Principles of Information Security

Indranil Pradhan

2019202008

M.Tech in CSIS

Evaluation 1

Question Number [Q]

Question :- Design a zero-knowledge proof for the Discrete-Logarithm Problem (DLP), that is, given prime *p*, generator *g* and the element $y = g^x \bmod p$, how does a prover claiming to know *x*, convince the veri_er, without revealing *x*? Moreover, using hash-functions (and assuming them to be random oracles) show how would to build a *digital signature* scheme based on your above zero-knowledge proof and the hardness of DLP? Also, show how would you design collision-resistant hash functions based on the hardness of DLP.

Answer:

- To achieve the goal following steps are followed
  1. Prover choose a random number 0 <= r < p - 1 and sends the verifier $h = g^r \pmod p$.
  2. Verifier sends back a random bit b.
  3. Prover sends $s = (r + b * x) \pmod{p - 1}$ to verifier.
  4. Verifier computes $g^s \pmod p$ which should equal to $h * y^b \pmod p$.

  The basic idea here is that if b = 1, the prover gives a number to the verifier (V) that looks random $s = r+x \pmod{p-1}$. But V already knows $h = g^r$ and $y = g^x$ and can multiply these and compare them to g^s.
  We should be careful what is proved by that. What verifier actually sees are h and s, and so what verifier knows is that $s = dlog(h) + x \pmod{p - 1}$, where dlog(h) is the discrete log of h relative to g. The verifier knows s and so do you, the prover. Now if you also know dlog(h), then it's clear that you know x.

- Here the assumption based on that the message M is hashed to a value m.
  1. Let x be a secret key known only to you, the signer. Let p be a large prime, and g be a generator of Z_p^*. You can publish $(g, p, g^x \pmod p)$ as your public key.
  2. In order to sign m, (prover) choose a random r and compute c (simulating verifier's choice) as the hash of $c = h(m^x \pmod p), m^r \pmod p), g^r \pmod p))$.
  3. Let $s = c * x + r$, you publish the digital signature which is m together with $(s, m^x \pmod p), m^r \pmod p), g^r \pmod p))$.
  4. To check the signature, a verifier first computes c as the hash of the values $(m^x \pmod p), m^r \pmod p), g^r \pmod p))$ which were published with the signature.

  Then the verifier checks that $g^s \pmod p = (g^x)^c * g^r \pmod p$ and $m^s \pmod p = (m^x)^c * m^r \pmod p$.
  Here the goal to convince the verifier that the prover knows x which private to the prover. So the prover instead of x gives s which depends on x. But verifier does not know about x as s is multiplied by random variable c. added a random value r to it. For that the distribution will be random

- The discrete log is hard. Because of that it can be safely told the verifier about $g^x \pmod p$ and $g^r \pmod p$. These values also does not help verifier to know about x and r. Th e value c is really a "challenge" to you, the prover, to prove that you know x as it is computed from random hash function. When x is private and not known, you will be challenged with with a c, the value $(g^x)^c * g^r \pmod p$ could be any element of Z_p^*. So it is hard discrete log problem to find out s that satisfy satisfying $g^s \pmod p = (g^x)^c *$

g^r (mod p). There are two many possible c's , so the signer also can't guess r by enumeration and the c which satisfies the possibility is of very probability. (For a one bit b it is fifty fifty)