Practical Project - Cryptographic Library & App - Part 1

Names: Elijah Immer, Khalid Rashid, Grant Koenig

Objective Implement (in Java) a library and an app for asymmetric encryption and digital signatures at the 128-bit security level.

Algorithms

1. SHA-3 hash and SHAKE extendable output functions.
2. ECDHIES encryption and Schnorr signatures.

We set up the project by making two classes, the SHA3SHAKE class which have the prebuilt foundation to build the algorithm as well as a main class. We started out with the absorb methods, squeeze methods, and the digest methods. Absorb methods XORs bytes into the sponge, tracks how many bytes filled in the current rate and calls Keccak to mix it, then it applies the SHA-3 padding and calls Keccak again. The squeeze method extract the length of the bytes from the sponge and copies bytes from the state into "out". The digest method squeezes "digest_length" bytes into the out buffer and returns it. The init method initialized and allocates the buffer and clears it, setting the "digest_length" from bits to bytes for the security size of the sponge.

After these, we decided to work on the steps of the Keccak function θ (Theta), $\rho + \pi$(Rho+Pi), χ (Chi), and ι (Iota). What we did first was try to translate it differently from reference C code but it failed badly so we ended up just using the code and directly translating it to Java and that was successful. We set the rounds to our constant in the instance field "KECCAK_ROUNDS" for the desired 24 rounds. We needed to implement a helper method used in Rho and Theta, as well as the rotate offset for Rho and the Pi lane permutations. In Main, it parses the security bits size (128,224,256,384,512), reads contents of the file into a byte array then goes through our sponge and prints the hashed result. By testing the code, we were comparing the outputs from our code to the outputs of the C code. Eventually we got the same output as the C code.