CS255: Cryptography and Computer Security **Winter 2015**

Assignment #3

Due: Friday, Mar. 13, 2015, by 5pm.

Problem 1. Let's explore why in the RSA public key system each person has to be assigned a different modulus $N = pq$. Suppose we try to use the same modulus $N = pq$ for everyone. Each person is assigned a public exponent e_i and a private exponent d_i such that $e_i \cdot d_i = 1 \bmod \varphi(N)$. At first this appears to work fine: to encrypt to Bob, Alice computes $c = x^{e_{\text{bob}}}$ for some value x and sends c to Bob. An eavesdropper Eve, not knowing d_{bob} appears to be unable to invert Bob's RSA function to decrypt c. Let's show that using e_{eve} and d_{eve} Eve can very easily decrypt c.

a. Show that given e_{eve} and d_{eve} Eve can obtain a multiple of $\varphi(N)$. Let us denote that integer by V.

b. Suppose Eve intercepts a ciphertext $c = x^{e_{\text{bob}}} \bmod N$. Show that Eve can use V to efficiently obtain x from c. In other words, Eve can invert Bob's RSA function.
Hint: First, suppose e_{bob} is relatively prime to V. Then Eve can find an integer d such that $d \cdot e_{\text{bob}} = 1 \bmod V$. Show that d can be used to efficiently compute x from c. Next, show how to make your algorithm work even if e_{bob} is not relatively prime to V.

Note: In fact, one can show that Eve can completely factor the global modulus N.

Problem 2. Time-space tradeoff. Let $f : X \to X$ be a one-way permutation. Show that one can build a table T of size B bytes ($B \ll |X|$) that enables an attacker to invert f in time $O(|X|/B)$. More precisely, construct an $O(|X|/B)$-time deterministic algorithm $\mathcal{A}$ that takes as input the table T and a $y \in X$, and outputs an $x \in X$ satisfying $f(x) = y$. This result suggests that the more memory the attacker has, the easier it becomes to invert functions.
Hint: Pick a random point $z \in X$ and compute the sequence

$$z_0 := z, \;\; z_1 := f(z), \;\; z_2 := f(f(z)), \;\; z_3 := f(f(f(z))), \;\; \ldots$$

Since f is a permutation, this sequence must come back to z at some point (i.e. there exists some $j > 0$ such that $z_j = z$). We call the resulting sequence $(z_0, z_1, \ldots, z_j)$ an f-cycle. Let $t := \lceil |X|/B \rceil$. Try storing $(z_0, z_t, z_{2t}, z_{3t}, \ldots)$ in memory. Use this table (or perhaps, several such tables) to invert an input $y \in X$ in time $O(t)$.

Problem 3. Commitment schemes. A commitment scheme enables Alice to commit a value x to Bob. The scheme is *secure* if the commitment does not reveal to Bob any information about the committed value x. At a later time Alice may *open* the commitment and convince Bob that the committed value is x. The commitment is *binding* if Alice cannot convince Bob that the committed value is some $x' \neq x$. Here is an example commitment scheme:

Public values: (1) a 1024 bit prime p, and (2) two elements g and h of $\mathbb{Z}_p^*$ of prime order q.

Commitment: To commit to an integer $x \in [0, q-1]$ Alice does the following: (1) she picks a random $r \in [0, q-1]$, (2) she computes $b = g^x \cdot h^r \bmod p$, and (3) she sends b to Bob as her commitment to x.

Open: To open the commitment Alice sends (x, r) to Bob. Bob verifies that $b = g^x \cdot h^r \bmod p$.

Show that this scheme is secure and binding.

a. To prove security show that b does not reveal any information to Bob about x. In other words, show that given b, the committed value can be any integer x' in $[0, q-1]$. Hint: show that for any x' there exists a unique $r' \in [0, q-1]$ so that $b = g^{x'} h^{r'}$.

b. To prove the binding property show that if Alice can open the commitment as (x', r') where $x \neq x'$ then Alice can compute the discrete log of h base g. In other words, show that if Alice can find an (x', r') such that $b = g^{x'} h^{r'} \bmod p$ then she can find the discrete log of h base g. Recall that Alice also knows the (x, r) used to create b.

Problem 4. Let's build a collision resistant hash function from the RSA problem. Let n be a random RSA modulus, e a prime relatively prime to $\varphi(n)$, and u random in $\mathbb{Z}_n^*$. Show that the function

$$H_{n,u,e} : \mathbb{Z}_n^* \times \{0, \ldots, e-1\} \to \mathbb{Z}_n^* \qquad \text{defined by} \qquad H_{n,u,e}(x, y) := x^e u^y \quad \in \mathbb{Z}_n$$

is collision resistant assuming that taking e'th roots modulo n is hard.

Suppose $\mathcal{A}$ is an algorithm that takes n, u as input and outputs a collision for $H_{n,u,e}(\cdot, \cdot)$. Your goal is to construct an algorithm $\mathcal{B}$ for computing e'th roots modulo n.

a. Your algorithm $\mathcal{B}$ takes random n, u as input and should output $u^{1/e}$. First, show how to use $\mathcal{A}$ to construct $a \in \mathbb{Z}_n$ and $b \in \mathbb{Z}$ such that $a^e = u^b$ and $0 \neq |b| < e$.

b. Clearly $a^{1/b}$ is an e'th root of u (since $(a^{1/b})^e = u$), but unfortunately for $\mathcal{B}$, it cannot compute roots in $\mathbb{Z}_n$. Nevertheless, show how $\mathcal{B}$ can compute $a^{1/b}$. This will complete your description of algorithm $\mathcal{B}$ and prove that a collision finder can be used to compute e'th roots in $\mathbb{Z}_n^*$.
Hint: since e is prime and $0 \neq |b| < e$ we know that b and e are relatively prime. Hence, there are integers s, t so that $bs + et = 1$. Use a, u, s, t to find the e'th root of u.

c. Show that if we extend the domain of the function to $\mathbb{Z}_n^* \times \{0, \ldots, e\}$ then the function is no longer collision resistant.

Problem 5. One-time signatures from discrete-log. Let $\mathbb{G}$ be a cyclic group of prime order q with generator g. Consider the following signature system for signing messages m in $\mathbb{Z}_q$:

> KeyGen: choose $x, y \xleftarrow{R} \mathbb{Z}_q$, set $h := g^x$ and $u := g^y$.
> output $\text{sk} := (x, y)$ and $pk := (g, h, u) \in \mathbb{G}^3$.
>
> $\mathsf{Sign}(\text{sk}, m)$: output s such that $u = g^m h^s$.
>
> $\mathsf{Verify}(pk, m, s)$: output '1' if $u = g^m h^s$ and '0' otherwise.

a. Explain how the signing algorithm works. That is, show how to find s using sk.

b. Show that the signature scheme is weakly one-time secure assuming the discrete-log problem in $\mathbb{G}$ is hard. The weak one-time security game is defined as follows:

> the adversary $\mathcal{A}$ first outputs a message $m \in \mathbb{Z}_q$ and in response is given the public key pk and a valid signature s on m relative to pk. The adversary's goal is to output a signature forgery (m^*, s^*) where $m \neq m^*$.

Show how to use $\mathcal{A}$ to compute discrete-log in $\mathbb{G}$. This will prove that the signature is secure in this weak sense as long as the adversary sees at most one signature.

[Recall that in the standard game defined in class the adversary is first given the public-key and only then outputs a message m. In the weak game above the adversary is forced to choose the message m *before* seeing the public-key. The standard game from class gives the adversary more power and more accurately models the real world.]

Hint: Your goal is to construct an algorithm $\mathcal{B}$ that given a random $h \in \mathbb{G}$ outputs an $x \in \mathbb{Z}_q$ such that $h = g^x$. Your algorithm $\mathcal{B}$ runs adversary $\mathcal{A}$ and receives a message m from $\mathcal{A}$. Show how $\mathcal{B}$ can generate a public key $pk = (g, h, u)$ so that it has a signature s for m. Your algorithm $\mathcal{B}$ then sends pk and s to $\mathcal{A}$ and receives from $\mathcal{A}$ a signature forgery (m^*, s^*). Show how to use the signatures on m^* and m to compute the discrete-log of h base g.

c. Show that this signature scheme is not 2-time secure. Given the signature on two distinct messages $m_0, m_1 \in \mathbb{Z}_q$ show how to forge a signature for any other message $m \in \mathbb{Z}_q$.

It is worth noting that a tweak of this signature scheme can be proven one-time secure in the standard sense of a chosen message attack. Consider the following scheme:

> KeyGen: choose $x_0, x_1, y \xleftarrow{R} \mathbb{Z}_q$, set $h_0 := g^{x_0}$ and $h_1 := g^{x_1}$ and $u := g^y$.
> output $\text{sk} := (x_0, x_1, y)$ and $pk := (g, h_0, h_1, u) \in \mathbb{G}^3$.
>
> $\mathsf{Sign}(\text{sk}, m)$: choose a random $s_0 \xleftarrow{R} \mathbb{Z}_q$ and output (s_0, s_1) such that $u = g^m h_0^{s_0} h_1^{s_1}$.
>
> $\mathsf{Verify}(pk, m, (s_0, s_1))$: output '1' if $u = g^m h_0^{s_0} h_1^{s_1}$ and '0' otherwise.

For extra credit, try to prove that this signature scheme is existentially unforgeable under a one-time chosen message attack assuming the discrete-log problem in $\mathbb{G}$ is hard. Recall that now the adversary submits his signature query *after* seeing pk.

Hint: Given some $h = g^x$ your goal is to compute x. Try defining the public key pk as $(g,\ h_0 = g^{a_0} h^{a_1},\ h_1 = g^{b_0} h^{b_1},\ u = g^{c_0} h^{c_1})$ for random $a_0, a_1, b_0, b_1, c_0, c_1$ in $\mathbb{Z}_q$.

Problem 6. In this problem we explore a vulnerability in RSA-PKCS1 v1.5 signatures that illustrates the fragility of the scheme. Let $(N, 3)$ be an RSA public-key: N is the RSA modulus and the signature verification exponent is 3. Recall that when signing a message m using PKCS1 v1.5 one first forms the block

$B =$	01	0xFF ... 0xFF	0x00	ASN1	hash

where hash = SHA256(m). The fields are:

- 01 is a two bytes (16 bits) field set to the value 01 (for PKCS1 mode 1),
- 0xFF ... 0xFF is a variable length padding block where each byte is set to 0xFF (i.e. the number 255),
- the $0x00$ field is 1 byte (8 bits) set to 0 indicating the end of the padding block,
- The ASN1 field encodes the type of hash function used to hash the message. For SHA256 this field holds a fixed 15 byte value.
- hash is the hash of the message m: for SHA256 this field is 32 bytes (256 bits).

The purpose of the variable length padding block is to ensure that B is about the size of N. In our case B will be padded to 256 bytes (2048 bits). Note that the ASN1 field was omitted in the lecture for simplicity.

When signing the message m the signer constructs B and then outputs ($B^{1/3} \bmod N$) as the signature σ. Recall that the signer computes the cube root of B using his secret RSA signing key.

To verify a message/signature pair (m, σ) using the public-key $(N, 3)$ one would naively carry out the following steps:

(a) set $B \leftarrow \sigma^3 \bmod N$

(b) parse B from left to right and do:

  i. if the top most 2 bytes are not 01 reject

  ii. skip over all 0xFF bytes until reaching a 0x00 byte and skip over it too

  iii. if the next 15 bytes are not the ASN1 identifier for SHA256 reject

  iv. read the following 32 bytes (256 bits) and compare them to SHA256(m). Reject if not equal.

(c) if all the checks above pass, accept the signature

While this procedure appears to correctly verify the signature it ignores one very crucial step: it does not check that B contains nothing right of the hash. In particular, this procedure will accept a 256 bytes (2048 bits) block B that looks as follows:

$B^* =$	01	0xFF ... 0xFF	0x00	ASN1	hash	more bits J

where J is chosen arbitrarily by the attacker. Here the attacker shortened the variable length block of 0xFF to make room for the value J so that the total length of B^* is still 256 bytes (2048 bits).

Your goal is to show that this leads to a complete break of the signature scheme. In particular, show that just given the public-key $(N, 3)$, an attacker can forge the signature σ on any message m of its choice.

Hint: To forge the signature on some message m, first compute SHA256(m) and then construct the block B (without your appended J) so that the length of B is less than 1/3 the length of the modulus N. Say B is only 80 bytes (640 bits). To do so, simply make the variable length padding block sufficiently short.

Next, your goal is to construct a 256-byte (2048 bits) integer B^* such that:
(1) the first 80 bytes of B^* are equal to B (the remaining bits of B^* are arbitrary), and
(2) B^* is a perfect cube (i.e. is the cube of some smaller integer).
Since B^* is a perfect cube you can easily compute its real cube root σ. Then $B^* = \sigma^3$ holds over the integers and therefore the same also holds modulo N. Since the first 80 bytes of σ^3 are equal to B the signature σ will be accepted as a valid signature on m.

Show how to construct the required 256-byte B^*: it must be a perfect cube and its top 80 bytes must be equal to B. Explain how to construct this B^* and prove that your construction produces a B^* with the required properties.

History: This vulnerability was discovered by Daniel Bleichenbacher in 2006. In 2014 it was discovered that all earlier versions of Mozilla's crypto library, NSS, were vulnerable to a variant of this attack.