ROYAL HOLLOWAY, UNIVERSITY OF LONDON

Enigma: Prime Numbers and Cryptosystems

by

Adam Mulligan

A final year project submitted in partial fulfillment for the degree of Bachelor of Science, Computer Science

in the
Department of Computer Science

February 2012

"Anyone can design a security system that he cannot break. So when someone announces, Heres my security system, and I cant break it, your first reaction should be, Who are you? If hes someone who has broken dozens of similar systems, his system is worth looking at. If hes never broken anything, the chance is zero that it will be any good."

Bruce Schneier, *The Ethics of Vulnerability Research*

Abstract

Modern cryptography allows us to perform many types of information exchange over insecure channels. One of these tasks is to agree on a secret key over a channel where messages can be overheard. This is achieved by Diffe-Hellman protocol. Other tasks include public key and digital signature schemes; RSA key exchange can be used for them. These protocols are of great importance for bank networks.

Most such algorithms are based upon number theory, namely, the intractability of certain problems involving prime numbers. The project involves implementing basic routines for dealing with prime numbers and then building cryptographic applications using them.

Acknowledgements

With thanks to my project advisor Yuri Kalnishkan, and the RHUL Department of Computer Science for three years worth of knowledge and experience.

Contents

List of Figures

List of Tables

Abbreviations

AES	**A**dvanced **E**ncryption **S**tandard

Chapter 1

Introduction

1.1 What it's about

Secrecy has always been of great importance, not just in modern society, but throughout history. Until very recently, Cryptography was consigned to the sending and receiving of messages, generally using pen and paper. However, increasingly cryptography – the study and implementation of techniques for secure communication – is becoming more vital to the smooth running of even basic systems, whether it's the cliché example of military secrets or simply a micro-payment for an online service. Initially, the usage of provably secure and efficient cryptographic algorithms was limited to governments and related contractors, however the advent of encryption standards, and more relevantly, the creation of public-key cryptography mechanisms, has pushed it in to a wider field of use as researchers gained a better understanding of the area.

1.2 Goals and Intentions

The primary goal of this project can be found in the title and abstract: to research, discuss and create algorithms within or related to the field of prime numbers. To complete this task I will be developing my thoughts and discoveries regarding prime numbers as this report progresses, as well as producing a number of deliverables in the form of software applications, test results and statistics.The topic of number theory is in itself fascinating, and extraordinarily vast, however I will only be scratching the surface.

Nonetheless, I hope to explain throughout this project how prime numbers have become such an important part of modern cryptography, and what they can be used for.

1.3 Project Summary

I will be splitting the project in to four main parts, excluding this report:

1. **Prime numbers in software**
   A discussion of how prime numbers can be efficiently produced programmatically, and software to prove it.
2. **Algorithms**
   The development of algorithms using prime numbers, such as RSA, and complementary cryptographic algorithms such as AES. Alongside this, the development of non-major algorithms in the field of prime number based cryptography that still present an academically interesting concept.
3. **Final Application**
   The production of a software program that utilises the implemented cryptographic algorithms in section 2 to display their efficacy in a real world application.
4. **Cryptanalysis**
   Taking the algorithms created and comparing them with official implementations for statistical analysis, alongside researching and performing "attacks" on them to prove or disprove the implementation's cryptographic security.

The primary algorithms to develop are:

- Asymmetric
  - RSA
  - Diffie-Hellman
- Symmetric
  - AES

- Identification and Authentication
  - RSA Signing
  - Digital Certificates

Other algorithms will be discussed and produced alongside these, however they will not be used in the final application testbed and are purely for research interest. These can be found listed in the contents in the relevant sections.

1.4 Other Information

This document was written and composed with LaTeXusing *Taco Software's* Latexian[1]. The LaTeXsource code for this report should have come bundled with the project documentation and files, however if you are unable to retrieve it please see section 1.4.2.

Eclipse Indigo[2] was used for Java development, the primary language used in this project, along with *MacVim*[3] for general source and text editing.

Git with *GitHub*[4] was used for source control, with *GitHub Issues* used for bug tracking. *Trello*[5] was used for idea and to-do management.

1.4.1 A note on openness

As with any field of study, the quality of research and development is dependent on the open distribution and sharing of ideas. This is *particularly* important with regards to cryptography. As said, cryptography was once reserved to government and research was conducted in secrecy. The open sharing of relevant information in this field is not just for the furthering of knowledge, but also to allow others to inspect and examine algorithms, a process that drastically improves the security of a system. As such, the entirety of this project is licensed under the **GNU Lesser General Public License version 3 or greater** and is available for access publicly online.

[1]http://tacosw.com/latexian/
[2]http://eclipse.org
[3]http://code.google.com/p/macvim/
[4]http://github.com
[5]http://trello.com

1.4.2 Project Repository

If any part of this report is missing, you believe that you do not have a full access to the source code discussed in this document, or if any files have been lost, it is available for full download at `http://cyanoryx.com/files/project.zip`.

Chapter 2

Cryptographic Primitives

2.1 Basics of Information Security

Despite how it is portrayed or colloquially used, Information Security is an entirely different concept and area of study compared to Cryptography. It might seem simple enough to implement a basic cryptographic protocol involving encryption and decryption, however to introduce this into a system and expect the information to be secure, is foolhardy. Cryptography is a *means* to providing information security when following certain rules and guidelines not the be all, end all solution. An understanding of information security, and the related issues, is necessary.

This can be proven using historical evidence: throughout history many complex systems of mechanisms, rules, and protocols have been developed to introduce information security to a system. As with modern day security, this cannot be achieved entirely through mathematical and cryptographic means – it is more than just computational intractability.

As such, stringent criteria for developing secure systems and protocol have been introduced. While institutes such as *The British Computing Society* and *Association for Computing Machinery* ensure their members follow a professional code of ethics, just as a doctor might, these information security criteria are of separate and equal importance. Indeed, there are now several international organisations that exist solely for the overseeing of cryptographic research and development (See: *International Association for Cryptologic Research*).

Often, as we will see, cryptographic systems are simplified for the purposes of presentation particularly for textbooks. This will be discussed further later, with regards to the differences and difficulties involved in developing systems that do not just follow a mathematical "recipe," but also include information security values and other subtleties.

The overall method of dealing with, and ensuring, information security is known as risk management. This encapsulates a large number of countermeasures (including cryptography) that reduce the risk of vulnerabilities in, and threats to, systems. We will only be encountering and discussing the technological areas of mitigation, however some of the solutions include[1]: access control, security policy, physical security, and asset management.

2.2 Objectives

As said, secure systems should follow a guideline, or set of criteria, that ensure the security and integrity of data stored and input. A clear and concise specification should be developed, that will aid the designer in selecting the correct cryptographic primitives, but also help the engineer implement the protocol correctly. There are many of these criteria, however each is derived from four primary objectives:

1. **Confidentiality** is the ability to ensure data is only accessed by those who are allowed to. Maintaining confidentiality of data is an obligation to protect someone else's secret information if you have been entrusted with it.

2. **Authentication** involves identifying both entities and data. Two or more entities wishing to communicate or transmit information to one another must identify each participant to ensure they are who they claim to be - this is known as entity authentication. Data received must be authenticated to ensure the validity of the origin, date sent, contents, etc - this is known as data origin authentication.

3. **Non-repudiation** prevents an entity from denying previous actions they have committed.

[1]For an excellent resource regarding information security, both technical and non-technical, see *Security Engineering*, Ross Anderson

4. **Data Integrity** is how faithfully data compares to it's true state, i.e. proving that a data object has not been altered.

2.3 Key Concepts

As with any discipline, there are a number of fundamental concepts that need to be thoroughly defined. This section will cover the definitions of basic information security and cryptography related concepts. While Computer Scientists and Mathematicians, unlike Biologists, tend to abstract ideas using existing words such as *normal* which ultimately cause confusion to those trying to understand the area of study, the field of Information Security fortunately uses phrases that are succinct and aptly describe notions.

TODO: er, do this bit.

2.4 Primitives

As we discussed in the Objectives, there are certain criteria that must be met for an application to be considered as secure under Information Security guidelines. Excluding physical and psychological measures, there are a number of methods to be implemented cryptographically to guarantee security.

2.4.1 Encryption

Being what is seen as the very 'core' of cryptography, we have defined encryption many times already and what the term means in terms of a process should be apparent. However encryption takes many forms, with each having an appropriate situation for it to be used.

2.4.1.1 Symmetric Key Encryption

Primarily, we will use symmetric key encryption algorithms to encipher data that is to be transmitted between entities.

Mathematically we can formally define symmetric encryption as:

For a message M, algorithm A and key K,

$M' = A(K, M)$

and thus:

$M = A'(K', A(K, M))$

where $A' = A$, $K' = K$ in a symmetric algorithm.

2.4.2 Key Agreement

Key agreement, or key exchange, primarily ensures data integrity and confidentiality. By preventing an attacker from discovering encryption keys used on transmitted data, the attacker should be unable to feasibly read confidential information or modify it (the latter is not entirely true, it may be possible in some cases to modify encrypted data, however we will discuss that later in *Symmetric Cryptography*).

While it might be easier to use asymmetric techniques to encrypt data for transmission, thus allowing us to distribute keys as cleartext, it is slow and inefficient for large quantities of data, such as in an instant messaging application. Because of this, it is prudent to implement an efficient symmetric key encryption algorithm, and share the key (known as a session key) with other entities. However, it would be trivial for an attacker to launch a man-in-the-middle attack and gain access to the encryption keys during the initiation of the conversation, allowing the easy and undetectable decryption of all transmitted messages. As such, session keys will need to be distributed using a key exchange protocol.

2.4.2.1 Key Distribution Centre

The simplest solution is known as a Key Distribution Centre (KDC), which involves the use of a trusted third party (TTP). It is easiest to explain using an example. Alice and Bob are users of a system, attempting to securely communicate. Each share a key securely with third-party Trent (somewhat amusingly, this algorithm does not include how these keys should be shared. We can assume that it was perhaps conducted through an in-person meeting of entities, or other means), who stores each key.

1. Alice initiates a conversation with Bob.
2. Alice requests a session key from Trent, who makes two copies of an identical key and encrypts one with Alice's stored key, and another with Bob's.
3. Alice receives both encrypted keys, and sends the appropriate one to Bob.
4. Alice and Bob decrypt their session keys, leaving both with a shared key.
5. Alice and Bob can now encrypt and decrypt data using the same key.

2.4.2.2 Asymmetric Cryptography

As can be obviously seen, using a KDC is dependent entirely on the ability of two entities to have previously, and securely, shared an encryption key with a TTP that is known to both entities. A solution to this is to eliminate the third party, and use an asymmetric algorithm to share keys directly. As defined in the *Key Concepts* section, asymmetric cryptography allows Alice to share a public-key, with which any entity can encrypt data that can only be decrypted using Alice's private key.

1. Alice and Bob share their public keys using a readily available database.
2. Alice downloads Bob's key, and vice versa.
3. Alice generates a session key, encrypts it using Bob's public key and sends it to Bob.
4. Bob can now decrypt the session key using his private key, resulting in both parties being in possession of a secure session key.

There is a security risk: how can you verify that the entity that sent the encrypted key is indeed the one with which you are trying to communicate? It would be easy for an attacker to encrypt their own session key with Alice's public key, and claim that the key is from Bob. The attacker would then be able to decrypt any messages intended for

Bob. The solution to this issue is the use of digital certificates, and signatures, which will be discussed in *Authentication*.

We will discuss specific algorithms further on, however it is worth pointing out that the current public-key algorithms used for these purposes are *RSA* (Rivest, Shamir, Adleman) and the*Diffie-Hellman Key Exchange* (not defined as a public-key algorithm, however it uses the sharing of public cleartext values). There exist a number of interesting algorithms that implement the public key architecture which will also be considered later.

There are other considerations in key management to ensure confidentiality and integrity. Some provisos exist such as changing the key for each session to ensure perfect forward secrecy, however these are trivial to implement and can be considered as part of the overall application security development.

2.4.3 Authentication

There are two types of authentication that are required in a secure application

1. Message authentication
2. Entity authentication

both of which require different protocols and algorithms. These terms have been defined in *Objectives*.

The methods of entity authentication are an interesting topic in themselves, with many, many protocols having been researched and created as the community tries to find a method that is both secure and easy for an entity to use. Some examples are: passwords, two-factor authentication, PINs, smart cards, biometrics, and so on. These are outside of the scope of this project – we will be implementing two broad, yet specific methods of authentication.

2.4.4 Digital Signatures

Digital signatures, as we will see, encompass three of the information security criteria: authentication, data integrity and non-repudiation (a sender cannot claim they did not send the message). A digital signature is a string that connects a message with its originating entity.

When transmitting a message, the sender signs the message with their private key which can then be verified with their public key, which should be available to the receiver.

1. Alice signs her message A with her private key.
2. Alice sends message A with signature S to Bob $(A|S)$
3. Bob retrieves Alice's public key from an available database, and verifies signature S

The first and most common implementation of digital signatures is RSA.

2.4.5 Public-key Certificates

Digital signatures and a public-key infrastructure, however, are not enough by themselves. A very simple attack can be orchestrated similar to that during key agreement: Mallory, the attacker, could sign a modified message A with her own private-key and then distribute her public-key, claiming it to be Alice's. To counter this, we can introduce a trusted third-party, known as a Certificate Authority (CA), who signs Alice's public-key with their own private-key. Most trusted CA's public keys come bundled with software such as browsers and operating systems.

1. Certificate Authority signs Alice's public key with their private key.
2. The CA distributes its public key with major software.
3. Bob receives Alice's message and signature, as well as her public key and signature.
4. Bob verifies Alice's public key with the CA's public key, and then verifies the message.

2.4.6 Hashing

While not of direct relevance in information security, hashing plays a significant part in cryptographic systems and thus is included as a concept to be implemented. Formally, we can define a hash function as mapping a large domain to a smaller range - in the case of data, mapping a set of bytes to a unique identifier with a set length. A hash function, at the very basics, takes as input a message and produces a fingerprint, or digest, of the input. Within the field of cryptography, they are used for message authentication and data integrity.

This is to say, given a domain D and range R for $f : D \rightarrow R, then |D| > |R|$. This is a many-to-one relationship, the downside of which means that collisions can occur – two input strings resulting with the same output string – however, this varies between algorithms, the more modern of which are less likely to result in collisions.

A hash function can be classed into two categories: keyed and unkeyed, taking both a message and secret key and taking just a message, respectively. Two conditions are necessary for a hash function to be effective:

1. compression – the function f maps input a of arbitrary length to an output of fixed length, n.
2. complexity – it must be easy to compute $f(a)$

Most commonly used are unkeyed, one-way hash functions. Some examples of which are: SHA-1, and MD5. In cryptography, hashes are commonly used for data integrity in combination with digital signatures. A message is hashed, and the fingerprint produced is signed by the entity. There are some algorithms designed specifically for this purpose, known generally as Message Authentication Codes (MACs) and Manipulation Detection Codes (MDCs).

A sample of each of these four primitives will be implemented further in the report.

2.5 Mathematics

This section will cover some of the basic mathematical concepts that will be used throughout the report, and form a foundation of understanding for the more complex abstract methods that will be used.

2.5.1 Notation

1. $\mathbb{Z}$ is the set of all integers.
2. $\mathbb{R}$ is the set of all real numbers.
3. $[a, b]$ is the set of integers n such that $a \leq n \leq b$.
4. $|A|$ is the cardinality of a finite set, i.e. the number of elements.
5. $n \in A$ denotes that an element n exists in set A.
6. $A \subseteq B$ denotes that set A is a subset of set B.
7. $A \subset B$ denotes that A is a subset of B, but $A \neq B$.
8. $\lceil a \rceil$ is the smallest integer greater than or equal to a.
9. $\lfloor a \rfloor$ is the largest integer less than or equal to a.
10. A function (mapping) denoted as $f : A \rightarrow B$ signifies that every element a in A is mapped to exactly one element b in B. This can also be written as $f(a) = b$.

2.5.2 Number Theory

We will begin by defining some fundamental rules of number theory.

Definition 2.5.1. Let a, b be integers. a divides b if an integer c exists such that $b = ac$. We define this as a divides b, or $a|b$.

Definition 2.5.2. An integer c is a common divisor if $c|a$ and $c|b$.

Definition 2.5.3. An integer c is the greatest common divisor of a and b, if:

1. c is a common divisor of a, b.

2. when $d|a$ and $d|b$, $d|c$.

3. $c \geq 0$

This is denoted as $c = gcd(a, b)$.

Definition 2.5.4. An integer a is prime if:

1. $a \geq 2$

2. the only positive divisors are 1 and a.

Otherwise, the number is referred to as composite.

Fact 2.5.1. There are an infinite number of prime numbers.

Definition 2.5.5. If a is prime and $a|bc$, then $a|b$ and $a|c$.

Definition 2.5.6. Integers a and b are relatively prime (coprime) to one another, if $gcd(a, b) = 1$.

Definition 2.5.7. The function ϕ is known as the Euler Phi function. Where $n \geq 1$, $\phi(n)$ is the number of integers in the interval $[1, n]$ that are relatively prime to n.

1. If n is prime, then $\phi(n) = n - 1$.

2. $\phi(n)$ is multiplicative: if $gcd(m, n) = 1$, then $\phi(mn) = \phi(m) \cdot \phi(n)$.

2.5.2.1 Modulo Arithmetic

Definition 2.5.8. Where a and b are integers, a is congruent to b modulo n (denoted as $a \equiv b(mod\ n)$. n is known as the modulus. Congruence is reflexive, transitive and and symmetric. That is to say, for every $a, b, c \in \mathbb{Z}$:

1. $a \equiv a(mod\ n)$

2. if $a \equiv b(mod\ n)$, then $b \equiv a(mod\ n)$

3. if $a \equiv b(mod\ n)$ and $c \equiv c(mod\ n)$, then $a \equiv c(mod\ n)$

This is known as an equivalence relation.

Definition 2.5.9. The set of all integers modulo n is denoted as $\mathbb{Z}_n$.

As you can see, there are a great number of theorems regarding prime numbers and modulo arithmetic (we have barely scratched the surface), some of which we will be using.

2.5.3 Abstract Algebra

Further on, particularly in the development of symmetric algorithms like AES, we will be using groups, fields and other abstract algebraic objects. Due to the complex nature of these concepts, they will be explained in tandem with the algorithms themselves.

2.5.4 Complexity

Computational complexity is a vast subject, mostly out of the scope of this report, however we will use the notation occasionally to classify algorithms. Big-oh notation, as it is known, will be used to represent the worst-case running time of an algorithm based on a standard input size. For example:

Definition 2.5.10. A polynomial-time algorithm is an algorithm where the worst-case running time can be represented as $O(n^c)$, where n is the input size, and c is a constant.

An algorithm with a running time that cannot be bounded as such is known as an exponential-time algorithm. It is generally considered that polynomially-time algorithms are efficient, wheres exponential-time algorithms are inefficient.

2.6 Moving On

This is just a basic overview of the cryptographic and mathematical primitives used in Information Security. As the report progresses, each concept will be explained in finer detail alongside their implementations. While it seems hard to apply these abstract definitions, the purpose of each within our algorithms will become quickly clear.

Chapter 3

Number Theory and Public-key Cryptography

3.1 Overview

Number theory, the unit of mathematics that studies integers and their properties, was once a predominantly useless area. However, as cryptographic algorithms became more prevalent, particularly schemes that require the use of large prime numbers. We have already covered the notation used in the majority of number theoretic algorithms, and so this section will build upon this and discuss basic concepts and applications: modular arithmetic, prime number generation, and factorisation.

In this section we will make the assumption that any attackers who may try to break our algorithms or eavesdrop on our messages are extremely powerful and capable, and so we will discuss algorithms with regards to their tractability – in this case, whether or not a significant percentage of all instances of a problem can be solved in polynomial time.

Throughout, where appropriate, programming examples are provided in Java.

3.2 Modular Arithmetic and Congruence

As we know, the core concept of modular arithmetic is to return the remainder of the division of two integers, represented as $a(mod\ n)$. It is said that b is congruent to $a(mod\ n)$ if $b \equiv a(mod\ n)$. Congruences play a great part in cryptography – one of the very first uses was the Caesar Cipher. Messages were made secret – encrypted – by shifting letters forward by an integer n, where $0 < n \leq 25$.

Given a sequence of characters $\{A, B, C, ..., Z\}$ where each letter is represented by a number $\{0, 1, 2, ..., 25\}$. Defined by a function $f(n)$, where n is the integer representation of a character:

$$f(n) = (n + x)\ mod\ 26$$

where x is the number of characters to shift by.

This is a very simple example of how congruences can be used in cryptography. As we will see in the public-key section, congruences can be applied in a similarly simple way as a vital component part of a far more complex algorithm.

Indeed, although the technicalities of this are not relevant, congruences can even be used to generate pseudo-random numbers.

3.3 Prime Numbers

Primes have already been covered, however they are a deeply fascinating subject with many current open problems and conjectures. For example, Goldbach's Conjecture states that every odd integer n, where $n > 2$, is the sum of two primes. This is intriguing because so far this has been *verified* for integers up to 2×10^{17}, but mathematicians have not yet found a formal proof for this, and despite this they continue to believe it to be true [1].

We, however, will be using them for a far more interesting[1] purpose: cryptography based around the intractability of factoring very large composite numbers into their smaller non-trivial prime divisors.

[1]http://xkcd.com/247/

3.3.1 Generation

First and foremost, we must devise an efficient method for generating the prime numbers. Although generating the keys for use in prime number based algorithms is generally not done on a regular basis, it is still a complex task to complete in a reasonable amount of time. As it stands, a number of algorithms already exist (and have done for many hundreds of years) for this purpose.

3.3.1.1 A Note on Random Number Generators

The random number generators mentioned thus far, and all used later, cannot be considered *truly* random. They are known as *pseudo-random number generators* – sequences generated deterministically based on a seed value. The sequences are produced algorithmically based on a relatively small set of values, Generally these sequence can be considered *practically* random for most uses, assuming the seed is not publicly known and is truly random.

Most truly random numbers are generated by analysing physical methods, such as nuclear decay or cosmic background radiation. However, typically this tends to be costly and is usually reserved only for applications that explicitly need high-security, non-deterministic random numbers. Recently the technology has become more available due to the widespread availability of online services – for example, RANDOM.ORG[2] offers an API that returns truly random numbers on request. This naturally comes with its own security downsides, but can be used for non-cryptographic applications. Even then, RANDOM.ORG charges for clients that go over a quota of requests.

The generation and application of random numbers is an extensive and current area of research. For the purposes of the cryptographic utilities to be developer, and the Enigma application, we will be used the *Java SDK* class `SecureRandom`. `SecureRandom` is a cryptographically strong pseudo-random number generator that complies and follows tests specified by FIPS-140-2[3], and also uses non-deterministic seed material, as defined by *RFC 1750*[4]. Assuming the proper use of this class, it is about as close as we will get

[2]http://random.org
[3]Section 4.9.1
[4]http://www.faqs.org/rfcs/rfc1750.html

to generating secure, non-deterministic random numbers without resorting to physical methods.

In short, and for our purposes, the generation of secure random numbers is based around maintaining the secrecy of a truly random seed on which a random sequence can be generated.

3.4 Public-key Cryptography

3.5 Integers

3.5.1 RSA

So far we have only covered the concepts of number theory and how they're used in cryptography. In this section we will discuss exactly how they are implemented, and go in to greater detail about the algorithms and techniques required.

3.5.1.1 Implementation

3.5.1.2 Textbooks and OAEP

3.5.2 Diffie-Hellman

3.5.2.1 Implementation

3.6 Other Variations Implementing PKI

Chapter 4

Symmetric Cryptography

Chapter 5

Identification and Authentication

5.1 Overview and Intentions

5.2 Basic and Common Schemes

5.3 Digital Signatures

5.4 Certificates

5.5 Other Methods

Chapter 6

Enigma: A Testbed

6.1 Overview and Intentions

The intention of this project is to produce an application that allows two users to securely identify and authenticate one another, and communicate with text messages via a presumed-insecure network. However, the primary *goal* is to create a testbed that allows any cryptographic algorithms to be integrated in place, to provide a platform for further analysis, such as comparison of run-times between open and closed source implementations.

6.2 Engineering Methodologies and Planning

6.2.1 Methodology

The main focus of the development process was to break the application down in to its component pieces, and iteratively develop them so that minimalist functionality sets could be produced and then combined in the shortest possible times. The overarching category for this style of production is known as *agile development*: a practice based around iterative and incremental development. More specifically, a subset of agile development will be used – Scrum. The goals of Scrum fit neatly with the desired development cycle in that programming is done in ”sprints,” with a deadline organised at the start of these sprints and a set of tasks that must be completed by the end. A sprint can last anywhere between one week and one month. This, combined with test driven development (TDD, writing tests for functionality before producing the functionality), produces a set of components matching a well-defined requirements document that can be combined in to the final product.

However, without results, methodologies are meaningless

6.2.2 Documentation

Documentation is an extremely important part of software engineering. It is a broad concept, and encompasses: requirements and specification, design overview, technical details, user manuals and even marketing information. However, in terms of the software development process, it is only the first three that are of direct importance.

It is said a program listing should be documentation unto itself: the programming style should allow an overall structure and purpose to be easily determined from examining the code [2]. However, this is an idealistic view and design and technical documentation are of great importance, not only for the developer currently producing the software, but for those possible developers in the future who have to maintain the code.

As a major component of this project is to develop a library of cryptographic algorithms, having a clear and accurate listing of all available methods in the API is vital. Conveniently, Java and the Eclipse IDE are tightly integrated with *JavaDoc*[1], a documentation generator that automatically produces standardised API documentation in HTML format based on comments inserted in the code. The comment blocks – distinguished using the format `/** ... */` – contain a method description, and a number of control sequences that give detailed information about what the method returns, the parameters it takes and other details such as exceptions thrown.

JavaDoc outputs are bundled with the appropriate packages in the file tree, and manual technical documentation is found in the appendices. A requirements specification has been compiled in *Appendix A*.

[1]http://java.sun.com/j2se/javadoc

6.3 Application Development

6.3.1 User Interface

*Packages covered in this section: com.cyanoryx.uni.enigma.gui.**

6.3.1.1 GUI Frameworks

6.4 Protocol Implementation

*Packages covered in this section: com.cyanoryx.uni.enigma.net.**

6.5 Algorithm Implementation

*Packages covered in this section: com.cyanoryx.uni.crypto.**

6.6 Usage

A user's instruction manual is included as *Appendix D*. This manual also briefly covers compilation, required dependencies, and other build related information.

6.7 Program Listing

Due to the size of the project a detailed, printed code listing is impossible, and thus it is recommended that the code be viewed using a text editor or other text environment on a device. If you do not have a digital copy of this project, please see *Section 1.4.2*.

Chapter 7

Cryptanalysis

7.1 Public-key Cryptography

7.1.1 RSA

A distinction must be made between the security of the RSA *algorithm* and the RSA *system*. This is known as semantic security [3], the use of other non-algorithmic techniques to resist attacks and make it, for example, difficult to recover information from the public key. As we discussed in Chapter 3, standards exist such as RSA-OAEP that introduce elements of randomness in to the RSA algorithm that limit certain basic attacks – this is the RSA algorithm implemented in to a cryptosystem. The RSA algorithm itself is distinct from this at is the core mathematics, rather than an implementation of a secure system.

In this section, we will discuss both attacks on the RSA *algorithm* and attacks against RSA *systems*, such as the one we have implemented.

7.1.1.1 Brute Force

The first attack we should mention is known as the *brute-force* attack: factorising the integer modulus N to determine the prime numbers used to generate the keys. This has been discussed throughout the paper, and due to the simplicity (to an extent) of the attack, we will not extensively cover it. It is interesting to note that there are no (publicly available) efficient algorithms with an acceptable running time for factoring large prime multiples. The current fastest algorithm is the *General Number Field Sieve*, which runs on $\exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}}\right) = L_n\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right]$ time, with an n-bit integer. See [4] for an interesting introduction to GNFS.

If an algorithm for factoring large integers in reasonable time periods is ever discovered, RSA will be rendered entirely insecure. However, until then we will only consider attacks that can decrypt RSA ciphertext *without* factoring the modulus N.

7.1.1.2 Elementary Attacks

Small Exponent

Private Exponent

Public Exponent It is common belief that using a small encryption exponent e does not adversely affect the capabilities of the RSA algorithm. For example, it is suggested $e = 3$.

This is best shown as an example. If Alice wishes to send a message m to three associates with the public moduli $n_{\{1,2,3\}}$ and exponent $e = 3$, she would send $c_i = m^3 \bmod n_i$. Through a eavesdropping, an attacker Mallory could use Guass's Algorithm all three values of c to find:

$$\begin{cases} x \equiv c_1 \ mod \ n_1 \\ x \equiv c_2 \ mod \ n_2 \\ x \equiv c_3 \ mod \ n_3 \end{cases}$$

Using the Chinese Remainder Theorem, we can determine that $x = m^3$, and thus $m = \sqrt[3]{x}$. Also, if a message $m < n^{\frac{1}{e}}$ we can calculate the e^{th} root of $(c = m^e)$ to get the plaintext message.

It should be noted that this attack cannot be considered a "break" of the algorithm. Both these attacks can be mitigated through the use of salting – adding randomly generated bits to messages before encryption. Another simple, but not recommended, solution is to make $e \geq 2^{16} + 1$.

Due to the use of hashing and salting, this attack does not affect RSA cryptosystems such as RSA OAEP, the system implemented for Enigma.

Key Exposure

Coppersmith's Short Pad Attack

Hastad's Broadcast Attack

Related Message Attack

Forward Search Attack

Common Modulus Attack A previously common solution to managing the keys of multiple entities within one network was creating a central authority that would select a modulus N, and then share exponent pairs e_i and d_i with the entities. While this attack relies on the ability to factor prime multiples, given (e_i, d_i), an attacker could determine the decryption exponents for all other entities within the network.

Cycling Attack

Message Concealing

7.1.1.3 System and Implementation Attacks

Timing

Random Faults

Bleichenbacher's Attack

7.1.2 Certificates and Authentication

7.1.2.1 Implementation Attacks

SSL BEAST

Authority Security and Impersonation The underlying security of the certification system is based upon a trusted third party (TTP) that verifies the chain of trust – the certificate authority (CA) – and so "a chain is only as strong as its weakest link" appears to be quite apt. If, through social or technical means, an attacker can gain access to the private keys or construction information for the private keys of a CA, they will be able to issue certificates as they wish until the breach is noticed and the CA can be removed from the chain.

This is not a theoretical attack by any means. In July 2011, Mozilla – the developers of many open source products such as Firefox and Thunderbird – was informed that a fraudulent SSL certificate belonging to Google, Inc. had been issued by CA *DigiNotar*. As it happens, DigiNotar's network had been breached allegedly without their knowledge, giving the attacker access to their private keys and thus allowing certificates to be issued arbitrarily. In this case, it was shown to be used by unknown entities in Iran to conduct a man-in-the-middle against Google services [5].

7.2 Symmetric Cryptography

This sections covers attacks affecting the AES symmetric cipher.

7.2.1 Brute-force

7.2.2 XSL Attack

7.2.3 Biryukov and Khovratovich

7.2.4 Related Key Attack

7.2.5 Known-key Distinguishing Attack

7.2.6 Bogdanov, Khovratovich, and Rechberger

7.2.7 Side-channel attacks

7.3 Hash Functions

7.3.1 Birthday Attacks

7.3.2 Collision and Compression

7.3.3 Chaining Attack

7.4 Emerging Threats

7.4.1 Quantum Cryptography and Cryptanalysis

Quantum computing is an emerging technology.

As well as affecting algorithms based around the intractability of integer factorisation, quantum cryptanalysis also affects algorithms that utilise the discrete logarithm problem such as ElGamal, Diffie-Hellman and DSA.

Quantum cryptography, in its current form, does not affect modern symmetric cryptographic algorithms – neither ciphers or hash functions. Grover's algorithm, an algorithm that improves the efficiency of searching an unsorted database, improves the speed at which a symmetric cipher key can be cracked, however this is preventable using standard counter-actions such as increasing the key size.

An interesting current field of study is post-quantum cryptography: algorithms designed such that a cryptanalyst with a powerful quantum computer (should they become prevalent, or even plausibly workable in the future) cannot easily break. See [6].

7.4.2 Ron was wrong, Whit is right

In early February 2012, a paper entitled "Ron was wrong, Whit was right" – a slight jab at RSA co-creator Ronald Rivest and nod towards cryptographer Whitfield Diffie – swept through the headlines of the cryptographic communities. Written by researchers at the *School of Computer and Communication Studies, École Polytechnique Fédérale De Lausanne.* The paper detailed a "sanity check" of a subsection of RSA public keys that can be found online, and analysed them to test the randomness of the inputs used to calculate the keys.

As we are now well aware, RSA is based entirely on the inability to efficiently factor prime numbers. However, there are other requirements for the good security of the algorithm. Namely, it is crucial that when the keys are generated, previous random numbers are not reused. [7] states that:

Given a study of 11.7 million public keys,

Conversely, the researchers were unable to find any of the common exponents, used in RSA public keys, being used for the other two major public-key algorithms: DSA and ElGamal. ECDSA was also investigated, however only one certificate was found to be using ECDSA.

7.4.2.1 Sony PlayStation 3

An unrelated, yet high-profile, realisation of the risks of poor entropy in random number generators is the cracking of Sony's PlayStation 3 console. Sony used public-key cryptography to sign its bootloaders and games, which prevents unauthorised software being executed on the device. The overall system used to protect the device consists of many different techniques and algorithms, however it was the implementation of *Elliptic*

Curve Digital Signature Algorithm (ECDSA) that was flawed. As we will show below, a poorly executed random number generator design led to the extraction of the private key stored securely within the device, previously implausible to retrieve.

ECDSA uses 11 parameters in its cryptographic algorithms: 9 public, and 2 private:

Public

$$p, a, b, G, N = \text{curve parameters}$$

$$Q = \text{public key}$$

$$e = \text{data hash}$$

$$R, S = \text{signature}$$

Private

$$m = \text{random number}$$

$$k = \text{private key}$$

The signature, R, S is computed:

$$R = (mG)_x$$

$$S = \frac{e+kR}{m}$$

Because of this, it is absolutely vital that a high-entropy random number generator be used to produce m, otherwise given two signatures with the same m, we are able to determine m *and* private key k:

$$\begin{array}{ll} R = (mG)_x & R = (mG)_x \\ S_1 = \frac{e_1+kR}{m} & S_2 = \frac{e_2+kR}{m} \end{array}$$

Rearranging:

$$S_1 - S_2 = \frac{e_1-e_2}{m}$$

$$m = \frac{e_1-e_2}{S_1-S_2}$$

$$\therefore k = \frac{e_1S_2-e_2S_1}{R(S_1-S_2)} = \frac{mS_n-e_n}{R}$$

Given k, we have now have the capability to sign arbitrary data, meaning: the chain of trust is broken, signed executables are no longer useful, encrypted storage can be accessed, and many other features of the security system rendered ineffective [8].

While the human-impact was virtually zero in terms of loss of life, injury, etc. this is an excellent example of how something seemingly easy to implement like an RNG can affect the overall security of an entire system and the impact it can have.

7.5 A comment on theory

It is interesting to note, based upon history, that in all likelihood these theoretical attacks are currently being implemented. Though

However, the topic of real-world security flaws and those that take advantage of them is extensive and could easily be covered by its own paper, if not several. It is left up to the reader's imagination to consider the many possible vulnerabilities currently being utilised unbeknownst to the vast majority of the community, and also to forget this idea lest they never use a networked device again.

Chapter 8

Outcomes and Further Research

8.1 Fulfilment of Specification

8.2 Testing

Various testing methodologies

8.2.1 Test Driven Development

8.2.2 Unit Testing

8.2.3 Code Coverage

8.2.4 Functional Testing

Functional testing is a subset of black box testing. However, it was decided in the case of Enigma that due to the low complexity of the software and use-flow,

8.3 Problems and Evaluation

8.4 Limitations

8.4.1 Standards Compliance

8.4.2 Hardware Implementation

8.5 Project Management

8.6 Summary

Appendix A

Enigma Application Specification

A.1 Introduction

A.1.1 Purpose

A.1.2 Scope

A.1.3 Definitions

A.1.4 Overview

A.2 Overall Description

A.2.1 Product Perspective

A.2.2 Product Functions

A.2.3 User Characteristics

A.2.4 Constraints

A.2.5 Assumptions and dependencies

Appendix B

Enigma Protocol Specification

B.1 Introduction

The Enigma Protocol is a communications protocol intended for use in Instant Messaging applications. It is very loosely based around the concept of *Jabber/XMPP*[1], primarily as it is an application of the Extensible Markup Language (XML) to allow for near-real-time communications between networked devices. The XMPP protocol is considerably more complex than Enigma, and the two should not be considered comparable.

This document covers the basic XML elements that must be implemented by an Enigma Protocol-based application. As its primary purpose is to provide a simple way of sending text with meta-data, it should only be used for research purposes. The Enigma Application can be considered a reference implementation, however it is not a complete implementation.

B.2 Requirements

An IM application for use in the testing of sending encrypted messages should have the capability to:

1. Be able to exchange brief text messages in near-real-time.

[1]http://xmpp.org/about-xmpp/history/

2. Transmit information that can be used to securely generate and exchange encryption keys.
3. Transmit information that can be used to identify and authenticate.

And thus, any application implementing the Enigma Protocol must be able to adhere to the above requirements.

B.3 History

This document covers version 1.0 of the Enigma protocol. There are no prior implementations.

B.4 Terminology

No specialised terminology is used without explanation in this document.

B.5 Format

A conversation between two users should be considered as the building of a valid XML document using the `enigma:client` namespace. The document should consist of a valid root element occurring only once, valid child elements matching those listed in the document, and finish with a closing tag matching the root element. Each packet, that is not a root element, must consist of a start-tag and end-tag, or an empty-element tag otherwise it **should not** be parsed and **should** be dropped.

The order of the messages is important if the `time` attribute is not included, in which case they should be handled and displayed in a first-in-first-out order.

B.6 Commands

B.6.1 Connection

A connection is represented by a streaming XML document, with the root element being `<stream>`.

- **Opening a connection:**

```
<stream  to="SERVER_NAME"
         from="USER_NAME"
         id="SESSION_ID"
         return-port="LOCALHOST_INBOUND_PORT"
         xmlns="enigma:client">
```

- **Closing a connection:**

```
</stream>
```

B.6.2 Authentication

- **Toggling encryption:**

```
<auth stage="streaming"
      id="SESSION_ID"
      type="toggle">
      [off|on]
</auth>
```

  Note: this requires the agreement of both users. The connection should be closed if one user disagrees to change the current encryption status.

- **Asserting the key agreement method:**

```
<auth stage="agreement"
      id="SESSION_ID"
      type="method">
      [method type identifier]
</auth>
```

- **Publishing a certificate:**

```
<auth stage="agreement"
      id="SESSION_ID"
      type="cert">
      [Base64 encoded Enigma Certificate]
</auth>
```

- **Publish an encrypted symmetric cipher key:**

```
<auth stage="agreement"
      id="SESSION_ID"
      type="key"
      [method="CIPHER_ALGORITHM"]>
      [Base64 encoded encrypted key]
</auth>
```

B.6.3 Messaging

- **Sending a message:**

```
<message  [to="REMOTE_USER_NAME"]
          from="USER_NAME"
          id="SESSION_ID"
          [type=""]>
    <body>MESSAGE_CONTENT</body>
</message>
```

B.6.4 Errors

Errors do not have a specific element themselves, but are included as a subelement of any other tag, setting `att:type` to `error`.

- **Sending an error:**

```
<element  [other attributes]
          type="error">
    <error type="ERROR_NUMBER">
        ERROR_MESSAGE
    </error>
    [element contents]
</element>
```

Appendix C

Licenced Software Usage

The Enigma application and associated utilities make use of several libraries and projects, all of which are used legally and in accordance with the matching software licence. The individual licence declarations are included within the files, or where appropriate. Below each library/project is listed, with their locations and other details.

1. **Base64 Encoder/Decoder**
   Author: Robert Harder
   Website: http://www.iharder.net/current/java/base64/
   Licence: None. Public domain release, full usage allowed.
   Location: *com.cyanoryx.uni.common.Base64*

2. **Cryptix Byte Utilities**
   Author: The Cryptix Foundation Ltd.
   Website: http://www.cryptix.org/
   Licence: Cryptix General Licence - http://www.cryptix.org/LICENSE.TXT
   Location: *com.cyanoryx.uni.common.Bytes*
   Notes: Most methods only used partially.

3. **Apache Xerces (XML parsing)**
   Author: Apache Software Foundation
   Website: http://xerces.apache.org/
   Licence: Apache Licence, v2.0 - See licence at *dep/APACHE-LICENSE-2.0.txt*
   Location: *dep/xerces.jar*

4. **Instant Messaging in Java**

   Author: Iain Shigeoka (Manning)

   Website: http://www.manning.com/shigeoka/

   Licence: The Shigeoka Software Licence - See licence at *dep/SHIGEOKA-LICENCE.txt*

   Location: *com.cyanoryx.uni.enigma.net.io.XercesReader.java*, plus occasional usage of code fragments and methods, which are noted within the code.

Appendix D

Enigma: User Manual

Bibliography

[1] Kenneth H. Rosen. *Discrete Mathematics and Its Applications.* McGraw-Hill, 6th edition, 2007.

[2] Steve McConnell. *Code Complete.* Microsoft Press, 2nd edition, 2004.

[3] S. Goldwasse. The search for provably secure cryptosystems, cryptology and computational number theory. *Proc. Sympos. Appl. Math.*, 42, 1990.

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

[5] August 2011. URL `http://googleonlinesecurity.blogspot.com/2011/08/update-on-attempted-man-in-middle.html`.

[6] Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen. *Post-Quantum Cryptography.* Springer, 2009.

[7] D. Loebenberger and M. Nusken. *Analyzing standards for RSA integers*, volume 6737 of *Lecture Notes in Computer Science.* Springer, 2011.

[8] Bushing, Marcan, Segher, and Sven. Ps3 epic fail. Technical report, fail0verflow, 2010.