Cryptography (SS2006) Prof. Michael Backes

Solutions for Exercise Sheet 6

Out: 06/15/2006 Saarland University

Problem 1: Security Notions of Keyed Hash Functions

Recall that hash is a collision-resistant keyed hash function iff

$$\mathsf{Pr}[m \neq m^* \wedge \mathsf{hash}(pk, m) = \mathsf{hash}(pk, m^*); pk \leftarrow \mathsf{Gen}(n), (m, m^*) \leftarrow \mathsf{A}(n, pk)] \text{ is negligible in } n,$$

and hash is one-way iff

$$\mathsf{Pr}[\mathsf{hash}(pk, m') = t; pk \leftarrow \mathsf{Gen}(n), m \leftarrow_{\mathcal{R}} \mathcal{M}_{pk}; t \leftarrow \mathsf{hash}(pk, m); m' \leftarrow \mathsf{A}(n, pk, t)] \text{ is negligible in } n.$$

Additionally, we know that every digest of hash has at least two preimages (*).

Now assume there exists an adversary A that can break one-wayness of a CRKHF hash with non-negligible probability, where hash has the additional pre-image property (*). We then construct an adversary B which does the following

1. $pk \leftarrow \mathsf{Gen}(n)$,
2. $m \leftarrow_{\mathcal{R}} \mathcal{M}$,
3. $t \leftarrow \mathsf{hash}(pk, m)$,
4. pass (n, pk, t) to A,
5. receive an m' from A, output (m, m').

A outputs an m' that hashes to t with non-negligible probability ϵ. Remember, that every digest has *at least* two pre-images, thus, with probability at least one half, we have $m \neq m'$.

But now, let us compute, with how much probability we have produced a collision for hash: we have created a collision if A outputs an m' that hashes to t and $m' \neq m$.

$$\mathsf{Pr}[(m, m') \text{ is a collision}] \geq 1/2 \cdot \epsilon.$$

Consequently, a collision-resistant keyed hash function with property (*) is one-way.

Problem 2: Key Generation for ElGamal

Given a prime p and prime q such that $q \mid (p-1)$. We want to find an efficient algorithm to compute an element g of order q.

Take a generator h of $\mathbb{Z}_p^*$ (these can be constructed efficiently) and compute $g := h^{\frac{p-1}{q}}$. With the repeated-squaring algorithm this can be done efficiently. Then g is of the order q, since

$$g^q = (h^{\frac{p-1}{q}})^q = h^{\frac{(p-1)q}{q}} = h^{p-1} = 1,$$

and the order cannot be smaller than q as this would generate a subgroup of G_q, but this has prime order.

Problem 3: Pollard's Rho Algorithm

(a) At the beginning our triple of matter is (x_0, a_0, b_0) and we are given α and $\beta = \alpha^u$, for some $u \in \{0, \ldots, p-2\}$. Let us represent (x_0, a_0, b_0) in the following form $(x_0, a_0, b_0) = (\alpha^{a_0 + u \cdot b_0}, a_0, b_0) = (1, 0, 0)$.

Now we can observe that everytime x_i is squared, the "exponents" a_i and b_i are double modulo p. Furthermore if x_i is multiplied by $\beta = \alpha^u$: $x_{i+1} = \beta x_i = \alpha^u x_i$, then the "exponent" b_i is incremented by 1 and $a_{i+1} = a_i$. And analogous if x_i is multiplied by α: $x_{i+1} = \alpha x_i$, then the "exponent" a_i is incremented by 1 and $b_{i+1} = b_i$. By induction the following holds for every i

$$x_i = \alpha^{a_i}(\beta)^{b_i} = \alpha^{a_i}(\alpha^u)^{b_i} = \alpha^{a+ub_i}.$$

(b) We can assume that $b_i \neq b_j$, since $\mathsf{z} \neq \mathsf{failure}$. Now we have found a triple such that

$$\alpha^{a_i + ub_i} = x_i = x_j = \alpha^{a_j + ub_j}.$$

In particular we have $a_i + ub_i = a_j + ub_j \bmod p$. Since $b_i \neq b_j$, the algorithm computes

$$\begin{aligned} z &= \frac{a_j - a_i}{b_i - b_j} \Leftrightarrow \\ a_i + zb_i &= a_j + zb_j \Leftrightarrow \\ a_i + ub_i &= a_j + ub_j \Leftrightarrow \\ u &= \frac{a_j - a_i}{b_i - b_j}. \end{aligned}$$

That is Pollard's Rho Algorithm does compute $\mathsf{DLog}_\alpha(\beta)$.

(c) We estimate the runtime of this algorithm under two assumptions, namely (i) the function f behaves "chaotic", i.e, each new element is choosen randomly, and (ii) if we found $i \neq j$ such that $x_i = x_j$, then $b_i \neq b_j$.

Let $\mathsf{Pr}[T(p) > i]$ denote the probability that we find no collision choosing the first k elements. Then we calculate, similarly as we did in the proof of the birthday paradox,

$$t_i(p) := \mathsf{Pr}[T(p) > i] = 1 \cdot \left(1 - \frac{1}{p}\right) \cdot \left(1 - \frac{2}{p}\right) \cdot \ldots \cdot \left(1 - \frac{i-1}{p}\right),$$

and can approximate this as we did before by

$$t_i(p) \approx e^{-\frac{i^2}{2p}}.$$

Then one easily sees that

$$\mathsf{Pr}[T(p) = i] = t_{i-1}(p) - t_i(p).$$

The expected value then is

$$\begin{aligned} \mathsf{E}[T(p)] = \sum_{i=1}^{\infty} i \cdot \mathsf{Pr}[T(p) = i] &= \sum_{i=1}^{\infty} i \cdot (t_{i-1}(p) - t_i(p)) \\ &= t_0(p) + \sum_{i=1}^{\infty} (i+1)t_i(p) - it_i(p) \\ &= \sum_{i=0}^{\infty} t_i(p). \end{aligned} \tag{1}$$

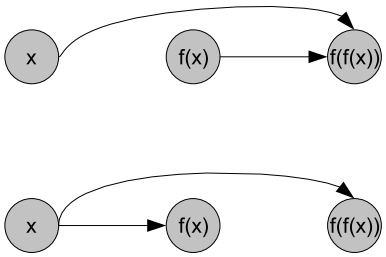

Figure 1: "The fast sequence" cannot jump over the "slow sequence"

Take a look at

$$\begin{aligned}\frac{\mathsf{E}[T(p)]}{\sqrt{p}} &= \frac{1}{\sqrt{p}} \cdot \sum_{i=0}^{\infty} t_i(p) \\ &\approx \frac{1}{\sqrt{p}} \cdot \sum_{i=0}^{\infty} e^{-\frac{k^2}{2p}} \\ &= \sum_{i=0}^{\infty} \frac{1}{\sqrt{p}} \cdot e^{-(\frac{k}{\sqrt{p}})^2/2}\end{aligned}$$

One sees that this is Riemannian sum and thus converges to the integral of $e^{-\frac{x^2}{2}}$:

$$\sum_{i=0}^{\infty} \frac{1}{\sqrt{p}} \cdot e^{-(\frac{k}{\sqrt{p}})^2/2} \to \int_{x=0}^{\infty} e^{-\frac{x^2}{2}} dx = \sqrt{\frac{\pi}{2}}.$$

Thus we see that $\mathsf{E}[T(p)]$ grows approximately as $\sqrt{\frac{\pi p}{2}}$, i.e., in $O(\sqrt{p})$.

Note: Please note that this calculation was indeed harder than we expected. If you derived equation (1) this is definitely sufficient, as we are not giving an analysis course.

(d) In order to reduce the storage we use Floyd's cycle-finding algorithm. We run two instances of the sequence (x_i, a_i, b_i) in parallel, one at "normal speed" and the other at "double speed". At step i we compute two tuples: (x_i, a_i, b_i) and (x_{2i}, a_{2i}, b_{2i}) and test whether x_i is equal to x_{2i}. The tuples at step $i+1$ are computed from the ones at step i as follows:

$$(x_i, a_i, b_i) \to (f(x_i), g(x_i, a_i), h(x_i, b_i))$$

$$(x_{2i}, a_{2i}, b_{2i}) \to (f(f(x_{2i})), g(f(x_{2i}), g(x_{2i}, a_{2i})), h(f(x_{2i}), h(x_{2i}, b_{2i})))$$

When $x_i = x_{2i}$ and $b_i \neq b_{2i} \mod p$ the algorithm finishes and outputs $(a_i - a_{2i})(b_{2i} - b_i)^{-1}$. Only two tuples are stored at any given time so the storage requirement of the algorithm is $O(1)$.

The algorithm performs at most $\lambda + \mu/2$ steps, since the "slow sequence" cannot get more than halfway around the cycle without meeting the "fast sequence", and the fast sequence" cannot jump over the "slow sequence" (Figure 1).