**Documentation**

**Setting Up Java ME**

To develop Java ME applications you must set up the Java ME SDK and configure it with your IDE. Oracle provides plugins for Eclipse and NetBeans. Projects developed for Java ME do not run in the same way as normal Java projects. Java ME projects require an emulator. Luckily, a number of emulators are provided with the Java ME SDK. The link below has a comprehensive guide on setting up Java ME SDK:

- http://docs.oracle.com/javame/8.1/sdk-dev-guide/install.htm

**NOTE:** There is a bug that may cause serious issues when trying to create projects with the Java ME SDK. After setup, if the device manager does not appear in the bottom right toolbar on your desktop, you may need to add some missing configuration files. Follow this post for the fix:

- http://www.threadstuck.com/question/java-emulator-not-starting-upgrading-java-8/

**Setting up the Galileo**

You will want to set up your Galileo before development. Since Java ME SDK is only supported on Windows you will have to do this on a Windows machine. Here is a comprehensive guide on how to do it:

- https://software.intel.com/en-us/get-started-galileo-windows

**Setting up Java ME on the Galileo**

You will have to install the Java ME embedded software onto the Galileo. Here is a comprehensive guide to do this:

- https://docs.oracle.com/javame/8.3/get-started-galileo/installing-java-me-embedded-software-intel-galileo-gen2-board.htm

**Creating a Java ME Project**

Once you have set up all your tools to work with Java ME and the Galileo you will want to start a Java ME project

1. In your IDE, go to File and select New
2. Next, select Java ME Project
3. Now name your project and select a device configuration
    a. It's important to note that device configurations are required and provided by the Java ME SDK. If you do not see any device configurations something may have gone wrong in the set up so please see the note in the section concerning the potential bug.

**Exporting a Project To the Galileo**

1. Save your project
2. Go to 'File' and select 'Export'

3. Select your project and a destination directory
4. Once the project has completed exporting you will see it in the chosen directory
5. Connect to the Galileo using the Galileos IP address with sftp to transfer the file
    a. Ex: sftp 10.0.0.21
6. Navigate to the directory in which your project has been exported
    a. To navigate on your local machine prepend an a **lowercase** "L" (stands for local) to standard bash commands
        i. Ex: 'lcd' = 'cd'
        ii. Ex: 'lls' = 'ls'
    b. To navigate on the remote host (Galileo) use standard bash commands
7. Navigate to your target directory on the Galileo
8. Transfer the .jar file from your project to the Galileo
    a. Ex: 'put *filename*.jar'

**Running a JAD**

Once your project JAR has been transferred to the Galileo you will want to execute it.

1. Connect to your Galileo using ssh
2. Navigate to the 'bin' directory
3. Run the bash script 'installMidlet.sh', with the path to your JAR file as the parameter
    a. Ex. "./installMidlet.sh *file/path/to/JAR/fileName.jar'*
4. You should now be able to see your project installed on the device by running the 'listMidlets.sh' script
    a. Ex: './listMidlets'
5. If successful you will see your project in the list of MIDlets. Note the Suite number
6. Next, run the script 'runSuite.sh' with the suite number of your project as the parameter
    a. Ex: './runSuite.sh *3'*
7. Your project will execute on the Galileo. Note you may see some errors that may not be related to your project, these are typically harmless and your project should still run. Read the errors carefully to make sure they are not associated with your project.

**Removing Projects from the Galileo**

1. Connect to your Galileo
2. Navigate to the 'bin' directory
3. Determine the suite number of the project you wish to remove
    a. Use the './listMidlets.sh' command
4. Run the 'removeSuite.sh' or 'removeMidlet.sh' command with the number associated to your project
    a. Ex: '*./removeMidlet.sh 3'*
5. Navigate to the directory containing the original JAR
6. Remove it by using the 'rm' command with your jar as the argument
    a. Ex: *'rm myJar.jar'*

**Socket Programming**

If you wish to integrate networking into your Java ME project you must first add the required permissions to your Application descriptor. Instructions for adding permissions to your Application Descriptor can be found in the “Permissions” section of this paper. It is important to note that when adding permissions for socket programming, you must specify the particular resource you want to access. For creating a server you simply need to supply the port number you wish to listen on, but for a client you also need the host name. For example, some socket permissions might look like this:

1. For a server: *javax.microedition.io.SocketProtocolPermission “socket://:4040”*
2. For a client: *javax.microedition.io.SocketProtocolPermission “socket://127.0.0.1:4040”*

When adding the permission, you must specify the host and the port. However, you may also replace a hostname or port number with an asterisk to allow permission on any port or hostname. For example:

1. For a server: *javax.microedition.io.SocketProtocolPermission “socket://:*”*
2. For a client: *javax.microedition.io.SocketProtocolPermission “socket://*:4040”*
3. For a client: *javax.microedition.io.SocketProtocolPermission “socket://*:*”*

Thus, if you wish to change the ports used in your program, you must also change the ports in your permissions. Additional documentation regarding Java ME security permissions and protected resources can be found here:
https://docs.oracle.com/javame/8.2/me-dev-guide/security.htm

**Class:** KeyGen

**Functions:**

| | |
|---|---|
| **Name** | ECGenerateKeyPair() |
| **Arguments** | None |
| **Return** | Returns AsymmetricCipherKeyPair object. |
| **Description** | Generates an asymmetric key pair using elliptic curve cryptography. |

| | |
|---|---|
| **Name** | ECgenerateSecret(AsymmetricKeyParameter, AsymmetricKeyParameter) |
| **Arguments** | Takes two AsymmetricKeyParameters, i.e. one public key and one private key. |
| **Return** | Returns a secret symmetric key in form of a  byte array |
| **Description** | Generates a shared secret using ECDH |

| | |
|---|---|
| **Name** | getPublicKey(AsymmetricKeyParameter) |
| **Arguments** | Takes a single AsymmetricKeyParameter |
| **Return** | Returns the public key as a byte array |
| **Description** | Converts an AsymmetricKeyParameter to a byte array |

| | |
|---|---|
| **Name** | createPublicParamFromKey(byte [ ]) |
| **Arguments** | Takes public key as a byte array |
| **Return** | Public key as AsymmetricKeyParameter |
| **Description** | Converts a byte array into AsymmetricKeyParameter |

| | |
|---|---|
| **Name** | deriveSymmetricKey(AsymmetricKeyParameter, AsymmetricKeyParameter, byte [ ]) |
| **Arguments** | Two AsymmetricKeyParameters corresponding to the public keys of both parties associated with the shared secret. A shared secret in the form of a byte array. |
| **Return** | A newly derived symmetric key |
| **Description** | Takes public keys from two parties along with a shared secret and generates a shared symmetric key. By default produces a 128 bit symmetric key. |

| | |
|---|---|
| **Name** | deriveSymmetricKey(AsymmetricKeyParameter, |

| | |
|---|---|
| | AsymmetricKeyParameter, byte [ ], int, String) |
| **Arguments** | Two AsymmetricKeyParameters corresponding to the public keys of both parties associated with the shared secret. A shared secret in the form of a byte array. Additionally takes an int as the desired output size, and a String indicating the hash algorithm. |
| **Return** | A newly derived symmetric key |
| **Description** | Takes public keys from two parties along with a shared secret and generates a shared symmetric key. |

**Class:** Authentication

**Functions:**

| | |
|---|---|
| **Name** | createHMAC128(byte [ ], byte [ ]) |
| **Arguments** | Takes data as a  byte array and a key as a byte array |
| **Return** | Returns a message authentication code as a byte array |
| **Description** | This function takes a message, typically ciphertext, and a symmetric key to create a message authentication code used for authentication and integrity. By default produces a 128 bit MAC using MD5 |

| | |
|---|---|
| **Name** | creatHMAC(GeneralDigest, byte [ ], byte [ ]) |
| **Arguments** | Takes the desired digest to be used, data in the form of a byte array and a key in the form of a byte array |
| **Return** | Returns a message authentication code as a byte array |
| **Description** | This function takes a message, typically ciphertext, and a symmetric key to create a message authentication code used for authentication and integrity |

| | |
|---|---|
| **Name** | createaHMAC(GeneralDigest, byte [ ], byte [ ]) |
| **Arguments** | Takes a digest(such as MD5), data as a byte array, and key as a byte array |
| **Return** | Returns a MAC with size specific to the chosen digest |
| **Description** | Creates a MAC given data, key, and a specific hash algorithm |

| | |
|---|---|
| **Name** | verfiyHMAC(GeneralDigest, byte [ ], byte [ ], byte[ ]) |
| **Arguments** | Takes a digest(same digest used to create the original MAC), the MAC to be verified as a byte array, data as a byte array, and key as a byte array |
| **Return** | Returns a boolean, if the MAC matches returns true, otherwise false |
| **Description** | Recomputes the MAC and compares to the received MAC. The digest used must match the digest used to create the original MAC |

| | |
|---|---|
| **Name** | encryptAndMac(byte [ ], byte [ ]) |
| **Arguments** | Takes data as a byte array and a key as a byte array |
| **Return** | Returns a byte array with ciphertext and a MAC |
| **Description** | Uses AES128 to encrypt the provided data then uses MD5 to created a 128 bit MAC. |

**Class:** Encryption

| | |
|---|---|
| **Name** | encrypt(byte[], int, int, int, int) |
| **Arguments** | Takes plaintext and key as a single byte array. Takes key length, key offset, plaintext length, and plaintext offset as integers |
| **Return** | Returns ciphertext as a byte array |
| **Description** | Encrypts plaintext using AES by default but the algorithm can be modified as a variable in the Encryption class |

| **Name** | encrypt(byte[ ], byte [ ]) |
|---|---|
| **Arguments** | Takes data as a byte array, and a key as a byte array |
| **Return** | Returns encrypted cipher text as a byte array |
| **Description** | Encrypts data with the given key using AES |

| **Name** | decrypt(byte [ ], int, int, int, int) |
|---|---|
| **Arguments** | Takes a byte array containing ciphertext and a key. Takes the key length, key offset, ciphertext length, and ciphertext offset as ints |
| **Return** | Returns plaintext as a byte array |
| **Description** | Decrypts ciphertext with the provided key and text and returns plain text. Algorithm used is modifiable within the Encryption class |

| **Name** | decrypt(byte [ ], byte [ ] ) |
|---|---|
| **Arguments** | Takes ciphertext as a byte array and a key as a byte array |
| **Return** | Returns plaintext a a byte array |
| **Description** | Simply takes encrypted message and decrypts it with the given key |

| **Name** | gcmEncrypt(byte[], byte[], byte[], byte[]) |
|---|---|
| **Arguments** | Takes a key, plaintext to be encrypted, additional non encrypted data, and a random nonce, as byte arrays |
| **Return** | Returns a byte array with ciphertext and an authentication tag |
| **Description** | Encrypts plaintext using Galois counter mode and returns a byte array with both ciphertext and an authentication tag. The nonce should be randomized at each use |

| **Name** | gcmDecrypt(byte[], byte[], byte[], byte[]) |
|---|---|
| **Arguments** | Takes a key, ciphertext to be decrypted, additional non encrypted data that was used in the authentication tag generation, and the nonce used to encrypt the data, as byte arrays |
| **Return** | Returns a byte array with plaintext |

| | |
|---|---|
| **Description** | Decrypts and authenticates the data using the ciphertext, additional data, key and nonce. The additional data and nonce must be the same as those used to encrypt the data and generate the authentication tag |

| | |
|---|---|
| **Name** | generateNonce() |
| **Arguments** | None |
| **Return** | Returns a secure nonce as a byte array |
| **Description** | Generates a secure random nonce |