TCSS 487 Project Part 1 Report
Douglas Johnston

Program Instructions

Command Line Mode:
To use this mode, run the program without any arguments. Then follow the instructions printed to the console.

File Mode:
To use this mode, run the program with an argument.

The order to enter arguments: 1.{filepath with name} 2. {flag} 3. {passphrase}

These are the acceptable flags: -H, -T, -S

If only a filename is provided, the program will default to -H

-H:
Computes a plain cryptographic hash of the given file and writes it to output.txt
A passphrase is not required for this flag to work

-T
Computes an authentication tag of the given file and writes it to output.txt
A passphrase is REQUIRED to be entered after the flag for this to work

-S
Does symmetric encryption and decryption of the given file and writes it to output.txt
A passphrase is REQUIRED to be entered after the flag for this to work

Solution Description

SHAKE.java
This is the class that models the sponge for sha3. It was made with a lot of inspiration from the provided example (tiny_sha).

absorb(theData, theLength)
This function absorbs bits into the state by XORing each bit of theData onto the state, then passes the state into the KECCAK function as per the SHAKE design.

changeMode()
This function should be called once all the data has been absorbed. It provides extensible output functionality and prepares the sponge for squeezing.

squeeze(theData, theLength)
This function squeezes bytes from the state into theData then passes the state into the KECCAK function as per the SHAKE design.

Sha_3.java
This class is the core of the solution. It utilizes the SHAKE class to implement cSHAKE256 and KMACXOF256. A lot of inspiration was taken from tiny_sha for many functions in this class.

bytePad, encodeString, leftEncode, and rightEncode were all derived from the NIST Special Publication, provided in the project description.

The hash() functions compute a simple cryptographic hash of the passed string or file using these parameters provided in the project description: h <- KMACXOF256(“”,m, 512, “D”)

The authenticationTag() functions compute an authentication tag of the passed string or file under the passed passphrase by using these parameters provided in the project description:
t <- KMACXOF256(pw, m, 512, “T”)

The encrypt() functions encrypt the passed string or file symmetrically under the passed passphrase by using this process provided in the project description:
z <- Random(512)
(ke || ka)
KMACXOF256(z || pw, “”, 1024, “S”)
C <- KMACXOF256(ke, “”, |m|, “SKE”) ^ m
t <- KMACXOF256(ka,m, 512, “SKA”)
symmetric cryptogram:(z, c, t)

The decrypt() function decrypts the passed SymmetricCryptogram under the passed passphrase by using this process provided in the project description:
(ke || ka) <- KMACXOF256(z || pw, “”, 1024, “S”)
m <- KMACXOF256(ke, “”, |c|, “SKE”) ^ c
t` <- KMACXOF256(ka, m, 512, “SKA”)
accept if, and only if, t` = t

SymmetricCryptogram.java
An object that represents a symmetric cryptogram(z, c, t) with getters for z, c, and t