Applied Cryptography and Computer Security

CSE 664 Spring 2017

Lecture 9: Hash Functions

Department of Computer Science and Engineering
University at Buffalo

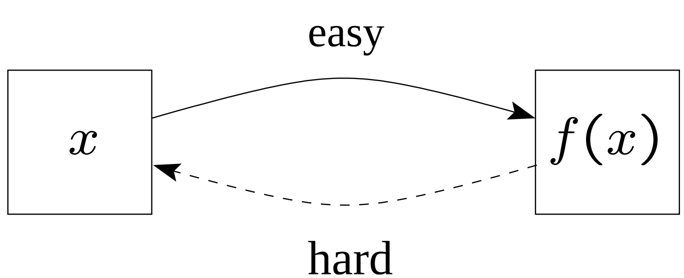
Lecture Outline

- So far **we learned** about
  - theoretical tools
  - practical algorithms
- **In this lecture** we learn about another practical tool of great importance in cryptography
  - hash functions
  - HMAC
  - other uses of hash functions

Quick Detour: One-Way Functions

- **A** **one-way function** **is easy to compute, but is hard to invert**

- **More formally, if f is one-way, then it is easy to compute $f(x)$ from x, but given $f(x)$ it is infeasible to find x**



- **Example: breaking a glass**

- **One-way functions are a very powerful tool**

- **It is not known whether they exist**

Hash Functions

- A hash function h at minimum should satisfy the following properties:
  - compression: h maps an input x of an arbitrary length to a (short) fixed-length output $h(x)$
  - ease of computation: given h and x, $h(x)$ is easy to compute
- Hash functions have many uses including hash tables
- We are interested in cryptographic hash function that must satisfy certain security properties
- Informally, what we are looking for in a hash function h is:
  - given $h(x)$, it is hard to compute x
  - it is hard to find x and x' such that $h(x) = h(x')$

Hash Functions

- **Cryptographic hash functions are often used as a real-life substitute for ideal one-way functions**
- **But they have other important uses as well:**
  - **data integrity**
  - **message authentication**
  - **password hashing and one-time passwords**
  - **in digital signatures**
  - **timestamping**
  - **and others**

Hash Functions

- More formally, let $h : X \to Y$ be a cryptographic hash function
- h must satisfy the following security properties:
  - **Preimage resistance** (one-way): given h and $y \in Y$, it is difficult to find $x \in X$ such that $h(x) = y$
  - **Second preimage resistance** (weak collision resistance): given h and $x \in X$, it is difficult to find $x' \in X$ such that $x' \neq x$ and $h(x') = h(x)$
  - **Collision resistance** (strong collision resistance): given h, it is difficult to find $x, x' \in X$ such that $x' \neq x$ and $h(x') = h(x)$

Hash Functions

- **Normally the input domain is all strings $\{0,1\}^*$ and the output is $\{0,1\}^{\ell(n)}$ for security parameter n**

- **Collision resilience formally: collision finding experiment $\mathsf{Hash\text{-}coll}_{\mathcal{A},h}(n)$:**

  1. **adversary $\mathcal{A}$ is given h and outputs x, x'**
  2. **output 1 ($\mathcal{A}$ wins) if $x \neq x'$ and $h(x) = h(x')$, and 0 otherwise**

- **Definition: A function h is collision resistant if any PPT adversary $\mathcal{A}$ can't win the game with more than a negligible probability, i.e.:**

$$\Pr[\mathsf{Hash\text{-}coll}_{\mathcal{A},h}(n) = 1] \leq \mathsf{negl}(n)$$

Hash Functions

- **A good cryptographic hash function (satisfying the definition) will have:**
  - **non-correlation: input bits and output bits should not be correlated (and it is desirable that every input bit affects every output bit)**
  - **near-collision resistance: it should be hard to find any two inputs x and x' such that $h(x)$ and $h(x')$ differ only in a small number of bits**
  - **partial-preimage resistance or local one-wayness: it should be as difficult to recover any substring as to recover the entire input**
    - **and even if part of the input is known, it should difficult to find the remainder**

Hash Function

- A cryptographic hash function can be **keyed**
  - it takes a secret key as its another parameter
  - that secret key defines the function's behavior
    - i.e., each new key makes it a new hash function
- Formally, a **hash family** is defined by algorithms $(\mathsf{Gen}, \mathsf{H})$
  - key generation algorithm Gen, on input security parameter 1^n, outputs key k
  - hashing algorithm H, on input a key k and string $x \in \{0,1\}^*$, outputs a string $y \in \{0,1\}^{\ell(n)}$
- The key k can be public or private

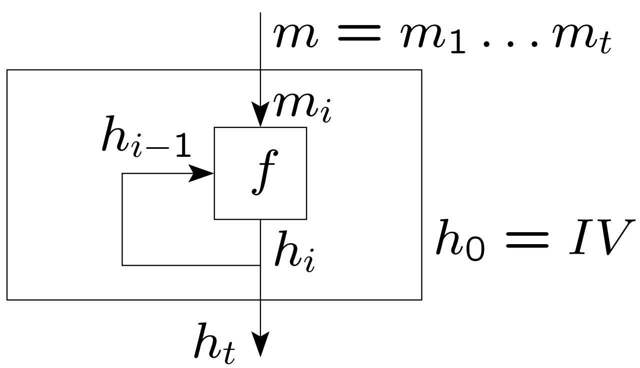
Hash Functions

- **Commonly used hash function algorithms:**
  - **MD5**
  - **SHA-1**
  - **SHA-2 family (SHA-256, SHA-384, and others)**
- **Normally hash function algorithms are iterated**
  - **they use a compression function**
  - **the input is partitioned into blocks**
  - **a compression function is used on the current block m_i and the previous output h_{i-1} to compute**

$$h_i = f(m_i, h_{i-1})$$

Hash Functions

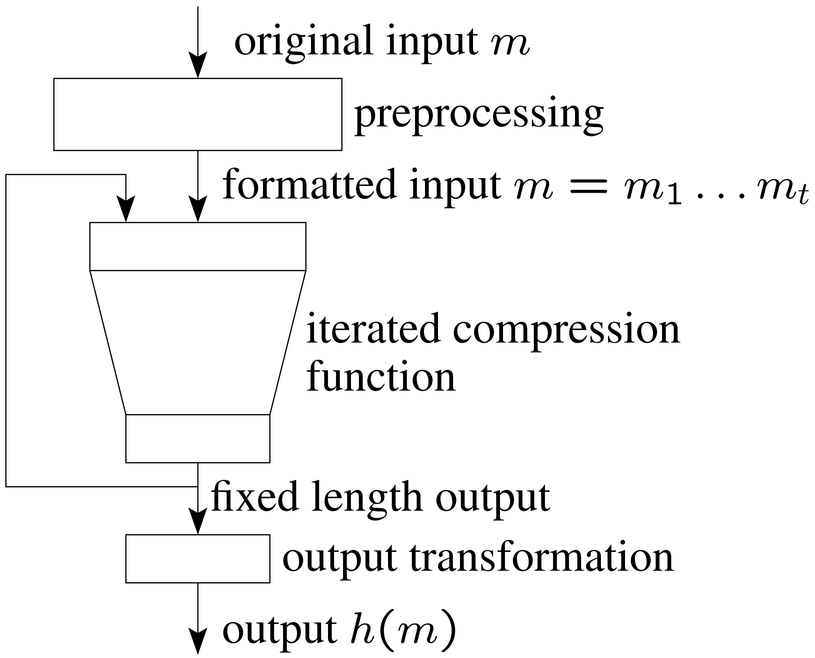
- **Most unkeyed hash functions use a compression function f**
  - **f takes a fixed length ℓ-bit input and outputs an intermediate result of length n ($\ell > n$)**
- **Most unkeyed hash functions use chaining**
  - **output of the current block depends on all previous blocks**
  - **let the input be $m = m_1 m_2 ... m_t$**
  - **set $h_0 = IV$; $h_i = f(m_i, h_{i-1})$; and $h(m) = h_t$**



Hash Functions

- **Often, before the iterated compression function is called a preprocessing step is used**

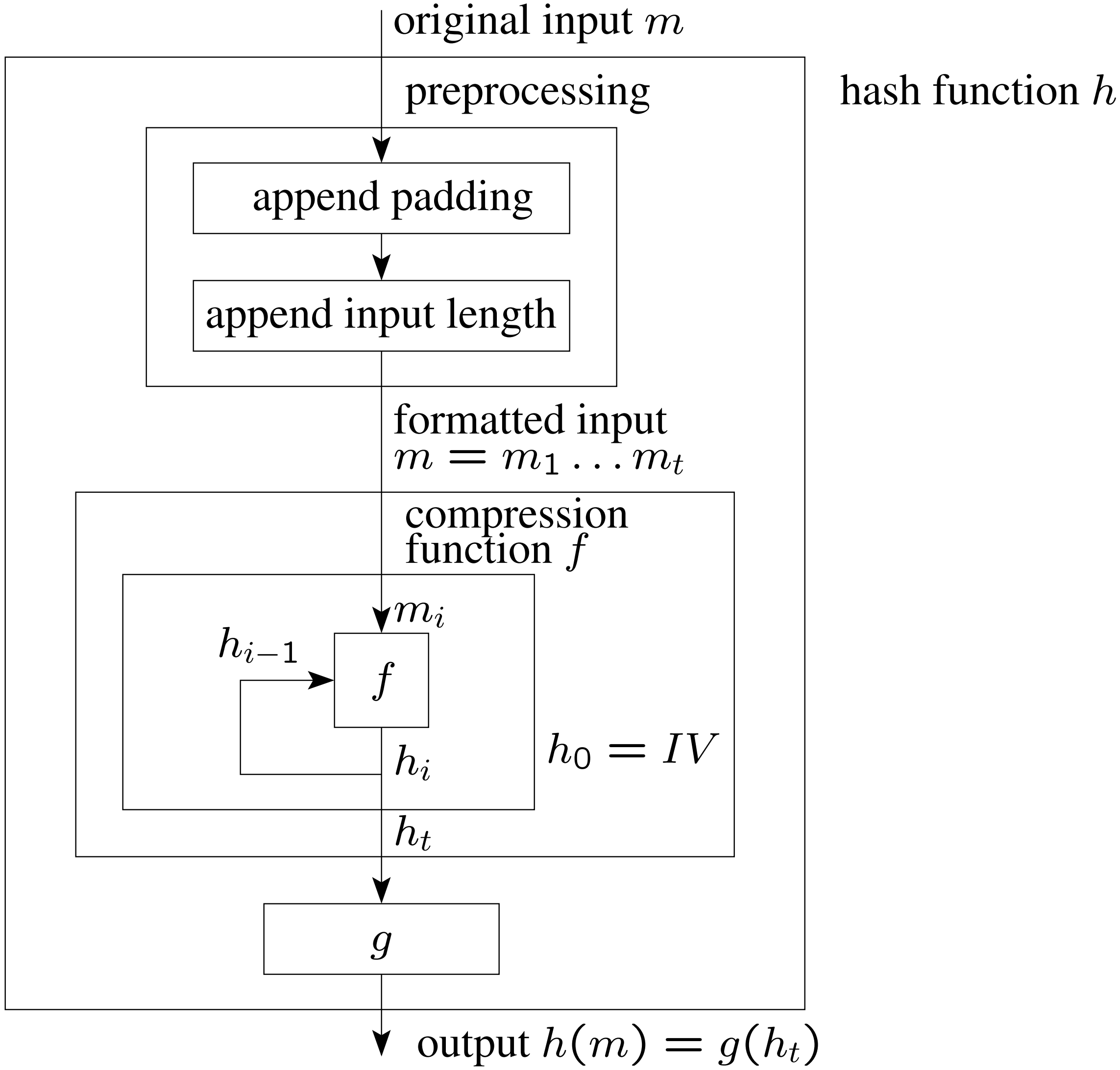
- **Also, after the compression function, output transformation can be applied**



Hash Functions

- **The preprocessing step typically includes:**
  - **padding the message (i.e., appending extra bits) to obtain a bitlength multiple of the blocklength ℓ**
  - **appending the length of the unpadded input**
    - **this prevents collisions and thus improves security**
- **The output transformation g is optional**
  - **it can map the n-bit output h_t to a result of another length**
  - **often $g(h_t) = h_t$**

Hash Functions: Detailed View



Hash Functions

- **Merkle-Damgard construction**
  - **we are given a compression function** $f : \{0,1\}^{\ell+n} \rightarrow \{0,1\}^n$
  - **divide the input** m **into** t **blocks** $m_1 m_2 \ldots m_t$ **of size** ℓ **padding the last block with 0s if necessary**
  - **define an extra final block** m_{t+1} **to hold the right justified binary representation of original** m**'s length**
  - **set** $h_0 = 0^n$ **and compute** $h_i = f(h_{i-1}||m_i)$ **for** $i = 1, \ldots, t+1$
  - **output** $h(m) = h_{t+1}$
- **Theorem: If** f **is (fixed-length) collision resistant hash function, this construction is collision resistant**

Hash Functions

- **Cascading hash functions**
  - **we are given two hash functions h_1 and h_2**
  - **if either h_1 or h_2 is collision resistant, $h(x) = h_1(x)||h_2(x)$ is a collision resistant hash function**
  - **if h_1 and h_2 are independent, have to find a collision in both simultaneously**
  - **hopefully this would require the product of the effort to attack them individually**
  - **this is a simple yet powerful way to increase strength using available functions**

Attacks on Hash Functions

- **Attacks on the bitsize of a hash**
  - **assume we are given a message m and its hash $h(m)$**
  - **we want to find another message m' with the same hash**
  - **a naive approach for finding a collision is to pick a random m' and check whether $h(m) = h(m')$**
  - **this can result in very little effort, but for well-distributed hashes the probability of a match is 2^{-n}**
  - **however, if we have control over m as well, the effort greatly reduces**
  - **colliding pairs of messages m and m' where $h(m) = h(m')$ can be done in $2^{n/2}$ time**

Birthday Attack

- **Birthday attack is one of cryptographic applications of birthday paradox**
- **Birthday paradox:**
  - **we are given a group of people**
  - **what is the minimum group size required to find two people who who share the same birthday with probability at least $1/2$?**
- **General problem statement:**
  - **we are given a random variable that is an integer with uniform distribution between 1 and n**
  - **given a selection of k instances ($k < n$) of the variable, what is the probability $\mathsf{Pr}(n, k)$ that there is at least one duplicate?**

Birthday Paradox

- **Calculating** $\mathsf{Pr}(365, k)$
  - **if we pick k random days out of 365, what is the probability that there are no collisions?**
  - **the number of possibilities with no collision:** $365 \times 364 \times \cdots \times (365 - k + 1) = 365!/(365 - k)!$
  - **the total number of possibilities:** 365^k
  - **thus, we obtain**

$$\mathsf{Pr}(365, k) = 1 - \frac{365!}{(365 - k)!365^k}$$

  - **if** $k = 23$**,** $\mathsf{Pr}(365, 23) = 0.5073$

Birthday Paradox

- **In general:**

$$\begin{aligned}\Pr(n,k) &= 1 - \frac{n!}{(n-k)!n^k} = 1 - \frac{n(n-1)\cdots(n-k+1)}{n^k} \\ &= 1 - \frac{n}{n}\cdot\frac{n-1}{n}\cdot\frac{n-2}{n}\cdots\frac{n-(k-1)}{n} \\ &= 1 - \left(1-\frac{1}{n}\right)\left(1-\frac{2}{n}\right)\cdots\left(1-\frac{k-1}{n}\right)\end{aligned}$$

  - **if x is a small real number, then $1 - x \approx e^{-x}$**
  - **using it in our equations, we obtain:**

$$\Pr(n,k) \approx 1 - e^{-\frac{1}{n}} \cdot e^{-\frac{2}{n}} \cdots e^{-\frac{k-1}{n}} = 1 - e^{-\frac{k(k-1)}{2n}}$$

Birthday Paradox

- **Say, we want $\mathsf{Pr}(n, k) > 0.5$. What k is needed?**

$$\frac{1}{2} = 1 - e^{-\frac{k(k-1)}{2n}} \Rightarrow e^{-\frac{k(k-1)}{2n}} = \frac{1}{2} \Rightarrow$$

$$-\frac{k(k-1)}{2n} = \ln(1/2) \Rightarrow \frac{k(k-1)}{2n} = \ln 2$$

- **For large k, $k(k-1) \approx k^2$, thus we obtain:**

$$\frac{k^2}{2n} \approx \ln 2 \Rightarrow k^2 \approx (\ln 2)2n \Rightarrow$$

$$k \approx \sqrt{(2\ln 2)n} = 1.18\sqrt{n} \approx \sqrt{n}$$

Security of Hash Functions

- **This directly applies to hash functions:**
  - for a hash function that produces n-bit output, there are 2^n possible output values
  - but about $\sqrt{2^n} = 2^{n/2}$ tries are needed to find a collision with a good probability
- **Choosing output length**
  - to achieve 128-bit security, we need 256-bit output values
- As applied to hash functions, birthday paradox is used in Yuval's birthday attack

Birthday Attack

- **We have a legitimate message m_1 and a fraudulent message m_2**
- **We want to find m_1' and m_2' resulting from minor modifications of m_1 and m_2 with $h(m_1') = h(m_2')$**
  - **then a signature on the hash of m_1' is a valid signature on m_2''s hash**
- **Birthday attack:**
  - **find $n/2$ places to tweak m_1**
  - **generate $2^{n/2}$ minor modifications m_1' of m_1**
  - **hash each modified message and store message-hash pairs (searchable by the hash value)**
  - **generate minor modifications m_2' of m_2 computing $h(m_2')$ for each and checking for matches with any m_1' above until a match is found**

Birthday Attack

- **Example:**
  - message m_1 and its 2^{14} modifications:

$$\left\{\begin{array}{c}\text{This letter is}\\ \text{I am writing}\end{array}\right\} \text{to introduce} \left\{\begin{array}{c}\text{you to}\\ \text{to you}\end{array}\right\} \left\{\begin{array}{c}\text{Mr.}\\ \text{–}\end{array}\right\} \text{Alfred} \left\{\begin{array}{c}\text{P.}\\ \text{–}\end{array}\right\}$$

$$\text{Barton, the} \left\{\begin{array}{c}\text{new}\\ \text{newly appointed}\end{array}\right\} \left\{\begin{array}{c}\text{chief}\\ \text{senior}\end{array}\right\} \text{jewelry buyer for}$$

$$\left\{\begin{array}{c}\text{our}\\ \text{the}\end{array}\right\} \text{Northern} \left\{\begin{array}{c}\text{European}\\ \text{Europe}\end{array}\right\} \left\{\begin{array}{c}\text{area}\\ \text{division}\end{array}\right\}\text{. He} \left\{\begin{array}{c}\text{will take}\\ \text{has taken}\end{array}\right\}$$

$$\text{over} \left\{\begin{array}{c}\text{the}\\ \text{–}\end{array}\right\} \text{responsibility for} \left\{\begin{array}{c}\text{all}\\ \text{the whole of}\end{array}\right\} \text{our interests in}$$

$$\left\{\begin{array}{c}\text{watches and jewellery}\\ \text{jewellery and watches}\end{array}\right\} \text{in the} \left\{\begin{array}{c}\text{area}\\ \text{region}\end{array}\right\}.$$

- No generic attacks of effort less than 2^n are known for other security properties (pre-image and second pre-image resistance)

Random Oracle Model

- The Random Oracle Model (ROM) models an "ideal" hash function
- This ideal function is such that
  - the only efficient way to determine the value of $h(x)$ is to actually evaluate the function on x
  - the output is truly random and cannot be predicted even if other values $h(x')$, $h(x'')$, etc. are known
- Every time the ideal hash function is used, you consult an "oracle"
  - you send x to the oracle and obtain $h(x)$ back
- This model was introduced by Bellare and Rogaway in 1993

Random Oracle Model

- The rationale for using the random oracle model is
  - collision or preimage resistance of a hash function is not always sufficient to prove security
  - constructions that use hash functions can be more efficient than constructions without them
  - if we use an ideal function, we can prove construction with hash functions secure
- Is this model secure?
  - generally it is secure, but there are counterexamples
  - avoid this model if alternatives exist

Hash Function Algorithms

- **Families of customized hash functions**
  - **MD2, MD4, MD5** (MD = message digest)
    - a family of cryptographic hash functions designed by Ron Rivest
    - all have 128-bit output
    - MD2 was perceived as slower and less secure than MD4 and MD5
    - MD4 is specified as internet standard in RFC 1320
    - MD5 was designed as a strengthened version of MD4 before weaknesses in MD4 were found
    - MD5 is specified as internet standard RFC 1321
  - **SHA-0, SHA-1**
  - **SHA-2 family**

Hash Function Algorithms

- **MD4/MD5**
  - for 128-bit hashes, collisions are expected in 2^{64} time
  - collisions have been found for MD4 in 2^{20} compression function computations (90s)
  - MD5 was widely used until relatively recently
  - attacks on MD5
    - Boer and Bosselaers found a pseudo collision (same message, two different IV's) in 1993
    - Dobbertin created collisions for MD5 compression function with a chosen IV in 1996
    - Wang et al. in 2004 found collisions for MD5 for any IV which are easy to find

Hash Function Algorithms

- **Secure Hash Algorithm (SHA)**
  - SHA was designed by NIST and published in FIPS 180 in 1993
  - In 1995 a revision, known as SHA-1, was specified in FIPS 180-1
    - it is also specified in RFC 3174
  - SHA-0 and SHA-1 have 160 bit output and MD4-based design
  - In 2002 NIST produced a revision of the standard in FIPS 180-2
  - SHA-2 hash functions have length 256, 384, and 512 to be compatible with the increased security of AES
    - they are known as SHA-256, SHA-384, and SHA-512
  - Also, SHA-224 was added to compatibility with 3DES

Hash Function Algorithms

- **Comparison of SHA parameters**

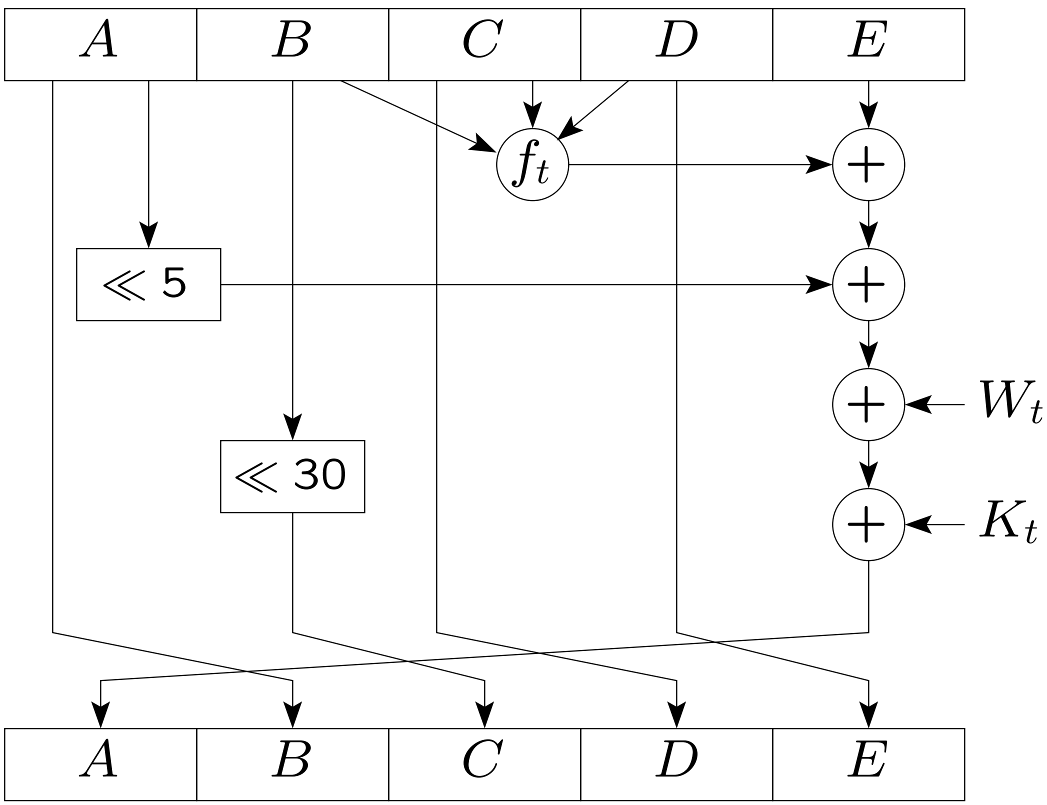
	SHA-1	SHA-256	SHA-384	SHA-512
hash size	160	256	384	512
message size	$< 2^{64}$	$< 2^{64}$	$< 2^{128}$	$< 2^{128}$
block size	512	512	1024	1024
word size	32	32	64	64
number of steps	80	64	80	80
security (birthday attack)	80	128	192	256

Hash Function Algorithms

- **SHA-1 algorithm**
  - **pad the input before processing**
  - **initialize the 5-word (160-bit) buffer with**
    - $A =$ **67452301**; $B =$ **EFCDAB89**; $C =$ **98BADCFE**
    - $D =$ **10325476**; $E =$ **C3D2E1F0**
  - **message is processed in 16 32-bit words**
    - **expand 16 words into 80 words by XORing and shifting**
    - **use 4 rounds of 20 steps each on a message block and the buffer**
  - **the buffer is updated as (t is the step number)**
    $(A, B, C, D, E) =$
    $((E + f_t(B, C, D) + (A \ll 5) + W_t + K_t), A, (B \ll 30), C, D)$

Hash Function Algorithms

- **One step of SHA-1**



Hash Function Algorithms

- **SHA-1 details**
  - t **is the step number**
  - K_t **is the a constant value derived from the sin function**
  - W_t **is derived from the message block** $m_i = W_0 W_1 \ldots W_{15}$ **as**
    - $W_t = W_t$ **for** $t = 0, \ldots, 15$
    - $W_t = (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \ll 1$ **for** $t = 16, \ldots, 79$
- **The difference between SHA-0 and SHA-1 is that SHA-0 doesn't have 1-bit shift in the construction of** $W_{16}, \ldots, W_{79}$

Hash Function Algorithms

- **Security of SHA**
  - brute force attack is harder than in MD5 (160 bits vs. 128 bits)
  - SHA performs more complex transformations that MD5
    - it makes finding collisions more difficult
  - Joux and also Wang et al. found collisions in SHA-0 in 2004
    - collisions can be found in SHA-0 in $< 2^{40}$
  - in 2005 collisions have been found in 58-round "reduced" SHA-1 (2^{33} work)
  - finding collisions in the full version of SHA-1 is estimated at $< 2^{69}$
  - several other results followed

Hash Function Algorithms

- **Search for SHA-3**
  - Feb 2007: NIST announces requests for candidate algorithms for SHA-3 family
  - Oct 2008: 64 algorithms were received
  - Dec 2008: 51 first-round algorithms meeting minimum requirements were announced
  - Jul 2009: 14 second-round candidates were announced
  - Dec 2010: 5 finalists were selected
  - Oct 2012: the winner, Keccak, was announced
  - 2013: controversy about NIST-announced changes
  - Aug 2015: SHA-3 standard was released

Hash Function Algorithms

- **SHA-3 Requirements**
  - **digest sizes of 224, 256, 384, and 512 bits**
  - **support of maximum message length of at least $2^{64} - 1$ bits**
  - **must be implementable in a wide range of hardware and software platforms**
  - **other requirements**
- **Evaluation criteria (ordered)**
  - **security**
  - **cost and performance**
  - **algorithm and implementation characteristics**

SHA-3

- SHA-3 is specified in NIST's FIPS 202 standard
  - it is based on Keccak family of sponge functions
  - the sponge construction is a mode of operation that builds a function mapping variable-length input to variable-length output using a fixed-length permutation and a padding rule
  - Keccak instances call one of seven permutations with SHA-3 using the largest permutation Keccak-f[1600]
  - each permutation uses a round function with simple operations such as XOR, AND and NOT and rotations
  - the design is dictinct from other widely used techniques (SHA-2, AES, etc.)

SHA-3

- **In December 2016, NIST released Special Publication (SP) 800-185 with SHA-3 derived functions:**
  - **cSHAKE is a customizable variant of the SHAKE function used in Keccak and is a building block for all functions below**
  - **KMAC (= Keccak MAC) is a PRF and keyed hash function based on Keccak**
    - **it is faster than HMAC**
  - **TupleHash is a variable-length hash function designed to hash tuples of input strings without trivial collisions**
  - **ParallelHash is a variable-length hash function that can hash very long messages in parallel**

Summary

- **Hash function design**
  - **iterated functions with chaining**
  - **Merkle-Damgard construction**
- **Attacks on hash functions**
  - **birthday attack applies to find collisions**
  - **finding preimage requires brute force search**
- **Customized hash functions**
  - **MD4/MD5**
  - **SHA-0, SHA-1, SHA-2**
  - **new SHA-3**