

One Way hash functions

(a.k.a. message digests, fingerprint, cryptographic checksum)

A *hash function* is a function h mapping large inputs to small outputs

h is *one-way* if, given x , it is hard to find a value M such that

$$h(M) = x$$

h is *collision-free* if it is hard to find two different messages M, M' such that $h(M) = h(M')$

We typically ask a hash function to be both one-way and collision free.

Speeding up Digital Signatures by one-way functions

If M is very long (e.g. a contract) it is faster to sign a hash of M rather than M :

$$S_{K_A^{-1}}(M) = (M, E_{K_A^{-1}}(h(M)))$$

$$V_{K_A}(M, sig) = \begin{cases} M & \text{if } D_{K_A}(sig) = h(M) \\ \text{error} & \text{otherwise} \end{cases}$$

1. authentication, unforgeability: suppose (malicious) Mallory wants to forge A's signature on message M' . Mallory can compute $h(M')$, but does not know K_A^{-1} so cannot compute $E_{K_A^{-1}}(h(M'))$
2. malleability of message, signature not reusable:

Suppose Mallory wants to use the signature of (M, sig) to create another signed message (M', sig) .

Note if $h(M') = D_{K_A}(sig) = h(M)$ then $V_{K_A}(M', sig) = M'$.

But constructing M' such that $h(M') = h(M)$ is hard when h is one-way.

Note: the cost of efficiency is reduced security, particularly re point compared with the previous scheme!

Digital Signature Standard/Algorithm DSS/DSA

proposed by US National Institute of Standards and Technology in 1991

developed by NSA

adopted as standard for US government communications in 1994 based on difficulty of discrete log & uses a hash SHA (Secure Hash algorithm) as part of algorithm

10-40 times slower than RSA for verification, but more secure in some ways

much political objection: many companies had already put considerable work into implementing RSA

patented but made available on royalty-free basis

possibly infringes on a patent by Schnorr (valid till 2008)

Some Statistics of Birthdays

Q1: How many people need to be at a party for the probability that at least one of them has the same birthday as you to be $> 1/2$?

Q2: How many people need to be at a party for there to be at least two with the same birthday?

Answer to Q1: 253

Answer to Q2: 23

Birthday attack on Hashed Signatures (Yuval 1979)

Alice signs messages using a hash function h .

If h is not collision-free, Bob cheats Alice as follows:

1. Bob prepares a long `Contract1`, with consideration "Bob will pay Alice \$100", and another long `Contract2`, with consideration "Alice will pay Bob \$1,000,000",
2. Bob prepares a large number of versions of `Contract1`, differing only wrt spacing (e.g. for each line, add a space at the end of the line, or not: K lines $\Rightarrow 2^K$ versions.) Similarly for `Contract2`.
3. Bob hashes all these versions, looking for a version V_1 of `Contract1` and a version V_2 of `Contract2` such that $h(V_1) = h(V_2)$.

4. Bob asks Alice to sign V_1 . Happy to receive \$100, Alice responds with $(V_1, E_{K_A^{-1}}(h(V_1)))$.

5. Bob sues Alice for \$1,000,000, and presents $(V_2, E_{K_A^{-1}}(h(V_1))) = (V_2, E_{K_A^{-1}}(h(V_2)))$ as evidence.

6. The judge checks the signature and orders Alice to pay up.

Moral: if someone asks you to sign a contract, add some random spaces before you do so.

Examples of Hash Functions

To withstand a birthday attack, hash length of 128 bits is recommended

1. A sequence MD2 (slow), MD4 (partially cryptanalysed), MD5 of 128 bits, by Ron Rivest, MIT
2. SHA (Secure Hash algorithm) a 160 bit hash designed by NIST and NSA, as part of DSS, variant of MD4
3. hashes derived from Block ciphers such as DES (tend to be too short), RSA (slow)

Schneier recommends SHA due to length, NSA cryptanalysis

Using Block Ciphers to create a hash

Break message M into blocks $M = M_1, M_2, \dots, M_n$

Define H_0 to be an initial random string

$$H_i = E_{H_{i-1}}(M_i) \oplus M_i$$

$$h(M) = H_n$$

Many variants, not all of them secure.

Combining benefits of Public and Private Keys

RSA doesn't require that A & B already have a shared key; but it's 1,000 times slower than DES (both in hardware)

Solution: Use public key cryptography to establish a shared key:

1. Alice generates a secret key K to be shared with Bob
2. A \longrightarrow B: E_{K_B} (the secret key is K)
3. A \longrightarrow B: $E_{K^{\text{shared}}}(M)$

Note

1. K is short, so encrypting it with a public key does not take too long
2. the long message M is encrypted/decrypted using faster shared key cryptography
(Used e.g. in PGP)

Combining Signatures with Encryption

Alice wants to send a signed message to Bob, but doesn't want Eve to learn the contents:

1. A \longrightarrow B: $E_{K_B}(S_{K_A^{-1}}(M))$
2. B computes $D_{K_B^{-1}}(E_{K_B}(S_{K_A^{-1}}(M))) = S_{K_A^{-1}}(M)$
3. B verifies the signature: $V_{K_A}(S_{K_A^{-1}}(M)) = M$.

Remark: why not encrypt, then sign?

i.e., why not A \longrightarrow B: $S_{K_A^{-1}}(E_{K_B}(M))$

1. if Bob were to try to convince somebody Alice signed M , he would have to reveal his private key.
2. Mallory could intercept this, extract $E_{K_B}(M)$, then M \longrightarrow B: $S_{K_M^{-1}}(E_{K_B}(M))$

(Example: M says "the answer to your puzzle is 42. I claim the prize")

3. a signature on text not comprehensible to the signer might not be legally valid

A Trap

Suppose Bob uses the same RSA key pair for encryption and signatures

A $(\rightarrow$ B) : $E_{K_B}(M)$

Mallory intercepts $E_{K_B}(M)$

M \rightarrow B: “the following is a DES encrypted message, sign it to confirm receipt and I will send you the key: $E_{K_B}(M)$ ”

B \rightarrow M: $S_{K_B^{-1}}(E_{K_B}(M))$

Mallory reads the signature part = $E_{K_B^{-1}}(E_{K_B}(M)) = M$

Moral: use different keys for signature and encryption, and know what you are signing

Computational Complexity and Cryptography

Computational Complexity

A *language* L over an alphabet Σ is a set of strings of the form $a_0a_1 \dots a_n$ where each $a_i \in \Sigma$, i.e., $L \subseteq \Sigma^*$.

The *decision problem* for L has the form:

Input: A string $x \in \Sigma^*$

Output: “yes” if $x \in L$, and “no” otherwise

If A is an algorithm for solving a decision problem, the *time complexity* of A is the function $f_A : \mathbf{N} \rightarrow \mathbf{N}$ such that $f_A(n)$ is the maximum number of steps taken by A on an input of length n

Polynomial Time

L is a *deterministic polynomial time* problem if there exists a deterministic algorithm A and a polynomial p such that for all $n \in \mathbf{N}$, $f_A(n) \leq p(n)$.

Also write $L \in P$

Nondeterministic Polynomial Time

L is a *nondeterministic polynomial time* problem if there exists a deterministic algorithm A and a polynomial p such that

1. A accepts L , and
2. on an input of length n , $A(x)$ halts in $\leq p(n)$ steps in all branches.

Also write $L \in NP$

Open Problem: Is $P = NP$?

Nondeterminism

A nondeterministic algorithm is like a standard algorithm, except that it may use the extra operation:

$x \leftarrow$ result of flipping a coin (H,T)

A *accepts* input x if $A(x) =$ "yes" for *some* outcome of the coin flips made while running $A(x)$

A *accepts* language L if $x \in L$ iff A accepts x , for all strings x

PTIME Reductions

Let L_1, L_2 be languages.

A *polynomial time reduction* from L_1 to L_2 is a function $f : \Sigma^* \rightarrow \Sigma^*$, computable in (deterministic) polynomial time, such that for all $x \in \Sigma^*$, we have $x \in L_1$ if $f(x) \in L_2$.

Proposition: If $L_2 \in P$ and f is a PTIME reduction from L_1 to L_2 then $L_1 \in P$.

NP-completeness

A problem L is *NP-complete* if

- L is in NP, and
- for all languages L' in NP, there exists a PTIME reduction from L' to L .

Thus, if L is NP-complete and $L \in P$, then all languages in NP are in P.

NP-complete languages exist, e.g.,

- SAT = the set of all satisfiable formulas of propositional logic
- Knapsack: the set of strings of the form $(n_1, n_2, \dots, n_k; N)$ where the n_i and N are numbers such that some subset of $\{n_1, n_2, \dots, n_k\}$ sums to N

The best known algorithms for NP complete problems take exponential time

Relevance of NP completeness to cryptography

We want encryption and decryption to be computable efficiently, e.g., given K, M , we want $E_K(M)$ to be computable in time polynomial in the size of K, M .

But then, the cryptanalyst who knows K , $E_K(M)$ can run the following NP algorithm to compute M :

1. flip $|M|$ coins to guess a value M' for M
2. compute $E_K(M')$
3. if $E_K(M') = E_K(M)$ then return M' , else fail.

So, if $P=NP$, then efficient public key cryptography cannot be secure!