Homework 2

Posted: Tuesday, October 4, 2016 – 11:59pm
Due: Friday, October 14, 2016 – 5pm (HFH 2108), with partial online submission.

Task 1 – Secret-Key Encryption (2 points)

Give an example of a property of the plaintext which is not hidden by semantic security.

Task 2 – Integrity and MACs (8 points)

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) $M$.

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

Task 3 – Authenticated Encryption (12 points)

We want to study the security of Encrypt-and-Mac (E&M) and Mac-then-Encrypt (MtE) as defined in class.

a) [Points: 3] Let MAC be a secure (unforgeable) message-authentication code. Use it to build a new message-authentication code $MAC'$ which is also unforgeable, but such that $MAC'(K, M)$ leaks some useful information about $M$.

Justify your answer! (Note that $MAC'$ can have longer tags than MAC, but you have to make sure the tag length is finite, and independent of $M$.)
Figure 1: Diagram of a hash function construction following the Merkle-Damgård (MD) paradigm using the compression function $h$.

b) [Points: 3] Give a variant of counter-mode encryption with the property that given any ciphertext $C$ under a key $K$, it is easy to find a new ciphertext $C' \neq C$ which encrypts the same plaintext as $C$ under $K$. Justify why your scheme is still semantically secure!

c) [Points: 3] Can E&M be semantically secure if we only assume that the underlying MAC is unforgeable? Does E&M preserve integrity?

d) [Points: 3] Does MtE preserve integrity if we only assume that the underlying encryption scheme is semantically secure? Is MtE semantically secure?

Hint: For the last two subtasks, use the first two subtasks.

Task 4 – Padding-Oracle Attacks (18 points)

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\(^1\) 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 ctext.

- 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

  ```
  turnin 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.

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