# CIS4362.01 Homework 3 Due 11/10/19

Brandon Thompson 5517

November 6, 2019

1. Suppose we define a MAC $\mathrm{I_{RAW}} = (S, V)$ where $S(k, m) = \mathrm{rawCBC}(k, m)$.
   *Explain a scenario that shows $I_{RAW}$ can be easily broken using a 1-chosen message attack* (This attack scenario can prove why the last encryption step must be included in ECBC-MAC).
   If adversary gives a one-block message $m \in X$. Request the tag for $m$, get $t = F(k, m)$. Output $t$ as MAC forgery for the 2-block message $(m, t \oplus m)$. Then:

$$\mathrm{rawCBC}(k, (m, t \oplus m)) = F(k, F(k, m) \oplus (t \oplus m)) = F(k, t \oplus (t \oplus m)) = t$$

2. Consider the encrypted CBC MAC built from AES. Suppose we compute the tag for a a long message $m$ comprising of $n$ AES blocks.

   Let $m'$ be the $n$-block message obtained from $m$ by flipping the last bit of $m$ (i.e. if the last bit of $m$ is $b$ then the last bit of $m'$ is $b \oplus 1$). How many calls to AES would it take to compute the tag for $m'$ from the tag for $m$ and the MAC key? (In this question ignore message padding and simply assume that the message length is always a multiple of the AES block size).

   Because only the last bit of the message changes, you would need to compute: $F(k, F(k_1, tag)) \oplus 1$ to generate the new message $m'$, and then compute the new tag by: $F(k_1, F(k, m'))$.
   So a total of 4 AES calls are made.

3. Consider the following MAC verification algorithm:

```
def Verify(key, msg, sig_bytes):
    return HMAC(key,msg) == sig_bytes
```

   (a) *Explain how the timing attack on the above MAC verification algorithm can occur.*

   The '==' is a byte-by-byte comparison so it will return false as soon as it finds an inequality. For a target message make a random tag, loop over all possible first bytes until the verification takes slightly longer. Continue until all bytes in the tag are valid.

   (b) *Write a pseudocode that defends the aforementioned verification timing attack.*

```
return false if sig_bytes has wrong length
result = 0
for x, y in zip( HMAC(key,msg), sig_bytes):
    result |= ord(x) ^ ord(y)
return result == 0
```

   Function checks if the `sig_bytes` is the correct length, XOR's the MAC and the `sig_bytes`, if they are the same then result should be 0. This works because result is fully calculated first, then checked if it is correct.

4. Suppose Alice is broadcasting packets to 6 recipients $B_1, \ldots, B_6$. Privacy is not important but integrity is. In other words, each of $B_1, \ldots, B_6$ should be assured that the packets he is receiving were sent by Alice.

   Alice decides to use a MAC. Suppose Alice and $B_1, \ldots, B_6$ all share a secret key $k$. Alice computes a tag for every packet she sends using key $k$. Each user $B_i$ verifies the tag when receiving the packet and drops the packet if the tag is invalid. Alice notices that this scheme is insecure because user $B_1$ can use the key $k$ to send packets with a valid tag to users $B_2, \ldots, B_6$ and they will all be fooled into thinking that these packets are from Alice.

Instead, Alice sets up a set of 4 secret keys $S = \{k_1, \ldots, k_4\}$. She gives each user $B_i$ some subset $S_i \subseteq S$ of the keys. When Alice transmits a packet she appends 4 tags to it by computing the tag with each of her 4 keys. When user $B_i$ receives a packet he accepts is as valid only if all tags corresponding to his keys in $S_i$ are valid. For example, if user $B_1$ is given keys $\{k_1, k_2\}$ he will accept an incoming packet only if the first and second tags are valid. Note that $B_1$ cannot validate the third and fourth tags because he does not have $k_3$ or $k_4$.

How should Alice assign keys to the 6 users so that no single user can forge packets on behalf of Alice and fool some other user?

$$S_1 = \{k_2, k_3\}, S_2 = \{k_2, k_4\}, S_3 = \{k_3, k_4\}, S_4 = \{k_1, k_2\}, S_5 = \{k_1, k_3\}, S_6 = \{k_1, k_4\}$$

5. Use SHA256 as the hash function to hash the content of the below text. By completing the assignment you will gain experience using crypto libraries such as PyCrypto (Python), Crypto++ (C++), or any other. Write your code and the output of your code for hashing the below message to get full credit.

   ”Anarchism is a political philosophy that advocates self-governed societies based on voluntary institutions. These are often described as stateless societies, although several authors have defined them more specifically as institutions based on non-hierarchical or free associations. Anarchism holds the state to be undesirable, unnecessary and harmful.”

```java
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Main {

        private static final String testMessage = "Hello World!";
        private static final String message = "Anarchism is a
            political philosophy that advocates self-governed
            societies based on voluntary institutions. These are
            often described as stateless societies, although
            several authors have defined them more specifically as
            institutions based on non-hierarchical or free
            associations. Anarchism holds the state to be
            undesirable, unnecessary and harmful.";

        public static void main(String[] args) {
            System.out.println("Test message hash:");
            getHash(testMessage);
            System.out.println("Message hash:");
            getHash(message);
        }

        private static void getHash(String msg) {
            System.out.println(hashHexString(msg));
        }

        private static String hashHexString(String msg) {
            return
                convertByteToHex(calculateHashBytes(msg)).toString();
```

```
        }

        private static byte[] calculateHashBytes(String msg) {
            MessageDigest md = createMessageDigest(''SHA-256'');
            md.update(msg.getBytes());
            return md.digest();
        }

        public static MessageDigest createMessageDigest(String
           algorithm) {
            try {
                return MessageDigest.getInstance(algorithm);
            }
            catch (NoSuchAlgorithmException e) {
                e.printStackTrace();
            }
            return null;
        }

        private static StringBuffer convertByteToHex(byte[] bytes)
           {
            StringBuffer hex = new StringBuffer();
            for(int i = 0; i < bytes.length; i++) {
                hex.append(hexRepresentation(bytes[i]));
            }
            return hex;
        }

        private static String hexRepresentation(byte aByte) {
            return Integer.toHexString(0xFF & aByte);
        }
}
```

Outputs:

```
Test message hash:
7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284addd20126d9069
Message hash:
51c18b42adfb8a7b3082852ae88b6c2b60b4a5895d2b70efeee06d95b7f5

Process finished with exit code 0
```