Secure key establishment for low power devices using ECC

Amol Dabholkar

2005

Amol Dabholkar. (2005). Secure key establishment for low power devices using ECC.
Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/2445>

<https://doi.org/10.32657/10356/2445>

Nanyang Technological University

Downloaded on 26 Apr 2023 21:45:32 SGT

a357894

SECURE KEY ESTABLISHMENT FOR LOW POWER DEVICES USING ECC

AMOL DABHOLKAR

SCHOOL OF COMPUTER ENGINEERING

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirements for the degree of
Master of Engineering.

2005

TK
5103.2
A523
2005

ACKNOWLEDGEMENT

I would like to thank my supervisor Dr. Yow Kin Choong for all his help during the course of this project. His suggestions and guidance, which covered a wide range of topics from algorithms and protocols to report writing, research and conference papers, has been invaluable.

I would also like to thank Dr. Tan Chik Ho from EEE, NTU; for the discussions I had with him on ECC.

Finally I am grateful to the lab technicians of the Database Technology Lab, NTU, for all their help in my project.

TABLE OF CONTENTS

ACKNOWLEDGEMENT.....	I
LIST OF FIGURES.....	V
LIST OF TABLES.....	VII
ABSTRACT.....	VIII
1 INTRODUCTION.....	- 1 -
1.1 Wireless devices and the need for security	- 1 -
1.2 Contributions of this thesis	- 3 -
1.2.1 <i>Implementation of ECC on PDAs</i>	- 3 -
1.2.2 <i>A new authentication and key exchange protocol</i>	- 3 -
1.3 A brief overview of Cryptography.....	- 3 -
1.3.1 <i>Symmetric Key cryptography</i>	- 4 -
1.3.2 <i>Public Key cryptography</i>	- 5 -
1.4 Security of Public Key Cryptosystems	- 7 -
1.4.1 <i>The Integer factorization problem</i>	- 7 -
1.4.2 <i>Discrete Logarithm Problem</i>	- 7 -
1.4.3 <i>Elliptic Curve Discrete Logarithm Problem</i>	- 8 -
1.5 The advantages of Elliptic Curve Cryptography	- 10 -
1.6 Summary	- 13 -
2 ELLIPTIC CURVES – INTRODUCTION AND LITERATURE REVIEW	- 14 -
2.1 Some basic notations.....	- 14 -
2.2 Elliptic curves over real numbers	- 16 -
2.3 Fields of Odd characteristics F_p	- 19 -
2.4 Finite fields of characteristic 2 (F_2^n).....	- 20 -
2.5 Polynomial Basis	- 22 -
2.6 Normal basis	- 23 -
2.7 Subfield base.....	- 24 -
2.8 Security considerations	- 25 -
2.8.1 <i>Attacks on ECC</i>	- 25 -
2.9 Summary	- 28 -
3 IMPLEMENTATION OF ECC ON A PDA	- 29 -
3.1 A Java based implementation	- 29 -
3.2 A C based implementation.....	- 30 -
3.2.1 <i>Polynomial base implementation</i>	- 31 -

3.2.2	<i>Optimal Normal Basis Implementation</i>	- 31 -
3.3	Multiplication	- 31 -
3.4	Inversion	- 33 -
3.5	Addition and Subtraction	- 35 -
3.6	Embedding points	- 35 -
3.7	Pre-Computational costs	- 36 -
3.8	Elliptic point multiplication	- 37 -
3.8.1	<i>Non Adjacent Form</i>	- 39 -
3.8.2	<i>NAF with pre-computation</i>	- 41 -
3.9	ECDSA (Elliptic Curve Digital Signature Algorithm)	- 46 -
3.9.1	<i>Discussion of the protocol</i>	- 46 -
3.9.2	<i>CertModule Application</i>	- 49 -
3.9.3	<i>ECDSA Verification on a PDA</i>	- 51 -
3.10	Summary	- 53 -
4	AUTHENTICATION AND KEY EXCHANGE PROTOCOLS	- 54 -
4.1	MAKEPs based on public key cryptography	- 55 -
4.1.1	<i>Diffie-Hellman key exchange protocol</i>	- 55 -
4.1.2	<i>The Massey-Omura key exchange protocol</i>	- 56 -
4.1.3	<i>Menezes-Qu-Vanstone key agreement scheme</i>	- 57 -
4.2	A Generic MAKEP architecture	- 59 -
4.3	MAKEPs for constrained devices	- 60 -
4.4	Case studies of MAKEPs for constrained systems	- 62 -
4.4.1	<i>Jacobsson-Pointcheval protocol</i>	- 62 -
4.4.2	<i>Server Specific MAKEP</i>	- 65 -
4.4.3	<i>Linear MAKEP</i>	- 68 -
4.4.4	<i>Client Server MAKEP</i>	- 71 -
4.5	A New MAKEP Protocol	- 73 -
4.5.1	<i>Protocol Description</i>	- 73 -
4.5.2	<i>Protocol Weaknesses</i>	- 76 -
4.5.3	<i>A More Secure Version</i>	- 78 -
4.6	Summary	- 80 -
5	IMPLEMENTATION AND EVALUATION OF PROTOCOLS	- 82 -
5.1	AES Implementation	- 82 -
5.1.1	<i>Description</i>	- 82 -
5.1.2	<i>Creation of AES keys from Key data</i>	- 84 -
5.2	Implementation of our algorithm	- 84 -
5.3	Implementation of the Client Server MAKEP	- 86 -
5.3.1	<i>The El Gamal Encryption protocol</i>	- 86 -
5.4	Comparison and Discussion of results	- 89 -
5.5	Peer-to-Peer version	- 96 -
5.5.1	<i>Protocol Description</i>	- 96 -
5.5.2	<i>Protocol Timing Analysis</i>	- 99 -
5.6	Other related work	- 101 -

5.6.1 Motivation - 102 -

5.6.2 The Secure Remote File Browsing Application..... - 104 -

5.7 Summary - 110 -

6 CONCLUSION AND FUTURE WORK- 111 -

REFERENCES..... - 114-

LIST OF FIGURES

Figure 1-1	Symmetric Key Cryptography	4
Figure 1-2	Public Key Cryptography	6
Figure 2-1	Geometric representation of $P + (-P)$	16
Figure 2-2	Geometric representation addition of two points	17
Figure 2-3	Implementation choices for ECC	19
Figure 3-1	Field Multiplication	33
Figure 3-2	Field Inversion	35
Figure 3-3	Initialisation Timings	37
Figure 3-4	Point multiplication timings	38
Figure 3-5	Digit-Slice Rake method	41
Figure 3-6	Comparison of Digit-Slice Rake with Normal multiplication	43
Figure 3-7	173 Bit multiplication comparison	45
Figure 3-8	Pre-computations for 173 bits	46
Figure 3-9	The CertModule application	50
Figure 3-10	ECDSA Verification on the PDA	52
Figure 4-1	Diffie Hellman Protocol	55
Figure 4-2	M-I-M attack on DH protocol	56
Figure 4-3	Generic MAKEP	59
Figure 4-4	Powerful Server - Low powered clients scenario	61
Figure 4-5	Peer-to-Peer MAKEP for wireless devices	62
Figure 4-6	Jakobsson-Pointcheval protocol	63
Figure 4-7	Interleaving attack on the Jakobsson-Pointcheval protocol	65
Figure 4-8	Server-Specific MAKEP protocol	66
Figure 4-9	M-I-M attack on Server Specific MAKEP protocol	68
Figure 4-10	Linear MAKEP	69
Figure 4-11	Client Server MAKEP	72
Figure 4-12	A new MAKEP for wireless devices	74
Figure 4-13	A more secure version	78
Figure 5-1	The AES encryption process	83
Figure 5-2	ECC version of the new algorithm	85

Figure 5-3	ECC version of the Client-Server MAKEP	87
Figure 5-4	Graph of total time comparison	90
Figure 5-5	Actual Run time graph	91
Figure 5-6	Step 1 Encryption Comparison	92
Figure 5-7	Graph of speed of the protocol using pre-computations	93
Figure 5-8	Protocol speed at different window sizes for 173 bits	94
Figure 5-9	Peer-to-Peer version of our protocol	97
Figure 5-10	A more secure Peer-to-Peer version	98
Figure 5-11	Performance of the peer-to-peer protocol	101
Figure 5-12	Application Modules	104
Figure 5-13	Message flow of the system	105
Figure 5-14	AES Key Generation	107
Figure 5-15	User application on the PDA	109

LIST OF TABLES

Table 1-1	Comparison of Key Sizes	10
Table 1-2	Comparison of Keys Sizes (2)	11
Table 2-1	Solving ECC by the Pollard-Rho method	27
Table 3-1	Embedding data timings	36
Table 3-2	Readings for the Digit-Slice Rake method for 5 windows	42
Table 3-3	Readings for the Precomputations in Digit-Slice Rake method for 5 windows	44
Table 3-4	Readings for the Digit-Slice Rake method for 173 bit field	44
Table 3-5	ECDSA Verification on the PDA	51
Table 5-1	1 Comparison for the total time taken	90
Table 5-2	Runtime comparison	91
Table 5-3	Step 1 Encryption Comparison	92
Table 5-4	Time analysis of pre-computation in the protocol	93
Table 5-5	Time analysis on number of windows in the pre-computation	94
Table 5-6	Peer-to-Peer version timings for DH key generation	99
Table 5-7	Peer-to-Peer version timings for ECDSA Certificate Verification	100
Table 5-8	Peer-to-Peer protocol timing analysis	100

ABSTRACT

Wireless protocols are characterized by low computational power and memory. Hence security protocols dealing with these protocols have to be designed to give minimal computational and memory load on them. In addition to this issue, security is also a very important consideration since anyone can intercept wireless communication with the right equipment. Hence wireless applications need more security.

One way to provide additional security without adding to the computational load is to use Elliptic Curve Cryptography in place of more traditional cryptosystems like RSA. ECC provides comparable levels of security for smaller key sizes as compared to the traditional cryptosystems and hence is ideal for low power devices. In this thesis we will investigate different ways of implementing ECC on low power devices like PDAs. We will present our findings and compare the different implementations. We have obtained the best results for an ECC implementation over the field F_{2^n} using optimal normal basis. We have also used some pre-computational algorithms to speed up the slower modules of our implementation.

The first phase of a typical secure transaction or communication is a secure key exchange where the involved parties agree upon a common secret known only to them. This security of the rest of the transaction depends on this first step and if there is a compromise at this phase, the rest of the communication will no longer be secure. We have investigated the problem of authentication and key exchange in detail for low power devices operating in wireless scenarios. We have also analyzed the design strategies of recent key exchange and authentication protocols proposed specifically for low power devices.

Based on our analysis, we have suggested some modifications and have proposed a more efficient authentication and key-exchange protocol for mobile devices. Our protocol is shown to be nearly thrice as fast as comparable protocols. The basic idea behind the modifications has been to use symmetric key operations in place of public key operations where ever possible and to use ECC for public key operations like signing and verification. We have implemented this protocol using our ECC implementation on HP-Compaq iPAQ PDA.

1 Introduction

1.1 Wireless devices and the need for security

The history of the internet, according to Bruce Schneier [19], can be divided into three stages. The first consisted of expensive mainframes and terminals; and the second was centered on personal computers, browsers and large application programs. The third wave which is starting now will see the connection of all sorts of devices that are currently in proprietary networks.

The year 2003 saw more mobile phones connected to the internet than computers. The trend is for all devices to be connected to each other via the internet. We already have various types of devices like mobile phones, air-conditioners, and cameras etc. which are 'Internet aware'. Many of these small low powered devices are wirelessly connected, since their very nature, e.g. mobiles phones and PDAs, requires them to be mobile. Wireless devices are on their way to become ubiquitous personal accessories [18]. As successive generations of users are born into a wireless world, the average individual in the society no longer regards these devices as luxuries but as a part of the public infrastructure, or even as an entitlement. The laws of economics dictate that as usage becomes more common, the device grows more affordable. We have seen this happening with personal computers and we are now seeing this phenomenon with mobile phones and PDAs as they usher in a wireless world.

Wireless devices are inherently less secure than their wired counter parts [20]. This is partly due to their limited bandwidth, memory and computation power and partly due to the nature of the wireless environment since wireless transmission implies anyone with similar technology can intercept it. Thus wireless technology by its very nature violates fundamental

security principles (which we will discuss in section 1.3). With the spurt of wireless applications and the inevitability of wireless devices taking part in monetary transaction (e.g. internet banking) or classified or business transactions, the need for proper security is paramount. -

We are faced with two major mutually problematic issues in dealing with wireless devices and environments [42]

- There is a need for high level of security due to the insecure nature of the wireless media (which generally needs correspondingly more powerful computational devices) and
- Limited computational and memory power of wireless devices (which implies greater delays in computing protocols providing higher security).

We can tackle these problems in two ways

- By using Elliptic Curve Cryptosystem (ECC) which provides similar levels of security as conventional cryptosystems using the Discrete Logarithm or the Integer Factorization problems, for relatively smaller key-sizes.
- By modifying the security protocols into equivalent protocols in which the wireless devices don't have to be involved in the bulk of computations.

Secure communication is usually initiated by key-exchange where the communicating parties agree on a session key with which to protect rest of the communication. A security compromise in key exchange therefore results in compromise of the security of the entire session.

This thesis investigates the problem of developing an effective key exchange and authentication protocols for wireless devices, using elliptic curve cryptography. We will be examining the security of the wireless environment through two devices – iDEN phones from Motorola and iPAQ PDAs. This chapter will look at the basic issues concerning cryptography and wireless networks, to lay the groundwork for the rest of the thesis. In section 1.3, we will look at the different types of cryptographic techniques and compare their characteristics with respect to small low powered devices. The ECC will be discussed in greater detail in Chapter 2. There are quite a few ways in which elliptic curve cryptography can be implemented. We will look at implementation issues in Chapter 3, and decide which

type of implementation will be best suited for PDA devices. Chapter 4 will be based on different authentication and key exchange protocols for mobile devices. A new protocol for key exchange will be introduced and analysed in Chapter 5. We will also discuss the implementation and evaluation of different key exchange algorithms using the most efficient implementation of ECC from Chapter 3.

1.2 Contributions of this thesis

1.2.1 Implementation of ECC on PDAs

We have used Mike Rosing's implementation of ECC and modified it to run on WinCE [3]. There are many different viewpoints on which type of ECC implementation is the most efficient and there are many issues regarding this (which will be covered in chapter 2 and 3). We have tested the different implementations on a PDA and have shown that our Optimal Normal Basis implementation is faster than the Polynomial Basis implementation in all the major ECC operations. We have also updated the implementation with the latest and most efficient algorithms.

An article based on the implementation titled "Efficient ECC implementation for PDAs" has been submitted to the "Kluwer Wireless Personal Communications special issue on Security for Next Generation Communications" and is currently under review.

1.2.2 A new authentication and key exchange protocol

We have analysed a number of key exchange protocols that deal with a low power client – high power server scenario and based on our observations we have proposed a new key exchange protocol for wireless devices. A paper based on this protocol, titled '**An Efficient Authentication and Key-Exchange Protocol for Low Power Clients**' was presented at the Fifth IFIP International Conference on Mobile and Wireless Communications Networks (MWCN 2003) held in Singapore in October, 2003. Our protocol has been shown to be around thrice as fast as comparable key exchange protocols for mobile devices.

1.3 A brief overview of Cryptography

We will begin with a few basic concepts which are relevant to this thesis. In a very generic sense, the fundamental goal of cryptography has been to achieve privacy [1]. Cryptography enables two (or more) people to send each other messages over an insecure channel in such a way that only the intended recipients can read the message.

For end to end security a cryptographic system needs to provide

- Encryption

This ensures privacy if a third party intercepts messages.

- Authentication

This validates the identity of the entity sending the message.

- Integrity

This ensures detection of a tampered message.

- Non-Repudiation

This ensures that a contract which has been agreed to by two or more parties cannot be later denied by any of the involved parties.

There are two kinds of cryptographic techniques used to provide the above services, described in the next two sections.

In symmetric key cryptography both the communicating parties know a common secret called the symmetric key. Public key cryptography on the other hand involves the two parties having public keys which are known to everyone, while their corresponding private keys are known only to themselves.

1.3.1 Symmetric Key cryptography

Figure 1-1 Symmetric Key Cryptography

Figure 1-1 illustrates the way in which a symmetric key cryptosystem works: To use symmetric key cryptography, both participants, say *Alice* and *Bob*, have to possess the same secret key. There are many ways in which this can be accomplished, ranging from a physical meeting to agree upon a secret key, to using public key cryptography to generate a shared secret (e.g. the Diffie Hellman protocol). The message m is encrypted using the symmetric key k to produce the ciphertext c , which can be decrypted by *Bob* using the same key k to get back the message m . Symmetric Key cryptography is adequate for encrypting the data but by itself it has many limitations like –

- If the key distribution is not secure the security of the entire system may be compromised.
- For a network of n users, for every user to communicate securely with every other user, each pair must have a unique secret key. That means every user will have $n(n-1)/2$ keys which may be unmanageable if n is very large.
- Authentication, integrity of the messages etc. are quite hard to prove as signing a message is not possible in symmetric key cryptography. Since *Alice* and *Bob* have the same secret key, it is hard for *Bob* to convince a third party that a message was received from *Alice*, as *Bob* has the capability of creating a similar message.

These and other deficiencies of private key cryptography are addressed by public key cryptography, which was first proposed by Diffie and Hellman in 1976.

1.3.2 Public Key cryptography

Public key cryptography is based on a trap-door function. A trap-door one-way function (TOF) is a one-way function which can be easily inverted using some extra information. In the absence of this extra information it is very ‘hard’ to invert the function. This information thus acts as a kind of a trap-door.

The public key cryptography process is shown in figure 1-2.

Alice selects random value a and calculates f_a (through some public-key algorithm) which is her public key and is publicly visible (e.g. through a certified public directory). The corresponding secret key is $t(a)$ where t is the trapdoor function. To send a message to *Alice*, *Bob* will use *Alice*’s public key f_a to encrypt his message. Only *Alice* can decrypt this message since only she knows $t(a)$ without which it will be computationally infeasible to decrypt the encrypted message.

Figure 1-2 Public Key Cryptography

Similarly *Alice* can sign a message using her private key $t(a)$. *Bob* will use her public key to verify the message. The signature can be used to authenticate *Alice*, since only *Alice* can sign the message using her secret key (because only she knows it). This approach also preserves the integrity of the message since if the message has been tampered to something else, the mistake will be noticed when *Bob* applies *Alice*'s public key to the signature and compares the results.

Having looked at the two types of cryptography, the question naturally arises as to which is better. This question is incorrect in the sense that public and private key cryptography can't be compared on an equal footing. They are very different from each other and are intended to solve different sorts of problems [21]. Symmetric key algorithms like DES and AES are designed for fast encryption of blocks of data. Public key algorithms like RSA on the other hand enable a secure and authenticated transfer of data. Private key cryptography is thus best for encrypting data. The RSA system not only uses keys of much larger size than DES, it runs at about a thousand times slower. However the RSA system can provide authentication, non-repudiation and key management which DES by itself cannot. Thus public key and private key cryptography are often used together as hybrid systems to deliver end to end security that includes confidentiality, authentication, integrity and non-repudiation. For example in our project, we will be using public key ECC to exchange AES keys securely, after which the actual encryption will be done using AES.

We will now look at the security of different types of public key systems and discuss their merits and demerits with regards to wireless environments.

1.4 Security of Public Key Cryptosystems

Practical Public Key cryptosystems are based on one of three mathematical problems [20]:

- **Integer factorization problem (IFP):** e.g. RSA.
- **Discrete Log problem (DLP):** e.g. DSA (Digital signature algorithm), DH (Diffie-Hellman) key agreement, ElGamal encryption and signature scheme etc.
- **Elliptic Curve Discrete Log problem (ECDLP):** e.g. Variant of DH called ECDH, variant of ElGamal called ECEG, variant of DSA called ECDSA etc.

We will look at these problems in the next few sections.

1.4.1 The Integer factorization problem

Definition 1:

Given an integer n , find its prime factorization.

RSA is the most famous public key based on IFP. RSA public key is the pair (n, e) where e is a member of the group Z_n^* (the multiplicative group of integers mod n). The order of the group $\Phi(n) = (p-1)(q-1)$ is also calculated. Modular exponentiation dominates the time taken for the calculations in the RSA system [20]. Thus efficiency of the RSA system depends on how efficiently modular exponentiation is implemented. The IFP problem has been studied for over 300 years and many attacks have been discovered on it. From its definition, we can break an IFP based cryptosystem if we manage to find the prime factors of n i.e. p and q . 512 bit RSA was cracked in December 2000 using 292 average configuration computers and in around 5 months time. As of now, 1024 bit RSA is considered intractable. (Note: We came across several discussions in cryptographic forums on the internet which discusses the feasibility of creating a machine proposed by Bernstein [50] that could be used to potentially break 1024 bit RSA. The cost of such a machine is speculated to be in the tune of a billion US \$, but if in the near future such a machine could be built, 1024 bit RSA would no longer be 'intractable').

1.4.2 Discrete Logarithm Problem

Definition 2:

Let p be a prime and g be a primitive element of $Z_p^* = \{1, 2, \dots, p-1\}$ and given $y \in Z_p^*$; DLP consists of finding the integer x where $y = g^x \pmod{p}$.

The prime p should be at least 512 bits for short term security (although this is debatable since feasible attacks can be mounted with special purpose hardware) and for intractable security we need 1024 bit security. Thus IFP and DLP are more or less similar in terms of the key size required. The index calculus method is the most popular attack on DLP and many algorithms which try to solve DLP are based on it. The index calculus basically consists of a database of small primes and their corresponding logarithms and can be efficiently parallelized.

Some algorithms that use the index calculus method are the number field sieve and the quadratic sieve. The mathematics on which the algorithms are based on is the index calculus, which was briefly discussed before; and a property of numbers mod n called *smoothness* [22]. We will not go into the details of the algorithms since the mathematics is not relevant for this thesis. The interested reader can find many references in [21] on related topics.

1.4.3 Elliptic Curve Discrete Logarithm Problem

Definition 3:

Given points P and Q on an elliptic curve find the integer x such that

$P = x \bullet Q$, where ' $\bullet$ ' is a elliptic curve point multiplication operation.

The ECDLP is a variation of the DLP. Just as DLP is defined over the group Z_p , the ECDLP is defined over the points of an elliptic curve. We will look at elliptic curves and ECDLP in detail in Chapter 2. The security of ECC depends on ECDLP, which is DLP applied to elliptic curves. DLP applied to elliptic curve gains some powerful and unique features which make ECC ideal for low powered and constrained devices. The reason for this is that the faster algorithms that work on DLP don't work on ECDLP and hence the fastest method to crack ECDLP is an older algorithm called the pollard-rho method (discussed in chapter 2). The reason why algorithms like number sieve and quadratic sieve don't work is because there is no definition of smoothness in elliptic curves [22]. As we have discussed in the earlier section, the smoothness property is required by these algorithms to break DLP. This is the reason ECC needs keys of lesser size than other cryptosystems, since the more advanced algorithms cannot be applied on it. Thus a 1024 bit RSA key provides the same amount of security as a 163 bit ECC key [24]. This is a very significant reduction in key size and this can lead to much faster and efficient implementations.

However, the obvious question at this stage is what if in the near future ECC becomes susceptible to advanced attacks. Mathematicians and cryptologists seem to be divided on this issue. The point of contention is if the lack of smoothness is a fundamental property of elliptic curves or a lack of knowledge on part of the mathematicians. On one hand, elliptic curves have been studied since the 17th century but, according to Bruce Schneier [22], on the other hand they have been studied from the point of cryptography only the past 20 years or so. As opposed to this the IFP has been under scrutiny for around 300 years. However Alfred Menezes points out in [23] that this is an over simplification of the state of affairs as all the work that has been done on the DLP is directly applicable to ECDLP. None of the problems have been proven intractable anyway. I.e. we can't say for certain if 1024 bit RSA is intractable. Also although IFP has been around for centuries but it's been only in the very recent past that it has been studied with respect to cryptography. It may be possible that new algorithms are able to crack it in polynomial time in the future. There has already been a recent major breakthrough in this area [28]. Another point to note is that the fast attacks on IFP like the number sieve and quadratic sieve were discovered a few years ago, in spite of the work being done on IFP for centuries. Along with the discovery of these algorithms efforts was made by the worlds leading mathematicians to port these ideas into elliptic curves but till now no one has been even remotely successful [23].

In reality there is a general consensus that ECC is very good for constrained systems and in the 'short run' can prove to be the cryptosystem of choice for small low powered devices. In the 'long run' however if studies in ECC lead to certain discoveries e.g. smoothness, then ECC 'may' well be as efficient as the others. However this is a worst case scenario. There are many arguments and counter arguments (see [22], [23]) about these possibilities but the general opinion is that as of now ECC is very much suited for constrained systems and should certainly be made use of where it is required.

This opinion is substantiated by the fact that ECC is being incorporated into many commercial applications. Apart from Certicom, which has been pioneering ECC research, many erstwhile opponents of ECC like RSA Labs are also incorporating ECC in their products. In 1998 the US Treasury incorporated ECC technology for their online store [25]. To quote Alfred Menzes again [23] - 'If ECC is considered good enough for the US Treasury, it should be considered good enough for any application!'. The latest news of the

trend of moving towards ECC is from the NSA, which recently (October 24 2003) signed a 25 million US\$ license deal with US based Certicom to use their 512 bit ECC implementation, to secure US government's classified information.

Let us now look at how ECC fares in comparison to other cryptosystems.

1.5 The advantages of Elliptic Curve Cryptography

This section will deal with the key sizes, storage requirements and speed of ECC as compared to other cryptosystems. It has been already mentioned that since ECC is not susceptible to advanced fast attacks like the number sieve algorithm, its key size can be much less than other cryptosystems. Lenstra and Verhulst [24], have developed a method for evaluating key sizes of various crypto systems. A part of their results are reproduced in table 1.1 to give an idea about equivalent key sizes in bits

Table 1.1 Comparison of Key Sizes

Year	Symmetric Key Size	Classical Public Key Size	ECC Key Size	MIPS Years
1982	56	417	105	$5 * 10^5$
2003	73	1068	129	$3.5 * 10^{10}$
...	...	...	...	...
2020	86	1881	161	$3 * 10^{14}$

The table thus says that by 2020, we will need to use symmetric keys of at least 86 bits, RSA keys of 1881 bits, and ECC of at least 161 bits (not taking cryptanalysis into account).

A standard way of comparing different algorithms is of course not available and there are many different variations of this result that we have seen. There have been many other efforts to compare key sizes especially by Certicom (which promotes ECC); RSA Labs, IEEE etc. Much of it depends on the assumptions made, like advances in research in future, advances in computing power etc. Another set of data from RSA Labs is given below [26] in table 1.2.

Table 1.2 Comparison of Keys Sizes (2)

Symmetric	ECC	RSA
80	161	1024
112	255	2048
128	257	3072

We can see that although the results may vary depending on the assumptions of the study, the ECC key size is always much lesser than the RSA key size.

Another obvious conclusion from the above data is that symmetric algorithms are always faster than public key algorithms and they require much shorter keys. Public key systems are generally used for secure transport and exchange of keys, as well as for signing and verifying which are unique to Public Key systems. The actual encryption of the data will be much faster if it is done by a symmetric algorithm using the keys that have been securely agreed and made available by the Public Key algorithm. The AES (Advanced Encryption Standard) algorithm (also called the *Rijndael* algorithm) is recommended for US federal agencies by NIST and 128 bit AES is considered industry strength secure and is fast becoming a de-facto world standard.

The security of a system is as weak as its weakest link. Thus if we have a public key – symmetric key system in which we use 128 bit AES encryption we need to use 257 bit ECC or 3072 bit RSA to maintain the security of the system at par. In future if 128 bit AES encryption is no longer feasible, the public key size may also be needed to be increased correspondingly. Looking at the immense gap between the ECC and RSA public key sizes, RSA may well be infeasible to use in the future.

Besides the ones already mentioned there are several other advantages of ECC [27] –

- Forward secrecy is much easier to implement in ECC

Forward secrecy is a security attribute that prevents an attacker from reading messages that were passed in old (expired) sessions. This is generally achieved by using ephemeral (temporal) keys, or keys that generated anew for a session and destroyed after the session

is over. Generating a key is very straightforward in ECC (as we will see in the later chapters) but is comparatively much more complex in RSA so generating new keys every time a session is started can be costly in RSA.

- ECC can deal with calculation errors better

RSA with Chinese Remainder theorem is used to give improved performance. However, if it gives a single undetected bit error then the entire private key of RSA may be exposed. This problem does not occur in ECC.

- Multi-Party and Zero-Party scenarios

Due to the ease of key generation, the computations required to generate keys in multi-party (where each member has a partial private key) and zero-party (in which no one knows the private key) systems, are much more straight forward in ECC than RSA.

- Ability to handle Special Purpose attacks

Special purpose attacks exploit special weaknesses in the mathematics or the algorithm of the cryptographic system. We shall study the special purpose as well as general attacks on ECC in the next chapter. As of now there are certain known weaknesses on ECC like super-singular curves and anomalous curves which can be exploited. Similarly there are special weaknesses in RSA (e.g. the Chinese Remainder Theorem vulnerability discussed earlier). The ability to handle special purpose attacks depends on the ability to isolate all the cases that can be susceptible to these attacks. In ECC the domain parameters and the group structure are public knowledge and special weaknesses can be isolated by specifying the conditions that the domain parameters should observe to avoid these weaknesses. In RSA the underlying group structure and domain parameters have to be kept secret and it is difficult to verify if they follow secure practices. For some attacks even increasing the key size may not help, as was demonstrated in the WEP weakness [20]. In ECC on the other hand, future special purpose attacks can be easily avoided by not using harmful domain parameters and group structures.

1.6 Summary

In this chapter we have looked at the different kinds of public key cryptosystems and their advantages and disadvantages. We have also looked at a general introduction of ECC and discussed its merits with regards to constrained environments. We have also seen how ECC compares with other public key systems like RSA, followed by a discussion of some unique advantages of ECC.

In the next chapter we will look at the mathematics behind elliptic curves which is necessary to get to the third chapter which deals with the implementation issues of ECC. The fourth chapter deals with authentication and key exchange protocols for wireless devices and the fifth chapter presents the results and conclusions.

2 Elliptic Curves – Introduction and Literature Review

This chapter will introduce some concepts related to the theory of elliptic curves. Only the basic concepts related to this thesis will be touched, as the theory of elliptic curves is a subject of pure mathematics and beyond the scope of this thesis. This chapter will review the literature that is directly related to the implementation of security systems and algorithms using elliptic curves. To obtain a more complete mathematical exposition on elliptic curves, the reader may wish to refer to the book by Alfred Menzes [1]. Formal definitions of mathematical structures mentioned in this chapter can be obtained from [2].

2.1 Some basic notations

The affine *Weierstrass* form [1] of an elliptic curve is given by:

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3 \quad (2.1)$$

where a_1, a_2, a_3, a_4, a_6 can be defined over any field. For cryptographic purposes we generally assume that all the coefficients are in a finite field $GF(q)$ where $q = pm$, where p is prime and is called the *characteristic* of the field.

If for any point P on the Weierstrass equation, all 3 partial derivatives of X , Y and Z become 0, the point P is called a *singular* point and the equation is called a *singular* elliptic curve. These curves are unsuitable for cryptography and the reasons will be discussed later on in this section. The *discriminant* is a quantity used to check if the curve is singular or not. This value is used in many implementations to do a quick check to see if the given curve can be used for cryptography.

For an Elliptic Curve E we define the following quantities –

$$d_2 = a_1^2 + 4a_2$$

$$d_4 = 2a_4 + a_1a_3$$

$$d_6 = a_3^2 + 4a_6$$

$$d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2$$

$$\text{The discriminant } \Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \quad (2.2)$$

The Elliptic curve defined by the equation (2.1) is non singular if and only if its *discriminant* Δ does not evaluate to 0.

Another important concept in Elliptic Curve theory is the identity element $\mathcal{O}$ also called ‘the point at infinity’. This is the point in the curve where the Z coordinate is zero [1] and there is exactly one such point (at $0:1:0$). The identity element is equivalent to a zero in normal algebra in the sense that a point-addition operation of any point and $\mathcal{O}$ gives the same point. As far as implementation is concerned this point is not used in the code.

Using isomorphism over elliptic curves [1], it has been shown that equation (2.1) is equivalent to the following equation defined over the field of real numbers.

$$y^2 = x^3 + ax + b \quad (2.3)$$

For a Weierstrass equation, the variables can be complex, real, integers, polynomials, or member of any other kind of field [3]. In the next section we will look at elliptic curves over real numbers and the various operations which can be performed on them.

2.2 Elliptic curves over real numbers

An elliptic curve over real numbers is defined as the set of points (x, y) which satisfy an elliptic curve equation (2.3) where x, y, a and b are real numbers.

In this case the *discriminant* $\Delta = 4a^3 + 27b^2$ should not be zero for the curve to be non singular. Elliptic curve groups are additive groups; that is, their basic function is addition [4]. The addition of two points in an elliptic curve is defined geometrically. Addition is defined such that the result of addition of two points on an elliptic curve is another point on that curve.

The identity element is the point at infinity $\mathcal{O}$ which has been discussed earlier. For any point P on the curve E [3] –

$P + \mathcal{O} = P$ and	(2.4)
$-P = \mathcal{O} - (P)$	(2.5)

If P has coordinates (x, y) then, for the real valued equation in (2.2), $-P$ has coordinates $(x, -y)$.

Figure 2-1 Geometric representation of $P + (-P)$

This can be seen from figure 2-1 where a straight line through P and $-P$ will intersect the curve at infinity. Thus $P - P = P + (-P) = \mathcal{O}$.

Suppose that P and Q are two distinct points on an elliptic curve, and the P is not $-Q$. To perform a point-addition operation the points P and Q , a line is drawn through the two points. This line will intersect the elliptic curve in exactly one more point, call $-R$. The point $-R$ is reflected in the x-axis to the point R . The law for addition in an elliptic curve group is $P + Q = R$. An example (using the applet from [4]) is given in figure 2-2 –

Figure 2-2 Geometric representation addition of two points

If $P = Q$, then a tangent is drawn to P and the point of intersection of the tangent and the curve is taken as $-R$.

From the geometric approach, algebraic formulae can be derived as follows [3] –

With $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ as points on the elliptic curve let

$$R = P + Q = (x_3, y_3).$$

Then

$$x_3 = \theta^2 - x_1 - x_2 \quad (2.6)$$

$$y_3 = \theta(x_1 + x_3) - y_1 \quad (2.7)$$

where

$$\theta = (y_2 - y_1) / (x_2 - x_1) \text{ if } P \neq Q \quad (2.8)$$

or

$$\theta = (3x_1^2 + a_4) / 2y_1 \text{ if } P = Q \quad (2.9)$$

Elliptic curves over real numbers are not useful for cryptography as they have infinite number of points. An essential property of cryptography is that there should be a finite number of elements to operate on [4]. This is possible in ECC where we can replace the real numbers with elements of a finite field. Selecting a proper finite field is the first part of implementing ECC [7].

Before we look into elliptic curves over finite fields it is necessary to introduce Hasse's theorem since the number of points on the curve over finite fields must satisfy this theorem [4].

For an elliptic curve $E(F_q)$ over a field F_q , let $\#E(F_q)$ be the number of points on that curve. This is also referred to as the *cardinality* or the *order* of the curve. The order of a point on the other hand is the number of times the point has to be added to itself to get to $\mathcal{O}$.

Hasse's Theorem If $\#E(F_q) = q + 1 - t$ then $ t \leq 2\sqrt{q}$ (2.10)
--

From this we can infer that the number of points satisfying the equation in F_q is approximately equal to the field size [1, 3]. This is because from Hasse's theorem the probability that $x_1 \in F_q$ is at least $1/2 - 1/\sqrt{q}$, which is nearly $1/2$ for a very large value of q .

There are two types of finite fields over which ECC can be implemented, prime fields F_p or fields of odd characteristic and binary fields F_{2^n} or fields of characteristic 2 [5]. The next sections will look at these two types and discuss their important features. The figure 2-3

from [11] illustrates the possible choices for selecting the arithmetic for the underlying fields in ECC.

Figure 2-3 Implementation choices for ECC

2.3 Fields of Odd characteristics F_p

The arithmetic is implemented in the field F_p where p is a large prime. The elements in the field range from 0 to $p-1$ and all the operations like addition etc. are reduced modulo p . The standard operations for F_p are addition, subtraction, multiplication and division of which multiplication and division are computationally quite costly. Using isomorphism over elliptic curves, the standard equation for curves over F_p can be reduced to eqn. (2.3) so that $a_1 = a_2 = a_3 = 0$ [6]. For ECC, a typical prime is chosen to be larger than 2^{160} , and must be stored in multiple computer words [7]. The problem with this representation is that during computation, the carries between words must be propagated, and the reduction modulo p must be performed over several words.

Addition (and subtraction) is the simplest operation which involves two steps. The first is adding the two operands and the second is reducing the result modulo p . Multiplication, including squaring, is the most critical operation in the computation of elliptic curve point multiplications [9]. The multiplication of two elements $a, b \in [0, p)$ yields a result $ab \in [0,$

$(p-1)^2]$. The reduction of this result can be performed with a division: $ab \bmod p = (qp+r) \bmod p = r \bmod p$, where $q = \lfloor ab/p \rfloor$ and $r = ab - qp \in [0, p)$ where q represents the quotient and r is the remainder of the division. Division is more complex than multiplication. Division requires quotient estimation, multiplications, and additions. A popular method used for improving the arithmetic efficiency in F_p is the Montgomery reduction [8, 9].

Another way of improving the efficiency of arithmetic over F_p is by choosing primes such that the modular reduction can be efficiently computed. The example from [10] illustrates this point –

Consider a prime $p = 2^{192} - 2^{64} - 1$. A positive integer $n < p^2$ can be written in the form

$$n = \sum_{j=0}^5 A_j \cdot 2^{64j}, \text{ then } n = T + S_1 + S_2 + S_3$$

$$\text{where } T = A_2 \cdot 2^{128} + A_1 \cdot 2^{64} + A_0$$

$$S_1 = A_3 \cdot 2^{64} + A_3$$

$$S_2 = A_4 \cdot 2^{128} + A_4 \cdot 2^{64}$$

$$S_3 = A_5 \cdot 2^{128} + A_5 \cdot 2^{64} + A_5$$

Thus dividing n and calculating the remainder is avoided by splitting it into sub parts from which finding the modulus is much simpler.

2.4 Finite fields of characteristic 2 (F_2^n)

These fields are also referred to as binary fields, as they can be viewed as a vector space of dimension n over F_2 [10]. Thus n elements of the form $\{\alpha_1, \alpha_2 \dots \alpha_{n-1}\}$ exist such that any element in F_2^n can be written in the form $\sum_{i=0}^{n-1} a_i \alpha_i$ where $a_i \in \{0, 1\}$

The set $\{\alpha_0, \alpha_1 \dots \alpha_{n-1}\}$ is called a basis of F_2^n over F_2 . Thus any element in F_2^n can be represented as a binary vector $\{a_0, a_1 \dots a_{n-1}\}$.

Since the curve is over F_{2^n} the constants are polynomial in nature. For such curves the following simplified form

$y^2 + xy = x^3 + a_2 x^2 + a_6$	(2.11)
----------------------------------	--------

Till date no method of attack is known that is less than fully exponential in time for such curves if the coefficients are chosen properly [3]. The constant a_6 must not be zero although a_2 can be zero. If a_6 is zero, it implies a super-singular curve [5]. Super-singular curves are especially susceptible to attacks and hence should be avoided [1] [3].

The rules of addition are the same as the rules that have been illustrated for elliptic curves over real numbers. The formulae for addition over F_2^n can be derived as follows

With $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ as points on the elliptic curve let $R = P + Q = (x_3, y_3)$.

Then if $P \neq Q$

$$x_3 = \theta^2 + \theta + x_1 + x_2 + a_2 \quad (2.12)$$

$$y_3 = \theta(x_1 + x_3) + y_1 \quad (2.13)$$

where

$$\theta = (y_2 - y_1) / (x_2 - x_1) \quad (2.14)$$

If $P = Q$

$$x_3 = \theta^2 + \theta + a_2 \quad (2.15)$$

$$y_3 = x^2 + (\theta + 1)x_3 \quad (2.16)$$

where

$$\theta = x + y/x \quad (2.17)$$

Field addition and subtraction can be implemented as *exclusive or* of the basis vectors that represent points on the curve. The addition of two elements requires the modulo 2 addition of the coefficients of the two elements and since it's a binary field the coefficients are 1 or 0 i.e. addition is the same as XOR. Multiplication and inversion depend on the type of basis chosen [3].

There are many reasons why Galois fields of characteristic 2 (F_2^n) are attractive for implementation, whether hardware or software [6] [11].

- The arithmetic is carry free in F_2^n since we are dealing with essentially polynomials whose coefficients are always mod 2.

- There are many types of basis (e.g. polynomial or normal) which can be used to represent the points on the curve.
- To achieve sub second performance with F_p , a crypto coprocessor is generally required which can increase the over all cost of the product.(However in software implementations if an arithmetic processor is already available for modular exponentiation the performance of F_p maybe improved so that in some cases it may exceed F_2^n).
- Point compression required negligible computation in F_2^n while it may need considerable resources in F_p .
- It is easier to get the y coordinate of a point, given its x coordinate, in F_2^n with a single additional bit of extra information. This is also useful in reducing the message expansion¹ in protocols like El-Gamal.

In this project F_2^n was hence used as the field for implementing ECC. In the next section we will look at some of the popular bases used for F_2^n .

Every basis that can be used to represent the elements of F_2^n has its own advantages and disadvantages and these can be carefully utilized according to the implementation environment. There are many different bases in which to represent elements of F_2^n . The major ones will be discussed in the sections 2.5, 2.6 and 2.7.

2.5 Polynomial Basis

A polynomial or standard base is of the form $(1, x, x^2, \dots, x^{n-1})$ where x is the root of the irreducible² polynomial $f(x)$ of degree n over F_2 [6]. The arithmetic now becomes that of polynomials of maximum degree $n-1$, modulo $f(x)$.

¹ Message expansion is the additional amount of data added to the length of the original message after encrypting it.

² Irreducible polynomial is analogous to a prime number i.e. a polynomial that cannot be further factored into polynomials of degree > 0 .

The elements of F_2^n in polynomial basis are polynomials of degree less than n , with coefficients in F_2 ; that is, $\{a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_2x^2 + a_1x + a_0 \mid a_i = 0 \text{ or } 1\}$. These elements can be written in vector form as $(a_{n-1} \dots a_1 a_0)$. F_2^n has 2^n elements. The main operations in F_2^n are addition and multiplication. Some computations involve a polynomial $f(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_2x^2 + a_1x + a_0$, where each a_i is in F_2 .

Addition and subtraction operations are just the bit wise XOR of the coefficients of the operands since the coefficients are always mod 2. Multiplication and division are polynomial operations with the result mod the irreducible polynomial $f(x)$. We will look at algorithms for polynomial arithmetic in the next chapter.

2.6 Normal basis

A normal base of F_2^n over F_2 has the form $(\alpha, \alpha^2, \alpha^{2^2}, \dots, \alpha^{2^{n-1}})$ for some element $\alpha \in F_2^n$. Such bases exist for all $n \geq 1$ [6]. Squaring operation is trivial in normal basis representations since it amounts to a cyclic shift of the binary vector representing the input operand. Normal basis also allows for the design of efficient bit-serial multipliers [5]. Optimal normal basis (ONB) is a form of normal basis which enables more efficient operations.

An ONB of F_2^n over F_2 exists if and only if

- $n+1$ is prime and 2 is primitive in F_{n+1} . In this case the n non trivial $(n+1)$ roots of unity form a Type I ONB.
- $2n+1$ is prime and either
 - 2 is primitive in F_{2n+1} or
 - $2n+1 \equiv 3 \pmod{4}$ and 2 generates the quadratic residues in F_{2n+1} .

then $\alpha = \lambda + \lambda^{-1}$ generates a Type II ONB of F_2^n over F_2 , where λ is a primitive $(2n+1)$ root of unity.

Normal bases are useful mostly in hardware implementations. The bit serial multipliers can be very efficient in hardware but do not map nicely to software implementations [6].

The ONB mathematics is more complex than polynomial basis but its implementation in hardware or software is quite easy [3]. Only AND, XOR and ROTATE operations are needed for implementing ONB and these are generally the fastest operations on any microprocessor. Multiplication is more complicated than in polynomial basis as it needs

lookup tables to be implemented efficiently. Squaring on the other hand is just a rotation of the vector representing the element. The algorithms for different operations in ONB will be discussed in detail in the next chapter.

Since elements in one representation can be efficiently converted to elements in the other representation by using an appropriate change-of-basis matrix, interoperability between systems using the two different types of field representation can be achieved easily [11]. This brings us to the subfield basis.

2.7 Subfield base

When n can be represented as $n_1 n_2$, F_2^n can be regarded as an extension of degree n_2 of $F_2^{n_1}$; using a base of the form $\alpha_i \beta_j$ where $0 \leq i \leq n_1$ and $0 \leq j \leq n_2$; where $\beta_0, \beta_1, \dots, \beta_{n_2-1}$ form a base of F_2^n over $F_2^{n_1}$ and $\alpha_0, \alpha_1, \dots, \alpha_{n_1-1}$ form a base of $F_2^{n_1}$ over F_2 [5]. Thus arithmetic is divided over 2 parts, with the outer part (for e.g. normal basis) doing operations on elements of F_2^n as vectors of elements from $F_2^{n_1}$ and the inner part (for e.g. polynomial basis) doing operations on elements of the binary field F_2 .

The author of [6] recommends the value of n_1 to be between 4 and 16. This representation is good for $n_1 > n_2$, with n_1 still small enough for very fast symbol operations for e.g. through lookup tables.

Normal and polynomial bases have their own advantages and disadvantages. Polynomial basis mathematics can generally be implemented faster in software than normal basis mathematics, however normal basis computations require less storage space [3]. It wasn't very clear from the existing literature which choice would be the best for a software implementation. Hence both cases were used and their results are compared in the next chapter.

We have looked at the basic arithmetic on which ECC is based. We will now look at the security of ECC and the factors that influence it as these play a major part in making implementation decisions.

2.8 Security considerations

In general the security of a cryptosystem is evaluated by the amount of time and computational power needed to break it, or in other words find the secret key used for encrypting the data [16].

2.8.1 Attacks on ECC

The attacks on ECC are based on solving the ECDLP [6], [16]. Most of the methods have been modified from algorithms that are used to solve the discrete logarithm problem (DLP).

We will first make a few assumptions. For a curve E , let P and Q be points on the curve. Let F be the finite field used to define E and let α be its generator and β be an element in it. We will now look at some of the popular attacks on ECC [1] [6].

1. Square root methods.

Baby step – Giant Step algorithm for an exhaustive search is a popular method for solving DLP and ECDLP. It can be briefly described as –

If $x = \log_{\alpha} \beta$, we can write $x = jm + i$ for $0 \leq i < m$. A list of pairs (i, α^i) is precomputed and sorted by α^i . Then for each j for $0 \leq j \leq m$, $\beta\alpha^{jm}$ is computed and compared with the α^i in the sorted list. If a match is found, then $\beta\alpha^{jm} = \alpha^i$ and hence $\log_{\alpha} \beta = i + jm$. The algorithm requires a table with $O(m)$ entries.

Pollard's Rho algorithm is based on this method. It has an expected time of $\sqrt{(\pi r/2)}$ steps, where r is the order of point P . Each step is equivalent to an elliptic curve point addition. It can be implemented in parallel and hence can be made quite fast [12]. Another modification reduced the complexity by a factor of $\sqrt{2}$ [13]. This is the fastest version of attack on ECDLP known till date.

We will discuss the Rho algorithm in more detail since it is the most famous attack on ECC. To solve ECDLP we need to find l given P, Q and the fact that $P = lQ$. The points on the curve are partitioned into 3 roughly equal sized sets S_1, S_2 and S_3 . The iteration function f is defined on a point Z as

$$\begin{aligned} f(Z) &= 2Z && \text{if } Z \in S_1 \\ &= Z + P && \text{if } Z \in S_2 \\ &= Z + Q && \text{if } Z \in S_3 \end{aligned}$$

Random numbers $A_0, B_0 \in [1, n-1]$ are then chosen and the starting point Z_0 is chosen such that $Z_0 = A_0P + B_0Q$. Now $Z_1 = f(Z_0)$, $Z_2 = f(Z_1)$ and so on. A_i, B_i are tracked such that $Z_i = A_iP + B_iQ$. At some point $Z_i = Z_j$ i.e. $A_iP + B_iQ = A_jP + B_jQ$ or in other words $l = (A_i - A_j) / (B_i - B_j) \bmod n$. The equation won't be applicable in the unlikely event of getting $B_i = B_j$, in which case the search continues for another collision.

The Pollard method has been optimized and improved by suggesting the use of more sets of points than just three and executing the algorithm in parallel [12]. ' m ' processors start their computations with different values for Z_0 for each processor. Iteration takes place till a *distinguished* point is reached. A distinguished point is a point that has certain characteristics that are known to all the processors (e.g. several zeroes at the end in its binary format). Distinguished points are stored in common memory and the iteration continues until a pair of equal distinguished points is found. This improves the efficiency by a factor of m .

Another improvement in [13] doubled the efficiency of the algorithm. The modification was based on the algorithm that given a point Z , $-Z$ was very easy to calculate, whether the arithmetic was in F_p or F_2^n . For every Z encountered, $-Z$ was computed and minimum integer value of Z and $-Z$ was kept as Z . Thus the search space was effectively reduced by half.

Certicom Corp. has issued a ECC Challenge inviting participants to break elliptic curves of various sizes. Their results for the Pollard Rho method are relevant in this section [17].

They have estimated that a Pentium 100 computer can perform 16,000 operations per second for a curve over F_2^{89} . This would then require 15,550 days for a single computer running 24 hours a day to find a single discrete logarithm. A network of 3,000 such machines would require about 5 days. This figure translates to about 80 iterations per second on a 1 MIPS (million instructions per second) machine. The following table taken from [17] summarises their findings and estimations:

Table 2.1: Solving ECC by the Pollard-Rho method

Field Size (bits)	Size of n (bits)	$\sqrt{(\pi n/2)}$	MIPS years
163	160	2^{80}	9.6×10^{11}
191	186	2^{93}	7.9×10^{15}
239	234	2^{117}	1.6×10^{23}
359	354	2^{177}	1.5×10^{41}

Thus the Pollard Rho algorithm can be made infeasible if the order of E is sufficiently large i.e. greater than 2^{160} bits.

2. The Pollig Hellman method

This is a method that takes advantage of factorization of the order of a point P in E . By reducing the problem of finding a secret key to the problem of finding the modulus of each of the prime factors of the order of P , the secret key can be recovered using the Chinese Remainder theorem. The algorithm factors the order of a curve into small primes and solves the discrete logarithm problem as a combination of discrete logarithms for small numbers. To prevent an attack by this method, the order of E should be divisible by a large prime i.e. $> 2^{160}$ bits.

3. The index calculus method

This method is used for solving DLP and does not work very well for ECDLP [14] unless we are dealing with special types of curves i.e. super-singular curves. It has been shown that in such curves, the ECDLP problem can be reduced to the DLP problem in the extension field of F_q , where the index calculus method can be applied [15].

These methods are used for general attacks on ECC. However there may be various special attacks that target some special weakness in a particular ECC [16]. Some elliptic curves are deemed weaker than others. We will look at two special types of curves which create exploitable vulnerabilities.

1. Super-singular curves

An elliptic curve over F_q is said to be super-singular if the trace $= q + 1 - \#E; t$ (see Hasse's theorem [3]); is divisible by the characteristic of F_q .

It has been proved that ECDLP can be reduced to DLP under certain assumptions, in some extension field F_q^k . This is practical only if the k is small and this condition is satisfied by super-singular curves. So they can be broken in sub exponential time just like the DLP problem. Super-singular curves are hence avoided in ECC.

2. Anomalous (or Koblitz) curves

For curves over $F(2^m)$ with coefficients in $F(2)$, called anomalous binary curves or Koblitz curves, the attack time can be reduced by a factor of the square root of $2m$ [13]. These curves have a structure which allows faster cryptosystem computations. Unfortunately, this structure also helps the attacker.

2.9 Summary

In this chapter we have discussed the theory of elliptic curves and its application to cryptography. We have looked at elliptic curves over F_2^n and over the field F_p and discussed the advantages of F_2^n for our software implementation. We have seen the different types of basis like polynomial, normal and subfield basis over the field F_2^n . Finally we have looked into the special cases of ECC which have inherent weaknesses and which should be avoided in implementations, and also the different types of attacks on ECC.

3 Implementation of ECC on a PDA

This chapter describes the implementation of the Elliptic Curve Cryptography over a HP-Compaq PDA. We will utilize the mathematical concepts from Chapter 2 to create an efficient software ECC implementation for WinCE. This implementation is primarily based on the implementation described in [3]. Taking this as a basic framework we have added different algorithms and techniques to get a more efficient and faster implementation. This code is written in C and has been modified to run on WinCE based PDAs.

We will first discuss some problems we encountered while writing a Java based encryption package for ECC. We will then briefly discuss the ECC implementation in C based on [3].

3.1 A Java based implementation

An attempt was made to create a Java based version of the code which could be used in J2ME enabled devices like mobile phones. Using the 'BigInteger' class from the java.math package we have created a J2SE compliant ECC package based on [3] which gives quite good results. However we encountered problems while porting this package into J2ME, the reason being the BigInteger package is not a part of J2ME as of now. A J2ME implementation of the BigInteger class, obtained from [29] was used instead. Since this was a purely java based implementation we got unacceptably slow results from this on the J2ME enabled iDEN phone simulator. The reason for this is that the original BigInteger class uses native code for low level bit manipulation. By using pure java to do this in the J2ME compliant BigInteger class we had added one layer of Virtual Machine translation for the time critical parts of the code. Further research revealed that mostly all Java cryptography packages used native code to do computationally expensive operations, using the Java portion to provide an interface for Java code to these functions. Since we had no access to

native code on the iDEN phones we had no means to create a realistically fast ECC package for J2ME.

Hence we concentrated mainly on developing the implementation of ECC in C on PDAs and then test algorithms for key exchange and authentication for the PDA using our implementation and on the mobile phone using Motorola's proprietary cryptographic libraries. The following section will describe the implementation of ECC in C from [3].

3.2 A C based implementation

Rosing [3] deals with two types of basis for his implementation of ECC – polynomial and optimal normal, both of which have been introduced in the previous chapter. We will look at the implementation of both in brief, but first let's look at the common constructs and data structures.

The cryptosystem considers non super-singular curves of the type –

$$y^2 + xy = x^3 + a_2x^2 + a_6 \quad (3.1)$$

where a_2 maybe zero but not a_6 may not be zero. The rules of addition, subtraction etc. for points on the curve over F_2^n are given in Chapter 2 and are applicable here.

The constants a_2 and a_6 are of the type FIELD2N, where FIELD2N is defined as a structure containing an array of unsigned long values. The size of the array is set to the number of unsigned long values that are required to accommodate an element of the field F_2^n .

A *curve* is defined as a structure with FIELD2N values for a_2 and a_6 , since all the curves will be of the form given by the above equation.

A *point* on the curve is a structure with FIELD2N values for its x and y coordinates.

Another interesting thing to note is that adding two elements of the field F_2^n will be the bit wise XOR of their values, as the coefficients have to be reduced modulo 2. This also implies that addition and subtraction will be identical operations.

3.2.1 Polynomial base implementation

As we have discussed in chapter 2, the elements of F_2^n in polynomial basis are polynomials of degree less than n , with coefficients in F_2 ; that is, $\{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_2x^2 + a_1x + a_0 \mid a_i = 0 \text{ or } 1\}$. These elements can be written in vector form as $(a_{m-1} \dots a_1 a_0)$. For e.g. the polynomial $x^6 + x^3 + 1$ can be written in an 8 bit vector as 01001001. Since the field is F_2^n , the coefficients will either be 1 or 0 (i.e. mod 2). We have already discussed that addition and subtraction of the elements will be a simple XOR operation.

3.2.2 Optimal Normal Basis Implementation

As we have already seen in the previous section, an element β in the field F_p^m can be represented as

$$\beta = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \dots + a_2x^2 + a_1x + a_0$$

A Normal Basis using this element over F_p^m will then be represented as

$$\{\beta^{p^{m-1}}, \dots, \beta^{p^2}, \beta^p, \beta\}.$$

For the field F_2^n we can say a normal base of F_2^n over F_2 has the form $\{\beta^{2^{n-1}}, \dots, \beta^{2^2}, \beta^2, \beta\}$ for some element $\beta \in F_2^n$.

Using this basis, we can represent any element e in F_2^n as $\{e_{m-1} \beta^{2^{n-1}} + e_{m-2} \beta^{2^{n-2}} + \dots + e_2 \beta^{2^2} + e_1 \beta + e_0 \mid e_i = 0 \text{ or } 1\}$

Or in other words

$e = \sum_{i=0}^{m-1} e_i \beta^{2^i} \tag{3.2}$
--

Let us now look at some of the other basic operations using these basis.

3.3 Multiplication

Polynomial Basis

In polynomial basis multiplication of 2 polynomials is a shift and XOR operation of their vectors. Rosing implements a normal multiplication algorithm in which every term in a polynomial is multiplied by the other polynomial. This automatically means the second polynomial is shifted by the degree of the term of the first polynomial. E.g. the term $x^3 * (\text{polynomial_2})$ means the polynomial_2 will be left shifted by 3 bits. Thus we get one new polynomial for every non zero term in the first polynomial. The resulting polynomial vectors

are XORed with each other, which correspond to addition of the polynomials in F_2^n . The result has to be modulo a prime polynomial.

Optimal Normal Basis

Squaring and multiplication are quite efficient in this notation. For any term

$(\beta^{2^i})^2 = \beta^{2^{i+1}}$ and $\beta^{2^m} = \beta$ (since F_2^m is a finite field). So squaring an element is a simple right shift and rotate operation.

Multiplication is also very efficient. If A, B are elements of F_2^n we can represent multiplication as

$$C = A * B = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} (a_i b_j \beta^{2^i} \beta^{2^j}) \quad (3.3)$$

$$\text{Where } C \text{ can be represented as } \sum_{k=0}^{m-1} (c_k \beta^{2^k}) \quad (3.4)$$

From [3], [35] this implies

$$c_k = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} (a_{i+k} b_{j+k} \lambda_{ij}) \quad (3.5)$$

Where λ is called the *lambda matrix* which is basically a mapping:

$$\beta^{2^i} \beta^{2^j} = \sum_{k=0}^{m-1} \lambda_{ijk} \beta^{2^k} \quad (3.6)$$

From equation 3.6 we can see that multiplication can be done by shifting A and B by appropriate amounts (i.e. by $k = 0$ to $m-1$), and after this all the c_k terms can be calculated in parallel. The lambda matrix can be pre-computed.

An Optimum Normal Basis is a Normal Basis which has minimum non-zero terms in the lambda matrix. This will naturally make the operations much more efficient (hence the word *optimal*). As we have discussed in Chapter 2, there are two types of ONB over F_2^m depending on the value of m . The difference is in the way the bits are set in the lambda vector.

Figure 3-1 shows the huge difference in the efficiency of the multiplication operation in Polynomial Basis and Optimal Normal Basis.

Figure 3-1 Field Multiplication

As we can see from figure 3-1, the ONB multiplication remains constant for all field sizes at 6ms, while the polynomial multiplication decreases from 11 to 8 ms for field sizes from 158 bits to 50 bits. At 158 bits the polynomial multiplication is twice as slow as the ONB, and for higher bits it should be even slower.

3.4 Inversion

Polynomial Basis

The inversion is carried out using Euclid's algorithm [34]. Given a polynomial $r(x)$ and an irreducible polynomial $M(x)$, let the inverse modulo $M(x)$ be $p(x)$.

$$r(x) * p(x) = 1 \text{ mod } M(x) \quad (3.7)$$

The algorithm uses variables r_{k-2} , r_{k-1} , p_{k-2} , p_{k-1} , q_{k-2} , and q_{k-1} , all polynomials defined as FIELD2N. These are initialized as –

$$r_{k-2} = M(x)$$

$$r_{k-1} = r(x)$$

$$p_{k-2} = q_{k-1} = 0$$

$$p_{k-1} = q_{k-2} = 1$$

We start with the following calculations

$$r_k = r_{k-2} \bmod r_{k-1}$$

$$q_k = r_{k-2} / r_{k-1}$$

$$p_k = q_k * p_{k-1} + p_{k-2}$$

If the remainder i.e. r_k is not zero, the variables are exchanged $r_{k-1} = r_k$ and $r_{k-2} = r_{k-1}$ and so on) and the same process is followed till r_k equals zero. The inverse is now p_k .

Using these basic operations, elliptic curve operations like adding two points to get a third point etc. as described in Chapter 2, can be carried out.

Optimal Normal Basis

Inversion in ONB can be defined as [39]

Given an element $(a_{m-1}, \dots, a_1, a_0)$ find an element $(b_{m-1}, \dots, b_1, b_0)$ such that

$(a_{m-1}, \dots, a_1, a_0) * (b_{m-1}, \dots, b_1, b_0) = 1$ where 1 is defined as

$$1 = \sum_{i=0}^{m-1} \beta^{2^i} \quad (3.8)$$

Thus the fundamental constant 1 is represented as all the bits set in normal basis.

So now if we are looking for an inverse of element ' a ' we have

$$a * a^{-1} = 1$$

By Fermats theorem

$$a^{-1} = a^{(2^m - 2)} = (a^{(2^{m-1} - 1)})^2 \text{ and so on} \quad (3.9)$$

This equation is converted to a more efficient form using sliding windows [3].

As we can see from figure 3-2, for higher bit fields (around 155) the ONB implementation is almost 2.5 times as fast as compared to the polynomial implementation. For lower bit fields the ONB is almost twice as fast (around 50) as the polynomial implementation.

Figure 3-2 Field Inversion

3.5 Addition and Subtraction

These operations are the same for in polynomial basis and ONB constitute an XOR of the vectors of the elements. They are not computationally expensive and will be of the order of a few micro seconds.

3.6 Embedding points

Embedding points is an important operation for ECC where we want to hide some data on the curve. We do this by solving the equation of the ECC for y given x , where x is the data we want to embed on the curve. We get a point P on the curve as a result. It may happen that no point on the curve may have an x -coordinate corresponding to our value of x , in this case Rosing suggest in [3] that x could be changed bit by bit till we hit a proper value. To solve the ECC equation, we need to calculate the 'Trace' of the equation. This is easy to compute in ONB, but for the polynomial basis it is computationally quite expensive. We will not discuss it in detail in this section as we are more concerned with the speed of the implementation. The Trace can be pre-computed and adds quite a bit of time to the initialization of the polynomial basis code as we will see in the next section. On the whole we found that the embedding operation took between 2 ms to 15 ms in the ONB implementation. For the polynomial basis it took between 12 ms to 315 ms. It did not depend

entirely on the field values as even for a lower field there was a chance that the random value we had chosen would not match a x -coordinate of any point on the curve. In some cases we observed the lower bit fields required much larger time as shown in table 3.1 in the case of 111 bit polynomial basis implementation. This may have been due to repeated attempts to embed the random value on the curve, changing it bit by bit after each attempt. On the whole however we can see that ONB performs much better than polynomial basis.

Table 3.1: Embedding data timings

Field Size (Bits)	ONB Type II (Milli Seconds)	Polynomial Basis (Milli Seconds)
158	15	144
155	14	-
134	12	-
130	-	107
128	-	113
119	34	-
113	22	-
111	-	315
93	-	12
90	9	-
75	-	63
65	3	-
64	-	85
50	2	-

3.7 Pre-Computational costs

There are some initial calculations that have to be made for both ONB and polynomial basis before we can actually use them for cryptographic functions. In the case of ONB, this involves calculating the values for the λ matrix, as discussed in section 3.3. For polynomial basis this involves calculating the trace vectors as mentioned in section 3.5, and storing them in an appropriate matrix form. As we can see from figure 3-3, the initialization costs are more or less constant for ONB at around 6ms. For polynomial basis they increase as the field size increases. For higher field sizes (around 155), the ONB initialization is nearly 20 times faster than the polynomial initialization.

Figure 3-3 Initialisation Timings

3.8 Elliptic point multiplication

We have looked at the implementation of the basic operations on elliptic curves. We will now look at a major operation that is essential in ECC – multiplication of an integer to a point on the curve to give another point on the curve. This operation is essential in most of the key exchange algorithms we will look at in chapters 4 and 5. The implementation follows the algorithm in [36] to create a *balanced representation* of the field element to be multiplied.

A balanced representation is created from the binary representation of a number by replacing some bits of the representation to ‘-1’. The conversion details can be found in [3, 36, 37]. The idea is that instead of adding a point to itself x times, we use subtraction and doubling operations to reduce the overall number of operations needed. We can now use the formulae from Chapter 2 of point addition and subtraction to calculate the result from the balanced representation.

We can see the timing analysis from figure 3.4. ONB is clearly much faster than polynomial basis. This is to be expected since ONB is faster than polynomial basis in all the basic operations that we have seen in the earlier section.

Figure 3-4 Point multiplication timings

Elliptic point multiplication is one of the costliest operations in elliptic curve cryptography. Point multiplication is also a very essential operation for ECC. It is analogous to exponentiation in the Discrete Log Problem algorithms, in the sense that it is very hard to reverse. For example let P be a point on the curve E and let x be an integer. Also, E is defined over F_2^n . Then point multiplication of point P with integer x will give us another point P' on the curve i.e. $P' = P x$.

Calculating x from P and P' is essentially breaking the ECDLP problem discussed in chapter 1 or in other words, breaking the security of ECC and is very difficult. Hence point multiplication is one of the fundamental operations of ECC. However we found that it is also relatively costlier, in terms of time, than the other operations. If the field size is large, the time for this operation will obviously be considerable.

We will look at the most popular methods of doing point multiplication in this section. The most straight-forward way of doing point multiplication would be a variation of the shift and add technique. In this we will be going through the integer x bit by bit and performing an addition when we hit a 1-bit, and doing nothing if we hit a 0-bit [1]. We will obviously have to double the result before checking the next bit, to take care of the place value.

For example, consider a point P and an integer $x = 10045_{10}$ or 10011100111101_2

So in this case the point multiplication operation will be:

$$(((((((P * 2 * 2 * 2 + P) * 2 + P) * 2 + P) * 2 * 2 + P) * 2 + P) * 2 + P) * 2 * 2 + P$$

The algorithm can be represented as:

Input: $x = (x_{n-1}, \dots, x_1, x_0)_2, P \in E(F_2^n)$

Output: xP

1. $Q \leftarrow 0$
 2. For $i := n-1$ to 0 do
 - 2.1 $Q \leftarrow 2Q$
 - 2.2 If $x_i = 1$ then $Q \leftarrow Q + P$
 3. return (Q)
-

3.8.1 Non Adjacent Form (NAF)

Point multiplication using NAF is based on the simple shift and add algorithm we have just seen. NAF form or balanced integer form is a signed-digit representation of an integer. NAF is particularly attractive for point multiplication as no two adjacent elements in its representation are one. By converting a normal binary representation to a NAF representation we increase the number of zeroes in the representation along with the introduction of a few negative ones (since it's a signed representation, it will have 1, 0 and -1 to represent the number). NAF is generally obtained from a binary representation by replacing a string of '1's in the number by '1', followed by a string of '0's, followed by '-1' [2].

For example consider $15 = 1111_2$. We can represent 15 as $(16 - 1)$ or $10000_2 - 1_2$. In NAF representation it would be $(1 \ 0 \ 0 \ 0 \ -1)$. This basically means $(1) * 2^4 + (0) * 2^3 + (0) * 2^2 + (0) * 2^1 + (-1) * 2^0$, which is 15.

The thing to note that in ordinary binary representation of 15 we have four '1's, while in NAF representation we have only one '1', three '0's and one '-1'. When we encounter a '0'

we need not perform any operation. When we encounter a '-1' we need to do a point subtraction. We have seen in the earlier sections that point addition and subtraction have the same efficiency. So by using NAF, we are increasing the number of '0's in the representation and thus making the point multiplication operation more efficient.

To illustrate this further consider a point P and an integer $x = 10045_{10}$ or 10011100111101_2 .

Converting to NAF we have

10100-101000-101.

So the multiplication operations are

$$(((P*2*2 + P)*2*2*2 - P)*2*2 + P)*2*2*2*2 - P)*2*2 + P$$

So we have only 5 additions/subtractions in the NAF form as opposed to 8 additions in the original binary form we saw in the previous section. Multiplication using the NAF form will hence be faster.

In using the NAF form, we will have to calculate the NAF representation of the number first before we start the actual multiplication. The algorithm has been discussed in [1], [2] and [3] in depth. We found that it takes negligible time in comparison to the multiplication part and hence makes no effect on the overall efficiency. The multiplication algorithm will be similar to the shift and add algorithm in the previous section, except that there will be a check to see if the bit is '-1' in which case the point P will be subtracted from the result.

The results for polynomial basis and optimal normal basis (Type II) for EC point multiplication using NAF are given in figure 3-4. ONB is clearly much faster than polynomial basis. At higher values of bits in the field, ONB is more than twice as fast.

3.8.2 NAF with pre-computation

A technique for a one time pre-computation for NAF is examined in this section. We will be working only with ONB since we have seen it is faster than polynomial basis for pure NAF multiplication anyway.

An example of multiplication using pre-computation would be the Digit-slice Rake scalar multiplication [3]. This method reduces the number of point doublings by using some pre-computation.

Digit-slice Rake method

Figure 3-5 Digit-Slice Rake method

The method is shown in figure 3-5. We move a rake of g teeth over the integer x . Each 'tooth' of the rake will travel over maximum L bits. L is also called the window size.

Now the point multiplication using NAF can be represented as:

$$P' = P x = \sum_{i=0}^{l-1} l_i 2^i P \quad (3.10)$$

Where x is made of l bits and l_i represents the value of the i^{th} bit. Since we are using the NAF representation, l_i can be 1, 0 or -1.

Expanding equation (3.10) by using $l = gL$, we get

$$P' = \sum_{i=0}^{g-1} \sum_{j=0}^{L-1} l_{iL+j} 2^{iL} 2^j P = \sum_{j=0}^{L-1} 2^j \left[\sum_{i=0}^{g-1} (l_{iL+j} 2^{iL} P) \right] = \sum_{j=0}^{L-1} 2^j \left(\sum_{i=0}^{g-1} (l_{iL+j} P_i^*) \right) \quad (3.11)$$

where $P_i^* = 2^{iL} P$ is the pre-computed value, where i goes from 0 to $g-1$, as we can see from equation (3.11). Thus we are reducing the doublings by 'g'.

The algorithm is

Input: $x = (l_{n-1}, \dots, l_1, l_0)_2$, $P \in E(F_2^n)$, pre-computed $P_i^* = 2^{iL} P$

Output: xP

1. $Q \leftarrow 0$
2. For $j = L-1$ to 0 do
 - 2.1 $Q \leftarrow Q + Q$
 - 2.2 $R \leftarrow 0$
 - 2.3 For $l = 0$ to $g-1$ do
 - 2.3.1 $R \leftarrow R + (l_{iL+j} * P_i^*)$
 - 2.4 $Q \leftarrow Q + R$
3. return Q

Analysis of the Digit-Slice Rake technique

We found that using this technique improves the performance substantially. For example, implementing the algorithm for 5 windows i.e. $g = 6$; increased the speed to almost thrice that of using NAF without pre-computations. However for higher bit values, the value of pre-computation is quite large. For example, it is more than 25 seconds for 270 bit field size. However this is a one time pre-computation only. The results can be stored in a file and loaded in to the program when required, after which the program can use it as much as required. The size of the file will not be more than a few Kb and for a commonly available 64 Mb PDA this is quite insignificant.

We show the results obtained for different field sizes for 5 windows in Table 3.2.

Table 3.2: Readings for the Digit-Slice Rake method for 5 windows

Field Size (Bits)	Normal multiplication (Secs)	Digit-Slice Rake Method (Secs)
270	14.756	5.88
230	9.800	3.85
210	6.560	2.60
189	4.998	2.03
173	3.733	1.55
155	2.700	1.11
134	1.760	0.71
119	1.314	0.53
98	0.746	0.31
81	0.348	0.13
65	0.195	0.07

As we can see from Figure 3-6, there is a uniform improvement of speed by a factor of almost 3 times for all field sizes. Series 1 is the multiplication without using windows while series 2 is the multiplication results using the windows method.

Figure 3-6 Comparison of Digit-Slice Rake with Normal multiplication

Table 3.3: Readings for the Precomputations in Digit-Slice Rake method for 5 windows

Field Size (Bits)	Pre-computations (Secs.)
270	25.25
230	22.34
210	15.50
189	11.63
173	8.21
155	6.38
134	4.12
119	2.17
98	1.24
81	0.83
65	0.33

Table 3.3 gives the readings for the pre-computations for different field sizes for 5 windows. As we can see there is a drastic increase for the higher bit value fields (from 0.3 to 25 seconds) however this is only a one time computation and does not affect the cryptosystem in anyway. It just has to be done once for a particular curve and a particular base point. Once this has been done the pre-computed values can be passed along with the implementation. So the speed of the implementations does not depend on it in anyway.

For a practical application, we will be using a field size of 155 to 170 bits for industrial strength security, as we have discussed in chapter 1. In this case, for 173 bit field size, we will have to endure a one time 8 sec pre-computation by which the speed will be about 2.5 times faster than normal multiplication.

We will now look at how different number of windows affects the speed of the algorithm for a field size of 173 bits. The results are given in table 3.4.

Table 3.4: Readings for the Digit-Slice Rake method for 173 bit field

Number of Windows	Normal multiplication (Secs)	Digit-Slice Rake Method (Secs)	Pre-computations (Secs.)
2	3.74	2.416	2.844
3	3.74	1.937	3.700
4	3.73	1.717	5.016
5	3.74	1.551	6.240
6	3.73	1.467	7.720
8	3.73	1.335	10.140
9	3.73	1.303	11.710
10	3.73	1.281	12.062
13	3.73	1.19	17.147
17	3.73	1.15	22.870

From the values a window size of 10 seems to be a good value, as it gives us multiplication that is more than thrice as fast for a one time pre-computation of 12 seconds. For higher number of windows the multiplication will be faster. The trend can be seen in figure 3-7.

But on the flip-side it implies a greater cost for pre-computation. As we can see from figure 3-8, the increase time for pre-computation is quite drastic as compared to the increase in the speed of the algorithm. For 2 windows pre-computation is just under 3 seconds and for 17 windows it jumps to 23 seconds. We can see from the graph that the increase is linear. However practically this will not be of any concern since it's a one time pre-computation that needs to be performed only once to store the values permanently into the PDA's memory.

Figure 3-7 173 Bit multiplication comparison

Figure 3-8 Pre-computations for 173 bits

We will end this chapter with a discussion on the ECDSA protocol which we will be using for signing certificates for our key establishment protocols. We will first look at the protocol and then discuss its implementation, using the basic ECC operations we have just discussed.

3.9 ECDSA (Elliptic Curve Digital Signature Algorithm)

3.9.1 Discussion of the protocol

The ECDSA is the elliptic curve equivalent of the popular DSA signature and verification protocol. ECDSA has been adopted as a standard in ANSI, IEEE and NIST as well as ISO. We will be looking at the ANSI standards and the IEEE P1363 standards [43] for a basic theoretical discussion and we will concentrate on the IEEE P1363 based implementation suggested by Rosing [3] for a proof of concept implementation.

Since we will be implementing ECDSA on the PDA, we choose to follow an ONB based implementation over F_2^m . This is because we have consistently obtained better performances for all the low level ECC operations, as shown in the earlier sections. We note that the ANSI X9.62 standard maintains that the number of rational points on the curve E , be divisible by a sufficiently large prime n . In order to avoid the Pollard’s rho and the Pohlig-Hellman attacks

we have discussed in chapter 2, $n > 2^{160}$. In general n is selected as large as possible so $n \approx 2^m$. The ‘co-factor’ is defined as $h = \#E(F_2^m)/n$, where $\#E$ is the number of points on the curve E . The ANSI and IEEE standards also maintain that the curve should be non-supersingular, to avoid the weaknesses we have discussed in chapter 2.

Both standards propose the same way of choosing a key pair. The private key is a random number and the public key is the private key multiplied by the Base point. We will be using this relation in all our protocols related to ECC in chapters 4 and 5. In our implementation we have considered a random key pair, and have chosen it to sign messages. So our certification authority is a hypothetical entity that has this key pair. For a proof-of-concept implementation this seems adequate.

The first step of the protocol involves computing a hash of the message to be signed using the SHA-1 algorithm. SHA-1 creates a 160 bit unique message digest of the message. There are many open source implementations of SHA-1 and we have used one from [45] in our code.

Having looked at the preliminary requirements for ECDSA let us now examine the protocol in brief [3]. We have a publicly known base point P on a curve E with order n (i.e. the number of points on the curve). The certification authority (CA) has its own private key – public key pair which are used to signing and verifying a message respectively. Let s be the secret key of the CA. Then its public key will be $Q = s*P$. We also have a random value k and a point R such that $R = k*P$. The message M to be signed is first hashed into a 160 bit value $e = \text{SHA-1}(M)$. The first signature component c is obtained by

$$c = R.x \bmod n \quad (3.12)$$

where $R.x$ is the x component of the point R . The second component d is derived as

$$d = k^{-1}(e + sc) \quad (3.13)$$

Thus (c,d) will constitute the *signature* on message M .

For verification, the verifier needs to know the CA's public key Q . The verifier will first get the hash e' of the received message M' which may or may not be modified on its way and calculate 3 values as follows

$$H = d^{-1} \mod n \quad (3.14)$$

$$H_1 = e'H \mod n \quad (3.15)$$

$$H_2 = cH \mod n \quad (3.16)$$

Now point R' is calculated as:

$$R' = H_1P + H_2Q \quad (3.17)$$

$$\text{And } c' = R'.x \quad (3.18)$$

c' must be equal to c (which is the first component of the signature (c, d)) for the signature to be *verified*.

The proof is quite straightforward:

From (3.13) and (3.14)

$$H = k(e + sc)^{-1} \quad (3.19)$$

Putting this value of H into (3.15) and (3.16) we get

$$H_1 = e'k(e + sc)^{-1} \quad (3.20)$$

$$H_2 = ck(e + sc)^{-1} \quad (3.21)$$

And substituting (3.20) (3.21) into (3.17) gives us

$$R' = e'k(e + sc)^{-1}P + sck(e + sc)^{-1}P \quad (3.22)$$

After grouping the common terms we get

$$R' = k(e' + sc)(e + sc)^{-1}P \quad (3.23)$$

From (3.23) we can see that the middle terms will cancel each other to give 1, if and only if the hash of the original message e is equal to the hash of the received message e' , and if the CA's private key s has been correctly used, and if the signature c has been correctly received. Thus a message can be signed and verified by ECDSA.

3.9.2 *CertModule Application*

Based on our discussion of ECDSA and the implementation of ECC, we will now take a look at a Windows based application which we used to make our certificates. This application will be useful later on to make and use our own certificates as and when needed to test key exchange protocols. Although this module is not for a PDA, we are presenting it here as a part of the overall implementation since certificates will be an integral part of key-establishment protocols.

We follow a generic certificate format as suggested in [31]. A certificate $Cert_A$ consists of $\langle ID_A, m, Sig_{TA}(ID_A, m) \rangle$. Here ID_A is the unique identity of the entity A , m is the message to be signed and $Sig_{TA}()$ is the signature from the trusted authority TA by a suitable signing algorithm (in our case ECDSA). Usually the certificate is encoded in an appropriate format (e.g. PEM), but we will leave it un-encoded for the sake of convenience. The message m in our case will be the public key of entity A . Since public keys in ECC are points on the curve we will convert the x and y co-ordinates to a string and append it in the certificate. So our basic certificate will look something like

$ID_A, x \text{ co-ordinate}, y \text{ co-ordinate}, c, d$

where c and d are the two components of signature.

The CertModule application is shown in figure 3-9

The screenshot shows the CertModule application window with the following fields and values:

- Identity:** Alice
- Public Key (File):** C:\public.txt
- CA seed:** 1234AABB
- CA Keys:**
 - Private Key : 96d3d8c a8d08ecf 91fed2f9 803c80d 21859453
 - Public Key : 35cf789d b149ec04 420dec75 bda5876b 1640caeb cd519e4 99855c9c e9b6bb3d 986543ba 115926d8
- Message to be signed:** Alice, 3d553104e7ec1bf49e6182be57d6da26b0944726, 25597cccd2a3e2424ce7401a58e0e961b20afcdf
- Certificate:** Alice, 3d553104e7ec1bf49e6182be57d6da26b0944726, 25597cccd2a3e2424ce7401a58e0e961b20afcdf, 20e1dd75bb29538 6558f5e 9b9e7eb3 97bc95d2 , 38c13ea c943f3fd ca15ddb6 9f6e3218 b47c7b84
- Certificate File (Path):** C:\Alice.cert
- Generate** button

Figure 3-9 The CertModule application

We will briefly describe this application.

The *Identity* field needs a unique identity of the entity to whom we are assigning the certificate. Here we have put the string 'Alice' as the entity identifier, in practice it may be a unique number assigned to *Alice*.

The *Public Key* field is given the path of the file which has *Alice*'s public keys. We have simply stored the x and the y co-ordinates as strings in a text file *public.txt*.

The *CA seed* field requires a hexa-decimal which is used as a seed in a random number generation function. This is used to calculate the CA's public and private keys (shown in *CA Keys* box). So the CA can use this as a sort of a password.

The '*Message to be signed*' box displays the Identity and the public key of *Alice* separated by commas. The CA signs this message with its private key according to the ECDSA algorithm. The two components of the signature c and d are appended to the message to make the certificate and displayed in the *Certificate* box. The *Certificate File* box has the

path of the file in which this certificate (Alice.cert) is stored. This file is then distributed by the CA.

3.9.3 ECDSA Verification on a PDA

The certificates which we have created in 3.9.2 will be used in key exchange protocols in chapters 4 and 5. The protocol entities will be using the ECDSA verification algorithm to check if the public keys they require are correctly obtained. For relatively powerful servers or desktop PCs, the verification time is negligible as compared to the verification time needed by a PDA client. Hence in this section we will look at the time required by the PDA to perform ECDSA verification on the certificates generated in section 3.9.2.

Table 3.5 ECDSA Verification on the PDA

Field Size (Bits)	Normal ECDSA Verification(Secs)	ECDSA Verification with Pre-computations (Secs)
113	1.9	0.619
131	3.129	1.004
134	3.469	1.107
158	5.779	1.796

We can see the results of our analysis in table 3.5, while figure 3-10 shows a graphical representation which points to the general trend.

Figure 3-10 ECDSA Verification on the PDA

Thus there is an improvement by a factor of three when we use pre-computations. We used a window size of 10 while taking these readings. A lesser window size (i.e. more number of windows) might make it more efficient. However we have found from our earlier analysis that the difference will not be that much. So we present these readings as the near optimal performance of our verification algorithm for the PDA.

3.10 Summary

In this chapter, we have seen the results of the implementation of ECC on F_2^n on a PDA. We have considered two implementations, one based on polynomial basis and the other on optimal normal basis. The ONB implementation seems more efficient than the polynomial basis implementations as far as speed is concerned for all the ECC operations. We have then looked at using pre-computations for point multiplication of ONB based ECC. We have shown that with our implementation, we can speed up the multiplication to almost thrice as fast. Lastly we have used this implementation to make a ECDSA module which can be used

to generate, sign and verify certificates. These will be useful in the next chapters where we will be looking at protocols involving certificates.

4 Authentication and Key Exchange Protocols

In this chapter we will look at ways for devices to authenticate each other and protocols for key exchange. We will be looking specifically at protocols that combine these two objectives, which are called Mutually Authenticated Key Exchange Protocols or MAKEPs. The goal of these protocols is to provide the communicating parties with some assurance that they know each others true identity and at the same time have a common key known only to them [30]. This is generally the first step in establishing a secure channel for communication between two or more devices or entities. If this step is compromised in any way, the rest of the encrypted communication will also be compromised.

MAKEPs for wireless environments have greater restrictions and different properties due to the nature of the wireless environment and the devices that operate in it. As we have discussed in Chapter 1, wireless environment is by its very nature insecure and limited in bandwidth. For example an adversary could easily tap into wireless communication channels or jam other people's devices. Furthermore personal wireless devices are generally of low power, low in computational abilities and low in memory. These reasons have prevented a simple migration of cryptographic protocols from fixed networks to wireless networks for authentication and security [31]. Hence MAKEPs for wireless devices need to be designed to account for the limitations of wireless devices and the wireless environment. They need to be efficient in terms of memory, computation power and bandwidth.

In the next section we will look at general public key MAKEPs and analyse their properties. We will then look at MAKEPs specifically designed for low powered devices in wireless environments.

4.1 MAKEPs based on public key cryptography

4.1.1 Diffie-Hellman key exchange protocol

The Diffie-Hellman key exchange protocol is the earliest protocol to use public keys to generate a shared secret [32]. Almost all protocols can be seen as extensions of this protocol which try to make it more secure or which try to provide different advantages and properties. In its original form it does not provide for authentication. We will discuss it in brief in this section.

Let g be a generator of a field Z . The *principal* (or in other words, an entity taking part in the protocol) *Alice* has $a \in Z$ as a private key and $g^a \in Z$ as her public key. Similarly the principal *Bob* has $b \in Z$ as a private key and $g^b \in Z$ as his public key. By the Discrete Log Problem we discussed in Chapter 1, it is computationally infeasible to get a from g^a , if the field is large enough. The protocol can be illustrated as –

Figure 4-1 Diffie Hellman Protocol

Thus both parties now have a secret key g^{ab} which is known only to them. However this protocol suffers from a man-in-the-middle attack. Suppose there is a malicious agent *Mallory* which can intercept and send messages in the communication channel between *Alice* and *Bob*. Then we have the following scenario.

Figure 4-2 M-I-M attack on DH protocol

According to Figure 4-2, *Mallory* can operate two parallel sessions, one with *Alice* using the key $g^{ab'}$, and the other with *Bob* using the key $g^{a'b}$. *Alice* thinks she is talking to *Bob* when in fact she is talking to *Mallory* and the same case will happen with *Bob*.

This attack can be avoided by authenticating *Bob* and *Alice* to each other. For example, the Station-to-Station protocol (also called the authenticated DH protocol) uses digital signatures and public key certificates to authenticate the parties to each other [33]. The basic idea is that *Alice* and *Bob* obtain a certificate from a trusted source for their public keys. They also use their private keys to sign messages which contain their public keys. Thus even if *Mallory* is able to intercept the messages, she cannot forge the signatures as the private keys are not known to her.

4.1.2 The Massey-Omura key exchange protocol

The Massey-Omura Cryptosystem is based on Shamir's three-pass protocol. This suggests using ciphers that can be applied to a message in any order to produce the same final result. In other words the ciphers are commutable with respect to each other. We will look at an ECC based version of the Massey-Omura key exchange protocol which is based on this idea [3].

Let E be an elliptic curve of the order N defined over a Galois Field GF . Let e be a point on the curve relatively prime to N i.e. $\gcd(e, N) = 1$. Let d be the inverse of e . Here d becomes the decryption key and e becomes the encryption key. The protocol can be described as –
Let *Alice* have a key pair (e_A, d_A) and *Bob* have a key pair (e_B, d_B) . *Alice* creates a key P_K which is a point on the curve E . She then multiplies it with her encryption key.

$$P_1 = e_A P_K$$

P_1 is sent to *Bob* who multiplies it with his encryption key to get P_2 .

$$P_2 = e_B P_1$$

P_2 is now sent to *Alice* who multiplies it with her decryption key to get P_3 .

$$P_3 = d_A P_2$$

Expanding, $P_3 = d_A (e_B P_1) = d_A (e_B (e_A (P_K))) = (d_A e_A) (e_B P_K) = (1) (e_B P_K)$

since d_A is the inverse of e_A therefore $(d_A e_A) = 1$.

Hence $P_3 = (e_B P_K)$. This is sent to *Bob*. *Bob* multiplies P_3 with his decryption key d_B to get the original key P_K , since –

$$d_B (P_3) = d_B (e_B P_K) = (d_B e_B) P_K = (1) P_K.$$

Thus the secret key P_K is transferred securely to *Bob*.

The Massey-Omura protocol foils the man-in-the-middle attack. However it has a few disadvantages –

1. There are a lot of communication steps to transfer one piece of data i.e. P_K .
2. Every step is computationally expensive as it involves an elliptic curve multiplication.
3. The initial setup of ‘ e ’ and its inverse ‘ d ’ requires computationally expensive large integer mathematics.

4.1.3 Menezes-Qu-Vanstone key agreement scheme

This scheme includes the DH protocol as a sub-set [3]. Besides providing authentication and secure key-exchange, it also provides forward secrecy i.e. even if any previous messages have been cracked by an adversary; it will take equal effort to crack any other messages. We will look at the IEEE P1363 draft description of the MQV protocol in brief [3]. The protocol is called ECKAS-MQV or ‘Elliptic Curve Key Agreement Scheme – Menezes Qu Vanstone’ in the draft. The following parameters are defined –

The elliptic curve E , a base point P , the order of the curve ‘ n ’, d_A and d_B are the secret keys of the two parties and Q_A and Q_B are their public keys, where –

$$Q_A = d_A P \text{ and } Q_B = d_B P$$

R_A and R_B are the ephemeral (temporary) keys of *Alice* and *Bob* respectively, were –

$$R_A = k_A P \text{ and } R_B = k_B P$$

Here k_A and k_B are random numbers generated by *Alice* and *Bob* respectively.

We specify the coordinates of the points in the following way –

$$Q = (a, b); R = (x, y); Q' = (a', b'); R' = (x', y')$$

The initial calculation done by *Alice* is –

$$s = k + xad \mod n$$

and by *Bob* is –

$$s' = k' + x'a'd'$$

now *Alice* will calculate the value U using *Bob's* data as –

$$U = R' + x'a'Q'$$

And *Bob* will calculate U' using *Alice's* data as –

$$U' = R + xad$$

Alice will calculate the shared secret K as –

$$K = sU$$

Bob will calculate the same K as –

$$K = s'U'$$

Explanation of the protocol:

We can expand U as

$$U = (R' + x'a'Q') = (k'P + x'a'd'P) = (k' + x'a'd')P$$

$$\text{Similarly } U' = (k + xad)P$$

$$\text{Thus the shared secret } K = sU = s'U' = (k + xad)(k' + x'a'd')P$$

The MQV protocol has the DH protocol at its core [3]. Additional security is incorporated by including the private keys d and d' in the formula. Since these values are secret, an attacker cannot spoof K which can be calculated by only *Alice* and *Bob*. The freshness of the key K is guaranteed by using new random values k and k' . Compromise of a shared secret will not compromise another shared secrets and k and k' will be different every time, ensuring forward secrecy.

We have seen some major authentication protocols in this section. In the next section we will look at a generic MAKEP architecture.

4.2 A Generic MAKEP architecture

Wong and Chan [30] have illustrated a generic MAKEP structure by identifying the main elements from the DH protocol and the SSL authentication protocol. We will look at this generic MAKEP in detail in this section. This will serve as a good reference for discussing specialized MAKEPs for wireless devices.

Let (PK_A, SK_A) be the public-key private-key pair of *Alice* and (PK_B, SK_B) be that of *Bob*. Sig_{TA} is the signature algorithm of a trusted authority TA, and Ver_{TA} is the corresponding verification algorithm. A certificate used by *Alice* to publish her public key will be $Cert_A = \langle ID_A, m, Sig_{TA}(ID_A, m) \rangle$ where ID_A is the identification data of *Alice* and 'm' is some message that incorporates *Alice*'s public key PK_A .

Figure 4-3 Generic MAKEP

The final session key in figure 4-3 is σ . In the protocol there are two challenge-response pairs (r_A, S_B) and (r_B, S_A) . In response to a random number r_A , generated by *Alice*, *Bob* responds with $S_B = \{ID_A, r_A, \beta\}_{SK_B}$. This is r_A with some other information signed by *Bob*'s

private key. This will be verified by *Alice* using *Bob*'s public key. After verification *Alice* will check if the value of r_A sent by *Bob* in S_B tallies with the value she had originally sent to *Bob*. Since r_A is signed by *Bob*'s private key and only *Bob* knows his private key, this effectively authenticates *Bob* to *Alice*. *Alice* is authenticated to *Bob* by (r_B, S_A) in a similar manner.

The encrypted session key elements of *Alice* and *Bob*, α and β respectively, are bound to the corresponding challenge nonces r_A and r_B respectively. This binding is done by incorporating the pairs (α, r_B) in S_A and (β, r_A) in S_B . This technique detects man-in-the-middle attacks as well as replay attacks as a different random number will be chosen by the two parties at the start of a new session. So the chances of the same pair (α, r_B) or (β, r_A) in a new session will be negligible. Hence replay or man-in-the-middle attacks can be detected.

The session key contributions c_A and c_B of *Alice* and *Bob* respectively, ensure freshness of the session key. The session key is finally created by combining the random c_A and c_B using a function f . This leads to forward secrecy i.e. even if earlier session keys are compromised, other sessions will be secure. Also since a new session key is used every time, cryptanalysis is more difficult.

Having looked at a generic MAKEP protocol and having seen the main features necessary for secure authentication and key exchange, we will now turn to the problem of designing MAKEPs suitable for wireless devices. In the next section we will look at four algorithms specifically designed for constrained devices in wireless environments.

4.3 MAKEPs for constrained devices

Low computation power and low memory capacity is the characteristic of wireless devices like mobile phones and pagers. Devices like PDAs have somewhat more memory and computational power but are more bulky and are general purpose devices i.e. they are not used specifically for wireless communication only. There are two main ways in which these devices communicate with each other.

One way is shown in figure 4-4. This case arises when the client needs to go through a server before communicating with another client, e.g. mobile phones. In this case the MAKEP is

negotiated through the server. The clients need to authenticate each other to the server. After the MAKEP protocol a common session key is created. Figure 4-4 shows the clients interacting through the server to create a common key which they then use to encrypt information that they send to each other in peer-to-peer mode. We could also have a client and a server creating a session key between themselves and this key can be securely transmitted to another client for secure peer-to-peer messaging.

Figure 4-4 Powerful Server - Low powered clients scenario

Figure 4-5 shows another case where there is no server to mediate between the clients. The low powered clients need to carry out authentication and key exchange by themselves. This scenario could be applicable when PDAs need to establish peer-to-peer connection between themselves for some data transfer.

Figure 4-5 Peer-to-Peer MAKEP for wireless devices

In section 4.2 we have seen the primitive elements of a generic MAKEP. We need to incorporate these elements into a MAKEP that caters to constrained systems operating in wireless environments in the 2 cases mentioned above. In general, an attempt is made to transfer parts of the protocol that require more or complicated computations to the server. Another important point to be incorporated is that symmetric key calculations are much faster than the corresponding public key calculation, as we have discussed in chapter 1. Attempts are also made to transfer most of the calculations to a pre-computational stage i.e. before the actual run of the protocol; so that there is no computational delay on the client side during the protocol.

4.4 Case studies of MAKEPs for constrained systems

We will first look at the Jakobsson-Pointcheval protocol for low power clients and powerful servers. We will look at an attack on this protocol as described in [30].

4.4.1 Jakobsson-Pointcheval protocol

The protocol is shown in figure 4-6. Here *Alice* is a low powered client and *Bob* is a powerful server. *Alice* decides the secret session key σ before the run of the protocol. σ is computed using K , where $K = (g^y)^x$ is computed using the DH key exchange protocol. *Alice* is authenticated to *Bob* using r and d . $r = g^t$ is passed to *Bob* from *Alice*. *Alice* then calculates $d = t - ea \bmod q$ using her private key a . *Bob* uses *Alice*'s public key g^a to verify the authenticity. If the values are correct then

$$g^d (g^a)^e = g^{(t-ea)} g^{ea} = g^t = r$$

(4.1)

The protocol is executed in 3 passes and most of the computation on the client side is done before the actual start of the protocol. However *Bob*'s challenge number e is passed in clear form to *Alice*, without binding it to any previous messages or encrypting it. Due to this there is an interleaving attack possible on this protocol [30]. H_0, H_1, H_2 are hash functions. It is assumed that *Alice* knows *Bob*'s public key.

Figure 4-6 Jakobsson-Pointcheval protocol

The attack is shown in figure 4-7.

In this attack, *Mallory* acts as a malicious agent who can intercept and inject messages in the communication link between *Alice* and *Bob*. When *Alice* initiates a session with *Bob* with the pair (X, r) , *Mallory* intercepts the message and starts another session with *Bob* with the pair (X', r) of which r is taken from *Alice*'s message and X' is calculated using *Mallory*'s local random value of x' . *Bob* thinks both session runs are initiated by *Alice*. *Bob* returns the pairs (R', e) and (R'', e) for the two sessions. *Mallory* intercepts these and creates a new pair (R', e') using R' from the first session and e' from the second session. *Alice* will check if $R' = R$

and will find that it matches. She will now calculate d' , using the e' value which was meant for the second session initiated by Mallory. This value will be sent to *Bob* by *Mallory* as the second session parameter from *Alice*. *Bob* will not get any reply for the first session and will terminate it after a while. *Mallory* can now operate the second session with *Bob* pretending to be *Alice*.

Figure 4.7 Interleaving attack on the Jakobsson-Pointcheval protocol

As discussed before, the attack is possible because the value of server challenge variable e is not bound to any previous message, nor is it signed or encrypted. But the tradeoff of such an action is that some of the pre-computation would be shifted into the actual protocol; and this may decrease its efficiency. For e.g. R and R' could be calculated as $H_2(g^y, X, K, e)$, thus binding e to R . Now this means A cannot pre-compute R , and according to Wong and Chan [31] this will decrease the speed and efficiency of the protocol.

Observations

1. The protocol does no public key computations at run time. The only run time computations are a few hash functions and computations involving field element additions/multiplications, which will be very fast.

2. The interleaving attack exposes obvious security holes in the protocol.
3. Wong and Chan [31] mention that removing the security weakness by incorporating the variable e in the H2 hash function; will add to the runtime cost. But from our results (shown in chapter 5), we found that hashing functions take negligible time as compared to public key functions, so this addition to the protocol will not make much difference.
4. The final session key is calculated as the hash of the values (g^y, X, K) which is (g^y, g^x, g^{yx}) . Out of these values x is the random value given by the client. So the final session key is only calculated using the client's contribution and the server does not decide what the final key will be. This may be a weakness for replay attacks as discussed in the previous sections.

4.4.2 Server Specific MAKEP

This protocol was proposed in [31] as an efficient MAKEP for communication between a low powered wireless client and a powerful server. The main feature of this MAKEP is that it eliminates any public key cryptographic operations on the client side. The client side performs only symmetric key operations. Symmetric key operations are always faster than equivalent public key operations, as we have seen in Chapter 1. Moreover public key operations can be inefficient on low powered devices. Hence by incorporating only symmetric key on the client side, the protocol becomes more efficient as far as the computational speed is concerned.

The protocol is shown in Figure 4-8. It uses server specific certificates to authenticate *Alice* (the low powered client) to *Bob* (the powerful server). The certificate can be described as $Cert_A^B = \langle ID_A, \{K_A\}_{PK_B}, Sig_{TA}(ID_A, \{K_A\}_{PK_B}) \rangle$. Here K_A is *Alice*'s long lived symmetric key. PK_B is the public key of *B*.

Figure 4-8: Server-Specific MAKEP protocol

It is assumed that the server’s public key PK_B is well known. On receiving the first message from *Alice*, *Bob* verifies the TA’s signature and obtains *Alice*’s long lived symmetric key K_A , by decrypting $\{K_A\}_{PK_B}$ from the certificate by using his private key SK_B . This step also authenticates *Alice*. *Bob* then encrypts a new random value r_B along with *Alice*’s random value r_A and his identity ID_B , with *Alice*’s symmetric key K_A . This step authenticates *Bob*. Now the session key σ is calculated as $r_A \oplus r_B$.

The protocol completes in 3 steps with hardly any pre-computations. Another major advantage is that the client uses only symmetric-key cryptography with her secret key K_A . This increases its speed and efficiency as compared to other comparable MAKEPs that use public-key cryptography. The freshness of the session key is maintained by using the random numbers r_A and r_B . Unlike the Jakobsson-Pointcheval, where only the client computes the session key σ , here we use client and server contributions, r_A and r_B respectively, to calculate σ . This increases the strength of the key.

Observations

There are some disadvantages to this protocol in its raw form –

1. The protocol needs server specific certificates. For n servers it will need n distinct certificates. This creates scalability issues.
2. The client's long lived secret key K_A is known to the server. If the server is malicious it can create non existent sessions and there is no way for *Alice* to deny these occurred. i.e. a malicious server can impersonate its clients to create false runs of the protocol.
3. Even if the server is trustworthy, it has access to the client's secret key K_A . If the server's security is compromised in any way it implies the client's security is also compromised since a hacker can get K_A from *Bob*'s memory.
4. The protocol needs the client to have a unique long lived secret key for every server it communicates with. This adds to the scalability problem. If the client has a single long lived secret key which it shares with all its servers, there is a obvious man-in-the-middle attack which can be played on a session between *Alice* and *Bob*, by a malicious server *Mallory*. We assume that *Mallory* knows *Alice*'s secret key K_A from an earlier transaction. The man-in-the-middle attack is shown in figure 4.8. *Mallory* just replaces r_B by r_B' in the message from *Bob* to *Alice*. She is now able to operate parallel sessions by calculating σ and σ' . Using σ , she can talk to *Bob* pretending to be *Alice*, and using σ' she can talk to *Alice* pretending to be *Bob*. The only way to prevent this scenario would be for *Alice* to use separate symmetric keys while dealing with *Bob* and *Mallory*. However, the transfer of the actual symmetric key still creates security hazards as described in point 3. If a hacker gets K_A from *B*, s/he could perform the attack in figure 4-9.

Figure 4-9: M-I-M attack on Server Specific MAKEP protocol

4.4.3 Linear MAKEP

The Linear MAKEP was proposed as a solution to some of the issues affecting Server-Specific MAKEP, specifically the impersonation weakness and the scalability problem [31]. However this protocol requires the client to have a number of private-key public key pairs. If $(a_1, a_2, \dots, a_{2n})$ are the private keys of Alice, her corresponding secret keys will be $(g^{a_1}, g^{a_2}, \dots, g^{a_{2n}})$. Alice now needs to obtain a signature for each pair of public keys $(g^{a_{2i-1}}, g^{a_{2i}})$ where $1 \leq i \leq n$. The signature will be of the form $\{Sig_{TA}(ID_A, g^{a_{2i-1}}, g^{a_{2i}})\}_{1 \leq i \leq n}$. Here Sig_{TA} is the signature of a trusted authority TA which can be verified by the server.

The protocol is shown in figure 4-10. The client Alice will construct a certificate $Cert_A^i = \langle ID_A, g^{a_{2i-1}}, g^{a_{2i}}, Sig_{TA}(ID_A, g^{a_{2i-1}}, g^{a_{2i}}) \rangle$ which is sent to the server Bob. Bob will verify the signature and reply with his random value r_B . On receiving r_B , Alice will compute her own random r_A , and encrypt it with Bob's public key so that she has $x = \{r_A\}_{PK_B}$. This operation will be computationally more expensive than the corresponding operation in the Server-Specific MAKEP, since we are dealing with public key encryption, which is

approximately a thousand times slower than symmetric key encryption. *Alice* then calculates the variable y as

$$y = a_{2i-1}(x \oplus r_B) + a_{2i} \bmod (p-1).$$

p is the large prime number that defines the field Z_p in which all the elements belong. The pair (x, y) is sent to the server in the third step. The new session key is calculated as

$$\sigma = r_A \oplus y.$$

Bob will authenticate *Alice*, and check if the variable y is valid by

checking if $(g^{a_{2i-1}})^{x \oplus r_B} g^{a_{2i}}$ equals g^y . Note that *Bob* has already received the pair $(g^{a_{2i-1}}, g^{a_{2i}})$ and has verified that they are *Alice*'s public keys. He also knows x and r_B from the previous steps.

Figure 4-10 Linear MAKEP

The client's random r_A , is encrypted under *Bob*'s public key and is also combined with *Alice*'s secret key and server random r_B while calculating y . So all the steps are bound to each other and the protocol is designed in such a way both the client and the server are authenticated to each other without any weakness that can be exploited by interleaving or m-i-m attacks.

Weaknesses in the protocol

There are two serious weaknesses in this protocol as shown in [49].

One problem is that *Bob* does not check if the decrypted value of x is a valid one. This can lead to an unknown key share attack by which an attacker *Eve*, who has previously eavesdropped on a valid conversation between *Alice* and *Bob*, can start a session with *Bob* pretending to be *Alice*.

Let us assume that *Eve* has obtained $Cert'_A$, x , y and r_B from a previous run of the protocol. *Eve* can start another session with *Bob* and obtain his new random value $r_{B'}$. She will then put the corresponding signature $x' = x \oplus r_B \oplus r_{B'}$. So now $x' \oplus r_{B'} = x \oplus r_B$ so the value of y will be same in both cases. *Eve* will send (x', y) to *Bob*. *Bob* will verify if y satisfies the verification equation we have looked at previously, and then decrypt x' using his secret key. The decryption process will obviously not give a correct output but *Bob* has no way of knowing that. He will assume it's a valid result and assume that a session has been established with *Alice*.

A more serious attack can occur if *Eve* can capture two protocol runs in which the same key pairs (a_{2i-1}, a_{2i}) have been used. In this case *Eve* will use the values (r_B, x, y) from the first run and $(r_{B'}, x', y')$ from the second run to get a_{2i-1} using $(y-y')/(x \oplus r_B - x' \oplus r_{B'})$, and then get a_{2i} . Thus secret keys of *Alice* can be compromised. The authors of [49] recommend using $x \oplus r_B \oplus r_A$ to calculate y . This prevents an eaves-dropper from using this attack. However the server *Bob* still knows the value of r_A and he can still obtain the secret keys of the client.

We would also like to point out a relatively smaller weakness which is similar to the one we will discuss in more detail in section 4.5.2. The protocol aims at creating fresh session keys, but the client has the power to decide what the final session key will be. It is solely up to the client to create the final session key so if the aim of the protocol was to create a key with equal random contributions from the client and the server, it is clearly not fulfilled here. This is because the client has the random value of the server before it commits its own random value. The solution is the same as the one in section 4.5.2. *Bob* sends $h(r_B)$ instead of r_B in the second message, where 'h()' is a hash function. In the final step, *Bob* sends the original

r_B , encrypted along with x under σ and the client *Alice* checks if the has value matches in both cases. Thus we are taking away the power of the client to decide the session key.

4.4.4 Client Server MAKEP

The authors of the Server-Specific protocol proposed a more robust protocol in [30] to deal with a powerful server – low powered client scenario. The protocol is shown in figure 4-11. Unlike the Server Specific MAKEP, this protocol uses public-key cryptography on the client side as well as the server side. Unlike the Linear MAKEP, this protocol has only one pair of public-private keys on the client side.

Alice chooses a random value r_A as its contribution towards creating the shared key, and encrypts it using *Bob*'s well known public key PK_B . *Alice* also computes a random value ' b ' and sends $\beta = g^b$ to *Bob*. This will be used to bind this message to later messages to prevent interleaving attacks of the type seen in section 4.3.1.

Bob replies by sending his shared key contribution r_B , encrypted with *Alice*'s contribution r_A . This authenticates *Bob* to *Alice*. Since r_A had been encrypted under *Bob*'s private key, only *Bob* will know r_A , and r_B encrypted under r_A implies that *Bob* is replying to the current session. *Alice* sends the value $y = ah(\sigma) + b \bmod q$ (q is the prime number used to define the field F_q) to *Bob* where σ is computed from r_A and r_B . *Bob* checks if $g^y = (g^a)^{h(\sigma)}\beta$. Now $g^y = g^{(ah(\sigma) + b)} = (g^a)^{h(\sigma)}g^b = (g^a)^{h(\sigma)}\beta$. *Bob* knows g^a from the certificate and he has received β in the previous message. Thus *Alice* is authenticated as the secret key portion of ' a ' in the value of y can only be computed by *Alice*, and the β value effectively binds the earlier and later messages, as *Alice* will compute a new value of ' b ' for each session.

Figure 4-11 Client Server MAKEP

This protocol removes the scalability and security issues of the Server-Specific protocol by using public key cryptography. Public key encryption has to be used in the pre-computation stage. During the actual run of the protocol the client does not perform any computationally expensive operations. The protocol has been shown to be secure against attacks by using the Bellare-Rogaway [52] model. Thus this protocol is efficient and secure.

Observations

1. We observed that the protocol used the public/private keys of the server only for encrypting and decrypting data. It does not make use of the unique mathematical relation between the public and private key of the server for authentication.
2. The third step is necessary because *Alice* has not been authenticated to *Bob*. Note that in the first step, when *Alice* sends “ $Cert_A, \beta, x$ ” to *Bob*, this does not authenticate *Alice*, as anyone could send this message by capturing previous protocol runs. The dotted line shows the binding between the first and the third step by b and β . The third step combines the secret key (a) and the session random (b) value of *Alice* to authenticate her to *Bob*. If *Alice* was authenticated in the first step itself, the third step will not be necessary.

3. The public key encryption of the client random r_A with the *Bob*'s public key is computationally expensive and takes a substantial amount of time. If this operation was somehow performed by symmetric key algorithms, the time taken would be much lesser.
4. The server knows the client random r_A before he proposes his own random r_B . So the power of deciding the final session key lies with the server and the contribution by the client can be made ineffective. A detailed discussion on this weakness will be done in chapter 5.

We propose that by using the keys of the client and the server to make a Diffie-Hellman key pair, described in section 4.1.1, in the first message we can have a protocol that can be executed in just 2 steps. By doing this, we will show that have effectively removed the need of a separate binding 'b' and β , between the first and the last message. This further removes the need for the server to perform exponential calculations to verify and authenticate the client. By making a few more adjustments we can also modify this method to suit a peer to peer scenario in which both peers are low powered clients. We will present this new protocol along with its implementation in the next chapter.

4.5 A new MAKEP protocol

4.5.1 Protocol Description

We will be using the same assumptions as the Client-Server protocol i.e. the client knows the server's public key before the run of the protocol. This fits in the scenario of a few servers handling a multitude of 'thin' clients (i.e. low power clients which are expected to perform minimal computations). Typically in such a scenario a client will cache a server's certificate for future use but a server will not, as it has to deal with many clients, so the processing and memory involved in storing and retrieving the client's certificate may not be very efficient. If the client does not have the certificate, it can retrieve it from a certificate authority, but we will not discuss this portion as it is outside the scope of the thesis.

The protocol is shown in figure 4-12. Let g be the generator of the field over which the DH problem is intractable.

Figure 4-12: A new MAKEP for wireless devices.

The client *Alice* has a private key a and a public key g^a and the server *Bob* has a private key ' y ' with a public key g^y . We assume that the public key of the server is well known and the client has a certificate $Cert_A$ from a trusted authority where $Cert_A = \langle \text{ID-A}, g^a, \text{Sig}_{TA}(\text{ID-A}, g^a) \rangle$. Here ID-A is the identity of *Alice* and Sig_{TA} is the signature of a trusted authority which can be easily verified. Unlike the Server-Specific MAKEP protocol in [31] we do not use a server-specific certificate and avoid scaling issues.

The protocol is shown in figure 4-12. *Alice* first computes a random number r . She then calculates the DH key $g^{(ya)}$ using her private key a and Bob's public key g^y . Our protocol is unique in the sense that we propose that this DH key be converted to an AES key. We leave the conversion operation open to the implementation. The main concern is that it should be derived from the DH key and both the server and the client should follow the same pattern. For example, we could take the hash of the DH key and take the first 128 bits as the AES key. Although calculating the DH key is computationally expensive since it involves an exponentiation (or a point multiplication as we will see in the case of the ECC version), the derivation of AES key from it, and the subsequent use of the AES symmetric key algorithm

in encryption and decryption of data leads to a substantial improvement in time as we will see in the later sections.

Alice sends her random contribution r encrypted under the AES key to *Bob* along with her certificate. *Bob* verifies *Alice*'s public key g^a and uses it to calculate the DH key $g^{(ya)}$ using his own private key y . He also converts the DH key to an AES key and uses it to decrypt the message and get *Alice*'s random number r . *Bob* computes his own random number r' and uses r as an AES key to encrypt it and send it to *Alice*. *Alice* can decrypt this message since she already knows her random r .

σ is the new session key and H is a cryptographic hash function. By using r (the client random) and r' (the server random) we are maintaining the freshness of the session key and the $g^{(ya)}$ value is used for authentication *Alice* and *Bob* to each other, as well as preventing any other party from mounting any replay attacks or Denial of Service attacks on *Alice* or *Bob*. Since we have authenticated *Alice* to *Bob* in the first step, we don't need a third step like in [31]. It should be noted that we are not using any public key encryption/decryption at the client side once the protocol has started to run. All public key methods at the client side may be pre-computed. Another point to note is that we are using AES, which is a symmetric block cipher, to encrypt r using $g^{(ya)}$. The encryption and decryption is thus much faster for the client. There is just one modular exponentiation for the client which can be pre-computed.

Another variation of this protocol could be made using elliptic curve cryptography (ECC). As we have seen in chapter 2, a 1024 bit RSA key is comparable to only a 164 bit ECC key i.e. it provides the same level of security so with the shorter key length the protocol will be much faster. In the ECC version of the protocol all the variables will now be points on an elliptic curve E . *Alice*'s private and public keys will be the integer a and the point Pa (obtained by point multiplication of the base point P and the secret key a), while *Bob*'s keys will be y and Py . P is a publicly known base point. Instead of encrypting r with $g^{(ya)}$ we will encrypt it with $(Py)a$. The rest of the protocol remains similar to the original one.

In [30] and [31] results have been demonstrated by using 512 bit modulus for the public key cryptography part. In reality 1024 bit is recommended to maintain an intractable level of

security in the discrete log problem (DLP). Using 160 bit ECC which is comparable to the 1024 bit RSA, the protocol becomes more efficient and more secure.

Advantages

Our protocol offers many advantages over similar protocols for a scenario where a low power client exchanges keys with a much more powerful server. Unlike [31] we don't have server-specific certificates which lead to greater scalability. We have used a public key private key pair for the client but we do not actually use public-key cryptography for encryption unlike in [31, 32]. The client does all the encryption and decryption using symmetric keys which is faster than public key cryptography. Similar to the Client-Server MAKEP in [31], we have only 1 modular exponentiation on the client side and this can be pre-computed. However during the run-time of our protocol, the client has only 1 symmetric key decryption; as opposed to the 1 symmetric key decryption and 1 modular multiplication and in [31]. We are reducing the key size of our public keys to 160 bits by using an ECC variant of the protocol, while the protocols in [30] and [31] use a 512 bit public key cryptography. As we have noted in chapter 1, 512 bit RSA is no longer considered intractable.

4.5.2 Protocol Weaknesses

Our protocol is based on the Client-Server MAKEP which has been shown to be secure against interleaving and man-in-the-middle attack. We have modified the Client-Server MAKEP by using the DH key exchange as a subpart in the first step to authenticate the client in the first step itself, thus vastly improving on the speed (as we will see in chapter 5 while discussing the implementation aspects). Also our all the variables of our first and second steps are bound to each other, thus preventing interleaving and other similar attacks.

There is however a problem with protocol based on trust issues. A weakness has been pointed for the Server MAKEP protocol in [49], which can also be seen in our protocol. The weakness is that the server has control over the session key. If we look at the server side computations before the second message is sent in figure 4-12, the server performs the following operations

1. Calculate $(g^a)^y$
2. Decrypt r from $\{r\}_g^{(ya)}$

3. $r' = \text{Random value}$

The server can decide before hand which session key (σ) will be used by calculating $r' = (r \oplus \sigma)$ and it can do this because it knows the value of the client random before the final session key is calculated. So the weakness is in allowing the server to choose the session keys that he wants.

While designing the protocol we did encounter this problem but we did not think it as an exploitable weakness, in the sense that there was no way for the server to obtain or compromise the client's secret key from this weakness. The problem is with the level of trust. The client certainly does not trust anyone enough to give up her secret key, but if the server was not to be trusted in creating his own random contributions and not sharing the session key with other entities, then we are faced with much bigger problems than the problem of the server having the power to decide what the session key is. If we cannot trust the server enough not to create exploitable session keys, can we trust it at all with keeping any of the conversation it has with the client a secret? If that much level of trust cannot exist no protocol could be secure.

This was our view at the time of designing the protocol and the Server-Specific MAKEP and the Client-Server MAKEP also seem to have followed similar design strategy. However we do realize there maybe instances or scenarios in which this weakness might be exploitable and we offer a very simple modification to remove this weakness. The modifications are shown in figure 4-13.

Another weakness that may be exploitable will be "naïve use" of the AES algorithm. By this we mean that if AES decryption in the second step is carried out by the server blindly without verifying if the decrypted result is valid, the server can be fooled into thinking it has successfully managed to establish a session key.

More specifically, if a malicious agent, say *Mallory*, has managed to get *Alice*'s certificate and the value $\{h(r_A)\}g^{r_A}$, she can start a session pretending to be *Alice*. *Bob* will reply with his random value encrypted by the hashed value received from *Mallory* according to the protocol. However *Mallory* will not be able to decrypt it since she does not know the DH

key g^{ya} . But if *Bob* naively decrypts using AES, *Mallory* can send him any message, say X , and *Bob* will decrypt it to some wrong value thinking he is getting correct results. At the end of the transaction, *Bob* thinks he has a valid session key established with *Alice*. However *Mallory* is not able to get the session key herself and will not be able to continue the communication pretending to be *Alice* anymore. But the weakness does exist in the sense that *Bob* can be fooled into thinking he has established a session key with *Alice*.

4.5.3 A more secure version

We can easily overcome this by adding a predetermined value before the random contributions before encrypting them. By doing this, we force the participants to check if the decryption has lead to a valid result. For example, *Alice* will send $Cert_A, \{ID_A, h(r)\}_g^{(ya)}$ and *Bob* will reply with $\{ID_B, r'\}_{h(r)}$.

Figure 4-13 A more secure version

As can be seen from figure 4-13, an extra step has been added to the protocol. In the first step we send $h(r)$, which is the hashed value of the client random r . So now the server *Bob* does not know the actual client random before calculating his own random contribution r' .

So there is no way for the server to decide the session key x before hand. In the third step we send the actual client random r , encrypted with a symmetric key derived from the DH key $g^{(ya)}$ and the server random r' . Please note that this will also be a symmetric key encryption and hence almost negligible in comparison to the public key operations like deriving the DH key in step 1 (we will see that it is almost 500 times faster in the implementation section). So the addition of the third step will not cause any difference in the speed of the protocol.

The value y , which is used as a 128 bit AES key, can be computed as follows:

$y = \text{First 128 bits of } (SHAI(g^{ya}) \oplus SHAI(r') \oplus h(r))$. We do this to randomize the data contributions from the individual elements.

We have to of course guard against interleaving attacks since we have added another step here. We do this by using the DH key (that authenticates the client and binds the first step to the third) and the r' value from the server (this authenticates the server and binds the second step to the third). These two variables are combined to give us a symmetric key which is used for encryption and decryption on the client and the server side respectively.

By forcing both the entities to verify that the decryption has taken place successfully by checking if the first part of the decrypted message contains the identity of the other entity, we have removed the risk of naïve decryption attacks.

Thus we have successfully removed the weaknesses.

We would also like to mention that our client's secret key is in no way related to this weakness because of the design of our protocol. It only affects the session key if the server is corrupt and we have shown a simple way of removing the weakness. However for the Client-Server MAKEP we note there may be an exploit based on this weakness.

From figure 4-11, we note that the client calculate the variable y to send to the server in the third message.

$$y = ah(\sigma) + b$$

We have already seen that the server can force the σ value to whatever it wants. So the server could gather any number of values from the past transactions with the client

$$\begin{array}{ll}
y_1 = ah(\sigma) + b_1 & (\text{known values: } y_1, h(\sigma), g^a, g^{b_1}) \\
y_2 = ah(\sigma) + b_2 & (\text{known values: } y_2, h(\sigma), g^a, g^{b_2}) \\
y_3 = ah(\sigma) + b_3 & (\text{known values: } y_3, h(\sigma), g^a, g^{b_3}) \\
\dots &
\end{array}$$

we note that the client's secret key is involved in this equation along with the session key which has been predetermined by the server. These equations have non unique solutions and it would be interesting to see if there could an exploit which uses this information to compromise the client's secret key. If this is possible there is indeed a big security flaw in the Client-Server MAKEP. However in our case our secret key is involved only in establishing a DH key and the DH protocol has been shown to be secure. Hence even with this weakness our client's secret key is beyond compromise.

One point to note is that if the DH key is compromised, the whole protocol is compromised. Since it is possible to get the client random and server random if the DH key is known. So the client and the server must take care that the DH key is not cached or if it is, it is stored securely encrypted in some form.

4.6 Summary

In this chapter we have looked at the major key exchange and authentication protocols. We have also looked at key exchange protocols designed for low powered clients. We have observed their key features and noted their weaknesses. From this we have come up with a new protocol which is more efficient in terms of speed and computational processing on the client.

5 Implementation and Evaluation of Protocols

In this chapter, we will present a new protocol for authenticated key establishment for low power clients and powerful servers. We will also discuss the implementation of the algorithms that we will propose. We have at this point in time implemented two of the algorithms that perform key exchange and authentication for low power devices. The first is the new Mutual Authentication and Key Exchange (MAKEP) protocol for low powered devices, that we have proposed; and the second is the Client-Server MAKEP protocol described in [30], which is one of the more recent algorithms we have encountered dealing with this topic. The Client Server MAKEP is more efficient than similar algorithms that have been proposed in the past few years, so we feel it is an ideal algorithm to compare against. The first section of our report deals with the implementation of our algorithm on an IPAQ PDA client. The server is a Pentium Fujitsu laptop and the communication between the client and the server is over IrDA and wireless LAN. We have used the ECC implementation described in Chapter 2 for public key cryptography and AES implementations from [38], [47] to implement symmetric key functions.

In section 5.1 we will present our protocol and its ECC version. We will also discuss its weaknesses and how we can avoid them. In section 5.2 we will look at the implementation of the AES algorithm since we will be using it to perform symmetric key operations in our protocol. In sections 5.3 and 5.4 we will describe the implementation our algorithm and of the Client-Server MAKEP based on ECC. The Client-Server MAKEP protocol as described by its authors is not in ECC so we had to come up with a parallel ECC version of it. The basic idea of the algorithm however is maintained in the ECC version. In section 5.5 we look

at the results of both the implementations for different key sizes. In section 5.6 we extend our algorithm implementation to operate in a peer-to-peer mode in which both devices are low-powered clients. Note that for doing the comparisons between the protocols, we have used the original versions and not the more secure versions we described in section 5.1.2. However whichever version we use, for our protocol the timings would remain the same as the additional modifications like hashing the random value and encrypting using symmetric keys will take negligible time. Finally in section 5.8 we present a PDA based application we developed to demonstrate the use of our protocol in the real world.

5.1 AES implementation

We have selected AES (Advanced Encryption Standard) to be the symmetric key algorithm in our protocol. AES is a 128 bit block cipher algorithm, which was originally known as ‘Rijndael’ (after its authors Joan Daemen and Vincent Rijmen). The algorithm was selected by NIST (US Government’s National Institute of Standards and Technology) as a new symmetric cipher to be called AES amongst much more acclaimed names like TwoFish and RC6 in May 2000 [46]. We will look at the major components of AES encryption and decryption modules in this section. The interested reader may refer to the book by the creators of Rijndael for a more detailed explanation [48].

5.1.1 Description

The encryption process is shown in Figure 5-1. The state matrix contains a 128 bit data that is encrypted into the cipher-text. The cipher-key matrix contains the key data. Please note that each box in the matrix represents a byte. 10 ‘Round key’ matrices are calculated from the cipher-key before the actual encryption from the key data in an ‘Initialisation’ function.

Since we are using a 128 bit key with a 128 bit data block size the encryption will take in 10 rounds. Initially the state matrix is added (XORed with) to the cipher-key matrix by the `AddRoundKey()` function.

Figure 5-1: The AES encryption process

After this the resulting state-matrix is put through the 4 functions shown in the middle part of the diagram 9 times.

SubBytes(): This function substitutes the bytes in the state matrix using ‘S boxes’ which are substitution matrices that form a part of the AES algorithm.

ShiftRows(): This function rotates the rows of the state matrix. There is no rotation in the first row, 1 rotation in the second row, 2 rotations in the third row and so on.

MixColumn(): This function multiplies each column with a Mixing Matrix which is a part of the AES algorithm.

AddRoundKey(): This function adds the corresponding RoundKey matrix for that round to the state matrix.

The final round is similar except the state-matrix is not put through the MixColumns() function.

After the tenth round the output is the cipher text. The decryption process is similar.

5.1.2 Creation of AES keys from Key data

In this section we will look at some issues in creating 128 bit AES keys from ‘key-material’ in the protocols described in chapter 4. The key-material we will be using consists of ECC-field size length random numbers or a Diffie-Hellman key. We hash this key material using SHA1 and take the first 128 bits as the AES key. We perform a hash operation first in case the key material is less than 128 bits, in which case we may have to pad the remaining bits with some default values. To prevent this we hash the key-material first to get a 160 bit output from which we extract the first 128 bits as the key.

We can use the generated key directly with out any further checks since AES has no known weak keys [51]. The nature of the cipher and its inverse eliminate the possibility of weak and semi weak keys such as those that exist in DES. Similarly there are no weak keys of the type that are caused by the dependence of the non-linear operations on the key value (since all the non-linearity in AES is in the fixed S-Box. For a complete discussion on this issue, the interested reader can read the Rijndael candidate-proposal for AES [52]. In fact, there are some recommendations that weak-key checks should not be performed for AES, in dynamic key-management as it is considered unnecessary.

5.2 Implementation of our algorithm

In Chapter 4, we have looked at a generic version of the algorithm and we have mentioned that this can be easily translated into an ECC version. We give the ECC version in figure 5-2

Figure 5-2 ECC version of the new algorithm

As can be seen from figure 5-2, the ECC version is different in the sense that the private-public keys are now (a, Pa) instead of (a, g^a) for Alice and similarly (b, Pb) for Bob. Here P is a known base point. The Diffie-Hellman public key will be Pab instead of g^{ab} . The client and the server will both have to convert it to an AES key. This will be used to encrypt a random value r_A on the client side, while the server will use this key to decrypt the encrypted r_A sent by the client. As we have already discussed in chapters 1 and 4, symmetric key encryption is always much faster than public key encryption for the same level of security. This is the reason why we do a 128 bit AES encryption of r_A even if we have to incur the extra cost of calculating the AES key from the DH key Pab . As we will see in the evaluation section, this cost is negligible with the cost of encrypting data using public key cryptography. Note that this step of calculating the AES key can be pre-computed and stored in both the client and the server.

After this the Server will once again use the same method to encrypt its random value r_B , using an AES key derived from r_A and send it to the client. The client will decrypt it in a similar fashion. Now both can get the session key which will be $(r_A \oplus r_B)$. This step cannot be pre-computed and will have to be done by the client at runtime.

5.3 Implementation of the Client Server MAKEP

We have modified the original Client Server MAKEP [30] described in chapter 4 to a new ECC version as shown in figure 5-3. As a first step the client random r_A is encrypted using the server's public key. The protocol doesn't explicitly specify the encryption technique to be used so we have used the ECC version of the El Gamal encryption protocol as proposed in [3]. We will take a look at this encryption technique as we have not discussed it in the previous chapters.

5.3.1 The El Gamal Encryption protocol

Let b , and Pb be the private and the public keys of *Bob* where P is a publicly known base point and $b \in \mathbb{F}_2^n$. Let Pm be the message point to be encrypted. *Alice* will compute a random bit r and use this to further compute 2 points

$$P_r = Pr$$

$$P_h = Pm + r(Pb)$$

Figure 5-3 ECC version of the Client-Server MAKEP

Note that $r(Pb)$ is another point on the elliptic curve as the multiplication of an integer and a point on the curve is another point on the curve. These points P_r and P_h are now sent to Bob. Bob will compute

$$P_x = (b) P_r = rbP \quad \text{using his secret key.}$$

The original message P_m can now be obtained by

$$P_h - P_x = P_h - rbP = P_m + rbP - rbP = P_m$$

Note that we have substituted this step of encryption using public key to encryption using AES based on a key derived from DH key exchange protocol.

The way we encrypt r_A is by embedding it on to a point on the elliptic curve by putting it as the x coordinate of a point and calculating the corresponding y value. This embedding has been described in some detail in chapter 3. On decrypting the data, the server will take the x coordinate as the decrypted r_A value.

The next step is encryption of the server random r_B by the decrypted value of r_A . We assume that this step is done by using r_A as a symmetric key. We use the same AES algorithm as we have used in our protocol to do the encryption and decryption in this case. Accordingly we convert the decrypted r_A to an AES key and use this to encrypt r_B . This value is sent to the client where it is decrypted using the same AES key derived from r_A .

The next step is a bit different from the original algorithm. The algorithm says to bind the step 1 and 3 together with a random value from the client i.e. b and the client's private key a . This is done in step 1 by sending $\beta = g^b$ to the server and in step 3 by sending $y = ah(\sigma) + b$ to the server. The server verifies this with $g^y = (g^a)^{h(\sigma)} * \beta$. Thus there is a binding between step 1 and 3 and this prevents an interleaving attack of the type observed in [41]. An exact similar step however is hard to do in our ECC version of this algorithm.

A parallel in ECC would be to send $\beta = Pb$ (equivalent to g^b) to the server in step 1. In step 3 the client would compute $y = ah(\sigma) + b$ and send this to the server. The server would compute Py and see if this was equal to $(Pa)h(\sigma) + \beta$. Here (Pa) is the client's public key. But this will not work in our implementation as a and b are all integers and Py , Pa , Pb are all points on the curve defined over F_2^n . The rules of addition and multiplication between the integers and the points on the curve are very different and cannot be compared. Thus even if $y = ah(\sigma) + b$; Py will not be equal to $(Pa)h(\sigma) + Pb$. So we cannot use this as an authentication/verification step.

The purpose of this step is to bind the secret key a with a random secret b ; so that there is a binding relation between the steps 1 and 3; which will be true for that particular session. Furthermore this binding can be easily calculated by the client without much computational power. Keeping these two aims in mind we propose a different ECC approach to achieve the same objective:

- In step 1 send $P(a + b)$ to the server; where a is the secret key and b is a client generated random number
- In step 3 send $y = (a + b + \sigma)$ to the server. The server will calculate $(y - \sigma)$ and multiply it with the base point P to get $P(y - \sigma)$ and compare it with the value received in step 1. The values should be equal.

This will be an approximately equivalent step. Note that in the original algorithm the client is required to compute the exponential g^b . In our version case the client is required to do the point multiplication $P^*(a + b)$. Point multiplication being equivalent to exponentiation in DLP as the most computationally expensive process this step is equivalent to the first step in the original algorithm. In the original algorithm, step 3 the client performs an inexpensive operation of $y = ah(\sigma) + b$; in our version this inexpensive operation is $y = (a + b + \sigma)$ so this is equivalent. On the server side in the original case we have one exponentiation (shown in **bold**) while calculating the formula $(g^a)^{h(\sigma)}\beta$. Similarly in our version the server has one elliptic point multiplication while calculating the formula ' $P^*(y - \sigma)$ '. Thus we have seen that our version of the Client Server protocol provides the same basic feature of authentication and verification and maintains the computational complexity of the original algorithm.

5.4 Comparison and Discussion of results

We will now show a few results and timing analysis for these two protocols. Our implementation of the Client-Server protocol may not be the most efficient implementation possible but we have used the same mathematical implementation for comparing both the protocols. Also to be noted is that we are concentrating only on the client as it is the bottle neck for speed and efficiency. Hence in the Client-Server protocol we have ignored the verification time needed by the server to check the 'y' value sent by the client. Even so our protocol performs three times as fast compared to the Client-Server MAKEP.

Table 5.1 Comparison for the total time taken

Field Size (Bits)	Our Protocol Total Time (Secs.)	Client-Server MAKEP Total Time (Secs)	Speed comparison
158	4.81	15.30	3.18 times
155	4.53	12.44	2.75 times
134	3.45	9.30	2.70 times
119	2.88	7.49	2.60 times
113	2.79	7.03	2.51 times
90	1.41	4.02	2.85 times
65	1.08	2.06	1.90 times
50	1.04	1.63	1.56 times

As we can see from the graph in figure 5-4, although the time gap between the algorithms closes substantially for lower field values, our algorithm is still one and half times as fast for lower values, and more than three times as fast for higher field bits.

Figure 5-4 Graph of total time comparison

We will now analyze the two protocols by breaking them up into smaller sections to find out the computationally expensive portions. The first subsection we will consider is the pre-

computation portion. This portion of the protocol does not need to be executed at runtime. The pre-computational time is given in the table 5.2.

Table 5.2: Runtime comparison

Field Size (Bits)	Our Protocol		Client-Server MAKEP	
	Pre-computational Time (Secs.)	Run Time (Secs)	Pre-computational Time (Secs.)	Run Time (Secs)
158	2.976	1.84	2.970	12.400
155	2.745	1.79	2.740	9.700
134	1.844	1.61	1.830	7.560
119	1.26	1.54	1.309	6.200
113	0.964	1.4	0.988	6.004
90	0.51	1.2	0.531	3.670
65	0.19	0.98	0.211	1.840
50	0.1	0.94	0.101	1.500

The actual time from table 5.1 minus the pre-computational time from table 5.2 gives us the run time of the protocol. Figure 5.5 shows these values plotted in a graph. If we compare the graphs 5.5 and 5.4 we can see that our protocol is still nearly seven times faster at the higher bit size fields and 1.5 times faster for the lower bit sizes.

Figure 5-5: Actual Run time graph

We will now look at a major improvement we have made in our protocol in the first step. This is converting the DH key into an AES key and carrying out symmetric key encryption of the client random value. As opposed to this, we have used El Gamal public key encryption to encrypt the client random value in the client-server MAKEP protocol. We can see the time difference in table 5.3.

Table 5.3 Step 1 Encryption Comparison

Field Size (Bits)	Our Protocol [AES Encrypt r_A (<u>Milli Secs.</u>)]	Client-Server MAKEP [El Gamal Encrypt r_A (<u>Milli Secs.</u>)]
158	13	5600
155	12	5090
134	12	3500
119	12	2700
113	13	2030
90	12	1040
65	12	400
50	12	200

Figure 5-6 Step 1 Encryption Comparison

As we can see from the above figure, this is a very big improvement in terms of speed.

These readings were taken without using the pre-computation method of point multiplication discussed in section 3.9 of chapter 3. We will now see the timing analysis of our protocol using the pre-computation method of the Digit-Slice technique. We have only compared our protocol with and with out pre-computations in this analysis. The relative efficiency of our protocol over the Client-Server MAKEP will remain constant no matter what the implementation as shown in the previous sections. Hence we are not considering the Client-Server MAKEP in this analysis. We will look at higher bit values than the ones we have examined previously, because that is where the efficiency of pre-computations really is apparent. Our results are shown in Table 5.4 and the corresponding graph is in figure 5-7.

Table 5.4 Time analysis of pre-computation in the protocol

Field Size (Bits)	Without Pre-computations (Seconds)	With Pre-computations (Seconds)
270	16.495	5.302
230	10.552	3.965
210	7.795	2.47
173	4.828	2.202
158	4.448	2.075
155	3.334	1.921

Figure 5-7 Graph of speed of the protocol using pre-computations

The above observations have been recorded for a window size of 10 bits. As we can see from the figure, the efficiency increases to quite a large extent for higher field size values of around 270 bits. At this point, the protocol runs thrice as fast. For lower field size values

(around 155 and less) the protocol runs a less than twice as fast, but for the field sizes in between it increases quite to more than twice the speed.

We will now analyse the speed of our protocol for different window sizes, for a constant field size of 173 bits. The results are shown in table 5.5 and figure 5-10.

Table 5.5 Time analysis on number of windows in the pre-computation

Protocol with Pre-computations			Protocol Without Pre-computations
Window size	No. of windows	Time (Sec)	Time (Sec)
44	4	2.7262	5.2
35	5	2.6777	5.2
28	6	2.3635	5.2
22	8	2.2200	5.2
14	13	2.1615	5.2
10	18	2.0700	5.2
5	43	1.8617	5.2

Figure 5-8 Protocol speed at different window sizes for 173 bits

As we can see from the figure 5.8 and the table 5.5, we have an improvement of more than twice the speed at 173 bits using the windows method. The protocol runs faster for a larger number of windows but the difference in speed is not substantial over the 10 to 30 windows range. For very low number of windows, i.e. less than 5, the protocol is seen to run

appreciably slower than the higher window values. So in this case 20 to 30 windows will prove to be quite good.

Note: The readings in this section may prove to be a bit different than the readings in the previous sections. The reason for this is that we are using Wireless LAN as the wireless media in this section, while we used iRDA in the previous sections. However the relative comparisons between the protocols and between different window sizes are not changed.

We will now summarize the key points of our analysis

- The idea of converting a key generated by the Diffie-Hellman key exchange protocol into a symmetric key vastly improved the performance of our algorithm. The symmetric key AES encryption step proved to be more than 450 times faster than the public key El Gamal encryption step for higher fields.
- The pre-computation was also much faster (almost twice) as we did not have to deal with calculating a β value, which involved a elliptic point multiplication process.
- The algorithm executes in two steps of communication between the client and the server, as opposed to the 3 steps required for the Client Server MAKEP.
- Over all, the entire algorithm was almost five times as fast as compared to the Client-Server MAKEP protocol.
- The protocol can be made to run twice to thrice as fast (depending on the field size and the window size) using the pre-computations technique mentioned in chapter 3.
- We get good performance for window sizes in the range of 20 to 30. Higher than this the increase in efficiency is not too high, while lower than this the decrease in efficiency is appreciable.
- There maybe some weaknesses in the original protocol as we have discussed in the previous section, but these are not serious and do not affect the security of the communication and the security of the secret keys. We have discussed in the earlier sections how we can easily remove the weaknesses by adjusting our protocol and this will not lead to any decrease in speed since the improvements will have negligible computational cost.

5.5 Peer-to-Peer version

5.5.1 Protocol Description

From table 5.2 we observe that for a field size of 158 bits, our algorithm takes 1.84 seconds to execute (not considering the pre-computations). Also we observe from figure 5-2 that the server has to perform 1 elliptic point multiplication (to calculate the DH Key) and the rest of the computations (like converting it into an AES key, decrypting and encrypting using the AES key) are negligible in comparison to this.

If we compare figure 5-2 and 5-3 we observe that in our case, the client and the server have to perform almost the same level of computations, (except the certificate verification by the server). As opposed to this, the server in the Client-Server MAKEP has to perform a much higher load of computations. Since in our protocol the computational load is somewhat similar, we have made a Peer-to-Peer version of our protocol in which both the peers are low powered clients.

Figure 5-9 Peer-to-Peer version of our protocol

In figure 5-9, the parts in ***bold italic*** font mark the changes to the original protocol. Since we are dealing with low powered peers, we assume they don't know each other's public keys so we need to transfer their certificates to exchange their keys and authenticate each other. We then securely exchange their random numbers according to our original protocol to get a temporary fresh session key.

Note that in this protocol we still have the weakness of the server being able to control the final session key, similar to the one we had in section 4.5.2. We can remove this weakness by the same method we used earlier i.e. by hashing the client random r_A , so that the server

does not know it before hand. A more secure version that incorporates this idea is shown in figure 5.10.

Figure 5-10 A more secure Peer-to-Peer version

The additions to the original protocol of figure 5-9 are shown in **bold** font. Instead of sending r_A to Bob, we now send $h(r_A)$ encrypted with AES key derived from the DH pair Pab . Bob encrypts his random r_B with this hash value. He cannot guess the original random r_A from the hash value (assuming it is a strong hashing function e.g. SHA1). Hence his random r_B will not be able to weaken the session key by fixing it to some predetermined value. After Alice gets the Bob's random r_B , she will calculate an AES key y by combining

the AES key from the DH pair (Pab) and the server random r_B and the hashed value $h(r_A)$, and encrypt her random r_A with y . Similar to the logic followed in section 4.5.2 this binds the earlier steps of the protocol to the last step and also authenticates *Alice* to *Bob*.

The value y , which is used as a 128 bit AES key, can be computed as follows:

$$y = \text{First 128 bits of } (\text{SHA1}(Pab.x\text{-coordinate}) \oplus \text{SHA1}(Pab.y\text{-coordinate}) \oplus \text{SHA1}(r_B) \oplus h(r_A)).$$

5.5.2 Protocol Timing Analysis

There will be hardly any difference between the speeds of the protocols in figure 5-11 and 5.12. This is because it will take negligible time to compute y , and encrypt and decrypt using AES.

However there will be a considerable amount of time spent in the certificate verification algorithms. For the client server protocols we did not have to worry about the certificate verification as the verification was performed on the server and for our implementation, the amount of time the server required to do verification was minimal. However for PDA peers the certificate verification timings will have to be considered. Another major chunk of time will be devoted by both the peers in calculating the DH key Pab .

We will now look at the timing analysis of both the computationally intensive portions of the protocol. Table 5.6 shows the time required for the DH key generation part of the peer-to-peer protocol. The pre-computation part uses 5 windows. The efficiency could be made slightly better by using more windows. For 5 windows we get more than twice as fast values.

Table 5.6 Peer-to-Peer version timings for DH key generation

Two Point Multiplications (creating the DH key on both peers)		
Field Size	Protocol without Pre-computations (Secs)	Protocol with Pre-computations (Secs)
270	29.512	11.760
230	19.600	7.702
210	13.120	5.200
189	9.996	4.062
173	7.466	3.102
155	5.400	2.230
134	3.520	1.412
119	2.628	1.066
98	1.492	0.622
81	0.696	0.260
65	0.390	0.144

Table 5.7 shows the total time required for the ECDSA certificate verification portion of the protocol. Certificate verification occurs twice, once of *Alice* and then for *Bob*.

Table 5.7 Peer-to-Peer version timings for ECDSA Certificate Verification

ECDSA Certificate Verification		
Field Size	Without Pre-computations	With Pre-computations
113	3.800	1.238
131	6.258	2.008
134	6.938	2.214
158	11.558	3.592

We have noted the pre-computation readings for a window size of 10. This gives us an improvement of a factor of three. Since the computations represented in tables 5.5 and 5.6 are the most time consuming parts of the protocol and the other computations like hashing and symmetric encryption/decryption take negligible time in comparison, we can get a fairly accurate timing analysis of the peer-to-peer protocol by adding the results of the tables. We chose 158 bit ECDSA certificates, as the field size needs to be around 160 bits for effective security (see chapter 3).

Table 5.8 tabulates the approximate time required by the peer-to-peer protocol for different field sizes, obtained by adding the results of table 5.5 to the ECDSA verification algorithm timings for 158 bit size certificates.

Table 5.8 Peer-to-Peer protocol timing analysis

ECDSA Certificate Verification		
Field Size	Without Pre-computations	With Pre-computations
270	41.070	15.352
230	31.158	11.294
210	24.678	8.792
189	21.554	7.654
173	19.024	6.694
155	16.958	5.822
134	15.078	5.004
119	14.186	4.658
98	13.050	4.214
81	12.254	3.852
65	11.948	3.736

We can see the trend from figure 5-11

Figure 5-11 Performance of the peer-to-peer protocol

For a 270 bit field size, the protocol takes around 15 seconds to establish a session key between two low power peers. We could bring this down to around 12 seconds by having a larger number of windows (say 20), going by our results in section 3.8.

5.6 Other related work

Our basic aim of the project was to create a ECC based implementation and protocol that could authenticate wireless devices and establish a session key quickly and securely. The session key would then be used by other applications or services on the mobile devices to encrypt/decrypt data passing to and fro from the server using a symmetric encryption/decryption algorithm. We have shown an efficient implementation of ECC for mobile devices like PDAs and have demonstrated a fast authentication and key establishment protocol for PDAs clients communicating with relatively more powerful servers like desktop PCs. We have also designed our ECC protocol implementation so that other applications could easily use it as a black box to establish a secure session key. The application needs to only provide implementation to some data transfer functions like creating appropriate sockets and sending and receiving data over them. We felt this was necessary since our protocol is intended to be used over any wireless media be it TCP

sockets over WLAN or GPRS or iRDA sockets or Bluetooth based data transfer. Hence we leave the implementation of exactly how the data is to be transferred to the application. We have already tested the protocol over WLAN and iRDA as seen in chapter 5 discussions.

We will now show describe how we integrated the ECC module into a practical application for PDAs. The application is a remote file browser and downloader which use our ECC and AES implementations to implement strong and fast security at the application layer. Let us first look at the motivation for our application.

5.6.1 Motivation

As we have mentioned in chapter 1, mobile devices are increasingly become a part of peoples everyday lives. We can see many people using their PDAs and laptops while traveling or in public places. With increase in the memory and the computational power of PDAs, these devices are becoming increasingly popular as they are not as bulky as laptops and not as limited as mobile phones.

However, PDAs will always lag behind in conventional desktop PCs or laptops as far as storage and processing is concerned. Thus there is a tradeoff between the ease of carrying a PDA while on the move, and the amount of information that can be carried on it.

One way to solve this problem would be to use some kind of dynamic synchronization i.e. dynamically get the relevant data as and when needed from a data repository that has been hosted on a server (which may be running on desktop PC). The PDA could connect to the server as and when needed and download data temporarily for the user to work with. This gives us two interesting scenarios

1. A user may want to retrieve his files and documents from his personal computer while on the move. In this case he may use his PDA to dynamically download data from a server running on his PC. His PDA will have to be equipped with a WLAN card or a GPRS card to access the internet.

2. Commercial businesses may offer services to users on the move by which they could dynamically download data on their PDAs as and when required. A few of such services are already in the market.

Such an application has to pay special heed to the aspect of security. There is an additional risk while doing data transfer on a wireless network as the wireless media is inherently less secure than wired networks. Security hazards are more in such scenarios, since anyone can intercept wireless transmissions if they have the appropriate equipment. The 'appropriate equipment' in many cases may be as simple as having a PCMCIA or a Compact Flash card for wireless LAN.

One way of solving this problem would be using internet security protocols like SSL or IPSec to encrypt the data, since we are using the internet to transmit the data in the first place. But since these were designed for the fixed network in mind, with clients as standard PCs or laptops, using these on low power devices like PDAs in wireless networks may cause more delays as the overheads will be quite large. Another way would be through the use of Virtual Private Networks, (VPNs) which support secure data transfer using encryption. But for this, the organization needs to make its network VPN enabled, which entails additional costs, especially for small home networks, where all the user desires is to browse his home computer using his PDA. Also practical experience of using VPNs of the NTU network from outside the campus have made us realize that VPNs can be quite slow, because of the security and network mapping overheads. We are dealing here with a small limited device like a PDA, so the delay on using VPNs will be much larger.

The other option is to use a dedicated encryption engine to perform fast encryption of data on the PDA before sending it to the client. We also need to perform other cryptographic operations like authentication and integrity checks to make sure the data is valid. We will describe this option which does encryption using AES and authentication and key-exchange using our ECC based protocol.

The interesting point to note about this application is that it is designed specifically for small low power devices like PDAs, so that there are minimum delays due to the security

overheads. Thus our application will enable a PDA user to retrieve information **quickly** and **securely** from his/her desktop PC.

5.6.2 The Secure Remote File Browsing Application

We will first give a broad overview of the application. The various modules that make the application are shown in Figure 5-12.

Figure 5-12: Application Modules

As can be seen from the figure, the file browsing application sits on top of the two security modules (the AES engine and the ECC key exchange module). It can send messages to the Encryption module of the AES engine and it can receive messages from the Decryption module. There is password based authentication module sitting on top of the file browsing application which requires a correct password before allowing access to the main application.

The AES engine has three sub modules: Initialisation, Encryption and Decryption. The Decryption module can receive encrypted messages from the wireless interface. The

Encryption module encrypts the messages it receives from the File Browsing application and sends them to the wireless interface. The initialization module carries some initialization routines and requires key material from the ECC key exchange module to create a 128 bit AES key.

The ECC key exchange module securely authenticates the server and the PDA to each other and creates a common 160 bit session key which is supplied to the Initialisation module of the AES engine.

Let us now look at the message flow of the system:

Figure 5-13 Message flow of the system

As shown in Figure 5-13 the following actions take place on starting the application:

Step 1: A temporary session key is established using a key exchange protocol based on ECC.

Step 2: The session key, along with the user's password is converted to an AES key. The server returns a list of file system resources as a text string encrypted with this key.

Step 3: The PDA decrypts the string and presents the information to the user. If the user wants to browse the file system, relevant requests are sent to the server as encrypted messages and Step 2 is repeated.

Step 4: If the user wants to download a particular file from the server's file system, a relevant encrypted request is sent to the server.

Step 5: The server encrypts the file with the AES key and sends it to the PDA.

Step 6: The PDA receives the encrypted file and decrypts it using the AES key.

So before we can browse resources from the server we need to establish a AES key.

Secure Key Exchange

The process of creating an AES key is shown in Figure 5-14. We create an AES key from two sources. One source is the key establishment protocol. The second is the password supplied by the user to log on to the application. We have a password based authentication scheme by which a user has to enter a valid user name and password to access the application. The server on the user's home PC also demands the same password in order to start. We store a list of valid user names and passwords in a file on the PDA and we use this file to check if the password is valid. Of course it would be pretty ridiculous to store passwords in clear text on the PDA. We use the SHA algorithm for calculating a unique 160 bit hash value of the password. Actually we add a small secret alpha-numeric string to the password before hashing it to prevent dictionary attacks (this technique is popularly known as 'salting'). The same process is carried out at the server side before it can be run.

Here is the format in which we store the username and password:

```
~testuser*75 17 61 CF C4 25 09 0D F3 79 53 B6 6E 09 5E 07 4C 49 3B 82|
```

'~' marks the start of the user name, '*' marks the end of the user name and start of the 160 bit hashed password (which we store as a hex value of 20 bytes each separated by a single space), and '|' represents the end of the password.

We thus give user level security in the application. This is necessary because there may be a chance that more than one user has access to the PDA in which case our application should not give them all the right to browse the resources on the PDA owners computer. In fact by providing a password based authentication, we also provide for a multi-user scenario where many people could use the same PDA to browse their personal computers securely.

The AES key generation and the password based authentication are shown in Figure 5-16.

Figure 5-14 AES Key Generation

Once we have got our key for the AES encryption we can proceed to the actual encryption and decryption of data.

We will now look at the application which uses the security modules

File Browsing Application

Browsing the file system of the server:

The file browsing module sits on top of the AES encryption module. We use a simple 'tree-control' to navigate the directory structure. The paths, folders and files are extracted from the decrypted data received by the PDA from the server. A typical decrypted message will look as follows:

```
{Drive1 (Folder1) (Folder2) | [File1][File2]}
```

This means Drive1 contains Folder1, Folder2 and File1 and File2. This will be populated in a tree control on the PDA.

Initially when the PDA first connects to the server, the server sends it a list of drives on the computer on which the server is running. The PDA displays this information to the user on a drop down menu. The user selects a drive which he/she wants to browse. This request is converted to the following message by the PDA and sent to the server:

DIRCHANGED DriveName

The server responds with a list of folders in that drive, which are displayed in a tree control. The PDA then asks for files on that drive with the following message

GETFILES Path

Where 'Path' is the DriveName in this case. The server returns with a list of files which are present in that drive. These files will be displayed in a list control. This message will also be sent when the user selects a folder in the tree, but the 'Path' in the message will be the location of the folder in the selected drive.

Similarly, if the user tries to expand a folder node in the tree, and the PDA does not currently hold the list of folders within this folder the PDA will request it from the server:

GETFOLDERS Path

Thus we have a 'on-demand' browsing scheme, where we don't upload the entire file system structure, but only parts of it as and when needed, thus saving on memory and increasing speed. A snapshot of file browsing is shown in figure 5-15.

Figure 5-15: User application on the PDA

Note that all the messages will be encrypted and decrypted with AES, before being sent or received over the network. There may be a maximum delay for each request of a few seconds due to the security and the wireless networking overheads.

Sending and receiving the files

The PDA can select a particular file and request it from the server using the SENDFILE message which contains the file path. The format is of the type:

SENDERFILE Drive1:\Folder1\File1

The server will now encrypt the file and send it to the PDA. As we have already discussed it will encrypt the entire file using AES, calculate the SHA hash of the file and add it to the

start of the encrypted file. Now the file is sent to the wireless interface module (in our case a module which sends data through a WLAN compact flash card), which sends it to the PDA. The PDA will extract the SHA hash from the file, calculate the SHA hash from the remaining part of the file and check if the two hashes match, which implies there is no loss of integrity. After this the PDA will decrypt the file using AES.

Thus users can use our protocols to navigate the file systems on their home computers and download a file they need, securely.

5.7 Summary

In this chapter we have introduced a new authentication and key establishment protocol for low power devices. We have compared our key exchange protocol with the ECC version of the client-server protocol. We have shown that our algorithm outmatches its competitor quite comprehensively. We have also proposed a peer-to-peer key exchange protocol based on our original protocol and analysed it. Finally we have seen how a practical application for a PDA would use our protocols in the initial authentication and key establishment phase.

6 Conclusion and Future Work

The project has looked into two important issues. The first is the efficient implementation of elliptic curve cryptography on constrained handheld devices and the second is efficient key exchange algorithms for low powered clients.

As of now we have looked into two types of ECC implementations – polynomial basis and optimal normal basis. We have discussed the security issues and the possible attacks on the cryptosystem. We have discussed the implementation issues and seen the comparisons between both the implementations for all the major operations of elliptic curve cryptography like field multiplication and inversion; and point multiplication and data embedding. From the results of our implementation on a iPAQ PDA, we conclude that our ONB implementation is faster than the polynomial implementation in all the cases. We also found that the point multiplication was the slowest operation in ECC and an important part of any security protocol. We implemented a windows based method of pre-computation for the point multiplication in ECC for ONB based implementation and increased the time efficiency by a factor of three.

The second part of this project dealt with mutual authentication and key exchange algorithms. We have looked into the major key exchange protocols and their advantages and weaknesses. We also discussed protocols specifically designed for low powered clients. We have seen their advantages and analysed their weaknesses. We have used our observations to propose some variations that seems to be more efficient than the other protocols that specifically deal with this issue of low power clients and powerful server scenarios. We have used the Client Server MAKEP protocol to compare with our protocol. We have created an ECC version of this protocol so as to compare it directly with our protocol. We have shown

that our protocol is more than three times faster than the Client-Server MAKEP protocol. We have also analyzed our protocol and suggested improvements to remove these without affecting the speed efficiency.

Our protocol is unique in the sense that it can be made symmetric by a few modifications. Hence we found we could easily transform it into a peer-to-peer protocol. We have presented a peer to peer version of our protocol and shown how it can be made faster using our pre-computation techniques. We have also pointed out some security weaknesses and suggested modifications to remove them.

Finally we have used our ECC implementation and the key establishment protocol in a PDA based remote file browsing application to show how our protocols and security implementations can be easily integrated into useful real world applications that require an underlying security layer to protect their communication data.

Future Work

This thesis has looked in to the problem of providing efficient implementations and protocols for low power mobile devices communicating over wireless networks. While our work has focused mostly on the basic implementation aspects of ECC on PDAs we have not focused on the ECC standards like PKCS #13 or the IEEE P1363 while designing our implementation. This is because the goal of this master's thesis was to investigate efficient implementation methods and efficient security protocols specifically designed for low power devices.

A possible extension of our work would be to use our ECC subroutines in the implementation of a recognized industry standard such as PKCS which could be used in commercial applications.

We have added some pre-computational methods for speeding up the slowest routines in our implementation. Our implementation could be made even faster by incorporating other techniques and algorithms for routines like inversion, which also take up considerable time.

Our key establishment and authentication protocol solves many problems related to mobile devices and we derive its security from an earlier protocol (the Client-Server MAKEP) that has been proven to be secure using the Bellare-Rogaway model. However we have not used any model to test the security of our protocol. It would be interesting to see if our protocol can actually handle all types of attacks using a simulation model or tool like a NRL protocol analyzer.

References

1. Menezes, A. (1993). '*Elliptic Curve Public Key CryptoSystems*'. ISBN 0-7923-9368-6. Boston : Kluwer Academic Publishers
2. '*Eric Weisstein's World of mathematics*', available online at <http://mathworld.wolfram.com/>, retrieved December 2002.
3. Rosing, M. (1999) '*Implementing Elliptic Curve Cryptography*'. Greenwich, CT: Manning.
4. Certicom's tutorial on ECC available online at http://www.certicom.com/resources/ecc_tutorial/, retrieved September 2002.
5. Blake I, Serroussi G & Smart N. '*Elliptic Curves in Cryptography*'. Volume 265 of London Mathematical Society lecture notes. Cambridge university press.
6. Pietilainen H. '*Elliptic Curve Cryptography on Smart cards*'. Masters thesis from the Faculty of Information technology, University of Helsinki.
7. Woodbury A, '*Efficient Algorithms for Elliptic Curve Cryptosystems on Embedded Systems*'. Masters thesis, Worcester Polytechnique Institute, September 2000.
8. Montgomery P. '*Modular Multiplication without Trial Division*' Mathematics of Computation, 44(170):519-521, April 1985.
9. Orlando G. '*Efficient Elliptic Curve Processor Architectures for Field Programmable Logic*', PhD Dissertation, Worcester Polytechnique Institute, March 2002.
10. Lopez J., Dahab R. '*An Overview of Elliptic Curve Cryptography*', Technical Report, Available online at <http://www.ic.unicamp.br/ic-tr-ftp/2000/Titles.html>, retrieved December 2002.
11. '*The Elliptic Curve Cryptosystem for smart cards*'; white paper from Certicom Corp. available online at http://www.certicom.com/resources/w_papers/w_papers.html, retrieved December 2002.
12. Van Oorschot P., Wiener M. '*Parallel Collision search with cryptanalytic applications*', Proceedings of the 2nd ACM Conference of Computer and Communications Security, 1994.

13. Wiener M., Zuccherato R., '*Faster attacks on elliptic curve cryptosystems*' Proceedings of Selected Areas in Cryptography 1998, Pg 190-200
14. Miller V. '*Uses of elliptic curves in cryptography*', Advances in cryptology – CRYPTO '85, Lecture Notes in Computer Science, 1990.
15. Menezes A., Okomoto T. & Vanstone S., '*Reducing Elliptic curve logarithms to logarithms in finite fields*', IEEE Transactions on Information theory, 1993.
16. Torii N. & Yokoyama K., '*Elliptic Curve Cryptography*', UDC 681.8, June 2000.
17. '*Elliptic Curve Cryptosystem: Remarks on Security of ECC*'; white paper from Certicom Corp. available online at http://www.certicom.com/resources/w_papers/w_papers.html, retrieved September 2002.
18. Christensen, G. (2001) '*Wireless Infrastructure Technologies*', Faulkner Information Services.
19. Anderson, R. (2002) '*Security Engineering*', Wiley.
20. Randall N. & Lekkass P. (2002) '*Wireless Security: Models, Threats and Solutions*' McGraw-Hill.
21. Schneier B., '*Applied Cryptography*', Wiley.
22. Schneier B., article on ECC in the Crypto-gram newsletter, November 1999. Available online at <http://www.counterpane.com/crypto-gram-9911.html>
23. Menezes A. '*Comparing The Security Of ECC and RSA*', articles available online at <http://www.cacr.math.uwaterloo.ca/~ajmenczc/misc/cryptogram-article.html>, retrieved January 2003.
24. Lenstra A., Eric R. & Verheul. (1999) '*Selecting Cryptographic Key Sizes*', Journal of Cryptology: the journal of the International Association for Cryptologic Research.
25. Press release from SSP Solutions; available online at: http://www.sspsolutions.com/news/press_release.php?Article=28, retrieved December 2002.
26. Kaliski B., RSA Laboratories, '*Cryptography Trends: A US Based Perspective*', IPA/TAO Cryptography Symposium 2000.
27. Jhonson D. '*ECC, Future Resiliency and High Security Systems*', 1999, white paper from Certicom Corp. available online at http://www.certicom.com/resources/w_papers/w_papers.html, retrieved November 2002.

28. Agrawal M., Kayal M. & Saxena N., '*PRIMES in P*', Department of Computer Science and Engineering, IIT Kanpur, available online at <http://www.cse.iitk.ac.in/primality.pdf>, retrieved December 2002.
29. http://www.bouncycastle.org/latest_releases.html, retrieved December 2002.
30. Wong D & Chan A. (2001). '*Efficient and Mutually Authenticated Key Exchange for Low Power Computing Devices*'. ASIACRYPT 2001.
31. Wong D & Chan A. (2001). '*Mutual Authentication and Key exchange for low power wireless communications*', IEEE MILCOM 2001 Conference Proceedings.
32. Diffie W. & Hellman M. (1976). '*New Directions in Cryptography*', IEEE Transactions on Information Theory 644-654.
33. Diffie W., von Oorschot P.C. & Wiener M. (1992). '*Authentication and authenticated key exchanges*', Proceedings of Designs, Codes and Cryptography.
34. Analysis of Euclid's algorithm
<http://www.msri.org/publications/In/msri/2000/introant/shallit/1/banner/22.html>,
retrieved January 2003.
35. Berlekamp E. (1986). '*Algebraic Coding Theory*', McGraw Hill.
36. Solinas J. '*An Improved Algorithm for Arithmetic on a Family of Elliptic Curves*', CRYPTO '97.
37. Tan Peng Chor, '*Efficient Implementation of ECC*' Masters Thesis NTU 1999.
38. AES implementation on www.codeproject.com; Available online at <http://www.codeproject.com/cpp/aes.asp?target=aes> (retrieved June 2003).
39. Zhi li, Higgins J. & Clement M. '*Performance of Finite Field Arithmetic in an Elliptic Curve Cryptosystem*' Proceedings of 9th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2001). Pg 249 – 258.
40. Ciet M., Lange T., Sica F. & Quisquater J. '*Improved Algorithms for Efficient Arithmetic on Elliptic Curves using Fast Endomorphisms*'. Lecture Notes in Computer Science Volume 2656 pp 388 ff.
41. Jakobsson M. & Pointcheval D. (2001) '*Mutual Authentication for low-power mobile devices*'. Proceedings of Financial Cryptography 2001, Soringer-Verlag.
42. Dabholkar A. & Yow K.C. (2003) '*Efficient authentication and key exchange protocol for low powered clients*' To be published in 5th IEEE Mobile and Wireless Communications Networks Conference, Singapore, October 2003.

43. P1363 standards. Available online at <http://grouper.ieee.org/groups/1363/> (retrieved October 2003).
44. Jhonson D., Menezes A. & Vanstone S. '*The Elliptic Curve Digital Signature Algorithm*' 1999, white paper from Certicom Corp. available online at http://www.certicom.com/resources/w_papers/w_papers.html, retrieved October 2003.
45. Message digest implementations including SHA 1 from codeproject.com. Available online at <http://www.codeproject.com/cpp/filedigest.asp> (Retrieved October 2003).
46. Overview of the AES developmental effort. Available online at <http://csrc.nist.gov/CryptoToolkit/aes/index2.html> , retrieved October 2003).
47. Rijndael implementation code. Available online at <http://csrc.nist.gov/CryptoToolkit/aes/rijndael/> (Retrieved October 2003).
48. Daemen J. & Rijmen V. '*The Design of Rijndael: AES - The Advanced Encryption Standard*' ISBN 3-540-42580-2
49. S.-L. Ng and C. J. Mitchell, '*Comments on mutual authentication and key exchange protocols for low power wireless communications*', *IEEE Communications Letters*, to appear. Available online at <http://www.isg.rhul.ac.uk/~cjm/comaak.pdf>, (Retrieved October 2003)
50. Bernstein D. Circuits for integer factorization: a proposal. Available online at <http://cr.yp.to/papers.html#nfsccircuit>, (Retrieved October 2003)
51. Daemen J. & Rijmen V. '*AES Proposal: Rijndael*'. Available online at <http://csrc.nist.gov/encryption/aes/rijndael/Rijndael.pdf>, (Retrieved September 2004)
52. Frankel S., Glenn R. et al. '*RFC 3602: The AES-CBC Cipher Algorithm and Its Use with IPsec*' Available online at <http://rfc3602.x42.com/>. (Retrieved September 2004)