HASH FUNCTIONS

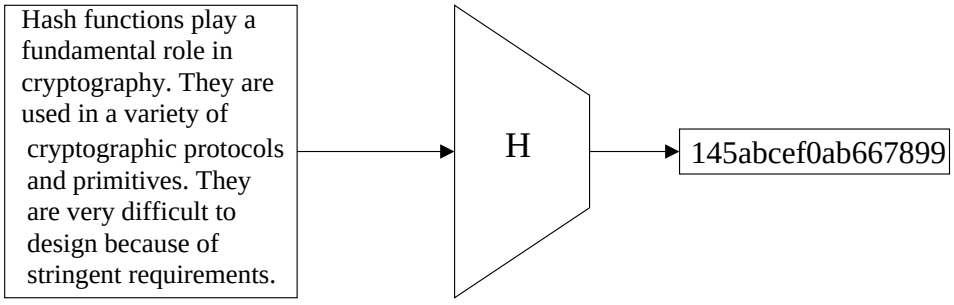
Outline

1. Definitions and Terminology
2. Applications of Hash Functions
3. Generic Attacks
4. Iterated Hash Functions
5. MDx-Family of Hash Functions
6. How to Exploit a Single Hash Collision

Definitions and Terminology

- Hash functions play a fundamental role in cryptography.
- They are used in a variety of cryptographic primitives and protocols.
- They are very difficult to design because of very stringent security and performance requirements.
- The only serious candidates available today are: SHA-1, RIPEMD-160, SHA-224, SHA-256, SHA-384 and SHA-512.

What is a Hash Function?



Definition of a Hash Function

- A *hash function* is a mapping H such that:
  (i) H maps inputs of arbitrary lengths to outputs of a fixed length n: $H : \{0,1\}^* \longrightarrow \{0,1\}^n$.
  (More generally, H maps elements of a set S to a set T where $|S| > |T|$.)
  (ii) $H(x)$ can be efficiently computed for all $x \in \{0,1\}^*$.
- H is called an *n-bit hash function*.
- $H(x)$ is called the *hash value*, *hash*, or *message digest* of x.

- Note: The description of a hash function is public. There are no secret keys.

Typical Cryptographic Requirements

- *Preimage resistance*: Given a hash value $y \in_R \{0,1\}^n$, it is computationally infeasible to find (with non-negligible probability of success) any input x such that $H(x) = y$.
  - x is called a *preimage* of y.
  - $y \in_R \{0,1\}^n$ means that y is chosen uniformly at random from $\{0,1\}^n$.
- *2nd preimage resistance*: Given an input $x \in_R \{0,1\}^*$, it is computationally infeasible to find (with non-negligible probability of success) a second input $x' \neq x$ such that $H(x) = H(x')$.
- *Collision resistance*: It is computationally infeasible to find (with non-negligible probability of success) two distinct inputs x, x' such that $H(x) = H(x')$.
  - The pair (x, x') is called a *collision* for H.

Some Relationships Between Properties

1. Collision res. implies 2nd preimage resistance. [Why?]
2. 2nd preimage resistance does not guarantee collision resistance. [Why?]
3. Collision resistance does not guarantee preimage resistance. Justification:
   - Let $H : \{0,1\}^* \to \{0,1\}^n$ be a collision-resistant hash function.
   - Consider $\overline{H} : \{0,1\}^* \to \{0,1\}^{n+1}$ defined by:

$$\overline{H}(x) = \begin{cases} 1 \,\|\, x, & \text{if } x \in \{0,1\}^n \\ 0 \,\|\, H(x), & \text{if } x \notin \{0,1\}^n. \end{cases}$$

   - Then $\overline{H}$ is collision-resistant (since H is). [Why?]
   - And $\overline{H}$ is not preimage-resistant because preimages can be easily found for (at least) half of all $y \in \{0,1\}^{n+1}$.

Some Relationships Between Properties (2)

However, if H is somewhat "uniform" (i.e., all hash values have roughly the same number of preimages), then the following argument shows that collision resistance of H does indeed guarantee preimage resistance:

Suppose that H is not preimage resistant. Select arbitrary $x \in \{0,1\}^*$ and compute $y = H(x)$. Find a preimage x' of y; this is successful with some negligible probability. Then, if H is uniform, we expect that $x' \neq x$ with very high probability. Thus, we have efficiently found a collision (x, x') for H, so H is not collision resistant.

Terminology

- A hash function that is preimage resistant is sometimes called a *one-way hash function* (OWHF).
- A hash function that is collision resistant is sometimes called a *collision-resistant hash function* (CRHF).
- A hash function that is both preimage resistance and collision resistant is called a *cryptographic hash function*.

Some Applications of Hash Functions

1. *Password protection* on a multi-user computer system:
   - Server stores $(\text{userid}, H(\text{password}))$ in a password file. Thus, if an attacker gets a copy of the password file, she does not learn any passwords.
   - Requires preimage-resistance.
2. *Modification Detection Codes* (MDCs).
   - To ensure that a message m is not modified by unauthorized means, one computes $H(m)$ and protects $H(m)$ from unauthorized modification.
   - e.g. virus protection.
   - Requires 2nd preimage resistance.

Some Applications of Hash Functions (2)

3. *Message digests for digital signature schemes*:
   - For reasons of efficiency, instead of signing a (long) message, the (much shorter) message digest is signed.
   - Requires preimage-resistance, 2nd preimage resistance and collision resistance. (More on this later)
   - To see why collision resistance is required:
     - Suppose that Alice can find two messages x_1 and x_2, with $x_1 \neq x_2$ and $H(x_1) = H(x_2)$.
     - Alice can sign x_1 and later claim to have signed x_2.

4. *Message Authentication Codes* (MACs). (More on this later)
   - Provides data integrity & data origin authentication.

Some Applications of Hash Functions (3)

5. *Pseudorandom bit generation*:
   - Distilling random bits s from several "random" sources $x_1, x_2, \ldots, x_t$.
   - Output $s = H(x_1, x_2, \ldots, x_t)$.

6. *Key derivation function* (KDF): Deriving a cryptographic key from a shared secret. (More on this later)

Notes:

- Collision resistance is not always necessary.
- Depending on the application, other properties may be needed, for example 'near-collision resistance', 'partial preimage resistance', ...

Generic Attacks

- A *generic* attack on hash functions $H : \{0,1\}^* \rightarrow \{0,1\}^n$ does not exploit any properties a specific hash function may have.
- In the *analysis* of a generic attack, we view H as a *random function* in the sense that for each $x \in \{0,1\}^*$, the value $y = H(x)$ was chosen by selecting y uniformly at random from $\{0,1\}^n$ (written $y \in_R \{0,1\}^n$).
- From a security point of view, a random function is an ideal hash function. However, random functions are not suitable for practical applications because they cannot be compactly stored.

Generic Attack for Finding Preimages

- Given $y \in \{0,1\}^n$, select arbitrary $x \in \{0,1\}^*$ until $H(x) = y$.
- Expected number of steps is $\approx 2^n$.
  (Here, a step is a hash function evaluation.)
- This attack is infeasible if $n \geq 80$.

Note: It has been proven that this generic attack for finding preimages is optimal, i.e., no better *generic* attack exists.

Generic Attack for Finding Collisions

- Select arbitrary $x \in \{0,1\}^*$ and store $(H(x), x)$ in a (conventional) hash table indexed by first column. Continue until a collision is found.
- Expected number of steps: $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$ (by birthday paradox). (Here, a step is a hash function evaluation.)
- It has been proven that this generic attack for finding collisions is optimal in terms of the number of hash function evaluations.
- Expected space required: $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$.
- This attack is infeasible if $n \geq 160$.
- If $n = 128$:
  - Expected running time: 2^{64} steps. [barely feasible]
  - Expected space required: 7×10^8 Tbytes. [infeasible]

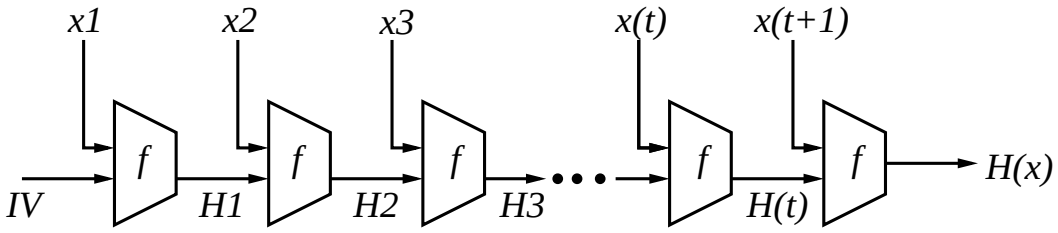
VW Parallel Collision Search

- Van Oorschot & Wiener (1993)
- See class notes for a description of the VW search algorithm.
- Very small space requirements.
- Easy to parallelize:
  - m-fold speedup with m processors.
- Described a 1996 US$10 million machine which can find collisions for 128-bit hash functions in 21 days. Collisions for 160-bit hash functions such as SHA-1 would take about 3500 years.
- The collision-finding algorithm can easily be modified to find “meaningful” collisions.

Iterated Hash Functions (Merkle's Meta-Method)

- Components:
  - Fixed *initializing value* $IV \in \{0,1\}^n$.
  - *Compression function* $f : \{0,1\}^{n+r} \longrightarrow \{0,1\}^n$.
- To compute $H(x)$ where x has bitlength $b < 2^r$ do:
  - Break up x into r-bit blocks: $\overline{x} = x_1, x_2, \ldots, x_t$, padding out the last block with 0 bits if necessary.
  - Define x_{t+1}, the *length-block*, to hold the right-justified binary representation of b.
  - Define $H_0 = IV$.
  - Compute $H_i = f(H_{i-1}, x_i)$ for $i = 1, 2, \ldots, t+1$.
  - H_i's are called *chaining variables*.
  - Define $H(x) = H_{t+1}$.

Collision Resistance of Iterated Hash Functions



- Theorem (Merkle): If the compression function f is collision resistant, then the function H is also collision resistant.
  Proof: See class notes.
- Merkle's theorem reduces the problem of finding collision-resistant hash functions to that of finding collision-resistant compression functions.

MDx-Family of Hash Functions

- Family of iterated hash functions.
- MD4 was proposed by Ron Rivest in 1990.
- 128-bit outputs.
- Dobbertin (1996) found “meaningful” collisions in a *few seconds* on a PC.
- Wang et al (2004) found collisions for MD4 *by hand*.
- Leurent (2008) discovered an algorithm for finding MD4 preimages in 2^{102} steps.

MD5 Hash Function

- MD5 is a strengthened version of MD4.
- Designed by Rivest in 1991.
- 128-bit outputs.
- UNIX command: “md5 filename”.
- Dobbertin (1996) found collisions for the MD5 compression function.
- Wang and Yu (2004) found collisions for MD5 in 2^{39} steps.
- Klima (2006) MD5 collisions is *31 seconds* on a notebook computer.

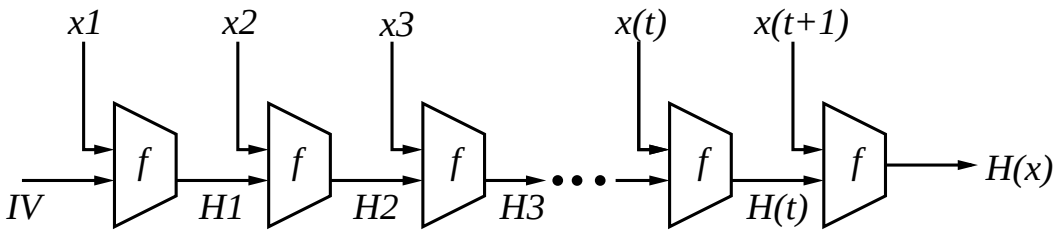
MD5 Hash Function (2)

- No preimage or 2nd preimage attacks are known on MD5.
- MD5 should not be used if collision resistance is required, but is probably okay as a one-way hash function.
- MD5 is widely used today (e.g. check root certs in IE)
  More on this later....
- MD5 shows up about 850 times in Windows source code.

SHA-1

- Secure Hash Algorithm (SHA) was designed by NSA and published by NIST in 1993 (FIPS 180).
- 160-bit hash function, based on MD4.
- Slightly modified to SHA-1 (FIPS 180-1) in 1994 in order to fix an (undisclosed) security weakness.
  - Wang et al (2005) found collisions for SHA in 2^{39} steps.
- Wang et al (2005) discovered a collision-finding algorithm for SHA-1 that takes 2^{63} steps.
  - No collisions for SHA-1 have been found as yet.
  - Search in progress: `http://boinc.iaik.tugraz.at/`
- No preimage or 2nd preimage attacks are known on SHA-1.

High-Level Description of SHA-1



- Iterated hash function (Merkle meta-method).
- $n = 160$, $r = 512$.
- Iteration function is $f : \{0,1\}^{160+512} \longrightarrow \{0,1\}^{160}$.
- Input: bitstring x of arbitrary bitlength $b \geq 0$.
- Output: 160-bit hash value $H(x)$ of x.

SHA-1 Notation

A, B, C, D, E	32-bit quantities.
$+$	addition modulo 2^{32}.
$\overline{A}$	bitwise complement.
$A \hookleftarrow s$	rotate A left through s positions.
AB	bitwise AND.
$A \vee B$	bitwise inclusive-OR.
$A \oplus B$	bitwise exclusive-OR.
$f(A, B, C)$	$AB \vee \overline{A}C$.
$g(A, B, C)$	$AB \vee AC \vee BC$.
$h(A, B, C)$	$A \oplus B \oplus C$.

SHA-1 Constants

- 32-bit initial chaining values (IVs):
  - h_1 = 0x67452301, h_2 = 0xefcdab89, h_3 = 0x98badcfe, h_4 = 0x10325476, h_5 = 0xc3d2e1f0.
- Per-round integer additive constants:
  - y_1 = 0x5a827999, y_2 = 0x6ed9eba1, y_3 = 0x8f1bbcdc, y_4 = 0xca62c1d6.

SHA-1 Preprocessing

- Pad x (with 1 followed by 0's) so that such that its bitlength is 64 less than a multiple of 512.
- Append a 64-bit representation of $b \bmod 2^{64}$.
- The formatted input is $x_0, x_1, \ldots, x_{16m-1}$, where each x_i is a 32-bit word.
- Initialize chaining variables: $(H_1, H_2, H_3, H_4, H_5) \leftarrow (h_1, h_2, h_3, h_4, h_5)$.

SHA-1 Processing

For each i from 0 to $m-1$ do the following:

- Copy the ith block of sixteen 32-bit words into temporary storage: $X_j \leftarrow x_{16i+j},\ 0 \le j \le 15$.
  Now process these as follows in four 20-step rounds before updating the chaining variables.
- Expand 16-word block into 80-word block:
  For j from 16 to 79,
  $X_j \leftarrow (\ X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \hookleftarrow 1$.
- Initialize working variables:
  $(A, B, C, D, E) \leftarrow (H_1, H_2, H_3, H_4, H_5)$.

Note: The rotate operation "$\hookleftarrow 1$" is not present in SHA. This is the only difference between SHA and SHA-1.

SHA-1 Processing (2)

- (*Round 1*) For j from 0 to 19 do:
  $t \leftarrow ((A \hookleftarrow 5) + f(B, C, D) + E + X_j + y_1)$,
  $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
- (*Round 2*) For j from 20 to 39 do:
  $t \leftarrow ((A \hookleftarrow 5) + h(B, C, D) + E + X_j + y_2)$,
  $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
- (*Round 3*) For j from 40 to 59 do:
  $t \leftarrow ((A \hookleftarrow 5) + g(B, C, D) + E + X_j + y_3)$,
  $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
- (*Round 4*) For j from 60 to 79 do:
  $t \leftarrow ((A \hookleftarrow 5) + h(B, C, D) + E + X_j + y_4)$,
  $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
- Update chaining values:
  $(H_1, H_2, H_3, H_4, H_5) \leftarrow (H_1+A, H_2+B, H_3+C, H_4+D, H_5+E)$.

SHA-1 Completion

Output: $H(x) = H_1 \| H_2 \| H_3 \| H_4 \| H_5$.

Open problem:
Find a message $x \in \{0,1\}^*$ such that SHA-1$(x) = 0$.

SHA-2

- In 2001, NSA proposed variable output-length versions of SHA-1.
- Output lengths are 224 bits (SHA-224), 256 bits (SHA-256), 384 bits (SHA-384) and 512 bits (SHA-512).
- The SHA-2 hash functions are standardized in FIPS 180-2.
- No weaknesses in SHA-224, SHA-256, SHA-384 or SHA-512 have been found.

Performance

[Wei Dai] `www.cryptopp.com/benchmarks.html`
Speed benchmarks for a software implementation on an Intel Core 2 1.83 GHz processor.

Algorithm	Speed (Mbits/sec)
DES	34
3DES	13
AES	99
RC4	128
MD5	258
RIPEMD-160	108
SHA-1	155
SHA-256	81
SHA-512	99

NIST's Present Policy on Hash Functions

- Stop using SHA-1 for digital signatures (and timestamping) as soon as possible.
- Must use SHA-2 for these applications after 2010.
- After 2010, may use SHA-1 only for HMAC, KDFs, and random number generators.
- All new applications and protocols are encouraged to use SHA-2.

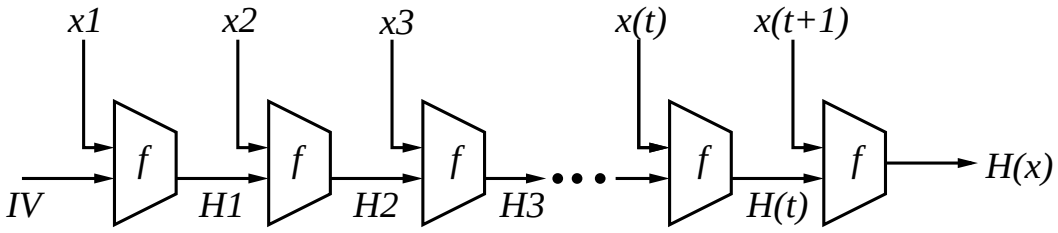
SHA-3

- The SHA-2 design is similar to SHA-1, and thus there are concerns that the SHA-1 weaknesses will extend to SHA-2.
- SHA-3: NIST hash function competition.
  - 64 candidates submitted by Oct 31 2008 deadline.
  - 51 were accepted for the first round.
  - (January 8, 2009): 8 candidates have been broken.
  - Feb 25-29 2009: First SHA-3 candidate conference where designers present their hash functions.
  - Encourage the public to study the hash functions.
  - Third quarter 2010: Select a list of finalists.
  - Encourage the public to study the finalists.
  - Second quarter 2012: Announce a winner.

How to Exploit a Single Hash Collision

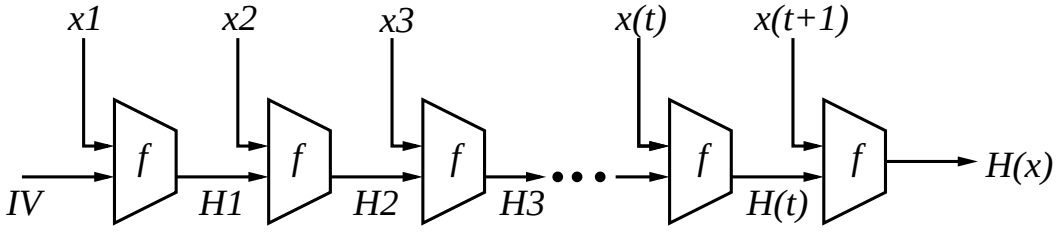
- Let H be a hash function (such as MD5 or SHA-1).
- Suppose that we can find two different messages x and y so that $H(x) = H(y)$.
- Suppose that the collision-finding method we use does not allow us to control the structure of x and y, so that these messages are essentially meaningless.
- Suppose also that the collision-finding methods takes considerable (but feasible) time.
- Question: How can an attacker, who has expended considerable resources to find two meaningless messages x and y that collide, make repeated use of this collision in a practical setting?

MD5 and SHA-1



- MD5 is a 128-bit iterated hash function with $r = 512$.
- SHA-1 is a 160-bit iterated hash function with $r = 512$.
- MD5 and SHA-1 have slightly different padding functions (and length-blocks) than in Merkle's meta-method.

Extending a Collision



- Notation: For a t-block message $x = (x_1, x_2, \ldots, x_t)$ and n-bit string I, define $F(I, x) = H_t$, where $H_0 = I$ and $H_i = f(H_{i-1}, x_i)$ for $i = 1, 2, \ldots, t$.
- Observation: Suppose x and y are two messages of the same block-length such that $F(I, x) = F(I, y)$. Then $F(I, x, z) = F(I, y, z)$ for *any* message z. Furthermore, if $I = IV$, then $H(x, z) = H(y, z)$ for any message z.

Wang's Collision-Finding Attack on SHA-1

- Fix any n-bit string I.
- Wang's attack finds two (different) 1-block messages $x = x_1$ and $y = y_1$ such that $F(I, x_1) = F(I, y_1)$.
- The attack gives limited, but not complete, control over x_1 and y_1.
- The attack takes about 2^{63} steps.
- By selecting $I = IV$ (where IV is the fixed initialization vector specified in SHA-1), Wang's attack can be used to find two one-block messages x and y such that SHA-1(x)=SHA-1(y). The attacker does not have much control over x and y, so these messages are essentially meaningless.

Wang's Collision-Finding Attack on MD5

- Fix any n-bit I.
- Wang's attack finds two (different) two-block messages $x = (x_1, x_2)$ and $y = (y_1, y_2)$ such that $F(I, x) = F(I, y)$.
- The attack gives limited, but not complete, control over x and y.
- The attack takes about 2^{39} steps.
- By selecting $I = IV$ (where IV is the fixed initialization vector specified in MD5), Wang's attack can be used to find two two-block messages x and y such that MD5(x)=MD5(y). The attacker does not have much control over x and y, so these messages are essentially meaningless.

Exploiting Wang's Collisions

- Suppose that MD5 is being used in a hash-then-sign signature scheme. [The example also works for SHA-1]
- Let M_1 and M_2 be two documents in *postscript* format. Suppose that M_1 is "harmless" for Alice, and M_2 is "harmful" for Alice.
- *Daum and Lucks* (2005) showed how Wang's attack on MD5 can be used to find two new postscript files $\widehat{M_1}$ and $\widehat{M_2}$ such that:
  1. When $\widehat{M_1}$ is viewed (using a standard postscript viewer) or printed (on a postscript printer), it looks the same as M_1. Similarly for $\widehat{M_2}$.
  2. MD5($\widehat{M_1}$) = MD5($\widehat{M_2}$).

Exploiting Wang's Collisions (2)

- The attacker Eve sends the postscript file $\widehat{M_1}$ to Alice. Alice views or prints the file, and then signs it and returns to Eve. Since MD5($\widehat{M_1}$) = MD5($\widehat{M_2}$), Eve also has Alice's signature on $\widehat{M_2}$.

Details of Daum and Lucks' Attack

The attacker Eve does the following:

1. Select a postscript "preamble" so that the string

   $$p = \text{“preamble (”}$$

   has bitlength a multiple of the block-length r. (If necessary, comments can be added as padding.)
2. Compute $I = F(IV, p)$.
3. Use Wang's attack to find two distinct two-block messages x and y such that $F(I, x) = F(I, y)$.
4. Select any two postscript files M_1 and M_2. Let T_1, T_2 be the postscript commands for displaying M_1 and M_2.

Details of Daum and Lucks' Attack (2)

5. Set $\widehat{M_1}$ = preamble $(x)(x)$eq{T_1}{T_2}
   and $\widehat{M_2}$ = preamble $(y)(x)$eq{T_1}{T_2}

Why this works:

1. MD5($\widehat{M_1}$)= MD5($\widehat{M_2}$)!!
2. Postscript interprets the commands $(S_1)(S_2)$eq{T_1}{T_2} as follows: If $S_1 = S_2$, then execute the commands T_1; else execute the commands T_2.
   Hence, if $\widehat{M_1}$ is viewed, then M_1 is displayed.
   If $\widehat{M_2}$ is viewed, then M_2 is displayed.

Notes on Daum and Lucks' Attack

- Of course, careful examination of the postscript files $\widehat{M_1}$ and $\widehat{M_2}$ would reveal Eve's deception – but noone reads raw postscript code before viewing or printing a postscript document.
- The collision x, y can be reused for any two postscript files M_1 and M_2.
- *Gebhardt, Illies and Schindler* (2005): Similar attacks on documents in PDF, TIFF and Word formats.
- Moral of this story: Don't be quick to dismiss attacks on cryptographic schemes, even if the attacks appear to have limited practical value.

Chosen-Prefix Collisions for MD5

- Lenstra, Stevens, de Weger; 2007
- Given *any* two messages M_1, M_2, one can find two suffixes S_1, S_2 such that MD5$(M_1, S_1) =$ MD5(M_2, S_2). The attack takes about 2^{50} valuations of the MD5 compression functions.
- Applications of the attack include:
  - Finding collisions for two execuatble files M_1, M_2.
  - Finding two X.509 certificates with different Names and Public Keys, but with the same signature.

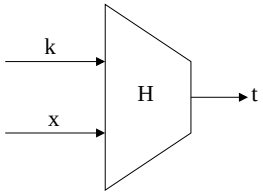
MESSAGE AUTHENTICATION CODE SCHEMES

Outline

1. Definition
2. Applications
3. Generic Attacks
4. MACs Based on Block Ciphers
5. MACs Based on Hash Functions
6. Key Escrow: The Clipper Chip

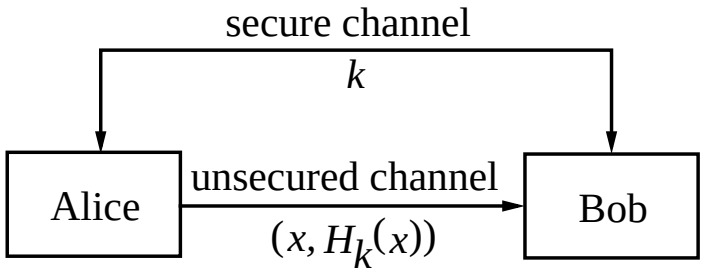
Definition

- A *message authentication code (MAC)* scheme is a family of functions $H_k : \{0,1\}^* \longrightarrow \{0,1\}^n$ parameterized by an l-bit key k, where each function H_k can be efficiently computed.
- $H_k(x)$ is called the *MAC* or *tag* of x.



- MAC schemes are used for providing (symmetric-key) data integrity and data origin authentication.

Applications of MAC Schemes



- To provide data integrity and data origin authentication:
  1. Alice and Bob establish a secret key $k \in \{0,1\}^l$.
  2. Alice computes $t = H_k(x)$ and sends (x,t) to Bob.
  3. Bob verifies that $t = H_k(x)$.
- Note: No confidentiality or non-repudiation.
- To avoid replay, add a timestamp, or sequence number.
- Widely used in banking applications.

Security Definition

- Let k be the secret key shared by Alice and Bob.
- The adversary does not know k, but is allowed to obtain (from Alice or Bob) tags for messages of her choosing. The adversary's goal is to obtain the tag of any message whose tag she did not already obtain from Alice or Bob.
- Definition: A MAC scheme is *secure* if given some MAC tags $H_k(x_i)$ for x_i's of one's own choosing, it is computationally infeasible to compute (with non-negligible probability of success) a pair $(x, H_k(x))$ for any new message x.
  - That is, the MAC scheme must be *existentially unforgeable against chosen-message attack*.
- Note: A secure MAC scheme can be used to provide data integrity and data origin authentication.

Generic Attacks

Guessing the MAC of a message x:

- Select $y \in \{0,1\}^n$ and guess that $H_k(x) = y$.
- Assuming that H_k is a random function, the probability of success is $1/2^n$.
- Note: Guesses cannot be directly checked.
- Depending on the application where the MAC algorithm is employed, one could choose n as small as 32 (say). In general, $n \geq 80$ is preferred.
  (cf. the Clipper Chip case study)

Generic Attacks (2)

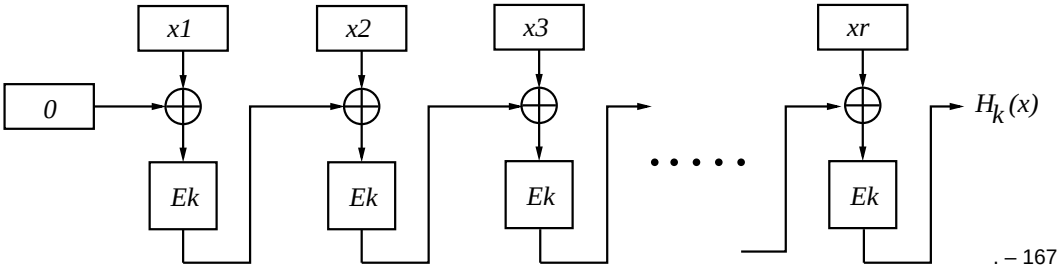
Exhaustive search on the key space:

- Given r known message-MAC pairs: $(x_1, t_1), \ldots, (x_r, t_r)$, one can check whether a guess k of the key is correct by verifying that $H_k(x_i) = t_i$, for $i = 1, 2, \ldots, r$.
- Assuming that the H_k's are random functions, the expected number of keys for which the tags verify is FK $= 2^l/2^{nr}$.
  - Example: If $l = 56$, $n = 64$, $r = 2$, then FK $= 1/2^{72}$.
- Expected number of steps is $\approx 2^l$.
- Exhaustive search is infeasible if $l \geq 80$.

MACs Based on Block Ciphers

CBC-MAC

- Let E be an n-bit block cipher with key space $\{0,1\}^l$.
- Assumption: Suppose that plaintext messages all have lengths that are multiplies of n.
- To compute $H_k(x)$:
  1. Divide x into n-bit blocks $x_1, x_2, \ldots, x_r$.
  2. Compute $H_1 = E_k(x_1)$.
  3. For $2 \leq i \leq r$, compute $H_i = E_k(H_{i-1} \oplus x_i)$.
  4. Then $H_k(x) = H_r$.



CBC-MAC (2)

- *DES CBC-MAC* (with $l = 56, n = 64$) is widely used in banking applications.
  - ANSI X9.9: Financial institution message authentication (wholesale).
  - ANSI X9.19: Financial institution message authentication (retail).
- Rigorous security analysis [Bellare, Kilian & Rogaway 1994]:
  Informal statement of a Theorem: Suppose that E is an "ideal" encryption scheme. (That is, for each $k \in \{0,1\}^l$, $E_k : \{0,1\}^n \to \{0,1\}^n$ is a 'random' permutation.) Then CBC-MAC *with fixed-length inputs* is a secure MAC algorithm.

Security of CBC-MAC

- CBC-MAC (as described on slide 159 without additional measures) is not secure if variable length messages are allowed.
- Here is a chosen-message attack on CBC-MAC:
  1. Let x_1 be an n-bit block.
  2. Let (x_1, t_1) be a known message-MAC pair.
  3. Request the MAC t_2 of the message t_1.
  4. Then t_2 is also the MAC of the 2-block message $(x_1, 0)$. (Since $t_2 = E_k(E_k(x_1))$.)

Encrypted CBC-MAC (EMAC)

- One solution for variable-length messages is *Encrypted CBC-MAC*:
  - Encrypt the last block under a second key s: $H_{k,s}(x) = E_s(H_r)$.
  - For example, ANSI X9.19 specifies that $H_{k,s}(x) = E_k(E_s^{-1}(H_r))$.
- Rigorous security analysis [Petrank & Rackoff 2000]: Informal statement of a Theorem: Suppose that E is an "ideal" encryption scheme. Then EMAC is a secure MAC algorithm (for inputs of any length).

MACs Based on Hash Functions

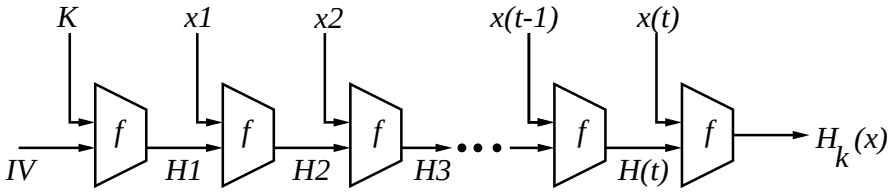
Hash functions were not originally designed for message authentication; in particular they are not "keyed" primitives.

Question: How to use them to construct secure MACs?

- Let H be an iterated n-bit hash function (without the length-block).
- Let $n + r$ be the input blocklength of the compression function $f : \{0,1\}^{n+r} \to \{0,1\}^n$.
  - Example: For SHA-1, $n = 160$, $r = 512$.
- Let $k \in \{0,1\}^n$.
- Let K denote k padded with $(r - n)$ 0's. (So K has bitlength r.)

Secret Prefix Method

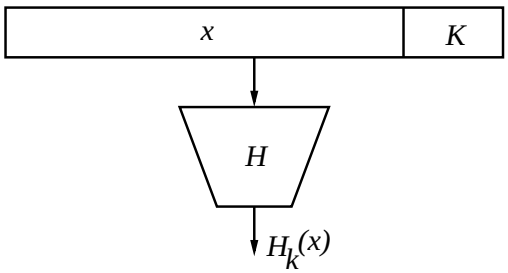
- MAC definition: $H_k(x) = H(K, x)$



- This is insecure. Here is a *length extension attack*:
  - Suppose that $(x, H_k(x))$ is known.
  - Suppose that the bitlength of x is a multiple of r.
  - Then $H_k(x \,\|\, y)$ can be computed for any y (without knowledge of k).
- Also insecure if a length block is postpended to $K \,\|\, x$ prior to application of H. [Exercise]

Secret Suffix Method

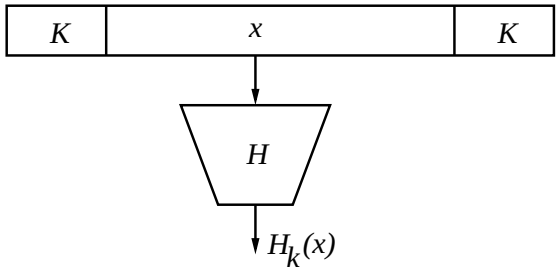
- MAC definition: $H_k(x) = H(x, K)$



- The attack on the secret prefix method does not work here.
- Suppose that a collision (x_1, x_2) can be found for H (i.e., $H(x_1) = H(x_2)$). We can assume that x_1 and x_2 both have bitlengths that are multiples of r.
  Thus $H(x_1, K) = H(x_2, K)$, and so $H_k(x_1) = H_k(x_2)$.
  Then the MAC for x_1 can be requested, giving the MAC for x_2.
  Hence if H is not collision resistant, then the secret suffix method MAC is insecure.

Envelope Method

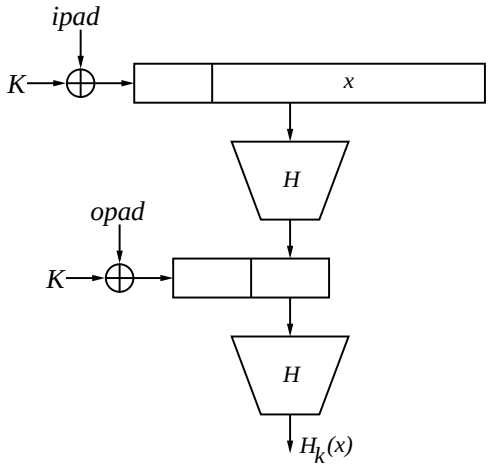
- The MAC key is used both at the start and end of the MAC computation.
- MAC definition: $H_k(x) = H(K, x, K)$



- The envelope method appears to be secure (i.e., no serious attacks have been found). However, the security analysis is ad-hoc — one cannot rule out future attacks.

HMAC

- "Hash-based" MAC; Bellare, Canetti & Krawczyk (1996).
- Define 2 r-bit strings (in hexadecimal notation): ipad = 0x36, opad = 0x5C; each repeated $r/8$ times.



- MAC definition: $H_k(x) = H(K \oplus \text{opad}, H(K \oplus \text{ipad}, x))$.

HMAC (2)

- Security analysis is rigorous:
  Informal statement of a Theorem: Suppose that the compression function used in H is a secure MAC with fixed length messages and a secret IV as the key. Then HMAC is a secure MAC algorithm.
- Recommended for practice: use SHA-1 as the hash function.
  - The weaknesses recently found in SHA-1 do not appear to affect the security of HMAC.
- HMAC is specified in IETF RFC 2104 and FIPS 198.
- HMAC is used in IPSEC (Internet Protocol Security) and SSL (Secure Sockets Layer)/TLS (Transport Layer Security).

Key Escrow: The Clipper Chip

- Webster definition of *escrow*: a deed, a bond, money, or a piece of property held in trust by a third party to be turned over to the grantee only upon the fulfillment of a condition.
- *Key escrow*: allows for the components of a secret encryption key to be deposited with one (or more) trusted parties (called *escrow agents*) in such a way that the escrow agents can, upon receiving proper authorization, recover the encryption key from the components.
- Uses of key escrow:
  - *Law enforcement*: Allows law enforcement authorities to decrypt intercepted encrypted data after obtaining the appropriate authorization from a court.
  - *Key recovery*: Allows users to recover data from ciphertext in the event that the encryption key is lost.

Clipper Chip

- The *Clipper Chip* is a US government family of cryptographic processors intended to protect unclassified but sensitive government and private-sector communications (phone conversations, data transmitted on computer networks, etc.).
- All transmitted data is encrypted in such a way that two escrow agents (e.g., FBI, Justice department) can cooperate to recover the encryption key and therefore decrypt the transmitted data (*wiretap*).
- Justification: Provides a balance between the security needs of individuals and the needs of law enforcement agencies.
- Its proposal in 1993 caused much controversy and debate (see `http://epic.org/crypto/clipper/`).

Clipper Chip Architecture

- Clipper uses the SKIPJACK symmetric-key encryption scheme which has a key length of 80 bits and a block length of 64 bits. [The SKIPJACK specification was made public in 1998]
- Each chip contains a unique 32-bit *device identifier* UID.
- Each chip contains a unique 80-bit *device key* KU.
  - KU is securely inserted into the chip during manufacture.
  - A random 80-bit string K1 is selected and K2 = K1 $\oplus$ KU is computed. K1 is securely distributed to one escrow agent; K2 to a second escrow agent. Both escrow agents must cooperate to recover KU.
- KF is an 80-bit *family key* (common to all chips) that is stored in each chip. KF is also known to law enforcement agencies.

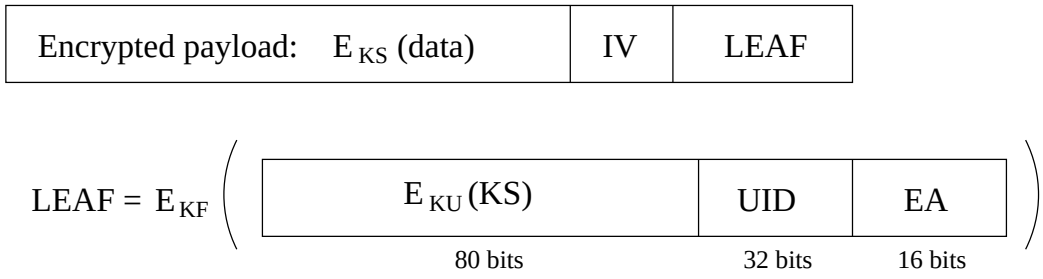
Clipper Chip Architecture (2)

- To encrypt data, an 80-bit *session key* KS is negotiated between the sender and receiver (using a variant of the Diffie-Hellman key agreement protocol), and then all transmitted data is encrypted with SKIPJACK (denoted by "E") under KS.
- Also appended to the ciphertext is a 64-bit IV, and a 128-bit *law enforcement access field (LEAF)*:

$$\mathsf{LEAF} = E_{\mathsf{KF}}(E_{\mathsf{KU}}(\mathsf{KS}) \,\|\, \mathsf{UID} \,\|\, \mathsf{EA}).$$

- Here, "EA" is a 16-bit "encryption authenticator" derived from the session key KS and the IV.

Clipper Chip Architecture (3)



EA = MAC(KS,IV)

The chip is *tamper-resistant*, so KF, KU and KS are unknown even to the legitimate user of the chip. Also, the MAC algorithm and the mode of operation used for LEAF encryption are classified.

A standard mode of operation (ECB, CBC, CFB, OFB) is used for encrypting the payload.

Legitimate Use of the Clipper Chip

- *Legitimate receipent*. The chip that receives $(E_{\mathsf{KS}}(\mathsf{data}), \mathsf{IV}, \mathsf{LEAF})$ first decrypts the LEAF using KF to obtain EA; and then computes EA'=MAC(KS,IV) and compares EA and EA'. It then uses KS to decrypt the encrypted payload only if EA=EA'.

- *Wiretaps*. A law enforcement agent intercepts $(E_{\mathsf{KS}}(\mathsf{data}), \mathsf{IV}, \mathsf{LEAF})$. It decrypts LEAF using KF to obtain UID and $E_{\mathsf{KU}}(\mathsf{KS})$. It then gets a court order and obtains K1 and K2 (identified by UID) from the escrow agents, and computes KU=K1$\oplus$K2. Finally it uses KU to decrypt $E_{\mathsf{KU}}(\mathsf{KS})$ thus obtaining KS, and then decrypts the encrypted payload using KS.

Illegitimate Use of the Clipper Chip

- Question: Can a user A use her Clipper chip to send messages to B (thus benefitting from the secure SKIPJACK encryption scheme, and also from the convenience of using available Clipper products) and at the same time defeat the key escrow mechanism (that is, prevent law enforcement from decrypting her data even if they obtain the appropriate court orders)?
- We require that B is an honest user. That is, unlike A, user B will not deviate from the legitimate protocol.
- Note that since the tamper-resistant hardware will not decrypt data unless it receives a valid LEAF, the above objective cannot be attained simply by not sending a LEAF.

Protocol Attack on the Clipper Chip

Observed by Matt Blaze in 1994.

1. A, B negotiate a session key KS using their Clipper chips.
2. A uses her Clipper chip to encrypt a message m using KS. The chip returns $(c, \mathsf{IV}, \mathsf{LEAF})$.
3. A selects a random 128-bit string v and feeds $(\mathsf{Test}, \mathsf{IV}, v)$ to her Clipper chip. This step is repeated until her chip accepts.
4. A then sends (c, IV, v) to B.
5. B's chip validates v and decrypts c.

The attack was successfully carried out using a standard interface to a prototype PCMCIA card implementation of Clipper chip functionality that was developed by NSA. No cryptanalysis or hardware reverse engineering was done.

Protocol attack on the Clipper Chip (2)

Notes on Blaze's attack:

1. Since v was randomly selected, its decryption under KF will with overwhelming probability not contain the correct UID or the field $E_{\mathsf{KU}}(\mathsf{KS})$. Thus the law enforcement agents will not be able to derive KS.
2. Assuming that the MAC algorithm is a good one, the probability that a randomly selected v is valid for a given KS and IV is $\frac{1}{2^{16}}$. Hence A has to try about 2^{16} v's before a valid one is obtained. This may be practical in some environments.
3. The main flaw is that the EA is too small (16 bits).

Required Reading: Sections 1, 3, 4, 5 of "Protocol failure in the escrowed encryption standard" by M. Blaze.