Henna Pietiläinen

Elliptic curve cryptography on smart cards

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietojenkäsittelytekniikan laitos

Helsinki University of Technology
Faculty of Information Technology
Department of Computer Science

HELSINKI UNIVERSITY OF TECHNOLOGY		ABSTRACT OF THE MASTER'S THESIS	
Author:	Henna Pietiläinen		
Name of the thesis:	Elliptic curve cryptography on smart cards		
Date:	October 12, 2000	**Number of pages:**	81
Department:	Faculty of Information Technology	**Professorship:**	Tik-106
Supervisor:	Professor Eljas Soisalon-Soininen		
Instructor:	Marko Nordberg, M.Sc.		

In 1985 Neal Koblitz and V.S. Miller proposed elliptic curves to be used for public key cryptosystems, whereas RSA, a nowadays widely used public key cryptosystem, was developed by Rivest, Shamir, and Adleman almost ten years earlier in 1977. The elliptic curve cryptosystem benefits from smaller key sizes than RSA, which makes its cryptographic operations, encryption, decryption, signing, and signature verification faster than RSA's operations.

A smart card is a single-chip microcomputer with a size of 25 mm^2 at most. Today smart cards are used mainly for electronic identification and storing user information. Smart cards are also used to store private keys and to execute cryptographic operations which use private keys.

This Master's thesis examines whether elliptic curve cryptography is better suited to be used on smart cards than the nowadays widely used RSA. It describes the elliptic curve cryptography and RSA implementations used to compare these two cryptosystems, and presents performance comparisons based on these implementations. In addition, this thesis contains security and space requirement comparisons between these two cryptosystems.

According to the test results, signing and decryption operations are faster with the elliptic curve cryptosystem than with RSA, but RSA is faster when encrypting messages or verifying signatures. On the other hand, the elliptic curve cryptosystem needs less space to store the private keys than RSA, and is thus well suited to be used on smart cards. The elliptic curve cryptosystem has the disadvantage that the Menezes-Vanstone encryption increases the size of encrypted messages considerably more than RSA encryption does. In addition, because an elliptic curve cryptosystem implementation is more complicated and requires deeper mathematical understanding than an RSA implementation, it is more susceptible to errors which diminishes its security.

Keywords: Elliptic curve cryptography, RSA, smart card, digital signatures, Nyberg-Rueppel signature algorithm, Menezes-Vanstone encryption algorithm, Java Card emulator.

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Henna Pietiläinen		
Työn nimi:	Elliptisten käyrien salausjärjestelmä toimikorteilla		
Päivämäärä:	12.10.2000	**Sivuja:**	81
Osasto:	Tietotekniikan osasto	**Professuuri:**	Tik-106
Työn valvoja:	professori Eljas Soisalon-Soininen		
Työn ohjaaja:	DI Marko Nordberg		

Elliptisten käyrien käyttöä julkisen avaimen salausjärjestelmissä ehdottivat Neal Koblitz ja V.S. Miller ensimmäisen kerran vuonna 1985. Nykyään hyvin yleisesti käytössä olevan julkisen avaimen salausjärjestelmän, RSA:n, taas kehittivät Rivest, Shamir ja Adleman melkein kymmenen vuotta aikaisemmin, vuonna 1977. Elliptisten käyrien salausjärjestelmän etuna RSA:han verrattuna on avainten pieni koko, jonka ansiosta myös kryptografiset operaatiot, salaus, salauksen purku, allekirjoittaminen ja allekirjoitusten tarkistus ovat nopeita.

Toimikortti on yksiprosessorinen minitietokone, jonka koko on enintään 25 mm^2. Nykyään toimikortteja käytetään pääasiassa sähköiseen tunnistamiseen ja käyttäjäkohtaisten tietojen säilyttämiseen. Korteilla säilytetään usein salaisia avaimia ja suoritetaan salaista avainta käyttäviä kryptografisia operaatioita.

Diplomityössäni tutkin soveltuuko elliptisten käyrien salausjärjestelmä paremmin toimikorteilla käytettäväksi kuin nykyään paljon käytetty RSA. Työ kuvaa vertailua varten tekemäni elliptisten käyrien salausjärjestelmän ja RSA-toteutuksen ja sisältää näiden toteutusten pohjalta tehtyjä suorituskykyvertailuja. Lisäksi diplomityöni sisältää järjestelmien turvallisuuden ja tilantarpeen vertailua.

Saatujen tutkimustulosten perusteella allekirjoitus- ja salauksen purkuoperaatiot ovat nopeampia elliptisten käyrien salausjärjestelmissä kuin RSA:ssa, mutta RSA:lla voi salata viestin nopeammin ja tarkistaa allekirjoituksen nopeammin. Toisaalta elliptisten käyrien salausjärjestelmä tarvitsee vähemmän tilaa avainten säilyttämiseen kuin RSA ja soveltuu tämän vuoksi hyvin toimikorteilla käytettäväksi. Elliptisten käyrien salausjärjestelmien ongelmana on se, että testattu salausalgoritmi Menezes-Vanstone kasvattaa salatun viestin pituutta enemmän kuin RSA:n salausalgoritmi. Lisäksi elliptisten käyrien salausjärjestelmän toteuttaminen vaatii laajempaa matematiikan hallintaa ja on järjestelmänä paljon monimutkaisempi kuin RSA-toteutus, mikä aiheuttaa virhealttiutta ja sen seurauksena turvallisuusriskin.

Avainsanat: Elliptisten käyrien kryptografia, RSA, toimikortti, digitaaliset allekirjoitukset, Nyberg-Rueppel -allekirjoitusalgoritmi, Menezes-Vanstone -salausalgoritmi, Java Card -emulaattori.

Acknowledgements

This Master's thesis has been done for Sonera SmartTrust.

I want to thank my supervisor, Professor Eljas Soisalon-Soininen, for his help and comments. I also want to thank Pekka Nikander for helping me getting started and supporting me.

I wish to thank my coworkers who read and commented on the draft versions of this thesis: Veera Lehtonen, Christian Wieczerkowski, Satu Laukkanen, Mikko Mättö, Ville Likitalo, and Otto Kolsi.

I would also like to thank Petri Paavilainen for giving me references and information on Smart Cards.

My gratitude also goes to Kristiina Volmari-Mäkinen for her grammatical comments.

Finally, I would like to thank my family and especially Ville Laurikari for their patience and advice.

Otaniemi, October 12, 2000

Henna Pietiläinen

Contents

Abbreviations and Mathematical Notations

Abbreviations

ANSI	American National Standards Institute
APDU	Application Protocol Data Unit
APEC	Asia Pacific Economic Co-operation
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
	Automatic Teller Machine
CPU	Central Processing Unit
CRT	Chinese Remainder Theorem
DES	Data Encryption Standard
DFA	Differential Fault Attack
DLP	Discrete Logarithm Problem
DPA	Differential Power Analysis
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECC	Elliptic Curve Cryptosystem
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
ECKA	Elliptic Curve Key Agreement
ECM	Elliptic Curve Factoring Method
ECNRA	Elliptic Curve Nyberg-Rueppel Algorithm
ECTP	Elliptic Curve Transport Protocols
EEPROM	Electrical Erasable Programmable Read Only Memory
EMSA	Encoding Method for Signatures with Appendix
EMSR	Encoding Method for Signatures with Message Recovery
EMV	Europay, MasterCard, Visa
ETSI	European Telecommunications Standards Institute
FINEID	Finnish National Electronic Identity
FSTC	Financial Services Technology Consortium
GCD	Greatest Common Divisor
GF	Galois Field

GSM	Global System for Mobile communications
ICC	International Chamber of Commerce
ICSA	International Computer Security Association
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineer Task Force
IFP	Integer Factorization Problem
ILPF	Internet Law & Policy Forum
IP	Internet Protocol
IPSEC	IP Security Protocol
ISAKMP	Internet Security Association Key Management Protocol
ISO	International Standards Organization
ISO-IEC	International Standards Organization - International Electrotechnical Commission
JCA	Java Card Assembly
JCRE	Java Card Runtime Environment
JCVM	Java Card Virtual Machine
JCWDE	Java Card Workstation Development Environment
JVM	Java Virtual Machine
MIPS	Million Instructions Per Second
MISPC	Minimum Interoperability Specification
MOV	Menezes-Okomoto-Vanstone attack
NAF	Nonadjacent Form
NFS	Number Field Sieve
NIST	National Institute of Standards
OECD	Organisation for Economic Co-operation and Development
ONB	Optimal Normal Base
OTP	Open Trading Protocol
PDA	Personal Digital Assistant
PKCS	Public Key Cryptography System
PROM	Programmable Read Only Memory
QS	Quadratic Sieve
RAM	Random Access Memory
RIPEMD-160	160-bit cryptographic hash function
ROM	Read Only Memory
RSA	Rivest-Shamir-Adleman public key encryption scheme
SET	Secure Electronic Transaction
SHA-1	Secure Hash Algorithm version 1
SPA	Simple Power Analysis
TOF	Trapdoor One-way Function
UN/CEFACT	United Nations Centre for Trade Facilitation and Electronic Business
UNCITRAL	United Nations Commission on International Trade Law
WAP	Wireless Application Protocol
WTLS	Wireless Transport Layer Security

Mathematical Notations

K^*	multiplicative group for field K
$\overline{K}$	algebraic closure of field K
$\mathbb{F}_q$	finite field with q elements
$\mathbb{Z}_p$	ring for any positive integer p
$\mathbb{Z}_p^*$	set of residues modulo p that are relatively prime to p
$\langle g \rangle$	cyclic group generated by g
E	elliptic curve
$\#E$	the order of curve E
$[m]P$	multiplication-by-m map applied to the point P
$\mathcal{O}$	point at infinity (on an elliptic curve)
$\mathcal{M}$	finite set of possible plaintexts
$\mathcal{C}$	finite set of possible ciphertexts
$\mathcal{K}$	keyspace, finite set of possible keys
$\mathcal{A}$	finite set of possible signatures
e_K	encryption algorithm
d_K	decryption algorithm
sig_K	signing algorithm
ver_K	signature verification algorithm
Δ	discriminant of the Weierstrass equation
$j(E)$	j-invariant of curve E (if $\Delta \neq 0$)
$\|A\|$	the cardinality of the set A, i.e. the number of elements in A
$\phi(n)$	Euler phi function of n, the number of integers in $\mathbb{Z}_n$ that are relatively prime to n.

Chapter 1

Introduction

The topic of this Master's thesis is to clarify how well elliptic curve cryptography (ECC) is suited to be used on smart cards, and how it differs from the widely used RSA cryptosystem.

Elliptic curve cryptography is a current topic because there is a great need to enable digital signatures and as strong encryption as with using, say 1024-bit RSA, to the different equipment using smart cards (i.e. mobile phones, thinkpads). At the same time there is a desire to use shorter keys than with 1024-bit RSA, because smart cards have restricted memory space. In addition, several sources claim that elliptic curve operations (signing, signature verification, encryption, and decryption) would be faster than with RSA of same strength.

Elliptic curve cryptography is discussed in [5, 28, 32]. Smart cards and cryptography are handled in [21, 23, 42].

This topic is interesting because the security of elliptic curve cryptosystems has been studied for some time, but there are not yet many commercial applications on the market which use elliptic curve cryptography. Still, if the claims are true that required key sizes with ECC are much smaller than with RSA, and the ECC operations are faster than RSA's operations, ECC might be very suitable to be used in smart cards.

1.1 Introduction to Thesis

The main objective of this Master's Thesis is to clarify whether the elliptic curve cryptography is better suited to be used with smart cards than the nowadays widely used 1024-bit RSA.

To reach this objective, I have studied how well elliptic curve based cryptography can be applied to sign and verify messages, and to encrypt and decrypt messages on smart cards. To facilitate the studies I have implemented the Nyberg-Rueppel signing and verification algorithms, and the Menezes-Vanstone encrypting and decrypting algorithms for smart cards in Java, and have carried out some tests on smart card emulator to compare the performance of the operations between elliptic curve and RSA algorithms. In addition I have compared the space required to store the private keys on smart cards when using elliptic curve and RSA cryptosystems, and the security which these two methods provide.

This work is limited to cover the description and implementation of the Nyberg-Rueppel and Menezes-Vanstone elliptic curve algorithms. The RSA algorithms for signing and encryption were already implemented and are only briefly described in this thesis. Furthermore, I describe the testing methods and the test results concerning the performance of the operations, security, and the space required to store the keys. On the basis of the test results I have made some analysis and estimates how well elliptic curve cryptography works on smart cards.

This thesis has the following structure:

Chapter 2 explains why encryption and digital signatures are used, and describes the two main cryptosystem types: private key cryptosystems and public key cryptosystems, including digital signatures. After these, I take a look at the present situation in digital signature legislation. Then, I describe what kinds of encryption and digital signatures are used today.

In Chapter 3 I give a brief description of smart cards, including the components of a smart card, and give a glance to the future of smart cards.

In Chapter 4 I go through the basic mathematical theory behind the elliptic curve cryptosystem and describe the Menezes-Vanstone encryption algorithm and the Nyberg-Rueppel signature algorithm. After these, I describe the standardization of elliptic curve cryptosystems.

In Chapter 5 I first present the basic mathematical theory behind the RSA cryptosystem. Then I compare the elliptic curve cryptosystem and RSA cryptosystem in terms of security, efficiency, and space requirements, based on published research.

Chapter 6 describes the implementation of RSA and elliptic curve cryptosystems. I also present the smart card emulator, for which the implementations are made.

Chapter 7 describes the test methods used for testing the implementation. It also describes the test parameters and the environment used for testing.

Chapter 8 presents test results obtained with RSA and elliptic curve cryptosystem implementations.

Chapter 9 contains analysis of the test results. The analysis concentrates on comparing the performance, security, and space requirements between the implemented RSA and elliptic curve cryptosystems.

In Chapter 10 I summarize the conclusions gained in this thesis, and give an opinion whether the elliptic curve cryptosystem is better suited to be used on smart cards than the RSA cryptosystem.

Chapter 2

Encryption and Digital Signatures

Today global electronic commerce in the Internet is common. To enable profitable and legal trading, confidentiality, integrity, and non-repudiability of the associated messages is necessary. In order to achieve this, cryptographic systems must play a major role in electronic commerce and, within the cryptography realm, public key cryptography has emerged over twenty years as the key element. However, public key cryptography requires a public key infrastructure to exist for it to become useful and for the associated algorithms to be proven and accepted, particularly where those algorithms are used for digital signature purposes [7].

Electronic commerce has advantages for both the merchant and the customer. It also brings new challenges that traditional stores do not face [23]:

> **Authentication.** A merchant must know the identity of the customer. For some kinds of businesses it is not sufficient that the customer authenticates herself by the use of a password. In these cases an electronic version of today's identity or credit card is required. The recipient of a message or an order should know the identity of the sender and should also be sure that the data wasn't altered during its transmission. These challenges, the authentication of the user and the integrity of the sent messages, are met using various cryptographic methods.
>
> **Non-repudiation.** It is often necessary to assert that a particular person sent an order or message and that no other person could possibly have sent it. In traditional business the personal hand-written signature is used to assert this, in cases of high importance combined with a witness of the signing act. In electronic commerce, this challenge is met using digital signatures based on public key cryptography.
>
> **Privacy.** The exchange of data between the merchant and the customer in most cases should be kept secret. No unauthorized party should be able to read or copy such a communication. This challenge, confidentiality, is met using encryption.

Payment. Another issue for electronic commerce is the method to pay the goods or services. Today, electronic shopping malls usually require the customer to enter her credit card number. This is not ideal for two reasons: First, the customer must trust that her credit card information is kept in confidence and not abused. Second, credit card transactions incur a cost that might be out of proportion for low cost goods like individual newspaper articles. A replacement for cash is desirable in these cases.

A public key cryptosystem together with smart cards can answer to all these challenges. Smart cards can be used to securely store secret keys and perform the secret key cryptographic operations, namely signing and decryption. Smart cards can also be used as an electronic purse to store digital money.

In the following I describe the two basic types of cryptosystems, the private key and public key cryptosystems. After these, I describe the notion of digital signatures and discuss the legislation of digital signatures.

Private Key Cryptosystems

The fundamental goal of cryptography has historically been to achieve privacy, i.e. to enable two people, Alice and Bob, to send each other messages over an insecure channel in such a way that only the intended recipient can read the message. This objective has traditionally been met by using private key cryptosystems [32].

The private key cryptosystem can be defined as follows [32]. Let $\mathcal{M}$ denote the set of all possible plaintext messages, $\mathcal{C}$ the set of all possible ciphertext messages (encrypted messages), and $\mathcal{K}$ the set of all possible keys. A private key cryptosystem consists of a family of pairs of functions $e_k : \mathcal{M} \to \mathcal{C}$, $d_k : \mathcal{C} \to \mathcal{M}$, $k \in \mathcal{K}$, such that $d_k(e_k(m)) = m$ for all $m \in \mathcal{M}$ and $k \in \mathcal{K}$.

To use such a system, Alice and Bob initially agree upon a secret key $k \in \mathcal{K}$. They may do this, for example, by physically meeting or by using the services of a trusted courier. If at a later time Alice wishes to send Bob a message $m \in \mathcal{M}$, she sends the ciphertext $c = e_k(m)$ to Bob, from which Bob can recover m by applying the decryption function d_k.

Clearly, some desirable properties of a private key cryptosystem are that the functions e_k and d_k should be easy to apply, and that it should be infeasible for an eavesdropper who sees c to determine the message m (or the key k). The latter property should hold even if the opponent knows everything about the cryptosystem being used (except, of course, the particular key chosen).

Although private key cryptography is adequate for many applications, it has the following disadvantages which make it unsuitable for use in certain applications [32]:

Key distribution problem. As described above, the two users have to select a key in secret before they can start communications over an insecure channel. A secure channel for selecting a key may not be available.

Key management problem. In a network of n users, every pair of users must share a secret key, for a total of $n(n-1)/2$ keys. If n is large, then the number of keys becomes unmanageable.

No signatures possible. A digital signature is an electronic analogue of a hand-written signature. This means that a digital signature allows the receiver of a message to convince any third party that the message in fact originated from the sender. In a private key cryptosystem, Alice and Bob have the same capabilities for encryption and decryption, and thus Bob cannot convince a third party that a message he received from Alice in fact originated from Alice.

Public Key Cryptosystems

The basic idea that led to public key algorithms was that keys could come in pairs of an encryption and decryption key and that it could be impossible to compute one key given the other. This concept was invented by Whitfield Diffie and Martin Hellman in 1976 [17] and independently by Ralph Merkle in 1978 [35].

Since then, many public key algorithms have been proposed, most of them insecure or impractical. All public key algorithms are very slow compared to secret key algorithms [23]. The RSA algorithm takes about 1000 times longer than the popular private key encryption algorithm, DES, when implemented in hardware, and 100 times longer in software to encrypt the same amount of data.

However, public key algorithms have a big advantage when used for ensuring privacy of communication [23]: Public key algorithms use different keys for signing and decryption, and for encryption and signature verification. The private key may only be known to its owner and must be kept in secret. It may be used for generation of digital signatures or for decrypting private information encrypted with the public key. The public key may be used for verifying digital signatures or for encrypting information. It needs not to be kept secret, because it is infeasible to compute the private key from a given public key. Thus, users can post their public key to a directory, where everybody who wants to send an encrypted message or verify an signature can look it up. Each entity in the network only needs to store its own private key and a public directory can store the public keys of all entities, which is practical even in large networks.

To construct a public key cryptosystem, we need a family $f_k : \mathcal{M} \to \mathcal{C}, k \in \mathcal{K}$, of trapdoor one-way functions (TOF) [32]. The family should have the property that for each $k \in \mathcal{K}$, the trapdoor, denoted $t(k)$, is easy to obtain. In addition, for each $k \in \mathcal{K}$, it must be possible to describe an efficient algorithm for computing f_k, such that it is infeasible to recover k (and thus $t(k)$) from this description.

A one-way function $f : \mathcal{M} \to \mathcal{C}$ is an invertible function, such that for each $m \in \mathcal{M}$ it is easy to compute $f(m)$, while for most $c \in \mathcal{C}$ it is hard to compute $f^{-1}(c)$ [32]. The term hard will usually mean computationally infeasible, i.e. infeasible using the best known algorithms and best available computer technology. At present, it is not known whether one-way functions truly exist, although there are several candidate one-way functions.

A one-way function $f : \mathcal{M} \to \mathcal{C}$ is said to be a trapdoor one-way function (TOF) if there is some extra information with which f can be efficiently inverted [32]. This extra information is called the trapdoor.

Given such a family of TOFs, each user selects a random $a \in \mathcal{K}$ and publishes the algorithm E_a for computing f_a in a public directory. E_a is the user's (lets call her Alice) public key, while the trapdoor $t(a)$, which is used to invert f_a, is Alice's private key. To send a message $m \in \mathcal{M}$ to Alice, user Bob simply looks up Alice's public key E_a in the directory and transmits $f_a(m)$ to Alice. Since Alice is the only person who has the ability to invert f_a, only Alice can recover the message m. There is no need to exchange keys in secret prior to communicating and there is only one key pair associated with each user. Public key cryptosystems thus overcome the key distribution and management problems inherent with private key systems [32].

Digital Signatures

If we order goods or services, we often have to sign a contract on paper to testify that we placed the order and are liable to pay for it. If we make the same deal over the network instead, we need the electronic equivalent of signing on paper: a digital signature. Such a digital signature must guarantee that a person cannot repudiate her order or statement.

The different methods for digital signing are based on public key cryptography [23]. The signing person has a private key, which cannot be accessed or used by anyone else. A second key, which is associated to the private key, is known to the public. Only the unique owner of the private key can sign an order or statement, while everybody can check the signature using the corresponding public key.

With a conventional signature, a signature is physically part of the document being signed. However, a digital signature can not be physically attached to the message that is signed, so the algorithm that is used must somehow bind the signature to the message [55].

To bind the digital signature to the message, we need to assume that $\mathcal{M} = \mathcal{C}$. If Alice wishes to send Bob a signed message m, she simply sends Bob the quantity $s = {f_a}^{-1}(m)$ together with m. Now, anyone can verify that $m = f_a(s)$ by using Alice's public key E_a, but only Alice could have computed s. Hence the quantity s serves as Alice's signature for the message m [32]. To keep the size of a signature relatively small, a one-way hash function is usually used to create a hash m' from a original message m, and sign m' instead of m.

For digital signatures it is crucial that the private key remains absolutely private [23]. If any person could copy another person's private key, the digital signature would no longer be unique to the owner. Therefore the private key has to be stored in a very secure place where nobody could possibly copy it and where nobody but the owner can use it.

A conventional signature is verified by comparing it to other, authentic signatures. For example, when someone signs a credit card purchase, the merchant is supposed to compare the signature on the sales slip to the signature on the back of the credit card in order to verify the signature. This is not a very secure method as it is relatively easy to forge someone else's signature. Digital signatures, on the other hand, can be verified using a publicly known verification algorithm [55]. Thus, anyone can verify a digital signature.

Another fundamental difference between conventional and digital signatures is that a copy of a signed digital message is identical to the original. On the other hand, a copy of a signed paper document can usually be distinguished from an original [55]. This feature means that care must be taken to prevent a signed digital message from being reused. For example, if Bob signs a digital message authorizing Alice to withdraw $500 from his bank account, he only wants Alice to be able to do so once. So the message itself should contain information, such as a date, that prevents it from being reused.

A signature scheme consists of two components: a signing algorithm and a verification algorithm [55]. Bob can sign a message m using a secret signing algorithm sig. The resulting signature $sig(m)$ can subsequently be verified using a public verification algorithm ver. Given a pair (m, s), where $s = sig(m)$, the verification algorithm returns an answer "true" or "false" depending on whether the signature is authentic. Usually in practice the secrecy of the signature algorithm is achieved through a secret key rather than a secret algorithm.

Legislation

The most secure place to store a private key is a cryptographic hardware unit, often called crypto token [23]. A smart card can be considered to be the most convenient and most portable cryptographic hardware unit. Most smart cards are able to perform cryptographic operations inside the card. At the same time they do not provide any function to export the private key to the outside. Legislation in some European countries requires that the private key must be generated on the smart card. In combination with the requirement that there must be no way to export the private key, this makes it highly unlikely that any additional copies of a private key could exist.

At the moment, all over the world people are working on a model digital signature law. The following organizations are currently developing legislation for digital signatures [59]: United Nations (United Nations Commission on Internal Trade Law (UNCITRAL) and UN/CEFACT), OECD (Organization for Economic Co-operation and Development), ICC (International Chamber of Commerce), European Union, APEC (Asia Pacific Economic Co-operation), ETSI (European Telecommunications Standards Institute), and ILPF (Internet Law & Policy Forum).

In addition, many countries in America, Asia, and Europe have started to legislate digital signatures [59]. In America at least Argentina, Brazil, Canada, Colombia, Mexico, and USA are developing laws for digital signatures. In Asia Hong Kong, India, Japan, Malaysia, Singapore, South Korea, and Thailand are in the same process, and in Europe Austria, Belgium, Czech Republic, Denmark, Estonia, Finland, France, Germany, Ireland, Luxemburg, Netherlands, Norway, Spain, Sweden, Switzerland, and United Kingdom are legislating digital signatures.

2.1 Encryption and Digital Signatures Today

Today, public key cryptosystems may be broadly characterized as belonging to one of three mathematical areas. These are:

Integer factorization, such as the RSA scheme,

Discrete logarithm schemes, such as the USA's DSA, or Digital Signature Algorithm, and

Elliptic curve cryptosystems, or more accurately the elliptic curve discrete logarithm scheme.

The last two schemes should be considered to be related since both their security depends on the difficulty of solving the discrete logarithm problem [7].

At present, the most commonly used public key cryptosystem is the Rivest-Shamir-Adleman encryption scheme, RSA (the recommended key size for it is at least 1024 bits), which can be used to create digital signatures and to encrypt messages. The most common private key cryptosystem is probably Data Encryption Standard, DES, but because private key cryptosystems can not be used to create digital signatures this thesis ignores them.

It is believed that ECC will become even more popular than RSA, because the key sizes needed with ECC are much smaller (about 163 bits) than with RSA. The key size affects the speed of the cryptographic operations, making them faster than RSA operations.

Commercial elliptic curve implementations are available at least from Certicom, and RSA Data Security has announced plans to support elliptic curve methods in a developer's kit.

One problem that may effect the widespread acceptance of elliptic curve cryptography as a part of any public key infrastructure is the number of patents held and pending on elliptic curve implementation techniques, routines, algorithms, and protocols [7]. In addition, the ECC uses a complex and sophisticated form of mathematics to set up secure curves, which is not readily understood by information technology professionals responsible for implementing security systems based on public key cryptography. In contrast, the RSA scheme is not that difficult to understand given normal high-school level mathematics.

Chapter 3

Smart Cards

Today smart cards are used for many different purposes in daily life. The smart card can be a phone card, a card carrying our health insurance information, or an electronic purse [23]. The latter allows us to store digital money and to use this money later to pay a ticket or buy a drink from a vending machine. In Finland the most popular way to use smart cards is in the GSM phones, but also banks use them as electronic purses (Osuuspankki), optician stores (i.e. Instrumentarium) use them to store customer's information (such as powers of lenses), and Väestörekisteriliitto has introduced the Finnish National Electronic Identity (FINEID) cards, etc.

The smart card itself is a device which is able to store data and execute commands. It is a single-chip microcomputer with a size of 25 mm^2 at most. This microcomputer is mounted on a plastic card of the size of a standard credit card (54 mm $\times$ 85.6 mm) [23]. Data transfer can take place either via the contacts on the card's surface, or without contact through electromagnetic fields [42].

The success of the smart cards in Europe begun in the early eighties, between 1982 and 1984 when Carte Bancaire (the French Bank Card Group) had the first smart card pilot running [23]. Together with Bull, Philips and Schlumberger, Carte Bancaire launched trials in the French cities of Blois, Caen, and Lyon. The trials were a great success. Following these trials, French banks launched the use of smart cards for banking. This was the first mass rollout of smart cards in the banking industry.

One of the most important advantages of smart cards consists of the fact that their stored data can be protected against unauthorized access and tampering [42]. As access to data takes place only via a serial interface supervised by the operating system and by a security logic system, it is possible to write confidential data to the card which can never be read from outside. This secret data can then only be processed internally by the chip's arithmetic unit. In principle, the memory functions of writing, erasing and reading can be controlled both by hardware and by software, and be linked to specific conditions. This allows the construction of numerous security mechanisms, which can also be tailored to the special demands of the relevant application.

The fundamental characteristics and functions of smart cards are laid down in the 7816 series of ISO standards.

In the following, I describe the components of a smart card chip and attempt to sketch the future of smart cards.

3.1 Smart Card Chip and its Components

The most important characteristic of a smart card is that it contains a computer with a CPU and a memory. Today's smart cards have approximately the same computing power as the first IBM PC [23].

The chip of a smart card (see Figure 3.1) consists of a microprocessor, ROM (Read Only Memory), EEPROM (Electrical Erasable Programmable Read Only Memory), and RAM (Random Access Memory) [23]. An EEPROM requires a larger surface than a PROM of the same size and this makes EEPROM more expensive and lets EEPROM size become an important factor for the price of a smart card.

At present, most smart cards have an inexpensive 8-bit microprocessor, but the high-end cards can have a 16-bit or 32-bit processor. An optional cryptographic coprocessor increases the performance of cryptographic operations [23].

By performing signature and decryption operations on the card itself, the user's private key never needs to leave the card. The information stored in the ROM is written during production. It contains the card operating system and might also contain some applications. The EEPROM is used for permanent storage of data. Even if the smart card is unpowered, the EEPROM still keeps the data. Some smart cards also allow storing additional application code or application specific commands in the EEPROM. The RAM is the transient memory of the card and keeps the data only as long as the card is powered.

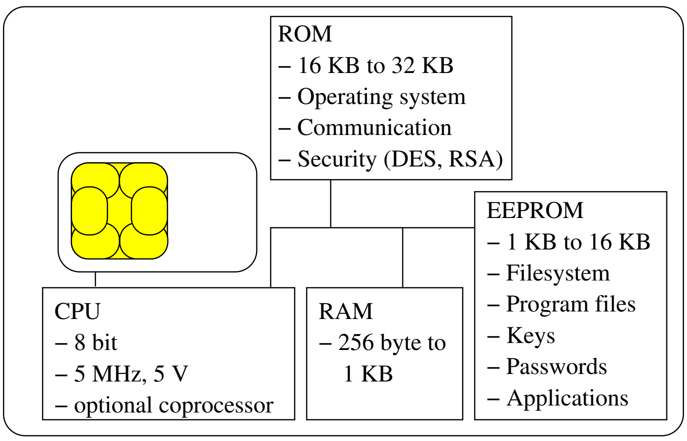

Figure 3.1: Example of a smart card chip and its components

3.2 Future of Smart Cards

Since the eighties, the smart card industry has grown. The smart card is the ultimate device for carrying data, it is considered secure, and it has a familiar shape, that of the widely used and known credit card [23].

The key characteristics of smart cards in today's world are security, ease of use, mobility, and multi-functionality, and they are thus likely to become the next major IT platform [23].

Many smart card applications are today being implemented - from citizen ID cards (e.g. FINEID), to health cards, electronic purse cards, and network access cards [23]. It is widely accepted that nearly all organizations will sooner or later need to assimilate smart cards into their routine business processes.

Today, there are several different wireless terminals on the market, with which the consumers can have Internet access. As a result, personal computers will begin to fade as the primary point of access to the Internet. They will not go away, but cellular phones, set-top boxes, video games, palm pilots, and even cars may outnumber them as network-connected consumer devices.

When these kind of wireless terminals, which need the user to authenticate themselves before use, are getting more common, the need for some kind of authentication method other than typing in passwords becomes apparent. The smart cards might offer an easier way to identify users. When using a GSM phone, a smart card chip is already hidden inside your handset to provide strong authentication to the network operator. If you watch satellite TV, chances are that your set-top box uses a smart card to hold your subscription rights securely. And maybe in a couple of years, your car will recognize you by your smart ignition key, which will hold your preferences for seat and mirror adjustments and favorite radio stations.

However, the growing use of smart cards may have some downsides, as well. When people are constantly authenticating themselves, the monitoring of individuals becomes very easy, and this can lead to some scary scenarios.

Chapter 4

Elliptic Curve Cryptography

While the 20-year history of public key cryptography has seen a diverse range of proposals for candidate hard problems, only two have stood the test of time. These problems are known as the discrete logarithm problem over a finite field and integer factorization [44, 52].

In 1985, Neal Koblitz [28] and V.S. Miller [36] independently proposed using elliptic curves for public key cryptosystems. They did not invent a new cryptographic algorithm with elliptic curves over finite fields, but they implemented existing algorithms, like Diffie-Hellman, using elliptic curves [49].

Elliptic curves are rich mathematical structures which have shown themselves to be useful in a range of applications including primality testing and integer factorization [24, 32]. One potential use of elliptic curves is in the definition of public key cryptosystems that are close analogues of existing schemes [44]. In this way, variants of existing schemes can be devised so that they rely their security on a different underlying hard problem.

In the following I describe the basic mathematics behind elliptic curve cryptography. I start with Weierstrass equation and definition of an elliptic curve. Then I define non-singular curves and describe different field types and the arithmetic operations on those fields. After that I present the elliptic curve discrete logarithm problem and the Menezes-Vanstone and the Nyberg-Rueppel algorithms. The rest of the chapter presents the current situation in elliptic curve cryptosystem standardization.

4.1 Weierstrass Equation and Elliptic Curves

Let $\mathbb{F}_q$ denote the finite field containing q elements, where q is a prime power. If K is a field, let $\overline{K}$ denote its algebraic closure. (If $K = \mathbb{F}_q$ then $\overline{K} = \bigcup_{m\geqslant 1}\mathbb{F}_{q^m}$.) The projective plane $P^2(K)$ over K is the set of equivalence classes of the relation $\sim$ acting on $K^3 \setminus \{0,0,0\}$, where $(x_1, y_1, z_1) \sim (x_2, y_2, z_2)$ if and only if there exists $u \in K^*$ such that $x_1 = ux_2, y_1 = uy_2$, and $z_1 = uz_2$. We denote the equivalence class containing (x, y, z) by $(x : y : z)$. A Weierstrass equation is a homogeneous equation of degree 3 of the form

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

where $a_1, a_2, a_3, a_4, a_6 \in \overline{K}$. The Weierstrass equation is said to be smooth, or non-singular, if for all projective points $P = (X : Y : Z) \in P^2(\overline{K})$ satisfying

$$F(X, Y, Z) = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3 = 0$$

at least one of the partial derivatives $\frac{\partial F}{\partial X}, \frac{\partial F}{\partial Y}, \frac{\partial F}{\partial Z}$ is non-zero at P. If all three partial derivatives vanish at some point P, then P is called a singular point, and the Weierstrass equation is said to be singular [5, 32].

An elliptic curve E is the set of all solutions in $P^2(\overline{K})$ of a smooth Weierstrass equation. There is exactly one point in E with Z-coordinate is equal to 0, namely $(0 : 1 : 0)$. This point is the point at infinity and it is denoted by $\mathcal{O}$ [5, 32].

For convenience, the Weierstrass equation for elliptic curves can be written using non-homogeneous (affine) coordinates, where $x = X/Z$, $y = Y/Z$,

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{4.1}$$

An elliptic curve E is then the set of solutions to equation above in the affine plane $A^2(\overline{K}) = \overline{K} \times \overline{K}$, together with the extra point at infinity $\mathcal{O}$ [5, 32].

4.2 Discriminant and j-invariant

Let E be a curve given by a non-homogeneous Weierstrass equation (4.1). Define the quantities

$$\begin{aligned}
d_2 &= a_1^2 + 4a_2 \\
d_4 &= 2a_4 + a_1a_3 \\
d_6 &= a_3^2 + 4a_6 \\
d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2 \\
c_4 &= d_2^2 - 24d_4 \\
\Delta &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\
j(E) &= c_4^3/\Delta
\end{aligned}$$

The quantity Δ is called the discriminant of the Weierstrass equation, while $j(E)$ is called the j-invariant of E if $\Delta \neq 0$. The curve E is non-singular, if and only if $\Delta \neq 0$ [5].

4.3 Fields

When implementing an elliptic curve cryptosystem, an important consideration is how to implement the underlying field arithmetic. There are two possible field types to choose: Fields of odd characteristic and fields of characteristic two.

Two questions of particular importance are whether to use even or odd characteristic fields and secondly, whether to restrict implementation to fields of a special type, for efficiency, or to support any type of finite field [5].

Fields of Odd Characteristic

Fields of odd characteristic are implementations of arithmetic in $\mathbb{F}_p$, where p is a large prime. Field elements will be represented as integers in the range of $0, 1, ..., p-1$, with the usual arithmetic modulo p [5].

There are four standard arithmetic operations needed in $\mathbb{F}_p$, namely addition, subtraction, multiplication and division. It is, however, the last two of these (and particularly the last) which produce the most challenge [5]. The arithmetic operations for point addition and doubling are described later in this chapter.

If an elliptic curve E is defined over a field K whose characteristic is neither 2 nor 3, then the Weierstrass equation for the curve can be simplified considerably:

$$E : y^2 = x^3 + ax + b,\ a, b \in K.$$

That is, we can select a Weierstrass equation for E so that $a_1 = a_2 = a_3 = 0$ [32].

Fields of Characteristic Two

We now go through the case of arithmetic in $\mathbb{F}_{2^n}$, where $n \geq 1$. In this case, the expression for the j-invariant reduces to $j(E) = a_1^{12}/\Delta$. In fields of characteristic two, the condition $j(E) = 0$, i.e. $a_1 = 0$, is equivalent to the curve being supersingular [5, 32]. This very special type of curve is avoided in cryptography because of the MOV attack (see Chapter 5.2).

Field elements in fields of characteristic two are represented as binary vectors of dimension n, relative to given basis $(\alpha_0, \alpha_1, ..., \alpha_{n-1})$ of $\mathbb{F}_{2^n}$ as a linear space over $\mathbb{F}_2$. Field addition and subtraction are implemented as component-wise exclusive-or, while the implementations of multiplication and inversion depend on the basis chosen [5, 32]. The arithmetic operations for point addition and doubling are described later in this chapter.

Finite fields of characteristic two are attractive to implementers because of their carry free arithmetic, and the availability of different equivalent representations of the field, which can be adapted and optimized for the computational environment at hand [5, 32].

If an elliptic curve is defined over a field K which is of characteristic two, the Weierstrass equation for the curve can be simplified considerably [5, 32]. The simplification depends on the j -invariant of E, $j(E)$, as follows:

If $j(E) \neq 0$, transforms E to the curve

$$E : y^2 + xy = x^3 + a_2x^2 + a_6$$

Else if $j(E) = 0$ (i.e. E is supersingular), transforms E to the curve

$$E : y^2 + a_3y = x^3 + a_4x + a_6$$

The three different bases, which can be used to implement fields of characteristic two are polynomial base, normal base, and subfield base [5, 32]. They are briefly described below.

Polynomial base. A polynomial (or standard) base is of the form $(1, \alpha, \alpha^2, ..., \alpha^{n-1})$, where α is a root of an irreducible polynomial $f(x)$ of degree n over $\mathbb{F}_2$.The field is then realized as $\mathbb{F}_2[x]/ \langle f(x)\rangle$, and the arithmetic is that of polynomials of degree at most $n-1$, modulo $f(x)$. $\langle f(x)\rangle$ is the cyclic group generated by $f(x)$ [5].

Normal base. A normal base of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ has the form $(\alpha, \alpha^2, \alpha^{2^2}, ..., \alpha^{2^{n-1}})$ for some $\alpha \in \mathbb{F}_{2^n}$. It is known that such bases exist for all $n \geq 1$. Normal bases are useful mostly in hardware implementations. First, the field squaring operation is trivial in normal base representations, as it amounts to just cyclic shifting of the binary vector representing the input operand. More importantly, normal bases allow for the design of efficient bit-serial multipliers [5].

The existence of optimal normal bases (ONB) has been completely characterized in [37] and [20]. In particular, an ONB of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$ exists if and only if one of the following conditions holds [5, 32]:

1. $n+1$ is prime, and 2 is primitive in $\mathbb{F}_{n+1}$; then the n non-trivial $(n+1)$st roots of unity form an ONB of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$, called a Type I ONB.

2. $2n+1$ is prime, and either

   (a) 2 is primitive in $\mathbb{F}_{2n+1}$ or

   (b) $2n+1 \equiv 3(\text{mod } 4)$ and the multiplicative order of 2 in $\mathbb{F}_{2n+1}$ is n; that is 2 generates the quadratic residues in $\mathbb{F}_{2n+1}$;

   then, $\alpha = \gamma + \gamma^{-1}$ generates an ONB of $\mathbb{F}_{2^n}$ over $\mathbb{F}_2$, where γ is a primitive $(2n+1)$st root of unity; this is called a Type II ONB.

The bit-serial multipliers that can be very effective for ONBs in hardware do not always map nicely to efficient software implementations, as single bit operations are expensive in the latter. It turns out, that by applying simple permutations, operations on ONB representations of both Types I and II can be handled through polynomial arithmetic, in a manner similar to the case of standard bases [5].

Subfield base. When $n = n_1 n_2$, we can regard $\mathbb{F}_{2^n}$ as an extension of degree n_2 of $\mathbb{F}_{2^{n_1}}$, and represent elements of $\mathbb{F}_{2^n}$ using a base of the form $\alpha_i \beta_j : 0 \le i \le n_1$, $0 \le j \le n_2$, where $\beta_0, \beta_1, ..., \beta_{n_2-1}$ form a base of $\mathbb{F}_{2^n}$ over $\mathbb{F}_{2^{n_1}}$, and $\alpha_0, \alpha_1, ..., \alpha_{n_1-1}$ form a base of $\mathbb{F}_{2^{n_1}}$ over $\mathbb{F}_2$ [5]. Thus, arithmetic can be done in two stages, with an outer section doing operations on elements of $\mathbb{F}_{2^n}$ as vectors of symbols from $\mathbb{F}_{2^{n_1}}$; and inner section performing the operations on the symbols as binary words. Any combination of bases can be used, e.g. normal base for the outer section, and polynomial base for the inner one.

The subfield representation is particularly advantageous when n_1 is large enough so that n_2 is small, but n_1 is still small enough so that symbol operations can be made very fast in the computational environment at hand, e.g. by implementing the $\mathbb{F}_{2^{n-1}}$ arithmetic through look-up tables [5]. Values of n_1 between 4 and 16 are typical.

4.4 Arithmetic

Group Law

The points on an elliptic curve form an Abelian group under a certain addition. Let E be an elliptic curve given by the Weierstrass equation (4.1). The additional rules for points P and Q are as follows [32]:

For all P, $Q \in E$,

1. $\mathcal{O} + P = P$ and $P + \mathcal{O} = P$. That is, $\mathcal{O}$ is identity element.
2. $-\mathcal{O} = \mathcal{O}$.
3. If $P = (x_1, x_2) \ne \mathcal{O}$, then $-P = (x_1, -y_1 - a_1 x_1 - a_3)$.
4. $Q = -P$, then $P + Q = \mathcal{O}$.
5. If $P \ne \mathcal{O}, Q \ne \mathcal{O}, Q \ne -P$, then let R be the third point of intersection (counting multiplicities) of either the line which intersects P and Q if $P \ne Q$, or the tangent line to the curve at P if $P = Q$, with the curve. Then $P + Q = -R$.

Point Addition

Let P and Q be two distinct rational points on E, $E : y^2 = x^3 + ax + b, a, b \in K$. The straight line joining P and Q must intersect the curve at one further point, R, since we are intersecting a line with a cubic curve. The point R will also be rational since the line, the curve and the points P and Q are themselves all defined over K [5, 32].

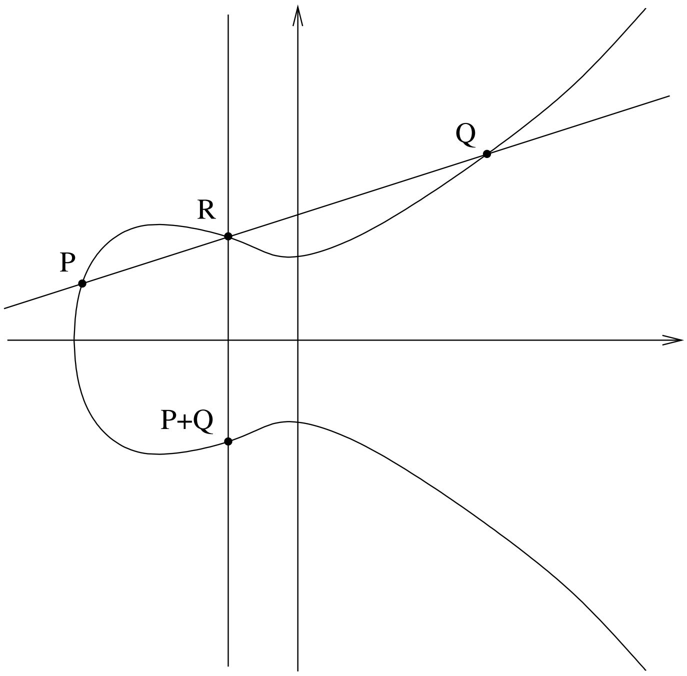

Figure 4.1: Adding two points on an elliptic curve

If we then reflect R in the x-axis, we obtain another rational point which we shall call $P + Q$ (see Figure 4.1) [5, 32].

There are different addition formulas for fields of characteristic $p > 3$ and for fields of characteristic two. Curve is defined in fields of characteristic $p > 3$ if $K = \mathbb{F}_q$, where $q = p^n$ for a prime $p > 3$ and an integer $n \geq 1$ [5, 32].

Addition formula for fields of characteristic $p > 3$:

The addition formula for fields of characteristic $p > 3$ is defined as follows [5, 32]. Let $P = (x_1, y_1) \in E$, then $-P = (x_1, -y_1)$. If $Q = (x_2, y_2) \in E$, $Q \neq -P$ and $P \neq Q$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

and

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

Addition formula for fields of characteristic two

The addition formula for fields of characteristic two depends on the j-invariant of E as follows [5, 32].

Addition formula, when $j(E) \neq 0$

Let $P = (x_1, y_1) \in E_1$; then $-P = (x_1, y_1 + x_1)$. If $Q = (x_2, y_2) \in E_1$, $Q \neq -P$ and $P \neq Q$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a_2$$

and

$$y_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + x_3 + y_1$$

Addition formula, when $j(E) = 0$ (i.e. E is supersingular)

Let $P = (x_1, y_1) \in E_2$; then $-P = (x_1, y_1 + a_3)$. If $Q = (x_2, y_2) \in E_2$, $Q \neq -P$ and $P \neq Q$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + x_1 + x_2$$

and

$$y_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + y_1 + a_3$$

Point Doubling

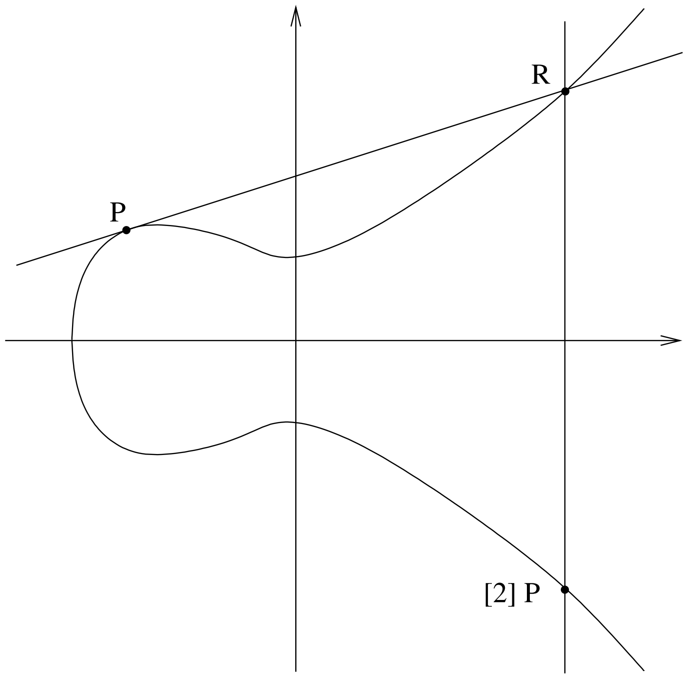

Figure 4.2: Doubling a point on an elliptic curve

To add P to itself, or to double P, we take the tangent to the curve at P. Such a line must intersect E in exactly one other point, say R, as E is defined by a cubic equation. Again we reflect R in the x-axis to obtain a point which we call $[2]P = P + P$ (see Figure 4.2) . If the tangent to the point is vertical, it intersects the curve at the point at infinity and $P + P = \mathcal{O}$, i.e. P is a point of order 2 [5, 32].

There are different doubling formulas for fields of characteristic $p > 3$ and for fields of characteristic two [5, 32].

Doubling formula for fields of characteristic $p > 3$

The doubling formula for fields of characteristic $p > 3$ is defined as follows [5, 32]. Let $P = (x_1, y_1) \in E$, $Q = (x_2, y_2) \in E$, $P = Q$ then $P + Q = (x_3, y_3)$, where

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

and

$$\lambda = \frac{3{x_1}^2 + a}{2y_1}$$

Doubling formula for fields of characteristic two

The doubling formula for fields of characteristic two depends on the j-invariant of E as follows [5, 32].

Doubling formula, when $j(E) \neq 0$

Let $P = (x_1, y_1) \in E_1$, $Q = (x_2, y_2) \in E_1$, and $P = Q$, then $P + Q = (x_3, y_3)$, where

$$x_3 = {x_1}^2 + \frac{a_6}{{x_1}^2}$$

and

$$y_3 = {x_1}^2 + \left(x_1 + \frac{y_1}{x_1}\right) x_3 + x_3$$

Doubling formula, when $j(E) = 0$ (i.e. E is supersingular)

Let $P = (x_1, y_1) \in E_2$, $Q = (x_2, y_2) \in E_2$, and $P = Q$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \frac{{x_1}^4 + {a_4}^2}{{a_3}^2}$$

and

$$y_3 = \left(\frac{{x_1}^2 + a_4}{a_3}\right)(x_1 + x_3) + y_1 + a_3$$

We would like to select a curve and field K so that the number of field operations involved in adding two points and doubling a point are minimized. Curves over $K = \mathbb{F}_{2^n}$ are preferred for the following reasons [10, 32]:

1. The arithmetic in $\mathbb{F}_{2^n}$ is easier to implement in computer hardware than the arithmetic in finite fields of characteristic greater than 2.

2. When using a normal basis representation for the elements of $\mathbb{F}_{2^n}$, squaring a field element becomes a simple cyclic shift of the vector representation, and thus the multiplication count in adding two points is reduced.

3. With curves over $\mathbb{F}_{2^n}$ it is easy to recover the y-coordinate of a point given its x-coordinate plus a single bit of extra information. This is useful in reducing message expansion in the ElGamal cryptosystem.

4. A fourth reason applies to supersingular curves. For supersingular curves over $\mathbb{F}_{2^n}$, the inverse operation in doubling a point can be eliminated by choosing $a_3 = 1$, further reducing the operation count.

5. In general software environments, the use of $\mathbb{F}_{2^n}$ offers significant performance advantages over $\mathbb{F}_p$. This holds true for platforms such as a Sun Sparc station, an HP server, an embedded system, and more importantly, for a low-cost, 8-bit smart card. To achieve equivalent performance with $\mathbb{F}_p$, a crypto coprocessor (a dedicated hardware component for cryptographic processing) is required. In software environments in which an arithmetic processor is already available for modular exponentiation, the performance of $\mathbb{F}_p$ can be improved so that in some cases it exceeds the performance of $\mathbb{F}_{2^n}$. This holds true for platforms such as those using Pentium processors or, in the case of smart cards, those having a crypto coprocessor to accelerate modular arithmetic.

4.5 Elliptic Curve Discrete Logarithm Problem

Now that we have gone through the basic mathematics used with elliptic curve cryptography, we can take a look at the elliptic curve discrete logarithm problem (ECDLP), the basis of elliptic curve cryptography. After ECDLP I describe the Menezes-Vanstone elliptic curve encryption algorithm and the Nyberg-Rueppel elliptic curve signature algorithm.

The elliptic curve discrete logarithm problem can be stated as follows [39]. Fix an elliptic curve. xP represents the point P added to itself x times. Suppose Q is a multiple of P, so that

$$Q = xP$$

for some x. Then the elliptic curve discrete logarithm problem is to determine x given P and Q.

The security of the ECC rests on the difficulty of the elliptic curve discrete logarithm problem. As is the case with the integer factorization problem and the discrete logarithm problem modulo p, no efficient algorithm is known at this time to solve the elliptic curve discrete logarithm problem [39].

One of the advantages of ECC is that the elliptic curve discrete logarithm problem is believed to be harder than both the integer factorization problem and discrete logarithm problem modulo p. This extra difficulty implies that ECC is one of the strongest public key cryptographic system known today [39].

The elliptic curve discrete logarithm problem is relatively easy for a small class of elliptic curves, known as supersingular elliptic curves and also for certain anomalous elliptic curves [39]. In both cases, the weak instances of the problem are easily identified, and an implementation merely checks that the specific instance selected is not one of the class of easy problems.

Basically, elliptic curve cryptography is constructed on similar concepts to those used for discrete logarithm systems, but the discrete logarithm functions are performed on elliptic curves over finite fields [7].

A major factor in accepting ECC is the fact of smaller cryptographic key sizes [7]. With small, electronic commerce and banking type transactions this may be an important consideration in overall system performance.

There are many possible algorithms to use for encryption with elliptic curves. As stated before (in the beginning of the Chapter 4), many discrete logarithm problems can be converted to use elliptic curves. The newest version of IEEE's P1363 standard does not however define any encryption algorithm to use with elliptic curves. In one of the earlier version of P1363 they recommended to use the ElGamal cryptosystem, but now it has been removed from the document. Because there are no recommendations which encryption algorithm to use, I decided to implement the Menezes-Vanstone elliptic curve cryptosystem, which is a variant of the ElGamal cryptosystem.

Only two elliptic curve signature schemes are given in the IEEE P1363 standard: Nyberg-Rueppel and ECDSA. They are similar in overall security. The security of both schemes depends on the order of the base point being a large prime number [45]. Supposedly the Nyberg-Rueppel version is patented, but U.S. patent number 5600725 patents are discrete logarithm versions and not the elliptic curve versions at explicit claim. The Digital Signature Algorithm (DSA) version is not patented at all [45]. I chose to implement the Nyberg-Rueppel signature scheme because Certicom's test results listed in Table 5.5 claim that its operations are faster than ECDSA's.

Both the Menezes-Vanstone elliptic curve cryptosystem and Nyberg-Rueppel signature scheme are described below.

4.6 Menezes-Vanstone Elliptic Curve Cryptosystem

The Menezes-Vanstone elliptic curve cryptosystem is defined as follows [55]. Let E be an elliptic curve defined over $\mathbb{Z}_p$ ($p > 3$ prime), or in $\mathrm{GF}(p^n)$ with $n > 1$, such that E contains a cyclic subgroup H in which the discrete logarithm problem is intractable.

Let $\mathcal{P} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, $\mathcal{C} = E \times \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and define

$$\mathcal{K} = \{(E, \alpha, a, \beta) : \beta = a\alpha\},$$

where $\alpha \in E$. The values α and β are public, and a is secret.

For $K = (E, \alpha, a, \beta)$, for a (secret) random number $k \in \mathbb{Z}_{|H|}$, and for $x = (x_1, x_2) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, define

$$e_K(x, k) = (y_0, y_1, y_2),$$

where

$$y_0 = k\alpha,$$

$$(c_1, c_2) = k\beta,$$

$$y_1 = c_1 x_1 \bmod p,$$

$$y_2 = c_2 x_2 \bmod p.$$

For a ciphertext $y = (y_0, y_1, y_2)$, define

$$d_K(y) = (y_1 c_1^{-1} \bmod p,\ y_2 c_2^{-1} \bmod p),$$

where

$$a y_0 = (c_1, c_2).$$

The Menezes-Vanstone cryptosystem is a more efficient variation of the well-known ElGamal cryptosystem.

There are some practical difficulties in implementing an ElGamal cryptosystem on an elliptic curve [55]. The ElGamal cryptosystem, when implemented in $\mathbb{Z}_p$, has a message expansion factor of two. An elliptic curve implementation has a message expansion factor of (about) four. This happens because there are approximately p plaintexts, but each ciphertext consists of four field elements. A more serious problem is that the plaintext space consists of the points on the curve E, and there is no convenient method known to deterministically generate points on E.

In the Menezes-Vanstone variation of the ElGamal cryptosystem an elliptic curve is used for masking, and plaintexts and ciphertexts are allowed to be arbitrary ordered pairs of (nonzero) field elements (i.e. they are not required to be points on E) [55]. This yields a message expansion factor of two, the same as original ElGamal cryptosystem.

In addition, as mentioned in Chapter 4.4 the message expansion can be reduced by using point compression. That is, the y-coordinate of the point can be recovered given its x-coordinate and a single bit of extra information. This yields a message expansion factor of 1.5.

4.7 Nyberg-Rueppel Signature Scheme

The Nyberg-Rueppel signature scheme is defined as follows [25, 45]. Let E be an elliptic curve defined over $\mathbb{Z}_p$ ($p > 3$ prime) such that E contains a cyclic subgroup H in which the discrete logarithm problem is intractable.

Let $\mathcal{P} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, $\mathcal{C} = E \times \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, and define

$$\mathcal{K} = \{(E, \alpha, a, \beta) : \beta = a\alpha\},$$

where $\alpha \in E$. The values α and β are public, and a is secret.

For $K = (E, \alpha, a, \beta)$, for a (secret) random number $k \in \mathbb{Z}_{|H|}$, and for $x = (x_1, x_2) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$, define

$$sig_K(x, k) = (c, d),$$

where

$$(y_1, y_2) = k\alpha$$
$$c = y_1 + \text{hash}(x) \bmod p$$
$$d = k - ac \bmod p$$

$$ver_K(x, c, d) = \text{true} \Leftrightarrow \text{hash}(x) = e,$$

where

$$(y_1, y_2) = d\alpha + c\beta$$
$$e = c - y_1 \bmod p$$

All signature schemes require a hash of a document that is to be signed. The IEEE's P1363 standard suggests SHA-1, defined by NIST, or RIPEMD-160, defined by the ISO-IEC. The reason to use the hash algorithm is to make it impossible to find a match between the real input and some minorly changed version that would give the same hash value. The problem is considered exceptionally difficult to solve with the above hash algorithms [45].

4.8 Standardization of Elliptic Curve Cryptosystems

Issues that affect the widespread use of a cryptographic system are interoperability, public acceptance, and technical scrutiny. ECC and other cryptographic systems like RSA and DES have addressed these issues through standardization [39].

The international standardization of cryptographic systems, protocols, and interfaces is an important process and is actively supported by the International Computer Security Association, ICSA, consortium membership. Standardization has three main benefits [39]:

1. It allows for interoperability among hardware and software systems from many different vendors.
2. It involves critical review of the security of the systems from a cryptographic standpoint.
3. It permits input into the design of cryptosystems from those who have to implement them in a wide range of environments.

Elliptic curves have been the subject of intensive scrutiny in the mathematical community for over ten years and have been scrutinized in standards organizations since 1995 [39]. This has given implementors the high degree of confidence in their security that could not be achieved through the support of only a few organizations.

An effort to define an acceptable standard for public key cryptography systems was started some years ago by the Institute of the Electrical and Electronics Engineers, IEEE, under the general banner of the P1363 draft Standard. Essentially this standard establishes formats and procedures for three separate public key cryptographic systems covering authentication, integrity, and confidentiality services [7, 25]. Similarly the X9 (banking and finance) standard committee of American National Standards Institute, ANSI, and the International Standards Organization, International Electrotechnical Commission, ISO/IEC SC27, have also begun to consider standards for ECC. In the ANSI case the pertinent draft standards are ANSI X9.25, Elliptic Curve Digital Signature Algorithm (ECDSA) and ANSI X9.63 covering associated key agreement and transport standards [1]. ECC is also included in the newer Internet standards discussions for security at the IP layer (IPSEC - ISAKMP - Oakley). In industry related standards, Secure Electronic Transaction, SET, the industry standard for retail commerce on the Internet, appears to be moving to acknowledge ECC as a digital signature option [7].

List of Existing Elliptic Curve Cryptosystem Standards

The following ECC standards have been initiated:

ANSI X9. The ECC is being drafted into two work items by the ANSI ASC X9 Financial Services: ANSI X9.62, the Elliptic Curve Digital Signature Algorithm (ECDSA), and ANSI X9.63, Elliptic Curve Key Agreement (ECKA) and Transport Protocols (ECTP). ANSI TG-17, The technical Guideline on Mathematical Background for Elliptic Curve Cryptosystems, contains extensive information on elliptic curve arithmetic, including an algorithm for counting the number of points on an elliptic curve [1, 8, 39].

ATM Forum. The ATM Forum Technical Committee's Phase I ATM Security Specification draft document aims to provide security mechanisms for ATM (Asynchronous Transfer Mode) networks. Security services provided include confidentiality, authentication, data integrity, and access control [39]. ECC is one of the systems supported.

Certicom. Certicom has published interesting documents on ECC. ECC in X.509 describes how to use elliptic curve keys within the X.509 framework. It e.g. defines the certificate format and the format of certificate revocation lists. Standards for Efficient Cryptography: SEC 1: Elliptic Curve Cryptography, specifies public key cryptographic schemes based on elliptic curve cryptography. In particular, it specifies signature schemes, encryption schemes, and key agreement schemes. Standards for Efficient Cryptography: SEC 2: Recommended Elliptic Curve Domain Parameters, lists example elliptic curve domain parameters at commonly required security levels [11, 12, 13].

FSTC. FSTC, Financial Services Technology Consortium, is concerned with electronic payment systems and other financial services. This innovative, all-electronic payment and deposit gathering instrument can be initiated from a variety of devices such as personal computers, screen phones, ATMs, or accounting systems. E-Check provides rapid and secure settlement of financial accounts between trading partners over open public or proprietary networks without pre-arrangement by interconnection with existing bank clearing and settlement systems infrastructures. ECC is used to encrypt email messages that transport electronic checks [8].

IEEE P1363. ECC is included in the draft IEEE P1363 standard (Standard Specifications for Public Key Cryptography), which includes encryption, signature, and key-agreement mechanisms. Elliptic curves may be defined either modulo p or over $\mathbb{F}_{2^m}$, the field with 2^m elements, for conformance with the standard [25, 39].

IETF. The OAKLEY Key Determination Protocol of the Internet Engineering Task Force, IETF, describes a key agreement protocol that is a variant of the Diffie-Hellman protocol. It allows for a variety of groups to be used, including elliptic curves. The document makes specific mention of elliptic curve groups over the fields $\mathbb{F}_{2^{155}}$ and $\mathbb{F}_{2^{210}}$ [39].

ISO/IEC. The draft document ISO/IEC 14888, Digital signature with appendix - Part 3: Certificate-based mechanisms, specifies elliptic curve analogues of some ElGamal -like signature algorithms [39].

NIST. NIST, National Institute of Standards, has stated that it expects to extend the specification of its Digital Signature Standard (DSS) to include ECC by incorporating content from X9.62. NIST is also including specifications for ECC in its Minimum Interoperability Specification (MISPC). In addition, NIST has produced a document of recommended elliptic curves for United States' federal government use [8, 38].

OTP 0.9. OTP, Open Trading Protocol, is a framework for encapsulating payment protocols. OTP seeks to provide a secure digital replication of the traditional paper-based methods of trading, buying and selling. The specification provides a unifying framework within which SET (Secure Electronic Transactions), EMV (Europay, MasterCard, Visa), E-Check, and other electronic commerce implementations can successfully interoperate. ECDSA (Elliptic Curve Digital Signature Algorithm) is supported for digital signatures in OTP [8].

SET. The Secure Electronic Transactions standard has been developed for Internet credit card transactions. ECC is being considered as a proposed enhancement to the SET standard for secure Internet commerce. The benefits ECC brings to this important application are currently being evaluated [8].

WAP. WAP, Wireless Application Protocol, provides secure Internet access and other advanced services to digital cellular phones and wireless terminals. The specification introduces a layered architecture that enables applications to scale across a variety of transport options and device types. ECC is incorporated into WAP security layer (Wireless Transport Layer Security, WTLS) specification [8, 62].

Numerous initiatives are under way of protocols that use public key cryptography, public key certificates, and other public key management systems. Most of these standards are being written in an algorithm independent manner so that any recognized public key algorithm can be implemented [39]. This will allow the use of algorithms, such as ECC, in environments where other public key cryptosystems would be impractical.

Chapter 5

Comparison of RSA and Elliptic Curve Cryptosystem

The operational characteristics of RSA versus ECC have been compared by Robshaw and Yin [44]. In their article (written in 1997) they claim that encryption time with ECC will be almost eight times longer than with RSA, but both decryption and signing processes will be some 6 to 7 times faster. These both assume that the necessary parameters and initialization processes have been performed in both cases, e.g. key generation etc.

The findings of Robshaw and Yin are contrasted by the claims of Certicom Corporation who claims that the most efficient implementations of elliptic curve cryptosystems are an order of magnitude (10 times) faster than comparable RSA systems [9]. Certicom formulated this conclusion when comparing fast implementations of elliptic curve cryptosystems over binary finite fields using specially developed hardware with the implementation of RSA claiming a similar level of security [7].

When comparing public key cryptographic systems, there are three distinct factors to take into account:

> **Security.** What is the security based on. How long has the cryptosystem been in wide use and how much its security has been studied.
>
> **Efficiency.** How much computation is required to perform the public key and private key transformations. How many bits must be communicated to transfer an encrypted message or signature.
>
> **Space requirements.** How many bits are required to store the key pairs and associated system parameters.

In this chapter, I first go through the basic mathematical theory behind the RSA cryptosystem. Then I compare the elliptic curve cryptosystem and RSA cryptosystem in terms of security, efficiency, and space requirements, based on published research papers.

5.1 Theory of RSA Cryptosystem

The RSA cryptosystem was invented in 1977 by Rivest, Shamir and Adleman [43], and was the first realization of Diffie and Hellman's abstract model for public key cryptography [32].

The RSA algorithm is the best known of the integer factorization family of cryptosystems where the strength of the cryptosystem lies in the mathematical difficulty of factoring large integers. In this scheme integers of large enough size are chosen so as to make brute-force factorization infeasible, in the absence of any faster or better mathematical or computational techniques [7].

So for RSA, the security offered is currently considered, but not proven to be, dependent upon the mathematical difficulty inherent in factoring large numbers while the speed of the system is dependent upon modular exponentiation performance times for large integers [7].

The RSA cryptosystem is the most widely used public key cryptosystem today [32]. Since multiplication of integers modulo n is a relatively complicated procedure to implement, and since an exponentiation requires repeated multiplication, the RSA system cannot achieve the speeds of private key systems such as DES. Of course, this is also true for all other existing public key systems. RSA encryption and signature verification can be speeded up significantly by selecting a small exponent b [32]. Typical values used in practice are $b = 3$ and $b = 2^{16} + 1$. The fastest existing hardware implementation of RSA can encrypt data at the rate of 64 Kbits/sec [26] with a 512-bit modulus n. Software implementations on the Motorola DSP56000 which can encrypt at the rate of 13.4 Kbits/sec [6] and 11.6 Kbits/sec [18] have been reported for a 512-bit modulus.

Integer Factorization Problem

As mentioned before, the RSA cryptosystem is based on the integer factorization problem. To set up RSA cryptosystem, a user (lets call her Alice) picks two large primes p and q and computes their product $n = pq$. The group used is $G = \mathbb{Z}_n^*$, the multiplicative group of units in the integers modulo n. It is well known that the order of G is $\phi(n) = (p-1)(q-1)$, where ϕ denotes the Euler phi function. Clearly, Alice can compute the group order $\phi(n)$. Alice's public key is the pair of integers (n, b) and her private key is a. [32]

Now, it is easily seen that the problem of computing $\phi(n)$ given only n is computationally equivalent to the problem of factoring n. Moreover, no efficient algorithm is known for taking b-th roots in $\mathbb{Z}_n^*$ without the knowledge of p and q [32]. Hence it is believed (although no proof is known) that breaking the RSA cryptosystem is equivalent to factoring n. We say that the security of RSA is based on the factoring problem. Table 5.1 is from [39] and shows historical data on the integer factorization problem.

Table 5.1: Historical data on the integer factorization problem

Year	Number of decimal digits	Number of bits	MIPS years
1984	71	236	0.1
1988	106	352	140
1993	120	399	825
1994	129	429	5000
1995	119	395	250
1993	130	432	750

In the following I define the RSA encryption algorithm and the RSA signature scheme.

RSA Encryption Algorithm

The RSA encryption algorithm is defined as follows [55]. Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, and define

$$\mathcal{K} = \{(n, p, q, a, b) : n = pq,\ p, q \text{ prime}, ab \equiv 1(\text{mod } (\phi(n)))\}.$$

For $\mathcal{K} = (n, p, q, a, b)$, define

$$e_K(x) = x^b \bmod n$$

and

$$d_K(y) = y^a \bmod n$$

$(x, y \in \mathbb{Z}_n)$. The values n and b are public, and the values p, q, and a are secret.

RSA Signature Scheme

The RSA signature algorithm is defined as follows [55]. Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{A} = \mathbb{Z}_n$, and define

$$\mathcal{K} = \{(n, p, q, a, b) : n = pq,\ p, q \text{ prime}, ab \equiv 1(\text{mod } (\phi(n)))\}.$$

The values n and b are public, and the values p, q, and a are secret.
For $\mathcal{K} = (n, p, q, a, b)$, define

$$sig_K(x) = x^a \bmod n$$

and

$$ver_K(x, y) = \text{ true } \Leftrightarrow x \equiv y^b (\text{mod } n)$$

$(x, y \in \mathbb{Z}_n)$.

5.2 Security

Attacks Against RSA

The attacks against the RSA cryptosystem are based on solving the integer factorization problem (IFP).

There are two types of algorithms: special purpose and general purpose algorithms. Special-purpose algorithms attempt to exploit special features of the system under consideration, such as the number n being factored. A general-purpose algorithm solves all cases of the problem under consideration.

General-purpose algorithms. Prior to the development of the RSA cryptosystem, the best general-purpose factoring algorithm was the continued fraction algorithm, which could factor numbers up to 40 decimal digits (133 bits) [39]. This algorithm was based on the idea of using a factor base of primes and generating an associated set of linear equations whose solution ultimately led to factorization. This is the same idea underlying the best general-purpose algorithms used today: the quadratic sieve (QS) and the number field sieve (NFS) [39]. Both these algorithms can be easily paralleled to permit factoring on distributed networks of workstations. Large mainframe computers or supercomputers are therefore not essential to factor large numbers.

In 1984 Carl Pomerance developed the quadratic sieve [39]. Initially, it was used to factor numbers in the 70-decimal digit (223-bit) range. In 1994, it was used by a group of researches led by Arjen Lenstra to factor the 129-decimal digit (429-bit) RSA challenge number that was posed by Martin Gardner in 1977 [39]. The factorization was carried out in eight months by about 1600 computers around the world. The total running time for the factorization was estimated to be 5000 MIPS (Million Instruction Per Second) years [39].

The number field sieve was first developed in 1989 and works best on numbers of special form [39]. Although NFS was initially thought to be slower in practice than the quadratic sieve for factoring integers having fewer than 150 decimal digits (500 bits), later experiments have suggested that the NFS is indeed the superior algorithm for factoring integers having at least 120 decimal digits (400 bits) [39]. In 1996, a group led again by Arjen Lenstra used the NFS to factor a 130-decimal digit (432-bit) number. The factorization was estimated to take less than 15 percent of the 5000 MIPS years that was required for the factorization of the 129-decimal digit RSA challenge number [39].

On August 22 1999, a team of scientists from six different countries, led by Herman te Riele of CWI (Amsterdam), found the prime factors of 512-bit number [16]. The factored number, indicated by RSA-155, was taken from the RSA Challenge List.

The factorization was found using the number field sieve factoring algorithm, and beats the 140-digit record RSA-140, that was set on February 2, 1999, also with the help of NFS [58]. In order to find the prime factors of RSA-155, about 300 SGI and SUN workstations and Pentium PCs spent about 35 years of computing time. The computers were running in parallel and the whole task was finished in approximately seven calendar months. The amount of computer time spent on this new factoring world record is estimated to be equivalent to 8000 MIPS years (little less than the estimation in Table 5.2, from [39], shows). Given the current big distributed computing projects on Internet with hundreds of thousands of participants, it is possible to reduce the time to factor a 512-bit number from seven months to one week. For comparison, the amount of computing time needed to factor RSA-155 was less than 2% of the time needed to break RSA's DES challenge.

The factorization of RSA-155 and Table 5.2 shows that a 512-bit modulus n provides only marginal security when used in the RSA cryptosystem. For long-term security, 1024-bit or larger moduli should be used. Computers took several technical jumps forward during 1994-1995 [39]. The effect was reduction in work factor of solving the IFP problem.

Table 5.2: Computing power required to factor integers using the general number field sieve

Size of integer to be factored (in bits)	MIPS years
512	3 x 10^4
768	2 x 10^8
1024	3 x 10^{11}
1280	1 x 10^{14}
1536	3 x 10^{16}
2048	3 x 10^{20}

Special-purpose algorithms. For integer factorization, there is a fast algorithm for $n = pq$ provided $p - 1$ or $q - 1$ only has small prime factors. This special-purpose factoring algorithm is the elliptic curve factoring method (ECM), invented in 1985 by Hendrik Lenstra, Jr [39]. The running time of this method depends on the size of the prime factors of n, and hence the algorithm tends to find small factors first.

On June 21, 1995, Andreas Mueller (Universitaet des Saarlandes, Germany) announced that he had found a 44-decimal digit (147-bit) factor of a 99-decimal digit (329-bit) composite integer using the ECM [39]. The computation was carried out on a network of workstations and the speed was approximately 60 MIPS. The largest prime factor found thus far by ECM is 47-decimal digit (157-bit) prime factor of a 135-decimal digit (449-bit) number. The computation was carried out by Peter Montgomery and reported on November 27, 1995.

Table 5.3: Computing power required to compute elliptic curve logarithms with the Pollard Rho method

Field size (in bits)	Size of r (in bits)	$\sqrt{\pi r/2}$	MIPS years
163	160	2^{80}	9.6 x 10^{11}
191	186	2^{93}	7.9 x 10^{15}
239	234	2^{117}	1.6 x 10^{23}
359	354	2^{177}	1.5 x 10^{41}
431	426	2^{213}	1.0 x 10^{52}

Attacks Against Elliptic Curve Cryptosystems

The attacks against the elliptic curve cryptosystem are based on solving the elliptic curve discrete logarithm problem (ECDLP).

The existing methods currently used for attacking the discrete logarithm problem depend on having a finite Abelian group G. Hence these methods can also be applied to the elliptic curve analogue of the discrete logarithm problem (ECDLP) [7]. These methods are generally much slower because of the added complexity of the elliptic curve point addition and point doubling operations.

Letting E be an elliptic curve defined over a binary finite field, P and Q be points on E, the main attacks to date can be summarized as follows [7, 39]:

1. **Exhaustive search (using the Baby-step giant-step algorithm).** This algorithm (also known as Pollard's Rho algorithm), due to Pollard [41], has an expected running time of $\sqrt{\pi r/2}$ steps, where a step is an elliptic curve point addition and r is the order of the point P. Table 5.3 from [39] shows the computing power required to compute elliptic curve logarithms with the Pollard Rho method. In 1993, Van Oorschot and Wiener [60] parallelized Pollard's Rho algorithm to obtain the expected running time of roughly $\sqrt{\pi r/2}/m$ steps on m parallel processors. Wiener and Zuccherato further reduced this by $\sqrt{2}$ in the year 1998 [61]. That is, using m processors results in an m-fold speed-up. Zuccherato and Wiener's version of Pollard's Rho algorithm is the fastest general-purpose algorithm known for the ECDLP. To avoid this Baby-step giant-step attack, the order of E should be sufficiently large, i.e. $\#E > 2^{160}$ bits.

2. **Pohlig-Hellman algorithm.** This algorithm, due to Pohlig and Hellman [40], exploits the factorization of r (the order of the point P). The algorithm reduces the problem of recovering the secret key a to the problem of recovering a modulo each of the prime factors of r, a can then be recovered using the Chinese Remainder Theorem. Thus to construct the most difficult instance of the ECDLP, one must select an elliptic curve whose order is divisible by a large prime r, i.e. $r > 2^{160}$ bits. Preferably, this order should be a prime or

almost prime (i.e. a large prime r times a small integer h).

3. **Index-Calculus Method.** Unlike other types of groups used in cryptology, the elliptic curve group does not have a good notion of smoothness, i.e. there is no set of small elements in terms of which a random element has a good chance of being expressed by a simple algorithm [2]. For this reason, the Index-Calculus Method does not work for ECDLP. In 1991 however, Menezes, Okamoto, and Vanstone (MOV) [33] showed how the ECDLP can be reduced to the DLP in extension fields of $\mathbb{F}_q$, where the index-calculus methods can be applied. However, this MOV reduction algorithm is only effective for a very special class of curves known as supersingular curves.

In addition, some special classes of elliptic curves are susceptible to particular attacks. The relevant classes and how to avoid these attacks are detailed below [7].

1. **Supersingular curves.** The curve $E(\mathbb{F}_q)$ is said to be supersingular if it is of field of characteristic two and has j-invariant equal to zero [34]. If on the other hand $E(\mathbb{F}_q)$ is field of odd characteristic, the curve is supersingular if $q = 2$ or 3 and $j(E) = 0$, or $q \geq 5$ and the trace of Frobenius satisfies $t = 0$.

   The j-invariant is defined in Chapter 4.2 and Frobenius trace is defined in [5].

   The supersingular curves are specifically prohibited by the IEEE's P1363 standard and by other standards developed for elliptic curve cryptosystems such as ANSI X9.62, and ANSI X9.63.

   Menezes [32], Okamoto, and Vanstone [33], and Frey and Ruck [19] showed how, under mild assumptions, the ECDLP in an elliptic curve E defined over a finite field $\mathbb{F}_q$ can be reduced to the DLP in some extension field of the original field, where the index-calculus algorithms apply. The reduction algorithm is only practical if the exponent of the extension field is small; this is not the case for most elliptic curves. To ensure that this reduction algorithm does not apply to a particular curve, one only needs to check that r, the order of the point P, does not divide $q^B - 1$ for all small B for which the DLP in $\mathbb{F}_q$ B is intractable ($1 \leq B \leq 2000/(\log_2{}^q)$ suffices). This is known as the MOV condition.

2. **Anomalous curves.** Smart [53] and Satoh and Araki [47] independently showed that the ECDLP for the special class of anomalous elliptic curves is easy to solve. An anomalous elliptic curve over $\mathbb{F}_q$ is an elliptic curve over $\mathbb{F}_q$ which has exactly q points. This attack does not extend to any other classes of elliptic curves. Consequently, by verifying that the number of points on an elliptic curve does not equal the number of elements in the underlying field, one can easily ensure that the Smart-Satoh-Araki attack does not apply to a particular curve - these curves too are specifically prohibited in all standards of elliptic curve systems.

Whether or not there exists a subexponential time algorithm for the ECDLP is an important unsettled question, and one of great relevance to the security of ECC. It is very unlikely that anyone will be able to prove that no subexponential time algorithm exists for the ECDLP in the near future. Consequently, it is also very unlikely that anyone will be able to prove that no polynomial time algorithm exists for other bases for public key cryptography, the integer factorization and discrete logarithm problems. However, much work has been done on the DLP over the past 20 years, and more specifically on the ECDLP over the past 12 years [7]. No subexponential time algorithm has been discovered for the ECDLP, perhaps no such algorithm exists.

Menezes and Jurisic compared the time required to break the ECC with the time required to break RSA for various modulus sizes using the best general algorithm known [39]. Values were computed in MIPS years, which represents a computing time of one year on a machine capable of performing one million instructions per second. The results are listed in Table 5.4, and the table is from [39]. As a benchmark, it is generally accepted that 10^{12} MIPS years represents reasonable security at this time, since this would require most of the computing power on the planet to work for a considerable amount of time.

Table 5.4: Key size: Equivalent strength comparison

Time to break (in MIPS years)	RSA key size (in bits)	ECC key size (in bits)	RSA/ECC key size ratio
10^4	512	106	5 : 1
10^8	768	132	6 : 1
10^{11}	1024	160	7 : 1
10^{20}	2048	210	10 : 1
10^{78}	21000	600	35 : 1

Menezes and Jurisic found that to achieve reasonable security, RSA would need to employ a 1024-bit modulus, whereas a 160-bit modulus should be sufficient for the ECC [39]. They found that ECC required a smaller modulus than RSA and that the security gap between the systems grew as the key size increased. For example, 300-bit ECC is significantly more secure than 2000-bit RSA, because the ECDLP problem is judged to be the harder problem.

Another way to look at this security issue is to compare the equivalent strength of RSA keys and ECC keys for smart card applications. The Table 5.4 shows that in smart card applications requiring higher levels of security, ECC is able to offer security without a great deal of additional system resources.

In September 1999, nearly 200 people using 740 computers managed to crack a message encrypted with 97-bit elliptic curve cryptography [50]. The process took 16,000 MIPS-years of computing, about twice as much as used by the team that recently cracked a 512-bit RSA encryption key.

On April 4, 2000, an international team of researchers led by Robert Harley, Damien Doligez, Daniel de Rauglaudre, and Xavier Leroy of INRIA (France) announced the solution of the Certicom ECC2K-108 Challenge [14, 15].

The solution to the ECC2K-108 challenge is believed to be the largest effort ever expended in a public-key cryptography challenge. It took four months and involved approximately 9500 machines and 1300 volunteers from 40 countries. It was expected that the ECC2K-108 challenge would require about 5 times as much effort as expended in the ECC2-97 challenge solved in September 1999 [14, 15].

The ECC2K-108 challenge was solved using the parallelized Pollard Rho method and exploiting orbits of the negation and Frobenius maps. This method was developed in 1998 independently by Certicom researchers Rob Gallant, Rob Lambert, and Scott Vanstone and by Harley's team. The amount of work required to solve the ECC2K-108 challenge was about 50 times more than that required to solve the 512-bit RSA cryptosystem, that is about 50×8000 MIPS-years [14, 15].

While the solution of the ECC2K-108 challenge is an impressive computational achievement, it was based on known methods for solving the ECDLP and not on any fundamental advances [14, 15]. Nonetheless, the effort expended by Harley's group is of vital importance because it provides practical confirmation of the theoretical estimates of the difficulty of the ECDLP and the security of ECC.

5.3 Efficiency

In both RSA and ECC, considerable computational savings can be made [39]. In RSA, a short public exponent can be employed (although this represents a trade-off and does incur some security risks) to speed up signature verification and encryption. In ECC, a large proportion of the signature generation and encrypting transformations can be precomputed. Also, various special bases for the finite field $\mathbb{F}_{2^m}$ can be employed to perform the modular arithmetic involved in ECC operation more quickly. State-of-the-art implementations of the systems from Certicom show that with all of these efficiencies in place, ECC is an order of magnitude (roughly 10 times) faster than RSA [9]. The use of a short public exponent in RSA can make RSA encryption and signature verification timings (but not RSA decryption and signature generation timings) comparable with timings for these processes using the ECC.

Table 5.5 lists test results from 1998, when Certicom compared the times required by 163-bit ECC and 1024-bit RSA operations [10]. ECNRA means an elliptic curve version of the Nyberg-Rueppel algorithm, and ECDSA means an elliptic curve version of DSA. Certicom has performed the tests using 167-MHz UltraSparc running Solaris 2.5.1.

Table 5.5: Benchmarks for Solaris

Function	Security Builder 1.2 163-bit ECC (ms)	BSAFE 3.0 1024-bit RSA (ms)
Key Pair Generation	3.8	4708.3
Sign	2.1 (ECNRA) 3.0 (ECDSA)	228.4
Verify	9.9 (ECNRA) 10.7 (ECDSA)	12.7
Diffie-Hellman Key Exchange	7.3	1654.0

5.4 Space Requirements

Elliptic curve cryptosystems have the potential to provide security equivalent to that of existing public key schemes, but with shorter key lengths. Having short key lengths is a factor that can be crucial in some applications, for example, the design of smart card systems. The arithmetic processor on a smart card is restricted in size to an area of roughly 25 mm^2 [34]. An RSA chip designed to do modular multiplication of 512-bit numbers has about 50,000 transistors, while a chip designed to perform arithmetic in the field $\mathbb{F}_{2^{593}}$ has about 100,000 transistors [34]. With current technology, these devices are too large to be placed on a smart card. By comparison, a chip designed to do arithmetic in $\mathbb{F}_{2^m}$, where m is about 200, would have less than 15,000 transistors, and would occupy about 15% of the 25 mm^2 area assigned for the processor. Another advantage to be gained by using elliptic curves is that each user may select a different curve E, even though all users use the same underlying field K.

Table 5.6 is from Certicom [9], and compares the size of the system parameters and selected key pairs for the different systems, and presents evidence that the system parameters and key pairs are shorter for the 160-bit ECC than for 1024-bit RSA [39].

Table 5.6: Space requirements

	System parameters (bits)	Public key (bits)	Private key (bits)
1024-bit RSA	n/a	1088	2048
160-bit ECC	481	161	160

Certicom claims that both of the systems have similar bandwidth requirements when they are used to encrypt or sign long messages, but say this situation changes for the case where short messages are being transformed [39]. Public key cryptographic systems are often employed to transmit short messages, for example to transmit session keys for use in a private key cryptographic system. For the sake of this comparison, both schemes are being used to sign a 2000-bit message, or to encrypt a 100-bit message. See Tables 5.7 and 5.8 from [39]. The encryption algorithm used in encrypting 100-bit message is an ElGamal variant with point compression. These are described in Chapter 4.6.

Table 5.7: Signature sizes on long messages (2000-bit)

	Signature size (bits)
1024-bit RSA	1024
160-bit ECC	320

Table 5.8: Size of the encrypted 100-bit messages

	Encrypted message (bits)
1024-bit RSA	1024
160-bit ECC	321

Therefore it would appear from this comparison that ECC offers considerable bandwidth savings over the RSA when being used to transform short messages.

In summary, ECC provides greater efficiency than either integer factorization systems, in terms of computational overheads, key sizes and bandwidth. In implementations, these savings mean higher speeds, lower power consumption, and code size reductions. However, the RSA system is globally accepted in vendor offerings and may dominate the field until more research information about ECC becomes available [39]. Times are changing, though, and recently, in 1998, RSA Inc. announced the inclusion of ECC technology in its basic cryptographic tool kit [39].

Chapter 6

Description of Implementation

I have implemented the elliptic curve cryptosystem and converted an already implemented RSA cryptosystem Java implementation to be used in Sun Microsystems' Java Card emulator.

As described in Chapter 4.4 there are several reasons why curves over $\mathbb{F}_{2^n}$ are preferred, so I decided to use them in my elliptic curve cryptosystem implementation.

As stated in Chapter 4.3 there are three different bases, which can be used to implement fields of characteristic two: polynomial base, normal base, and subfield base. I decided to implement both the polynomial base and normal base (actually only optimal normal base) fields.

This chapter begins with an introduction of Java Cards. Then I describe the functionality of the Java Card emulator used in the tests. After these the general structure of the implementation is shown and the common classes for RSA and ECC implementations are defined. The rest of the chapter describes the RSA implementation and the implementation of the elliptic curve cryptosystem.

Java Card

The Java Card platform allows the on-card application to be written in Java. This brings the main advantages of Java to on-card software development. In addition, it provides a good basis for multi-application cards, where on the same card more than one application is supported. Many of the disadvantages of Java also tag along, unfortunately, like inefficiency and clumsiness of doing unsigned 16-bit arithmetic.

The on-card executable code consists of byte codes that are interpreted by the Java Card Runtime Environment, which controls the execution of the different applications while making sure that these applications do not interfere [23]. The goal is that Java Card applets can be run in any Java Card. This goal is not fully achieved yet because current implementations still differ slightly from the present specification and from each other.

The software stack of a Java Card is shown in Figure 6.1 [23]. The Java Card Runtime Environment (JCRE) has the following interfaces: The Card Executive manages the card and is the communication link between the card applet and the off-card code. The Java Virtual Machine (JVM) executes the bytecode of the applet and of the library functions it uses. The Java Card Framework provides the library functions. They form the standard Java Card API.

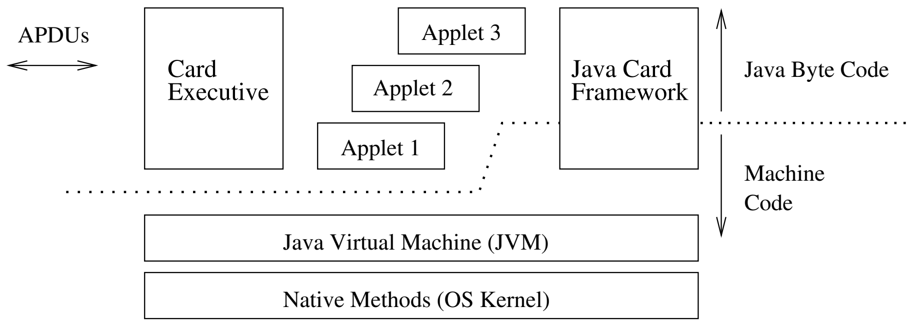

Figure 6.1: Software stack of a Java Card

The operating system kernel and the Java virtual machine (JVM) are native code, the layers above it are Java byte code [23].

Java Card Emulator

The Sun Microsystems' Java Card Development Kit is a suite of tools for designing Java Card technology-based implementations and developing applets based on the Java Card API 2.1 Specification [57].

The basic differences between the standard Java implementation and the specific Java Card features are the following: The Java Card Virtual Machine (JCVM) does not support dynamic class loading, garbage collection, or the `finalize()` method. It also does not support threads, cloning, default visibility override, and package visible interfaces can not be extended to public visibility. In addition, static instantiated classes and types `int`, `double`, `float`, and `long` are not supported [56, 57].

The Sun Microsystems' Java Card API is compatible with international standards, such as ISO7816, and industry-specific standards, such as Europay, MasterCard, Visa (EMV) [56].

Sun Microsystems' Java Card Development Kit contains JCWDE, Converter, Capgen and APDU tools to aid the application development.

Two types of outputs are supported [56, 57]: one output for mask production that is suitable for manufacturing smart card read-only memory. The other enables the production and installation of portable, interoperable CAP files, so Java Card applets can be added even after the card is produced. Both types of output start by compiling and inputting Java language source code to a Java Card Converter. They also support Java Card byte code running in a JCRE (Java Card Runtime Environment).

JCWDE stands for Java Card Workstation Development Environment. The JCWDE tool kit allows the simulated running of a Java Card applet as if it were masked in ROM. It emulates the card environment [56, 57].

The Java source code can also be run through the Converter tool to verify that the applet uses only the supported subset of Java programming language. As an additional purpose, the Converter tool outputs a JCA (Java Card Assembly) file containing a representation of the applet [56, 57]. A JCA file is a human-readable ASCII file to aid testing and debugging. Finally, the JCA file can be inputted into the Capgen tool to convert the applet to a CAP file conforming to the Sun Microsystems' Java Card 2.1 Virtual Machine Specification.

The Java Card and card reader communicate by sending packets of data known as APDUs (Application Protocol Data Unit). The APDU tool reads a script file containing command APDUs and sends them to the JCWDE. Each command APDU (C-APDU) is processed by the JCWDE and returned to the APDU tool, which displays both the command and response APDUs on the console [56, 57].

Structure of Implementation

The implementation has been divided into three separate packages: `RSA`, `PB_ECC`, and `ONB_ECC`.

The `RSA` package contains all the classes needed to implement the RSA cryptosystem, the `PB_ECC` package contains the classes needed to implement polynomial base elliptic curve cryptosystem, and respectively `ONB_ECC` package contains classes needed to implement optimal normal base elliptic curve cryptosystem.

The classes in each package are shown in Figure 6.2.

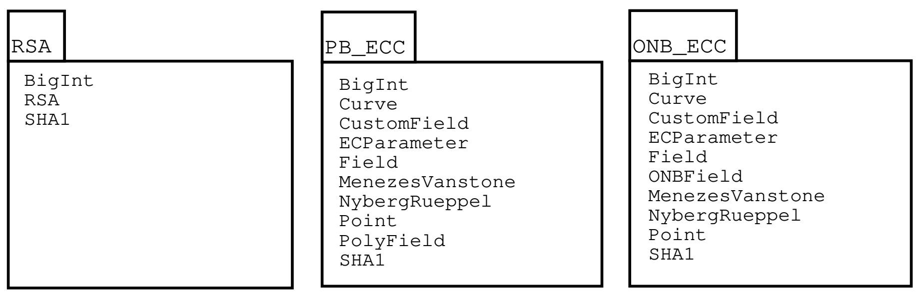

Figure 6.2: Structure of implementation

As can be seen from the Figure 6.2 the only difference between the `PB_ECC` and

ONB_ECC packages is one class. PB_ECC uses the PolyField class and ONB_ECC uses the ONBField class.

Description of Common Classes

As is shown in Figure 6.1, there are two classes common to all packages, BigInt and SHA1.

The reason I had to implement these classes is that Sun Microsystems' Java Card Development Kit does not contain implementations of large integers and message digests, although many commercial Java Card Development Kits and Java Cards do. Also, having to run all code in the Java Virtual Machine guarantees that the test results are not biased because of running some parts in native code and some parts in the Java Virtual Machine.

BigInt

The implementation of BigInt follows loosely Michael Rosing's implementation given in [45]. Rosing's implementation is however in C.

The BigInt class includes methods for addition, subtraction, multiplication, and division of large integers. It also contains methods for doing modular arithmetic, such as modular exponentiation (modPow), $g^x \bmod n$, and modular inversion (modInverse), $ax \equiv 1 \bmod n$.

The BigInt class is based on mutable arrays and thus there is no need for dynamic memory allocation which is not possible on Java Cards. A mutable implementation possibly also incurs some speedup over the standard immutable BigInteger class and allows for more flexibility for doing optimizations.

The public methods of BigInt are shown in Figures 6.3, 6.4, and 6.5.

SHA1

The SHA1 class contains an implementation of the SHA-1 algorithm used in the RSA and Nyberg-Rueppel signature algorithms.

The SHA-1 is defined by NIST. The IEEE P1363 among other standards suggests it to be used with signature schemes.

This SHA-1 implementation is based on a Java version by Chuck McManis [31]. McManis's implementation is not however made for Java Cards.

The public methods of SHA1 are shown in Figures 6.3, 6.4, and 6.5.

6.1 RSA Implementation

The RSA implementation consists of the `RSA` package. The package contains a class named `RSA` and the previously described classes `BigInt` and `SHA1`.

`RSA`

The `RSA` class implements the RSA encryption algorithm and RSA signature scheme, described in Chapter 5.1. In addition it contains CRT (Chinese Remainder Theorem) variations of the algorithms (`crtDecrypt`, `crtSign`). The CRT method is described later in this chapter.

All the methods in the `RSA` class use `BigInts`. In addition, the RSA signature scheme and the CRT variation of that scheme use `SHA1` to compute SHA-1 message digests.

The public methods of `RSA` are shown in Figure 6.3.

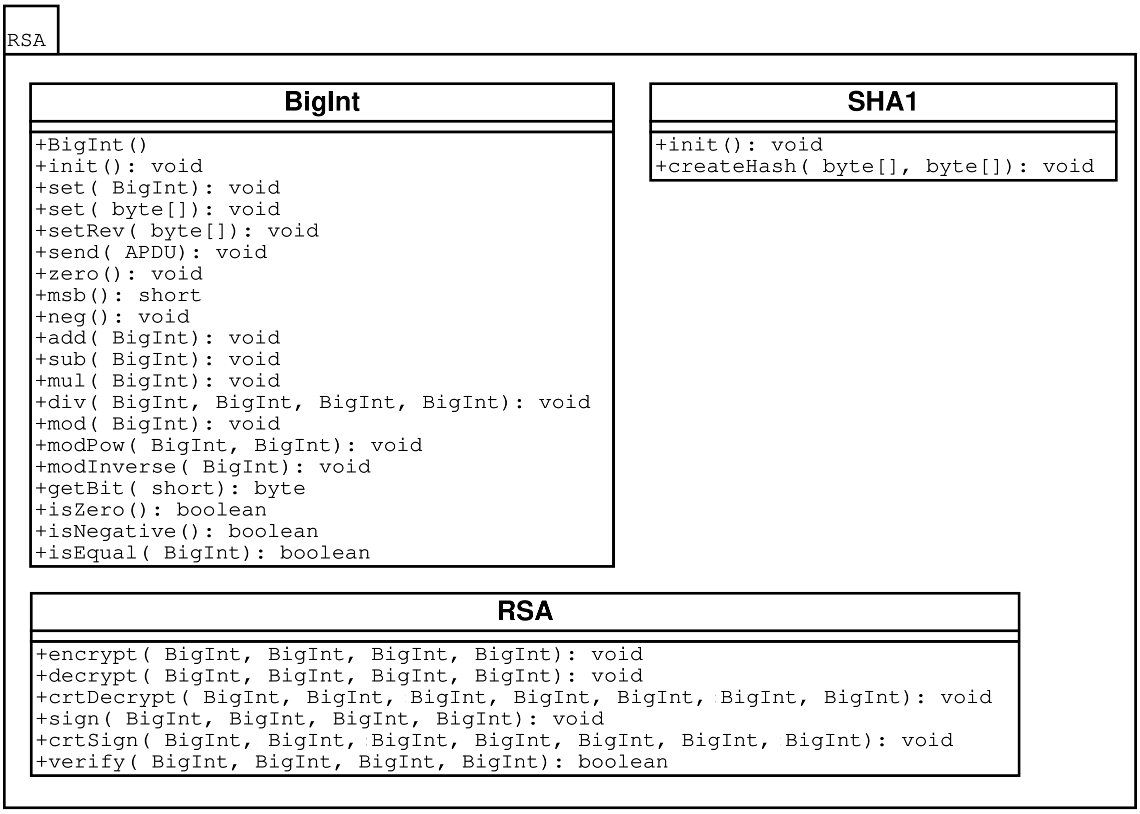

Figure 6.3: Class structure of RSA implementation

RSA with Chinese Remainder Theorem

As described before in Chapter 5.1, the RSA encryption and signature verification can be speeded up significantly by selecting a small public exponent b.

Another way to speed up RSA is to use Chinese Remainder Theorem (CRT) as described in PKCS #1 v2.1 RSA Cryptography Standard [46]. In this method the RSA private key consists of a quintuple $(p, q, dP, dQ, qInv)$, where the components have the following meaning: p is the first factor, q is the second factor, dP is the first factor's exponent, dQ is the second factor's exponent and $qInv$ is the CRT coefficient. All components are nonnegative integers [46].

In a valid RSA private key the two factors p and q are the prime factors of the modulus n, the exponents dP and dQ are positive integers less than p and q respectively, b is the public exponent, satisfying

$$b \cdot dP \equiv 1(\mathrm{mod}(p-1))$$
$$b \cdot dQ \equiv 1(\mathrm{mod}(q-1))$$

and the CRT coefficient $qInv$ is a positive integer less than p, satisfying

$$q \cdot qInv \equiv 1(\mathrm{mod}\ p).$$

RSA Decryption with Chinese Remainder Theorem

The RSA decryption using Chinese Remainder Theorem is defined as follows [46]. c is a ciphertext representative, an integer between 0 and $n-1$. m is a message representative, an integer between 0 and $n-1$.

$$m_1 = c^{dP} \bmod p$$
$$m_2 = c^{dQ} \bmod q$$
$$h = qInv(m_1 - m_2) \bmod p$$
$$m = m_2 + h \cdot q$$

RSA Signature Scheme with Chinese Remainder Theorem

The RSA signature scheme using Chinese Remainder Theorem is defined as follows [46]. m is a message representative, an integer between 0 and $n-1$. s is a signature representative, an integer between 0 and $n-1$.

$$s_1 = m^{dP} \bmod p$$
$$s_2 = m^{dQ} \bmod q$$

$$h = qInv(s_1 - s_2) \bmod p$$

$$s = s_2 + h \cdot q$$

6.2 Implementation of Elliptic Curve Cryptosystem

There are two main representation of the field $\mathbb{F}_{2^n}$, polynomial representation and normal base representation. In polynomial base representation the binary multipliers are written from the highest power to the lowest. The normal base is handled respectively, the multipliers are listed from the most significant to the least significant. I have chosen to use the point $(0, 0)$ to be the point at infinity because it is never on the curve.

The elliptic curve operations require addition, multiplication, squaring and inversion in the underlying field. The inversion operation is by far the most expensive.

The elliptic curve cryptosystem implementation consists of `PB_ECC` and `ONB_ECC` packages. These both packages are very similar, the only difference between them is that because the `PB_ECC` uses polynomial base fields it has a `PolyField` class, and because `ONB_ECC` uses optimal normal base fields it has an `ONBField` class. Most of the procedures are based on those in the book [45] by Michael Rosing.

Below are described all the classes which are common to both these packages, namely classes `Field`, `CustomField`, `Curve`, `Point`, `ECParameter`, `MenezesVanstone`, and `NybergRueppel`.

`Field`

The `Field` class is the only abstract class in this implementation. The `Field` class contains methods for shifting bits in the field (`shiftLeft`, `shiftRight`, `rotLeft`, `rotRight`), adding field to another field (`sum`), and generating a random field (`random`).

Field addition is simply a bitwise exclusive-or operation.

The public methods in `Field` class are shown in Figures 6.4 and 6.5.

`CustomField`

The `CustomField` class implements custom sized fields. It contains methods for shifting bits in `CustomFields`, and a method to solve the equation $b = a \times u^n$ (`cusTimes`).

The optimal normal base implementation uses the `cusTimes` method when executing the Schroeppel's Almost Inverse Algorithm. The `cusTimes` method consists of shift and exclusive-or operations. The Almost Inverse Algorithm is described in Chapter 6.2.1.

The public methods in `CustomField` class are shown in Figures 6.4 and 6.5.

`Curve`

The `Curve` class implements basic elliptic curve operations. It has methods for doubling points (`doublep`), adding two points (`sum`), subtracting two points (`sub`), and multiplying a point with a scalar (`mul`) on a given curve.

The formulae for adding and doubling points are from IEEE P1363 standard [25]. The arithmetic is described in Chapter 4.4.

The scalar multiplication of points follows Solinas's article [54] and is described in Chapter 6.2.1.

The public methods in `Curve` class are shown in Figures 6.4 and 6.5.

`Point`

The `Point` class is a container for points on elliptic curves. This class contains only constructors and a `set` function that sets `Point`'s coordinates.

The public methods in `Point` are shown in Figures 6.4 and 6.5.

`ECParameter`

The `ECParameter` class is a container for elliptic curve parameters. The parameters include the chosen elliptic curve to be used, the point order of the curve, and the base point of the chosen curve.

The public methods in `ECParameter` are shown in Figures 6.4 and 6.5.

`MenezesVanstone`

The `MenezesVanstone` class implements the Menezes-Vanstone elliptic curve cryptosystem, described in Chapter 4.6.

The `encrypt` and `decrypt` methods in the class use `ECParameter`s, `Field`s, and `Point`s.

The public methods in `MenezesVanstone` are shown in Figures 6.4 and 6.5.

`NybergRueppel`

The `NybergRueppel` class implements the Nyberg-Rueppel signature scheme following the IEEE P1363 standard [25]. The Nyberg-Rueppel algorithm is described in Chapter 4.7.

The `sign` and `verify` methods in the `NybergRueppel` class use `ECParameter`s, `Field`s, and `Point`s. In addition, the signature scheme uses the `SHA1` class to compute SHA-1 message digests.

The public methods in `NybergRueppel` are shown in Figures 6.4 and 6.5.

6.2.1 Used Algorithms

The elliptic curve cryptosystem implementation uses the following algorithms to speed up the inversion and scalar multiplication.

Schroeppel's Almost Inverse Algorithm

This algorithm is from [51]. For the field we are working in, say $\mathbb{F}_{2^{155}}$, the problem to be solved is:

Given a non-zero polynomial $A(u)$ of degree less than or equal to 154, find the (unique) polynomial $B(u)$ of degree less than or equal to 154 such that

$$A(u)B(u) \equiv 1 \bmod u^{155} + u^{62} + 1.$$

The problem has a simple, but relatively slow, recursive solution, exactly analogous to the related algorithm for integers. The Almost Inverse algorithm is considerably faster. It borrows ideas from Berlekamp [3] and from the low-end GCD algorithm of Roland Silver, John Terzian, and J. Stein (described in Knuth [27] p.297). The Almost Inverse algorithm computes $B(u)$ and k such that

$$AB \equiv u^k \bmod M,\ deg(B) < deg(M), \text{ and } k < 2\ deg(M)$$

where $deg(B)$ denotes the polynomial degree of B. After executing the algorithm, we will need to divide B by $u^k (\bmod\ M)$ to get the true reciprocal of A.

The pseudo-code for the algorithm is given below. The computer implementation relies on a few representational items:

- Multiplication of a polynomial by u is a left-shift by 1 bit.
- Division of a polynomial by u is a right-shift by 1 bit.
- A polynomial is even if its least significant bit, the coefficient of u^0 is 0. Otherwise it is odd.

The algorithm will work whenever $A(u)$ and $M(u)$ are relatively prime, $A(u) \neq 0$, $M(u)$ is odd, and $deg(M) > 0$.

```
The Almost Inverse Algorithm

        Initialize integer k = 0, and polynomials B = 1, C = 0, F = A,
        G = M.
```

```
loop:   While F is even, do F = F / u, C = C * u, k = k + 1.
        If F = 1, then return B, k.
        If deg(F) < deg(G), then exchange F, G and exchange B, C.
        F = F + G, B = B + C.
        Goto loop.
```

Solinas's Addition-Subtraction Method

This algorithm is from [54]. The basic technique for elliptic scalar multiplication is the addition-subtraction method. This begins with the nonadjacent form (NAF) of the coefficient n: a signed binary expansion with the property that no two consecutive coefficients are nonzero.

Just as every positive integer has a unique binary expansion, it also has a unique NAF. Moreover, NAF(n) has the fewest nonzero coefficients of any signed binary expansion of n. There are several ways to construct the NAF of n from its binary expansion.

The idea is to divide repeatedly by 2. One can derive the binary expansion of an integer by dividing by 2, storing the remainder (0 or 1), and repeating the process with the quotient. To derive NAF, one allows remainders of 0 or ± 1, one chooses whichever makes the quotient even.

```
NAF Algorithm

        k = n, S = ().

        While k > 0 do
                If k is odd, then
                        set u = 2 - (k (mod 4))
                else
                        set u = 0.
                k = k - u.
                Prepend u to S.
                k = k / 2.
        End While
        Output S
```

We now implement elliptic scalar multiplication using NAF. Given the NAF

$$n = \sum_{i=0}^{l-i} e_i 2^i,$$

the elliptic scalar multiplication $Q = nP$ is performed as follows.

```
Addition-Subtraction Method

        Q = P.

        For i = l - 2 downto 1 do
                Q = 2Q.
                If e_i = 1 then set Q = Q + P.
                If e_i = -1 then set Q = Q - P.
        Output Q.
```

This is about one-eight faster than the binary method, which uses the ordinary binary expansion in place of the NAF [54].

6.2.2 Optimal Normal Base Implementation of ECC

The optimal normal base implementation consists of the `ONB_ECC` package. The `ONB_ECC` package contains the `ONBField` class and the previously described classes `Curve`, `CustomField`, `ECParameter`, `Field`, `MenezesVanstone`, `NybergRueppel`, and `Point`.

The implementation of normal basis arithmetic is quite simple, only bitwise and, bitwise exclusive-or, and shift operations are needed. The fact that these are the fastest operations possible on any microprocessor makes optimal normal base (ONB) attractive [45].

Squaring a normal base number amounts to a rotation. Addition is simply an exclusive-or operation.

The inversion uses Schroeppel's Almost Inverse algorithm, described in Chapter 6.2.1.

The basics of multiplication are the same in any mathematical system, just multiply coefficients and sum over all those that have the same power. The optimal normal base implementation uses a precomputed `lambda` matrix to speed up the multiplication.

As mentioned in Chapter 4.3, there are two types of optimal normal bases over $\mathbb{F}_{2^n}$, called Type I and Type II. The only implementation related difference between them is the way the bits in the `lambda` matrix are set. For Type I ONB only one vector is stored whereas for Type II ONB two vectors are stored [45].

The `lambda` vector for Type I ONB stores all the values of j for each value of i that satisfies the equation $2^i + 2^j = 1 \bmod (m+1)$. The `lambda` matrix for Type II ONB is built by working with group of four equations. To build the `lambda` matrix, we have to find solutions to

$$2^i + 2^j = 1$$

$$2^i + 2^j = -1$$

$$2^i - 2^j = 1$$
$$2^i - 2^j = -1.$$

The operation for field addition is implemented in the `Field` class. The rest of the operations (multiplication, squaring and inversion) needed in elliptic curve cryptosystem for optimal normal base fields are implemented in the `ONBField` class.

`ONBField`

The `ONBField` class implements optimal normal base fields over $\mathbb{F}_{2^n}$.

The `genLambda` method together with the `initTwo` method create the lambda vectors described above.

The field multiplication is implemented in the `mul` method, which uses the precomputed lambda vectors. Squaring a field is implemented in the `square` method.

The `inv` method computes the inversion of a field using the Schroeppel's Almost Inverse algorithm. The algorithm is described in detail in Chapter 6.2.1.

The public methods in `ONBField` are shown in Figure 6.4.

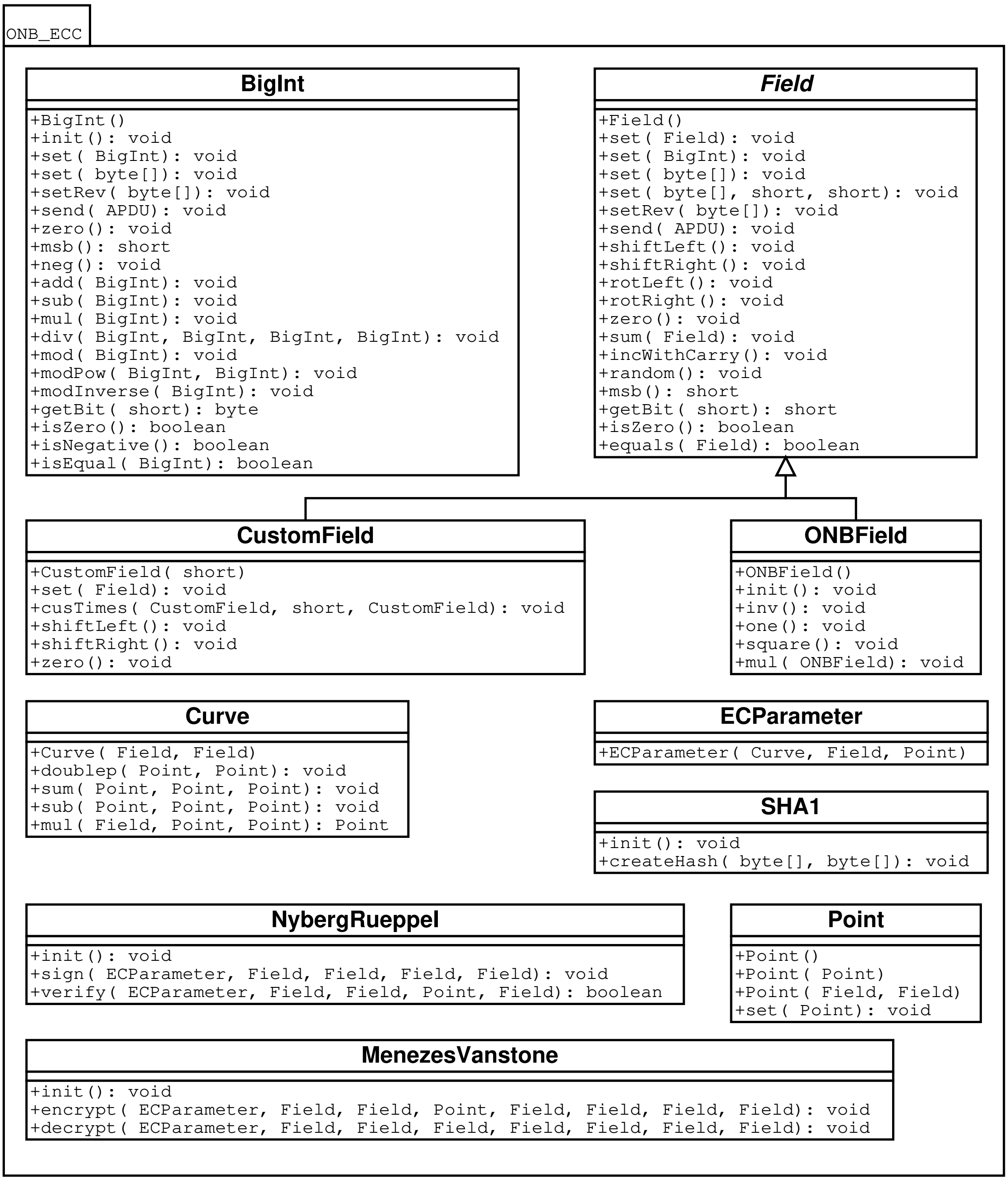

Figure 6.4: Class structure of optimal normal base ECC implementation

6.2.3 Polynomial Base Implementation of ECC

The polynomial base implementation consists of the `PB_ECC` package. The `PB_ECC` package contains the `PolyField` class and the previously described classes `Curve`, `CustomField`, `ECParameter`, `Field`, `MenezesVanstone`, `NybergRueppel`, and `Point`.

The polynomial base implementation is simple because when it applies to base 2 computer arithmetic the addition is just exclusive-or and multiplication is shifting and adding. Division is also simple, it is shifting and adding, the result is quotient and remainder.

The inversion uses Schroeppel's Almost Inverse algorithm, described in Chapter 6.2.1.

The operation for field addition is implemented in the `Field` class. The rest of the operations (multiplication, squaring and inversion) needed in elliptic curve cryptosystem for polynomial base fields are implemented in the `PolyField` class.

PolyField

The `PolyField` class implements polynomial base fields over $\mathbb{F}_{2^n}$.

Field multiplication is implemented in the `mul` method, which uses the `mulPartial` method.

The `div` method implements field division and squaring a field is implemented in the `square` method.

The `inv` method computes the inverse of a field using the Schroeppel's Almost Inverse algorithm. The algorithm is described in detail in 6.2.1.

The `PolyField` class contains also a method for computing modular multiplication, $a \times b \bmod n$, (`mulMod`).

The public methods in `PolyField` are shown in Figure 6.5.

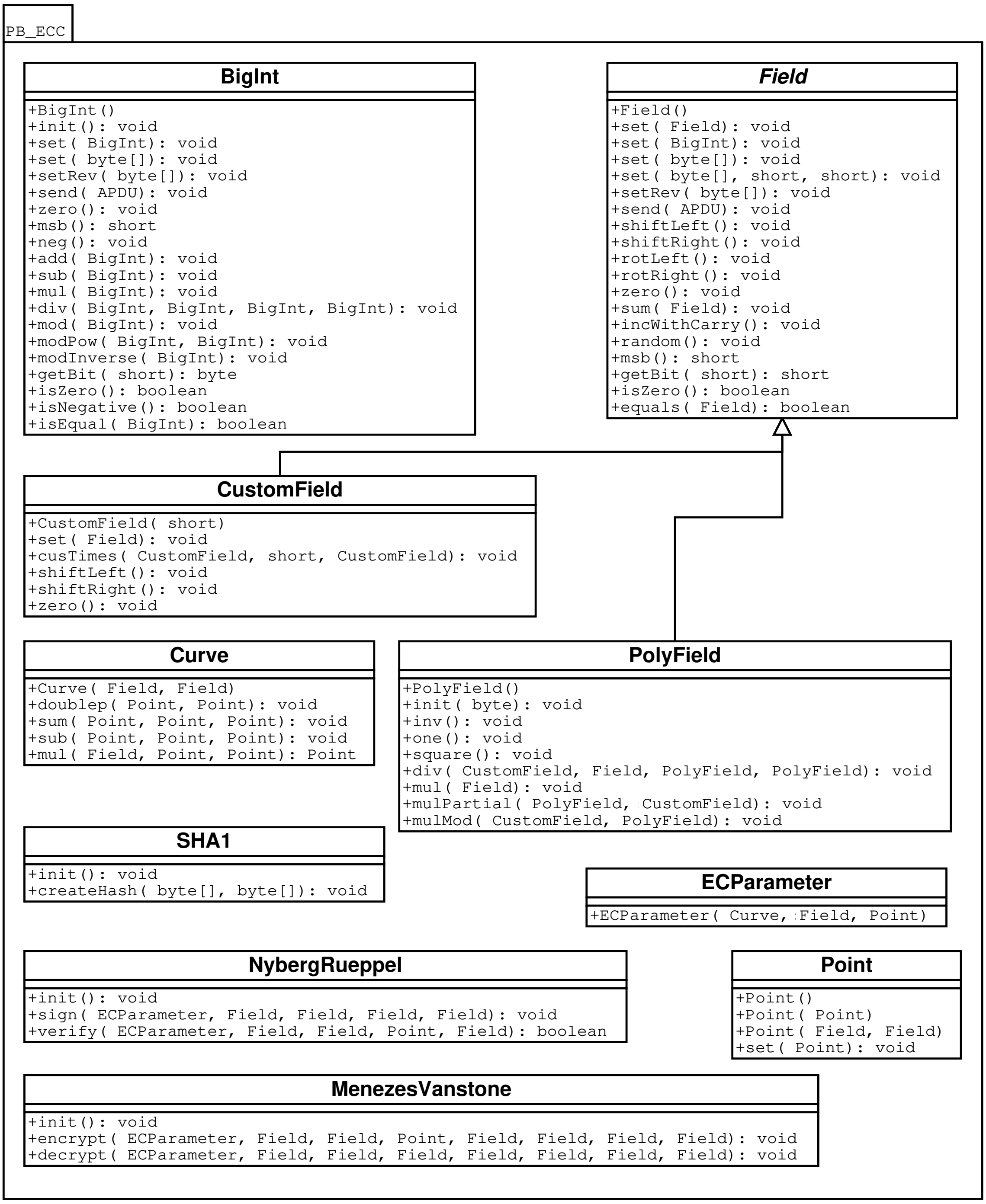

Figure 6.5: Class structure of polynomial base ECC implementation

Chapter 7

Test Setup

As mentioned before the objective of this thesis is to clarify whether the elliptic curve cryptography is better suited to be used with smart cards than the RSA cryptosystem.

There are three different characteristics I have decided to compare between these two cryptosystems, namely performance, security, and space requirements.

The rest of this chapter states the algorithms, test parameters, test environment, and methods I used to test the performance of the cryptosystems described in Chapter 6. The analysis of the performance, security and space requirements can be found in Chapter 9. Security and space requirements of the cryptosystems can be derived theoretically, so there is no need to conduct benchmarks to measure them. Benchmarking security is not even possible.

The test results from performance tests are listed in Chapter 8.

Tested algorithms were the Menezes-Vanstone elliptic curve cryptosystem, the Nyberg-Rueppel signature scheme, and the RSA encryption and signing algorithms. The Menezes-Vanstone algorithm is described in Chapter 4.6 and the Nyberg-Rueppel algorithm is described in Chapter 4.7. The implementation of both these algorithms is described in Chapter 6.2. The RSA cryptosystem is described in Chapter 5.1 and the description of its implementation can be found in Chapter 6.1.

The tests consisted of timing the encryption, decryption, signing and signature verification operations using the above algorithms and test parameters listed in Appendix A. All the necessary parameters and initialization processes have been performed before testing e.g. key generation, generation of the lambda matrix, etc.

As shown in Table 5.4, a 1024-bit RSA key has roughly the same strength as a 160-bit ECC key, and a 2048-bit RSA key has about the same strength as a 210-bit ECC key. Based on this I have decided to compare the speed of 1024-bit RSA operations to 158-, 163-, and 174-bit ECC operations, and 2048-bit RSA operations to 193-, 194-, 209-, and 233-bit ECC operations.

As already mentioned in Chapter 6.1 the RSA cryptosystem can be speeded up by using a small public exponent or by using the Chinese Remainder Theorem. The test parameters contain both the RSA keys with a small public exponent and RSA keys containing CRT parameters.

All the elliptic curves and RSA parameters used in the tests can be found in Appendix A. There are four RSA keys (A.1, A.2, A.3 and A.4), four polynomial base elliptic curves (A.5, A.6, A.7 and A.8), and four optimal normal base elliptic curves (A.9, A.10, A.11 and A.12). All the RSA keys are generated with SSH's key generation software. Some of the elliptic curves are found from standards and books, some of them are created with curve generating software, because curves of that type and field size could not be found in literature. The reference for each test parameter is given in Appendix A.

Each operation for every test parameter was benchmarked separately in batches of 8-100 operations (depending on the time the operations take). Each batch was run 20 times in order to reach satisfactory confidence intervals. A 95% confidence interval was calculated from the test results using the T-distribution.

The tests were performed using a PC with a 433 MHz Celeron processor (128 KB L2 cache, 66 MHz front side bus), 128 MB memory and running Linux-2.2.16. Tests were performed on the Java Card emulator described in Chapter 6, using Java™ 2 SDK, Standard Edition, version 1.2.2 (Blackdown).

Chapter 8

Test Results

This chapter contains the test results of the RSA and elliptic curve cryptosystem implementations. The test setup is described in Chapter 7. The test results contain the lower and upper limits of 95% confidence interval calculated using the T-distribution.

8.1 RSA Test Results

Table 8.1: RSA cryptosystem's test results

RSA key	Encryption (ms)	Decryption (ms)	Signing (ms)	Signature Verification (ms)
1024-bit (A.1)	45.8 ± 2.9	80427.9 ± 5.2	80271.1 ± 2.8	213.6 ± 2.7
1024-bit (A.2)	826.5 ± 3.5	77664.2 ± 3.6	77919.8 ± 3.8	988.6 ± 3.2
2048-bit (A.3)	123.0 ± 10.2	622209.1 ± 260.8	621918.8 ± 366.3	827.4 ± 24.8
2048-bit (A.4)	3081.5 ± 9.3	626272.5 ± 367.8	624746.2 ± 166.4	3863.9 ± 82.6

Table 8.2: RSA with Chinese Remainder Theorem test results

RSA key	Encryption (ms)	Decryption (ms)	Signing (ms)	Signature Verification (ms)
1024-bit (A.1)	45.8 ± 2.9	43567.8 ± 13.1	43529.8 ± 13.5	213.6 ± 2.7
1024-bit (A.2)	826.5 ± 3.5	43633.8 ± 5.0	43390.1 ± 3.0	988.6 ± 3.2
2048-bit (A.3)	123.0 ± 10.2	344482.0 ± 391.3	343580.4 ± 106.9	827.4 ± 24.8
2048-bit (A.4)	3081.5 ± 9.3	347713.0 ± 109.1	346647.4 ± 135.3	3863.9 ± 82.6

8.2 Elliptic Curve Cryptosystem Test Results

Table 8.3: Polynomial base ECC test results

Elliptic curve	Encryption (ms)	Decryption (ms)	Signing (ms)	Signature Verification (ms)
163-bit (A.5)	4026.4 ± 4.5	1951.8 ± 3.8	2009.3 ± 0.3	4031.2 ± 0.8
163-bit (A.6)	4018.7 ± 0.6	2013.3 ± 0.3	2011.3 ± 0.8	3906.4 ± 3.0
193-bit (A.7)	6708.4 ± 2.1	3302.1 ± 0.3	3353.7 ± 0.3	6445.8 ± 0.6
233-bit (A.8)	11617.9 ± 13.0	5844.6 ± 13.1	5797.5 ± 1.0	11477.8 ± 2.8

Table 8.4: Optimal normal base ECC test results

Elliptic curve	Encryption (ms)	Decryption (ms)	Signing (ms)	Signature Verification (ms)
158-bit (A.9)	1613.0 ± 0.3	830.1 ± 0.3	806.0 ± 0.3	1569.7 ± 0.5
174-bit (A.10)	2169.5 ± 0.3	949.8 ± 0.4	1085.0 ± 0.6	1851.8 ± 0.6
194-bit (A.11)	3006.3 ± 1.6	1415.2 ± 0.4	1502.5 ± 0.3	2700.1 ± 4.5
209-bit (A.12)	3730.3 ± 0.6	1814.2 ± 0.4	1863.0 ± 1.2	3492.2 ± 5.0

Chapter 9

Analysis of Test Results

9.1 Performance

According to the test results listed in Chapter 8, encryption with 1024-bit RSA is always faster than encryption with 158-, 163- or 174-bit ECC. The greatest difference appeared when encryption was made using 1024-bit RSA with public exponent 3, then RSA encryption was 88 times faster than encryption with 163-bit polynomial base ECC. The smallest difference occurred when RSA's public exponent was $2^{16}+1$, then RSA encryption was only 2 times faster than encryption with 158-bit optimal normal base ECC.

Signature verification with 1024-bit RSA was also always faster than with 158-, 163- or 174-bit ECC. The greatest difference appeared when signature verification was made using 1024-bit RSA with public exponent 3, then RSA signature verification was 19 times faster than signature verification with 163-bit polynomial base ECC. The smallest difference occurred when RSA's public exponent was $2^{16}+1$, then RSA signature verification was only 2 times faster than signature verification with 158-bit optimal normal base ECC.

On the other hand, decryption with 158-, 163- or 174-bit ECC was always faster than with 1024-bit RSA. The greatest difference appeared when decryption was made using 158-bit optimal normal base ECC, then ECC decryption was 97 times faster than decryption with 1024-bit RSA using public exponent 3. The smallest difference occurred with 163-bit polynomial base ECC, then ECC decryption was 22 times faster than decryption with 1024-bit RSA using CRT and public exponent 3.

Signing with 158-, 163- or 174-bit ECC was also always faster than with 1024-bit RSA. The greatest difference appeared when signing was made using 158-bit optimal normal base ECC, then ECC signing was 100 times faster than signing with 1024-bit RSA using public exponent 3. The smallest difference occurred with 163-bit polynomial base ECC, then ECC signing was 22 times faster than signing with 1024-bit RSA using CRT and public exponent $2^{16}+1$.

Now that we have compared 1024-bit RSA with 158-, 163-, and 174-bit ECC we can move to compare the stronger keys, 2048-bit RSA and 193-, 194-, 209-, and 233-bit ECC.

The greatest difference in encryption appeared when encryption was made using 2048-bit RSA with public exponent 3, then RSA encryption was 94 times faster than encryption with 233-bit polynomial base ECC. The smallest difference occurred when RSA's public exponent was $2^{16} + 1$, then RSA encryption was nearly as fast as encryption with 194-bit optimal normal base ECC.

The greatest difference in signature verification appeared when signature verification was made using 2048-bit RSA with public exponent 3, then RSA signature verification was 14 times faster than signature verification with 233-bit polynomial base ECC. The smallest difference occurred when RSA's public exponent was $2^{16} + 1$, then signature verification with 194-bit optimal normal base ECC was 1.5 times faster than RSA signature verification.

Again, decryption with 193-, 194-, 209- or 233-bit ECC was always faster than with 2048-bit RSA. The greatest difference appeared when decryption was made using 194-bit optimal normal base ECC, then ECC decryption was 400 times faster than decryption with 2048-bit RSA using public exponent $2^{16}+1$. The smallest difference occurred with 233-bit polynomial base ECC, then ECC decryption was 60 times faster than decryption with 2048-bit RSA using CRT and public exponent 3.

Signing with 193-, 194-, 209- or 233-bit ECC was also always faster than with 2048-bit RSA. The greatest difference appeared when signing was made using 194-bit optimal normal base ECC, then ECC signing was 400 times faster than signing with 2048-bit RSA using public exponent $2^{16} + 1$. The smallest difference occurred with 233-bit polynomial base ECC, then ECC signing was 60 times faster than signing with 2048-bit RSA using CRT and public exponent 3.

Figures 9.1 - 9.4 illustrate the differences in times when performing these four cryptographic operations. Differences between the elliptic curves arise from the key sizes and the base used. The figures show that optimal normal base curves are faster than the polynomial base curves. The RSA operations shown in the figures use CRT and public exponent 3.

As described in Chapter 5, Robshaw and Yin claim in their article [44] that encryption with ECC takes more time than with RSA, but both decryption and signing processes are faster with ECC. The test results from my performance tests indicate the same. On the other hand, Certicom claims in [9] that the most efficient implementations of elliptic curve cryptosystems are an order of magnitude faster than comparable RSA systems, but they used specially developed software and highly optimized algorithms, which I did not.

As can be seen from the test results, the CRT and small public exponent can speed up RSA significantly. When implementing the ECC, a major speed up was accomplished by using the Schroeppel's Almost Inverse algorithm. The Almost Inverse algorithm made the both polynomial and optimal normal base implementations 4 times faster than when using the basic Euclidean algorithm. In addition, the precomputed Lambda matrix speeded up the optimal normal base implementation.

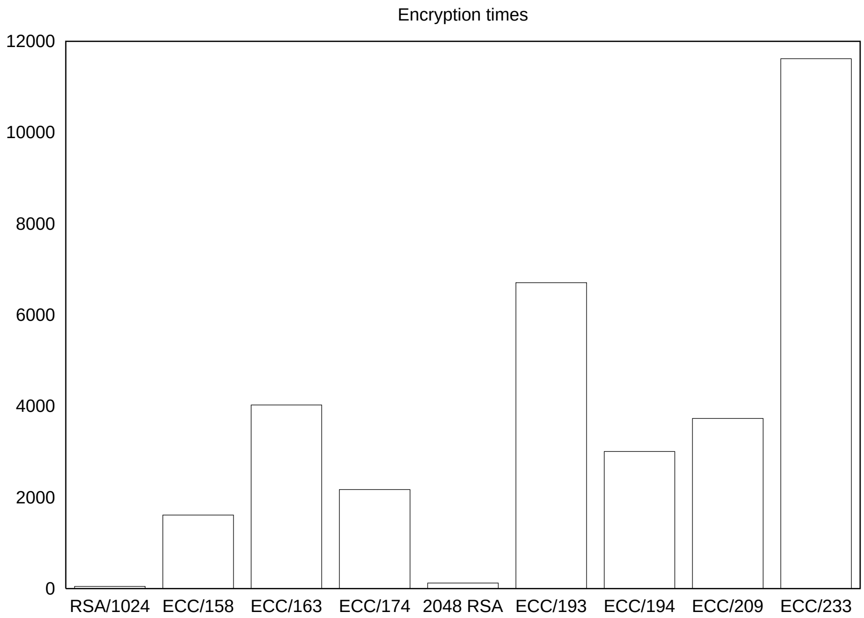

Figure 9.1: Encryption times (in milliseconds)

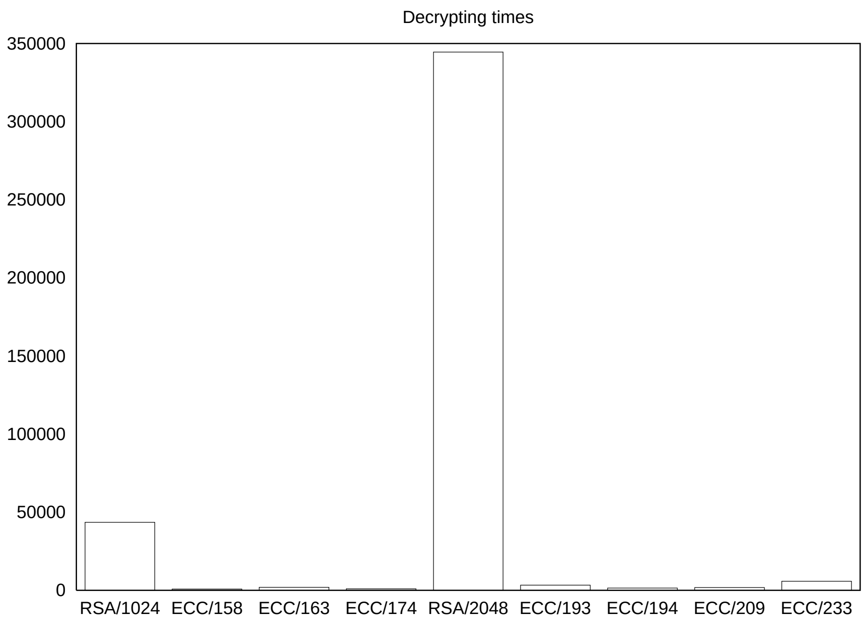

Figure 9.2: Decryption times (in milliseconds)

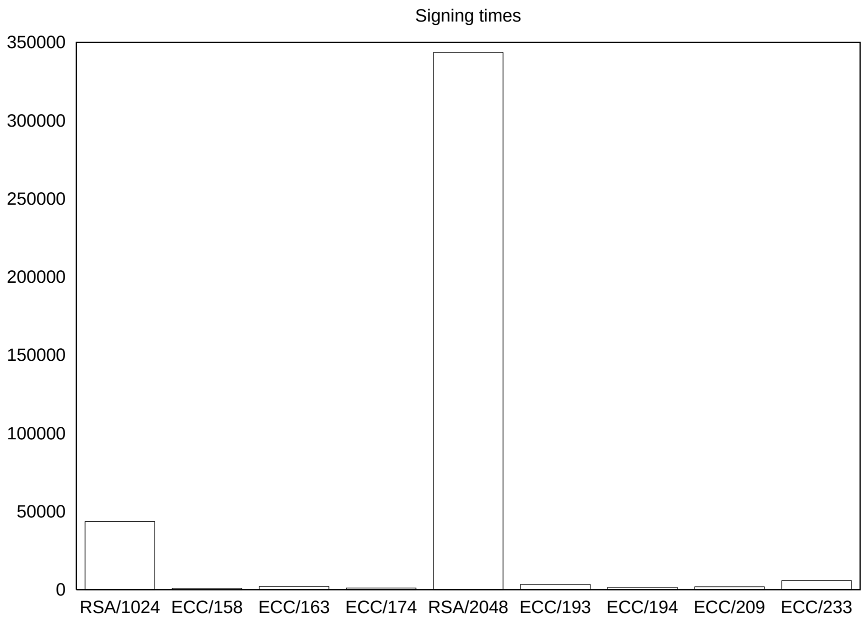

Figure 9.3: Signing times (in milliseconds)

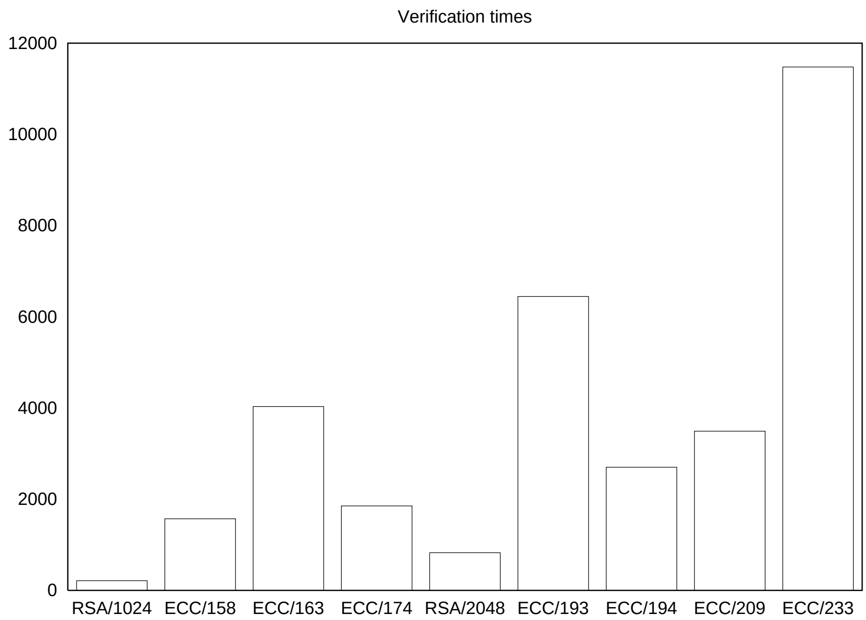

Figure 9.4: Signature verification times (in milliseconds)

All this performance analysis is based on the ECC and RSA implementations described in Chapter 6, so changing the encryption and signing algorithms, using another programming language, using more sophisticated optimizations and algorithms or using real smart cards can change the performance results significantly.

9.2 Security

Chapter 5.2 describes the most common attacks against RSA and elliptic curve cryptosystem. I chose the test parameters listed in Appendix A in a way that it should be hard to succeed in those attacks.

As the Table 5.4 shows, a 1024-bit RSA key has approximately the same strength as a 160-bit ECC key, and it is estimated that it should take 10^{11} MIPS years to break a key of that strength. The table also shows that a 2048-bit RSA key corresponds to a 210-bit ECC key, and the estimated time to break these is 10^{20} MIPS years. This means basically that all the keys used in testing should be cryptographically strong.

The test parameters contain RSA keys with public exponent 3 and with public exponent $2^{16} + 1$, because small public exponents make RSA a bit faster.

There are no supersingular or anomalous elliptic curves in the test parameters. Many of the curves are from standards (NIST [38] and WAP WTLS [62]) and those curves should be secure to use. Some of the curves are generated with curve generation software and they can not be considered secure without further analysis.

As described in Chapter 5.2 the order of the curve E should be sufficiently large, i.e. $\#E > 2^{160}$ bits, to prevent exhaustive search and attacks using the Pohlig-Hellman algorithm. The tested elliptic curves A.5 - A.8, in Appendix A, all have curve order larger than 2^{160}. Elliptic curve A.9 has curve order smaller than 2^{160}, and is thus vulnerable to these attacks. The curve generation software used to generate test curves A.10 - A.12 did not reveal the cofactor of these curves and thus I do not know how large curve orders these curves have.

There are also three attack types not mentioned in Chapter 5.2 which are considered significant on smart card environments: Differential Fault Attacks, Power Analysis Attacks, and Timing Attacks.

The Differential Fault Attack (DFA) has been successful in attacking RSA, DES, and even helps reverse engineering unknown cryptosystems. The basic idea of DFA is the enforcement of bit errors into the decryption or signing process which is done inside the smart card [4]. Then information on the secret key can leak out of the card. For example, in RSA implementations this information can be used to factor the RSA modulus, which is equivalent to computing the secret RSA key.

Kocher et al. have presented attacks based on Simple and Differential Power Analysis (referred to as SPA and DPA respectively) to recover the secret key by monitoring and analyzing the power consumption signals [29, 30]. Such power signals can provide useful side channel information to the attackers.

The central idea of Timing Attacks is to determine a secret parameter from differences between running times needed for various input values [48].

All these three types of attacks can be used to attack both the RSA and elliptic curve cryptosystems.

The fact that the ECC implementation is much more complicated and needs deeper mathematical understanding than the RSA implementation makes it more susceptible for errors and thus diminishes its security.

The Menezes-Vanstone encryption algorithm, described in Chapter 4.6, also has a security problem. A drawback of the method is that if an intruder happens to know x_1 (or x_2), he can then easily obtain x_2 (or x_1). This attack can however be prevented by only sending $(k\alpha\ ,\ y_1)$ [32].

9.3 Space Requirements

In my RSA cryptosystem implementation the public key consists of n and b, where n is the product of two primes p and q, and b is the public exponent. The private key consists of the private exponent a and the modulus n. In the CRT version of RSA the private key consists of the private exponent a and the primes p and q. RSA needs no system parameters.

Table 9.1: Space requirements

	System parameters (bits)	Public key (bits)	Private key (bits)
1024-bit RSA	n/a	1048	2048
1024-bit RSA with CRT	n/a	1048	3072
158-bit ECC	806 (649)	316 (159)	158
163-bit ECC	994 (832)	326 (164)	163
174-bit ECC	886 (713)	348 (175)	174
2048-bit RSA	n/a	2072	4096
2048-bit RSA with CRT	n/a	2072	6144
193-bit ECC	1174 (982)	386 (194)	193
194-bit ECC	986 (793)	388 (195)	194
209-bit ECC	1061 (853)	418 (210)	209
233-bit ECC	1414 (1182)	466 (234)	233

In my ECC implementation the public key consists of the point $\beta = a\alpha$, where α is the generating point of the curve. The private key consists of the field a. The system parameters needed for elliptic curve cryptosystem are the following: Field size, curve parameters a_2 and a_6, generating point α, order of the generating point, and irreducible polynomial (only needed in case of polynomial base elliptic curves).

If point compression as described in Chapter 4.6 is used, the space required for the elliptic curve system parameters and public keys can be reduced. These reduced values are shown in parentheses in Table 9.1.

The values in Table 9.1 and Table 5.6 in Chapter 5 differ in ECC system parameters. These are larger in my implementation than Certicom's results. Certicom's implementation probably uses less parameters than my implementation or has some compressed form to store the system parameters.

Table 9.2 compares the sizes of encrypted 1000-bit messages. The values in parentheses in Tables 9.2 - 9.4 are gained by using point compression. Because point compression is very easy to implement, the comparisons below use these reduced values. Table 9.2 shows that the ECC encrypted message, when point compression is used, is nearly two times longer than the message encrypted with RSA. ECC encryption also takes a longer time, because the encryption of 1000-bit messages with a 163-bit key takes four Menezes-Vanstone encryption operations, while only one RSA encryption is needed for 1000-bit messages.

Table 9.2: Size of the encrypted 1000-bit messages

	Size of encrypted message (bit)	Time required for encryption (ms)
163-bit ECC	2608 (1960)	16074.8
1024-bit RSA	1024	45.8

Table 9.3: Size of the encrypted 500-bit messages

	Size of encrypted message (bit)	Time required for encryption (ms)
163-bit ECC	1304 (980)	8037.4
1024-bit RSA	1024	45.8

Table 9.4: Signature sizes on 1000-bit long messages

	Size of signature (bit)	Time required for signing (ms)
163-bit ECC	326	2009.3
1024-bit RSA	1024	77919.8

Table 9.3 shows that a 500-bit ECC encrypted message, when point compression is used, is almost as long as a 500-bit message encrypted with RSA. ECC encryption however takes a longer time, because the encryption of 500-bit messages with a

163-bit key takes two Menezes-Vanstone encryption operations, while only one RSA encryption is needed for 500-bit messages.

Table 9.4 shows that the ECC signature size is about one third of the signature size achieved with RSA. ECC signing is also considerably faster than RSA signing. All these results in Tables 9.2 - 9.4 are similar to the ones ICSA lists in Tables 5.7 and 5.8.

The size of the message to be signed is not as relevant as it is when encrypting the message, because to keep the size of a signature small, both the RSA and ECC implementations use a one-way hash function to create a hash from an original message, and sign the hash instead of the original message.

Furthermore, as described in Chapter 6, the elliptic curve cryptosystem implementation is more complicated than the RSA implementation and may therefore require more space from the smart card.

9.4 Improvement Proposal and Further Development

To prevent more sophisticated attacks, the message should be encoded before both the elliptic curve signing and encryption operations, and RSA signing and encryption operations. The P1363 standard [25] recommends the EMSA1 encoding to be used with elliptic curve signatures with appendix, and EMSA2 encoding to be used with RSA signatures with appendix. It also recommends EME1 encoding to be used with RSA encryption. The P1363 standard has no recommendation for encoding to be used with elliptic curve encryption, because it does not contain any elliptic curve encryption methods. To make the elliptic curve and RSA implementations more secure, the above encoding methods and some encoding method for elliptic curve encryption should be used.

It is also important to develop countermeasures for the Differential Fault Attacks, Power Analysis Attacks, and Timing Attacks, described in Chapter 9.2. These countermeasures can be either hardware countermeasures on smart cards or software countermeasures in the software implementation.

To speed up ECC and especially RSA, better implementations of the `BigInt` and `SHA1` classes than the ones described in Chapter 6 should be used. As mentioned before, there are some commercial Java Card APIs available which contain classes for big integers and message digests.

To further speed up the implementation there are many algorithms available to make the elliptic curve operations faster. For example to speed up the polynomial multiplication one can use the Fixed Base Windowing algorithm [22].

I hope that in the near future I will have an opportunity to download the implementation described in Chapter 6 to a real smart card and measure the times the elliptic curve and RSA operations take, and also measure the power consumption of those operations.

Chapter 10

Conclusions

The objective of this Master's thesis was to clarify whether elliptic curve cryptography is better suited to be used with smart cards than the nowadays widely used 1024-bit RSA. I have used performance, security, and space requirements as criteria when comparing these two cryptosystems.

From the performance point of view, my implementations of the elliptic curve and RSA cryptosystems show that the elliptic curve cryptosystem is faster than RSA for decrypting and signing messages. On the other hand, RSA is faster when encrypting messages and verifying signatures. In addition the Menezes-Vanstone algorithm causes the encrypted messages to become much longer than with 1024-bit RSA when encrypting long messages. Menezes-Vanstone encryption is, however, efficient if the size of the encrypted message is shorter than (or equal to) two times the key size used, which makes Menezes-Vanstone encryption favorable to encrypt short messages, for example session keys to be used in a private key cryptosystem.

Elliptic curve cryptography signing seems nevertheless superior to RSA. Nyberg-Rueppel signing is faster than RSA signing and the size of a signature is much smaller than the size of a 1024-bit RSA signature.

Based on published research, elliptic curve cryptosystems can be considered as secure as RSA cryptosystems. When using elliptic curve cryptosystem though one must remember to use secure parameters, i.e. the curve should not be supersingular or anomalous and the curve order should be large. Of course it is also important to use secure parameters when using RSA, this usually is achieved by using a good random number generator.

In addition, the fact that the elliptic curve cryptosystem implementation is much more complicated and requires deeper mathematical understanding than the RSA implementation, makes it more susceptible to errors and thus diminishes its security.

There is also a security flaw in the Menezes-Vanstone algorithm. The vulnerability is described in Chapter 9.2, but it can be prevented by only sending $(k\alpha\ ,\, y)$.

The space required in the smart cards is a very important factor, because there is only restricted space available for storing the secret keys. Elliptic curve cryptosystems need much smaller keys than RSA to achieve the same level of security. Despite the fact that elliptic curve cryptosystem needs system parameters and RSA does not, the situation does not change, because even if system parameters were stored in the smart card, the space required for ECC system parameters and ECC secret key would be smaller than the space required for RSA secret key. However, the ECC implementation is more complicated than an RSA implementation and it may use up more space from the smart card.

Of course, usually there is no need to store the system parameters in the restricted memory space of the smart card. Instead, the system parameters can be stored in the memory of the mobile phone, PDA (Personal Digital Assistant) or other device which the smart card is connected to.

I believe that if the elliptic curve cryptosystem implementation is made with care and proper optimizations are used, ECC will be at least as good as RSA. When working in a constrained environment where longer keys just cannot fit into, elliptic curves are probably better than RSA. On the other hand, if there are no performance constraints, RSA might be the better choice. If security over decades is a concern, the RSA cryptosystem is better than ECC, because an ECC implementation is complicated and thus prone to errors and RSA has been under public scrutiny for a longer time.

Bibliography

[1] American National Standards Institute. *Working Draft: American National Standard X9.63, Public Key Cryptography For The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, January 1999.

[2] K. Araki, S. Miura, and T. Satoh. Overview of elliptic curve cryptography. In *International Workshop on Practice and Theory in Public Key Cryptography*, pages 1–14, 1998.

[3] Elwyn Berlekamp. *Algebraic Coding Theory*. McGraw-Hill, 1968.

[4] Ingrid Biehl, Bernd Meyer, and Volker Muller. Differential fault attacks on elliptic curve cryptosystems. In *CRYPTO'2000: Proceedings of Crypto*, pages 131–146. Springer, 2000.

[5] Ian Blake, Gadiel Seroussi, and Nigel Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, first edition, 1999.

[6] J. Buchmann and V. Muller. *Computing the number of points of elliptic curves over finite fields*, July 1991. presented at International Symposium on Symbolic and Algebraic Computation, Bonn.

[7] William J. Caelli, Edward P. Dawson, and Scott A. Rea. PKI, elliptic curve cryptography, and digital signatures. *Computers & Security*, 18(1):47–66, 1999.

[8] Certicom Corp. *ECC Standards*. http://www.certicom.com/research/ECC_standards.html.

[9] Certicom Corp. *Current Public-Key Cryptographic Systems*, April 1997. http://www.certicom.com/research/wecc2.html.

[10] Certicom Corp. *The Elliptic Curve Cryptosystems for Smart Cards*, May 1998. http://www.certicom.com/research/wecc4.html.

[11] Certicom Corp. *ECC in X.509*, August 1999. http://www.secg.org/drafts.htm.

[12] Certicom Corp. *Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography*, September 1999. http://www.secg.org/drafts.htm.

[13] Certicom Corp. *Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters*, October 1999. http://www.secg.org/drafts.htm.

[14] Certicom Corp. *Crypto news: ECC2K-108 Challenge Solved*, April 2000. http://www.certicom.com/research/cc2k108.html.

[15] Certicom Corp. *Press release: Robert Harley and Team Win $10,000 Prize in Certicom's ECC Challenge*, April 2000. http://www.certicom.com/news/00/apr1700.html.

[16] CWI, Amsterdam. *Press release: Security of E-commerce threatened by 512-bit number factorization*, August 1999. http://www.cwi.nl/ kik/persb-UK.html.

[17] W. Diffie and M.E. Hellman. Multiuser cryptographic techniques. In *Proceedings of AFIPS National Computer Conference*, pages 109–112, 1976.

[18] S. Dussé and B. Kaliski. A cryptographic library for the Motorola DSP56000. In *EUROCRYPT'90: Advances in Cryptology: Proceedings of EUROCRYPT*, pages 230–244. Springer, 1990.

[19] G. Frey and H. Ruck. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62:865–874, 1994.

[20] S. Gao and H.W. Lenstra. Optimal normal bases. *Designs, Codes and Cryptography*, 2:315–323, 1992.

[21] Scott B. Guthery and Timothy M. Jurgensen. *Smart Card Developer's Kit.* Macmillan Technical Publishing, first edition, 1998.

[22] Darrel Hakerson, Julio Lopez Hernandes, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. In *CHES 2000*, pages 1–24. Springer, 2000.

[23] Uwe Hansmann, Martin S. Nicklous, Thomas Schäck, and Frank Seliger. *Smart Card Application Development Using Java.* Springer, first edition, 2000.

[24] Jr. H.W. Lenstra. Factoring integers with elliptic curves. *Annuals of Mathematics*, (126):649–673, 1987.

[25] IEEE. *IEEE P1363 / D13 (Draft Version 13), Standard Specifications for Public Key Cryptography*, November 1999. http://grouper.ieee.org/groups/1363/.

[26] P. Ivey, S. Walker, J. Stern, and S. Davidson. An ultra-high speed public key encryption processor. In *Proceedings of IEEE Custom Integrated Circuits Conference, Boston*, pages 19.6.1–19.6.4, 1992.

[27] Donald E. Knuth. Seminumerical algortihms. *The Art of Computer Programming*, 2, 1969.

[28] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation / American Mathematical Society*, 48(177):203–209, 1987.

[29] P. Kocher, J. Jaffe, and B. Jun. *Introduction to Differential Power Analysis and Related Attacks*, 1998. http://www.cryptography.com/dpa/technical.

[30] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO'99: Proceedings of Crypto*, pages 388–397. Springer, 1999.

[31] Chuck McManis. *Pegwit (v8)*.
http://ds.dial.pipex.com/george.barwood/v8/pegwit.htm.

[32] A.J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[33] A.J. Menezes, T. Okomoto, and S. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39:1639–1646, 1993.

[34] A.J. Menezes and Scott A. Vanstone. Elliptic curve cryptosystems and their implementation. *Journal of Cryptology*, 6:209–224, 1993.

[35] R.C. Merkle. Secure communication over insecure channels. *Communications of the ACM*, 21(4):294–299, 1978.

[36] V.S. Miller. Use of elliptic curves in cryptography. In *CRYPTO'85: Proceedings of Crypto*, pages 417–426. Springer, 1985.

[37] R. Mullin, I. Onyszchuk, S.A. Vanstone, and R. Wilson. Optimal normal bases in $GF(p^n)$. *Discrete Applied Mathematics*, 22:149–161, 1988/89.

[38] National Institute of Standards. *Recommended Elliptic Curves for Federal Government Use*, July 1999.

[39] Randall K. Nichols. *ICSA Guide to Cryptography*. Computing McGraw-Hill, first edition, December 1999.

[40] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, 24:106–110, 1978.

[41] J. Pollard. Monte carlo methods for index computation mod p. *Mathematics of Computation*, 32:918–924, 1978.

[42] W. Rankl and W. Effing. *Smart Card Handbook*. John Wiley & Sons, 1997.

[43] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[44] M.J.B. Robshaw and Yiqun Lisa Yin. Elliptic curve cryptosystems. http://www.rsasecurity.com/rsalabs/ecc/elliptic_curve.html, June 1997.

[45] Michael Rosing. *Implementing Elliptic Curve Cryptography*. Manning Publications, first edition, 1999.

[46] RSA Laboratories. *PKCS #1 v2.1 RSA Cryptography Standard*, September 1999. http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html.

[47] T. Satoh and K. Araki. Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. 1997.

[48] Werner Schindler. A timing attack against RSA with the Chinese Remainder Theorem. In *CHES 2000*, pages 110–125. Springer, 2000.

[49] Bruce Schneier. *Applied Cryptography*. John Wiley and Sons, Inc., 1994.

[50] Bruce Schneier et al. Elliptic curve public-key cryptography. *Crypto-Gram*, November 1999.

[51] Richard Schroeppel, Hilarie Orman, Sean O'Malley, and Oliver Spatscheck. Fast key exchange with elliptic curve systems. In *CRYPTO'95: Proceedings of Crypto*, pages 43–56. Springer, 1995.

[52] G.J. Simmons. *Contemporary Cryptology - The Science of Information Integrity*, 1992.

[53] N. Smart. Announcement of an attack on the ECDLP for anomalous elliptic curves. Hewlett Packard Laboratories, U.K., 1997.

[54] Jerome A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In *CRYPTO'97: Proceedings of Crypto*, pages 357–371. Springer, 1997.

[55] Douglas R. Stinson. *Cryptography: theory and practice*. CRC Press, first edition, 1995.

[56] Sun Microsystems. *Java Card 2.1 Platform*, November 1999. http://www.java.sun.com/products/javacard/javacard21.html.

[57] Sun Microsystems. *Java Card™ 2.1 Development Kit User's Guide*, November 1999.

[58] Herman te Riele. *Factorization of a 512-bits RSA key using the Number Field Sieve*. CWI, August 1999. ftp://ftp.cwi.nl/pub/herman/NFSrecords/RSA-155.

[59] Simone van der Hof. Digital signature law survey. http://cwis.kub.nl/ frw/people/hof/ds-lawsu.htm, February 2000.

[60] P. van Oorschot and M. Wiener. Parallel collision search with cryptoanalytic applications. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, Virginia*, pages 210–218, 1994.

[61] M.J. Wiener and R.J. Zuccherato. Faster attacks on elliptic curve cryptosystems. September 1998.

[62] Wireless Application Forum. *WAP WTLS, Wireless Application Protocol, Wireless Transport Layer Security Specification*, February 1999. http://www.wapforum.org.

Appendix A

Test Parameters

RSA Parameters Used in Tests

Table A.1: 1024-bit RSA key, public exponent is 3

Prime q	`c73c6da1 93cf8b6e a8f68a86 63843443 1018f745 32188142`
	`d5feab27 c84846ac a90f7c21 c26cb3e3 cf4d579b 5314d926`
	`a7b54821 6902aa0d 731e39aa be2ff1d3`

Prime p	`ed7f3478 4ba6e62d 1fc2638a 30b25784 952aaf68 ca9ffe9b`
	`77509e88 5b67456e 58abd81c 81029c18 012addcd de41dae6`
	`42e8034a e5e98644 e433d6f2 d2e0e7b1`

CRT coefficient $qInv$	`9a716362 52865ace 4d22dcaa 42d61e5f def61fbb d508ef23`
	`20b1a09f 75dfe4bc c449e424 c5f93a9e 122f208e dc7acbcf`
	`027686e0 1823f480 5bacff56 36dce408`

q's exponent dQ	`84d2f3c1 0d35079f 1b4f0704 425822d7 6010a4d8 cc10562c`
	`8ea9c76f dadad9c8 70b4fd6b d6f32297 df88e512 37633b6f`
	`1a78dac0 f0ac715e 4cbed11c 7ecaa137`

p's exponent dP	`9e54cda5 87c49973 6a8197b1 75cc3a58 6371ca45 dc6aa9bc`
	`fa35bf05 9244d8f4 3b1d3abd ab5712ba ab71e933 e9813c99`
	`81f00231 ee9baed8 98228f4c 8c95efcb`

Modulus n	`b8d5f14e de78463f e5d797e9 42ba91fe 2989f1ea 2a2535b6`
	`fefc67c3 7de9ea6f 7e9d07aa 79335de9 c7bcc7cc fd752735`
	`38efa899 c37ff9e0 531136fb 973c7d94 12f4f4e7 60f0e992`
	`5c2fcb79 c755db86 73c40269 f3b5c64b dad4e941 af74b072`
	`809fa1a8 1ad940f4 d715e4e8 414c3c12 1d55fddc f106bf4a`
	`59daaa73 70fb97e3`

Private exponent a	1ece52e2 7a69610a a64e9951 8b1f1855 06ec52fc 5c5b88f3 d52a114b 3fa6fc67 ea6f8147 14333a51 a14a214c d4e8dbde 3427f16e f5eaa9a5 632d8929 ee8a14ed ba5ee322 403f13fe 6de924e6 ddda8d4a 77c00f49 fe7f8bbc eceb9a98 41f630b9 3fd0b791 a3e6fd7e d66f9d3f d7fe4156 331ec868 1b046d29 55c16ef8 faa71fbb
Public exponent b	3
Reference	Created with SSH's ssh-keygen script See: http://www.ssh.com/products/ssh/

Table A.2: 1024-bit RSA key, public exponent is 65537

Prime q	cf6a53a7 a703db00 d1b4f7d1 83c7d1f7 e18b8dd6 9cd276be d96d084d 6d218348 740ef997 8719fba7 9a2496e5 93f52664 8f564f5e 10826cef 95284536 2ea80e1b
Prime p	eb835925 d5ceda49 0208cf0a b6fc806a a0607464 29a3f2cb 81ced897 911d0556 7c7219cf 8494b746 7a13fe93 7e8649a8 4efc0642 4cf7ae79 ce6b1d4f 1627da03
CRT coefficient $qInv$	50d2873c 7ff7b293 1958fdd8 db8f1c16 41e2bcdb 78e090fe a6e10927 3cc4e832 a129d305 7ac8a92f b2a4f69b 5d768bb5 5e1efea3 587bad9f 373d5eb2 4c588443
q's exponent dQ	5546219e 8bcbc9fe 4c395aa9 88c45e4e cb2bd4e4 a714e03d 16a6fd02 d81b2f1e 5a98cd84 bec6afef f82f92da 503a39ce 9e1f9e01 ce47d101 af111f63 674b3081
p's exponent dP	96941e75 90647730 62ba9b15 f0b534ad f7e873c2 6ed54231 57c796b8 21b1fa6d b7978465 2811a5b4 cbece8c2 090d771e c39163b7 145af015 35234b53 fb88735d
Modulus n	bed1066e 53c97153 cdb19031 c3d43d54 bc1ad361 479f9fc0 37bfe846 dc4162de 5245cbc3 e47d5971 d2ce9038 44be3b05 f9e0d952 0710785f 7b88c169 bf9bc472 8246c612 9029f30d cc46a18d 4b1a10c2 daa1da96 64d3d2cd 2f1ec10b f1f240e8 53b76f08 b03cf8a1 3fd744ab cb8ebccf 5b89fa5d 7723ba44 4c6ae983 20182851
Private exponent a	1b2d1e45 db4ab433 13ccbcb5 21bcaf69 75e03a32 6a26d549 3a876b4c f59f4860 a7db5c97 248a7d11 3022b98e 92027d2f 2f89f304 483b9e71 36e10b6e da0514cd 5d244196 ec176a40 12bc6702 7dc995c8 c209a56b fdaf89c7 79f19f81 5cb1c01c 3a49951e 5341ecb2 5a6834c5 3c1a5568 83b89ce3 f917867c e0aef02b ba83329b
Public exponent b	10001
Reference	Created with SSH's ssh-keygen script See: http://www.ssh.com/products/ssh/

Table A.3: 2048-bit RSA key, public exponent is 3

Prime q	e5025fe9 c61cf464 3760a4cd 1513aa0a 4528b909 ac1ee4b1 8c8eed0b 443fdbd1 866f34a5 ccfd8746 df0e13da 0c64dfd7 a82188f1 84feb7a3 446476f9 c1a3f601 ff0b42e5 1905ee4d 9460be1c 91268ed2 d16ebb65 4255767d 30285569 14d5cd0c 2943b286 30c6c476 0e1cd4f2 c64b4ef4 1a53f239 7daff7ab 9fbb91aa 8b94606f
Prime p	ed03f124 a3ce4def 683cb846 408276ff a4e0ffee 30832e69 4d318c0a e10753ac 8040f551 59373c26 a8f4f839 f5f83449 581fc535 2c8fecd4 f6fe9fd1 d0e67351 77fa4eb6 30836fba 11e30e2f 40416689 5e27d503 1a35d167 3e33ba74 9e01e1bb 9a1845d9 cf5dffac fa3da174 6126dac2 8988601d d7904f1b 05e58215 14f4dd81
CRT coefficient $qInv$	8934ba05 d1f17542 efc6e671 3720e2b4 815ca1e3 928799a4 7714d119 4de1df58 ec7d1b55 36d95b3c 2813cd47 4b455731 e6230b1b a23a8fca 520c6e93 0ce2ea9b 08c5767e 503e4453 7d727822 db2e8489 ffe085da d0335054 d5cf9a81 95861525 d3f9dc1e f9c76f89 fd3b20ca d7e1f0b0 f0ea8de8 64a5dce6 c410ff62 9f762ff8
q's exponent dQ	98ac3ff1 2ebdf842 cf95c333 6362715c 2e1b2606 72bf4321 085f48b2 2d7fe7e1 044a2319 3353af84 94b40d3c 08433fe5 1ac105f6 58a9cfc2 2d984f51 2bc2a401 54b22c98 bb594989 0d95d413 0b6f09e1 e0f47cee 2c38f9a8 cac58e46 0de3de08 1b827704 208482f9 5ebde34c 843234a2 bc37f6d0 fe754fc7 bfd2611c 5d0d959f
p's exponent dP	9e02a0c3 17dede9f 9ad32584 2b01a4aa 6deb5549 75acc99b 88cbb2b1 eb5a37c8 5580a38b 90cf7d6f 1b4dfad1 4ea57830 e56a8378 c85ff338 a4a9bfe1 35eef78b a55189ce cb024a7c 0becb41f 802b99b0 e96fe357 66ce8b9a 2977d1a3 14014127 bc102e91 34e9551d fc29164d 9619e72c 5bb04013 e50adf67 5943ac0e 0df893ab
Modulus n	d406b98a 8f4fc1ab ab27d40a b2a9d845 d91a257b a9873149 6830a196 61f3d48f 84134447 ec5c1c7b d182373a 11984724 1a6604c3 089ec715 87e7a785 0baddf20 9ec7a237 693559d8 abebc57a 3fd4d348 3c025649 273050ad 37be1c95 ab0c7c3e 8df30c3c 64486274 a1edf652 352c0217 39015bc3 228d7a9f c09e26de 7c306643 fbf21c61 521864ec d60f781e 3152cc7b 60fb0fc9 25605b40 5f4196f1 4bdcb071 e5ae0ebd f509b389 1ecad057 454dd09a 4d586cca 9f99f83f 92ada291 f36ace0b c30dfec8 de19eb81 156aeb6c 53227a61 42b30079 24470209 98e4f6bd 4360d1b2 b62ef8dd 02ae85c8 a813e26c 717c2ce7 a6b3eb73 ad0698b8 ce28897f 02d06aef

Private exponent a	23567441 c28d4af1 f1dbf8ac 731c4eb6 4ed9b0e9 f1968836 e6b2c599 10534e17 eb588b61 520f5a14 a2eb0934 58440bdb 59bbab75 d6c5212e 41514696 2c9cfa85 6fcbf05e 9188e44e c751f63f 0aa3788c 0a0063b6 dbdd62c7 894a5a18 f1d76a0a 6cfdd75f 660c1068 c5a7a90d b38755ae 898039f5 db17946f f56fb125 14b2bbb5 b1a74c8d d15cdb19 89130481 cf4a1c92 e9283922 e1750c06 40eada4f 3118ead3 4fd4fb76 77ce2804 99214b60 8b281f69 8cd92fc5 fd01e34b e3e1c1f6 657abb74 0cac1232 436d6ce9 92868530 159f162b 832f6802 cbf49f06 31c17bcf ed6c307c 7dcdd56a 2b16f59b 9a9ee756 3701ab32 d5ce9984 b94bb853 06c13e4a 9061322b
Public exponent b	3
Reference	Created with SSH's ssh-keygen script See: http://www.ssh.com/products/ssh/

Table A.4: 2048-bit RSA key, public exponent is 65537

Prime q	da91a6cb 3ff7b6e6 96d41a4a df4ff3fe 24add282 53d22cef 4b23caad f78c2cad 02c97077 e67c4496 f20f00c5 d25ef945 7b7c0c73 0dfd9b77 f1a3692c bd6428cd d50a60e5 117b2937 68bf0bc3 712675a4 605e0ded 15d4362b c0d7776b 713ab0f6 15f94c2f 0fe4e1c2 7a400e02 9d20fd76 93b86268 75ad9241 527df191 2b054113
Prime p	f0efc19f f4313ca4 a98d5623 643437ff df2916e9 f656e003 d43e182b 1798da55 bb4dd245 397ab70d 2b5a6e04 b7de8d80 b23c334f 313f1a61 6d5479ed 86194982 95f2a011 5e3f4814 7a0408e4 f1310ced cc4d3c5d 5de71763 0df74a34 b16e6174 feb9aff2 a0a63460 04ff4aca 6047f929 c7556828 f6314c01 861d2796 4f5c825b
CRT coefficient $qInv$	afc995c6 0a7c8467 93597caf f491ff81 fe910383 a47f9c79 32acf75e a53b469d 21590a46 ac803e3e 73782964 76872816 88a7651a 91eeea45 58193bb6 8f309c9d 4658104f b02dea4f 32f58ef7 5cb9664a a8bd423d edd7f580 d14e8c4e 77a18eec 7e2e6014 9cf9a9d5 129154a8 440e2280 42a2ddcd 26ea45ef a64e9373 550d3160
q's exponent dQ	6fa32970 7311d187 e1ce2047 d1dc02a1 f2961e09 974aa8cc e8b23559 ba310cb8 a9cdbdf6 5ff1650f 3c3fd183 393a6d53 c208d04b 0c005722 6277b6c6 b54dad51 c4d8f238 ce3efc04 632a0196 27fb1ee7 809949bf a900a552 9fc2ee2a 20c17b95 960fa98b f8304de7 898da21a 4fab840a c6d4ccde d20c7fc1 81a2f679 d16173a5

p's exponent dP	`19bed4e6 ddb6212f cb3a6218 f80d286a bc12ecf2 e532d54f`
	`336fcb2f 579f2abd 7c7d4d59 be51a19b d3a51f1a 7bf75911`
	`867a2e55 6902c033 e6f5deed 85a5d018 1723ccfc ee203ef8`
	`892d4332 0b9fdd2c cfe6f4eb 592c6770 5b3eb07c 3ef5778e`
	`83e59daa 84861e14 a094130e db08f5d4 040bd94c 7f3a374f`
	`f65fc92a 23a2d3ff`

Modulus n	`cdb53fa9 a36ba3d3 c974c269 56f0c8fb b514c5d0 3fb60a91`
	`385880d8 1056dbaf 433d8368 8c057931 ad743d2a 46ab840a`
	`04b9a050 cf862c04 972779a6 8126d069 982daee9 67cfe61a`
	`2dca2335 3bfa7ef9 c4b17fda a5a65a8f 6a652db8 0358ddc7`
	`413cc0c9 10cb1347 5a8f34f4 b27283d4 29b487d3 16fb6f9a`
	`65f012a3 e0c13d37 a85efc3d 3a5d39b8 827b1284 d4ae2d78`
	`e48e32ab 6312e44f 1196b75a b725c9a4 25db15db 6dfd4ad4`
	`4b4978c6 b70f0338 43524a34 36e160ac 7dcc805b 81ea2fc9`
	`c8678a30 fb632685 75de813c d2f9975f f2317363 89580d3e`
	`e72e9ce3 763877ac 0f4fa4af 7de4aea1 32c7cd8b cad4d3a9`
	`93d501cd 016fd943 3707b18d 35bdc7c1`

Private exponent a	`089e5579 fe849836 802430d4 ba4c6a70 6b04e148 86fad20b`
	`85672f49 9e2758e4 372457b4 526c5db6 2bc25233 b82afb52`
	`04fdf245 0e328164 e78e557c ab52b7ea 3a605a77 7cd578f5`
	`e5f83a75 8b96b23b ec91d386 b743db12 2ebca055 86d6f1f5`
	`80b1926d 55936db2 6bd68971 f3f72ab8 13f41902 b826da40`
	`abfe8abf 19f6a0f3 aed63050 38902e41 f1e23dc3 c591a59d`
	`cc68960a 1b5e2de4 58cbfdfc b8f4bb32 d26b58ba 6886e425`
	`2c8a4c83 15767b6d 807ee80f 8de4aba3 d57e7d17 53a0028c`
	`85d2137f a88424a4 32636b65 f6ba64da 9956a1fc e321d953`
	`92354a7e 646aa2f6 2b7c1c5c 9bff6b0d 6a32599b 8ae3093d`
	`aa06d2b4 f20530c9 4486edce c95722f7`

Public exponent b	`10001`
Reference	Created with SSH's ssh-keygen script See: http://www.ssh.com/products/ssh/

Elliptic Curves Used in Tests

Table A.5: 163-bit polynomial base elliptic curve

Field size	163
Irreducible polynomial	$x^{163} + x^7 + x^6 + x^3 + 1$
Elliptic curve E	$y^2 + xy = x^3 + a_2x^2 + a_6$, over $GF(2^{163})$
Parameter a_2	1
Parameter a_6	1
Generating point α	00000002 fe13c053 7bbc11ac aa07d793 de4e6d5e 5c94eee8 00000002 89070fb0 5d38ff58 321f2e80 0536d538 ccdaa3d9
Order of α	00000004 00000000 00000000 00020108 a2e0cc0d 99f8a5ef
Cofactor K	2
Reference	NIST [38], WAP WTLS [62]

Table A.6: 163-bit polynomial base elliptic curve

Field size	163
Irreducible polynomial	$x^{163} + x^8 + x^2 + x + 1$
Elliptic curve E	$y^2 + xy = x^3 + a_2x^2 + a_6$, over $GF(2^{163})$
Parameter a_2	00000007 2546b543 5234a422 e0789675 f432c894 35de5242
Parameter a_6	00000000 c9517d06 d5240d3c ff38c74b 20b6cd4d 6f9dd4d9
Generating point α	00000007 af699895 46103d79 329fcc3d 74880f33 bbe803cb 00000001 ec23211b 5966adea 1d3f87f7 ea5848ae f0b7ca9f
Order of α	00000004 00000000 00000000 0001e60f c8821cc7 4daeafc1
Cofactor K	2
Reference	WAP WTLS [62]

Table A.7: 193-bit polynomial base elliptic curve

Field size	193
Irreducible polynomial	$x^{193} + x^{15} + 1$
Elliptic curve E	$y^2 + xy = x^3 + a_2x^2 + a_6$, over $GF(2^{193})$
Parameter a_2	00000000 17858feb 7a989751 69e171f7 7b4087de 098ac8a9 11df7b01
Parameter a_6	00000000 fdfb49bf e6c3a89f acadaa7a 1e5bbc7c c1c2e5d8 31478814
Generating point α	00000001 f481bc5f 0ff84a74 ad6cdf6f def4bf61 79625372 d8c0c5e1 00000000 25e399f2 903712cc f3ea9e3a 1ad17fb0 b3201b6a f7ce1b05
Order of α	00000001 00000000 00000000 00000000 c7f34a77 8f443acc 920eba49
Cofactor K	2
Reference	Certicom [13]

Table A.8: 233-bit polynomial base elliptic curve

Field size	233
Irreducible polynomial	$x^{233} + x^{74} + 1$
Elliptic curve E	$y^2 + xy = x^3 + a_2x^2 + a_6$, over $GF(2^{233})$
Parameter a_2	0
Parameter a_6	1
Generating point α	00000172 32ba853a 7e731af1 29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126 000001db 537dece8 19b7f70f 555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3
Order of α	00000080 00000000 00000000 00000000 00069d5b b915bcd4 6efb1ad5 f173abdf
Cofactor K	4
Reference	NIST [38]

Table A.9: 158-bit optimal normal base elliptic curve

Field size	158
Elliptic curve E	$y^2 + xy = x^3 + a_2x^2 + a_6$, over $GF(2^{158})$
Parameter a_2	0
Parameter a_6	2aaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
Generating point α	1c8f0f0b cdc2ecbb 4ba01987 7c0a4003 a85a46fd 11f25b0a 47f14158 ce27d93f 2b7f80e1 52be9749
Order of α	10000000 00000000 00003963 014ffc9a 4d192e17
Cofactor K	4
Reference	Rosing [45]

Table A.10: 174-bit optimal normal base elliptic curve

Field size	174
Elliptic curve E	$y^2 + xy = x^3 + a_2x^2 + a_6$, over $GF(2^{174})$
Parameter a_2	00000d34 d34d34d3 4d34d34d 34d34d34 d34d34d3 4d34d34d
Parameter a_6	00001e79 e79e79e7 9e79e79e 79e79e79 e79e79e7 9e79e79e
Generating point α	00001ae3 888995be 6ae8481a 9c271d1a 8d8c865a feba1fec 00001e74 888ae6f8 4c3fd36f e44d3448 fdac7ac9 a94e5fc0
Order of α	0017a070 ad8c3efa 3f7c6e73 a56efb66 e6b35a81
Reference	Created with George Barwood's software See: http://ds.dial.pipex.com/george.barwood/crypto.htm

Table A.11: 194-bit optimal normal base elliptic curve

Field size	194
Elliptic curve E	$y^2 + xy = x^3 + a_2x^2 + a_6$, over $GF(2^{194})$
Parameter a_2	0
Parameter a_6	00000002 aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
Generating point α	00000003 246aefce 66d36ee7 e637f052 590453df 4288f3f1 6415fed3 00000002 1613ded8 592317c3 62b909ef 514b4341 14fe5934 c80948c2
Order of α	000480fe 78298529 e3bec39a 2b893bb8 e338e205 543ff703
Reference	Created with George Barwood's software See: http://ds.dial.pipex.com/george.barwood/crypto.htm

Table A.12: 209-bit optimal normal base elliptic curve

Field size	209
Elliptic curve E	$y^2 + xy = x^3 + a_2x^2 + a_6$, over $GF(2^{209})$
Parameter a_2	00004a49 49292524 a4949292 524a4949 292524a4 94929252 4a494929
Parameter a_6	00014368 6d0da1b4 3686d0da 1b43686d 0da1b436 86d0da1b 43686d0d
Generating point α	00011f19 d44866ea 8b7bc073 519562b8 336d0b1d 5463c835 03eb2d4c 0000f5db 57c9ef31 e227ef9f 32c79db3 43d4156f 26581e15 11bdf74c
Order of α	0000003f a08f2942 1cd4c0de b1f51067 22b50813 44c32c78 6953e21f
Reference	Created with George Barwood's software See: http://ds.dial.pipex.com/george.barwood/crypto.htm