CAS CS 538. Problem Set 4

Due in class Tuesday, October 2, 2012, *before* the start of lecture

Problem 1. (25 points) This problem gives an example of what's wrong with deterministic encryption. (Note that there are *many* reasons that deterministic encryption is generally a bad idea—this is just one of them. The only time deterministic encryption seems to make sense is when the message itself is assumed can be assume to have a lot of randomness, which is not the case here, since the same message is being encrypted twice, so the "second" message has no randomness given the "first" message, because it's the same message. Deterministic encryption of high-entropy messages has actually been studied—I can point you to references if you are interested.)

Suppose Bob and David have two independent Rabin public keys n_B and n_D, respectively. Suppose Alice has a single message m to send to both of them, and m is a square in both $\mathbb{Z}^*_{n_B}$ and $\mathbb{Z}^*_{n_D}$. She encrypts it with plain-Rabin twice to get $c_B = m^2 \bmod n_B$ and $c_D = m^2 \bmod n_D$. Show how an eavesdropper Eve who intercepts c_B and c_D can recover m. (Hint: if Rabin public keys are generated independently, then they are relatively prime with all but negligible probability. You can use the fact that Chinese remainder theorem applies not only to primes, but to any pair of relatively prime integers.) Remark (no work required on your part): the same will be true if we use RSA with public exponent 3 and encrypt with three different moduli.

Problem 2. (25 points) In this problem, we will see how powerful hybrid arugments can be.

Suppose (Gen, E, D) is a secure multi-bit public-key cryptosystem, and G is a secure pseudorandom generator. We want to consider the following public-key cryptosystem. To encrypt m of length l, select a random seed s, generate $p = G(s)$ of length l, and output $c = (E_{\text{PK}}(s), p \oplus m)$. (This is how encryption is often done in real life: a symmetric key s is encrypted first, and then the actual "payload" is encrypted with a symmetric cryptosystem keyed by s.) Show that this cryptosystem is secure. Suggestion:

(a) Show that $(E_{\text{PK}}(s), G(s))$ is indistinguishable from $(E_{\text{PK}}(s), R)$, where $R \in_R \{0,1\}^l$. Hint: use a hybrid argument, with $E_{\text{PK}}(s), G(t)$, for a random unrelated t, as a hybrid point.

(b) Show that $(E_{\text{PK}}(s), p \oplus m_0)$ is indistinguishable from $(E_{\text{PK}}(s), p \oplus m_1)$. Hint: use a hybrid argument, with two hybrid points, and the previous part.

Problem 3. (25 points) Let p_1 and p_2 be two primes of length k bits each. Let $n = p_1 p_2$, let $c \in Z^*_n$ be a $2k$-bit value, and let d be a random $2k$-bit exponent. Assume that the operation of computing $xy \bmod z$ takes time exactly k^2 when x, y and z are all of length k.

(a) How long does it take to compute $m = c^d \bmod n$ using the square-and-multiply technique of Problem Set 1?

(b) Let $c_1 = c \bmod p_1$ and $d_1 = d \bmod (p_1 - 1)$. Note that $m \equiv c_1^{d_1} \pmod{p_1}$. How long does it take to compute $m_1 = c_1^{d_1} \bmod p_1$? Assume that $q_1 = p_1^{-1} \bmod p_2$ and $q_2 = p_2^{-1} \bmod p_1$ are known. How to compute m faster than in part **(a)** using the Chinese Remainder Theorem? How much of a speed-up do you get?

(c) In fact, you can save a little on the Chinese Remainder Theorem computation, and you don't even need q_1, just q_2. Simply compute $m_1 = c^d \bmod p_1$ and $m_2 = c^d \bmod p_2$ as before, then $h = q_2(m_1 - m_2) \bmod p_1$, and then m as $m_2 + hp_2$. Prove that this computation is correct. (Hint: first prove that $m_2 + hp_2$ is congruent to m_1 modulo p_1 and congruent to m_2 modulo p_2. Now prove that $0 \leq m_2 + hp_2 < n$. Recall that CRT states that a value satisfying these three conditions is unique—hence it has to be equal to m).

Problem 4. (25 points) Let p_1 and p_2 be two primes of length k bits each, such that $p_1 \equiv p_2 \equiv 3 \pmod 4$. Let $u_1 = (p_1 + 1)/4$, and $u_2 = (p_2 + 1)/4$. Recall from Problem Set 2 that if s is a square modulo p_1, then a square root of s is $t = s^{u_1} \bmod p_1$.

(a) Prove that t itself is a square modulo p_1.

(b) Prove that 2^ℓ-th root of s is equal to $s^{u_1^l} \bmod p_1$ and is itself a square modulo p_1.

(c) In light of the above and Problem 3, how (and how much) can you speed up Blum-Goldwasser decryption of an ℓ-bit message as compared to simply taking square roots ℓ times? (Ignore the costs of anything but the modular arithmetic. Assume that $u_1^l \bmod p_1 - 1$ and $u_2^l \bmod p_2 - 1$ are computed at key generation and kept with the secret key, so the cost of computing u_1 and u_2 need not be taken into account.)