1: Hash Function and HMAC

Implement the hash function described in the file “hash_spec”, then HMAC based on this hash function. A driver is provided to test the implementation.

2: Stream Cipher

Implement the stream cipher [Trivium](), a cipher in the [eSTREAM]() portfolio. The details are found in the [specification](). [Test vectors]() will prove helpful.

.

3: Finite Field

Implement the finite field GF(2^1279) using GF(2)[x]/(f(x)) where irreducible polynomial f(x)=x^1279 + x^319 + x^127 + x^63 + 1.

- The field operations are addition (trivially XOR) and multiplication; the latter calculates a*b mod f(x). You can find simple algorithms in Section 3 [here]()([non-springer mirror]()).
- You will also implement a fast exponentation method that calculates b^n mod f(x) for some integer n using O(lg n) steps. Find numerous algorithms in the[Handbook of Applied Crypto]() 14.6.

4: Public Key Encryption

Using the preceeding finite field and multiplicative generator g=x (0x2), implement the ElGamal cryptosystem ([HAC](), Chapter 8.4; focus on 8.4.2 and algs 8.25/8.26 and note 8.27). First add a field inversion routine to your finite field implementation; use the extended Euclidean algorithm (Alg. 8 of the previous reference paper, or HAC Chapter 14).

Your implementation can assume messages with minimum length 1 byte and maximum length 158 bytes. Message padding works as follows. Denote the length of the message by x. Append 159-x bytes with a value of 159-x. Then append a zero byte. This results in 160 bytes.

To map the padded message m to a field element e, the first byte of m is the first byte (least significant, lower degree) of e (and the last is the last). (This is very natural in C but awkward in Java; you will most likely have to reverse the padded message.)

To pack the field elements to bytes during encryption, the mapping from field elements to bytes is big endian: the most significant byte of the field element comes first. (This is natural in Java but somewhat awkward in C.)

5: Block Cipher

Implement the Threefish-256 block cipher along with the [counter mode]() of operation. Threefish is a [tweakable]() block cipher. The [specification]() ([mirror]()) is part of the Skein hash function description; focus on those chapters relevant to Threefish (2.2, 3.3).

Abstractly, counter mode turns a block cipher into a stream cipher; the keystream is the encrypted counter M_i. Denote E the Threefish-256 block transformation function. We define counter mode with a tweakable block cipher as follows.

- K_i = K for all blocks (the key is fixed)
- T_i = T = IV for all blocks (the tweak is the IV, fixed)
- M_i = i for all blocks (the counter is the message; it starts at zero)
- Keystream block S_i = E(K_i,T_i,M_i), the encryption of the counter under the given key and tweak
- Ciphertext block C_i = S_i XOR P_i (like most stream ciphers, you XOR keystream with plaintext to produce ciphertext)

This has a few implications on implementations:

- Encryption is the same as decryption.
- You don't have to implement the block transformation inverse.
- No message padding is necessary.
- You implement en/decryption of blocks on the fly.

The language tracks are similar to those for the stream cipher assignment.