# CS255 - Problem Set 3

Alex Churchill (achur)

Mar. 13, 2012

( 1 )

( a ) Note that by definition, $e_{eve} \cdot d_{eve} = 1 \bmod \varphi(N)$, so we can get a multiple of $\varphi(N)$ by $e_{eve} \cdot d_{eve} - 1$.

( b ) We follow the hint. Let $k$ be the multiple we determined in part (a). We let $g$ be a generator. Note to find $g$, we can pick any value with $gcd(g, N) = 1$. If we pick some random $g$ for which this is not the case, then $gcd(g, N) = p$ or $gcd(g, N) = q$ where $N = pq$, so we have randomly factored $N$ and are done.

We take the sequence $g^k, g^{k/2}, ...$ until we find some value not equal to $\pm 1$ in $\mathbb{Z}^N$. Note that we can always do this because $g^k = 1 \bmod N$, since $g^k = g^{\ell\varphi(n)}$ for some integer $\ell$.

Note that if we get a value of 1 at $g^{2k/\tau(K)}$ (as opposed to -1), the value we get is a quadratic residue mod $N$ not equal to $-1$. This happens with probability approximately 1/2. Let $x = g^{k/\tau(K)}$. Then $x \neq -1$ and $x^2 = 1 \bmod N$. Therefore, $x^2 - 1$ is a multiple of $N$, so $(x+1)(x-1) = nN$ for $n$ a positive integer. Further, since $x \neq \pm 1 \bmod N$, $x + 1$ cannot be both a multiple of $p$ and $q$. Similarly, $x - 1$ cannot be a multiple of both $p$ and $q$. Thus $x + 1$ must be a multiple of exactly one of $p$ or $q$. Therefore, taking $lcm(x+1, N)$ gives us a prime divisor of $N$.

Since we can factor with probability 1/2, it takes an expected number of two guesses to factor. Further, the probability of failure is exponential in 1/2, so with arbitrarily high probability, we can efficiently factor $N$.

( 2 )

Let $t$ be as in the hint. We construct the table as suggested in the hint, picking a random point $z$ and letting $z_i = f^i(z)$ where $f^i = f \circ f \circ ... \circ f$. Now, pick random $z$ and store $z_0, z_t, z_{2t}, ...$ until our sequence of composing $f$ takes us back to $z$. Now, continue picking random unseen $z$ elements until our table is composed of exactly $B$ bytes. Note that by construction, for any element $z$, applying $f$ to $z$ will give an element in our table after at most $t$ iterations. Therefore, our procedure to invert an input $y$ is as follows: if $y$ is in the table, just return the reverse lookup. Otherwise, apply $f$ to $y$ until we get an element of our table, $z_{(n+1)t}$. Now, take the previous element $z_{nt}$ and apply $f$ repeatedly. Note that such application must give us $z_{(n+1)t}$ in $O(t)$, and must traverse past $y$ along the way, so we choose the element immediately before $y$ as the inverse.

( 3 )

( a ) We note that since $h$ and $g$ are of order $q$, $hg$ must also be either of order $q$ or order 1. If it is of order 1, it means $h^n g^n = 1$ for all $n$, so clearly $h$ and $g$ generate the same subgroup. If it is of order $n$, we note that $hg^i$ over free $i$ and fixed $h$ generates a subgroup of order $q$ (since $h \neq 0$ and $g$ generates a subgroup of order $q$), so for some $k$, $hg^k = 1$ so $h = g^{-k}$. Therefore, the subgroup generated by $h$ contains an element of the subgroup generated by $g$, so they generate the same group.

Note that the previous argument demonstrates the well-known fact that for any $q$, there is at most one power-generated subgroup of $\mathbb{Z}_n$ of size $q$.

Now, note we know fixing $x'$, $g^{x'}h^{r'}$ generates a subgroup of size $q$. From the previous result, we know it must always be the same subgroup. Therefore, for values $r' \in [0, q-1]$, there is a unique $r'$ with $b = g^{x'}h^{r'}$ in that subgroup. Since $x'$ and $r'$ can independently and uniquely generate all members of the subgroup, neither reveals any information about the committed result, so the scheme is secure.

( b ) Imagine we can generate a collision:

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

so then

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

Use the extended Euclidian algorithm to get $z := 1/(r'-r)$. Then

$$h = g^{(x-x')z}$$

so $log_g(h) = (x-x')z$. Since we believe computing discrete logs to be hard, the scheme is binding.

( 4 )

( a ) Using $\mathcal{A}$, get $x^e u^y = x'^e u^{y'} \pmod{n}$.

Since $n$ is an RSA modulus, $n = pq$ where $p$ and $q$ are prime. Note that we can assume WLOG $x$ and $x'$ are relatively prime (since otherwise we could simply divide each by their GCD).

Since $x$ and $x'$ are relatively prime, they must not both be multiples of $n$. If one is a multiple of $p$ and the other is a multiple of $q$, we can use $gcd(x, n)$ and $gcd(x', n)$ to factor $n$, which we know to be a hard problem (from the class notes it is as hard as taking the $e$-th root). Therefore, we can assume that at least one

$x$ or $x'$ doesn't contain a factor of $p$ or $q$. Assume WLOG that it is $x$. Then we can invert $x$, so we take $a = x'/x$ and $b = y - y'$. Note that $a^e = u^b$ by definition.

( b ) Now, we proceed by the hint. Since $gcd(b, e) = 1$, we can find $s$ and $t$ efficiently such that $bs + et = 1$. Therefore,

$$a^{1/b} = a^{s+et/b}$$

$$= a^s \cdot a^{et/b}$$

$$= a^s \cdot (u^b)^{t/b}$$

$$= a^s u^t$$

so $a^s u^t$ is an $e$-th root of $u$ that we can find efficiently.

( c ) We have an immediate collision: let $x = 1, y = e$ and $x' = u$, $y' = 0$. Then $x^e u^y = x'^e u^{y'}$.

( 5 )

( a ) We note that since $S_1 \neq H(C)^d \pmod p$ and $S_2 = H(C)^d \pmod q$, we can say that $\hat{S}^e = H(C) \pmod q$ but $\hat{S}^e \neq H(C) \pmod p$.

Therefore, $z := \hat{S}^e - H(C)$ is a multiple of $q$ but not of $p$. Therefore, $gcd(N, z) = q$, so we can efficiently factor $N$.

( b ) Verify the certificate before releasing it. Note that making a mistake that results in compromising the private key is very likely, but making a mistake in verifying a bad certificate that results in it being certified as good is *very* unlikely.

( 6 )

( a ) User $u$ can simply compute $c = \prod_{j \in S_u \setminus \{i\}} e_j$, then calculate $key_i = K_u^c$.

( b ) We proceed using the same trick as 4(b). We know since $d_1$ and $d_2$ are relatively prime, we can efficiently find $s$ and $t$ such that $sd_1 + td_2 = 1$. Further, since $d_2$ is relatively prime to $\varphi(N)$, we can calculate its inverse, $y$. Then $ysd_1 + t = y \pmod{\varphi(N)}$. Now, use $\mathcal{A}$ on $x^{d_1}$ and $x$ to get $x^{d_1/d_2}$.

Now, we can efficiently compute $x^{sd_1/d_2} \cdot x^t = x^{ysd_1+t} = x^y = x^{1/d_2}$. Therefore, we can efficiently compute $d_2$'th roots in $\mathbb{Z}_N^*$.

( c ) Now, assume we could. Then we could find an algorithm $\mathcal{A}$ for part $(b)$. However, that would imply that we could compute $d_2$'th roots in $\mathbb{Z}_N^*$, which we assume to be hard. Therefore, we cannot efficiently.