UNIVERSITY OF VAASA
FACULTY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Johan Dams

PORTABLE ELLIPTIC CURVE CRYPTOGRAPHY FOR MEDIUM-SIZED EMBEDDED SYSTEMS

VAASA 2008

Table of Contents

I Introduction **1**

1 Overview **1**

2 Research Outline **2**

II Elliptic Curve Cryptography **3**

3 Operations on Elliptic Curves **5**

3.1 Point Multiplication 6

3.2 Point Addition 6

3.3 Point Doubling 8

4 Finite Fields **10**

4.1 Operations 11

4.2 Domain Parameters 14

4.3 Choosing the Field 15

III Practical Implementation **16**

5 Requirements and Design **16**

5.1 Asymmetric Module 17

5.2 Symmetric Module 18

5.3 Hash Module 19

6 Algorithms **19**

6.1 Karatsuba Multiplication 19

6.2 Itoh-Tsujii Inversion 20

6.3 de Rooij Point Multiplication 21

IV Conclusions and Future Work **23**

7 Results **23**

8 Future Work **24**

Part I

Introduction

1 Overview

With the emergence of ubiquitous computing, pervasive computing, and all the other interesting areas which promise network and computer resource access everywhere, interesting developments can be witnessed. Small devices that fit in the palm of your hand are able to do complex operations for which super computers were needed a only a couple of decades ago. The healthcare industry is looking into ways of constantly monitoring patients while sending this data across the network for a doctor to monitor - hence allowing patients to recover after surgery at home instead of at the hospital. Using sensor networks to monitor facilities at companies allows for a constant supervision in order to reduce maintenance cost and predict faults well in advance. With these developments however, securing communications across the network becomes ever more important. However, due to the mobile aspect of many devices and also the computing resources available in for instance sensor nodes in a sensor network, this can be quite a challenge. Embedded chips don't have the same computing resources available then say, a desktop computer. Battery life in mobile devices is again a limiting factor in what can be achieved. Cryptography in general is mathematically very intensive, and in order to allow such resource constrained devices to encrypt the data that is send over the network, special care has to be taken as to what form of cryptography one can use, and to optimise the algorithms used to limit the power consumption and computing cycles required.

In this paper, a form of public key cryptography called elliptic curve cryptography (ECC) is investigated to determine feasibility for the kind of systems described above. ECC promises the ability to generate cryptographic keys which provide equivalent strength compared to other systems, but do so with shorter key lengths. Extensive research into optimising the algorithms at hand has been done, and a reference implementation has been made.
It should be noted that the focus lies on the implementation of ECC for medium sized embedded systems. This means that low end 8 or 16 bit devices are not considered, but that low power 32 bit devices (typically sub 1 Watt range) are the focus of the work. The reason for this is the anticipation of better battery life for these devices while developments towards low power consumption for these chips is making incredible

headway. Typical applications for these kinds of chips with the need to encrypt data are point-of-sale terminals, HVAC building and control systems, medical instrumentation and monitors, fire/security control and monitoring systems and factory and automation systems.

2 Research Outline

In general, encryption methods are very intensive on memory and computing power. The reason for this is are the mathematical operations necessary on really big numbers. While these operations don't pose any major issues on the latest consumer personal computer, embedded systems often only have a fraction of this raw power available. The goal of this paper is to come up with a way for these devices to communicate with other such devices in a secure way, that is, with all the communication of data between these devices encrypted.

From the beginning, ECC has been the most favoured method to achieve this goal. ECC promises the same level of security for smaller key sizes (and thus smaller numbers), but the trade-off is an increase in mathematical complexity. Through optimisation, this form of cryptography has become feasible for embedded systems as well.

The research in this paper is focused on the development and implementation of an elliptic curve based cryptographic system for embedded devices with which can provide digital signature, key generation and encryption/decryption. The main topics to achieve this goal are the selection and implementation of a suitable elliptic curve, a cryptographic hash function and a strong block cypher.

Part II

Elliptic Curve Cryptography

Since the discovery of RSA (and El-Gamal) their ability to withstand attacks has meant that these two cryptosystems have become widespread in use. They are being used every day both for authentication purposes as well as encryption/decryption. Both systems cover the current security standards - so why invent a new system? Even though ECC is relatively new, invented independently by Miller and Koblitz in 1985, what makes it stand apart from RSA and El-Gamal is its ability to be more efficient than those two. The reason why this is important are the developments in information technology - most importantly hand held, mobile devices, sensor networks, etc. Somehow, there must be an way to secure communications generated by these devices, however, their computing power and memory are not nearly as abundant as on their desktop and laptop counterparts. A current desktop system has no problems working with 1024 bit keys and higher, but these small embedded devices do, as we don't want to spend a lot of their resources and bandwidth securing traffic. What is needed is a crypto system with small keys and small signature size. ECC has those properties due to the fact that there are no known sub-exponential algorithms for the elliptic curve discrete logarithm problem (ECDLP), which means we can use shorter keys for security levels where RSA and El-Gamal would need much bigger keys. As an example, below are two typical keys with the same security level. The first one is an RSA key, the second one is an ECC key.

```
RSA (1024 bit):
B52264FB7B9154350F1BE765F2979A13091E539B40167BC8FE58F5AB5C4DF3C8B0CC
06A68BF6BBBA30D777345A48F81AC60F2397EDE31E6BCCDF78A584D0E913EC10F07C
A55D368B44ADBB3B82E3606310083DF41318872196852E5B20FA1C6AB1B44C943E21
```

```
ECC (192 bit):
FFDF1C7C598311CC1287836B540FB29AF8A35393797D11C8
```

As one can see, a 192 bit ECC key offers the same level of security as a 1024 bit RSA key. Actually, it's even more secure: Table 1 below gives the equivalent key sizes for ECC and RSA, provided by the National Institute of Standards and Technology (NIST) [1].

Still, these keys are longer than equivalent symmetric cryptography keys. Table 2 below, also provided in [1], gives a short comparison.

ECC key size	RSA key size	Key size ratio
(bits)	(bits)	
163	1024	1/6
256	3072	1/12
384	7680	1/20
512	15360	1/30

Table 1: ECC and RSA Equivalent Key Sizes

Bits of Security	Symmetric Algorithm	RSA	ECC
80	2TDEA	$k = 1024$	$f = 160 - 223$
112	3TDEA	$k = 2048$	$f = 224 - 255$
128	AES-128	$k = 3072$	$f = 256 - 383$
192	AES-192	$k = 7680$	$f = 384 - 511$
256	AES-256	$k = 15360$	$f = 512+$

Table 2: Equivalent Key Sizes for Symmetric and Asymmetric Cryptography

It should be clear that in order to provide a security of 80 bits in a public key cryptographic system, one needs a 160 bit ECC key or a 1024 bit RSA key. It should also be apparent that key lengths for RSA grow much faster in length than their ECC equivalent.

The way that the elliptic curve operations are defined is what gives ECC its higher security at smaller key sizes. An elliptic curve is defined in a standard, two dimensional x,y Cartesian coordinate system by an equation according to the Weierstrass model as follows:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

The graph of an elliptic curve can appear for instance as shown in Figure 1.

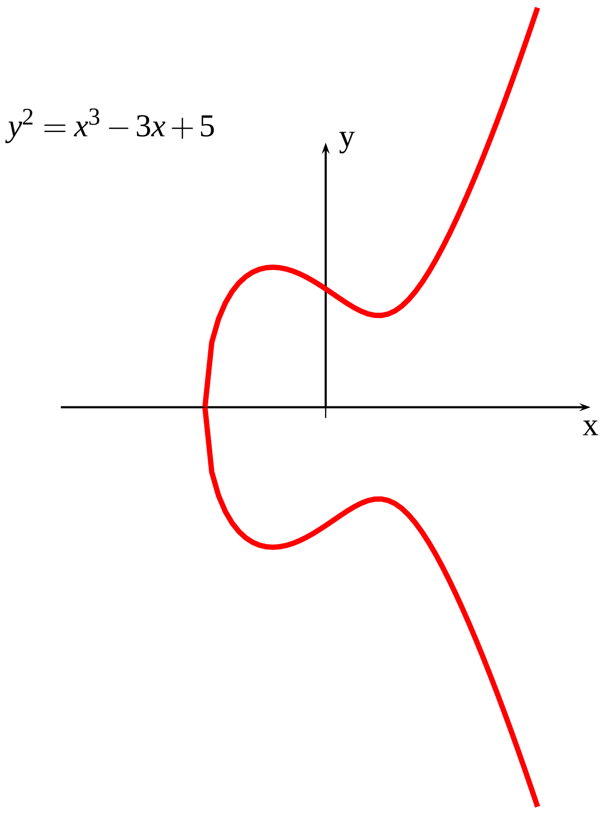

Figure 1: An Elliptic Curve

3 Operations on Elliptic Curves

As said before, the security of ECC depends on the difficulty of the Elliptic Curve Discrete Logarithm Problem. This problem is defined as follows: let P and Q be two points on an elliptic curve such that $kP = Q$, where k is a scalar. Given P and Q, it is computationally infeasible to obtain k, if k is sufficiently large. Hence, k is the discrete logarithm of Q to P. We can see that the he main operation involved in ECC is point multiplication, namely, multiplication of a scalar k with any point P on the curve to obtain another point Q on the curve.

Each curve has a specially designated point G called the base point chosen such that a large fraction of the elliptic curve points are multiples of it. To generate a key pair, one selects a random integer k which serves as the private key, and computes kG which serves as the corresponding public key. For cryptographic application the order of G, that is the smallest non-negative number n such that $nG = O$, with O the point at infinity, must be prime.

3.1 Point Multiplication

In point multiplication a point P on the elliptic curve is multiplied with a scalar k using elliptic curve equation to obtain another point Q on the same elliptic curve, giving $kP = Q$. Point multiplication can be achieved by two basic elliptic curve operations, namely point addition and point doubling. Point addition is defined as adding two points P and Q to obtain another point R written as $R = P + Q$. Point doubling is defined as adding a point P to itself to obtain another point Q so that $Q = 2P$.

Point multiplication is hence achieved as follows: let P be a point on an elliptic curve. Let k be a scalar that is multiplied with the point P to obtain another point Q on the curve so that $Q = kP$. If $k = 23$ then $kP = 23P = 2(2(2(2P) + P) + P) + P$. Thus point multiplication uses point addition and point doubling repeatedly to find the result. The above method is called the 'double and add' method for point multiplication. There are other, more efficient methods for point multiplication which will be discussed later.

3.2 Point Addition

Point addition is the addition of two points P and Q on an elliptic curve to obtain another point R on the same elliptic curve. This is demonstrated geometrically in Figure 2 for the condition that $Q \neq -P$.

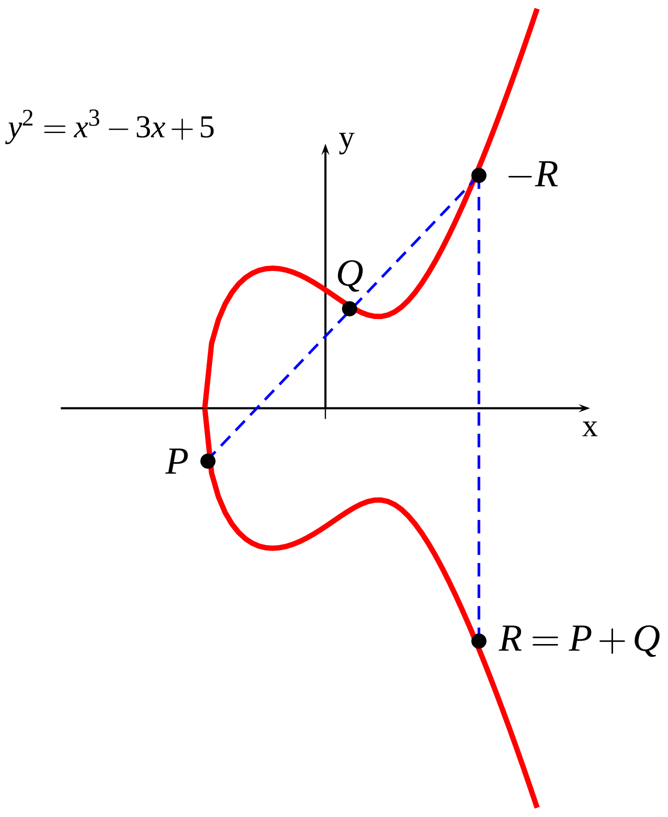

Figure 2: Elliptic Curve Addition Operation for $Q \neq -P$

If Q = -P the line through this point intersects at a point at infinity O. Hence P + (-P) = O. This is shown in Figure 3. O is the additive identity of the elliptic curve group. A negative of a point is the reflection of that point with respect to x-axis.

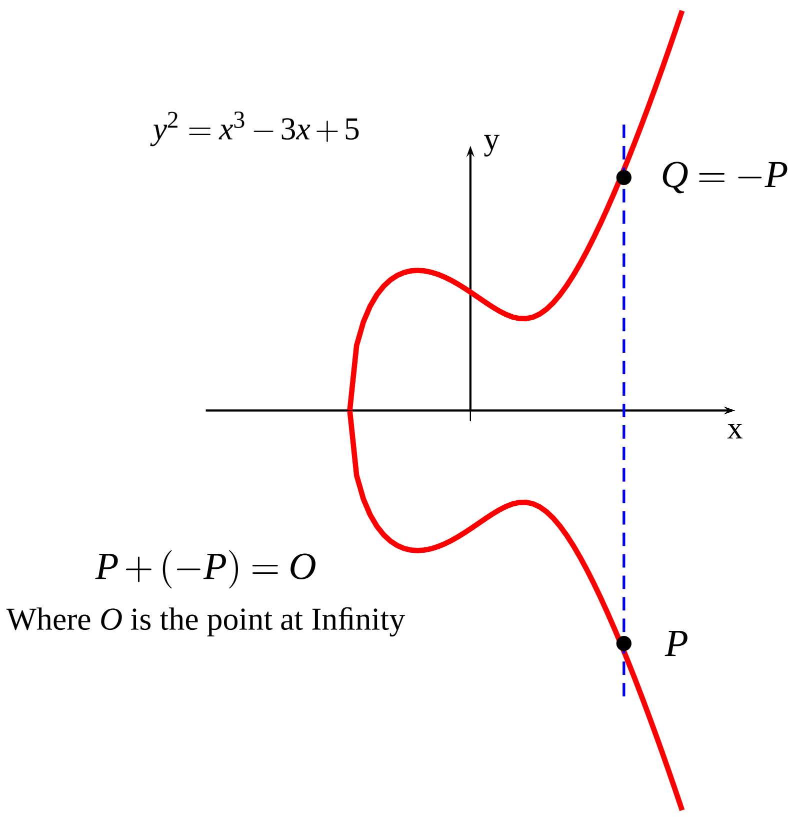

Figure 3: Elliptic Curve Addition Operation for $Q = -P$

Analytically, we can perform a point addition as follows. Consider two distinct points P and Q so that $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$.
Let $R = P + Q$ where $R = (x_R, y_R)$, then
$x_R = s^2 - x_P - x_Q$
$y_R = -y_a + s(x_P - x_Q)$
$s = (y_P - y_Q)/(x_P - x_Q)$, thus s is the slope of the line through P and Q.
If $Q = -P$ i.e. $Q = (x_P, -y_P)$ then $P + Q = O$ where O is the point at infinity.
If $P = Q$ then $P + Q = 2P$ then point doubling equations are used.
Also note that the addition is commutative, thus $P + Q = Q + P$.

3.3 Point Doubling

Point doubling is the addition of a point P on the elliptic curve to itself to obtain another point Q on the same elliptic curve. To double a point P to get Q, i.e. to find $Q = 2P$, consider a point P on an elliptic curve as shown in Figure 4. If the y coordinate of the point P is not zero then the tangent line at P will intersect the elliptic curve at exactly one more point $-Q$. The reflection of the point $-Q$ with respect to x-axis gives the point Q, which is the result of doubling the point P.

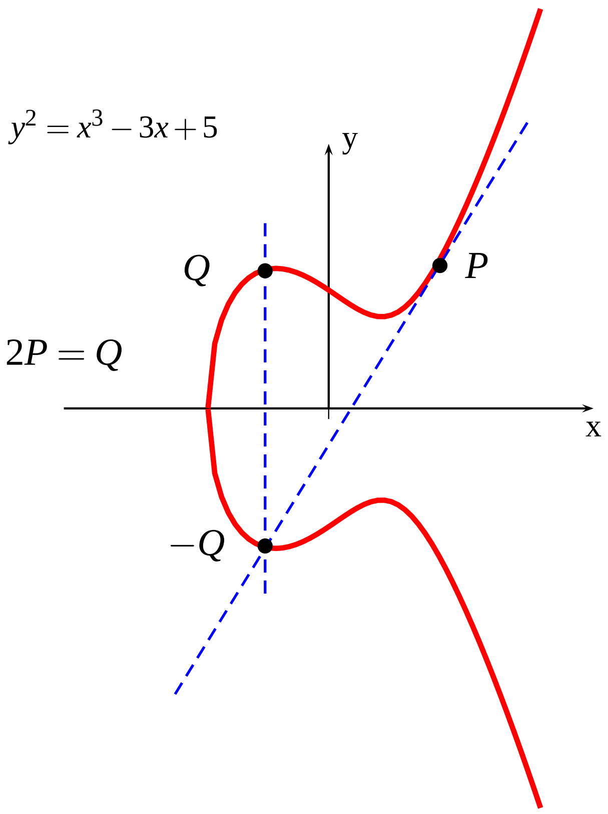

Figure 4: Elliptic Curve Doubling Operation for $y_P \neq 0$

If the y coordinate of the point P is zero then the tangent at this point intersects at a point at infinity O. Hence $2P = O$ when $y_P = 0$. This is shown in Figure 5.

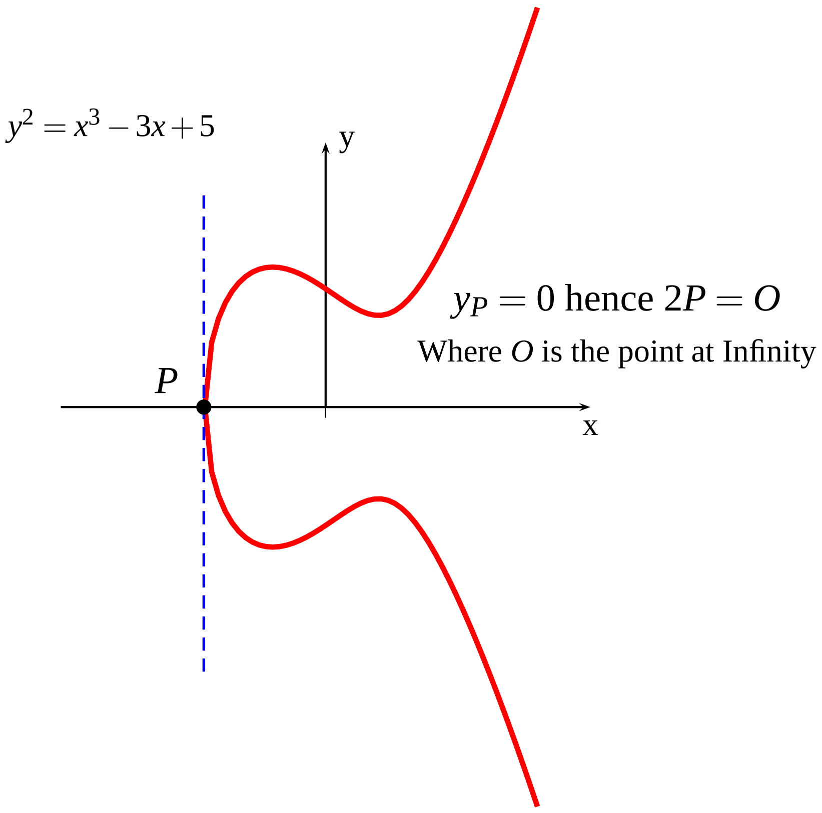

Figure 5: Elliptic Curve Doubling Operation for $y_P = 0$

Analytically, we can again write this as follows. Consider a point P such that $P = (x_P, y_P)$, where $y_P \neq 0$.
Let $Q = 2P$ where $Q = (x_Q, y_Q)$, Then
$x_Q = s^2 2x_P$
$y_Q = -y_P + s(x_P - x_Q)$
$s = (3x_P^2 + a)/(2y_P)$, where s is the tangent at point P and a is one of the parameters chosen with the elliptic curve.
If $y_P = 0$ then $2J = O$, where O is the point at infinity.

4 Finite Fields

The elliptic curve operations defined in the previous section are on real numbers. Operations over the real numbers are slow and inaccurate due to rounding errors. Cryptographic operations have to be fast and accurate. To make operations on elliptic curve accurate and more efficient, the elliptic curve cryptography is defined over two finite fields, also called Galois fields in honor of the founder of finite field theory, Évariste Galois:

Prime field $GF(p)$

Binary field $GF(2^m)$

The field is chosen with finitely large number of points suited for cryptographic operations. Figure 6 shows a graph of the elliptic curve $y^2 + xy \pmod{p} = x^3 + x^2 + 1 \pmod{p}$ with $p = 191$. Even though the curve is no longer a gently flowing graph, the algebraic equations for point addition and doubling still apply.

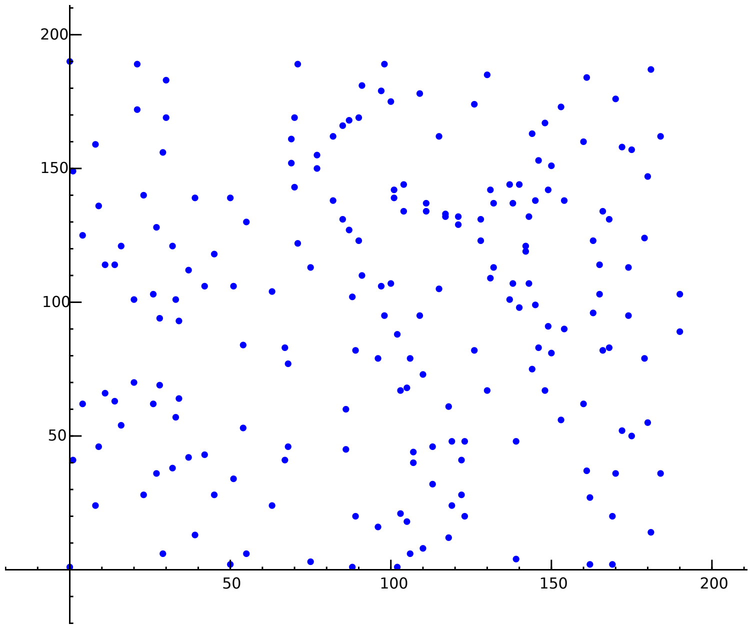

Figure 6: Elliptic Curve $y^2 + xy = x^3 + x^2 + 1$ over $GF(191)$

4.1 Operations

4.1.1 Operations over Prime Field F_p

Let F_p be a prime finite field so that p is an odd prime number, and let $a, b \in F_p$ satisfy $4a^3 + 27b^2 \pmod{p} \neq 0$. Then an elliptic curve $E(F_p)$ over F_p defined by the parameters $a, b \in F_p$ consists of the set of solutions or points $P = (x, y)$ for $x, y \in F_p$ to the equation:

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

together with the extra point O at infinity. The equation $y^2 \equiv x^3 + ax + b \pmod{p}$ is called the defining equation of $E(F_p)$. For a given point $P = (x_P, y_P)$, x_P is called the x-coordinate of P, and y_P is called the y-coordinate of P. The prime number p is chosen such that there is a finitely large number of points on the elliptic curve to make the cryptosystem secure, usually between 112 and 521 bits.

The number of points on $E(F_p)$ is denoted by $\#E(F_p)$. The Hasse Theorem states that:

$$p + 1 - 2\sqrt{p} \leq E(F_p) \leq p + 1 + 2\sqrt{p}$$

It is then possible to define an addition rule to add points on E. The addition rule is specified as follows:

1. Rule to add the point at infinity to itself:

$$O + O = O$$

2. Rule to add the point at infinity to any other point:

$$(x,y) + O = O + (x,y) = (x,y) \forall (x,y) \in E(F_p)$$

3. Rule to add two points with the same x-coordinates when the points are either distinct or have y-coordinate 0:

$$(x,y) + (x,-y) = O \forall (x,y) \in E(F_p)$$

which also means that the negative of the point (x,y) is $-(x,y) = (x,-y)$

4. Rule to add two points with different x-coordinates: Let $(x_1,y_1) \in E(F_p)$ and $(x_2,y_2) \in E(F_p)$ be two points such that $x_1 \neq x_2$. Then $(x_1,y_1) + (x_2,y_2) = (x_3,y_3)$, where:

$$x_3 \equiv \lambda^2 - x_1 - x_2 \pmod{p}$$
$$y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}$$
$$\lambda = \frac{y_2 - y_1}{x_2 - x_1} \pmod{p}$$

5. Rule to add a point to itself (double a point): Let $(x_1,y_1) \in E(F_p)$ be a point with

$y_1 \neq 0$. Then $(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$, where:

$$x_3 \equiv \lambda^2 - 2x_1 \pmod{p}$$
$$y_3 \equiv \lambda(x_1 - x_3) - y_1 \pmod{p}$$
$$\lambda = \frac{3x_1^2 + a}{2y_1} \pmod{p}$$

The set of points on $E(F_p)$ forms a group under this addition rule. Furthermore the group is Abelian, meaning that $P_1 + P_2 = P_2 + P_1$ for all points $P1, P_2 \in E(F_p)$. Notice that the addition rule can always be computed efficiently using simple field arithmetic. What about scalar multiplication of elliptic curve points? These can be computed efficiently using the addition rule together with the double-and-add algorithm or one of its variants.

4.1.2 Operations on Binary Field F_{2^m}

Let F_{2^m} be a characteristic 2 finite field, and let $a, b \in F_{2^m}$ satisfy $b \neq 0$ in F_{2^m}. Then a (non-supersingular) elliptic curve $E(F_{2^m})$ over F_{2^m} defined by the parameters $a, b \in F_{2^m}$ consists of the set of solutions or points $P = (x, y)$ for $x, y \in F_{2^m}$ to the equation:

$$y^2 + xy = x^3 + ax^2 + b$$

where $b \neq 0$ together with an extra point O at infinity. (Here the only elliptic curves over F_{2^m} of interest are non-supersingular elliptic curves) Here the elements of the finite field are integers of length at most m bits. These numbers can be considered as a binary polynomial of degree $m - 1$. In this binary polynomial the coefficients can only be 0 or 1. All the operation such as addition, substation, division, multiplication involves polynomials of degree $m - 1$ or lesser.

The number of points on $E(F_{2^m})$ is denoted by $\#E(F_{2^m})$. The Hasse Theorem states that:

$$2^m + 1 - 2\sqrt{2^m} \leq E(F_{2^m}) \leq 2^m + 1 + 2\sqrt{2^m}$$

It is again possible to define an addition rule to add points on E as it was done for $E(F_p)$. The addition rule is specified as follows:

1. Rule to add the point at infinity to itself:

$$O + O = O$$

2. Rule to add the point at infinity to any other point:

$$(x,y)+O=O+(x,y)=(x,y)\forall(x,y)\in E(F_{2^m})$$

3. Rule to add two points with the same x-coordinates when the points are either distinct or have x-coordinate 0:

$$(x,y)+(x,x+y)=O\forall(x,y)\in E(F_{2^m})$$

   which also means that the negative of the point (x,y) is $-(x,y)=(x,x+y)$

4. Rule to add two points with different x-coordinates: Let $(x_1,y_1)\in E(F_{2^m})$ and $(x_2,y_2)\in E(F_{2^m})$ be two points such that $x_1\neq x_2$. Then $(x_1,y_1)+(x_2,y_2)=(x_3,y_3)$, where:

$$\begin{aligned}&x_3=\lambda^2+\lambda+x_1+x_2+a \;\; in \;\; F_{2^m}\\&y_3=\lambda(x_1,x_3)+x_3+y_1 \;\; in \;\; F_{2^m}\\&\lambda\equiv\frac{y_1+y_2}{x_1+x_2} \;\; in \;\; F_{2^m}\end{aligned}$$

5. Rule to add a point to itself (double a point): Let $(x_1,y_1)\in E(F_{2^m})$ be a point with $x_1\neq 0$. Then $(x_1,y_1)+(x_1,y_1)=(x_3,y_3)$, where:

$$\begin{aligned}&x_3=\lambda^2+\lambda+a \;\; in \;\; F_{2^m}\\&y_3=\lambda(x_1^2)+(\lambda+1)x_3 \;\; in \;\; F_{2^m}\\&\lambda\equiv x_1+\frac{y_1}{x_1} \;\; in \;\; F_{2^m}\end{aligned}$$

The set of points on $E(F_{2^m})$ forms an Abelian group under this addition rule. Notice that the addition rule can always be computed efficiently using simple field arithmetic. As before scalar multiplication is the process of adding P to itself k times. The result of this scalar multiplication is denoted kP and can be computed efficiently using the addition rule together with the double-and-add algorithm or one of its variants.

4.2 Domain Parameters

Apart from the curve parameters a and b, there are other parameters that must be agreed by both parties involved in secured and trusted communication using ECC. These are domain parameters. The domain parameters for prime fields and binary fields are described below.

4.2.1 Domain Parameters for F_p

The domain parameters for Elliptic curve over F_p are p, a, b, G, n and h. p is the prime number defined for finite field F_p while a and b are the parameters defining the curve $y^2 \pmod{p} = x^3 + ax + b \pmod{p}$. G is the generator point (x_G, y_G), a point on the elliptic curve chosen for cryptographic operations and n is the order of the elliptic curve. The scalar for point multiplication is chosen as a number between 0 and $n-1$. h is the co-factor where $h = \frac{\#E(F_p)}{n}$. $\#E(F_p)$ is the number of points on an elliptic curve.

4.2.2 Domain Parameters for F_{2^m}

The domain parameters for elliptic curve over F_{2^m} are m, $f(x)$, a, b, G, n and h. m is an integer defined for finite field F_{2^m}. The elements of the finite field F_{2^m} are integers of length at most m bits. $f(x)$ is the irreducible polynomial, known as the reduction polynomial, of degree m used for elliptic curve operations while a and b are the parameters defining the curve $y^2 + xy = x^3 + ax^2 + b$. G is again the generator point (x_G, y_G) and a point on the elliptic curve chosen for cryptographic operations while n is the order of the elliptic curve. The scalar for point multiplication is chosen as a number between 0 and $n-1$. h is the co-factor where $h = \frac{\#E(F_{2m})}{n}$. $\#E(F_{2^m})$ is the number of points on an elliptic curve.

4.3 Choosing the Field

Should we choose a curve over the prime field F_p or the binary field F_{2^m}? This decision has to be made based on the way the system is going to be implemented. While curves over the prime field can be more efficient to implement in software, there are optimisations possible[2] that puts the efficiency of curves over a binary field on par with those over a prime field. This paper describes an ECC system using a curve over the binary field. The lack of existing implementations of ECC over binary fields compared to ECC over prime fields for low power devices gave an extra challenge and motivation for the paper.

Part III

Practical Implementation

The goals of the system we want to implement can be summarised as follows:

- Portable: the system should be easy to port to different hardware platforms
- Fast: optimisations should be made where possible, however assembly code should be limited as it hampers our first requirement.
- Efficient: use as little system resources as possible, yet don't overuse assembly language as this hinders portability.

Besides these, the following functionality should be implemented:

- Signing and verification for authentication purposes
- Encryption and decryption of data

5 Requirements and Design

Due to the computational overhead of ECC, and for public key cryptography in general, it is not really feasible to encrypt whole messages this way. Instead, a combination of symmetric and asymmetric methods are employed. In essence, the public key cryptography system based on elliptic curve cryptography is used to encrypt a symmetric key. AES is used on the symmetric side to encrypt and decrypt the actual data.

Thus, the recipient of the data first generates a public/private key-pair. The public key P is generated by such that $P = dQ$ where the multiplier d is the hashed secret password, in essence, the private key. Q is the fixed point on the curve, common to all communicating entities. The sender of the message suggests a common secret key K so that $K = kQ$ where k is a random multiplier. k can then be communicated to the receiver in the form of message $M = kP$, where the receiver can recover the symmetric key K using its private key d:

$$e = d^{-1} mod p$$

$$Me = (kP)e = (kdQ)e = (kQ)de = K$$

To implement the system, the three modules (symmetric, asymmetric and hash function) were chosen as follows:

- The asymmetric module, based on an elliptic curve over (2^{191})
- The symmetric key module, based on AES (Rijndael) with a 192 bits key in counter mode
- 192 bit TIGER[3] hash function

In other words, the cryptographic system provides a 192bit asymmetric key length for the public key operation coupled with a 192 bit symmetric key encryption.

5.1 Asymmetric Module

To get ECC working, we need to identify an elliptic curve suitable for cryptographic purposes.

The equation of the elliptic curve used is defined as follows:

$$y^2 + xy = x^3 + x^2 + (s^{12} + s^{10} + s^6 + s^2 + 1)$$

The primitive polynomial is:

$$s^{191} + s^9 + 1$$

The order of the fixed point on the curve is a number of 190 bits, p, which satisfies the MOV[4] (from Menezes, Okamoto and Vanstone) condition until 100 iterations (the minimum is 9). This means the MOV algorithm for attacking elliptic curve cryptosystems which is completed in sub-exponential time for super-singular elliptic curves does not apply.

The curve was found using Schoof's algorithm[5], with an implementation provided in the Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL) by Shamus Software [6].

The points of the curve are pairs of polynomials (x, y) but for storage, only the polynomial x is stored, and one bit that decides which of the two solutions to the quadratic equation for y one needs to pick.

5.1.1 Digital Signatures

The DSA algorithm of elliptic curves (Elliptic Curve Digital Signature Algorithm) is used to generate the signatures. The algorithm generates a pair of numbers, (r, s) from the private key of the user, d, and the hash h of the message being signed, as follows for r:

$$r = [kQ]_x mod p$$

Where k is a random multiplier. That is, r is the result of converting the polynomial x to a number of the product of the elliptic curve kQ. On the other hand for s:

$$s = k(h+dr)^{-1} mod p$$

The message and the pair (r,s) are sent to the addressee, which verifies the signature by calculating the hash of the message, h, and taking the public key of the sender, P. Then the following calculation is performed:

$$r' = [(hs)Q + (rs)P]_x mod p$$

If $r' = r$ then the signature is accepted as a valid. This is easily verifiable by replacing P and s in r'.

5.2 Symmetric Module

Although the names AES and Rijndael are often used to indicate the same system, there is a significant difference. Rijndael supports a larger range of block and key sizes; AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits, where instead Rijndael can be specified with key and block sizes in any multiple of 32 bits, with a minimum of 128 bits and a maximum of 256 bits. Since a byte normally equals 8 bits, the fixed block size of 128 bits is normally 16 bytes. AES operates on a 44 array of bytes, termed the state (versions of Rijndael with a larger block size have additional columns in the state). Most AES calculations are done in a special finite field. The cipher is specified in terms of repetitions of processing steps that are applied to make up rounds of keyed transformations between the input plain-text and the final output of cipher-text. A set of reverse rounds are applied to transform cipher-text back into the original plain-text using the same encryption key.

AES in counter mode (AES-CTR) offers a number of features over other block cipher modes and stream ciphers, among others, it provides a saving of 17 to 32 bytes per record compared to Cipher Block Chaining (AES-CBC) used in Transport Layer Security (TLS). 16 bytes are saved from not having to transmit an explicit initialisation vector (IV), and another 1 to 16 bytes are saved from the absence of the padding block.

5.3 Hash Module

The emphasis when researching an appropriate hash function was placed on speed. TIGER is one of the fastest hash functions suitable for cryptographic purposes, at least in software implementations. The size of the hash value is 192 bits. This makes a perfect match for the 190 bit asymmetric module and a 192 bit AES implementation. While TIGER was originally developed for 64-bit platforms, it can be easily adapted to 32-bit platforms. However, due to the size of TIGER's S-boxes (4 S-boxes, each with 256 64-bit entries totals 8 KB) means that implementation in hardware or smaller micro-controllers is not evident.

6 Algorithms

The most important factor to consider to make ECC feasible on relatively low-powered devices is the choice of algorithms in order to provide optimised arithmetic. This section will briefly introduce some of the key algorithms used in the implementation.

6.1 Karatsuba Multiplication

Multiplication of two elements in the polynomial basis is a very intensive operation. Using the straightforward method to multiply two elements in $GF(p^m)$ requires up to m^2 multiplications in $GF(p)$ and up to m^2m additions in $GF(p)$. Applying a method developed by Karatsuba and Ofman[7], the amount of multiplications can be reduced in exchange for an amount of additions. This trade-off will be more efficient for as long as the the time ratio for multiplication is higher than addition. The overhead in breaking down and recombining the parts involved in Karatsuba make it less suited for hardware implementations, but it is often used in software.

Let $A(x)$ and $B(x)$ be two polynomials of degree one.

$$A(x) = a_1x + a_0$$
$$B(x) = b_1x + b_0$$

The traditional method for multiplying $A(x)$ and $B(x)$ would require the following steps:

$$\begin{aligned} D_0 &= a_0b_0 \\ D_1 &= a_0b_1 \\ D_2 &= a_1b_0 \\ D_3 &= a_1b_1 \end{aligned}$$

After which the product $C(x) = A(x).B(x)$ is calculated:

$$C(x) = D_3x^2 + (D_2 + D_1)x + D_0$$

Following the Karatsuba method, we can start by taking the two polynomials and calculating the following products:

$$\begin{aligned} E_0 &= a_0b_0 \\ E_1 &= a_1b_1 \\ E_2 &= (a_0 + a_1)(b_0 + b_1) \end{aligned}$$

The result $C(x) = A(x).B(x)$ is then calculated as follows:

$$C(x) = E_1x^2 + (E_2 - E_1 - E_0)x + E_0$$

The end result is that in the traditional method requires four multiplications and one addition, however, the Karatsuba method requires three multiplications and four additions. We thus exchanged one multiplication for three additions. In $GF(2^m)$, addition is especially easy, since addition and subtraction modulo 2 are the same thing and can be done using a basic XOR operation.

6.2 Itoh-Tsujii Inversion

Inversion operations in the field are costly operation. While originally developed for use in normal basis representation over $GF(2^m)$, the Itoh-Tsujii Inversion[8], is generic and can be applied for other bases such as the polynomial basis. The general form of the algorithm is as follows:

$$A^{-1} = (A^r)^{-1}A^{r-1} \text{ where } r = \frac{p^{m-1}}{p-1}$$

The algorithm presented in Algorithm 1 below illustrates the general workings of this algorithm.

Algorithm 1: Itoh-Tsujii Inversion

```
Input: $A \in GF(p^m)$
Output: $A^{-1}$
1 $r \leftarrow (p^m - 1)/(p - 1)$
2 compute $A^{r-1}$ in $GF(p^m)$
3 compute $A^r = A^{r-1}A$
4 compute $(A^r)^{-1}$ in $GF(p)$
5 compute $A^{-1} = (A^r)^{-1}A^{r-1}$
6 return $A^{-1}$
```

This algorithm is fast because steps 3 and 5 both involve operations in the sub-field $GF(p)$. Similarly, if a small value of p is used, a look-up table can be used for inversion in step 4. The majority of time spent in this algorithm is in step 2, the first exponentiation. This is one reason why this algorithm is well-suited for the normal basis, since squaring and exponentiation are relatively easy in that basis. However, since r is known ahead of time, an efficient addition chain for the exponentiation in step 2 can be precomputed and hard-coded into the algorithm.

6.3 de Rooij Point Multiplication

As pointed out before, the most occurring operation in ECC is point multiplication; $Q = kP$. For large values of k, computing kP is an expensive task. Some methods used for ordinary integer exponentiation can be adapted to improve these operations. The (binary)-double-and-add algorithm [9] is perhaps the most well-known algorithms in this regard. It is also known as the square-and-multiply algorithm or binary exponentiation outside of the application in additive groups. It has a complexity of $\log_2(k) + WH(k)$ group operations, where WH is the Hamming weight of the multiplier k. On average, we can expect this algorithm to require $1.5\log_2(k)$ group operations. Using more advanced methods, such as signed digit, k-ary or sliding window, the complexity may be reduced to approximately $1.2\log_2(k)$ group operations on average [10]. However, it can get better, if the point is known in advance. One of the important applications of ECC is providing digital signatures. The Elliptic Curve Digital Signature Algorithm (ECDSA) [11] works by multiplying a fixed curve point by the user-generated private key as its main operation. Because the curve point is known ahead of time, pre-computations may be performed to speed up the signing process. Using a method devised by de Rooij [12], we are able to reduce the number of group operations necessary by a factor of four over the binary-double-and-add algorithm. The method, known as fixed point multiplication using pre-computation and vector addition chains, is implemented as shown in Algorithm 2.

Algorithm 2: de Rooij Fixed Point Multiplication using Pre-Computation and Vector Addition Chains

```
Require: $\{b^0A, b^1A, ..., b^tA\}, A \in E(GF(p^m))$, and $s = \sum_{i=0}^{t} s_i b^i$
Ensure: $C = sA, C \in E(GF(p^m))$
1  Define $M \in [0,t]$ such that $z_M \geq z_i$ for all $0 \leq i \leq t$
2  Define $N \in [0,t], N \neq M$ such that $z_N \geq z_i$ for all $0 \leq i \leq t, i = M$
3  for $i \leftarrow 0$ to $t$ do
4      $A_i \leftarrow b^i A$
5      $z_i s_i$
6  end
7  Determine $M$ and $N$ for $\{z_0, z_1, ..., z_t\}$
8  while $z_N \geq 0$ do
9      $q \leftarrow \lfloor Z_M / Z_N \rfloor$
10     $A_N \leftarrow qA_M + A_N$          (Here we apply binary-double-and-add)
11     $z_M \leftarrow z_M \bmod z_N$
12     Determine $M$ and $N$ for $\{z_0, z_1, ..., z_t\}$
13 end
14 $C \leftarrow z_M A_M$
```

Part IV

Conclusions and Future Work

7 Results

The cryptographic system was implemented on a couple of different platforms to provide a clear picture of the performance one can expect on different embedded systems. All systems are running Linux, with either GNU Libc or uClibc. The target platforms consists of:

- Nokia N800, TI OMAP 2420 clocked at 330MHz (GNU Libc)
- Freescale MPC5200B clocked at 400MHz (GNU Libc)
- Renesas SH7203 clocked at 200MHz (uClibc)
- Freescale ColdFire MCF54455 clocked at 266MHz (GNU Libc)
- Freescale ColdFire MCF52277 clocked at 160MHz (uClibc)

Table 3 gives an overview of the time each CPU needs to calculate field operations. The most used operations here include point multiplication, squaring, solving a quadratic equation to find the correct y coordinate of a point and inversion.

	multiplication	squaring	quad-solving	inversion
MPC5200B	0.0014 s	0.0000 s	0.0500 s	0.0500 s
Nokia N800	0.0043 s	0.0000 s	0.1100 s	0.0300 s
SH7203	0.0057 s	0.0100 s	0.2700 s	0.1300 s
MCF54455	0.0214 s	0.0050 s	0.4800 s	0.3500 s
MCF52277	0.0486 s	0.0250 s	1.1200 s	0.7800 s

Table 3: Field Operations

ECC scalar multiplication $Q = kP$ time needed is shown in Table 4. This is the main factor limiting speed in the rest of the cryptographic system.

The time needed to perform a set of cryptographic operations, including key generation, encryption/decryption and signature generation/verification is given in Table 5.

	scalar multiplication
Nokia N800	0.035 s
MPC5200B	0.037 s
SH7203	0.088 s
MCF54455	0.297 s
MCF52277	0.737 s

Table 4: Scalar Multiplication

	Encrypt-Decrypt	Sign/Verify	Key generation
Nokia N800	0.122 s/cycle	0.087 s/cycle	0.036 s/key
MPC5200B	0.130 s/cycle	0.100 s/cycle	0.030 s/key
SH7203	0.303 s/cycle	0.223 s/cycle	0.086 s/key
MCF54455	1.021 s/cycle	0.763 s/cycle	0.300 s/key
MCF52277	2.559 s/cycle	1.928 s/cycle	0.761 s/key

Table 5: Cryptographic Operations

It should be noted that the seemingly slow execution on ColdFire is most likely due to missing optimisations in the GCC compiler suite for this kind of CPU. Other compilers with specific optimisations for ColdFire (such as CodeWarrior) could result in better performance, but was not tested due to lack of time.

8 Future Work

The focus of future work will be on the development of an efficient Public Key Infrastructure (PKI) with implementations for sensor networks and other applications where large quantities of communicating nodes are present. Management of a large number of keys especially while certifying every key is a major obstacle that needs to be tackled before easy and large scale deployment becomes feasible.

References

[1] NIST Special Publication 800-57, “Recommendation for Key Management - Part 1: General (Revised),” 2007.

[2] E. D. Win, A. Bosselaers, S. Vandenberghe, P. D. Gersem, and J. Vandewalle, “A Fast Software Implementation for Arithmetic Operations in GF(2n),” in *ASIACRYPT ’96: Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security*, (London, UK), pp. 65–76, Springer-Verlag, 1996.

[3] R. Anderson and E. Biham, “Tiger: a fast new hash function,” in *Fast Software Encryption, Third International Workshop Proceedings*, pp. 89–97, Springer-Verlag, 1996.

[4] A. Menezes, T. Okamoto, and S. A. Vanstone, “Reducing Elliptic Curve Logarithms to Logarithms in a Finite Field,” *IEEE Transactions on Information Theory, Volume 39*, 1993.

[5] R. Schoof, “Elliptic Curves over Finite Fields and the Computation of Square Roots mod p,” *Mathematics of Computation*, vol. 44, pp. 483–494, 1985.

[6] `http://www.shamus.ie/`, “Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL),” Accessed 30/08/2008.

[7] A. A. Karatsuba and Y. Ofman, “Multiplication of Multidigit Numbers on Automata,” *Soviet Physics Doklady*, vol. 7, pp. 595–+, Jan. 1963.

[8] T. Itoh and S. Tsujii, “A Fast Algorithm for Computing Multiplicative Inverses in GF(2m) Using Normal Bases,” *Information and Computation*, vol. 78, pp. 171–177, 1988.

[9] D. E. Knuth, “The Art of Computer Programming. Volume 2: Seminumerical Algorithms. 2nd edition,” 1981.

[10] S. A. V. Alfred Menezes, Paul C. van Oorschot, “Handbook of Applied Cryptography,” 1997.

[11] “Standard Specifications for Public Key Cryptography,” *IEEE Standard*, pp. 1363–2000, 2000.

[12] P. de Rooij, “Efficient Exponentiation using Precomputation and Vector Addition Chains,” *Advances in Cryptography - EUROCRYPT ’98*, pp. 389–399, 1998.