Module Four Algorithm Ciphers Assignment Guidelines and Rubric

As computing becomes pervasive, embedded systems are deployed in a wide range of domains, including industrial systems, critical Infrastructures, private and public spaces as well as portable and wearable applications. An integral part of the functionality of these systems is the storage, access and transmission of private, sensitive or even critical information. Therefore, the conﬁdentiality and integrity of the resourcesand services of said devices constitutes a prominent issue that must be considered during their design.

There is a variety of cryptographic mechanisms which can be used to safeguard the conﬁdentiality and integrity of stored and transmitted information. In the context of embedded systems, however, the problem at hand is exacerbated by the resource-constrained nature of the devices, in conjunction with the persistent need for smaller size and lower production costs. This paper provides a comparative analysis of lightweight cryptographic algorithms applicable to such devices, presenting recent advances in the ﬁeld for symmetric and asymmetric algorithms as well as hash functions.

A classiﬁcation and evaluation of the schemes is also provided, utilizing relevant metrics in order to assesstheir suitability for various types of embedded systems Lightweight and ultra-lightweight ciphers usually oﬀer 80 to 128 bit security. 80 bit security is considered adequate for constrained devices, like 4-bit micro-controllers and RFID tags, while 128 bits is typical for mainstream applications. For one way authentication, 64 to 80 bit security would suﬃce.

Three main approaches are followed in implementing lightweight ciphers. In the ﬁrst case, researchers try to improve the performance of well-known and well-studied ciphers such as AES and DES. A state of the art AES hardware implementation uses 2400 GE and is used as a benchmark for newer ciphers. In the second case, re-searchers design and implement new ciphers, speciﬁc for this domain.

The absence of decryption is another factor that can reduce the requirements of such ciphers, especially for ultra-lightweight cryptography. Hummingbird-2 is a combination of cipher and protocol and adopts this strategy. This approach is suitable for devices that need only one way authentication.

Before transmitting any data, the sender encrypts its message, and the receiver must in turn decrypt the message before processing it. The encryption and decryption is accomplished through a method called "public key encryption."

In order for public key encryption to provide secure communication, one more more of the communicating parties must have some way of proving to the other that they are, in fact, who they claim to be. SSL provides this proof by requiring that one or more of the parties present a digital certificate into the initial negotiation of the connection, prior to the transmission of any encrypted data. This process is called "handshaking."

The keys Tomcat will use for SSL transactions are stored in a password-protected file called, creatively, the "keystore." The first step to enabling SSL on your server is to create and edit this file. You can create this file in one of two ways - by importing an existing key into the keystore, or by creating an entirely new key.

It's time to create a file called the Certificate Signing Request, or CSR, which will be used by the Certificate Authority of your choice to generate the Certificate SSL will present to other parties during the handshake.

SSL verifies the authenticity of a site's certificate by using something called a "chain of trust," which basically means that during the handshake, SSL initiates an additional handshake with the Certificate Authority specified in your site's certificate, to verify that you haven't simply made up your own CA.

In order to "anchor" your certificate's chain of trust, you have to download an additional certificate, called a "Root Certificate," from your CA, and then import both this certificate and your site's new certificate into your keystore. Your CA should provide information about obtaining a Root Certificate on their website.

To import the Root Certificate -

keytool -import -alias root -keystore [path/to/your/keystore] -trustcacerts -file [path/to/the/root\_certificate]

To import your new Certificate -

keytool -import -alias [youralias] -keystore [path/to/your/keystore] -file [path/to/your\_keystore]

  Enabling SSL in Tomcat's server.xml file causes all files to be run both as secure and insecure pages, which can cause unnecessary server load. You can choose which applications offer SSL connections on a per-application basis by adding the following <security-constraint> element to the application's WEB-INF/web.xml file:

<security-constraint>

<web-resource-collection>

<web-resource-name>YourAppsName</web-resource-name>

<url-pattern>/\*</url-pattern>

</web-resource-collection>

<user-data-constraint>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>