Massachusetts Institute of Technology
6.857: Network and Computer Security
Professor Ron Rivest

Handout 4
March 11, 2013
Due: March 22, 2013

Problem Set 3

This problem set is due on *Friday, March 22* at **11:59 PM**. Please note that no late submissions will be accepted. Please submit your problem set, in PDF format, *by email* to `6857-staff@mit.edu`. Submit only one problem set per group (with all of your group members' names on it).

You are to work on this problem set with your assigned group of three or four people. Please see the course website for a listing of groups for this problem set. If you have not been assigned a group, please email `6857-tas@mit.edu`. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration.

Homework must be submitted electronically! Each problem answer must appear on a separate page. Mark the top of each page with your group member names, the course number (6.857), the problem set number and question, and the date. We have provided templates for LaTeX and Microsoft Word on the course website (see the *Resources* page).

Grading: All problems are worth 10 points.

With the authors' permission, we will distribute our favorite solution to each problem as the "official" solution—this is your chance to become famous! If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this in your profile on the homework submission website.

Problem 3-1. Side-Channel attack on AES

An attacker can use "side-channel" information to obtain a secret AES key.

We set up a server to simulate this type of side-channel attack. Our server encrypts random strings using AES with a secret key k. The server also "leaks" the number of ones in the XOR outputs during each encryption (i.e. counting only the outputs of the XOR's when some round key is XOR-ed with the current state.)

If you query `http://courses.csail.mit.edu/6.857/2013/aes.php?num=1000` you will receive `num` (plaintext, ciphertext, XOR-output one-count) triples. In this case, `num` = 1000, but feel free to specify any `num` $\in [1 \ldots 10000]$ (it might take a bit to load the triples for a large `num`). Also, feel free to query the server multiple times if you need more triples, because the plaintexts are randomly generated so you will get new data.

The plaintext and ciphertext are printed as space-separated byte values in decimal; and the plaintext is separated from the ciphertext by a comma. The XOR-output one-count is an integer between 0 and 11*16*8 = 1408. (There are 128-bits in the secret key and 10 rounds each having its own round key, so there are 11 XOR's of a round key to the state, each of which XOR's 16 8-bit bytes.) This count is followed by a semicolon.

You may find useful the NIST AES specification in this problem: `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

(Hint: Problem set 3, problem 1 of 2009 was a similar problem, but the solution technique is different. The solution is available here `http://courses.csail.mit.edu/6.857/2013/sol20093-1.pdf`)

(a) Let X be a random variable that is the total number of heads in t independent coin-flips of a fair coin, and let Y another independent random variable that counts the number of heads in t coin flips, and then adds one. Suppose t is known.

Let Z is either X or Y, but you don't know which. How many draws of Z do you expect to need, in order to determine with reasonable reliability whether Z is X or Y?

(b) Describe the algorithm that for recovering an AES key k given the AES side-channel information described above. (Hint: try using the result of part (a) on the first bit of the first round key. What is t?)

(c) Recover the secret key k. Submit any code you used.

(d) How many (plaintext, ciphertext, XOR-output one-count) triples did your attack require? (Note that even when your attack seems to recover almost all the keybits correctly, it may still get one or two key bits wrong, which will result in an incorrect decryption of a ciphertext. Please report the number of plaintexts for which your algorithm has a good chance of outputting all key bits correctly.)

Problem 3-2. Bitdiddle Block Mode

(a) Ben Bitdiddle knows from 6.857 lecture that CTR mode only provides protection against passive (eavesdropping) adversaries, not adversaries who attempt to modify messages. Ben decides to try to remedy this by adding a form of integrity protection to the output. He remembers Merkle hash trees from a previous lecture, so his new construction works like this:

- Pad the message using a reversible transform such that it consists of n blocks, where n is a power of 2. (Note that this will at most double the length of the message.)
- Encrypt using CTR mode as usual, resulting in ciphertext $(r, C_1, C_2, ...C_n)$.
- Using a *separate* key, encrypt the XOR of pairs of ciphertexts in a tree structure. Thus, let $C_{1,2} = E(C_1 \oplus C_2)$, $C_{3,4} = E(C_3 \oplus C_4)$, $C_{1,4} = E(C_{1,2} \oplus C_{3,4})$, etc. Thus, the root of the tree will be $C_{1,n}$.
- Perform an additional encryption with a third key: let $C^* = E(C_{1,n} \oplus r)$.
- Output the final ciphertext $(r, C_1, ...C_n, C^*)$.

(See figure 1 at end of problem set.)

Ben reasons that since C^* now depends on the entire message, an attacker should not be able to forge a new message.

Give an attack that shows that this form of encryption is *malleable*. That is, if an adversary sees one ciphertext (or multiple ciphertexts), they can come up with a different valid ciphertext. What kinds of modifications on the underlying plaintext can the attacker perform?

(b) Perform the same analysis as in part (a), assuming that Ben starts with CBC mode instead of CTR mode (and uses the IV instead of the value r to xor with $C_{1,n}$).

(c) Consider a variant of UFE mode in which the CBC MAC and the value r are output directly, instead of being XORed together. That is, the output consists of $(r, C_1, ...C_n, MAC(C_1...C_n))$. Furthermore, assume that the same key is used for the CTR encryption and the first stage of the CBC MAC. (Using the terminology from class, if the three keys used in UFE mode are K_1, K_2, and K_3, then assume $K_1 = K_2$.)

Give an attack against this scheme. If your attack needs access to encryption or decryption boxes (oracles), say so; if it is a malleability attack (that is, if the attacker only needs to view ciphertexts in order to create a new valid ciphertext), mention this.

Problem 3-3. MACs: From Little to Large

Alice is the financial resource coordinator at a wealthy company and Bob is the transaction manager at a bank. Bob receives transaction requests from Alice to make payments to various individuals, so it is important for Bob to make sure that Alice is indeed sending those messages.

Alice and Bob know of a secure MAC scheme ("Message Authentication Code") called $\mathsf{MAC256}$ that only works on messages of size 256 bits as follows. Alice and Bob share a key k. Given a message m of 256 bits, Alice can compute $\sigma = \mathsf{MAC256}(k, m)$ using its secret key k. Alice sends (m, σ) to Bob and Bob can indeed verify using the key k that Alice sent m.

This MAC is secure as discussed in class: no adversary can come up with a message m' and a MAC for it σ' such that Bob accepts this as a correct MAC and m' is a new message for which Alice did not issue a MAC.

However, Alice wants to send Bob messages that are larger than 256 bits, and Bob needs to make sure that the entire content of those messages was sent by Alice. Let's help Alice design a MAC for long messages $\mathsf{MACLong}$ from the MAC for 256-bit messages $\mathsf{MAC256}$. Let $M = (m_1, \ldots, m_n)$ be the message M represented as n 256-bit blocks (with any padding if necessary).

(a) Alice proposes the following MAC scheme $\mathsf{MACLong}$ that can authenticate messages of any lengths. Alice applies $\mathsf{MAC256}$ to each block of the message M and sends it to Bob. Concretely, Alice sends $(M, \mathsf{MAC256}(k, m_1), \ldots, \mathsf{MAC256}(k, m_n))$. Bob checks the MAC on each block and accepts the whole message if it verifies.

Argue this is not a secure MAC scheme. Give an example of why it is insecure.

(b) Alice understands the scheme above is not secure, so she proposes another scheme. Alice computes the XOR of these blocks $X = m_1 \oplus m_2 \oplus \cdots \oplus m_n$ and then computes the MAC of the result. That is, $\mathsf{MACLong}(k, M) = \mathsf{MAC256}(k, X)$. Bob verifies this MAC given M by also computing X, then computing $\mathsf{MAC256}(k, X)$, and comparing this result with the MAC received from Alice.

Is this a secure MAC scheme? If so, argue its security. Otherwise, show an attack on the scheme.

(c) Alice's boss hires a cryptography consultant which provides the following MAC scheme. Let $\mathsf{hash} : \{0,1\}^* \to \{0,1\}^{256}$ be a hash function that maps arbitrary length messages into a 256-bit message. To compute the MAC of a long message M, one hashes it to a 256-bit block and then computes the MAC of the resulting block with $\mathsf{MAC256}$. That is, $\mathsf{MACLong}(k, M) = \mathsf{MAC256}(k, \mathsf{hash}(M))$. What property of the hash function do we need for the scheme to be secure? Assuming this property, show that the resulting MAC scheme $\mathsf{MACLong}$ is secure.

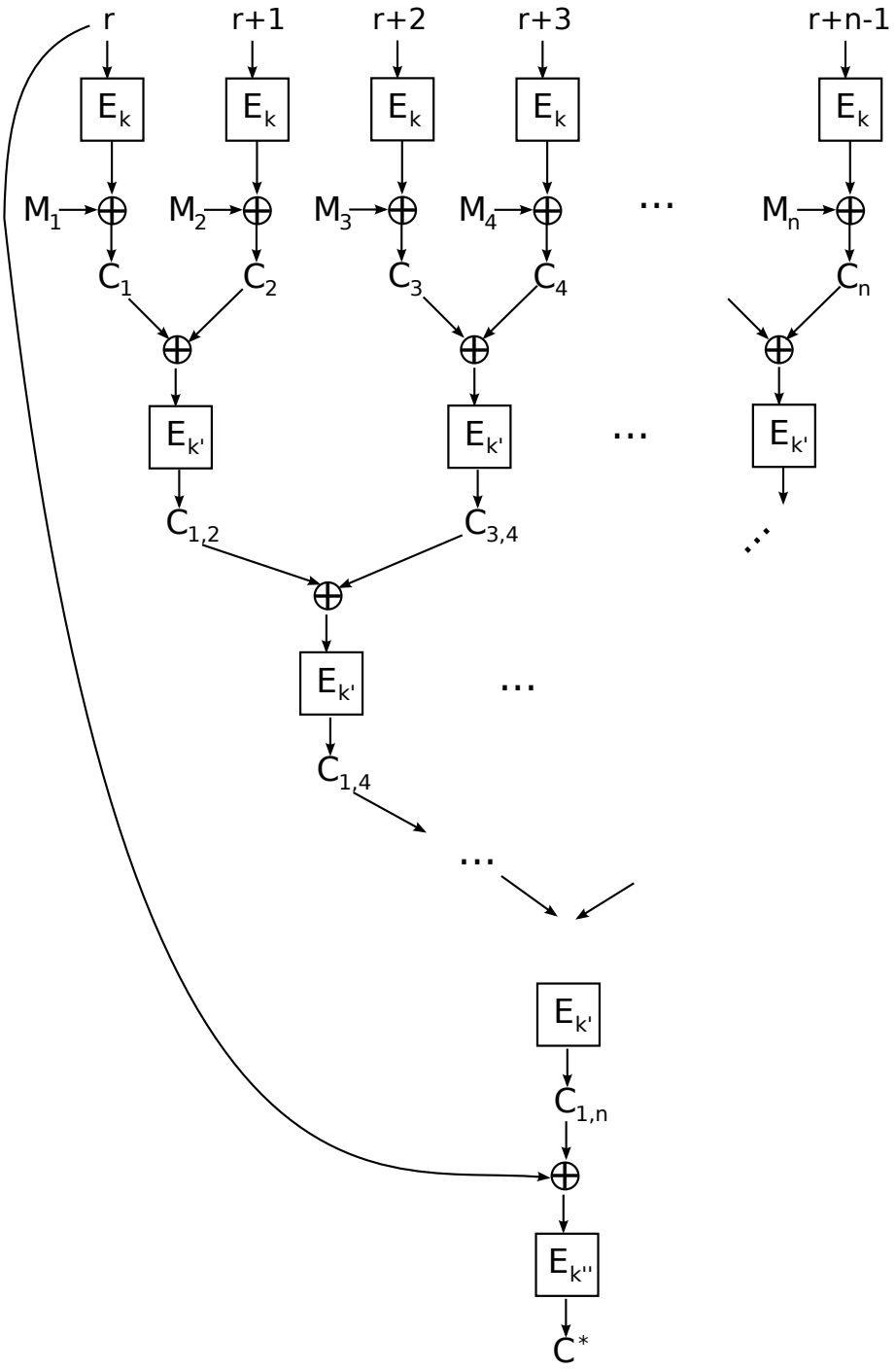

Figure 1: Ben Bitdiddle's proposed block cipher mode for problem 2(a).