CS255: Cryptography and Computer Security **Winter 2014**

Assignment #3

Due: Friday, Mar. 14, 2014, 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 Last week Apple released a softare patch that fixes a significant vulnerability in their TLS implementation. The following code was used to verify a signature in a client-side function:

```
// initialize the hashing context
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
// Hash the signed parameters
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
// read the final hash output into hashOut
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

// check that *signature is a valid signature on hashOut
err = sslRawVerify(ctx,
                   ctx->peerPubKey,
                   hashOut,
                   signature,
                   signatureLen);
 if(err) {  // Report inalid signature error
    sslErrorLog("sslRawVerify returned %d\n", (int)err);
    goto fail;
 }

fail:
   SSLFreeBuffer(&signedHashes);
   SSLFreeBuffer(&hashCtx);
   return err;
```

a. Note the two gotos following the second if statement. Does the function properly check the signature in the buffer `signature`?

b. This function is used in the TLS EDH key exchange to verify the server's signature on the ephemeral Diffie-Hellman parameters in the `server_key_exchange` message. Explain in detail how a network attacker can exploit the error in the code to eavesdrop on all traffic between the client and the server. Draw a diagram of the messages sent from browser to server and vice versa and how an attacker would subvert them.

Problem 4 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 5. 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 6. 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. That is, suppose there is an adversary $\mathcal{A}$ that asks for a signature on a message $m \in \mathbb{Z}_q$ and in response is given the public key pk and a signature s on m. The adversary then outputs 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 as long as the adversary sees at most one signature.

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 $m \in \mathbb{Z}_q$.

d. Explain how you would extend this signature scheme to sign arbitrary long messages rather than just messages in $\mathbb{Z}_q$.