introduction to Cryptography*, Springer-Verlag, New York.

Canteaut, A. & Chabaud, F. (1998), 'A New Algorithm for Finding Minimum-Weight Words in a Linear Code: Application to McEliece's Cryptosystem and to Narrow-Sense BCH Codes of Length 511', *IEEETIT: IEEE Transactions on Information Theory* **44**, 367–378.

Canteaut, A. & Sendrier, N. (1998), Cryptanalysis of the Original McEliece Cryptosystem, *in* 'Advances in Cryptology – ASIACRYPT '98', Vol. 1514 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 187–199.

Chaum, D. (1983), Blind signatures for untraceable payments, *in* 'Advances in Cryptology – Proceedings of Crypto 82', pp. 199–203.

Chaum, D. & van Antwerpen, H. (1990), Undeniable signatures, *in* 'Advances in Cryptology – CRYPTO '89', Vol. 435 of *Lecture Notes in Computer Science*, pp. 212–216.

Coppersmith, D. (1996*a*), Finding a small root of a bivariate integer equation; factoring with high bits known, *in* 'CRYPTO 96 – Advances in Cryptology', Vol. 1070 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 178–189.

Coppersmith, D. (1996*b*), Finding a small root of a univariate modular equation, *in* 'CRYPTO 96 – Advances in Cryptology', Vol. 1070 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 155–165.

Courtois, N., Finiasz, M. & Sendrier, N. (2001), How to achieve a McEliece-based digital signature scheme, *in* C. Boyd, ed., 'Asiacrypt 2001 – Advances in Cryptology', Vol. 2248 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 157–174.

Daemen, J. & Rijmen, V. (2002), *The Design of Rijndael: AES – The Advanced Encryption Standard*, Springer Verlag.

Davey, M. C. & MacKay, J. C., D. (1998), 'Low Density Parity Check Codes over GF(q)', *IEEE Communications Letters* **2**(6), 165–167.

den Boer, B. & Bosselaers, A. (1994), Collisions for the Compression Function of MD5, *in* 'EUROCRYPT '93 – Advances in Cryptology', Springer-Verlag, pp. 293–304.

Diommisfois, A. (1988), 'BigNum C++'. Available on the Internet at: `http://www.geocities.com/SiliconValley/Foothills/6573/bignum/`.

Dolev, D., Dwork, C. & Naor, M. (1991), Non-malleable cryptography, *in* '23rd Annual ACM Symposium on Theory of Computing', pp. 542–552.

Dolev, D., Dwork, C. & Naor, M. (2000), 'Non-malleable cryptography', *SIAM Journal of Computing* **30**(2), 391–437.

ElGamal, T. (1985*a*), A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, *in* 'CRYPTO 84 – Advances in Cryptology', Springer-Verlag, pp. 10–18.

ElGamal, T. (1985*b*), 'A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms', *IEEE Transactions on Information Theory* **31**, 469–472.

Gabidulin, E. M., Ourivski, A. V., Ammar, B. & Honary, B. (2003), 'Reducible Rank Codes and Their Applications to Cryptography', *IEE Transactions On Information Theory* **49**(12), 2030–2033.

Goldwasser, S. & Kilian, J. (1999), 'Primality Testing Using Elliptic Curves', *Journal of ACM* **46**(4), 450–472.

Goldwasser, S. & Micali, S. (1984), 'Probabilistic Encryption', *Journal of Computer and System Sciences* **28**(2), 270–299.

Gordon, D. M. (1993), 'Discrete Logarithms in GF(p) using the Number Field Sieve', *SIAM Journal on Discrete Mathematics* **6**(1), 124–138.

Herkommer, M. (1999), *Number Theory: A Programmer's Guide*, McGraw Hill.

Jonsson, J. & Kaliski, B. (2003), 'Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1', RFC 3447.

Kabashima, Y., Murayama, T. & Saad, D. (2000), 'Cryptographical Properties of Ising Spin Systems', *Physical Review Letters* **84**, 2030–2033.

Knuth, D. E. (1981), *Selected Papers of D. H. Lehmer, Volumes I–III*, The Charles Babbage Research Centre, St. Pierre, Canada.

Kocher, P. C. (1996), Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems, *in* N. Koblitz, ed., 'CRYPTO 96 – Advances in Cryptology', Vol. 1109 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 104–113.

Kravitz, D. W. (1993), 'Digital signature algorithm', U.S. Patent # 5,231,668.

Lehman, R. S. (1974), 'Factoring Large Integers', *Mathematics of Computation* **28**, 637–646.

Lehmann, D. J. (1982), 'On Primality Tests', *SIAM Journal on Computing* **11**(2), 374–375.

Lehmer, D. H. (1981), *Selected Papers of D. H. Lehmer, Volumes I–III*, The Charles Babbage Research Centre, St. Pierre, Canada.

Lenstra, A. K., Lenstra, Jr., H. W., Manasse, M. S. & Pollard, J. M. (1990), The Number Field Sieve, *in* 'ACM Symposium on Theory of Computing', pp. 564–572.

Li, Y. X., Deng, R. H. & Wang, X. M. (1994), 'On the equivalence of McEliece's and Niederreiter's public-key cryptosystems', *IEEE Transactions on Information Theory* **IT-40**(1), 271–273.

MacKay, J.C., D. (2003), *Information Theory, Inference and Learning Algorithms*, Cambridge University Press.

MacKay, J.C., D. & Neal, R. (1995), 'Source code for modular encoding, transmission and sum-product decoding'. Available on the Internet at: `http://www.inference.phy.cam.ac.uk/mackay/code/MNC.tar.gz` and `http://www.inference.phy.cam.ac.uk/mackay/code/code.tar.gz`.

Manger, J. (2001), A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0, *in* J. Kilian, ed., 'Crypto 2001', Vol. 2139 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 230–238.

McCurley, K. S. (1990), Odds and Ends from Cryptology and Computational Number Theory, *in* C. Pomerance, ed., 'Proceedings of Symposia in Applied Mathematics', Vol. 42, American Mathematical Society, pp. 49–74.

McEliece, R. J. (1978), A Public-Key Cryptosystem Based on Algebraic Coding Theory, Technical report, Jet Propulsion Laboratory, California Institute of Technology.

Mendenhall, W., Beaver, R. L. & Beaver, B. M. (1999), *Introduction to Probability and Statistics*, Duxbury Press, United States of America.

Menezes, A., van Oorschot, P. & Vastone, S. (1997), *Handbook of Applied Cryptography*, CRC Press, Inc.

Niederreiter, H. (1986), 'Knapsack-type Cryptosystems and Algebraic Coding Theory', *Problems of Control and Information Theory* **15**(2), 159–166.

Pomerance, C. (1996), 'A Tale of Two Sieves', *Notices of the AMS* **43**(12), 1473–1485.

Rabin (1979), Digitalized signatures and public-key functions as intractable as factorization, Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science.

Rackoff, C. & Simon, D. (1991), Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack, *in* 'Advances in Cryptology – CRYPTO '91', pp. 433–444.

Rivest, R. L. (1990), 'The MD4 Message Digest Algorithm', RFC 1186.

Rivest, R. L. (1991), The MD4 Message Digest Algorithm, *in* 'CRYPTO 90 – Advances in Cryptology', Springer-Verlag, pp. 303–311.

Rivest, R. L. (1992*a*), 'The MD4 Message Digest Algorithm', RFC 1320.

Rivest, R. L. (1992*b*), 'The MD5 Message Digest Algorithm', RFC 1321.

Rivest, R. L., Shamir, A. & Adleman, L. M. (1978), 'A method for obtaining digital signatures and public-key cryptosystems', *Communications of the ACM* **21**(2), 120–126.

Robshaw, M. (1993), Implementations of the Search for Pseudo-Collisions in MD5, Technical Report TR-103, RSA Laboratories. Version 2.0.

Robshaw, M. (1994), On Pseudo-Collisions in MD5, Technical Report TR-102, RSA Laboratories. Version 1.1.

Rosen, K. (1993), *Elementary Number Theory and its Applications*, 3rd edn, Addison-Wesley.

Schneier, B. (1991), 'One-Way Hash Functions', *Dr. Dobb's Journal* **16**(9), 148–151.

Schneier, B. (1996), *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd edn, John Wiley & Sons.

Schnorr, C. P. (1991), 'Efficient signature generation by smart cards', *Journal of Cryptology* **4**, 161–174.

SEMATECH, N. . (n.d.), 'Engineering Statistics Handbook'. Available on the Internet at: `http://www.itl.nist.gov/div898/handbook/`.

Shoup, V. (1998), Why Chosen Ciphertext Security Matters, Technical report, IBM Research Division, Zurich Research Laboratory.

Shoup, V. (2001), 'OAEP Reconsidered', Available on the Internet at: `http://eprint.iacr.org/`. This is a revision, with corrections and imporvements, of the original November 16, 2000 version.

Simmons, G. J. (1984), 'A 'Weak' Privacy Protocol Using the RSA Cryptosystem', *Cryptologia* **7**(2), 180–182.

Solovay, R. & Strassen, V. (1977), 'A Fast Monte-Carlo Test for Primality', *SIAM Journal on Computing* **6**, 84–85. Erratum in ibid, v. 7, 1978, p. 118.

Stallings, W. (1999), *Cryptography and Network Security: Principles and Practice*, 2nd edn, Prentice Hall, USA.

Stinson, D. R. (1995), *Cryptography: Theory and Practice*, CRC Press LLC, USA.

Vanstone, S. A. & Zuccherato, R. J. (1995), 'Short RSA keys and their generation', *Journal of Cryptology* **8**, 101–114.

Waidner, M. & Pfitzmann, B. (1990), The dining cryptographers in the disco: Unconditional sender and recipient untraceability with computationally secure serviceability, *in* 'Advances in Cryptology – EUROCRYPT '89', number 434 *in* 'Lecture Notes in Computer Science', p. 690.

Wang, X. & Yu, H. (2005), How to Break MD5 and Other Hash Functions, *in* R. Cramer, ed., 'Advances in Cryptology – EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings', Vol. 3494 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Germany, pp. 19–35. ISBN: 3-540-25910-4.

Wang, X., Yin, Y. L. & Yu, H. (2005), Finding Collisions in the Full SHA-1, *in* 'Advances in Cryptology – CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings', Vol. 3621 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 1–16.

Welch, B. B. (2000), *Practical Programming in Tcl and Tk*, 3rd edn, Prentice-Hall, Upper Saddle River, New Jersey, 07458, USA.

Wiener, M. J. (1990), 'Cryptanalysis of Short RSA Secret Exponents', *IEEE Transactions on Information Theory* **36**(3), 553–558.

Wu, C. K. & Wang, X. M. (1993), 'Determination of the True Value of the Euler Totient Function in the RSA Cryptosystem from a Set of Prossibilities', *Electronic Letters* **29**(1), 84–85.

Yan, S. Y. (1999), A New Cryptographic Scheme based on the kth Power Residuosity Problem, *in* '15th British Colloquium for Theoretical Computer Science (BCTCS 15)', Keele University, pp. 14–16.

Yan, S. Y. (2000), *Number Theory for Computing*, Springer-Verlag.

Appendix A

Number Theory

A.1 Groups and Generators

The order of a finite group $\mathcal{G}$, denoted by $|\mathcal{G}|$ (or by $\#(\mathcal{G})$), is the number of elements in $\mathcal{G}$. Let $a \in \mathbb{Z}_n^*$. The *order* of a is the least positive integer k, such that $a^k \equiv 1 \pmod{n}$.

Let $a \in \mathbb{Z}_n^*$. If the order of a is $\phi(n)$, then a is a *generator* or a *primitive element* of $\mathbb{Z}_n^*$. If $\mathbb{Z}_n^*$ has a generator, then it is a *cyclic* group.

A.2 Theory of Congruences

A.2.1 The Chinese Remainder Theorem

A very widely known method for solving systems of linear congruences is the Chinese Remainder Theorem (CRT), which was discovered by the ancient Chinese mathematician Sun Zi (also known as Sun Tsu in the West). This theorem states that:

If $m_1, m_2, \ldots, m_n$ are pairwise relatively prime and greater than 1, and $a_1, a_2, \ldots, a_n$ are any integers, then there is a solution x to the following simultaneous congruences:

$$\left.\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_n \pmod{m_n} \end{aligned}\right\} \tag{A.1}$$

If x and x' are two solutions, then $x \equiv x' \pmod{M}$, where $M = m_1 m_2 \cdots m_n$.

If the system of linear congruences (A.1) is soluble, then its solution can be described as follows:

$$x \equiv \sum_{i=1}^{n} a_i M_i M_i' \pmod{m},$$

where

$$\begin{aligned} m &= m_1 m_2 \cdots m_n, \\ M_i &= m/m_i, \\ M_i' &\equiv M_i^{-1} \pmod{m_i} \end{aligned}$$

A.2.2 Legendre and Jacobi Symbols

A quadratic congruence is one of the form:

$$x^2 \equiv a \pmod{n}$$

where $\gcd(a, n) = 1$. If this is soluble, then a is called a *quadratic residue modulo* n, otherwise it is called a *quadratic non-residue modulo* n.

A.2.2.1 The Legendre Symbol

Let p be an odd prime and a an integer. Supposing that $\gcd(a, p) = 1$, then the Legendre symbol $\left(\frac{a}{p}\right)$ is defined by

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & \text{if } a \in Q_p \ (a \text{ is a quadratic residue modulo } p) \\ -1, & \text{if } a \in \overline{Q}_p \ (a \text{ is a quadratic non-residue modulo } p) \\ 0, & \text{if } a \mid n \ (a \text{ divides } n) \end{cases}$$

A.2.2.2 Properties of the Legendre Symbol

Let p, q be distinct odd primes and $a, b \in \mathbb{Z}$. Then, the Legendre symbol has the following properties:

1. $\left(\frac{a}{p}\right) = a^{(p-1)/2} \pmod{p}$
2. $\left(\frac{1}{p}\right) = 1$
3. $\left(\frac{-1}{p}\right) = (-1)^{(p-1)/2}$
4. $a \equiv b \pmod{p} \Rightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$
5. $\left(\frac{a_1 a_2 \cdots a_k}{p}\right) = \left(\frac{a_1}{p}\right)\left(\frac{a_2}{p}\right)\cdots\left(\frac{a_k}{p}\right)$
6. $\left(\frac{ab^2}{p}\right) = \left(\frac{a}{p}\right)$, for $p \nmid b$
7. $\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$
8. Law of quadratic reciprocity for two distinct primes p, q

   $\left(\frac{p}{q}\right) = (-1)^{(p-1)(q-1)/4}\left(\frac{q}{p}\right)$

A.2.2.3 The Jacobi Symbol

Let a be an integer and n an odd positive integer. If $n = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$, then the Jacobi symbol is defined by

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{a_1}\left(\frac{a}{p_2}\right)^{a_2}\cdots\left(\frac{a}{p_k}\right)^{a_k}$$

where $\left(\frac{a}{p_i}\right)$ is the Legendre symbol for the odd prime p_i. Evaluating the Jacobi symbol gives the following results:

$$\left(\frac{a}{n}\right) = \begin{cases} 1, & a \text{ may be a quadratic residue modulo } n \\ -1, & \text{if } a \in \overline{Q}_p \ (a \text{ is a quadratic non-residue modulo } n). \end{cases}$$

It is worth noting that a is a quadratic residue modulo n if and only if a is a quadratic residue modulo p_i for each prime divisor p_i of n.

For instance, as can be seen from Table A.1, the following sets exist:

$a \in \mathbb{Z}_{21}^*$	1	2	4	5	8	10	11	13	16	17	19	20
$a^2 \bmod n$	1	4	16	4	1	16	16	1	4	16	4	1
$\left(\frac{a}{3}\right)$	1	-1	1	-1	-1	1	-1	1	1	-1	1	-1
$\left(\frac{a}{7}\right)$	1	1	1	-1	1	-1	1	-1	1	-1	-1	-1
$\left(\frac{a}{21}\right)$	1	-1	1	1	-1	-1	-1	-1	1	1	-1	1

Table A.1: Jacobi symbols of elements in $\mathbb{Z}_{21}^*$

$$
\begin{aligned}
Q_3 &= \{1, 4, 10, 13, 16, 19\} \\
\overline{Q}_3 &= \{2, 5, 8, 11, 17, 20\}
\end{aligned}
$$

$$
\begin{aligned}
Q_7 &= \{1, 2, 4, 8, 11, 16\} \\
\overline{Q}_7 &= \{5, 10, 13, 17, 19, 20\}
\end{aligned}
$$

$$
\begin{aligned}
J_{21} &= \{1, 4, 5, 16, 17, 20\} \\
\overline{J}_{21} &= \{2, 8, 10, 11, 13, 19\} \subset \overline{Q}_{21} \\
Q_{21} &= \{1, 4, 16\} \\
\tilde{Q}_{21} &= \{5, 17, 20\} \\
\overline{Q}_{21} &= \{2, 5, 8, 10, 11, 13, 17, 19, 20\}
\end{aligned}
$$

By definition, a is a quadratic residue modulo 21 only if it is a quadratic of both 3 and 7. The elements in $\tilde{Q}_{21}$ are called *pseudosquares*, since the Jacobi symbol $\left(\frac{a}{21}\right) = 1$, but both $\left(\frac{a}{3}\right) = \left(\frac{a}{7}\right) = -1$.

The following list regarding the number of elements in each set is also useful:

- Number of elements in $\mathbb{Z}_n^*$ that are invertible: $\phi(n)$.
- Number of elements such that $\left(\frac{a}{n}\right) = 1$: $|J_n| = \phi(n)/2$
- Number of quadratic residues (squares): $|Q_n| = \phi(n)/4$
- Number of pseudosquares: $|\tilde{Q}_n| = \phi(n)/4$
- Number of quadratic non-residues: $\overline{Q}_n = |\mathbb{Z}_n^*| - |Q_n|$

Finally, the computation of the Jacobi symbol $\left(\frac{a}{n}\right)$ with $1 \leq a \leq n$ can be performed in $\mathcal{O}\left(\log_2 a\right)^3$ bit operations.

A.2.2.4 Properties of the Jacobi Symbol

Let m, n be any positive odd composites and $a, b \in \mathbb{Z}$ with $\gcd(a, n) = \gcd(b, n) = 1$. Then, the Jacobi symbol has the following properties:

1. If $a \equiv b \pmod{n}$, then $\left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$
2. $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$
3. $\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right)$
4. $\left(\frac{a}{mn}\right)\left(\frac{a}{m}\right) = \left(\frac{a}{n}\right)$, if $\gcd(m, n) = 1$
5. $\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$
6. $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}$
7. $\left(\frac{m}{n}\right)\left(\frac{n}{m}\right) = (-1)^{(m-1)(n-1)/4}$, if $\gcd(m, n) = 1$

Appendix B

Other Aspects of Cryptography

B.1 The Diffie–Hellman Key Exchange

This protocol can be used for establishing a secret key between two communicating parties, Alice and Bob. Breaking the system requires overcoming the Diffie–Hellman problem, which relates to the discrete logarithm problem. The scheme is vulnerable to an active eavesdropper, since no communicating party has any assurances about either the identity of the source that took part in the key exchange, or the identity of the entity knowing the resulting key.

Algorithm B.1 Diffie–Hellman key exchange

1: Both parties publicly choose a prime p and a random generator g, where $2 \leq g \leq p-2$.
2: Alice chooses a random integer, a: $1 \leq a \leq p-2$ and sends $g^a \bmod p$ to Bob.
3: Bob chooses a random integer, b: $1 \leq b \leq p-2$ and sends $g^b \bmod p$ to Alice.
4: Both parties can compute $g^{ab} \bmod p \equiv (g^a)^b \bmod p \equiv (g^b)^a \bmod p$, which can be used as their secret key.

B.2 The Birthday Paradox

Birthday attacks are can successfully be applied to cryptanalysing one-way hash functions. They are based on the *birthday paradox*, which can become apparent by the

following two problems:

- What is the number of people that need to be present in a room, so that the probability of *someone having the same birthday as you* (day and month) is more than 0.5? – Answer: 235.

- What is the number of people that need to be present in a room, so that the probability of *any two of them having the same birthday* (day and month) is more than 0.5? – Answer: 23!!!

This paradox is adapted in the search for collisions when trying to attack one-way hash functions. Hence, instead of trying to find a collision for a specific plaintext (first case), it is much easier to look for a random pair of plaintexts that hash to the same value (second case).

In order to explain the birthday paradox mathematically it is easier to calculate the probability of the inverse problem, if all n birthdays are different.

For the first case the probability is easily calculated as

$$P(n) = 1 - \left(\frac{365 - 1}{365}\right)^n.$$

For the second case, since the second person cannot have the same birthday as the first (364/365), the third cannot have the same birthday as the first two (363/365) and so on, the probability is

$$\overline{P'}(n) = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{n-1}{365}\right).$$

The required probability is therefore $P'(n) = 1 - \overline{P'}(n)$.

B.2.1 MD4

This unkeyed one-way hash function was designed by Ron Rivest (Rivest 1990, Rivest 1991, Rivest 1992*a*). The initials stand for *message digest* (another name for a hash function) and it produces a 128-bit hash value. The design goals of the algorithm were (Rivest 1991):

- *Security.* It is computationally infeasible to find two messages that hash to the same value.
- *Direct security.* MD4's security is not based on any assumption, like the difficulty of factorisation.
- *Simplicity and compactness.* MD4 is as simple as possible, without large data structures, or a complicated program.
- *Favour little-endian architectures.* It is optimised for Intel microprocessors. Larger and faster computers make the necessary translations.

The algorithm of MD4 is stated in the next section.

B.2.1.1 Algorithm

Four 32-bit buffers A, B, C, and D are used in three rounds. The buffers are initialised as follows (in hexadecimal format):

$$A = \texttt{0x01234567}$$
$$B = \texttt{0x89abcdef}$$
$$C = \texttt{0xfedcba98}$$
$$D = \texttt{0x76543210}$$

Three non-linear functions are defined and used, one for each round:

$$F(X,Y,Z) = (X \wedge Y) \vee ((\neg X) \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$

The input message is broken into 16-word blocks and the last block is padded out, if necessary. In each round, the respective non-linear function is repeated 16 times, each time processing the kth sub-block (m_k) of the input message m. The four buffers are re-arranged after each step as follows: $ABCD \rightarrow DABC \rightarrow CDAB \rightarrow BCDA$. Every four steps the arrangement is reset. Their purpose is to provide input for the

operation function of each round, by setting their values to the variables a, b, c, d. For example, in the second step the values will be: $a \leftarrow D$, $b \leftarrow A$, $c \leftarrow B$ and $d \leftarrow C$. Left circular shifts of s bits are also performed. The initial values are added to the result at the end of each round.

The three operations are the following:

1. $FF(a, b, c, d, k, s)$ sets $a \leftarrow (a + F(b, c, d) + m_k) \lll s$,
   where s takes the values 3, 7, 11, 19 cyclically.

2. $GG(a, b, c, d, k, s)$ sets $a \leftarrow (a + G(b, c, d) + m_k + \texttt{0x5a827999}) \lll s$,
   with s takes the values 3, 5, 9, 13 cyclically.

3. $HH(a, b, c, d, k, s)$ sets $a \leftarrow (a + H(b, c, d) + m_k + \texttt{0x6ed9eba1}) \lll s$,
   with s takes the values 3, 9, 11, 15 cyclically.

Two out of the three rounds of the algorithm were successfully cryptanalysed by several people, using different techniques. Although none of these attacks extended to the full algorithm, MD4 was strengthened, thus resulting in MD5.

B.2.2 MD5

Despite the fact that the algorithm is more complex than MD4 (Section B.2.1), it is similar in design and also produces an 128-bit hash (Rivest 1992*b*, Schneier 1991). The improvements of MD5 over MD4 are outlined by Rivest (1992*b*) and are the following:

1. A fourth round has been added.

2. Each step now has a unique additive constant.

3. The function G in round 2 was changed from $((X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z))$ to $((X \wedge Z) \vee (Y \wedge \neg Z))$, so as to make G less symmetric.

4. Each step now adds in the result for the previous step. This promotes a faster avalanche effect[1].

[1]By *avalanche effect* we mean that the dependency of the output bits on the input bits spreads faster.

5. The order in which message sub-blocks are accessed in rounds 2 and 3 is changed, to make these patterns less alike.

6. The left circular shift amount in each round has been approximately optimised, to yield a faster avalanche effect. The four shifts used in each round are different from the ones used in other rounds.

One could therefore say that MD5 is MD4 with improved bit hashing, an extra round and better avalanche effect.

Quite a successful attack was launched by den Boer & Bosselaers (1994), which produces collisions using the compression function in MD5; furthermore, some experimental results on collisions for MD5 have been obtained (Robshaw 1993, Robshaw 1994).

A more effective attack was recently proposed by Wang & Yu (2005). The method used was a variation of the differential attack, called *modular differential*, which used modular integer subtraction, instead of using the exclusive-or as a measure of difference. By launching this attack, collisions could be found for MD5 within 15 to 60 minutes of computation time. What is more, the attack is applicable to other hash functions, such as RIPEMD, HAVAL and MD4. Regarding the last one, a collision can be found in less than a fraction of a second.

B.2.3 Other Digital Signature Schemes

Menezes et al. (1997) identify a few more types of digital signatures that are worth mentioning:

1. *One-time digital signature schemes* can be used to sign at most one message; otherwise, signatures can be forged. Once a message has been signed, a new public key needs to be generated. One of their most important advantages is that signature generation and verification are very efficient, which makes them particularly

useful for applications such as smartcards, where low computational complexity is required. The first one-time signature scheme was proposed by Rabin (1979).

2. *Blind signature schemes* were first introduced by Chaum (1983). They are two-party protocols between a sender (Alice) and a signer (Bob), unlike the digital signature schemes described above. The basic idea is as follows: Alice sends a piece of information to Bob, which he signs and returns to Alice. From this signature, Alice can compute Bob's signature on an *a priori* message m of Alice's choice. Hence, when the message and the signature are presented to Bob at some later stage (possibly by a party other than Alice), he is able to verify the signature but cannot deduce which party was originally given the signed value (namely Alice).

   The purpose of a blind signature is to prevent the signer from observing the message he signs and the signature; hence, it is later impossible to associate the signed message with the sender. Such signatures can be found in electronic cash applications, where the spender wishes to (and should!) remain anonymous to the receiver, so that spending patterns cannot be monitored.

3. *Undeniable signature schemes* are distinct from digital signatures in the sense that verification cannot be performed without the co-operation of the signer. They were initially introduced by Chaum & van Antwerpen (1990).

4. *Fail-stop signature schemes* permit an individual to prove that a signature purportedly (but not actually) signed by him is a forgery. They were initially proposed by Waidner & Pfitzmann (1990).

B.3 PKCS #1 v2.1 – RFC 3447

The Public-Key Cryptography Standards (PKCS) #1 are defined in RFC 3447 (Jonsson & Kaliski 2003), where recommendations are provided for implementations of public-

key cryptography based on the RSA algorithm. It involves encryption schemes and signature schemes with appendix. In addition, it includes data conversion primitives, as well as encryption/decryption and signature/verification primitives. Furthermore, in this RFC the information about PKCS #1 v1.5 is included, but it is only done for compatibility reasons with older versions. Its use is not recommended, as it has security flaws. In the following sections, the algorithms for implementing encryption and signature schemes are presented.

Since the lengths of the variables used in RFC 3447 are measured in octets (groups of eight bits) instead of bits, this convention will be preserved for what follows in this section.

Finally, there is support for including a label L (a character string) in both the encryption and signature processes. However, since it may be equal to the empty string, the corresponding values for SHA-1, SHA-256, SHA-384 and SHA-512 are provided.

B.3.1 RSAES-OAEP

The RSA Encryption Standard (RSAES-OAEP) is based on Optimal Asymmetric Encryption Padding (OAEP) and is the recommended way for implementing RSA nowadays. The current version is an improvement to the previous one, v2.0, against which Manger (2001) managed to launch an attack which successfully recovered the plaintext.

A note also is made about taking care not to leak partial information in more subtle ways. In particular, two such points are mentioned. The first one is about not being able to distinguish among the various error conditions that may occur during the operations of encryption and decryption. Hence, an error occurring in any intermediate stage should be reported as a single encryption/decryption error at the end of the respective operation. For that reason, the available error messages are specified. Secondly, no timing information should be leaked in any way.

The attack by Manger (2001) was based on distinguishing such error cases. In

particular, using a decryption oracle that would indicate whether a decryption failure was due to a integer-to-octets conversion or not, proved to be enough information for the attacker. The reason for a possible failure in the integer-to-octet string conversion can be a non-zero leading octet of the encoded message. By requesting the decryption of a carefully selected multiple of the target ciphertext, the range of the desired plaintext could be determined. Repeating this procedure eventually narrowed down the range to only one value, namely the plaintext m. The attack required about 1100 oracle queries for a 1024-bit RSA key and about 2200 queries for a 2048-bit key.

B.3.1.1 Algorithm

The algorithm allows a string L to be associated with the message. If this is not provided, it is set to the default value, the empty string. As in the case of plain RSA the public key is the pair (e, n) and the private one is d.

Encryption Operation

Algorithm B.2 RSA-OAEP length checking (encryption)

1: Check whether the length of L is within the input limitations of the hash function ($2^{61} - 1$ octets for SHA-1). If not, output 'label too long' and halt.
2: Check whether the message length lies within the required limit of $|M| > |n| - 2|h| - 2$, where $|M|$ is the length of the message, $|n|$ the length of the RSA modulus and $|h|$ the length of the hash value. If this does not hold, output 'message too long' and halt.

Algorithm B.3 RSA-OAEP message encoding

1: If a label L is provided, hash it into a string of length $|h|$.
2: Generate an octet string, PS, consisting of $|n| - |M| - 2|h| - 2$ zero octets (this string will be used for padding purposes, hence its size may be zero).
3: Form a data block, DB, of length $|n| - |h| - 1$ octets as:
DB $\leftarrow h$ || PS || 0x01 || M.
4: Generate a random octet string seed, s, of length $|h|$.
5: Using a Mask Generation Function (MGF), generate
dbMask $\leftarrow$ MGF(s, $|n|$, $|h| - 1$).
6: Set
maskedDB $\leftarrow$ DB $\oplus$ dbMask,
seedMask $\leftarrow$ MGF(maskedDB, $|h|$),
maskedSeed $\leftarrow$ $s \oplus$ seedMask.
7: The encoded message is: EM $\leftarrow$ 0x00 || maskedSeed || maskedDB.

Algorithm B.4 RSA-OAEP encryption

1: Convert the encoded message, EM, into an integer message representative, m.
2: Apply standard RSA encryption using modular exponentiations, to produce the ciphertext $c = m^e \bmod n$.
3: Convert the integer representative, c, into a ciphertext C of length $|n|$ octets.

Decryption Operation

Algorithm B.5 RSA-OAEP length checking (decryption)

1: Check whether the length of label L is within the input limitation of the hash function (2^{61} octets for SHA-1). If not, output 'decryption error' and halt.
2: Check whether the length of the ciphertext is $|n|$ octets. If this condition is violated, output 'decryption error' and halt.
3: **if** $|n| < |h| + 2$ **then**
4: Output 'decryption error' and halt execution.
5: **end if**

Algorithm B.6 RSA-OAEP decryption

1: Convert the ciphertext C into an integer representative, c.
2: Apply standard RSA decryption to get the integer message representative: $m = c^d \bmod n$. If the ciphertext representative is out of range, output 'decryption error' and halt.
3: Convert the message representation m into an encoded message string, EM, of length $|n|$ octets.

Algorithm B.7 RSA-OAEP message decoding

1: Compute the hash value, h, of the label L.
2: Break the encoded message EM into a single octet Y, an octet string maskedSeed of length $|h|$ and an octet string maskedDB of length $|n| - |h| - 1$ as:
EM $\leftarrow Y \mid\mid$ maskedSeed $\mid\mid$ maskedDB.
3: Set
seedMask $\leftarrow$ MGF(maskedDB, $|h|$)
$s \leftarrow$ maskedSeed $\oplus$ seedMask,
dbMask $\leftarrow$ MGF(s, $|n| - |h| - 1$),
DB $\leftarrow$ maskedDB $\oplus$ dbMask.
4: Break DB into its component parts: DB $= h' \mid\mid$ PS $\mid\mid$ 0x01 $\mid\mid M$.
5: **if** $Y \neq 0$ OR $h \neq h'$ OR the octet that separates PS from $M \neq$ 0x01 **then**
6: Output 'decryption error' and halt the decryption process.
7: **end if**
8: Output the message, M.

B.3.1.2 Performance Issues

The high performance of RSAES-OAEP is because of the low message expansion, which is in turn due to the use of the special padding scheme. In particular, the true plaintext conveyed by each block is reduced by the size of the hash value, $|h|$. Hence, the new information ratio is $\frac{(|n|-1)-|h|}{|n|}$, which usually turns out to be quite high, given that the hash value is much smaller than the modulus n.

B.3.2 RSASSA-PSS

This part of the standard describes a secure way for creating digital signatures, the RSA Signature Scheme with Appendix – Probabilistic Signature Scheme (RSASSA-PSS). It is based on the work of Bellare & Rogaway (1996) and Bellare & Rogaway (1998).

Let x be the maximal bit length of the integer representative of EM. It should be at least $8|h| + 8|s| + 9$ bits long, where $|h|$ the octet length of the hash value and $|s|$ the octet length of the seed.

B.3.2.1 Algorithm

Signature Generation Operation

The message needs to be encoded into EM, of length $\lceil(\lceil\log_2 n\rceil - 1)/8\rceil$ octets, where n is the RSA modulus.

Algorithm B.8 RSA-PSS length checking (signature generation)

1: If the length of M is greater than the input limitation for the hash function ($2^{61}-1$ octets for SHA-1), output 'message too long' and halt.
2: Hash the message M into a hash value, h, of length $|h|$.
3: **if** $|\text{EM}| < |h| + |s| + 2$ **then**
4: Output 'encoding error' and halt.
5: **end if**
6: Generate a random octet string, s, of length $|s|$, to be used as the salt. It may be the empty string of zero length.
7: Set $M' \leftarrow$ (0x)00 00 00 00 00 00 00 00 $||$ h $||$ s.
8: Let H be the hash value of M', of length $|H|$.
9: Generate an octet string PS of $|\text{EM}| - |s| - |h| - 2$ zero octets. It may be of zero length.
10: Set
DB $\leftarrow$ PS $||$ 0x01 $||$ s, of length $|\text{EM}| - |h| - 1$,
dbMask $\leftarrow$ MGF(H, $|\text{EM}| - |h| - 1$),
maskedDB $\leftarrow$ DB $\oplus$ dbMask.
11: Set the leftmost $8|\text{EM}| - x$ bits of the leftmost octet in maskedDB to zero.
12: Set EM $\leftarrow$ maskedDB $||$ H $||$ 0xbc.
13: Output EM.

It is worth noting that step number 7 in Algorithm B.8 above is not related to the vulnerability exploited in RSAES-OAEP, mentioned in Section B.3.1. Here the zeroes are used for padding purposes, before the hashing operation. The produced result serves as a seed to the Mask Generation Function.

Algorithm B.9 RSA-PSS signature generation

1: Convert the encoded message EM, to an integer representative, m.
2: Apply standard RSA decryption to get the signature integer representative: $r = m^d \bmod n$.
3: Convert the signature representative, r, into a signature R, of length $|n|$ octets.
4: Output signature R.

Algorithm B.10 RSA-PSS length checking (signature verification)

1: **if** $|R| \neq |n|$ **then**
2: Output 'invalid signature' and halt.
3: **end if**

Algorithm B.11 RSA-PSS signature verification

1: Convert the signature R to a integer signature representative, r.
2: Apply standard RSA encryption to get the integer message representative: $m = r^e \bmod n$. If m is out of range, then output 'invalid signature' and halt.
3: Convert the integer message representative, m, to an encoded message, EM. If $|\text{EM}| \neq |n| - 1$, output 'invalid signature' and halt.

Algorithm B.12 RSA-PSS message verification

The obtained decrypted message (M') is processed, so as to produce a signature. The latter is then be compared to the one attached to the encrypted message and is deemed valid if and only if the two match.

1: If $|M|$ is greater than the input limitation for the hash function ($2^{61}-1$ for SHA-1), output 'inconsistent' and halt.
2: Hash the message M into hash value h, of length $|h|$.
3: **if** ($|\text{EM}| < |h| + |s| + 2$) OR the rightmost octet of EM $\neq$ 0xbc **then**
4: Output 'inconsistent' and halt.
5: **end if**
6: Let maskedDB be the leftmost $|\text{EM}| - |h| - 1$ octets of EM and let H be the next $|h|$ octets.
7: If the leftmost $8|\text{EM}| - x$ bits of the leftmost octet in maskedDB are not all equal to zero, output 'inconsistent' and halt.
8: Set
dbMask $\leftarrow$ MGF(H, $|\text{EM}| - |h| - 1$),
DB $\leftarrow$ maskedDB $\oplus$ dbMask.
9: Set the leftmost $8|\text{EM}| - x$ bits of the leftmost octet in DB to zero.
10: If the $|\text{EM}| - |h| - |s| - 2$ leftmost octets in DB are not zero, or if the octet at position $|\text{EM}| - |h| - |s| - 1$ does not have the value 0x01, output 'inconsistent' and halt. It should be noted that enumeration starts from 1, from left to right.
11: Let
The salt s be the last $|s|$ octets of DB.
$M' \leftarrow$ (0x)00 00 00 00 00 00 00 00 || h || s (Namely, M' is an octet string of length $8 + |h| + |s|$, with eight initial zero octets).
H' be the hash value of the recovered message (M'), of length $|h|$.
12: **if** $H' = H$ **then**
13: Output 'consistent'.
14: **else**
15: Output 'inconsistent'.
16: **end if**

Final Step

If the result of the message verification procedure above is 'consistent', output 'valid signature', else output 'invalid signature'.

Appendix C

The Pythia System

C.1 Naming of the Project

The end system in Chapter 4 has been named after Pythia, a respected woman in the oracle at Delphi in Ancient Greece. People who went to her looking for an oracle, paid a fee and sacrificed an animal, usually a goat. Pythia then entered the temple, burned daphne and barley flour, and incensed the sacrificed animal. She would then enter a private place in the temple, drink water from Cassiotida spring, chew daphne leaves and sit (or simply touch) a holy tripod that was located therein. By breathing the *spirit* that came out of a small cranny, Pythia would get in a state of ecstacy and then give the oracle. These oracles were the power of the Delphi divination. They could determine the way cities, kings and individuals were acting in order to face serious issues. What is more, these oracles were well-known for their ambiguity. This was deliberate, so as to always prove the Gods right, in case of a misinterpretation.

In the developed cryptosystem, the 'ambiguity' is formed by the existence of two, or more, ciphertexts in the transmitted ciphertext. Only 'God' (namely, the one who has the key) can distinguish the intended ciphertext from the rest.

C.2 How to Install and Run Pythia

The program has been developed to run on UNIX/Linux systems only. Once all the files have been copied to a directory, the only changes that may be needed are:

1. The C++ compiler declared in the first line of the `makefile`.

2. The path of the Tcl/Tk interpreter, `wish` (version 8.3, or higher is required) declared in the first line of `gui_pythia.tcl`.

Typing `make install` in the command prompt will compile all the required files, in order to produce the executables. There are a few more options available, as stated in the `makefile` itself.

In order to launch the GUI, it must have its execute permission set. This can be achieved by typing `chmod 700 gui_pythia.tcl`. It can then be launched by typing `./gui_pythia.tcl`. Once launched, there is an on-line help file available, to tackle with questions that can possibly arise.

Appendix D

Additional Experimental Results

D.1 Ciphertext Size Distribution

D.1.1 Pythia

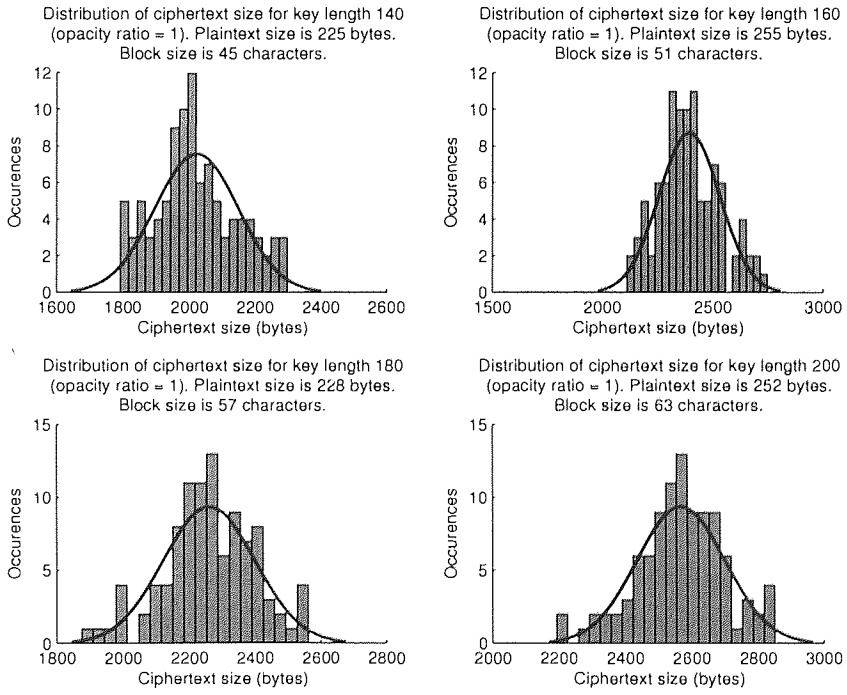


Figure D.1: Ciphertext size distribution of Pythia, for an opacity ratio of 1, for various keys.

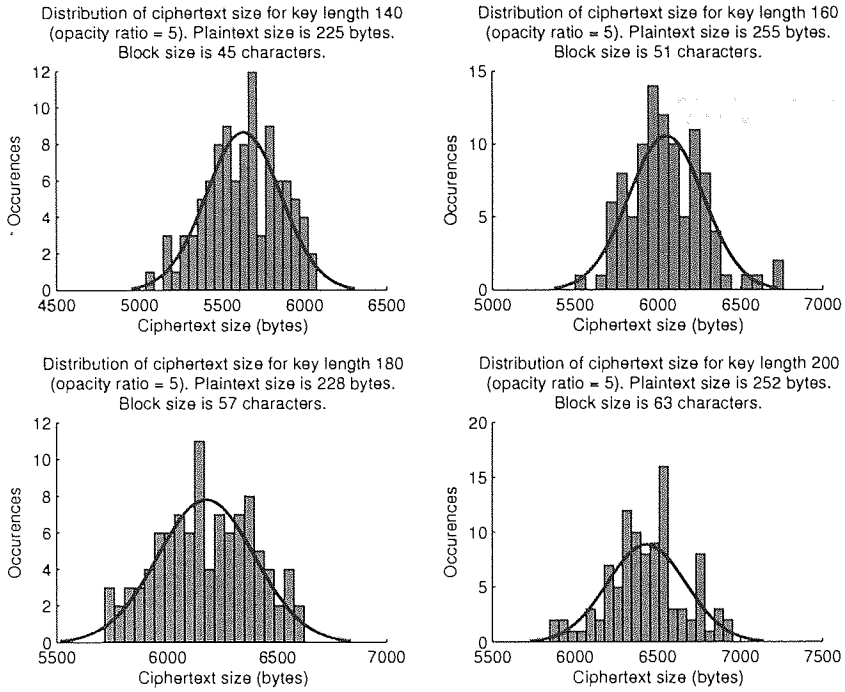


Figure D.2: Ciphertext size distribution of Pythia, for an opacity ratio of 5, for various keys.

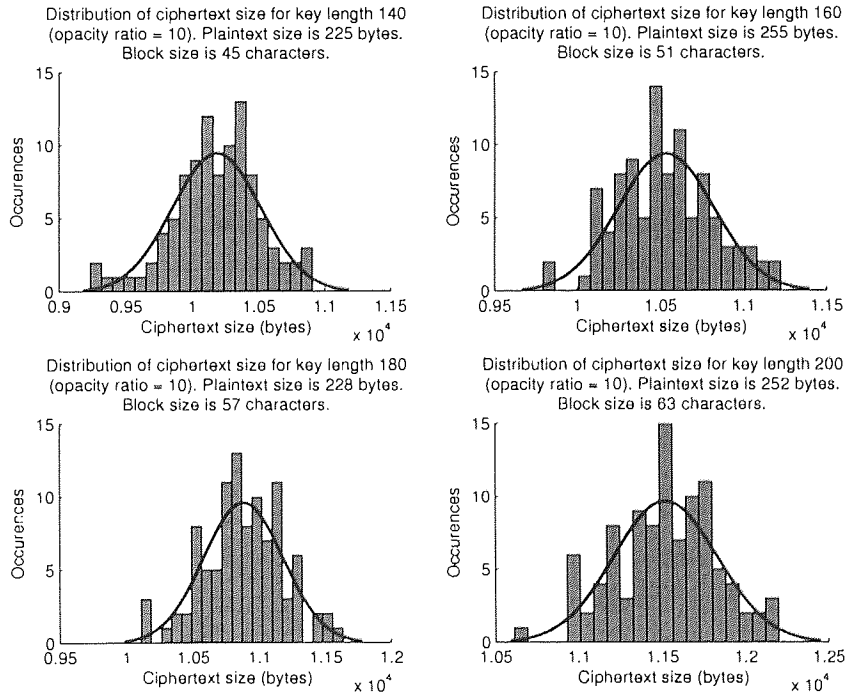


Figure D.3: Ciphertext size distribution of Pythia, for an opacity ratio of 10, for various keys.

D.1.2 Goldwasser–Micali

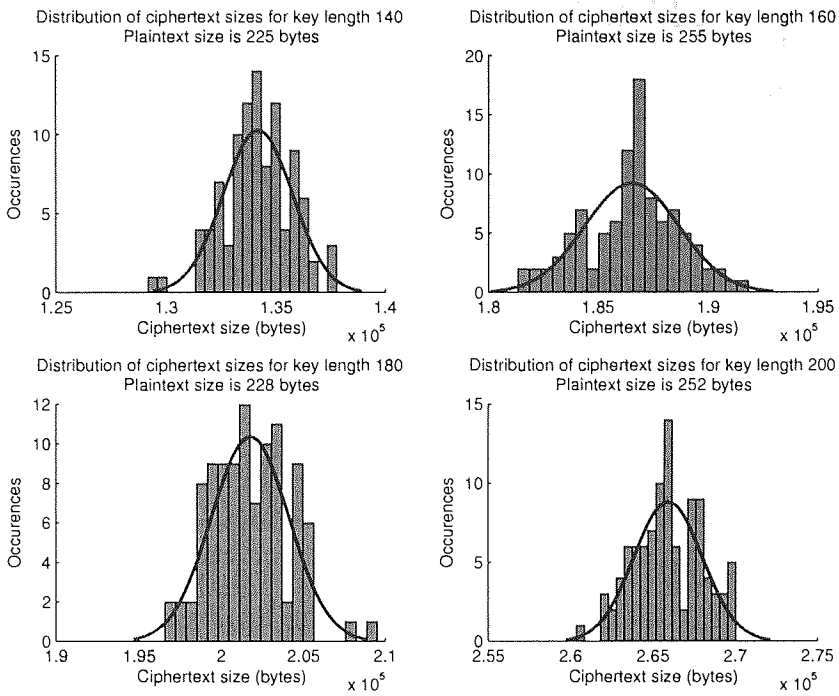


Figure D.4: Ciphertext size distribution of the Goldwasser–Micaly scheme, for various keys.

D.2 Ciphertext Size Experiments of Pythia

Opacity Ratio	Key Length	Theoretical	Experimental	Difference	
				Absolute	%
1	140	1888	2022.07	134.07	7.10
	160	2144	2393.56	249.56	11.64
	180	1963	2260.48	297.48	15.15
	200	2184	2567.88	383.88	17.58
5	140	5428	5629.46	201.46	3.71
	160	6164	6050.22	113.78	1.85
	180	5587	6177.20	590.20	10.56
	200	6216	6434.68	218.68	3.52
10	140	9853	10183.61	330.61	3.36
	160	11189	10533.04	655.96	5.86
	180	10119	10880.47	761.47	7.53
	200	11256	11521.24	265.24	2.36

Table D.1: Difference in average ciphertext size (in bytes) between the theoretical estimates and the experimental results.

D.3 Encryption and Decryption Rates

Cryptosystem	Encryption Rate			
	140	160	180	200
Pythia (1)	33.25	25.84	20.32	16.38
Pythia (5)	12.19	10.41	7.62	6.73
Pythia (10)	6.73	6.01	4.35	3.79
Rabin	49395.97	43007.52	39413.13	36519.26
RSA	63.18	48.93	37.74	30.42
G–M	6.95	4.81	3.52	2.65

Table D.2: Encryption rates (in bytes/s) for various key lengths.

Cryptosystem	Decryption Rate			
	140	160	180	200
Pythia (1)	56.53	44.40	33.92	27.96
Pythia (5)	53.81	42.13	32.28	26.80
Pythia (10)	49.29	40.09	30.35	25.17
Rabin	90.40	69.52	57.86	47.64
RSA	58.72	45.59	35.14	29.10
G–M	11.17	7.27	5.01	3.54

Table D.3: Decryption rates (in bytes/s) for various key lengths.

Appendix E

Linguistic Steganography

E.1 Open Codes

Open codes use illusions or code words. In World War I, for example, German spies used fake orders for cigars to represent various types of British warships (cruisers & destroyers). Thus, 5000 cigars needed in Portsmouth meant that 5 cruisers were in Portsmouth.

E.2 Null Ciphers

In null ciphers, the first letter of each word spells out a message. However, messages are very hard to construct and they always sound strange. The strangeness of such a message can be reduced, if the constructor has enough space and time. Hence, there are several examples of secret messages spelled out as the first letter of each chapter, or paragraph in a book.

E.3 Geometrical Codes

There also are geometrical open codes, of which the oldest one is the Cardan Grille. In the 16th century Girolameo Cardano produced the Cardano Grille, a rigid sheet

with letter or word holes cut into it at random intervals. The grille was laid over an empty page and then the secret message was written in the holes. The grille was then removed and the blank spaces were filled, so as to produce an innocent looking message. However, constructing a sensible sounding cover message is, as with null ciphers, difficult. Grille messages always sound funny, so they were often detected.

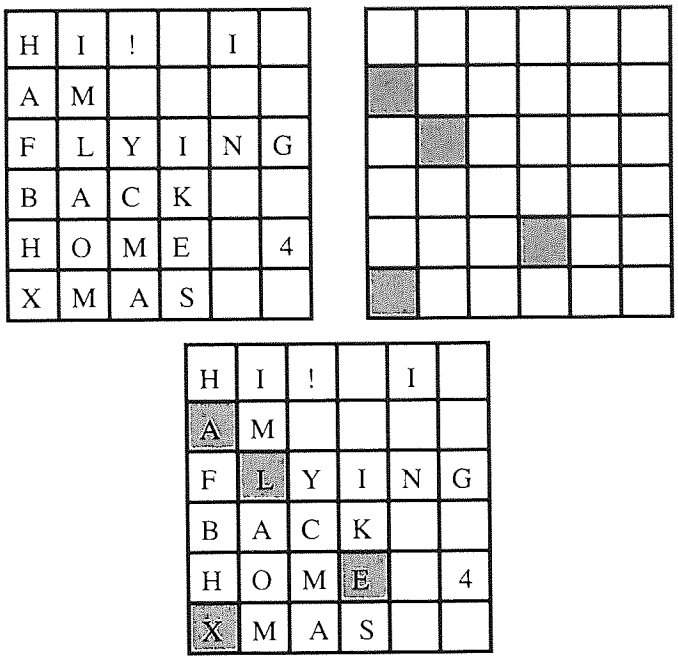


Figure E.1: The Girolameo Cardano Grille. Shaded boxes represent the grille holes, that reveal the true message.

Appendix F

Detailed Calculations

F.1 Comparing Cryptographic Strength of McEliece and RSA

The best algorithm for factorising an RSA modulus is GNFS, which has a complexity of

$$\mathcal{O}\left(\exp\left(\left(\frac{64}{9}\right)^{1/3} \cdot (\log_2 n)^{1/3} \cdot (\log_2(\log_2 n))^{2/3}\right)\right)$$

(Lenstra et al. 1990, Pomerance 1996).

The number of binary operations would therefore be

$$(\log_2 n)^2 \cdot e^{\left(\frac{64}{9}\log_2 n\right)^{1/3} \cdot (\log_2(\log_2 n))^{2/3}}$$

As claimed by Canteaut & Sendrier (1998), the required number of binary operations for breaking McEliece with its original parameters ($n = 1024$, $k = 524$, $t = 50$) is $2^{64.2}$.

We need to find an approximation of an RSA key length that would require the same number of binary operations to factorise. So, we actually need to find n (or $\log_2 n$) such that:

$$(\log_2 n)^2 \cdot e^{\left(\frac{64}{9}\log_2 n\right)^{1/3} \cdot (\log_2(\log_2 n))^{2/3}} = 2^{64.2}.$$

Taking logs on both sides gives

$$\ln(\log_2 n)^2 + \left(\frac{64}{9}\log_2 n\right)^{1/3} \cdot (\log_2(\log_2 n))^{2/3} = \ln 2^{64.2}$$

For simplicity let $\log_2 n = \beta$ (hence β is the bit length of n). We therefore have

$$\ln \beta^2 + \left(\frac{64}{9}\beta\right)^{1/3} \cdot (\log_2 \beta)^{2/3} = 44.5 \tag{F.1}$$

Equation (F.1) is monotonic and hence an approximation may be attempted by setting values to it.

Let $f(\beta) = \ln \beta^2 + \left(\frac{64}{9}\beta\right)^{1/3} \cdot (\log_2 \beta)^{2/3}$. By substitution we have:

$$f(123) = 44.4281$$

and

$$f(124) = 44.5775$$

It can therefore be said that breaking the McEliece cryptosystem with its original parameters is computationally equivalent to the factorisation of a 124-bit (412 digits) long RSA modulus, using the GNFS.

F.2 Estimating the Behaviour of Equation (5.1)

We need to know the behaviour of the expression below when N tends to infinity. It should be noted that $w > 0$ and $r \geq 0$. In a practical code, we would set t to be a small fraction of N (our experiments have used values in the range $[0.05, 0.15]$) depending on the code rate. Since t is a fraction of N, it may be replaced by αN, where $\alpha < 1$ is the fraction.

The initial expression is:

$$\left(\frac{N!}{(N-t)!t!}\right)^w \cdot 2^r$$

By applying Stirling's formula, the above expression becomes:

$$\left(\frac{\sqrt{2\pi N} \cdot (\frac{N}{e})^N}{\sqrt{2\pi(N-t)} \cdot \frac{(N-t)^{N-t}}{e^{N-t}} \cdot \sqrt{2\pi t} \cdot \frac{t^t}{e^t}}\right)^w \cdot 2^r =$$

$$\left(\frac{\frac{\sqrt{2\pi N}\cdot N^N}{e^N}}{\frac{\sqrt{(2\pi)^2(N-t)t}\cdot(N-t)^{N-t}\cdot t^t}{e^{N-t+t}}}\right)^w \cdot 2^r =$$

$$\left(\frac{\sqrt{2\pi N}\cdot N^N}{2\pi\cdot\sqrt{(N-t)t}\cdot(N-t)^{N-t}\cdot t^t}\right)^w \cdot 2^r \tag{F.2}$$

By substituting $t = \alpha N$, Equation (F.2) becomes:

$$\left(\frac{\sqrt{2\pi N}\cdot N^N}{2\pi\cdot\sqrt{(N-\alpha N)\alpha N}\cdot(N-\alpha N)^{N-\alpha N}\cdot(\alpha N)^{\alpha N}}\right)^w \cdot 2^r =$$

$$\left(\frac{\sqrt{2\pi N}\cdot N^N}{2\pi\cdot\sqrt{N^2\alpha(1-\alpha)}\cdot(N(1-\alpha))^{N-\alpha N}\cdot(\alpha N)^{\alpha N}}\right)^w \cdot 2^r =$$

$$\left(\frac{\sqrt{2\pi N}\cdot N^N}{2\pi N\cdot\sqrt{\alpha(1-\alpha)}\cdot(N(1-\alpha))^{N-\alpha N}\cdot(\alpha N)^{\alpha N}}\right)^w \cdot 2^r =$$

$$\left(\frac{\cancel{\sqrt{2\pi N}}\cdot N^N}{\sqrt{2\pi N^{\cancel{2}}}\cdot\sqrt{\alpha(1-\alpha)}\cdot N^{N-\alpha N}\cdot(1-\alpha)^{N-\alpha N}\cdot(\alpha N)^{\alpha N}}\right)^w \cdot 2^r =$$

$$\left(\frac{N^N}{\sqrt{2\pi N}\cdot\sqrt{\alpha(1-\alpha)}\cdot N^{N-\cancel{\alpha N}}\cdot(1-\alpha)^{N-\alpha N}\cdot\alpha^{\alpha N}\cdot\cancel{N^{\alpha N}}}\right)^w \cdot 2^r =$$

$$\left(\frac{\cancel{N^N}}{\sqrt{2\pi N}\cdot\sqrt{\alpha(1-\alpha)}\cdot\cancel{N^N}\cdot(1-\alpha)^{N-\alpha N}\cdot\alpha^{\alpha N}}\right)^w \cdot 2^r =$$

$$\left(\frac{1}{\sqrt{2\pi N}\cdot\alpha^{1/2}\cdot(1-\alpha)^{1/2}\cdot(1-\alpha)^{N-\alpha N}\cdot\alpha^{\alpha N}}\right)^w \cdot 2^r =$$

$$\left(\frac{1}{\sqrt{2\pi N}\cdot\alpha^{1/2+\alpha N}\cdot(1-\alpha)^{N-\alpha N+1/2}}\right)^w \cdot 2^r \tag{F.3}$$

Since both α and $(1-\alpha)$ are less than 1, for large values of N Equation (F.3) exhibits exponential growth.