Symmetric Encryption Using Diffie-Hellman Key Agreement

CA547 Cryptography and Security Protocols
David Kernan
59597883
CASE4
Lecturer - Geoff Hamilton

Declaration

I the undersigned declare that the project material, which I now submit, is my own work. Any assistance received by way of borrowing from the work of others has been cited and acknowledged within the work. I make this declaration in the knowledge that a breach of the rules pertaining to project submission may carry serious consequences.
David Kernan

Assignment 1

In this assignment, we were tasked with performing the Symmetric Encryption of a message using a block cipher (AES in this case). The assignment spec gave us the needed publiccally shared keys.
The Generator Key = g
The Shared Prime = p

I am not given private key a. That is kept secret, however I am given public key A which is calculated using a
Public Key A = $g^a (mod\ p)$

g is raised to the power of *a* and then has the modulo operation applied to it. This operation becomes a problem when the numbers are on a large scale (such as here), it requires too many operations. I would have to multiple *g* by itself *a* times. Which ends up becoming *a* operations. We are able to reduce the magnitude of operations drastically by using several methods. In my mod_pow() implementation I use Right-to-Left Binary Exponentiation[1].
In the *while* loop, we check if the exponent is still a positive number. We keep working until it's not.
We then discover if the exponent is odd or even. If odd, we then set the *result* variable to equal $result * base\ (mod\ modulus)$
then subsequently shifting the exponent to the right one bit before finally updating the base to equal $base * base\ (mod\ modulus)$

[1] http://primes.utm.edu/glossary/xpage/BinaryExponentiation.html

I created my own random private key, b. It's 1023 bit long key. With this I was able to create my Public Key, this is the key that I will be able to share over a insecured connection.

Public Key B = $g^b (mod\ p)$

I then calculated a shared key, this is the key that will be calculated privately and will be the same result as what the other host has calculated.

Shared Key s = $A^b (mod\ p)$

My message will be 128 zero'd bits. I initiate this by creating a byte array that will contain 16 zero'd bytes. $16 * 8 = 128$

Once the shared key has been calculated, it will be 1024 bits long. AES is a block cipher, with each block is 128 bits, and AES keys being a maximum of 256 bits. So we must shorten the key so it can be used as an AES key.
SHA-256 is a hashing algorithm. Hashing a string will give a specific output to any specific input. Running the 1024 bit key into the hashing algorithm will output a 256 bit hash which is a AES key size.

Once I have the key in the correct size, I send it to the encrypt() function. This function creates and initiates my AES Cipher Object. I choose ECB mode and request no padding (the key is the correct size and requires no padding to fit).
I set it to encrypt, pass the shared key into the Cipher and return the ciphertext using the toHex() function.

Likewise, I can go through the same steps with the decrypt function. Pass the shared AES key with the ciphertext string. It initiates the AES Cipher, however using decrypt mode and returns to me a byte array containing 128 zero'd bits.

I make extensive use of the Java librarys for this assignment.
Javax.crypto has the Cipher and SecretKey Class while Java.Math holds the BigInteger class. Using BigInteger is needed as normal integers would not be able to hold the integers used and would overflow.