Cryptography

Prof. Dr. Joachim von zur Gathen, Konstantin Ziegler

8 Assignment: Hashing and Discrete Logging

(Due: Thursday, 14 January 2010, 23^{59})

Exercise 8.1 (Hashing long messages)**.** We start with the already familiar hash function

$$\begin{aligned} h_C\colon\ \mathbb{Z}_d \times \mathbb{Z}_d &\to \mathbb{Z}_p^\times, \\ (x, y) &\mapsto g^x z^y \end{aligned}$$

using the prime number $p = 1031, d = 1030$ and the two generators $g = 14$ and $z = 835$. This hash function cannot hash more than 21 bits of information. But we want more. In the lecture you have been introduced to the Merkle-Damgård construction for hashing long messages. The starting point is a function of the form

$$\begin{aligned} h\colon\ \mathbb{Z}_2^m &\to \mathbb{Z}_2^t, \\ x &\mapsto h(x) \end{aligned}$$

with $m \geq t + 2$. We pick $m = 20, t = 11$ and use h_C from above to define the image $h(x)$ for $x \in \mathbb{Z}_2^m$ as follows:

1. Read the bit-string $x \in \mathbb{Z}_2^m$ as the binary representation of an integer $\bar{x}$.
2. Let a and b be quotient and remainder of $\bar{x}$ under division by d.
3. Let $c = h_C(a, b)$.
4. Interpret the binary representation of c as a bit-string in $\mathbb{Z}_2^t$ and define this to be $h(x)$.

The construction is illustrated by the following diagram

$$\begin{array}{rccc} h\colon & \mathbb{Z}_2^m & \dashrightarrow & \mathbb{Z}_2^t \\ & \downarrow & & \uparrow \\ & \mathbb{Z}_d \times \mathbb{Z}_d & \xrightarrow{h_C} & \mathbb{Z}_p^\times \,. \end{array}$$

Now, it is your turn. Use the MERKLE-DAMGÅRD construction on h to construct a hash function

$$h^*\colon \bigcup_{n\geq 0} \mathbb{Z}_2^n \to \mathbb{Z}_2^t$$

for hashing messages of arbitrary length n.

(i) (8 points) Implement h^* in a CAS of your choice. What is $h^*(\text{'00000000'})$? Compute also the hash of the binary representation of your Student ID.

We proved in the security reduction for h^*, that every collision for h^* leads to a collision of h in polynomial time. Several cases have to be distinguished. We revisit them by examples.

To make them more interesting we want to use messages with letters instead of bits. Therefore we identify each letter with the last five bits of the corresponding binary ASCII encoding, i.e.

<blank>	00000	...	...	=	11101
A	00001	Y	11001	>	11110
B	00010	Z	11010	?	11111
C	00011	;	11011		
...	...	<	11100		

Consider the following six messages (all starting and ending with a non-space character):

$$X_1 = \texttt{Merry Christmas and a Happy New Year}$$
$$X_1' = \texttt{?PDASADM;T AVPUAX<WBDFHMRBJKFAD D;M}$$
$$X_2 = \texttt{Frohe Weihnachten und ein Gutes Neues Jahr}$$
$$X_2' = \texttt{VJCYH<IDW<USHDES>ZORJNDTAB?WYX >;< ZDF<QRQI}$$
$$X_3 = \texttt{Joyeux Noel et une Bonne Annee}$$
$$X_3' = \texttt{RULZYY?FYYSOZEEWUFRB<GRCIFDMNBXYXVWZHJ}$$

(ii) (12 points) These messages provide three collisions (X_i, X_i') for h^*. Find the corresponding collisions of h and document your steps properly.

(iii) (6 bonus points) Try to compute $\operatorname{dlog}_g z$ from the collisions of h. What did you expect?

Exercise 8.2 (Algorithms for Discrete Logarithms)**.** You have encountered several algorithms for the discrete logarithm problem in a multiplicative group $\mathbb{Z}_p^\times$. You have also seen the results on their time and storage requirements. Now it is time to put these results into perspective and gain some hands-on experience.

We start with the multiplicative group $G = \mathbb{Z}_p^\times$ where p is the following 8-digit prime:

$$41723027$$

As usual the first task is to determine a generator g of G which will serve as the base for our discrete logarithms. You do not have to find one for yourself, since we have hidden one in the following set of group elements:

$$S = \{4, y = 1063, 1069, y^{-1} = 12049830, 41723026\} \subset G$$

You can rule out some of the elements right away. In fact, given the additional information that there is exactly *one* generator in this set, you can determine it without any computation.

(i) (3+3 bonus points) Find the generator g of G in the set S and give a complete argument for your decision. Avoid computations to get bonus points.

We want to consider two algorithms to solve the problem of the discrete logarithm:

- Chinese remaindering, where you call Pollard's rho method with Floyd's trick for the computations in the subgroups
- Index calculus

(ii) (12 points) Implement the two algorithms in a CAS of your choice. Use them to compute the discrete logarithm $\operatorname{dlog}_g x$ where x is

$$24122008.$$

These algorithms rely on random choices, so several calls – even for the same x – will result in different CPU times.

(iii) (4 points) Add a variable to your programs that outputs the total time that was needed for the computation and compare the average over 10 runs.

(iv) (+4 points) Formulate expectations for the runtime of your algorithm depending on the bit-length of x. Try and find experimental evidence for your claim.

(v) (+4 points) Formulate expectations for the runtime of your algorithm for different group orders, based on the estimates that were established in the lecture. Put your predictions to a test by using different groups (maybe of order near $2p, 4p, 8p, \dots$) and document your results in a table.