Tyler Lorella and Maxfield England
TCSS 487
3/5/2020

Cryptography Project: KMACXOF256 and Schnorr/ECHDIES

Our project uses console/command line input and output, able to read from files or parse user input for encryption services. Instructions are given on a line-by-line basis. The user can generate a hash, MAC, public and private keys, a digital signature, or a cryptogram symmetrically using only a private key or asymmetrically, along with the option to decrypt such a cryptogram. Each option is presented with a numeric option, and the user also has the option to receive their output via console output or to files to a given directory.

Part 1: We have written a Java-based implementation of SHA-3, adapted mostly from the C implementation by Markku-Juhani Saarinen (https://github.com/mjosaarinen/tiny_sha3/blob/master/sha3.c). Our solution is adapted based on the NIST SP 800-185 specifications of the derived functions of cSHAKE256 and KMACXOF256, and the helper methods described there. For the functionality of part one, to compute a hash of file or user input, we simply need an instance of our KMACXOF256 class, using the data, retrieving 512 bits.

Part 2: To encrypt the file, we generate a 512-bit generated number, and use instances of KMACXOF256 using the given passcode, random number and data to generate a cipher mask and a message authentication code. The random number, ciphertext formed from the mask xored against the data, and the message authentication code all is output as the encrypted data.

Part 3: For the elliptic key pair, we implemented in Java a Point class for the specifications for elliptic curve modular arithmetic, with the given constructors and the given modular square root code. The key pair is generated by another instance of KMACXOF256, using a passphrase and the public point G.

Part 4: Following the specifications, and using our Point class, we use multiple calls of KMACXOF256, as well as a random-seeded BigInteger, to create a cryptogram much like our symmetric cryptogram. Like with symmetric decryption, asymmetric decryption was a simple change.

Part 5: Using the elliptic curve global parameters and our previous solutions, we implemented digital signatures with the previously written public key file.