Alex Berke March 22, 2023

6.5610 Problem Set 2

Collaborators: *Rui-Jie Yew, Amelia Meles*

Problem 1

(a) False. Considering the key generation algorithm from class, x is randomly chosen from $\{1, \ldots, |G|\}$ and $pk = g^x$. In an attack we can use $m = 1$. Then the output ciphertext is $(g^r, pk^r \cdot m) = (g^r, g^{xr} \cdot m) = (g^r, g^{xr})$. i.e. the ciphertext c is of form g^{xr}. We can efficiently test if g and c are quadratic residues (QRs). If g is not a QR, then g^{xr} is a QR with probability 3/4. Otherwise the probability that c is a QR is 1/2. In the CPA security game we can then send $m_0 = 1$ along with another m_1 and distinguish between the output ciphertexts with more than negligible probability, breaking CPA security.

(b) True. Again the output ciphertext has the form $(g^r, g^{xr} \cdot m)$. Under the DDH assumption we cannot apply the attack from (a) because $g^{xr} \simeq g^u$ for random u. No matter what message m we send, we cannot recover more information about g^x, g^r, g^{xr}. This is the case even when we send multiple messages since each time the r is randomly chosen and hence the tuple (g^x, g^r, g^{xr}) different, so there is no way to combine results from output ciphertexts in the CPA game to gain information.

(c) False. We might use a OWF H s.t. the last bit of output is always 0. An adversary can then send m_0, m_1 in the CPA game, where the last bits are different. Since the encryption scheme will not change the last bit, they can then distinguish between ciphertexts to help identify whether the encryption for m_0 or m_1 was returned, with more than negligible probability.

(d) True. Since the output of H is truly random, and different each time, there is no information that can be recovered about m.

(e) False. There is no ciphertext integrity for Enc_2, so it cannot be CCA secure.

Problem 2

(a) No. This is essentially an "Encrypt-and-MAC" scheme. We know $Enc(k_1, m)$ is randomized but we could have a secure MAC scheme $MAC(k_2, m)$ that is deterministic. In the CPA security game, an attacker can send message m_1 and observe the tag t_1 on output (c_1, t_1). An attacker can then send 2 messages, m_1, m_2, where the same m_1 is used again and observe the repeated tag t_1 in the output (c_1', t_1), (c_2, t_2) to distinguish between the ciphertexts. This breaks CPA security. Since AE implies CPA security, this scheme is not necessarily an AE scheme.

(b) No. This is a "MAC-then-encrypt" scheme. Suppose Enc is a CPA-secure encryption scheme that requires plaintext to be a specific size (e.g. a multiple of block size) and achieves this by adding padding to the end of a message. The padding scheme here reserves the last byte to indicate how many bytes of padding are used. Then the bytes between the plaintext and that final byte are random. For example, for a message m that requires n bytes of padding, $c = Enc(k, m||r_1||r_2|| \dots ||r_n||n)$.

The attack when $m||t$ is used as the message can be shown as follows.

The adversary plays the CCA game and sends 2 messages, a long message m_0 and a short message m_1. The long message requires no padding. The short message does. The challenger returns c and the adversary knows c either has the form $c_0 = Enc(k, m_0||t_0||0)$ or $c_1 = Enc(k, m_1||t_1||r||1)$, where r is a random byte. The goal of the adversary is to discern whether c is c_0 or c_1.

As seen in class, suppose Enc works by combining pseudorandom bytes produced from $AES(k, \cdot)$ to the plaintext by applying $\oplus$. The attacker then produces c' by applying a string of 0 bytes, except where the final padding byte is a non-zero value: $c' = c \oplus (0|| \dots ||0||v||0)$ for some non-zero v. i.e. c' and c are the same except for the second to last byte. This byte is part of the tag for c_0 and random padding for c_1. Therefore, if $c = c_0$ this is not a valid ciphertext, but if $c = c_1$ it can be.

Playing the CCA game, the adversary asks the challenger to decrypt c'. If the challenger returns fail then the adversary assumes $c = c_0$, winning the CCA game.

Since AE security implies CCA security, by breaking CCA security here, we show the scheme is not AE secure.

(c) No. Suppose both Enc and MAC use the same PRF, F, computed over k and a value that is randomly sampled for each message.

For Enc: $r_e \leftarrow \{0,1\}^n$, $Enc(k, m) = (r_e, F(k, r_e) \oplus m) = c$

For MAC: $r_m \leftarrow \{0,1\}^n$, $MAC(k, c) = (r_m, F(k, r_m) \oplus c) = t$

Alone, Enc can be CPA secure and MAC can be secure. The issue when they are used together here is that Enc and MAC share the key, k.

Attack: An adversary playing the CPA game has m_0 encrypted and sees output $(c_0, t_0) = ((r_0, F(k, r_0) \oplus m_0), t_0)$.

Since they know m_0, then can recover the PRF output $F(k, r_0)$ by computing $m_0 \oplus c_0 = m_0 \oplus F(k, r_0) \oplus m_0$. Even without knowing k they can then use $(r_0, F(k, r_0))$ to compute a valid tag t^* for a new ciphertext, c^*, as $t^* = (r_0, F(k, r_0) \oplus c^*)$ and send the spoofed encrypted message (c^*, t^*). Since the tag is valid, the recipient returns $Dec(k, c^*)$ rather than throwing an error. This violates ciphertext integrity.

(d) Yes. This is essentially "encrypt-then-mac and then encrypt again". Since the "encrypt-then-mac" part of the scheme uses a CPA-secure encryption scheme and secure MAC, using different keys, that part of the scheme provides AE, as shown in class. The additional layer of encryption on top does not then change that.

Problem 3

(a) mG. This is because we evaluate m levels of the tree.

(b) F_{m+1} is a PRF because of the recursive definition of F_m. $F_{m+1}(k,i) = G_{i_m}(F_m(k,i))$ for $G_{i_m} = G_0$ or G_1. Since G is a PRG, both $G_0(x), G_1(x)$ are indistinguishable from a random distribution and hence an adversary does not know which of G_0, G_1 was used given input x. Given F_m is a PRF, input x also looks random when evaluated at level $m+1$. The output of PRG G_0 or G_1 applied to random looking input x then also looks random, which is F_{m+1}.

(c) n. We assume Bob knows G_0, G_1 and knows how F_m is constructed from G_0, G_1. Then Bob only needs to know k in order to evaluate F_m, which is size n.

(d) $O(m \times n)$. Given a parent node value, Bob can evaluate all the descendent node values. $m-1$ is the number of minimal nodes s.t. all leaf nodes excluding i and its sibling are descendants. We must then include an additional node for the sibling of node i, which gets us m. Each node is size n.

Problem 4

(a) The probability that at least two students share i is:

$0.9698 = P(A) = 1 - P(A') = 1$ - Pr(no students share i).

$P(A') = \frac{B}{B} \times \frac{B-1}{B} \times \ldots \times \frac{B-82}{B} = \frac{B!}{B^{83}(B-83)!} = \frac{1000!}{1000^{83}(1000-83)!}$

For the second part of this problem - if given $r > B$ uniformly random IDs, and a table with B entries: I would use a hash table.

Given the pigeon hole principle and $r > B$, we know there are collisions. To find a collision we iterate through each ID, i. We compute $i \mod B = x$ and store this in our hash table as $\{x : i\}$. But before storing x, we first check if x is already in the table. If so, we found the collision: the present ID collides with the value that x already maps to.

From the birthday paradox, we expect a collision with probability $\frac{1}{2}$ after approximately $\sqrt{B}$ such i. This means both the hash table size and running time are $O\sqrt{B}$ before we expect to find a collision.

(b) My MIT ID is 914786628. The discrete log m is 78906543209135.

Code: `https://github.com/aberke/applied-crypto-and-security-6.5610/blob/master/pset2/programming/dl.py`