Homework 2

Posted: Wednesday, October 11, 2017 – 11:59pm
Due: Monday, October 23, 2017 – 5pm (HFH 2108), with partial online submission.

Task 1 – There is only so much one can do

Mr. Cipher is a deep undercover agent from the Republic of Cryptonia. Every day, he sends one of two messages back to base:

- $M_1 = \text{“Nothing to report”}$,
- $M_2 = \text{“Meet me tomorrow at the rendez-vous point, I have information”}$.

To send this messages, Mr. Cipher uses Encrypt-then-MAC, with counter-mode encryption based on AES with 128-bit key, and HMAC with SHA-256 for authentication. In particular, it encrypts the current date (using 8 bytes), followed by the ASCII encoding of one of the above two messages. Mr. Cipher uses the same secret key every day.

You work for the counterintelligence – you know the messages $M_1$ and $M_2$, you know the exact location of the rendez-vous point, and you are intercepting Mr. Cipher’s ciphertexts. Your task is to predict when Mr. Cipher will show up at the rendez-vous location.

a) [Points: 2] Describe a strategy to obtain this information.

b) [Points: 3] How could Mr. Cipher protect himself from your attack strategy?

Task 2 – Integrity and MACs

Hash functions like MD5, SHA-1, and SHA-256 are built from a (very efficient) compression function $h : \{0, 1\}^n \times \{0, 1\}^b \rightarrow \{0, 1\}^n$. To compute $H(M)$, first, the message $M$ is padded into $b$-bit blocks $M_1, \ldots, M_\ell$ as in Task 2. Then, the hash function outputs $H(M) = H_\ell$, where (for a given fixed initialization value $IV$)

$$H_0 = IV \ , \ H_i = h(H_{i-1}, M_i) \text{ for all } i = 1, \ldots, \ell .$$

This construction approach is known as the Merkle-Damgård (MD) paradigm, and is illustrated in Figure 1.

We now build a message-authentication code $MAC_K(M) = H(K \parallel M)$ from a hash function $H$, where the key $K$ is a $b$-bit string, and $M \in \{0, 1\}^*$ is an arbitrarily long message. (Here, $\parallel$ denotes string concatenation.)

a) [Points: 5] Show that $MAC$ does not satisfy unforgeability if $H$ follows the MD paradigm, i.e., given $(M, T = MAC_K(M))$ for an unknown secret key $K$ and a known message $M$, show that it is possible to efficiently find $M' \neq M$ and $T'$ such that $MAC_K(M') = T'$.

**Hint:** Show first that (regardless of what $h$ is) one can always compute from $H(M)$ (using $h$) the hash $H(M')$ for a message $M'$ related to (yet different from) $M$. 

Figure 1: Diagram of a hash function construction following the Merkle-Damgård (MD) paradigm using the compression function $h$.

b) [Points: 3] Why is this attack not possible with HMAC?

**Task 3 – Authenticated Encryption**

We consider Encrypt-and-Mac (E&M) as defined in class. In particular, imagine Alice encrypts her data using counter-mode encryption with AES, and then, to get the final ciphertext, appends to the counter-mode ciphertext a MAC of the data using HMAC with SHA-256.

Imagine that you intercept the following two ciphertexts $C_1$ and $C_2$ generated by Alice, using the same secret key for both (which is however unknown to you):

$$
C_1 = 43371380524753188fd571d8622ae61f64ccbb551d9b348119adbc410cbdef77cf180f7529d0da0c6f0a4fd28a3d56e2105c7eb4b13b8c4cd9001523ba1e55dc2cd5608e84c093cd21d1126dddac1e7b2a5e9
$$

$$
C_2 = 853b56f79857359cad582eb6e6cb1a23b9a08d1c32e8638da8067144b9d781795a0a79496ca15f8e8865408aa83194df66d87eb4b13b8c4cd9001523ba1e55dc2cd5608e84c093cd21d1126ddac1e7b2a5e9
$$

a) [Points: 3] What can you infer about the plaintexts encrypted by the two ciphertexts above? Justify your answer!

b) [Points: 3] Can E&M ever be semantically secure? Explain your answer!

c) [Points: 3] Does E&M satisfy integrity?

**Task 4 – Padding-Oracle Attacks**

In class, we have seen an example of a padding-oracle attack which recovers one plaintext byte from a ciphertext encrypted with CBC encryption using PKCS#7 padding. The attack only needs to make so-called validity checks, each telling us only whether the padding inside the encryption is correct or not. We want now to elaborate on this attack.
a) [Points: 2] Consider the scenario from the class slide, where we want to recover the last byte of the last plaintext block (which may or may not be validly padded). Show that in general there may be two values $X_1$ and $X_2$ such that xoring $X_1 \oplus 0x01$ and $X_2 \oplus 0x01$ to the last byte of the second-last block leads to correct decryption.

**Hint:** What if the second-to-last byte of the last (plaintext) block has value $0x02$ and the last one has value $0x08$?

b) [Points: 2] In case both $X_1$ and $X_2$ lead to decryption, show that with one additional validity check we can determine which one of the two is the actual value.

c) [Points: 4] Explain how to extend the padding-oracle attack presented in class to recover the entire message $M$ given its CBC encryption $C$. How many validity checks does your attack need?

d) [Points: 10] We now want to implement the padding oracle attack from c) against CBC. To this end, we provide `oracle.py` which contains a function `PadOracle` which takes as argument a string (whose length must be a multiple of 16 bytes) and checks whether it encrypts a correctly padded message, for a hard-coded fixed key. In particular, it returns either `True` or `False` to indicate whether the padding is valid or not.

Extend `oracle.py` into a Python program that decrypts any given ciphertext (in a file whose name is passed as an argument) encrypted under the hard-coded key by only using calls to `PadOracle`.

Note in particular the following:

- `oracle.py` is meant to work with Python 2.7 on the CSIL cluster. Unfortunately, CSIL does not seem to support the pycrypto library on Python3. So try to stick with that.

- You can test your implementation on two sample ciphertexts encrypted with the hard-coded key, available at [https://www.cs.ucsb.edu/~tessaro/cs177/hw/1.ctxt](https://www.cs.ucsb.edu/~tessaro/cs177/hw/1.ctxt) and [https://www.cs.ucsb.edu/~tessaro/cs177/hw/2.ctxt](https://www.cs.ucsb.edu/~tessaro/cs177/hw/2.ctxt). Their correct decryption will result in English plaintexts with clearly recognizable structure.

- Only edit `oracle.py` in the designated area in the file (check out the comments). If run on a valid ciphertext, the latter will be in the variable `ciphertext`.

- The key is visible in `oracle.py`, but you should stick to the rules and **not** decrypt directly using it, but only indirectly using `PadOracle`.

- Submit your solution using turnin. Use

```
turin hw2@cs177 oracle.py
```

- We will post further instructions and clarifications on Piazza whenever necessary, so check this out regularly. In particular, we will give some further hints on manipulating strings.

---

1from [https://www.cs.ucsb.edu/~tessaro/cs177/hw/oracle.py](https://www.cs.ucsb.edu/~tessaro/cs177/hw/oracle.py)