Comparative Study of Elliptic Curve Cryptography and RSA in Constrained Environment

GRADUATE PROJECT REPORT

Submitted to the Faculty of
The Department of Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Soumya Kancheti
Spring 2010

Committee Members

Dr. Dulal Kar
Committee Chairperson

Dr. David Thomas
Committee Member

Dr. Hongyu Guo
Committee Member
ABSTRACT

The tremendous increase in the use of mobile and wireless devices with limitations on power and bandwidth postulates a new generation of Public Key Cryptography (PKC) schemes. These PKC schemes should overcome the limitations of the mobile and wireless devices and help in achieving efficiency. At the same time, the new PKC schemes should be capable enough to provide an adequate level of security for such devices.

This project examines the use of Elliptic Curve Cryptography (ECC) in such a constrained environment along with the other two aspects of ECC, namely its security and efficiency. In the project, the performance of ECC is compared with the other PKC applications which should prove that ECC performs better and is more suitable for constrained environments.
# TABLE OF CONTENTS

Abstract.........................................................................................................................ii

Table of Contents........................................................................................................iii

1. Background and Rationale.........................................................................................1
   1.1. Mobile Phone Characteristics (Hardware Info)...............................................4
   1.2. Cryptography......................................................................................................6
       1.2.1. The Need for Public Key Cryptography.................................................6
           1.2.1.1. Key Management Problem.................................................................7
           1.2.1.2. No Digital Signatures Possible............................................................7
       1.2.2. Choice of Public Key Cryptography System..........................................8
   1.3. Elliptic Curve.....................................................................................................9
       1.3.1. Elliptic Curve Arithmetic............................................................................9
       1.3.2. Elliptic Curves over GF(2ⁿ).......................................................................9
       1.3.3. Elliptic Curve Cryptography....................................................................10
       1.3.4. Key Exchange.............................................................................................11
       1.3.5. Elliptic Curve Encryption/Decryption.......................................................11
   1.4. RSA Arithmetic................................................................................................12
       1.4.1. Key Generation..........................................................................................13
       1.4.2. Communication..........................................................................................14
           1.4.2.1. Encryption..........................................................................................14
           1.4.2.2. Decryption..........................................................................................15
   2. Narrative .................................................................................................................16
   2.1. Properties of Java-Enabled Devices.................................................................17
2.1.1. Implementation Issues ......................................................... 19
  2.1.1.1. Performance .............................................................. 19
  2.1.1.2. Code Size ............................................................... 20

3. Proposed System Design or Proposed Research .................................. 21
  3.1. Public-Key Cryptography Challenges for Constrained Devices .......... 21
  3.2. Current Mobile and Wireless Devices Characteristics ...................... 22
  3.3. Limitations of J2ME .......................................................... 22
  3.4. Implementation issues for MIDlets .......................................... 23
  3.5. Implementation of RSA ...................................................... 24
  3.6. Implementation of ECC ...................................................... 24

4. Testing and Evaluation .................................................................. 26
  4.1. Computational Overhead ........................................................ 27
  4.2. Experiment Conducted .......................................................... 29
  4.3. Experimented Results .......................................................... 34

5. Conclusion .................................................................................... 38
  5.1. Fighting Tomorrow’s Hackers .................................................. 38
  5.2. Secure for Centuries ............................................................. 39

References and Bibliography ............................................................. 40
Appendix A: Partial Code .................................................................. 41
LIST OF FIGURES

Figure 4.1. Application Start Screen.................................................................29
Figure 4.2. Key Generation using RSA...............................................................30
Figure 4.3. Key Generation Elapsed Time using RSA........................................31
Figure 4.4. Encrypt/Decrypt using RSA..............................................................32
Figure 4.5. Key Generation using ECC...............................................................33
Figure 4.6. Encrypt/Decrypt using RSA and ECC for comparison.....................34
Figure 4.7. Comparison of RSA and ECC Key Generation Time......................36
Figure 4.8. Comparison of RSA and ECC Memory Requirement......................36
Figure 4.9. Comparison of RSA and ECC Encrypt/Decrypt time.......................37
1. BACKGROUND AND RATIONALE

Rapid growth of mobile communications and computing devices like cell phones and PDA’s is in today’s market. Integrations of Java Virtual Machine (JVM) address the application development for such a great number of devices. Sun Microsystems java 2 platforms conforms most of today’s devices fast deployment of mobile and wireless applications allowed by J2ME. Security is needed for many of these applications such as applications for mobile electronic payment and authentication to access security networks etc. Satisfaction of this need of security is employed by cryptographic secret-key and public-key algorithm.

Public key infrastructure (PKI) is required by the public key methods. Connection and validation of the public keys to the respective owner is done by PKI. Generation, provision and revocation of user certificates are done by PKI’s which are usually done by trusted parties called (CA). Once established all network enabled devices are provided with PKI’s.

The use of security systems as open systems is a good practice. An open system complies with specified, publicly maintained, readily available standards and can therefore be connected to other systems that comply with the same standards. Such systems bear many advantages, e.g. interoperability, flexibility, and public acceptance. For example an operator of common portal may want to enable custom to buy using cell phone. Secrecy, protection from data manipulation and non repudiation are the main requirements.

In addition users may need e-commerce portal to authenticate their own selves. A set of public and secret key cryptographic methods provide all these services. The
usefulness of public-key methods in mobile and wireless applications is evident, but they come at a price. Every public-key algorithms need sophisticated math computations. Constrained devices cannot offer needed resources to allow an implementation of the needed public-key protocols.

The use of hand held devices using JVM has greatly beneficiated the development of mobile applications. Many applications such as e-commerce in which security is very much needed is satisfied by the public key cryptography methods.

Today’s high mobile workforce largely depends on cell phones and PDA’s. These devices are not only used for voice calls, text messages, personal information management (PIM’s), but also used in sending and receiving emails, browsing web, storing and modifying data which are becoming inexpensive as a result included in the upcoming devices, in addition to built-in software’s such as GPS, Camera, Wi-Fi, Bluetooth etc.

As those provide productive benefits, they also have too many risks:

- Because of their small size, they can be easily stolen or misplaced. Thereby Information can be accessed easily if they fall in wrong hands.
- Malware can be delivered to handheld devices by a communication network, desktop synchronization and tainted storage. Malware is always in the form of game, device patch, utility, or other third party application.
- Similar to desktop computers, cell phones and PDAs are subject to spam, but this can include text messages and voice mail, in addition to electronic mail. Besides the inconvenience of deleting spam, charges may apply for inbound activity. Spam can also be used for phishing attempts.
• Phone calls, messages and other wireless transmitted data can be eavesdropped by various techniques. Spy software installation on a device is used to collect and forward data to somewhere else. Conversation can be captured by using a built-in microphone.

• Registered cell phones can be monitored and their location can be tracked.

• Clones of the cell phones can also be created which can masquerade the original one.

• Sensitive information can be exposed by the Server-resident content which is maintained by the network carrier.

Today, dangers such as malware are limited in the handheld devices compared to desktops. The main reason for this is no single operating systems dominates the single device to same extent. Closed system approach which executes control over device and application are adopted by the network carriers. Increase of the malware for handheld devices has been detected in the past few years which affects the increase in recent trend of open systems and which allows application development and flexibility in choosing devices and application from other sources. But this would also affect the increase of malware development and potentially increase the chance of attack to exploit.

Even though the productivity benefits are provided by many handheld devices, these devices could be potential threat to an organization security. The handheld devices such as SIMs and memory cards hoard the sensitive organizational and personal information. Over time topics like financial statements, product announcements and litigation issues can be discussed through mobile email on a device. The attacker may pose a threat to information such as passwords, phone book and calendar entries, audio
and video, electronic documents etc. The associated and information risk is directly proportional to the memory capacity of devices. The potential threats can also be the remote sources directly accessed by a device by wired and wireless communications. Voice mail, email repositories, cell phone services and data and applications on accessible corporate networks can also be under attack.

Mobile hand held devices fall short of many significant security features mostly found on desktops. These devices are important to organizations infrastructure, where administering them centrally is difficult. Organizations and individuals who are associated with them are aware of the risks involved and can reduce the associated risks with the security features and other techniques:

The Security challenges are

- Less processing power on devices
- Slow modular exponentiation and primarily checking (i.e., RSA)
- Crypto operations drain batteries (CPU intensive!)
- Less memory (keys, certs, etc. require storage)
- Few devices have crypto accelerators, or support for biometric authentication
- No tamper resistance (memory can be tampered with, no secure storage)
- Primitive operating systems with no support for access control (Palm OS)

1.1. Mobile Phone Characteristics (Hardware Info)

Mobile phones are highly mobile communications devices that perform an array of functions ranging from that of a simple digital organizer to that of a low-end personal
computer. Designed for mobility, they are compact in size, battery powered, and lightweight. Most cell phones have a basic set of comparable features and capabilities. They house a microprocessor, read only memory (ROM), random access memory (RAM), a radio module, a digital signal processor, a microphone and speaker, a variety of hardware keys and interfaces, and a liquid crystal display (LCD). The operating system (OS) of the device is held in ROM, which with the proper tools typically can be erased and reprogrammed electronically. RAM, which for certain models may be used to store user data, is kept active by batteries, whose failure or exhaustion causes that information to be lost.

The latest cell phones come equipped with system-level microprocessors that reduce the number of supporting chips required and include considerable memory capacity. Built-in Mini Secure Digital MiniSD, MultiMedia Card Mobile, MMCmobile, or other types of card slots support removable memory cards or specialized peripherals, such as an SDIO Wi-Fi card. Wireless communications such as infrared (i.e., IrDA) or Bluetooth may also be built into the device.

Different devices have different technical and physical characteristics (e.g., size, weight, processor speed, memory capacity). Devices may also use different types of expansion capabilities to provide additional functionality. Furthermore, cell phone capabilities sometimes include those of other devices such as PDAs, global positioning systems, and cameras. Overall, they can be classified as basic phones that are primarily simple voice and messaging communication devices; advanced phones that offer additional capabilities and services for multimedia; and smart phones or high-end phones that merge the capabilities of an advanced phone with those of a PDA.
1.2. Cryptography

In 1976, Whitfield Diffie and Martin Hellman introduced the concept of Public Key Cryptography (PKC). Since then, many implementations of it have been proposed, and many of these cryptographic applications base their security on the intractability of hard mathematical problems, namely the Integer Factorization Problem (IFP) and the finite field Discrete Logarithm Problem (DLP). Over the years, sub-exponential time algorithms were developed to solve these problems. As a result, key sizes grew to more than 1000 bits, so as to attain a reasonable level of security. In constrained environments where computing power, storage and bandwidth are limited, carrying out thousand-bit operations becomes an impractical approach to providing adequate security. This is most evident in hand-held devices such as the mobile phones, pagers and PDAs that have very limited processing power and battery life. [Koblitz 2008]

In 1985 both Victor Miller and Neal Koblitz independently proposed Elliptic Curve Cryptography which has the unique feature that to date is the most effective known algorithm which runs in full exponential time can solve it. The protection for ECC comes from elliptic curve logarithm, which is DLP in a collective group specified by points on an elliptic curve over a finite field. This leads to a striking decrease in key size required to attain the similar level of security extended in conventional PKC schemes. [Koblitz 2008]

1.2.1. The Need for Public Key Cryptography

Encryption of data is mostly done by Private Key Cryptography because of its speed. Data Encryption Standard (DES) which has tremendously fast encryption speed
and efficiency is the most widely used today. But DES is unsuitable for the mobile commerce environment applications due to certain shortcomings [Koblitz 2008].

1.2.1.1. Key Management Problem

The wireless user must be capable of doing business with a large number of different enterprises, instead of only one. Therefore, communication on a public network is done by many users interacting with each other, rather than confining it to one-on-one. If $n$ is small, $n(n-1)/2$ private keys are to be generated for a network of $n$-users and if $n$ is large, the number of private keys become unmanageable [Koblitz 2008].

When one has to generate such a large number of keys, the task of generating the keys and finding a secured distribution channel becomes a difficult task on networks.

1.2.1.2. No Digital Signatures Possible

An electric analogue for a hand written signature is digital signature. If Bob got an encrypted message from Alice, then Bob must be able to make sure that the received message is from Alice, with the help of Alice’s signature. This capability is not a feature of Private Key Cryptography.

In direct contrast, two keys are used by Public Key Cryptography (PKC). On a network, every user puts out a public encryption key that anyone can use to send messages; however, the private key is kept secret for decryption. PKC requires $n$ private and $n$ public keys on a network of $n$-users. This decreases the number of keys required from $O(n^2)$ to $O(n)$. Moreover, PKC admits the use of digital signatures, which guarantees non-reunification. Yet Public Key Cryptography is much slower compared to Private Key Cryptography. DES requires just 64-bits while RSA, the most widely user public key algorithm that supports encryption and digital signatures, requires at least
1024-bit keys. In truth, Public and Private Key Cryptography work best together. Private Key Cryptography is best suited for ensuring confidentiality like encrypting data and communication channels whereas Public Key Cryptography is most suited for ensuring data integrity, key distribution and management, providing authentication and non-repudiations which are the most important objectives that play a vital role in any cryptographic application. [Schneier 1994]

1.2.2. Choice of Public Key Cryptography System

Bandwidth, battery life and memory are the main constraints one has to keep in mind when deciding which type of Public Key Cryptosystem should be used in mobile environments. Wireless pagers, PDAs and mobile phones have constrained environments and are highly limited. So an appropriate public key system would be one that is more efficient in terms of key sizes, computing costs as well as security.

Until today, no Public Key Cryptography system has a higher strength-per-bit than ECC. Small key sizes transform into savings in processing power, bandwidth and memory. ECC has been the best choice in these cases. Other aspects are also required to be taken into account. The next chapter discusses the different mathematical problems that are vital in Public Key Cryptography systems which are used today. Some of the most effective algorithms that solve them are also discussed. This gives us a more clear understanding of the security on which many Public Key Cryptography systems are established.
1.3. Elliptic Curve [Stallings 2003]

1.3.1. Elliptic Curve Arithmetic

Elliptic curves are not ellipses. They are so named because they are described by cubic equations, similar to those used for calculating the circumference of an ellipse. The shape of OZONE layer which is found in upper atmosphere has the elliptical shape. Since OZONE is used for Encryption and Decryption of Sunlight and Heat from Earth, and its shape is determined by ellipse circumference measuring equations, these equations for the cryptography purpose are called ELLIPTIC CURVES. In general cubic equations for elliptic curves take the form

\[ Y^2 + AXY + BY = X^3 + CX^2 + DX + E \]

Where \( a, b, c, d \) and \( e \) are real numbers and \( x \) and \( y \) take on their values in real numbers.

The order of a point \( P \) is the smallest positive integer such that \( nP = O \).

The order of an elliptic curve \( E \) defined over the field \( F_q \) is the number of points defined over \( F_q \), and including the point at infinity \( O \).

1.3.2. Elliptic Curves over (Galois field)\( GF(2^n) \)

For elliptic curves over \( GF(2^n) \), we use a cubic equation in which the variables and coefficients all take on values in \( GF(2^n) \), for some number \( n \), and in which the calculations are performed using the rules of arithmetic in \( GF(2^n) \). \( GF(2^n) \) is widely used in applications such as error correction codes, and complicated combinations of arithmetic operations are performed in those applications.

The cubic form appropriate for cryptographic applications for elliptic curves for \( (2^n) \) is

\[ Y^2 + XY = X^3 + AX^2 + B \]
Where variables x and y and the coefficients a and b are elements of GF \((2^n)\) and the calculations are performed in GF \((2^n)\).

Algorithm to find all possible elliptic curves

The point at infinity is \(O\).

1. **ADDITIVE IDENTITY:**
   \[ P + O = P \]

2. **NEGATIVE POINT:**
   \[ P = (x_P, y_P) \text{ then } -P = (x_P, x_P + y_P) \]

**ADDITION:**

\[ P = (x_P, y_P) \text{ and } Q = (x_Q, y_Q) \text{ with } P \neq -Q \text{ and } P \neq Q \text{ then} \]
\[ R = P + Q = (x_R, y_R) \text{ is determined by the following rules:} \]
\[ x_R = \lambda 2 + \lambda + x_P + x_Q + a \]
\[ y_R = \lambda (x_P + x_R) + x_R + y_P \]
Where \( \lambda = (y_Q + y_P)/(x_Q + y_P) \)

3. **DOUBLE:**

\[ P = (x_P, y_P), \text{ then } R = 2P = (x_R, y_R) \]
\[ x_R = \lambda^2 + \lambda + a \]
\[ y_R = x_R^2 (\lambda + 1) x_R \]
where \( \lambda = x_P + y_P/ x_P \)

1.3.3. **Elliptic Curve Cryptography**

To form a cryptographic system using elliptic curves, we need to find a “hard problem” corresponding to factoring the product of two primes or taking the discrete algorithm.

Consider the equation \( Q = kP \), where \( Q, P \) belongs to elliptic curve over GF\((2^n)\) and \( k < 2^n \). It is relatively easy to calculate \( Q \) given \( k \) and \( P \), but it is relatively hard to determine \( k \) given \( Q \) and \( P \). This is called the discrete algorithm problem for elliptic curves.
1.3.4. Key Exchange

Key exchange can be done in the following manner. A large integer $q = 2^n$ is picked and elliptic curve parameters $a$ and $b$. This defines an elliptic curve group of points. Now, pick a base point $G = (x_1, y_1)$ in $E(a,b)$ whose order is a very large value $n$. The elliptic curve $E$ and $G$ are the parameters known to all participants.

A key exchange between users A and B can be accomplished as follows:

1. A selects an integer $n_A$ less than $n$. This is A’s private key. A then generates a public key $P_A = n_A * G$; the public key is appoint on $E$.
2. B similarly selects a private key $n_B$ and computes a public key $P_B$.
3. A generates the secret key $K = n_A * P_B$. B generates the secret key $K = n_B * P_A$.

The calculations in step 3 produce the same result.

$$n_A * P_B = n_A * (n_B * G) = n_B * (n_A * G) = n_B * P_A$$

To break this scheme, an attacker would need to be able to compute $k$ given $G$ and $kG$, which is assumed hard.

1.3.5. Elliptic Curve Encryption/Decryption

The plaintext message $m$ is taken as input in the form of bits of varying length. This message $m$ is encoded and is sent in the cryptographic system as $x$-$y$ point $P_m$. This point is encrypted as cipher text and subsequently decrypted. The SHA hash function algorithm can be used as Message digestion and Signature authentication and verification for the message.

As with the key exchange system, an encryption/decryption system requires a point $G$ and an elliptic group $E(a,b)$ as parameters. Each user A selects a private key $n_A$ and generates a public key $P_A = n_A * G$. 
To encrypt and send a message $P_m$ to $B$, $A$ chooses a random positive integer $k$ and produces the ciphertext $C_m$ consisting of pair of points

$$C_m = \{ kG, P_m + kP_B \}$$

$A$ has used $B$’s public key $P_B$. To decrypt the ciphertext, $B$ multiplies the first point in the pair by $B$’s secret key and subtracts the result from the second point:

$$P_m + kP_B - n_B(kG) = P_m + k(n_BG) - n_B(kG) = P_m$$

The implementation of elliptic curve algorithm is done over $GF(2^{163})$ for providing security of more than 128 bits.

### 1.4. RSA Arithmetic

The RSA scheme is a block cipher. Each plaintext block is an integer between 0 and $n - 1$ for some $n$, which leads to a block size $\leq \log_2(n)$. The typical block size for RSA is 1024 bits. The details of the RSA algorithm are described as follows. The RSA algorithm was invented by Ronald L. Rivest, Adi Shamir, and Leonard Adleman in 1977. The security of the algorithm is based on the hardness of factoring a large composite number and computing eth roots modulo a composite number for a specified odd integer.

<table>
<thead>
<tr>
<th>Key Generation</th>
<th>Encryption $C = P^e \mod n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Generate two large prime numbers, $p$ and $q$</td>
<td>2. Let $n = pq$</td>
</tr>
<tr>
<td>3. Let $m = (p-1)(q-1)$</td>
<td>4. Choose a small number $e$, coprime to $m$</td>
</tr>
<tr>
<td>5. Find $d$, such that $de \mod m = 1$</td>
<td>6. Publish $e$ and $n$ as the public key.</td>
</tr>
<tr>
<td></td>
<td>7. Keep $d$ and $n$ as the secret key.</td>
</tr>
</tbody>
</table>

$x \% y$ means the remainder of $x$ divided by $y$
The reasons why this algorithm works are discussed in the mathematics section. Its security comes from the computational difficulty of factoring large numbers. To be secure, very large numbers must be used for p and q - 100 decimal digits at the very least.

I'll now go through a simple worked example.

1.4.1. Key Generation

1) Generate two large prime numbers, p and q

To make the example easy to follow I am going to use small numbers, but this is not secure. To find random primes, we start at a random number and go up ascending odd numbers until we find a prime.

Let’s have:

\[
\begin{align*}
p &= 7 \\
q &= 19
\end{align*}
\]

2) Let \( n = pq \)

\[
\begin{align*}
n &= 7 \times 19 \\
&= 133
\end{align*}
\]

3) Let \( m = (p - 1)(q - 1) \)

\[
\begin{align*}
m &= (7 - 1)(19 - 1) \\
&= 6 \times 18 \\
&= 108
\end{align*}
\]

4) Choose a small number, e coprime to m

\( e \) coprime to \( m \), means that the largest number that can exactly divide both \( e \) and \( m \) (their greatest common divisor, or GCD) is 1. Euclid's algorithm is used to find the GCD of two numbers, but the details are omitted here.

\[
\begin{align*}
e = 2 &\Rightarrow \text{GCD}(e, 108) = 2 \text{ (no)} \\
e = 3 &\Rightarrow \text{GCD}(e, 108) = 3 \text{ (no)} \\
e = 4 &\Rightarrow \text{GCD}(e, 108) = 4 \text{ (no)} \\
e = 5 &\Rightarrow \text{GCD}(e, 108) = 1 \text{ (yes!)}
\end{align*}
\]
5) Find $d$, such that $de \equiv m = 1$

This is equivalent to finding $d$ which satisfies $de = 1 + nm$ where $n$ is any integer. We can rewrite this as $d = (1 + nm) / e$. Now we work through values of $n$ until an integer solution for $e$ is found:

- $n = 0 \Rightarrow d = 1 / 5$ (no)
- $n = 1 \Rightarrow d = 109 / 5$ (no)
- $n = 2 \Rightarrow d = 217 / 5$ (no)
- $n = 3 \Rightarrow d = 325 / 5 = 65$ (yes!)

To do this with big numbers, a more sophisticated algorithm called Extended Euclid must be used.

<table>
<thead>
<tr>
<th>Public Key</th>
<th>Secret Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 133$</td>
<td>$n = 133$</td>
</tr>
<tr>
<td>$e = 5$</td>
<td>$d = 65$</td>
</tr>
</tbody>
</table>

1.2. Results for above calculation for comparison

1.4.2. Communication

1.4.2.1. Encryption

The message must be a number less than the smaller of $p$ and $q$. However, at this point we don't know $p$ or $q$, so in practice a lower bound on $p$ and $q$ must be published. This can be somewhat below their true value and so isn't a major security concern. For this example, let's use the message "6".

$$C = P^e \mod n$$
$$= 6^5 \mod 133$$
$$= 7776 \mod 133$$
$$= 62$$
1.4.2.2. Decryption

This works very much like encryption, but involves a larger exponentiation, which is broken down into several steps.

\[ P = C^d \mod n \]
\[ = 62^{65} \mod 133 \]
\[ = 62 \times 62^{64} \mod 133 \]
\[ = 62 \times (62^2)^{32} \mod 133 \]
\[ = 62 \times 3844^{32} \mod 133 \]
\[ = 62 \times (3844 \mod 133)^{32} \mod 133 \]
\[ = 62 \times 120^{32} \mod 133 \]

We now repeat the sequence of operations that reduced \( 62^{65} \) to \( 120^{32} \) to reduce the exponent down to 1.

\[ = 62 \times 36^{16} \mod 133 \]
\[ = 62 \times 99^{8} \mod 133 \]
\[ = 62 \times 92^{4} \mod 133 \]
\[ = 62 \times 85^{2} \mod 133 \]
\[ = 62 \times 43 \mod 133 \]
\[ = 2666 \mod 133 \]
\[ = 6 \]
2. NARRATIVE

I have implemented Public-key Cryptography using JAVA in Constrained Device and I have chosen NOKIA devices for this implementation. Elliptic curve and RSA are the two public key cryptographic algorithms that are implemented on the mobile device to analyze the efficiency.

Public-key methods are generally based upon so-called trapdoor one-way functions. These are mathematical functions which are easy to compute in one direction, but are hard to invert. However, the inversion can be facilitated if one is in the possession of some piece of additional information: the so-called trapdoor information. The main current public-key algorithms rely on the hardness of one of the two mathematical problems: integer factorization (IF) or discrete logarithm problem (DLP). The equivalent of the DLP for elliptic curves—denoted as elliptic curve discrete logarithm problem (ECDLP)—is particularly interesting, as no sub exponential-time algorithm for solving it has been discovered so far. This fact distinguishes Elliptic Curve (EC) algorithms from other cryptosystems which are based on IF or DLP like RSA and Diffie-Hellman. In contrast to the ECDLP, there are known sub exponential-time algorithms for solving both the IF problem and the DLP in conventional number fields.

A general relation exists between the hardness of the underlying problem and the minimal length of the operands, i.e., keys and other parameters, of the public-key algorithm. For a given cryptographic algorithm and a desired level of security the operands must have a certain length. The operand length has a direct impact on the performance and memory requirements of an implementation of the algorithm. The assumed hardness of ECDLP results in shorter operands for EC methods in comparison
to other algorithms. For instance, a key size of 190 bit for an EC algorithm is approximately equivalent to an RSA key size of 1937 bit under the condition that there will be some progress made towards more efficient solutions of the ECDLP in the future. If no such progress is made, then the required RSA key for equivalent security even grows to over 3137 bit.

2.1. Properties of Java-Enabled Devices [Stefan 2004]

Attached selections to implement public-key algorithms in constrained devices are EC methods. But every public-key procedure needs significant computing resources. Moreover in spite of the several advantages extended by java had some constraints like, poor performance in comparison to native code. Hence attaining a decent performance is the toughest challenge of performing public-key algorithms in java on constrained devices.

Sun Microsystems’ Java 2 Platform, Micro Edition (J2ME) is the best supported mechanism for java deployment on hand held devices. Connected limited device profile (CLDC) which is presently available in version 1.0 and 1.1 had the base set of API’s defined in them. Today’s most java enabled hand held devices use CLDC combined with MIDP make the java runtime environment.

The MIDP is available in version 1.0 and 2.0. MIDlets are the applications which adapt to MIDP. These days several mobile devices are implemented by a thinned down variant of the standard java API which is provided by MIDP version 1.0. MIDP 2.0 offers defined support for public-key certificates, games and media control. However secure socket streams and secure connections over https are provided. More over there is no
access to the cryptographic algorithms which carry out the secure connection. Hence jobs such as data signing cannot be performed with the MIDP 2.0 API.

Including the needed cryptographic algorithms in to MIDP is the most beneficial solution for the provision of general public-key methods for the application programmer. In this case the computation extensive techniques can be applied efficiently by java runtime environment (JRE) and can utilize each and every feature of the corresponding device. Till now no version of MIDP extends support for cryptographic methods. Even though it is the most useful big integer class from standard java API, which allows low level arithmetic for several cryptographic algorithms is admitted. But the java specification request (JSR). This is presently being developed for first release, offers the J2ME devices with security element (SE).

Secure execution environment for security features, cryptographic operations and secure storage of sensitive data are supported by proposed security element. The big achievement for wireless and mobile application security is integrating APIs into J2ME devices without such support. The producer supplies device-specific APIs for cryptographic operations. But anyway several java MIDlets versions for different devices will demand API’s. Hence this can be applied to only small number of devices. MIDP version 2.0 does not allows the installation of user libraries because of problems with code signing this makes impossible to install a cryptographic library which is used by several MIDlets to decrease code size. This present scenario gives only single practical choice for application which needs access to cryptographic algorithms. This choice is to clump the actual application classes with needed cryptographic classes. The main setback of clumping is that if goes to comparatively large code sizes. This may result in
troubles with devices which impose a limit of the application size and thus suppress
application deployment.

2.1.1. Implementation issues

For MIDlets which admits public-key algorithms two important points have to be
considered for optimization. They are as follows

2.1.1.1. Performance

The potency of several performance optimizations is subjected to underlying
machine and JRE. But many important general rules have to be considered. Suggestions
for accomplishing better performance in java are available from many articles. Few of
these suggestions are mostly applied to public-key implementations than others.

Using the optimization switch of the java compiler is the convenient way for
optimization to reduce code size and increase performance such a compiler is used.

Application profiling is a significant step of optimization. Profiling describes
techniques which are mostly called and worth optimizing. Many tools are used for
profiling of java applications. Avoidance of string concatenation, replacement of short
library methods, in lining of short methods and object recycling are general rules that
have to be considered.

Implemented of public-key algorithms mainly deals look-up tables and multi-
word values. A certain overhead for index range verifying which is introduced by array
accesses, but java array is the best selection. If elements of array are indexed statically
then it is feasible to break the array in to several separate variables.

Lengthy initialization code many create other problems. these initializations
which can be displaced in to a static initialize block of a class which is loaded at start up.
In this way the implementation of the public-key algorithm can be reduced at the cost of a longer MIDlets load time. One more approach is to initialize them in a separate background thread. ECDSA signature generation is a case where this strategy can be useful to gain performance.

2.1.1.2. Code size

The initial approach to reduce code size is to get rid of unused classes. The developer can use MIDlets to his requirements if all of them are written by him. The code size can be made small in this way.

Almost all J2ME devices need additional resources and class files to be kept in a java archive file (JAR). Additional metadata and ZIP format are confirmed in a JAR file. Compression of clumped files is provided by JAR files. Compression must be used for MIDlets even though it is optional.

Obfuscation is other method for code size reduction. Mostly Obfuscation is to keep away de-compilation of java class files. Compressing constants, removing debug information and replacing method, field and class names with shorter ones is the function of obfuscator. A small byte-code can be achieved by these measures. The functionality of a program can be affected by obfuscation. Mainly the explicit dynamic loading of classes can be prevented by renaming of classes. Obfuscation tools which are configured to implement code modifications are recommended with careful selection.
3. PROPOSED RESEARCH

The security functions of most mobile devices cannot achieve the level we expect, due to the limited ability of computation of their processors. The data can be easily eavesdropped during the process of mobile communication.

Many good security protocols based on RSA operations have been applied to wired networks; however, these protocols often take much time to finish a computational process. They are therefore not suitable for mobile networks.

The limit of computational ability and the limit of memory size are two obstacles to overcome when we implement RSA-based security technologies on a mobile device. The RSA algorithm is probably the most popular and most well defined public-key cryptosystem.

It is widely used in digital signature and digital envelope, thanks to its privacy proof and authentication, both of which are the minimum requirements for the basic security functions of communication system. To ensure system security, RSA has to keep its key large enough. Yet, as the key becomes larger, the longer time is needed for the task of computation. The primary concern of applying the RSA-based security protocol to mobile networks is to implement RSA algorithm on mobile devices in a more efficient way. Same is true for elliptic curves.

3.1. Public-key Cryptography Challenges for Constrained Devices

Challenges that the devices face while working with the cryptographic algorithms are as follows:

- Very long operands (lengths of hundreds or thousands of bits)
- Lengthy algorithms
- Mathematical operations in non-conventional fields
- Sometimes requires considerable amounts of memory to store precomputation results

3.2. **Current Mobile and Wireless Devices Characteristics**

The following characteristics of the current mobile and wireless devices constrains other cryptographic algorithms from perform their task. But Elliptic Curve Cryptography could perform better even in this constrained environment.

- Small word size of the processor (today normally 16 or 32 bit)
- Limited hardware support for mathematical operations (e.g. no hardware multiplier)
- Limited memory for application code
- Limited working memory
- Limited energy supply

3.2. **Limitations of J2ME**

Limitations of J2ME which is used to coding and execute the cryptographic algorithms are as follows:

- Java applications (MIDlets) which require access to cryptographic algorithms (e.g. to sign data) have to be bundled with the according cryptographic library. This leads to rather large applications (50–60 Kb).
• The other problem is that the cryptographic functionality cannot be shared between MIDlets. That is, a lot of compatibility issues for MIDlets. The code is dependent upon machine. One code cannot be used on every machine/mobile. So the code has to be written and compiled for every machine.

3.4. Implementation issues for MIDlets

Issues that are raised or considered while implementing MIDlets are as follows:

• Most important: Maximal performance
• Always use the -O compile switch for java compilation and running.
• Use profiling to identify frequently called methods and optimize them
• Object recycling, avoid String concatenation, inline short methods, replace short library methods by local ones
• Break statically indexed arrays into separate variables
• Declare constants static final
• Do precalculations in static initialize block or background thread if application has to wait for input
• Reduce code size
• Remove unused classes from libraries (possible problems with reflection)
• Use compression for jar file
• Obfuscation (must avoid class renaming of dynamically loaded classes)

Keeping in view the above aspect, we used J2ME classes for elliptic curves and RSA, and we have implemented it on NOKIA Mobile.
The results are seen through logs. The Bandwidth, CPU consumption, data transfer, key length in bits, memory consumption are compared.

3.5. Implementation of RSA

The RSA class has one constructor, two methods and two utility functions. The constructor creates the p and q values, calculates phi and the modulus. The next step is to create the public key from a random number between 0 and $2^{16}$, and then generate the private key computing the modulus-inverse function, that is $\text{privateKey}^{-1} \mod \phi$ (using the extended Euclid algorithm implemented within the BigInteger class). This code must be inside a loop because we want to make sure the public key generated has modInverse. The other functions are encrypt(), which receives the String to encrypt and returns the BigInteger representing the encrypted text; and decrypt(), which receives this BigInteger and returns the decrypted String. The other two are just utility functions to get the values of the private and public keys.

3.6. Implementation of ECC

ECC is a little difficult to implement. It creates a point and initializes it using the random() function, then creates the elliptic curve using the previously made point as G, then proceeds with creating the private and public keys. The encryption/decryption algorithm, creates a ECPoint t and packs it into a BigInteger secret, which would be the text to encrypt in a real world application the text would have been converted into ECPoints using already defined algorithms, and then converted into BigIntegers each of those points. The encodeSecret() function is called passing the public key and the secret to
encrypt and it returns the pair of \((kG, Pm + kPb)\), which is also used by decodeSecret() in conjunction with the private key to decrypt the original BigInteger’s again, in a real-world application the program would then convert the BigInteger into the ECPoints, and then into the original String.
4. TESTING AND EVALUATION

Elliptic Curve Cryptographic algorithm along with RSA are executed on a mobile device and logs of the results are produced which helps us in comparing the performance, key size and other respective features. By looking at the results log we can compare and conclude that Elliptic Curve Cryptographic algorithm is efficient enough for constrained environment.

1. Less EEPROM and Shorter Transmission Times

The strength (difficulty) of the ECDLP algorithm means that strong security is achievable with proportionately smaller key and certificate sizes. The smaller key size in turn means that less EEPROM is required to store keys and certificates and that less data needs to be passed between the card and the application so that transmission times are shorter. [Certicom 1998]

2. Scalability

Smart card applications require stronger and stronger security (with longer keys), ECC can continue to provide the security with proportionately fewer additional system resources. This means that with ECC, smart cards are capable of providing higher levels of security without increasing their cost. [Certicom 1998]

3. No Coprocessor

The nature of the actual computations more specifically, ECC’s reduced processing times also contribute significantly to why ECC meets the smart card platform requirements so well. Other public-key systems involve so much computation that a dedicated hardware device known as a crypto coprocessor is required. The crypto coprocessors not only take up precious space they add about 20 to 30 percent to the cost.
of the chip, and about three to five dollars to the cost of each card. With ECC, the algorithm can be implemented in available ROM, so no additional hardware is required to perform strong, fast authentication. [Certicom 1998]

4. On Card Key Generation

As mentioned earlier, the private key in a public-key pair must be kept secret. For true no repudiation, the private key must be completely inaccessible to all other parties. In applications using the other types of public key systems currently in use, cards are personalized (keys are loaded or injected into the cards) in a secure environment to meet this requirement. Because of the complexity of the computation required, generating keys on the card is inefficient and typically impractical. [Certicom 1998]

With ECC, the time needed to generate a key pair is so short that even a device with the very limited computing power of a smart card can generate the key pair, provided a good random number generator is available. This means that the card personalization process can be streamlined for applications in which no repudiation is important. Summarizing, ECC key size advantages afford many benefits for smart cards, and the superior performance offered by Certicom’s ECC implementations make applications feasible in low end devices without dedicated crypto hardware.

4.1. Computational Overhead

In any public-key system, the private key is used for signing and decrypting and the public key is used for verifying and encrypting. The RSA crypto algorithm really only uses one type of calculation which is exponentiation. RSA takes a message to the power of an exponent. The nature of the exponent determines the speed of the operation. For
signing and decryption, the exponent (private key) is large, so the calculation is quite slow. For verification and encryption, however, the exponent (public key) can be quite small. Some implementers use a public key as small as 3 for very fast verification. Therefore, any analysis of RSA speed consists of slow times for signing and decrypting and much faster speeds for verification and encryption.

<table>
<thead>
<tr>
<th>Public Key</th>
<th>Private Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verifying</td>
<td>Signing</td>
</tr>
<tr>
<td>Encrypting</td>
<td>Decrypting</td>
</tr>
</tbody>
</table>

Table: Public and Private Key Operations

In both types of systems, considerable computational savings are achievable. In RSA, a short public exponent can be employed (although this does incur some security risks) to speed up signature verification and encryption. In both DSA and ECC, a large portion of the signature generation and encrypting transformations can be precomputed. Various special bases for the finite field $F_{2^m}$ can be used to accelerate the arithmetic involved in ECC operation.

As mentioned previously, cryptosensitive operations (those using the private key) must be performed in a secure environment because disclosure of the private key would compromise the system. Operations using the public key can often be performed in a nonsecure environment because key disclosure is not an issue.

Since the cryptosensitive operations (signing and decrypting) can be many times faster using ECC than using RSA, ECC is more appropriate for use in secure devices such as smart cards and wireless devices with constrained computational power. The noncrypto-sensitive (public key) operations can usually be performed in terminal or PC...
environments that typically have more computational power. Because the RSA crypto-
sensitive operations require more computational power, they are less suitable for use in
constrained environments, and as security (key size) requirements increase in the future,
the problem could become worse.

4.2. Experiment Conducted

When loaded, the MIDlet shows a list with two options, to run the RSA Algorithm
or the ECC Algorithm.

Figure 4.1. Application Start Screen
An exit option is also available to exit the MIDlet. Once in here the option chosen is “Clicked” and the corresponding form appears. Let’s say if wanted to run the RSA algorithm first, the “Run RSA Implementation” option is selected and the RSA form shows up.

Figure 4.2. Key Generation Using RSA
The form shows the size of the generated keys and their values. In this run the private key size is of 307 bytes, and its value. The form also shows the private key size, 5 bytes, and its value. The private key doesn’t have to be longer than that and it keeps being secure. The value of the elapsed time is in milliseconds for encryption strength of 1024 bits. The elapsed time value of RSA can be compared with the elapsed time value obtained in ECC implementation.

Figure 4.3. Key Generation Elapsed Time Using RSA
The Test command can be selected and some text is inserted to encrypt and decrypt. If the test option is hit the MIDlet shows a TextBox to insert the text. After hitting OK, the application shows the algorithm used, the original text inserted, the encrypted text and the same text after decryption. Time can also be seen that is taken to make the whole operation in milliseconds. The “Done” command can be selected to see the main menu again so as to select the next algorithm to compare results.

Figure 4.4. Encrypt/Decrypt Using RSA
The same steps are followed for ECC algorithm and the results are compared.

Figure 4.5. Key Generation using ECC
Do compare the elapsed times. ECC is way faster than RSA.

Figure 4.6. Encrypt/Decrypt using RSA and ECC for comparison

Compare Encrypt/Decrypt Time for ECC and RSA.

4.3. Experimental Results

The main focus of the project is to verify whether the Elliptical Curve Cryptography is better for constrained environments like mobile phones than RSA.
There are three different characteristics that were decided to compare between these two cryptographic algorithms, namely performance, security and space requirements.

1024-bit RSA key has roughly the same strength as a 160-bit ECC key, and a 2048-bit RSA has about the same strength as a 210-bit ECC key. Based on this comparison was made with the speed and memory space of 2048-bit RSA operations to 210-bit ECC, 1024-bit RSA operations to 160-bit ECC, 768-bit RSA operations to 132-bit ECC and 512-bit RSA operations to 106-bit ECC. And the results are shown in the tables below:

<table>
<thead>
<tr>
<th></th>
<th>Key Generation Time</th>
<th>Memory Requirement</th>
<th>Encrypt/Decrypt Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC (106 bits)</td>
<td>57 ms</td>
<td>108 bytes</td>
<td>11 ms</td>
</tr>
<tr>
<td>RSA (512 bits)</td>
<td>383 ms</td>
<td>157 bytes</td>
<td>77 ms</td>
</tr>
</tbody>
</table>

Table 1. Comparison of RSA(512 bits) and ECC(106 bits)

<table>
<thead>
<tr>
<th></th>
<th>Key Generation Time</th>
<th>Memory Requirement</th>
<th>Encrypt/Decrypt Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC (132)</td>
<td>98 ms</td>
<td>117 bytes</td>
<td>17 ms</td>
</tr>
<tr>
<td>RSA (768)</td>
<td>889 ms</td>
<td>236 bytes</td>
<td>160 ms</td>
</tr>
</tbody>
</table>

Table 2. Comparison of RSA(768 bits) and ECC(132 bits)

<table>
<thead>
<tr>
<th></th>
<th>Key Generation Time</th>
<th>Memory Requirement</th>
<th>Encrypt/Decrypt Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC (160)</td>
<td>108 ms</td>
<td>125 bytes</td>
<td>16 ms</td>
</tr>
<tr>
<td>RSA (1024)</td>
<td>2609 ms</td>
<td>313 bytes</td>
<td>388 ms</td>
</tr>
</tbody>
</table>

Table 3. Comparison of RSA(1024 bits) and ECC(160 bits)

<table>
<thead>
<tr>
<th></th>
<th>Key Generation Time</th>
<th>Memory Requirement</th>
<th>Encrypt/Decrypt Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECC (210)</td>
<td>121 ms</td>
<td>140 bytes</td>
<td>15 ms</td>
</tr>
<tr>
<td>RSA (2048)</td>
<td>18399 ms</td>
<td>621 bytes</td>
<td>1867 ms</td>
</tr>
</tbody>
</table>

Table 4. Comparison of RSA(2048 bits) and ECC(210 bits)
Graphs were drawn based on a series of results, where encryption strength is taken from 10bits to 4086bits.

Figure 4.7. Comparison of RSA and ECC Key Generation Time

Figure 4.8. Comparison of RSA and ECC Memory Requirement
Figure 4.9. Comparison of RSA and ECC Encrypt/Decrypt Time

Based on the graphs drawn above on different aspects like key generation time, memory space and encrypt/decrypt time comparing RSA and ECC, we can say that ECC is far better than RSA.
5. CONCLUSION

We can conclude that Elliptic curve is better than RSA on cellular phones by comparing the log results.

Recommendations for signatures on J2ME devices

- For signature verification alone, RSA is suited best (short public key)
- For signature generation alone, use Elliptic Curves
- For both, consider ratio of signature generations to verifications
- Public-key cryptography is feasible on present day J2ME devices -but not on all!
- High-end devices (e.g. Ericsson P900) already perform very well
- Still a lot of compatibility issues for MIDlets
- Better cryptographic APIs required (more efficient implementation, code sharing)

5.1. Fighting Tomorrow's Hackers [American Friends 2009]

“One of the themes of Dan Brown's The Da Vinci Code is the need to keep vital and sensitive information secure. Today, we take it for granted that most of our information is safe because it's encrypted. Every time we use a credit card, transfer money from our checking accounts -- or even chat on a cell phone -- our personal information is protected by a cryptographic system. “
But the development of quantum computers threatens to shatter the security of current cryptographic systems used by businesses and banks around the world.

“We need to develop a new encryption system now, before our current systems -- such as RSA -- becomes instantly obsolete with the advent of the first quantum computer,” says Prof. Oded Regev at Tel Aviv University's Blavatnik School of Computer Science. To accomplish that, Prof. Regev has proposed the first safe and efficient system believed to be secure against the massive computational power of quantum computers and backed by a mathematical proof of security.

5.2. Secure for Centuries

Prof. Regev stresses it is imperative that a new cryptographic system be developed and implemented as soon as possible. One reason is that current information, encrypted with RSA, could be retroactively hacked in the future, once quantum computers are available. That means that bank and other financial information, medical records, and even digital signatures could instantly become visible.

“You don't want this information to remain secure for just 5 or 10 years until quantum computers are built,” says Prof. Regev. “You want it to be safe for the next century. We need to develop alternatives to RSA now, before it's too late.”
REFERENCES AND BIBLIOGRAPHY


package eccm;
import eccm.rsa.*;
import eccm.ecc.*;
import eccm.ecc.ECCrypt.Pair;
import eccm.crypto.BigInteger;
import eccm.crypto.SecureRandom;

/** This class manages the encryption and decryption algorithms used in
the applet */
public class CryptoManager {
  /** Encryption strength */
  /* ECC - Change the value for N here */
  public static final int N = 106;
  /* RSA - Change the value for M here */
  public static final int M = 512;
  /** Instance of the RSA class used in this run */
  public static RSA rsa = null;
  /** Instance of the EC class used in this run */
  public static ECCrypt ec = null;
  /** Size of the private key */
  public static int prvKeySize = 0;
  /** Size of the public key */
  public static int pubKeySize = 0;
  /** String form of the private key */
  public static String prvKey = null;
  /** String form of the public key */
  public static String pubKey = null;
  /** The encrypted text is stored here */
  public static String encryptedText = null;
  /** The decrypted text is stored here */
  public static String decryptedText = null;
  /** Used when working with the ECC algorithm. Stores the text prior
to be
    * encrypted
    */
  public static String ecText = null;

  /** This method starts the process of generating the public and
private keys
    * using the given algorithm
    * @param usingRSA true if we are using the RSA algorithm */
  public static void startAlg(boolean usingRSA) {
    if (usingRSA) {
      rsa = new RSA();
      prvKey = rsa.getPrivateKeyStr();
      }}
pubKey = rsa.getPublicKeyStr();
prvKeySize = prvKey.length();
pubKeySize = pubKey.length();
} else {
    SecureRandom rnd = new SecureRandom();
    ECPoint cp = new ECPoint();
    cp.random(rnd);
    ec = new ECCrypt(cp);
    BigInteger prv = new BigInteger(N, rnd);
    BigInteger pub = ec.makePublicKey(prv);
    prvKey = prv.toString();
    pubKey = pub.toString();
    prvKeySize = prvKey.length();
    pubKeySize = pubKey.length();
}

/** This method starts the encrypting and decryption process of a
   given String.
   * @param usingRSA true if we are using the RSA algorithm
   * @param str the string to be encrypted
   */
public static void startEncDec(boolean usingRSA, String str) {
    if (usingRSA) {
        encryptedText = rsa.encrypt(str).toString();
        decryptedText = rsa.decrypt(new BigInteger(encryptedText));
    } else {
        SecureRandom rnd = new SecureRandom();
        ECPoint t = new ECPoint();
        t.random(rnd);
        BigInteger secret = t.pack();
        ecText = secret.toString();
        Pair p = ec.encodeSecret(new BigInteger(pubKey.getBytes()),
                                 secret);
        encryptedText = p.toString();
        BigInteger s = ec.decodeSecret(new BigInteger(prvKey.getBytes()), p);
        decryptedText = s.toString();
    }
}

private void initialize() {//GEN-END:|0-initialize|0|0-
    // write pre-initialize user code here
    //GEN-LINE:|0-initialize|1|0-postInitialize
    // write post-initialize user code here
    }//GEN-BEGIN:|0-initialize|2|
    //</editor-fold>//GEN-END:|0-initialize|2|
    }

    //<editor-fold defaultstate="collapsed" desc=" Generated Method:
    startMIDlet ">//GEN-BEGIN:|3-startMIDlet|0|3-preAction
    /**<
     * Performs an action assigned to the Mobile Device - MIDlet
     * Started point.
     */
public void startMIDlet() {//GEN-END:|3-startMIDlet|0|3-preAction
    // write pre-action user code here
    switchDisplayable(null, getList()); //GEN-LINE:|3-startMIDlet|1|3-postAction
    // write post-action user code here
}//GEN-BEGIN:|3-startMIDlet|2|
//</editor-fold>//GEN-END:|3-startMIDlet|2|

//editor-fold defaultstate="collapsed" desc=" Generated Method: resumeMIDlet ">//GEN-BEGIN:|4-resumeMIDlet|0|4-preAction
/**
* Performs an action assigned to the Mobile Device - MIDlet
* Resumed point.
*/
public void resumeMIDlet() {//GEN-END:|4-resumeMIDlet|0|4-preAction
    // write pre-action user code here
    //GEN-LINE:|4-resumeMIDlet|1|4-postAction
    // write post-action user code here
}//GEN-BEGIN:|4-resumeMIDlet|2|
//</editor-fold>//GEN-END:|4-resumeMIDlet|2|

//editor-fold defaultstate="collapsed" desc=" Generated Method: switchDisplayable ">//GEN-BEGIN:|5-switchDisplayable|0|5-preSwitch
/**
* Switches a current displayable in a display. The
* <code>display</code> instance is taken from <code>getDisplay</code>
* method. This method is used by all actions in the design for switching
* displayable.
* @param alert the Alert which is temporarily set to the display;
* if <code>null</code>, then <code>nextDisplayable</code> is set
* immediately
* @param nextDisplayable the Displayable to be set
*/
public void switchDisplayable(Alert alert, Displayable
nextDisplayable) {//GEN-END:|5-switchDisplayable|0|5-preSwitch
    // write pre-switch user code here
    Display display = getDisplay(); //GEN-BEGIN:|5-switchDisplayable|1|5-postSwitch
    if (alert == null) {
        display.setCurrent(nextDisplayable);
    } else {
        display.setCurrent(alert, nextDisplayable);
    } //GEN-END:|5-switchDisplayable|1|5-postSwitch
    // write post-switch user code here
}//GEN-BEGIN:|5-switchDisplayable|2|
//</editor-fold>//GEN-END:|5-switchDisplayable|2|

//editor-fold defaultstate="collapsed" desc=" Generated Getter: form ">//GEN-BEGIN:|14-getter|0|14-preInit
/**
* Returns an initialized instance of form component.
* @return the initialized component instance
*/
public Form getForm() {
    if (form == null) { //GEN-END:|14-getter|0|14-preInit
        // write pre-init user code here
        return null;
    } return form; //GEN-BEGIN:|14-getter|1|14-postInit
    // write post-init user code here
}
form = new Form("", new Item[]{getStringItem(),
getStringItem1(), getStringItem2(), getStringItem3(), getStringItem4(),
getSpacer2()}); //GEN-BEGIN:|14-getter|1|14-postInit
form.addCommand(getBackCommand());
form.addCommand(getItemCommand());
form.setCommandListener(this); //GEN-END :|14-getter|1|14-
postInit

// write post-init user code here
} //GEN-BEGIN:|14-getter|2|
return form;
} //GEN-END:|14-getter|2|

//<editor-fold defaultstate="collapsed" desc=" Generated Getter:
form1 ">
/**
 * Returns an initialized instance of form1 component.
 * @return the initialized component instance
 */
public Form getForm1() {
    if (form1 == null) {//GEN-END:|15-getter|0|15-preInit
        // write pre-init user code here
        form1 = new Form("", new Item[]{getStringItem5(),
getStringItem6(), getSpacer(), getStringItem7(), getStringItem8(),
getStringItem9(), getSpacer1()}); //GEN-BEGIN:|15-getter|1|15-postInit
form1.addCommand(getOkCommand1());
form1.setCommandListener(this); //GEN-END:|15-getter|1|15-
postInit
        // write post-init user code here
    } //GEN-BEGIN:|15-getter|2|
    return form1;
} //GEN-END:|15-getter|2|

//<editor-fold defaultstate="collapsed" desc=" Generated Method:
commandAction for Displayables ">
/**
 * Called by a system to indicated that a command has been invoked
 on a particular displayable.
 * @param command the Command that was invoked
 * @param displayable the Displayable where the command was invoked
 */
public void commandAction(Command command, Displayable displayable) {
    if (displayable == form) {//GEN-BEGIN:|7-commandAction|0|7-
preCommandAction
        if (command == backCommand) {//GEN-END:|7-
commandAction|1|21-preAction
            switchDisplayable(null, getList()); //GEN-LINE:|7-
commandAction|2|21-postAction
        } else if (command == itemCommand) {//GEN-LINE:|7-
commandAction|3|23-preAction
            if (usingRSA) {
                switchDisplayable(null, getTextBox());
            } else {
                Form t = getForm1();
            }
        }
    }
} //GEN-END:|7-commandAction|0|7-preCommandAction
switchDisplayable(null, t);
t.setTitle(usingRSA ? "Encrypt / Decrypt using RSA" : "Encrypt / Decrypt using ECC");
getStringItem5(). setText(usingRSA ? " RSA" : " ECC");
startEncryptionDecryptionEngine(null);
}

//GEN-LINE:|7-commandAction|4|23-postAction
        //GEN-BEGIN:|7-commandAction|5|44-preAction
    } else if (displayable == form1) {
        if (command == okCommand1) { //GEN-END:|7-commandAction|5|44-preAction
            switchDisplayable(null, getList()); //GEN-LINE:|7-commandAction|6|44-postAction
        } else if (command == okCommand2) { //GEN-BEGIN:|7-commandAction|7|18-preAction
            Form t = getForm();
            switchDisplayable(null, t);
            t.setTitle(usingRSA ? "RSA Implementation" : "EC Implementation");
            startKeysGeneration();
        } else if (command == exitCommand) { //GEN-LINE:|7-commandAction|9|25-preAction
            exitMIDlet(); //GEN-LINE:|7-commandAction|10|25-postAction
        } else if (displayable == textBox) { //GEN-BEGIN:|7-commandAction|11|42-preAction
            Form t = getForm1();
            switchDisplayable(null, t);
            t.setTitle(usingRSA ? "Encrypt / Decrypt using RSA" : "Encrypt / Decrypt using ECC");
            getStringItem5(). setText(usingRSA ? " RSA" : " ECC");
            startEncryptionDecryptionEngine(getTextBox().getString());
            //GEN-LINE:|7-commandAction|12|42-postAction
        } //GEN-BEGIN:|7-commandAction|13|7-postCommandAction
        //GEN-END:|7-commandAction|13|7-postCommandAction
    } //GEN-BEGIN:|7-commandAction|14|
    //GEN-END:|7-commandAction|14|

    //<editor-fold defaultstate="collapsed" desc=" Generated Getter: list ">
    public List getList() {
        if (list == null) { //GEN-END:|16-getter|0|16-preInit
            // write pre-init user code here
            list = new List("Elliptic Curves & RSA Implementation", Choice.IMPLICIT); //GEN-BEGIN:|16-getter|1|16-postInit
            list.append("Run RSA Implementation", null);
            return list;
        }
        return list;
    }
    //</editor-fold>
}
list.append("Run ECC Implementation", null);
list.addCommand(getExitCommand());
list.setCommandListener(this);
list.setSelectedFlags(new boolean[] { false, false });
// write post-init user code here
//GEN-BEGIN:16-getter|2|
return list;
}