Cryptographic hash function

Lesson 7

Hash function definitionm

Definition (hash function). A function takes as input an arbitrarily long document D and return a short bit string H:

- Computation of Hash(D) should be fast and easy: H = Hash(D).
- Inversion of $Hash^{-1}(H)$ should be difficult: given H = Hash(D), it's difficult to find D.
- Hash be collision resistant: it is difficult to find two documents D1, D2 whose H(D1)=H(D2).

Hash function implementation

- Using a mixing algorithm M that transforms a bit string of length n and into another bit string of length n;
- Breaking a long document D into blocks;
- Successively using M to combine each block with the previously processed material.

$$D = D1 \,||\, D2 \,||\ldots||\, Dk$$

$$H0 = \text{initial bit string}$$

$$H_i = H_{i-1} \text{ XoR } M(Di),\ 1 \leq i \leq k$$

$$H = H_k$$

Practical hashes

- Since speed is of fundamental importance for hashes, one tends to use hashes constructed using ad hoc mixing operations, rather than basing them on hard problems.
- The hashes in most widespread use today: MD5 (Message Digest algorithm 5) , SHA (Secure Hash Algorithm).
- SHA: SHA-1 (160 bits), SHA-n (n bits: 224, 256, 512).
- SHA-n: ~2^n steps to invert SHA-n, and $2^{n/2}$ steps to find a collision.

SHA-1 algorithm

1. Break D into 512-bit chunks.
2. Start with 5 initial values h0, …, h4.
3. LOOP over the 512-bit chunks:
    1. Break a 512 bit chunk into sixteen 32-bit words.
    2. Create a total of eighty 32-bit words w0,…, w79 by rotating the initial words.
    3. LOOP i=0 $\rightarrow$ 79:
        1. a = h0, b = h1, c = h2, d = h3m e = h4.
        2. Compute f using XoR and AND on a, b, c, d, e.
        3. Mix a, b, c, d, e by rotating some their bits, permuting them, and add f and wi to a.
    4. h0 = h0+a, h1 = h1+b, h2 = h2+c, h3 = h3+d, h4 = h4+e.
4. Output h0||h1||h2||h3||h4.

Random and pseudo- random numbers

- Ideally, we would like a device that generates a completely random list of 0's and 1's.
- Such devices exist (Geiger counter).
- Unfortunately, as a practical matter, it's expensive to build Geiger counter for each computer.

- So we can just generate pseudo-random numbers.

Pseudorandom number generator

- PRNG is a function of two variable F(X, Y).
- In order to get started, choose a truly seed value S (or as random as we can make it).
- Compute R0 = F(0, S), R1 = F(1, S), …
- List R0||R1||… is the (pseudo) random bit string.

Cryptographically secure PRNG

A PRNG is cryptographically secure if:

1. If Ever knows the first k bit of random bit string, Ever should have no better than 50% change of predicting whether the next bit will be 0 or 1.
2. Suppose that Ever can find out the values R_t, R_{t+1}, … This should not help Ever to determine the earlier par R_0, …, R_{t-1}.

Implementation

- One can build a PRGN out of Hash by choosing an initial random value S and setting: Ri=Hash(i||S).
- One con build a PRGN from a A/symmetric cryptosystem E_K, for example RSA, AES: R=E_K(C XoR S), where X=E_K(D) and D: computer time.

- MAC (Message Authentication Code): M=M0||M1||… $\rightarrow$ MAC(M): Ci = Mi XoR Ri, where R0 = Seed.