Cryptography (SS2006) Prof. Michael Backes

Exercise Sheet 6

Out: June 6, 2006 Saarland University

Problem 1: Security Notions of Keyed Hash Functions

Prove the following lemma presented in the lecture: A collision-resistant family H of keyed hash functions is one-way if additionally it holds that every digest has at least two pre-images. More precisely, for every security parameter n, for every key pk from the carrier set $[\mathsf{Gen}(n)]$ of the probabilistic key generation algorithm $\mathsf{Gen}(n)$, every element t in the range of $H(pk, \cdot)$ has at least two pre-images with respect to the function $H(pk, \cdot)$.

Recall the definition of one-wayness of such a family as presented in the lecture (it's syntactically close to the definition of collision-resistance presented in the lecture notes):

Definition 1 *A family H of keyed hash functions is* one-way *if for all efficient adversaries A, we have that*

$$\mathsf{Pr}\left[H(pk, m') = t; pk \leftarrow \mathsf{Gen}(n), m \leftarrow_{\mathcal{R}} \mathcal{M}_{pk}, t := H(pk, m), m' \leftarrow \mathsf{A}(n, pk, t)\right]$$

is negligible in n, where $\mathcal{M}_{pk}$ denotes the domain of $H(pk, \cdot)$.

Problem 2: Key Generation for ElGamal

For using ElGamal in subgroups of $\mathbb{Z}_p^*$, we need to pick a random element of order q for setting up a subgroup G_q of $\mathbb{Z}_p^*$ with q elements. Let p, q be two primes with $q \mid p - 1$. Show how one can efficiently find an element in $\mathbb{Z}_p^*$ of order q (which then consequently constitutes a generator of a subgroup G_q of $\mathbb{Z}_p^*$).

Problem 3: Pollard's Rho Algorithm

In this problem we explore Pollard's Rho algorithm for computing discrete logarithms $\mathsf{DLog}_\alpha(\beta)$ in arbitrary groups of prime order, i.e., the algorithm does not exploit any special group structure.

Let a group G of prime order p be given along with a "suitable" partition into sets G_0, G_1, G_2, i.e., $G = G_0 \cup G_1 \cup G_2$ and $G_i \cap G_j = \emptyset$ for $i \neq j$. The algorithm starts with a tuple $(x_0, a_0, b_0) = (1, 0, 0)$ and successively computes tuples $(x_{i+1}, a_{i+1}, b_{i+1}) := (f(x_i), g(x_i, a_i), h(x_i, b_i))$ using the following functions f, g, h:

$$f(x) := \begin{cases} x^2 & x \in G_0 \\ \beta x & x \in G_1 \\ \alpha x & x \in G_2 \end{cases}$$

$$g(x, a) := \begin{cases} 2a \bmod p & x \in G_0 \\ a & x \in G_1 \\ a + 1 \bmod p & x \in G_2 \end{cases}$$

$$h(x, b) := \begin{cases} 2b \bmod p & x \in G_0 \\ b + 1 \bmod p & x \in G_1 \\ b & x \in G_2 \end{cases}$$

It computes and stores these tuples until it finds two tuples with $x_i = x_j$. Then it tests if $b_i = b_j \bmod p$, in which case it outputs failure, otherwise it outputs $z := (a_j - a_i)(b_i - b_j)^{-1} \bmod p$ as $\mathsf{DLog}_\alpha(\beta)$.

(a) Find an invariant for (x_i, a_i, b_i), i.e., a relation that is fulfilled for each index i.

(b) Prove that, if the algorithm outputs $z \neq \mathsf{failure}$, then $z = \mathsf{DLog}_\alpha(\beta)$.

(c) Estimate the running time of the above algorithm, i.e., the number of iterations of f needed to find the first collision. (Note that termination only depends on the values x_i and thus only on f.) You may assume that the function f behaves "chaotic", i.e., each new element x_i is a random element of G, and that the algorithm never outputs $\mathsf{failure}$. (These assumptions of course depend on the choice of G_i and are an idealization, but practice shows that they are realistic.)

(d) The algorithm as described above needs to store all values $x_1, \ldots x_i$ and to compare x_{i+1} with all of them. Can you think of a modification that needs to store only two such tuples, while only moderately decreasing speed?