Cryptography (SS2006) Prof. Michael Backes

Solutions for Exercise Sheet 7

Out: 06/22/2006 Saarland University

Problem 1: OWFs

(a) Fix $x_0 \in \{0,1\}^n$ and let $\mathsf{F}(x) := x_0$ for all x. This function is obviously *not one-way*, as the any x is an pre-image of x_0. On the other hand, it fulfills the modified definition (called OW'), as the adversary gets *no* information about the value x from seeing x_0. Consequently,

$$OW \not\Leftarrow OW'.$$

On the other, an adversary breaking the new definition can obviously be used to break the old definition as well, as from $x = x'$ is obviously follows that $\mathsf{F}(x) = \mathsf{F}(x')$. Thus $\neg OW' \Rightarrow \neg OW$, thus

$$OW \Rightarrow OW'.$$

Thus we say the modified definition OW' is *weaker* then the original definition OW.

(b) Again we see that the function F fulfills the new definition OW'': For a random element $y \leftarrow_{\mathcal{R}} \{0,1\}^n$ in the range we have probability $1 - \frac{1}{2^n}$ that it is not invertible at all. Thus

$$OW \not\Leftarrow OW''$$

On the other hand, we are going to prove that $OW \Rightarrow OW''$ by proving $\neg OW \Leftarrow \neg OW''$. Let $\mathsf{Pr}[y = \mathsf{F}(x');\ y \leftarrow_{\mathcal{R}} \{0,1\}^n, x' \leftarrow \mathsf{A}(n,y)]$ be not negligible for some efficient A. Then there exist sets $S_n \subseteq \{0,1\}^n$ with $\frac{|S_n|}{2^n}$ is not negligible, and for all $y \in S_n$: $\mathsf{Pr}[y = \mathsf{F}(x');\ x' \leftarrow \mathsf{A}(n,y)]$ is not negligible. But then $|\mathsf{F}^{-1}(S_n)| \geq |S_n|$, thus we have for all $x \in \mathsf{F}^{-1}(S_n)$ that $\mathsf{Pr}[y = \mathsf{F}(x');\ y := \mathsf{F}(x), x' \leftarrow \mathsf{A}(n,y)]$ is not negligible and $\frac{|\mathsf{F}^{-1}(S_n)|}{2^n}$ is not negligible, thus $\mathsf{Pr}[y = \mathsf{F}(x');\ x \leftarrow_{\mathcal{R}} \{0,1\}^n, y := \mathsf{F}(x), x' \leftarrow \mathsf{A}(n,y)]$ is not negligible. This completes the proof.

Problem 2: More Insecurities of Naive RSA

Remember that for the secret message m, we have $m < N$. We are given N_1, N_2, N_3, and $c_1 = m^3 \bmod N_1, c_2 = m^3 \bmod N_2$ and $c_3 = m^3 \bmod N_3$.

1. Compute $egcd(N_1, N_2) = g_{12}$. If $g_{12} \neq 1$ we found a non-trivial factor of N_1 and we are done (as then we can calculate m directly). Otherwise, it gives us u, v such that $u \cdot N_1 + v \cdot N_2 = 1$.

2. For $N_{12} := N_1 \cdot N_2$, we construct $c_{12} := u \cdot N_1 \cdot c_2 + v \cdot N_2 \cdot c_1 \bmod N_{12}$. By the CRT we have $c_{12} = c_i \bmod N_i$ for $i = 1, 2$.

3. Compute $egcd(N_{12}, N_3) = g_{123}$. If $g_{123} \neq 1$ we found a non-trivial factor of N_3 and we are done. Otherwise, it gives us u, v such that $u \cdot N_{12} + v \cdot N_3 = 1$.

4. Now for $N_{123} := N_{12} \cdot N_3$, we construct $c_{123} := u \cdot N_{12} \cdot c_3 + v \cdot N_3 \cdot c_{12} \bmod N_{123}$, thus by the CRT and the above result $c_{123} = c_i \bmod N_i$ for $i = 1, 2, 3$.

5. Thus we have $c_{123} = m^3 \bmod N_{123}$. As $m^3 < N_{123}$, actually $c_{123} = m^3$ over the integers, so one can take the root efficiently (over the integers). This yields m.

Problem 3: On Factoring N and Computing $\varphi(N)$

Let us assume we have an efficient algorithm for factoring N, i.e. we know p and q such that $N = pq$. Thus computing $\varphi(N) = (p-1)(q-1)$ is immediate.

Now, we want to show the converse direction, i.e., if an efficient algorithm exists that computes $\varphi(N)$ given N, then there also exists an efficient algorithm for factoring N, i.e. we can find p and q such that $N = pq$.

We know that

$$\varphi(N) = (p-1)(q-1) = pq - p - q + 1 = N - (p+q) + 1$$

and

$$pq = N.$$

Thus we have two equations and two unknown p and q, and we are going to solve them. Writing $q = \frac{N}{p}$ we get

$$\varphi(N) = N - (p + \frac{N}{p}) + 1,$$

multiplying with p and reordering yields

$$p^2 + p(\varphi(N) - N - 1) + N = 0$$

This equation is easily solvable:

$$p_{1,2} = \frac{(N - \varphi(N) + 1) \pm \sqrt{(N - \varphi(N) + 1)^2 - 4N}}{2}.$$

Problem 4: Hardcore Predicates

Let p be a prime, g a generator of $\mathbb{Z}_p^*$, and $g^x \in \mathbb{Z}_p^*$, we want to compute $\mathsf{DLog}_g(g^x)$ given an algorithm $\mathsf{A}(g^y) = \pi(y)$ that computes the predicate π. The main idea is to compute the discrete logarithm by computing the square roots. In detail we make the following case distinction:

- If $g^x = 1$, then
$$\mathsf{DLog}_g(g^x) = \mathsf{DLog}_g(1) = 0.$$
- If g^x is a QNR, that is, the Legendre symbol is -1 (which can be computed efficiently), in particular x is odd,
$$\mathsf{DLog}_g(g^x) = 1 + \mathsf{DLog}_g(g^x/g).$$
Note that $g^x/g = g^{x-1}$ can be computed efficiently, and is a QR.
- If g^x is a QR, that is, the Legendre symbol is 1, in particular x is even, there is a k s.t. $x = 2k$. So we compute the two square roots $sqrt(g^x) := \{g^k, g^{k+(p-1)/2}\}$ of g^x, this can be done efficiently in $\mathbb{Z}_p^*$. We choose the square root $g^u = s \in sqrt(g^x)$ that fulfills $\mathsf{A}(s) = \pi(u) = 0$, that is $u = k$, since $k + (p-1)/2 \leq p/2$ and then we have
$$\mathsf{DLog}_g(g^x) = 2 \cdot DLog_g(g^k) = 2 \cdot \mathsf{DLog}_g(s).$$

By induction we can show, in each recursion-step the exponent is less then the exponent in the previous step, while the exponent is not 0, since we always choose in the third case the "correct" square root (g^k), that halfs the exponent. Hence the algorithm terminates. And in every step we at most half the exponent, thus the runtime of this algorithm is in $O(log(p))$.