CS255: Introduction to Cryptography **Winter 2012**

Assignment #3: Solutions

Answer 1. (1+2 points)

(a) We have $d_{\text{eve}} \equiv e_{eve}^{-1} \pmod{\phi(N)}$, so that $\phi(N)$ divides $d_{\text{eve}} \cdot e_{\text{eve}} - 1$.

(b) Pick a random nonzero g in $\mathbb{Z}_N$. In the unlikely event that $g \equiv 0 \pmod{p}$, then $\gcd(p, N) = p$, and we have factored N, so assume that this didn't happen. Let $\tau_p(g), \tau_q(g)$ (which we don't yet know) be the largest powers of 2 that divide $\text{ord}_p(g)$ and $\text{ord}_q(g)$, respectively. Without loss of generality, let $\tau_p(g) \leq \tau_q(g)$.

Note that if g is not a quadratic residue mod p, which happens with probability $\frac{1}{2}$, then $\tau_p(g) = \tau(p-1)$, and otherwise $\tau_p(g) < \tau(p-1)$. Since $\tau_p(g)$ and $\tau_q(g)$ are independent, no matter what $\tau_q(g)$ is, the probability that $\tau_p(g) = \tau_q(g)$ is at most $\frac{1}{2}$.

Now compute $x := g^{K/\tau(K)}$, then repeatedly square x. We know that $x^{\tau_q(g)} \equiv 1 \pmod{N}$, so consider the previous term $y := x^{\tau_q(g)/2}$, if there is one. If $\tau_p(g) < \tau_q(g)$, which happens with probability $\frac{1}{2}$, then $y \equiv 1 \pmod{p}$, but $y \not\equiv 1 \pmod{q}$. Note that in this case we are guaranteed to have a y, because $\tau_q(g) > \tau_p(g) \geq 1$. Then $\gcd(y-1, N) = p$, and we have factored N.

Since the above takes roughly cubic time in $\log N$, and works with probability $\frac{1}{2}$ each time, we will factor N in cubic expected time.

∎

Answer 2. (3 points) First, let us define $t := \lceil |X|/B \rceil$ as the hint suggests. We start by building a table T in the following way: first, choose a random element k, and add k to T. Repeatedly apply f to k until one of the two following things happen: if we reach k again before we have applied f to k exactly t times, then delete k from T. If we have not reached k again after applying f to k exactly t times (i.e. we are in a cycle of length $\leq t+1$) then add the element we get after applying f to k exactly t times to the table t. Let us call this new element j. We then apply f to j either until we reach k again (in which case we stop) or until we have applied f exactly t times, at which point we add the resulting element into the table and continue onward in the same fashion as we have been doing. When we do reach k again, which must happen eventually due to the fact that every element in the group must be a cycle, we choose some element that we have not come across and repeat the process above as though the new element were k. We can repeat this process until we have gone through all of the elements. Of course, this algorithm takes quite some time, but since we have no constraints at this time, we are free to use a sluggish algorithm (and this is why we also don't bother to mention implementation here—any implementation that works is fine).

Now we explain how to invert an input $y \in X$ in time $O(t)$. Our procedure is relatively simple. First, if y is in our table, then we are done, so we clearly do not have to worry about these cases. So suppose that y is not in the table. In this case, we apply f to y repeatedly until we reach an element in our table or until we reach y again, always keeping track of the previous element. Note that if y is in a cycle that does not contain a table entry, then the cycle length must be less than t, so in this case we will find x in time $\leq t+1$. Thus, we may assume that y is in a cycle that contains a table entry. So, we may assume that after applying f enough times to y, we will reach some element z in our table. Suppose we now consider the element in the table immediately preceding z, which

we will call a. Clearly a must be in the same cycle as both z and y and x since we have included the first element of every cycle with at least $t+1$ elements in our table, as well as one additional element. Thus, we can just apply f to a until we reach y, at which point we will know both y and x, which will give us x. If this is the case, we have certainly applied f less than or equal to $t+1$ times, which concludes our algorithm.

Since all branches of our algorithm run in $t+1$ calls to f, it should be clear that our algorithm runs in time $O(t)$. Additionally, it should also be clear that our table has maximum size $\leq 2B$, which is also permissible by the problem (if one wishes to be strict about the B size requirement, it is possible to use $2t$ instead of t when making the table, and our algorithm will run in $2t+1$ time but our table will have exactly B entries). The correctness of our algorithm follows simply from the explanation.

∎

Answer 3. (1+2 points) **(a)** We will first prove that g and h generate the same subgroup G of order q. Briefly, let γ be a generator of $\mathbb{Z}_p^*$, and let $g = \gamma^a$ and $h = \gamma^b$. Let $c = \gcd(a, b)$.

Now, by the extended gcd algorithm, $c = ax + by$ for some x, y, so that $\eta := \gamma^c$ is in the subgroup G generated by g, h. But by the definition of gcd, a/c and b/c are integers, so that $g = \eta^{a/c}$ and $h = \eta^{b/c}$ are generated by η. Therefore every element of G is generated by η. But η has order q, because g and h do, and so $|G| = q$. Furthermore, g and h both generate G.

Whew. On to the meat of the proof. Note that any valid commitment b must be in G, because it's generated by g and h, and each $x \in [0, q-1]$ gives a different value of g^x. Since h also generates G, there is exactly one $r \in [0, q-1]$ such that $h^r = b/g^x$. Since r was chosen uniformly, and each x can give $b = g^x h^r$ in exactly one way, b gives no information about x.

This proof is very similar to the proof of the one-time pad: since h^r is uniformly random in G, multiplying it hides g^x perfectly.

(b) Suppose that given g and h, Alice can construct a commitment b that she can open as x and also as x'. That is, she can produce $x \neq x', r, r'$ such that

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

Dividing through, $g^{x-x'} = h^{r'-r}$. Since $x \neq x', g^{x-x'} \neq 1$ and so $r \neq r'$. Since $\mathbb{Z}_q$ is a field, we can compute $\frac{1}{r'-r} \pmod q$. Raising both sides of the above equation to this power, we have

$$g^{\frac{x - x'}{r' - r}} = h$$

∎

Answer 4. (1+2+1 points) **(a)** Let (x_1, y_1) and (x_2, y_2) be the collision given by $\mathcal{A}$ on inputs n, u. Thus, $x_1^e \cdot u^{y_1} = x_2^e \cdot u^{y_2} \pmod n$. Or equivalently, $(x_1/x_2)^e = u^{y_1 - y_2} \pmod n$. Note that $1/x_2$ is well defined as $x_2 \in \mathbb{Z}_n^*$. As $0 \leq y_1, y_2 < e$, it implies $|y_1 - y_2| < e$. Next, if $y_1 = y_2$, then we get $x_1^e = x_2^e \pmod n$, which implies $x_1 = x_2$. The last fact follows from the fact that the RSA function

with exponent e, $\text{RSA}(x) = x^e \pmod n$ is a permutation. If $y_1 = y_2$, then $x_1 = x_2$ and the tuples given are not a valid collision. Thus $y_1 \neq y_2$, i.e., $|y_1 - y_2| > 0$.

It suffices for $\mathcal{B}$ to therefore run $\mathcal{A}$ on u, n and output $a = x_1 x_2^{-1} \pmod n$ and $b = y_1 - y_2$.

(b) As given in the hint, consider b, s, e, t such that $bs + et = 1$. Consider $\alpha = a^s \cdot u^t$. We get $\alpha^b = a^{sb} \cdot u^{tb}$. However, $u^b = a^e$, which implies $u^{tb} = a^{et}$. Thus $\alpha^b = a^{sb+et} = a^1 = a$. Hence α is $a^{1/b}$. Note, all operations are performed in $\mathbb{Z}_n$.

(c) One possible collision is (u, e) and $(u^2, 0)$. $H(u, e) = u^e \cdot u^e = u^{2e} = (u^2)^e \cdot u^0 = H(u^2, 0)$.

∎

Answer 5. (2+1 points) **(a)** The correct signature S satisfies $S^e = H(C) \pmod N$. In computing $\tilde{S}$, as $S_2 = H(C)^d \pmod q$ is computed correctly, we have $\tilde{S}^e = H(C) \pmod q$. However, as the RSA function is a permutation, we have $\tilde{S}^e = \tilde{S}_1^e \neq H(C) \pmod p$. Compute $A = \tilde{S}^e - H(C) \pmod N$. We have seen $A = 0 \pmod q$ and $A \neq 0 \pmod p$. Thus, $q \mid A$, but $p \nmid A$. Note that $q \mid N$, therefore, $\text{gdc}(A, N) = q$. Thus we recover q by computing gcd (which can be computed efficiently). This factors N.

(b) By simply checking that $S^e = H(C) \pmod N$ after computing the signature S.

∎

Answer 6. (1+2+1 points)

(a) User u is given $K_u = r^{1/b}$. To compute $\mathsf{key}_i = r^{1/e_i}$, user u computes $(K_u)^\gamma$ where

$$\gamma = \prod_{j \neq i, j \in S_u} e_j.$$

By construction, if $i \in S_u$ then $\gamma = b/e_i$ from which it follows, $(K_u)^\gamma = r^{(1/b)\cdot\gamma} = r^{(1/b)\cdot(b/e_i)} = r^{1/e_i}$.

(b) We construct $\mathcal{B}$ as follows: Given x first compute x^{d_1}. Let $y = \mathcal{A}(x^{d_1}, x)$. It is easy to see that $y = x^{d_1/d_2}$. Now, since d_1 and d_2 are relatively prime, there are integers α and β such that

$$d_1 \cdot \alpha + d_2 \cdot \beta = 1.$$

Therefore, knowing that d_1 and d_2 are relatively prime to $\varphi(N)$, we get

$$d_2^{-1} d_1 \alpha + \beta = d_2^{-1} \pmod{\varphi(N)}.$$

Now, we know $x^{d_2/d_1} = y$. Consider $z = y^\alpha \cdot x^\beta$. By modular arithmetic, and invoking Euler's theorem,

$$\begin{aligned} z &= x^{d_2^{-1} d_1 \alpha} \cdot x^\beta \\ &= x^{d_2^{-1} d_1 \alpha + \beta} \\ &= x^{d_2^{-1}} = x^{1/d_2}, \end{aligned}$$

as required. Note, $\mathcal{B}$ does not assume $\mathcal{A}$ works for more than two specific values d_1 and d_2.

(c) The contrapositive requires us to construct an adversary $\mathcal{A}$ (like in part (b)) that computes x^{1/d_2} given $x, x^{1/d_1}$ for some d_2 and d_1. Since we know computing d_2^{th} roots must be hard for a d_2 relatively prime to $\varphi(N)$, this proves part (c).

Assume user u can efficiently obtain key_i for $i \notin S_u$. Note that user u has access to e_i, N and $r^{1/b}$ for some b. This allows him to construct the value of $key_i = r^{1/e_i}$ for a fixed i. Simply set $d_1 = b$ and $d_2 = e_i$. Because e_i's are relatively prime and $i \notin S_u$, d_1 and d_2 are relatively prime as required. Therefore, given r and $r^{1/b} = r^{1/d_1}$ user u computes $r^{1/e_i} = r^{1/d_2}$ as required. Note that we chose r randomly from $\mathbb{Z}_n^*$ therefore this must work for (almost) all r. Thus we construct $\mathcal{A}$ as required.

∎