Grenoble University – M2 SCCI Security Proofs - JL Roch

Chapter 4

Cryptographic hash functions

References:

- A. J. Menezes, P. C. van Oorschot, S. A. Vanstone: Handbook of Applied Cryptography – Chapter 9 - Hash Functions and Data Integrity [pdf available]
- D Stinson: Cryprography – Theory and Practice (3rd ed), Chapter 4 – Security of Hash Functions
- S Arora and B Barak. Computational Complexity: A Modern Approach (2009). Chap 9. Cryptography (draft available) http://www.cs.princeton.edu/theory/complexity/ (see also Boaz Barak course http://www.cs.princeton.edu/courses/archive/spring10/cos433/)

Hash function

- Hash functions take a variable-length message and reduce it to a shorter *message digest* with fixed size (k bits)

  $h: \{0,1\}^* \rightarrow \{0,1\}^k$
- Many applications: "Swiss army knives" of cryptography:
  - Digital signatures (with public key algorithms)
  - Random number generation
  - Key update and derivation
  - One way function
  - Message authentication codes (with a secret key)
  - Integrity protection
  - code recognition (lists of the hashes of known good programs or malware)
  - User authentication (with a secret key)
  - Commitment schemes
- Cryptanalysis changing our understanding of hash functions
  - [eg Wang's analysis of MD5, SHA-0 and SHA-1 & others]

Hash Function Properties

- *Preimage resistant*
  - Given only a message digest, can't find any message (or *preimage*) that generates that digest. Roughly speaking, the hash function must be one-way.
- Second preimage resistant
  - Given one message, can't find another message that has the same message digest. An attack that finds a second message with the same message digest is a *second pre-image* attack.
    - It would be easy to forge new digital signatures from old signatures if the hash function used weren't second preimage resistant
- *Collision resistant*
  - Can't find any two different messages with the same message digest
    - Collision resistance implies second preimage resistance
    - Collisions, if we could find them, would give signatories a way to repudiate their signatures
  - Due to birthday paradox, k should be large enough !

- Collision_attack $\leq_P$ 2^{nd}-Preimage_attack

- Careful: Collision_resistance NOT$\leq_P$ Preimage_resistance
  - Let g : $\{0,1\}^* \rightarrow \{0,1\}^n$ be collision-resistant and preimage-resistant.
  - Let f: $\{0,1\}^* \rightarrow \{0,1\}^{n+1}$ defined by f(x):=if (|x|=n) then "0||x" else "1||g(x)".
  - Then f is collision resistant but not pre-image resistant.

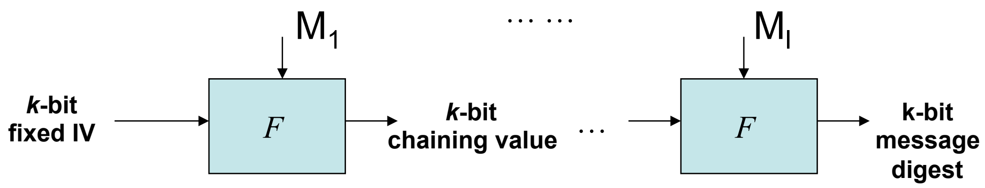
- But :
  (Collision_resistance and one way) $\Rightarrow_P$ Preimage_resistance

Building hash functions: ***compression + extension***

- Let F be a basic "***compression function***" that takes in input a block of fixed size (k+r bits) and delivers in ouptut a digest of size k bits :
  - For some fixed k and n, F "compresses" a block of n bits to one of k=n-r bits
    F: $\{0,1\}^{k+r} \rightarrow \{0,1\}^k$ (eg. for SHA2-384 k=384 bits and r=640 bits)

- ***One-to-one padding***: M → M || pad(M) to have a bit length multiple of r :
  - M || pad(M) = $M_1, M_2, M_3\ldots,M_l$ [one-to-one padding: M≠M' ⇔ M||pad(M) ≠ M"||pad(M')]
    - Ex.1: pad(M)="0…0"||s, where s=64 bits that encode the bitlength of M
    - Ex.2: pad(M)="0…0"||u||1||v, where u=bitlength(M) and v="0"$^{\log(u)}$

- F is extended to build h: $\{0,1\}^* \rightarrow \{0,1\}^k$
  based on a provable secure ***extension scheme***.
  - Eg: Merkle scheme: last output of compression function is the *h*-bit digest.



Provable **compression** functions

- **Example**: Chaum-van Heijst - Pfitzmann
  - two prime numbers q and p=2q+1.
  - α and β to primitive elements in F_p.
  - Compression function h_1

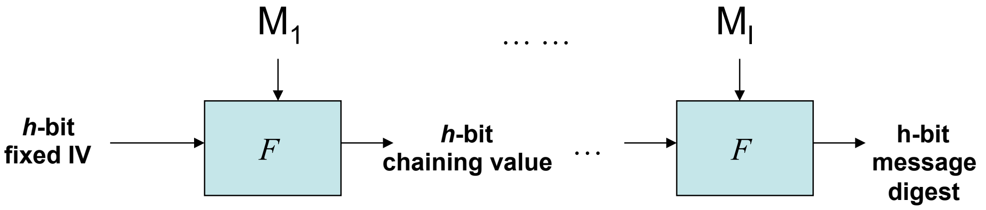
$$\begin{array}{llll} h_1 : & \mathbb{F}_q \times \mathbb{F}_q & \rightarrow & F_p \\ & (x_1, x_2) & \mapsto & \alpha^{x_1}.\beta^{x_2} \mod p \end{array}$$

- **Theorem**: If $LOG_\alpha(\beta)$ mod p is impossible to compute
  (i.e. to find x such that $\alpha^x=\beta$ mod p),
  then h_1 is resistant to collision.
  - Proof ?

  -> Training exercises (Form 4 : on the web): building a provable secure compression function F and a provable secure parallel extension scheme.

Provable **Extension** schemes

- Example: Merkle-Damgard scheme:
  - Preprocessing step: add padding to injectively make that the size of the input is a multiple of r: Compute the hash of x || Pad(x).



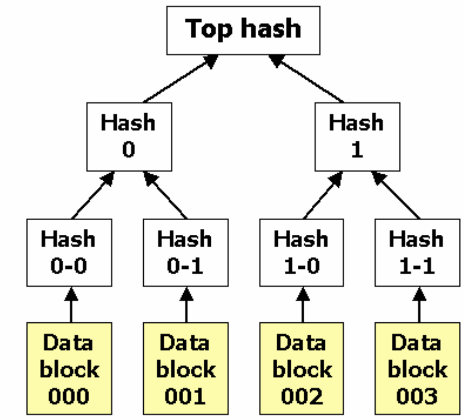
$$\mathbf{h_i = F\ (\ h_{i-1}\ ||\ x_i\ )}$$

- **Theorem**: If the compression function *F* is collision resistant then the hash function *h* is collision resistant .
  - Proof: by contradiction (reduction) and induction.

- Note: Drawback of Merkle-Damgard: pre-image and second preimage
  - There exist $O(2^{k-t})$ second-preimage attacks for 2^t-blocks messages [Biham&al. 2006]

Other extension schemes

- Merkle tree:

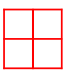


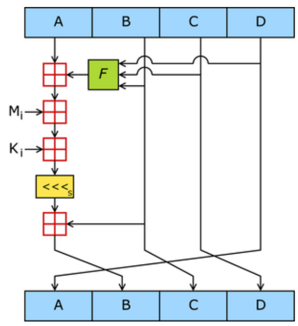
  - Variants: Truncated Merkle-tree, IV at each leave
- HAIFA : $h_i = F\ (\ h_{i-1}\ ||\ x_i\ ||\ i_{\textit{encoded on 64 bits}})$
  - where compression F: $\{0,1\}^{k+r+64} \rightarrow \{0,1\}^k$
  - Lower bound $W(2^k)$ for 2nd-preimage[Bouillaguet&al2010]
- …

NIST recommendations
[april 2006, Bill Burr]

	n	k	r	Unclassified use		Suite B	
				Through 2010	After 2010	Secret	Top Secret
MD4	512	128	384				
MD5	512	128	384				
SHA1	512	160	352	√			
SHA2-224	512	224	288	√	√		
SHA2-256	512	256	256	√	√	√	
SHA2-384	1024	384	640	√	√	√	√
SHA2-512	1024	512	512	√	√		

MD5

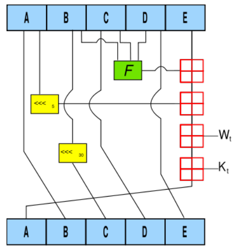
- The message is divided into blocks of n = 512 bits
  - Padding: to obtain a message of length multiple of 512 bits
    - $[B_1..B_k]$ => $[B_1..B_k10..0k_0..k_{63}]$
      where $[k_0..k_{63}]$ is the length k of the source (in 32 bits words)
- One step: 4 rounds of 16 operations of this type:
  - M_i plaintext (32 bits): 16*32=512 bits
  - A,B,C,D: current hash -or IV-: 4*32=128bits
  - K_i: constants
  - F: non linear box,  + mod 2^{32}



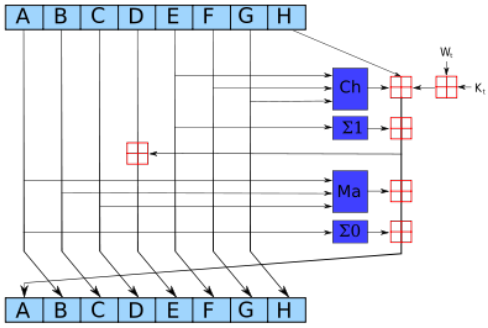
- First collisions found in 2004 [Wang, Fei, Lai,Hu]
  - No more security guarantees
  - Easy to generate two texts with the same MD5 hash

Secure Hash Algorithms SHA

- SHA1: n=512, k=160; 80 rounds with 32 bits words:
  - W_t plaintext (32 bits; 16*32=512 bits)
  - A,B,C,D,E: current hash -or IV-: 5*32=160bits
  - K_t: constants
  - F: non linear box, + mod 2^{32}
  - Weaknesses found from 2005
    - 2^{35} computations [BOINC…]



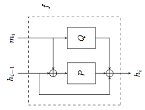
- SHA2: 4 variants: k=224/384/256/512
  - k=Size of the digest
  - SHA-256: n=512, k=256
    - 64 rounds with 32 bits words
    - Message length $<2^{64}-1$
    - SHA-224: truncated version
  - SHA-512: n=1024, k=512
    - 80 rounds with 64 bits words
    - Message length $<2^{128}-1$
    - SHA-384: truncated version

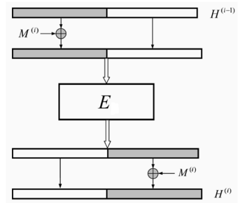


SHA-3 initial timeline (the Secure Hash Standard)

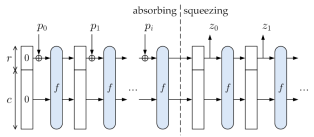
- **April 1995** FIPS 180-1: SHA-1 (revision of SHA, design similar to MD4)
- **August 2002** FIPS 180-2
  specifies 4 algorithms for 160 to 512 bits digest
  message size < 2^{64}: SHA-1, SHA-256 ; < 2^{128} : SHA-384, and SHA-512.
- **2007** FIPS 180-2 scheduled for review
  - **Q2- 2009** First Hash Function Candidate Conference
  - **Q2- 2010** Second Hash Function Candidate Conference
- **Oct 2008** FIPS 180-3 http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf
  specifies 5 algrithms for SHA-1, SHA-224, SHA-256, SHA-384, SHA-512.
- **2012**: Final Hash Function Candidate Conference
- **2 October 2012** : SHA-3 is **Keccak** (pronounced “catch-ack”).
  - Creators: Bertoni, Daemen, Van Assche (STMicroelectronics) & Peeters (NXP Semiconductors)

The five SHA3 finalists

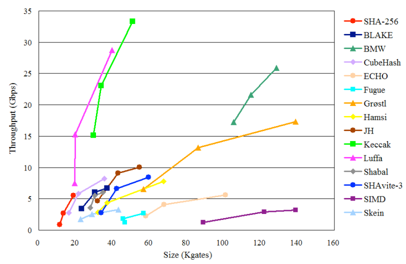
- BLAKE
  - New extension scheme (HAIFA) + stream cipher (Chacha)
- Grøstl
  - Compression function (two permutations)  +
    Merkle-Damgard extension + output transformation (Matyas-Meyer-Oseas)



- JH
  - New extension scheme + AES/Serpent cipher



- Keccak
  - Extension « sponge construction » + compression
- Skein
  - Extension « sponge construction » + Threefish block cipher



SHA-3 : Keccak

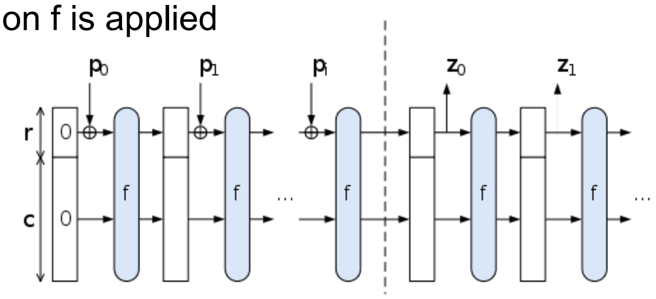
- Alternate, non similar hash function to MD5, SHA-0 and SHA-1:
  - Design : block permutation + Sponge construction
- But not meant to replace SHA-2
- Performance 12.5 cycles per byte on Intel Core-2 cpu; efficient hardware implementation.
- Principle (sponge construction):
  - message blocks XORed with the state which is then permuted (one-way one-to-one mapping)
  - State = 5x5 matrix with 64 bits words = 1600 bits
  - Reduced versions with words of 32, 16, 8,4,2 or 1 bit

Keccak block permutation

- Defined for $w = 2^\ell$ bit (w=64, ℓ = 6 for SHA-3)
- State = 5 x 5 x *w* bits array : notation: a[i, j, k] is the bit with index $(i\times5 + j)\times w + k$ (*arithmetic on i, j and k is performed mod 5, 5 and w*)
- block permutation function = $12+2\ell$ iterations of 5 subrounds :
  - *θ: xor each of the 5xw colums of 5 bits parity of its two neighbours* : $a[i][j][k] \oplus=$ parity$(a[0..4][j-1][k]) \oplus$ parity$(a[0..4][j+1][k-1])$
  - *ρ: bitwise rotate each of the 25 words by a different number, except a[0][0]* for all $0\le t\le24$, $a[i][j][\, k\,] = a[i][j][\, k-(t+1)(t+2)/2\,]$ with $\begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}^t \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
  - *π:* Permute the 25 words in a fixed pattern: $a[\,3i+2j\,][\,i\,] = a[i][j]$
  - *χ:* Bitwise combine along rows: $a[i][j][k] \oplus= \neg a[i][j+1][k]\ \&\ a[i][j+2][k]$
  - *ι:* xor a round constant into one word of the state. In round *n*, for $0\le m\le\ell$, $a[0][0][2^m-1] \oplus= b[m+7n]$ where b is output of a degree-8 LFSR.

Sponge construction = absorption+squeeze

- To hash variable-length messages by r bits blocks (c = 25w – r)
- Absorption:
  - The r input bits are XORed with the r leading bits of the state
  - Block function f is applied



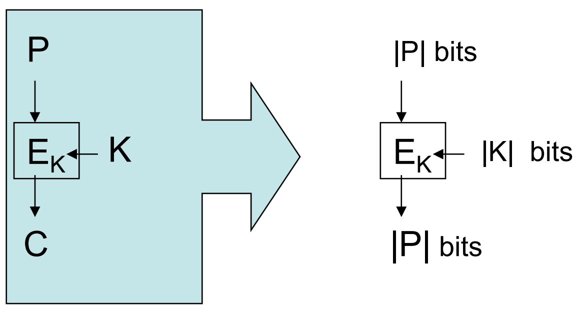
- Squeeze:
  - r first bits ot the states produced as outputs
  - Block permutation applied if additional output required
- « Capacity » : c = 25w-r bits not touched by input/output
  - SHA-3 sets c=2n where n = size of output hash (1 step squeeze only)
- Initial state = 0. Input padding = 10*1

Provable secure hash functions

- Due to birthday paradox, the expected number of k-bit hashes that can be generated before getting a collision is $2^{k/2}$
  - Security of a hash function with 128 bits digest cannot be more than 2^{64}
- Choose a provable secure compression function $F : \{0,1\}^{k+r} \rightarrow \{0,1\}^k$
  - eg Chaum-van Heijst-Pfitzmann (discrete logarithm, cf exrecise)
  - Or based on a (provably secure) symmetric block cipher E_K
    eg Matyas-Meyer-Oseas; Davies-Meyer; Miyaguchi-Preneel; Meyer-Shilling (MDC2)
  - Or …
- Choose a provable secure extension scheme to build h_F from F
  - Eg: Merkle scheme: $h_F(x \| b_1..b_r) = F(h(x) \| b_1..b_r)$ [cf course]
  - Or (usually when k=r) : $h_F(x \| y) = F(h_F(x) \| h_F(y))$ [cf exercise]
  - And use an initial value IV of k bits to initialize the scheme
    $h_F(b_1..b_r) = F(IV \| b_1..b_r)$

Building a compression function from a symmetric block cipher (1/3)

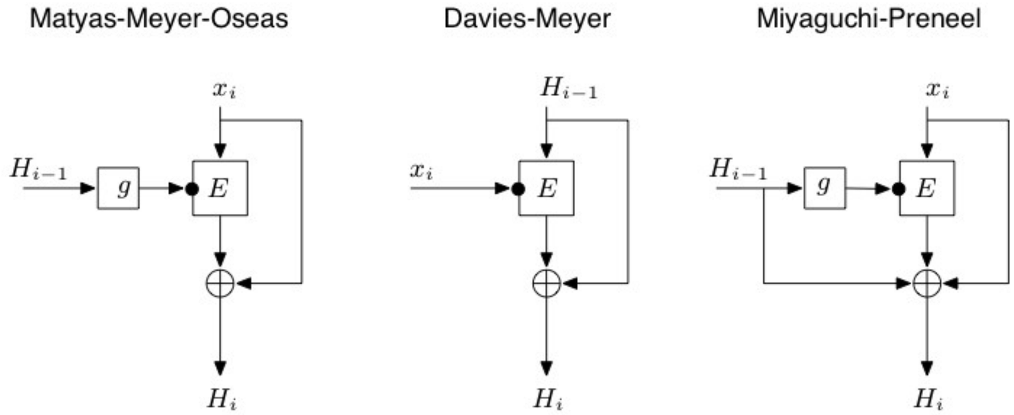
- Bloc cipher : [key K , plaintext P] -> ciphertext C with |C| = |P| < |C| + |P|
  -> Can be used as a compression function



- Expected number of operations to find a collision by brute force less than $2^{|P|/2}$
- But: a hash function is public, so is IV => cannot be used as is !

Building a compression function from a symmetric block cipher (2/3)

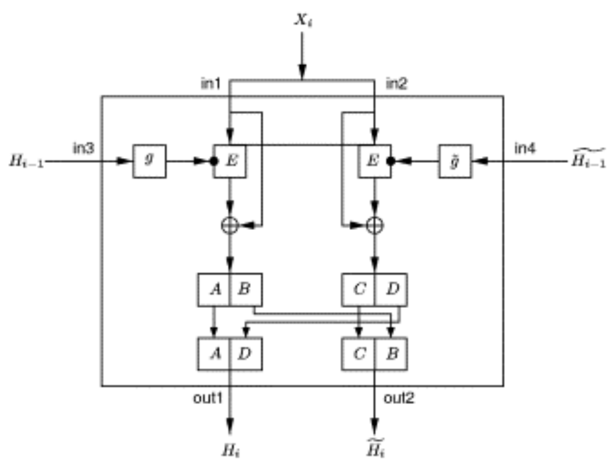
- Examples with a block cipher E with block size k and Merkle extension scheme :
  - g is a function that extends the hash to match the key size (might be identity)



- Theorem: Under the black-box model for the underlying block cipher, the 3 schemes are proved secure.
  Expected number of operations to find
  - a collision = $2^{k/2}$
  - a pre-image: 2^k

Building a compression function from a symmetric block cipher (3/3)

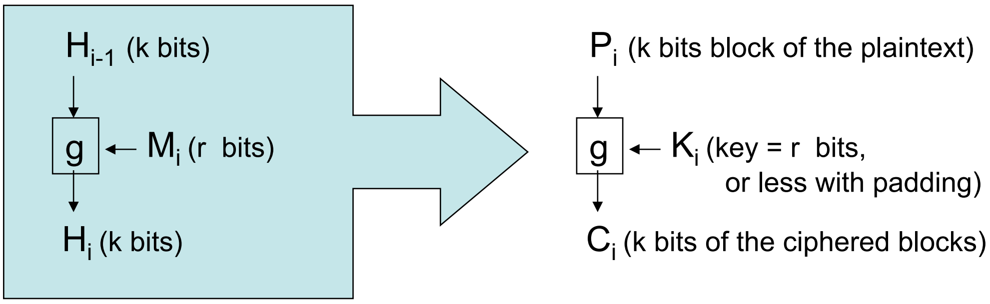
- Use of a block cipher with block size k to built a compression function with 2k digest
  - Examples: MDC-2 and MDC-4, based on Merkle extension scheme
- MDC2 :



- Theorem [Steinberger 2007]: Under the black-box model for the underlying block cipher, expected number of operations to find a collision ≥ $2^{3k/5}$
  - Better than 2 pre-image: $2^{k/2}$, even if far from the upper bound 2^k

Building a Block-cipher from hash function

- Building:
  Basic compression function   Block cipher



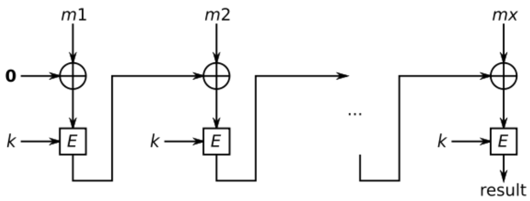
- Examples: SHACAL-1 (from SHA-1) SHACAL-2 (from SHA-256)

Other hash functions

- Based on modular arithmetic:
  - Eg MASH [Modular Arithmetic Secure Hash] based on RSA [MASH1: 1025 bits modulus -> 1024 bits digest
- Keyed hash functions :
  - Use a private key to build a hash
  - MAC (Message Authentication Code)
    - Based on a block cipher   - HMAC Based on a hash function

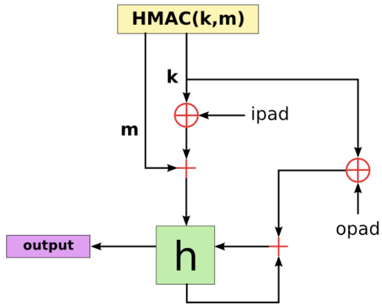
Keyed hash functions

- Use a private key to build a hash
  - MAC (Message Authentication Code)
- Examples:
  - Based on a block cipher - HMAC:based on a hash fn



CBC-MAC: based on CBC

$$C_{i+1} = E_k(C_i \oplus M_i)$$



$$\mathrm{HMAC}_K(m) = h\Big((K \oplus \mathrm{opad}) \| h\big((K \oplus \mathrm{ipad}) \| m\big)\Big)$$

What we have seen today

- Importance of hash function
- Hash function by compression + extension
  - Provable security
  - SHA1, SHA2
- SHA 3 : sponge construction
- Other hash functions :
  - Hash function built from sym. Cipher (and reverse)
  - Keyed hash function / HMAC
    [detailed construction at next lecture]

Hash functions : Security of MAC / HMAC

Outline

- Message Authentication Codes (MAC) and Keyed-hash Message Authentication Codes (HMAC)
- Keyed hash family
- Unconditionally Secure MACs

  - Ref: D Stinson: Cryprography – Theory and Practice (3rd ed), Chap 4.

Universal hash family

- Notations:
  - $\mathcal{X}$ is a set of possible messages
  - $\mathcal{Y}$ is a finite set of possible message digests or authentication tags
  - $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$
- Definition 4.1:
  A **keyed** hash family is a four-tuple $\mathcal{F} = (\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where the following condition are satisfied:
  - $\mathcal{K}$, the **keyspace**, is a finite set of possible keys
  - $\mathcal{H}$, the **hash family**, a finite set of at most $|\mathcal{K}|$ hash functions.
    For each $K \in \mathcal{K}$, there is a hash function $h_K \in \mathcal{H}$. Each $h_k: \mathcal{X} \rightarrow \mathcal{Y}$
- Compression function:
  - $\mathcal{X}$ is a finite set, $N=|\mathcal{X}|$. Eg $\mathcal{X} = \{0,1\}^{k+r}$ $N = 2^{k+r}$
  - $\mathcal{Y}$ is a finite set $M=|\mathcal{Y}|$. Eg $\mathcal{Y} = \{0,1\}^r$ $M=2^r$
  - $|\mathcal{F}^{\mathcal{X},\mathcal{Y}}| = M^N$
  - $\mathcal{F}$ is denoted (N,M)-hash family

Random Oracle Model

– Model to analyze the probability of computing preimage, second pre-image or collisions:
– In this model,
  - a hash function h_K: $\mathcal{X} \rightarrow \mathcal{Y}$ is chosen randomly from $\mathcal{F}$
  - The only way to compute a value $h_K(x)$ is to query the oracle.

– THEOREM 4.1
Suppose that $h \in \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is chosen randomly, and let $\mathcal{X}_0 \subseteq \mathcal{X}$. Suppose that the values h(x) have been determined (by querying an oracle for h) if and only if $x \in \mathcal{X}_0$.

Then, for all $x \in \mathcal{X} \setminus \mathcal{X}_0$ and all $y \in \mathcal{Y}$,
$$\Pr[h(x)=y] = 1/M$$

Algorithms in the Random Oracle Model

– Randomized algorithms make random choices during their execution.

– A Las Vegas algorithm is a randomized algorithm
  - may fail to give an answer
  - if the algorithm returns an answer, then the answer must be correct.

– A randomized algorithm has average-case **success** probability **ε** if the probability that the algorithm returns a correct answer, averaged over all problem instances of a specified size , is at least ε ($0 \leq \varepsilon < 1$).

For all x (randomly chosen among all inputs of size s):
$$\Pr(\text{ Algo}(x) \text{ is correct}) \geq \varepsilon$$

– (ε,q)-algorithm : terminology to design a Las Vegas algorithm such that:
  - the average-case success probability ε
  - the number of oracle queries made by algorithms is at most q.

Example of (ε,q)-algorithm

- Algorithm 4.1: FIND PREIMAGE (h, y, q)
  - choose any $\mathcal{X}_0 \subseteq \mathcal{X}, |\mathcal{X}_0| = q$
  - **for each** x $\in \mathcal{X}_0$ **do** { **if** h(x) = y **then return** (x) ; }
  - **return** (failure)

- THEOREM 4.2 For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0| = q$, the average-case success probability of Algorithm 4.1 is **ε=1 - (1-1/M)^q**. Algorithm 4.1 is a (**1 - (1-1/M)^q** ; **q**) – algorithm

- **Proof** Let y $\in \mathcal{Y}$ be fixed. Let $X_0 = \{x_1, x_2 .., x_q\}$.
  The Algo is successful iff there exists i such that $h(x_i) = y$.
- For $1 \le i \le q$, let E_i denote the event "$h(x_i) = y$".
  The E_i's are independent events; from Theo. 4.1, $Pr[E_i] = 1/M$ for all $1 \le i \le q$.
  Therefore, $$\Pr[E_1 \vee E_2 \vee \ldots \vee E_q] = 1 - \left(1 - \frac{1}{M}\right)^q$$
  The success probability of Algorithm 4.1, for any fixed y, is constant.
  Therefore, the success probability averaged over all y $\in \mathcal{Y}$ is identical, too.

Message Authentication Codes

- One common way of constructing a MAC is to incorporate a secret key into an unkeyed hash function.

- Suppose we construct a keyed hash function h_K from an unkeyed iterated hash function h,
  by defining IV=K and keeping this initial value secret.

- Attack: the adversary can easily compute hash without knowing K (so IV) with a (1-1)–algorithm:
  - Let r = size of the blocks in the iterated scheme
  - Choose x and compute y = h (x) (one oracle call)
  - Let x'= x || pad(x) || w, where w is any bitstring of length r
    Let x' || pad(x') = x || pad(x) || w || pad(x') (since padding is known)
  - Compute y' = IteratedScheme(y, w || pad(x')) (iterated scheme is known)
  - Return (x', y') which is a valid pair ; (we have y'=h(x'))

Message Authentication Codes (ε,q)-forger

– Assume MD iterated scheme is used, let $z_r = h_K(x)$
The adversary computes $z_{r+1} \leftarrow compress(h_K(x)||y_{r+1})$
$z_{r+2} \leftarrow compress(z_{r+1}||y_{r+2})$
…
$z_{r'} \leftarrow compress((z_{r'-1} || y_{r'})$
and returns $z_{r'}$ that verifies $z_{r'}=h_K(x')$.

- Def: an **(ε,q)-forger** is an adversary who
  – queries message $x_1,\ldots,x_q$,
  – gets a valid (x, y), $x \notin \{x_1,\ldots,x_q\}$
  – with a probability at least ε that the adversary outputs a **forgery** (ie a correct couple (x, h(x))

Hash functions : Security of MAC / HMAC

Outline

- *Message Authentication Codes*
  – *Intoduction. Choosing K=IV isn't a good idea.*
- **Keyed hash family**
  – **Security proof for nested HMAC**
- Unconditionally Secure MACs

Nested MACs and HMAC

- A nested MAC builds a MAC algorithm from the composition of two hash families
  - $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{G})$, $(\mathcal{Y},\mathcal{Z},\mathcal{L},\mathcal{H})$
  - composition: $(\mathcal{X},\mathcal{Z},\mathcal{M},\mathcal{G} \circ \mathcal{H})$
  - $\mathcal{M} = \mathcal{K} \times \mathcal{L}$
  - $\mathcal{G} \circ \mathcal{H} = \{ g \circ h: g \in \mathcal{G}, h \in \mathcal{H} \}$
  - $(g \circ h)_{(K,L)}(x) = g_K(h_L(x))$ for all $x \in \mathcal{X}$
- **Theorem: the nested MAC is secure if**
  - $(\mathcal{Y},\mathcal{Z},\mathcal{L},\mathcal{H})$ is secure as a MAC, given a fixed key
  - $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{G})$ is collision-resistant, given a fixed key

Nested MACs and HMAC Security proof with 3 adversaries

- (1) a forger for the nested MAC (**big MAC attack**)
  - (K,L) is chosen and kept secret
  - The adversary chooses x and query a big (nested) MAC oracle for values of $g_K(h_L (x))$
  - **output (x’,z) such that $z = g_K(h_L (x’))$** (x’ was not query)
- (2) a forger for the little MAC (**little MAC attack**) $(\mathcal{Y},\mathcal{Z},\mathcal{L},\mathcal{H})$
  - L is chosen and kept secret
  - The adversary chooses y and query a little MAC oracle for values of $h_L(y)$
  - **output (y’,z) such that $z = h_L(y’)$** (y’ was not query)

Nested MACs and HMAC Security proof with 3 adversaries

- (3) a collision-finder for the hash function $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{G})$, when the key is secret (unknown-key collision attack) i.e. a collision finder for the hash function $\mathbf{g_K}$
  - K is secret
  - The adversary chooses x and query a hash oracle for values of $g_K(x)$
  - output x', x'' such that $x' \neq x''$ and $g_K(x') = g_K(x'')$

Nested MACs and HMAC Security proof

- THEOREM 4.9 Suppose $(\mathcal{X},\mathcal{Z},\mathcal{M},\mathcal{G}\circ\mathcal{H})$ is a nested MAC.

  (3) Suppose there does not exist an $(\varepsilon_1,q+1)$-collision attack for a randomly chosen function $g_K \in \mathcal{G}$, when the key K is secret.

  (2) Further, suppose that there does not exist an (ε_2,q)-forger for a randomly chosen function $h_L \in \mathcal{H}$, where L is secret.

  (1) Finally, suppose there exists an (ε,q)-forger for the nested MAC, for a randomly chosen function $(g\circ h)_{(K,L)} \in \mathcal{G}\circ\mathcal{H}$.

  Then $\varepsilon \leq \varepsilon_1+\varepsilon_2$

Proof

- From (1) Adversary queries $x_1,..,x_q$ to a big MAC oracle and get $(x_1, z_1)..(x_q, z_q)$.
  It outputs a [possibly] valid (x, z) with Prob [$z=(g^\circ h)_{(K,L)}(x)$] = ε

- With previous x, $x_1,.., x_q$ make q+1 queries to a hash oracle g_K :
  $$y = g_K(x),\ y_1 = g_K(x_1),...,\ y_q = g_K(x_q)$$

- if $y \in \{y_1,..,y_q\}$, say $y = y_i$, then x, x_i is solution to Collision; from (3), the probability of forging such a collision is ε_1.
- else, output (y, z) which is a [possibly] forgery for h_L with probability $\geq \varepsilon-\varepsilon_1$.

- Besides, q (indirect) little MAC queries have been performed for$(y_1,z_1), ..., (y_q,z_q)$. From (2), (y,z) is a [possibly] forgery for h_L with probability $\leq\varepsilon_2$.

- Finally, little MAC attack probability is $\geq \varepsilon-\varepsilon_1$and $\leq \varepsilon_2$: thus $\varepsilon-\varepsilon_1 \leq \varepsilon_2 \Rightarrow \varepsilon\leq\varepsilon_1+\varepsilon_2$. □

Nested MACs and HMAC

- **HMAC** is a nested MAC algorithm that is proposed by FIPS standard
  - for MD5 and SHA1 : [RFC 2202]

- $HMAC_K(x)$ = SHA-1((K ⊕ opad) || SHA-1((K ⊕ ipad) || x))

  - x is a message
  - K is a 512-bit key
  - ipad = 3636…..36 (512 bit)
  - opad = 5C5C….5C (512 bit)

CBC-MAC(x, K)

A popular way to construct a MAC using a block cipher $\mathbf{E_K}$ with secret key K :

Cryptosystem 4.2: CBC-MAC (x, K)

- denote $x = x_1 || \ldots || x_n$, x_i is a bitstring of length t
- IV $\leftarrow$ 00..0 (t zeroes)
- $y_0 \leftarrow$ IV
- **for** i $\leftarrow$ 1 **to** n

  do $y_i \leftarrow \mathbf{E_K}(y_{i-1} \oplus x_i)$
- **return** (y_n)

CBC-MAC(x, K)
Birthday collision attack

- **(1/2, O($2^{t/2}$))-forger attack**
  - $n \geq 3$, $q \approx 1.17 \times 2^{t/2}$
  - $x_3, \ldots, x_n$ are fixed bitstrings of length t.
  - choose any q distinct bitstrings of length t,

    $x_1^1, \ldots, x_1^q$, and randomly choose $x_2^1, \ldots, x_2^q$
  - define $x_l^i = x_l$, for $1 \leq i \leq q$ and $3 \leq l \leq n$
  - define $x^i = x_1^i || \ldots || x_n^i$ for $1 \leq i \leq q$
  - $x^i \neq x^j$ if $i \neq j$, because $x_1^i \neq x_1^j$.
  - The adversary requests the MACs of $x^1, x^2, \ldots, x^q$

CBC-MAC(x, K)

- In the computation of MAC of each x^i, values $y_0^i \cdots y_n^i$ are computed, and y_n^i is the resulting MAC. Now suppose that and x^i and x^j have identical MACs.
- $h_K(x^i) = h_K(x^j)$ if and only if $y_2^i = y_2^j$, which happens if and only if $y_1^i \oplus x_2^i = y_1^j \oplus x_2^j$.
- Let x_δ be any bitstring of length t
  - $v = x_1^i \,||\, (x_2^i \oplus x_\delta) \,||\ldots||\, x_n^i$
  - $w = x_1^j \,||\, (x_2^j \oplus x_\delta) \,||\ldots||\, x_n^j$
- The adversary requests the MAC of v
- It is not difficult to see that v and w have identical MACs, so the adversary is successfully able to construct the MAC of w, i.e. $h_K(w) = h_K(v)$!!!

Hash functions : Security of MAC / HMAC

Outline

- *Message Authentication Codes*
  - *Intoduction. Choosing K=IV isn't a good idea.*
- *Keyed hash family*
  - *Security proof for nested HMAC*
- **Unconditionally Secure MACs**

Unconditionally Secure MACs

- **Unconditionally secure MACs**
  - a key is used to produce only one authentication tag
  - Thus, an adversary makes at most one query.
- **Deception probability Pd_q**
  - maximum value of ε such that (ε,q)-forger for q = 0, 1
- **payoff** (x, y) = probability of a vaild pair (x, y=h_{K0}(x)) :

$$\Pr[y = h_{K0}(x)] = \frac{|\{K \in \mathcal{K} : h_K(x) = y\}|}{|\mathcal{K}|}$$

- **Impersonation attack** ((ε,0)-forger)
  - Pd_0 = max{ payoff(x,y): x ∈ $\mathcal{X}$, y ∈ $\mathcal{Y}$}   (4.1)

Unconditionally Secure MACs

- **Substitution attack** ((ε,1)-forger)
  - query x and y is reply, x ∈ $\mathcal{X}$, y ∈ $\mathcal{Y}$
  - Probability(x', y') is valid = payoff(x',y';x,y), x' ∈ $\mathcal{X}$ and x ≠ x'
  - payoff(x',y';x,y) = Pr[y' = h_{K0}(x')) | y = h_{K0}(x)] =

$$\frac{\Pr[y' = h_{K0}(x') \wedge y = h_{K0}(x)]}{\Pr[y = h_{K0}(x)]} = \frac{|\{K \in \mathcal{K} : h_K(x') = y', h_K(x) = y\}|}{|\{K \in \mathcal{K} : y = h_K(x)\}|}$$

  - Let $\mathcal{V}$ = {(x, y): | {K ∈ $\mathcal{K}$: h_K(x) = y} | ≥1}
  - Pd_1 = max{ payoff(x', y'; x, y): x, x' ∈ $\mathcal{X}$, y, y' ∈ $\mathcal{Y}$, (x,y) ∈ $\mathcal{V}$, x ≠ x'}   (4.2)

Unconditionally Secure MACs

- Example 4.1 $\mathcal{X} = \mathcal{Y} = Z_3$ and $\mathcal{K} = Z_3 \times Z_3$
  for each $K = (a,b) \in \mathcal{K}$ and each $x \in \mathcal{X}$,
  $h_{(a,b)}(x) = ax + b \bmod 3$
  $\mathcal{H} = \{h_{(a,b)}: (a,b) \in Z_3 \times Z_3\}$
  - $Pd_0 = 1/3$
  - query x = 0 and answer y = 0
    possible key $K_0 \in \{(0,0),(1,0),(2,0)\}$.
    The probability that K_0 is key is 1/3
    $Pd_1 = 1/3$

    But if (1,1) is valid then $K_0 = (1,0)$

Key / x	0	1	2
(0,0)	0	0	0
(0,1)	1	1	1
(0,2)	2	2	2
(1,0)	0	1	2
(1,1)	1	2	0
(1,2)	2	0	1
(2,0)	0	2	1
(2,1)	1	0	2
(2,2)	2	1	0

Authentication matrix

Strongly Universal Hash Families

- Definition 4.2: Suppose that $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$ is an (N,M) hash family.
  This hash family is **strongly universal** provided that the following condition is satisfied :

  for every x, x' $\in \mathcal{X}$ such that $x \neq x'$, and for every y, y' $\in \mathcal{Y}$:

$$|\{K \in \mathcal{K}: h_K(x) = y, h_K(x') = y'\}| = |K|/M^2$$

- Example 4.1 is a strongly universal (3,3)-hash family.

Unconditionally Secure MACs

- LEMMA 4.10 Suppose that $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$ is a strongly universal (N,M)-hash family.
  Then for every x $\in \mathcal{X}$ and for every y $\in \mathcal{Y}$
  $$|\{K \in \mathcal{K} : h_K(x) = y\}| = |\mathcal{K}|/M.$$

- **Proof** x, x' $\in \mathcal{X}$ and y $\in \mathcal{Y}$, where x ≠ x'
  $$|\{K \in \mathcal{K} : h_K(x) = y\}| = \sum_{y' \in \mathcal{Y}} |\{K \in \mathcal{K} : h_K(x) = y, h_K(x') = y'\}| \quad \square$$
  $$= \sum_{y' \in \mathcal{Y}} \frac{|\mathcal{K}|}{M^2} = \frac{|\mathcal{K}|}{M}$$

Unconditionally Secure MACs

- THEOREM 4.11 Suppose that $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$ is a strongly universal (N,M)-hash family. Then $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$ is an authentication code with $Pd_0 = Pd_1 = 1/M$

- **Proof** From Lemma 4.10
  payoff(x,y) = 1/M for every x $\in \mathcal{X}$ and y $\in \mathcal{Y}$, and $Pd_0 = 1/M$
  x,x' $\in \mathcal{X}$ such that x ≠ x' and y,y' $\in \mathcal{Y}$, where (x,y) $\in \mathcal{V}$

  payoff(x',y';x,y)= $\dfrac{|\{K \in \mathcal{K} : h_K(x') = y', h_K(x) = y\}|}{|\{K \in \mathcal{K} : h_K(x) = y\}|}$
  $$= \frac{|\mathcal{K}|/M^2}{|\mathcal{K}|/M} = \frac{1}{M}$$

  Therefore $Pd_1 = 1/M$

Unconditionally Secure MACs

- THEOREM 4.12 Let p be prime.
  For a, b $\in Z_p$, let $f_{a,b}$: $Z_p \rightarrow Z_p$ with $f_{(a,b)}(x) = ax + b \bmod p$.
  Then (Z_p, Z_p, $Z_p \times Z_p$, $\{f_{a,b}: Z_p \rightarrow Z_p\}$) is a strongly universal (p,p)-hash family.

- **Proof** x, x', y, y' $\in Z_p$, where $x \neq x'$.
  $ax + b \equiv y \pmod p$, and $ax' + b \equiv y' \pmod p$
  $a = (y-y')(x'-x)^{-1} \bmod p$, and
  $b = y - x(y'-y)(x'-x)^{-1} \bmod p$
  (note that $(x' - x)^{-1} \bmod p$ exists because x !≡ x' (mod p) and p is prime) □

Unconditionally Secure MACs

- THEOREM 4.13 Let l be a positive integer and let p be prime. Define $\mathcal{X} = \{0,1\}^l \setminus \{(0,\ldots,0)\}$
  For every $r \in (Z_p)^l$, define $f_r: \mathcal{X} \rightarrow Z_p$ by :
  $f_r(x) = \langle r , x \rangle = \Sigma_{i=1,\ldots,l}\, r_i \,.\, X_i \mod p$

Then ($\mathcal{X}$, Z_p, $(Z_p)^l$, $\{f_r : r \in (Z_p)^l\}$) is a strongly universal (2^l - 1,p)-hash family.

Unconditionally Secure MACs

- **Proof** Let $x, x' \in X$, $x \neq x'$, and let $y, y' \in Zp$.
  Show that the number of vectors $\vec{r} \in (Z_p)^l$ such that $r.x \equiv y \pmod p$ and $r.x' \equiv y' \pmod p$ is p^{l-2}.
  The desired vector r are the solution of two linear equations in l unknowns over Z_p.
  The two equations are linearly independent, and so the number of solution to the linear system is p^{l-2}.
  Then $|\{K \in \mathcal{K}: h_K(x) = y, h_K(x') = y'\}| = p^{l-2} = |K|/M^2$.

□

Unconditionally Secure MACs

- 4.5.2 Optimality of Deception Probabilities
  - THEOREM 4.14 Suppose $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$ is an (N, M)-hash family. Then $Pd_0 \geq 1/M$. Further, $Pd_0 = 1/M$ if and only if

    $|\{K \in \mathcal{K}: h_K(x) = y\}| = |\mathcal{K}|/M$ (4.3)

    for every $x \in \mathcal{X}$, $y \in \mathcal{Y}$.

$$\sum_{y\in\mathcal{Y}} payoff(x,y) = \sum_{y\in\mathcal{Y}} \frac{|\{K\in\mathcal{K} : h_K(x) = y\}|}{|\mathcal{K}|} = \frac{|\mathcal{K}|}{|\mathcal{K}|} = 1$$

Unconditionally Secure MACs

- THEOREM 4.15 Suppose $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$ is an (N, M)-hash family. Then $Pd_1 \geq 1/M$.

$$\sum_{y\in\mathcal{Y}} payoff(x',y';x,y) = \sum_{y\in\mathcal{Y}} \frac{|\{K\in\mathcal{K} : h_K(x')=y', h_K(x)=y\}|}{|\{K\in\mathcal{K} : h_K(x)=y\}|}$$

$$= \frac{|\{K\in\mathcal{K} : h_K(x)=y\}|}{|\{K\in\mathcal{K} : h_K(x)=y\}|} = 1$$

Unconditionally Secure MACs

- THEOREM 4.16 Suppose $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$ is an (N, M)-hash family. Then $Pd_1 \geq 1/M$ if and only if the hash family is strongly universal.
- **proof** ⇨ has already proved in Theorem 4.11.
  First show $\mathcal{V} = \mathcal{X}\times\mathcal{Y}$
  Let (x, y’) $\in \mathcal{X}\times\mathcal{Y}$; We will show (x’, y’) $\in \mathcal{V}$
  Let x $\in$ X, x $\neq$ x’. Choose y $\in \mathcal{Y}$ such that (x,y) $\in \mathcal{V}$
  From Theorem 4.15

$$\frac{|\{K\in\mathcal{K} : h_K(x')=y', h_K(x)=y\}|}{|\{K\in\mathcal{K} : h_K(x)=y\}|} = \frac{1}{M} \qquad (4.4)$$

  for every x, x’ $\in\mathcal{X}$, y, y’ $\in\mathcal{Y}$ such that (x,y) $\in\mathcal{V}$.

Unconditionally Secure MACs

$|\{K \in \mathcal{K}: h_K(x') = y', h_K(x) = y\}| > 0$
$\Rightarrow |\{K \in \mathcal{K}: h_K(x') = y'| > 0$
This prove that $(x',y') \in \mathcal{V}$, and hence $\mathcal{V} = \mathcal{X} \times \mathcal{Y}$.
From (4.4) we know that $(x,y) \in \mathcal{V}$ and $(x',y') \in \mathcal{V}$, so we can interchange the roles of (x, y) and (x', y').
$|\{K \in \mathcal{K}: h_K(x) = y\}| = |\{K \in \mathcal{K}: h_K(x') = y'\}|$
for all x, x', y, y'.
$|\{K \in \mathcal{K}: h_K(x) = y\}|$ is a constant.
$|\{K \in \mathcal{K}: h_K(x') = y', h_K(x) = y\}|$ is a constant

□

Unconditionally Secure MACs

- COROLLARY 4.17 Suppose $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$ is an (N, M)-hash family such that $Pd_1 = 1/M$. Then $Pd_0 = 1/M$.
- **Proof** Under the stated hypotheses, Theorem 4.16 says that $(\mathcal{X},\mathcal{Y},\mathcal{K},\mathcal{H})$ is strongly universal.
  Then $Pd_0 = 1/M$ from Theorem 4.11. □

Conclusion

- Hash function :
  - Compression + extension
  - Provably secure compression (ex.) + extension
  - Examples of hash functions (SHA-3)
- MAC and HMAC
  - Hash family and oracle model (forger adversary)
  - Security conditions
  - Unconditionally secure MAC (key used once)
    - Strongly universal hash families

ANNEX / Back slides

- Slides à réviser pour integration

4.2 Security of Hash Functions

- If a hash function is to be considered secure, these three problems are difficult to solve
  - Problem 4.1: Preimage
    - **Instance:** A hash function h: $\mathcal{X} \rightarrow \mathcal{Y}$ and an element y $\in \mathcal{Y}$.
    - **Find:** x $\in \mathcal{X}$ such that f(x) = y
  - Problem 4.2: Second Preimage
    - **Instance:** A hash function h: $\mathcal{X} \rightarrow \mathcal{Y}$ and an element x $\in \mathcal{X}$
    - **Find:** x' $\in \mathcal{X}$ such that x' ≠ x and h(x') = h(x)
  - Problem 4.3: Collision
    - **Instance:** A hash function h: $\mathcal{X} \rightarrow \mathcal{Y}$.
    - **Find:** x, x' $\in \mathcal{X}$ such that x' ≠ x and h(x') = h(x)

Security of Hash Functions

- A hash function for which Preimage cannot be efficiently solved is often said to be one-way or preimage resistant.
- A hash function for which Second Preimage cannot be efficiently solved is often said to be second preimage resistant.
- A hash function for which Collision cannot be efficiently solved is often said to be collision resistant.

Security of Hash Functions

- 4.2.1 The Random Oracle Model
  - The random oracle model provides a mathematical model of an “ideal” hash function.
  - In this model, a hash function h: $\mathcal{X} \rightarrow \mathcal{Y}$ is chosen randomly from $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$
    - The only way to compute a value h(x) is to query the oracle.
  - THEOREM 4.1 Suppose that h $\in \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is chosen randomly, and let $\mathcal{X}_0 \subseteq \mathcal{X}$. Suppose that the values h(x) have been determined (by querying an oracle for h) if and only if x $\in \mathcal{X}_0$. Then Pr[h(x)=y] = 1/M for all x $\in \mathcal{X} \setminus \mathcal{X}_0$ and all y $\in \mathcal{Y}$.

Security of Hash Functions

- 4.2.2 Algorithms in the Random Oracle Model
  - Randomized algorithms make random choices during their execution.
  - A Las Vegas algorithm is a randomized algorithm
    - may fail to give an answer
    - if the algorithm does return an answer, then the answer must be correct.
  - A randomized algorithm has average-case success probability ε if the probability that the algorithm returns a correct answer, averaged over all problem instances of a specified size , is at least ε ($0 \le \varepsilon < 1$).

Security of Hash Functions

- We use the terminology (ε,q)-algorithm to denote a Las Vegas algorithm with average-case success probability ε
  - the number of oracle queries made by algorithms is at most q.
- Algorithm 4.1: FIND PREIMAGE (h, y, q)
  - choose any $\mathcal{X}_0 \subseteq \mathcal{X}$, $|\mathcal{X}_0|$ = q
  - **for each** x $\in \mathcal{X}_0$
    do if h(x) = y
    then return (x)
  - **return** (failure)

Security of Hash Functions

- THEOREM 4.2 For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0|$ = q, the average-case success probability of Algorithm 4.1 is ε=1 - (1-1/M)^q.
  - **proof** Let y $\in \mathcal{Y}$ be fixed. Let $X_0 = \{x_1, x.., x_q\}$.
    For 1 ≤ i ≤ q, let E_i denote the event "h(x_i) = y".
    From Theorem 4.1 that the E_i's are independent events, and Pr[E_i] = 1/M for all 1 ≤ i ≤ q.
    Therefore $\Pr[E_1 \vee E_1 \vee \ldots \vee E_q] = 1 - \left(1 - \frac{1}{M}\right)^q$
    The success probability of Algorithm 4.1, for any fixed y, is constant.
    Therefore, the success probability averaged over all y $\in \mathcal{Y}$ is identical, too.

□

Security of Hash Functions

- Algorithm 4.2: FIND SECOND PREIMAGE (h,x,q)
  - y → h(x)
  - choose $\mathcal{X}_0 \subseteq \mathcal{X} \setminus \{x\}$, $|\mathcal{X}_0| = q - 1$
  - **for each** $x_0 \in \mathcal{X}_0$

    do if $h(x_0) = y$

    then return (x_0)
  - **return** (failure)
- THEOREM 4.3 For any $\mathcal{X}_0 \subseteq \mathcal{X} \setminus \{x\}$ with $|\mathcal{X}_0| = q - 1$, the success probability of Algorithm 4.2 is $\varepsilon = 1 - (1 - 1/M)^{q-1}$.

Security of Hash Functions

- Algorithm 4.3: FIND COLLISION (h,q)
  - choose $\mathcal{X}_0 \subseteq \mathcal{X}$, $|\mathcal{X}_0| = q$
  - **for each** $x \in \mathcal{X}_0$

    do $y_x \leftarrow h(x)$
  - **if** $y_x = y_{x'}$ for some $x' \neq x$

    then return (x, x')
  - **else return** (failure)

Security of Hash Functions

- Birthday paradox
  - In a group of 23 randomly chosen people, at least two will share a birthday with probability at least ½.
  - Finding two people with the same birthday is the same thing as finding a collision for this particular hash function.
  - ex: Algorithm 4.3 has success probability at least ½ when q = 23 and M = 365
- Algorithm 4.3 is analogous to throwing q balls randomly into M bins and then checking to see if some bin contains at least two balls.

Security of Hash Functions

- THEOREM 4.4 For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0| = q$, the success probability of Algorithm 4.3 is

$$\varepsilon = 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\dots\left(\frac{M-q+1}{M}\right)$$

  - **proof** Let $\mathcal{X}_0 = \{x_1,..,x_q\}$.
    E_i : the event "$h(x_i) \notin \{h(x_1),..,h(x_{i-1})\}$." , $2 \leq i \leq q$
    Using induction, from Theorem 4.1 that $\Pr[E_1] = 1$ and

$$\Pr[E_i \mid E_1 \wedge E_2 \wedge .. \wedge E_{i-1}] = \frac{M-i+1}{M}$$ for $2 \leq i \leq q$.

$$\Pr[E_1 \wedge E_2 \wedge ... \wedge E_q] = \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)..\left(\frac{M-q+1}{M}\right)$$

Security of Hash Functions

x is small
$1-x \approx e^{-x}$

The probability of finding no collision is

$$\prod_{i=1}^{q-1}(1-\frac{i}{M}) \approx \prod_{i=1}^{q-1} e^{\frac{-i}{M}} \approx e^{-\sum_{i=1}^{q-1}\frac{i}{M}} = e^{\frac{-q(q-1)}{2M}}$$

- ε denotes the probability of finding at least one collision

$$e^{\frac{-q(q-1)}{2M}} \approx 1-\varepsilon \qquad \frac{-q(q-1)}{2M} \approx \ln(1-\varepsilon) \qquad q^2 - q \approx 2M\ln\frac{1}{1-\varepsilon}$$

  - Ignore –q, $q \approx \sqrt{2M\ln\frac{1}{1-\varepsilon}}$
  - ε= 0.5, q ≈ 1.17 $\sqrt{M}$
  - Take M = 365, we get q ≈ 22.3

Security of Hash Functions

- This says that hashing just over $\sqrt{M}$ random elements of X yields a collision with a prob. of 50%.
- A different choice of εleads to a different constant factor, but q will still be proportional to $\sqrt{M}$. So this algorithm is a (1/2, O($\sqrt{M}$))-algorithm.

Security of Hash Functions

- The birthday attack imposes a lower bound on the size of secure message digests. A 40-bit message digest would be very in secure, since a collision could be found with prob. ½ with just over 2^20 (about a million) random hashes.
- It is usually suggested that the minimum acceptable size of a message digest is 128 bits (the birthday attack will require over 2^64 hashes in this case). In fact, a 160-bit message digest (or larger) is usually recommended.

Security of Hash Functions

- 4.2.3 Comparison of Security Criteria
  - In the random oracle model, solving Collision is easier than solving Preimage of Second Preimage.
  - Whether there exist reductions among these three problems which could be applied to arbitrary hash functions? (Yes.)
  - Reduce Collision to Second Preimage using Algorithm 4.4.
  - Reduce Collision to Preimage using Algorithm 4.5.

Security of Hash Functions

– Algorithm 4.4: COLLISION TO SECOND PREIMAGE (h)

- **external** ORACLE2NDPREIMAGE
- choose x $\in \mathcal{X}$ uniformly at random
- **if** (ORACLE2NDPREIMAGE(h,x) = x') (!error here in the text)
  then return (x, x')
- **else return** (failure)

Security of Hash Functions

– Suppose that ORACLE2NDPREIMAGE is an (ε, q)-algorithm that solves Second Preimage for a particular, fixed hash function h.
Then COLLISIONTOSECONDPREIMAGE is an (ε, q)-algorithm(!error here in text) that solves Collision for the same hash function h.

– As a consequence of this reduction, collision resistance implies second preimage resistance.

Security of Hash Functions

- Algorithm 4.5: COLLISION TO PREIMAGE (h)
  - **external** ORACLEPREIMAGE
  - choose $x \in \mathcal{X}$ uniformly at random
  - $y \leftarrow h(x)$
  - **if** (ORACLEPREIMAGE(h,y) = x') **and** (x' ≠ x)

    then return (x, x')
  - **else return** (failure)

Security of Hash Functions

- THEOREM 4.5 Suppose h: $\mathcal{X} \rightarrow \mathcal{Y}$ is a hash function where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are finite and $|\mathcal{X}| \geq 2|\mathcal{Y}|$. Suppose ORACLEPREIMAGE is a (1,q) algorithm for Preimage, for the fixed hash function h.(and so h is surjective(onto)) Then COLLISION TO PREIMAGE is a (1/2, q+1) algorithm for **Collision**, for the fixed hash function h.

Security of Hash Functions

- **proof** For any x $\in \mathcal{X}$, define equivalence class $\mathcal{C}$: [x]= {$x_1 \in \mathcal{X}$: h(x) = h(x_1)}
  (see text for detailed notation)
  Given the element x $\in \mathcal{X}$, the probability of success is (|[x]| - 1) / |[x]| in ORACLEPREIMAGE.
  The probability of success of algorithm COLLISION TO PREIMAGE is (average)

$$\Pr[success] = \frac{1}{|\chi|}\sum_{x\in\chi}\frac{|[x]|-1}{|[x]|} = \frac{1}{|\chi|}\sum_{C\in\mathcal{C}}\sum_{x\in C}\frac{|C|-1}{|C|}$$

$$= \frac{1}{|\chi|}\sum_{C\in\mathcal{C}}(|C|-1) = \frac{1}{|\chi|}\left(\sum_{C\in\mathcal{C}}|C| - \sum_{C\in\mathcal{C}}1\right)$$

$$= \frac{|\chi|-|\mathcal{Y}|}{|\chi|} \geq \frac{|\chi|-|\chi|/2}{|\chi|} = \frac{1}{2}$$

4.3 Iterated Hash Function

- Compression function: hash function with a finite domain
- A hash function with an infinite domain can be constructed by the mapping method of a compression function is called an iterated hash function.
- We restrict our attention to hash functions whose inputs and outputs are bitstrings (i.e., strings formed of 0s and 1s).

4.3 Iterated Hash Function

- Iterated hash function h: $\bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \rightarrow \{0,1\}^l$

Suppose that compress: $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ is a compression function (where $t \geq 1$).

- **Preprocessing**
  - given x ($|x| \geq m + t + 1$)
  - construct $y = x \,||\, pad(x)$
    such that $|y| \equiv 0 \pmod t$
    $y = y_1 \,||\, y_2 \,||\ldots||\, y_r$, where $|y_i| = t$ for $1 \leq i \leq r$
  - pad(x) is constructed from x using a padding function.
  - the mapping x -> y must be an injection (1 to 1)

Iterated Hash Function

- **Processing**
  - IV is a public initial value which is a bitstring of length m.
  - $z_0 \leftarrow IV$
  - $z_1 \leftarrow compress(z_0 \,||\, y_1)$
  - ….
  - $z_r \leftarrow compress(z_{r-1} \,||\, y_r)$
- **Optional output transformation**
  - g: $\{0,1\}^m \rightarrow \{0,1\}^l$
  - $h(x) = g(z_r)$

compress function:
$\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ ($t \geq 1$)

Iterated Hash Function

- 4.3.1 The Merkle-Damgard Construction
  - Algorithm 4.6: MERKLE-DAMGARD(x)
    - **external** compress
    - **comment:** compress: $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$, where $t \geq 2$
    - $n \leftarrow |x|$
    - $k \leftarrow \lceil n/(t - 1) \rceil$
    - $d \leftarrow n - k(t - 1)$
    - **for** $i \leftarrow 1$ **to** $k - 1$

      do $y_i \leftarrow x_i$

Iterated Hash Function

- $y_k \leftarrow x_k \,||\, 0^d$
- $y_{k+1} \leftarrow$ the binary representation of d
- $z_1 \leftarrow 0^{m+1} \,||\, y_1$
- $g_1 \leftarrow \text{compress}(z_1)$
- **for** $i \leftarrow 1$ **to** k

  do $z_{i+1} \leftarrow g_i \,||\, 1 \,||\, y_{i+1}$

  $g_{i+1} \leftarrow \text{compress}(z_{i+1})$
- $h(x) \leftarrow g_{k+1}$
- **return** $(h(x))$

Iterated Hash Function

- THEOREM 4.6 Suppose compress : $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ is a collision resistant compression function, where $t \geq 2$. Then the function

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \longrightarrow \{0,1\}^m$$

as constructed in Algorithm 4.6, is a collision resistant hash function.

- **proof**

Suppose that we can find $x \neq x'$ such that $h(x)= h(x')$.
$y(x) = y_1 \| y_2 \|..\| y_{k+1}$, x is padded with d 0's
$y(x') = y'_1 \| y'_2 \|..\| y'_{l+1}$, x' is padded with d' 0's
g-values : $g_1,.., g_{k+1}$ or $g'_1,.., g'_{l+1}$

Iterated Hash Function

- **case 1:**$|x| \,!\!\equiv |x'| \pmod{t - 1}$

$d \neq d'$ and $y_{k+1} \neq y'_{l+1}$
$compress(g_k \| 1 \| y_{k+1}) = g_{k+1}= h(x) = h(x') = g'_{l+1}$
$= compress\ (g'_l \| 1 \| y'_{l+1})$,
which is a collision for compress because $y_{k+1} \neq y'_{l+1}$

- **case2:** $|x| \equiv |x'| \pmod{t - 1}$
- **case2.a:** $|x| = |x'|$

$k = l$ and $y_{k+1} = y'_{k+1}$
$compress(g_k \| 1 \| y_{k+1}) = g_{k+1} = h(x) = h(x') = g'_{k+1}$
$= compress\ (g'_k \| 1 \| y'_{k+1})$
If $g_k \neq g'_k$, then we find a collision for compress, so assume $g_k = g'_k$.

Iterated Hash Function

compress(g_{k-1} || 1 ||y_k) = g_k = g'_k
= compress (g'_{k-1} || 1 || y'_k)

Either we find a collision for compress, or $g_{k-1} = g'_{k-1}$ and $y_k = y'_k$.

Assuming we do not find a collision, we continue work backwards, until finally we obtain

compress(0^{m+1} || y_1) = g_1 = g'_1 = compress (0^{m+1}||y'_1)

If $y_k \neq y'_k$, then we find a collision for compress, so we assume $y_1 = y'_1$.

But then $y_i = y'_i$ for $1 \leq i \leq k+1$, so y(x) = y(x').

Iterated Hash Function

- This implies x = x', because the mapping $x \rightarrow y(x)$ is an injection.

  We assume $x \neq x'$, so we have a contradiction.
- **case 2b**: $|x| \neq |x'|$

  Assume |x'| > |x|, so l > k

  Assuming we find no collisions for compress, we reach the situation where

  compress(0^{m+1} || y_1) = g_1 = g'_{l-k+1} =
  compress (g'_{l-k} || 1 || y'_{l-k+1}).

  But the (m+1)st bit of 0^{m+1} || y_1 is a 0

  and the (m+1)st bit of g'_{l-k} || 1 || y'_{l-k+1} is a 1.

  So we find a collision for compress.

Iterated Hash Function

- Algorithm 4.7: MERKLE-DAMGARD2(x) (t = 1)
  - **external** compress
  - **comment:** compress: $\{0,1\}^{m+1} \rightarrow \{0,1\}^m$
  - $n \leftarrow |x|$
  - $y \leftarrow 11 \,||\, f(x_1) \,||\, f(x_2) \,||\ldots||\, f(x_n)$
    denote $y = y_1 \,||\, y_2 \,||\ldots||\, y_k$, where $y_i \in \{0,1\}$,
    $1 \le i \le k$
  - $g_1 \leftarrow$ compress$(0^m \,||\, y_1)$
  - **for** $i \leftarrow 1$ **to** $k - 1$
    do $g_{i+1} \leftarrow$ compress$(g_i \,||\, y_{i+1})$
  - **return** (g_k)

$f(0)=0$
$f(1)=01$

Iterated Hash Function

- The encoding $x \rightarrow y = y(x)$, as defined algorithm 4.7 satisfies two important properties:
  - If $x \neq x'$, then $y(x) \neq y(x')$ (i.e. $x \rightarrow y = y(x)$ is an injection)
  - There do not exist two strings $x \neq x'$ and a string z such that $y(x) = z \,||\, y(x')$ (i.e. no encoding is a postfix of another encoding)

Iterated Hash Function

- THEOREM 4.7 Suppose compress : $\{0,1\}^{m+1} \rightarrow : \{0,1\}^m$ is a collision resistant compression function. Then the function $$h : \bigcup_{i=m+2}^{\infty} \{0,1\}^i \longrightarrow \{0,1\}^m,$$ as constructed in Algorithm 4.7, is a collision resistant hash function.
- **proof** Suppose that we can find $x \neq x'$ such that
  $h(x) = h(x')$.
  Denote $y(x) = y_1y_2\ldots y_k$ and $y(x') = y'_1y'_2\ldots y'_l$
  case1: $k = l$
  As in Theorem 4.6, either we find a collision for compress, or we obtain $y = y'$.
  But this implies $x = x'$, a contradiction.

Iterated Hash Iterated Hash Function Function

case 2: $k \neq l$

Without loss of generality, assume $l > k$

Assuming we find no collision for compress, we have following sequence of equalities:

$y_k = y'_l$

$y_{k-1} = y'_{l-1}$

… …

$y_1 = y'_{l-k+1}$

But this contradicts the "postfix-free" property We conclude that h is collision resistant.

□

Iterated Hash Function

- THEOREM 4.8 Suppose compress: $\{0,1\}^{m+t} \rightarrow \{0,1\}^m$ is a collision resistant compression function, where $t \geq 1$. Then there exists a collision resistant hash function

$$h: \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \longrightarrow \{0,1\}^m,$$

The number of times compress is computed in the evaluation of h is at most

$$\begin{cases} 1+\left\lceil \frac{n}{t-1} \right\rceil & \text{if } t \geq 2 \\ 2n+2 & \text{if } t = 1 \end{cases}$$

where $|x| = n$.

Iterated Hash Function

- 4.3.2 The Secure Hash algorithm
  - SHA-1(Secure Hash Algorithm)
    - iterated hash function
    - 160-bit message digest
    - word-oriented (32 bit) operation on bitstrings
  - Operations used in SHA-1
    - $X \wedge Y$ bitwise "and" of X and Y
    - $X \vee Y$ bitwise "or" of X and Y
    - $X \oplus Y$ bitwise "xor" of X and Y
    - $\neg X$ bitwise complement of X
    - $X + Y$ integer addition modulo 2^{32}
    - $ROTL^s(X)$ circular left shift of X by s position ($0 \leq s \leq 31$)

Iterated Hash Function

- Algorithm 4.8 SHA-1-PAD(x)
  - **comment:** $|x| \leq 2^{64} - 1$
  - $d \leftarrow (447-|x|) \bmod 512$
  - $l \leftarrow$ the binary representation of $|x|$, where $|l| = 64$
  - $y \leftarrow x \,||\, 1 \,||\, 0^d \,||\, l$ ($|y|$ is multiple of 512)
- $f_t(B,C,D) =$
  - $(B \wedge C) \vee ((\neg B) \wedge D)$ if $0 \leq t \leq 19$
  - $B \oplus C \oplus D$ if $20 \leq t \leq 39$
  - $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ if $40 \leq t \leq 59$
  - $B \oplus C \oplus D$ if $60 \leq t \leq 79$

Iterated Hash Function

- $K_t =$
  - 5A827999 if $0 \leq t \leq 19$
  - 6ED9EBA1 if $20 \leq t \leq 39$
  - 8F1BBCDC if $40 \leq t \leq 59$
  - CA62C1D6 if $60 \leq t \leq 79$
- Cryptosystem 4.1: SHA-1(x)
  - **extern** SHA-1-PAD
  - **global** $K_0,\ldots,K_{79}$
  - $y \leftarrow$ SHA-1-PAD(x) denote $y = M_1 \,||\, M_2 \,||..||\, M_n$, where each M_i is a 512 block
  - $H_0 \leftarrow$ 67452301, $H_1 \leftarrow$ EFCDAB89, $H_2 \leftarrow$ 98BADCFE, $H_3 \leftarrow$ 10325476, $H_4 \leftarrow$ C3D2E1F0

Iterated Hash Function

– **for** $i \leftarrow 1$ **to** n
  - denote $M_i = W_0 \,||\, W_1 \,||..||\, W_{15}$, where each W_i is a word
  - **for** $t \leftarrow 16$ **to** 79
    do $W_t \leftarrow ROTL^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$
  - $A \leftarrow H_0$, ,$B \leftarrow H_1$, $C \leftarrow H_2$, $D \leftarrow H_3$, $E \leftarrow H_4$
  - **for** $t \leftarrow 0$ **to** 79
    $temp \leftarrow ROTL^5(A) + f_t(B,C,D) + E + W_t + K_t$
    $E \leftarrow D$, $D \leftarrow C$, $C \leftarrow ROTL^{30}(B)$, $B \leftarrow A$, $A \leftarrow temp$
  - $H_0 \leftarrow H_0 + A$, $H_1 \leftarrow H_1 + B$, $H_2 \leftarrow H_2 + C$, $H_3 \leftarrow H_3 + D$, $H_4 \leftarrow H_4 + E$

Return (H || H || H || H || H)

Iterated Hash Function

- **MD4 proposed by Rivest in 1990**
- **MD5 modified in 1992**
- **SHA proposed as a standard by NIST in 1993, and was adopted as FIPS 180**
- **SHA-1 minor variation, FIPS 180-1**
- **SHA-256**
- **SHA-384**
- **SHA-512**