Soumodipta Bose| 2021201086

Using DLP to build fixed-length Collison Resistant Hash function (Theory)

We have previously looked at DLP to build ourselves a Pseudo-random generator and then progressed to make many other constructions. We can also use DLP to build a Collision resistant Hash function which is very useful in a lot of different constructions like HMAC.

Sketches/Ideas:

Hash function $h_s(x_1, x_2)$

$h_s : Z_p \times Z_p \to Z_p$

p = prime no. (safe) → i.e. $q = (p-1)/2$ is also prime

$h_s(x_1, x_2) = (g^{x_1} h^{x_2}) \bmod p$

g: generator of the group

h: randomly chosen between 0 to p-1

Construction:

THEOREM 7.73 *If the discrete logarithm problem is hard relative to $\mathcal{G}$, then Construction 7.72 is collision resistant.*

To build a fixed length collision resistant hash function we need the discrete log assumption again, that is if the discrete log problem is hard , then we can construct a collision resistant hash function.

CONSTRUCTION 7.72

Let $\mathcal{G}$ be as described in the text. Define (Gen, H) as follows:

- *Key generation algorithm* Gen: On input 1^n, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$ and then select $h \leftarrow \mathbb{G}$. Output $s = \langle \mathbb{G}, q, g, h \rangle$.
- *Hash algorithm* H: On input $s = \langle \mathbb{G}, q, g, h \rangle$ and message $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$, output $g^{x_1} h^{x_2} \in \mathbb{G}$.

A fixed-length hash function.

Here Gen is the polynomial time algorithm that finds the group such that s=(G,q,g,h). In our implementation I have used safe prime number for p, where p represents the prime number of the group , such that q=(p-1)/2 such that it is also prime. I have used one of the Sophie Germain Primes close to 16bits.

Here I am generating the value of g as the primitive root of the prime number and randomly choosing h within the range of 2,p-1. So now based on these parameters we can build a function that takes two inputs x1 and x2 and applies the discrete log over it.

Finally I am using a wrapper to make sure that the operations are performed on a binary string, this is specific to my implementation.

Now let us consider that x1 and x2 will collide

$$H_s(x_1\|x_2) = H_s(x_1'\|x_2') \Rightarrow g^{x_1}h^{x_2} = g^{x_1'}h^{x_2'}$$
$$\Rightarrow g^{x_1-x_1'} = h^{x_2'-x_2}. \tag{7.3}$$

Let $\Delta \stackrel{\text{def}}{=} x_2' - x_2$. Note that $[\Delta \bmod q] \neq 0$ since this would imply that $[(x_1 - x_1') \bmod q] = 0$, but then $x = x_1\|x_2 = x_1'\|x_2' = x'$ in contradiction to the assumption that there was a collision. Since q is prime and $\Delta \neq 0 \bmod q$, the inverse Δ^{-1} exists. Raising each side of Equation (7.3) to the power Δ^{-1} gives:

$$g^{(x_1-x_1')\cdot\Delta^{-1}} = \left(h^{x_2'-x_2}\right)^{\Delta^{-1}} = h^{[\Delta\cdot\Delta^{-1} \bmod q]} = h^1 = h,$$

and so

$$\log_g h = [(x_1 - x_1')\Delta^{-1} \bmod q] = \left[(x_1 - x_1')\cdot(x_2' - x_2)^{-1} \bmod q\right],$$

We see that A correctly solves the discrete logarithm problem with probability exactly ε(n). Since, by assumption, the discrete logarithm problem is hard relative to G, we conclude that ε(n) is negligible.