CS255 Homework 3

Rory MacQueen SUNet ID: macqueen

March 16, 2014

Problem 1

A. We know that: $e_{eve} \cdot d_{eve} = 1 \bmod \varphi(N)$

By definition of the modulus operator, this means that:

$$e_{eve} \cdot d_{eve} = 1 + \varphi(N) \cdot K$$

$$e_{eve} \cdot d_{eve} - 1 = \varphi(N) \cdot K = V$$

Now V is a multiple of $\varphi(N)$

B. Suppose e_{bob} is relatively prime to V. This means that $\gcd(e_{bob}, V) = 1$

Meaning that we can use the extended Euclidean algorithm to efficiently solve:

$ae_{bob} - bV = 1$

Since, from part a), we know that $V = \varphi(N) \cdot K$:

$$ae_{bob} - b\varphi(N)K = 1$$

$$ae_{bob} = b\varphi(N)K + 1$$

Now:

$$x^{ae_{bob}} = x^{b\varphi(N)K+1}$$

Since $c = x^e_{bob}$, we have:

$$c^a = x^{b\varphi(N)K+1}$$

By Euler's theorem, $x^{\varphi(N)} = 1$, so therefore we have

$$c^a = 1^{bK}x = x$$

Now suppose V is not relatively prime to e_{bob}. Then: $\gcd(e_{bob}, V) = d$ where $d \neq 1$

What we can do is define a V' where:

$V' = V/Q$ where Q is a multiple of all the factors that e_{bob} and V have in common. As a result, $\gcd(e_{bob}, V') = 1$

Since $\gcd(e_{bob}, \varphi(N)) = 1$, we know that by dividing out Q from V, we have not affected the fact that it is still a multiple of $\varphi(N)$. So, for some other K',

$$V' = \varphi(N) \cdot K'$$

With this, now we can use the same approach that we identified above, but using V' instead of V, and we will again get to x.

Problem 2

We will construct the table T as follows: We select a random element $z \in X$ and start applying f to z, so that $z_0 := z, z_1 := f(z) = f^1(z), z_2 := f(f(z)) = f^2(z)$ and so on. We continue applying f until we hit a cycle - that is, for some value of i, $f^i(z) = z$. Along the way, we drop 'pebbles' at every t^{th} iteration of the chain - that is, we store in the table the values for z_0, z_t, z_{2t}, z_{3t} etc. If, when we finally complete our cycle, we have not seen every element in X, then this means that the permutation function f must have more than one $f - cycle$, so we pick another random $z \in X$ and do this whole process again. We repeat until we have seen all elements in X, at which point we may have multiple tables of z_t 'strings' of elements. If we let $t := \lceil |X|/B \rceil$, then we will store on the order on B pebbles across our tables.

Now that we have this table, suppose we have a set of these tables and a $y \in X$ and want to find an x such that $f(x) = y$. Our algorithm $\mathcal{A}$ will do the following. We start applying f to y - that is, we compute $(f(y), f(f(y)), f(f(f(y)))...)$. Along the way we keep track of the current element, and the element immediately before the current one. If we hit a cycle, namely $f^i(y) = y$, then we simply return the previous element, namely $f^{i-1}(y)$, which is the x we are looking for so we are done. If we reach a 'pebble' element, namely some z_{kt} before we hit a cycle, then we jump back to the pebble preceding this one, $z_{(k-1)t}$, and start applying f from there until we get back to y, at which point we return the element before y in the chain, which is our desired x. Note that x is guaranteed to be in the section of the chain between $z_{(k-1)t}$ and z_t since the pebbles were constructed while computing a cycle. This algorithm runs in time $O(t)$, since it only has to jump back once to a previous pebble.

Problem 3

A. No, the function does not properly check the signature. In the C programming language, only one line after an if clause can be indented and still be covered by the if statement. If there are no curly braces, any lines after that first one will always be executed, regardless of whether the 'if' condition was true or false. Therefore, this code will always hit the second 'goto fail' line, and will therefore never check the signature. In the case where no error actually occurred before this line, the function will return a value indicating that there was no problem, despite the fact that the signature was never checked.

B. In the ServerKeyExchange message, the server sends to the client 'cryptographic information to allow the client to communicate the premaster secret' (RFC5246). In the case of ephemeral Diffie-Hellman protocol, this information is the Diffie-Hellman public key. The client will then use this to send the premaster secret to the server in the ClientKeyExchange message.

An attacker can exploit this error in the following way: it can position itself as a man-in-the-middle between the server and the client. When the server sends its public key in the ServerKeyExchange, the attacker will intercept it, and instead send his own, evil public key to the client. Since the signature on it will not be verified, the client will think that it is talking to the legitimate server, and hence will send back the premaster secret encrypted with the evil public key. The attacker can decrypt this, and gain access to the premaster secret, from which he can derive the session key which is used for all communication between the server and the client for the remainder of the conversation. Attacker can also forward the premaster secret on to the legitimate server (encrypting it with the legitimate sever's public key which was intercepted) so that both sides (client and server) think that nothing is wrong.

ServerKeyExchange

$$CLIENT \quad \overleftarrow{pk_{evil}} \quad ATTACKER \quad \overleftarrow{pk_{real}} \quad SERVER$$

ClientKeyExchange

$$CLIENT \quad \overrightarrow{E(pk_{evil}, premastersecret)} \quad ATTACKER \quad \overrightarrow{E(pk_{real}, premastersecret)} \quad SERVER$$

As stated above, once the attacker has the premaster secret, he can successfully complete the protocol with both server and client and subsequently eavesdrop on all traffic between them.

Problem 4

A. We know that, mod p:

$$b = g^x h^r$$

and we want to show that, for any x', we can construct a r' such that this equation for b will hold. In other words, we want to find:

$$g^x h^r = g^{x'} h^{r'}$$

Now, we know that both g and h are in $\mathbb{Z}_p^*$ and are of order q, which is to say that the size of the subgroup generated by both g and h is q. From our definitions of cyclic groups, since p is prime, we know that $\mathbb{Z}_p^*$ has a unique subgroup of order q. We can therefore conclude that the subgroup generated by g is equal to the subgroup generated by h. Since that is true, there must exist some $k \in [0, q-1]$ such that: $h = g^k$. So we can rewrite our desired equation above as:

$$g^x (g^k)^r = g^{x'} (g^k)^{r'}$$

So now:

$$g^{x+kr} = g^{x'+kr'}$$

$$x + kr = x' + kr'$$

mod q

$$x - x' = kr' - kr$$
$$x - x' = k(r' - r)$$

Now since q is prime and $k \in \mathbb{Z}_q$, we know that k is invertible. So we have:

$$k^{-1}(x - x') = k^{-1}k(r' - r)$$

$$k^{-1}(x - x') = (r' - r)$$

$$r' = r + k^{-1}(x - x')$$

This demonstrates that, given any x', we can construct a unique r' such that the equation for b holds.

B. If Alice can open the commitment as (x', r') then we have that:

$$g^x h^r = g^{x'} h^{r'}$$

Recall from part a) that since the subgroup generated by g is equivalent to the subgroup generated by h, we can rewrite $h = g^k$ for some $k \in [0, q-1]$. If we can solve for k, then we have solved the discrete log problem.

$$g^x (g^k)^r = g^{x'} (g^k)^{r'}$$

$$x + kr = x' + kr'$$

mod q

$$x - x' = kr' - kr$$
$$x - x' = k(r' - r)$$

We know that $(r' - r)$ is efficiently invertible since it is in $\mathbb{Z}_q$ and is not equal to 0 (because $x' \neq x$). So:

$$k = (x - x')(r' - r)^{-1}$$

So we have solved the discrete log problem using the collision on the commitment.

Problem 5

A. $H_{n,u,e}(x, y) = x^e u^y$ $\mathcal{A}$ is an algorithm which outputs a collision for H. Specifically, given some x and y, $\mathcal{A}$ can come up with x' and y', where either $x \neq x'$ and $y \neq y'$ and:

$$H_{n,u,e}(x, y) = H_{n,u,e}(x', y')$$

$$x^e u^y = x'^e u^{y'} \in \mathbb{Z}_n$$

Thus, we can derive the desired result as follows:

$$\frac{x^e}{x'^e} = \frac{u^{y'}}{u^y}$$

$$\left(\frac{x}{x'}\right)^e = u^{y'-y}$$

Now we construct a and b such that:

$$a = \frac{x}{x'}$$
$$b = y' - y$$

and we have:

$$a^e = u^b$$

Recognize that since $y \neq y'$, it must be the case that $|b| \neq 0$. Moreover, since the domain of y is defined to be $\{0, \ldots, e-1\}$, it must be the case that $b = y' - y < e$

B. Since e is prime and, as stated above, $0 \neq b < e$, we know that b and e are relatively prime. Therefore, there are integers s, t such that $bs + et = 1$. Now to derive $a^{1/b}$

$$bs + et = 1$$
$$s + \frac{et}{b} = \frac{1}{b}$$

Take our a from part a) above, and raise it to the power of both sides of this equation to get:

$$a^{s+\frac{et}{b}} = a^{\frac{1}{b}}$$

$$a^s a^{\frac{et}{b}} = a^{\frac{1}{b}}$$

$$a^s (a^e)^{\frac{t}{b}} = a^{\frac{1}{b}}$$

Now, from our result in part a), we know that $a^e = u^b$. Therefore:

$$a^s (u^b)^{\frac{t}{b}} = a^{\frac{1}{b}}$$

$$a^s u^t = a^{\frac{1}{b}}$$

Since s and t are simply integers, we know we can easily compute $a^s u^t$, which will then give us $a^{\frac{1}{b}}$

C. If we extend the domain of the function to $\mathbb{Z}_n^*$ x $\{0, \ldots, e\}$, then we can set $y := e$, and so $H_{n,u,e}(x, y) = x^e u^e = (xu)^e$. From this we can derive a collision on H.

Problem 6

A. The signing algorithm outputs an s such that:

$$u = g^m h^s$$

$$g^y = g^m (g^x)^s$$

$$g^y = g^{m+xs}$$

So, in mod q, we have:

$$y = m + xs$$

$$y - m = xs$$

We need to output an appropriate s here to make this equation hold. Note that if $y = m$, then the left side is 0, so we can just set $s = 0$. If $y \neq m$, then $x \neq 0$, and since $x \in \mathbb{Z}_q$, it must be invertible, so x^{-1} must exist. In that case, we set $s = x^{-1}(y - m)$.

B. Our goal is to derive x, since $h = g^x$, so this would be solving the discrete log problem. Note that, since $h = g^x$ in mod q, we know that $h^q = 1$ because all members of the group generated by g also have order q. Let us set $s = q$ and $u = g^m$. Then s is a valid signature for m since:

$$u = g^m h^s = g^m h^q = g^m$$

So our public key pk is:

$$pk = (g, h, u) = (g, h, g^m)$$

We know send this pk and s to algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ will then send back m' and s' such that $m \neq m'$ and $u = g^{m'} h^{s'}$. So now we have:

$$g^m h^s = u = g^{m'} h^{s'}$$

$$g^m = g^{m'} (g^x)^{s'}$$

$$g^m = g^{m' + xs'}$$

So in mod q:

$$m = m' + xs'$$

$$m - m' = xs'$$

Since $m \neq m'$, we know that $m - m' \neq 0$. Therefore, $s' \neq 0$, and so s' is invertible. So we can derive x as:

$$x = s'^{-1}(m - m')$$

So we have resolved the discrete log problem using algorithm $\mathcal{A}$.

C. Suppose we are given signatures on two distinct messages, m_0 and m_1. Let those signatures be s_0 and s_1 respectively. Then we know that:

$$u = g^{m_0} h^{s_0} = g^{m_1} h^{s_1}$$
$$g^{m_0} h^{s_0} = g^{m_1} h^{s_1}$$

$$g^{m_0} (g^x)^{s_0} = g^{m_1} (g^x)^{s_1}$$

$$g^{m_0 + xs_0} = g^{m_1 + xs_1}$$

So in mod q:

$$m_0 + xs_0 = m_1 + xs_1$$

$$m_0 - m_1 = x(s_1 - s_0)$$

$$x = (m_0 - m_1)(s_1 - s_0)^{-1}$$

Now we have x, suppose we want to compute a new signature s_3 on a new message m_3. We know that they must satisfy:

$$y = m_3 + xs_3$$

So we have:

$$m_3 + xs_3 = m_1 + xs_1$$

$$xs_3 = m_1 + xs_1 - m_3$$

We know that x must have an inverse since it was chosen from $\mathbb{Z}_q$. So now we can derive s_3, which is what we want, as:

$$s_3 = (m_1 + xs_1 - m_3)x^{-1}$$

Hence we have an efficient formula for computing a new signature on a new message.

D. In order to sign a message m of arbitrary length, what we would do is take m and convert it into base q format. Then we could compute a signature for each digit and send that over. By definition, each digit is guaranteed to be between 0 and $q - 1$. This would result in us sending $\log_q(m)$ signatures for a message m.