Chapter 4 Solution Idea and implementation

Overview

Here I will be discussing the implementation and approach of the different algorithms, languages and language specific modules I will be using in my solution. I will explain in depth on how the algorithms work as most of the algorithms themselves are done through the java modules, a clear understanding of the encryption methods is required to justify their use and any limitations.

Encryption Algorithms

Encryption algorithms can be split into two kinds, asymmetric and symmetric ciphers, considering the time consideration I will be mostly looking at symmetric ciphers due to their quicker computation times, however I will included an RSA example in the work due to how secure it is and to use as a baseline comparison for other algorithms.

Block Cipher AES

One algorithm I looked at is the widely used block cipher AES (advanced Encryption Standard) which replaced The Data encryption Standard (DES) used by the American government. It uses a fixed block size of 128 bits and a key size of 128,192 or 256 bits. AES is a great block cipher to use in the comparison due to its efficient implementation in both software and hardware. The key size used in AES dictates the number of rounds that the plaintext goes through. Step 1 is a key expansion that derives a round key from the cipher key. AES requires a separate 128-bit round key block for each round plus a final one. Next is an initial round key addition which combines (using bitwise XOR) each byte of the states with a byte from the round key. For the next rounds (depending on key size), run as follows.

1. a non-linear substitution step where each byte is replaced with another according to a lookup table.

2. a transposition step where the last three rows of the stat are shifted cyclically a certain number of steps

3. a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.

4. adding the round key where the subkey is combined with the state

Then there is a final round which involves a the same as the previous rounds, but with step 3 the linear mixing operation removed.

In terms of security, despite a lot of research into AES there are no practical attacks on AES with some side channel attacks on the implementation of AES which are easy to avoid.

Stream Cipher ChaCha20

Salsa20 and ChaCha20 are a close family of stream ciphers with ChaCha variant aiming to increase the diffusion per round and reduce the number of rounds needed for security. The Salsa20 encryption function is a long chain of three operations on 32-bit words.

* a 32-bit addition, producing the sum a+b mod 2^32 of two 32-bit words a, b.
* a 32-bit exclusive-or, producing the xor a , b of two 32-bit words a, b; and
* constant-distance 32-bit rotation, producing the rotation a <<< b of a 32-bit word a by b bits to the left where b is a constant.

Salsa20 expands a 256-bit key and a 64-bit nonce into a 2^70-byte stream. Encryption is achieved by xor’ing the b-byte plaintext with the first b bytes of the stream and discarding the rest. For decryption it follows the same process. The stream is generated in 64-byte (512-bit) blocks. Each block is an independent hash of the key, the nonce and a 64-bit block number; there is no chaining from one block to the next. This means the output block can therefore be accessed randomly, and any number of blocks can be computed in parallel.

Chacha follows the same base design principles as Salsa20 but has an increased diffusion per round allowing a smaller minimum number of secure rounds for ChaCHa. The extra diffusion does not come at the expense of extra operations as ChaCha round has 16 additions, 16 xors and 16 constant distance rotations of 32-bit words, just like a Slasa20 round. The main speed and diffusion differences come from the quarter round where each word is updated twice and the word matrix where it is built with attacked controlled inputs words on the bottom and ChaCha sweeps the matric through rows in the same order every round.

Java Modules

I will be using Java as the implementation language for its wide range of importable modules such as the java crypto packages for cryptographic operations for AES and ChaCha. Java crypto specific modules that are of most help are the Cipher and key generator class. For the RSA algorithm that was used more as a comparison tool to show the difference to these quicker encryption methods the java security module was used for the secure random feature.

The Javax.crypto.cipher class provides the functionality of a cryptographic cipher. In order to create a Cipher object, the class calls the Cipher's getInstance() method, and passes the name of the requested transformation to it. A transformation is a string that describes the operation (or set of operations) to be performed. It includes the name of a cryptographic algorithm (e.g., AES), and may be followed by a feedback mode and padding scheme. In my case it includes the mode type of AES and a padding scheme for extra security.



The cipher is then initialised with the mode (either encrypt or decrypt), the secret key and an initial vector.



The final function of the cipher class used is the final encryption of the plain text using the doFinal() function which will encrypt or decrypt a byte form input so if you are sending a string it must be converted into a bytes first.



The other crypto modules used are the Key generator class which is a re-usable key generation object. It can only be used for symmetric secret key generation. There are two ways to generate a key: in an algorithm-independent manner, and in an algorithm-specific manner. The only difference between the two is the initialization of the object.

GUI

For the GUI I have used the java swing and awt modules which are common graphically interfaces with options to select the Encryption method and options to select the message type with the average time taken to complete the encryption and decryption times to be sent. The network type such as 4G or wireless should be able to be selected.

\*\* screenshots of GUI plus further discussion

Network Class

For testing the algorithms, I needed a testing class to simulate a test network that the GUI could run to get the results. It needed a sender object and a receiver object that could either work as a simple client-server model as a simple peer-peer communication or work as a publisher-subscriber model for the GOOSE and SMV communication types. The Network would then act as the data stream maintaining subscriber lists in the case of the subscribe topology and forward the messages to designated locations. It would also handle the setup for the algorithms such as the secure transmission of the secret keys for the symmetric communication.

Nodes

For the publisher subscribers you can set up two separate classes sender and receiver as they don’t have to fill both roles on the network though this means there is limited functionality to handle peer-peer communication between the “IEDS” on this test network.