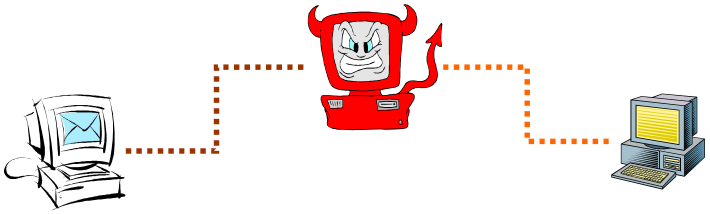
Cryptographic Hash Functions

Topics

- **Overview of Cryptography Hash Function**
- Usages
- Properties
- Hashing Function Structure
- Attack on Hash Function
- The Road to new Secure Hash Standard

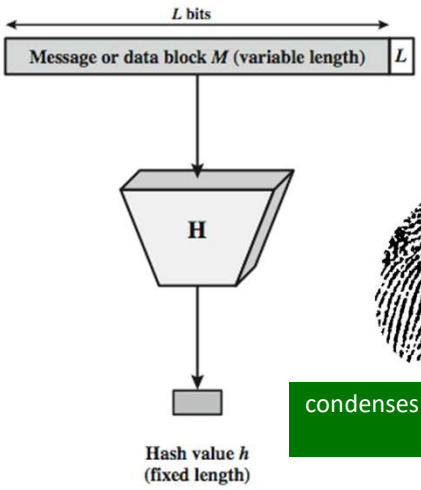
Data Integrity and Source Authentication



- Encryption does not protect data from modification by another party.
  - Why?
- Need a way to ensure that data arrives at destination in its original form as sent by the sender and it is coming from an authenticated source.

Modified ciphertext can often still be decrypted into plaintexts?

Hash Function



- The hash value represents concisely the longer message
  - may called the *message digest*

A message digest is as a "digital fingerprint" of the original document

condenses arbitrary message to fixed size

$$h = H(M)$$

4

A hash function H accepts a variable-length block of data M as input and produces a fixed-size hash value h = H(M). A "good" hash function has the property that the results of applying the function to a large set of inputs will produce outputs that are evenly distributed, and apparently random. In general terms, the principal object of a hash function is data integrity. A change to any bit or bits in M results, with high probability, in a change to the hash code. The kind of hash function needed for security applications is referred to as a cryptographic hash function. A cryptographic hash function is an algorithm for which it is computationally infeasible (because no attack is significantly more efficient than brute force) to find either (a) a data object that maps to a pre-specified hash result (the one-way property) or (b) two data objects that map to the same hash result (the collision-free property). Because of these characteristics, hash functions are often used to determine whether or not data has changed.

Stallings Figure 11.1 depicts the general operation of a cryptographic hash function. Typically, the input is padded out to an integer multiple of some fixed length (e.g., 1024 bits) and the padding includes the value of the length of the original message in bits. The length field is a security measure to increase the difficulty for an attacker to produce an alternative message with the same hash value.

Chewing functions

- Hashing function as “chewing” or “digest” function

Hash functions, Tuma and properties.

One-wayness

First and second preimage resistance

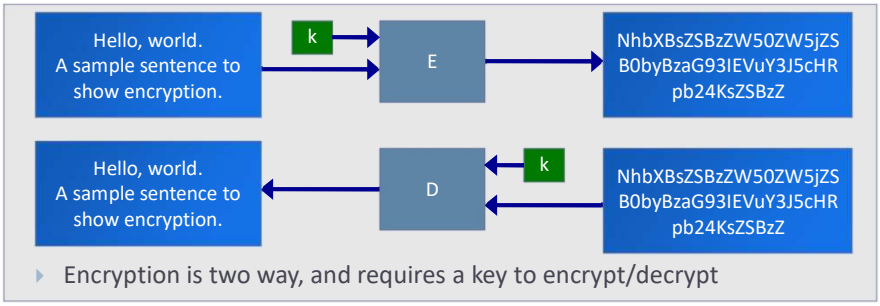
Collision resistance

After a while he started to call them chewing functions

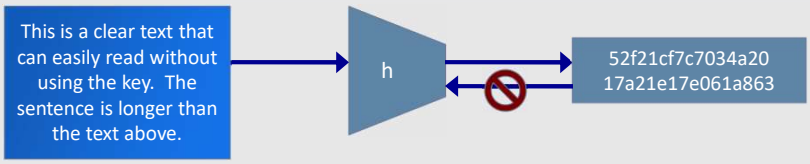
-->

So here we have some classic chewing functions

Hashing V.S. Encryption



- Encryption is two way, and requires a key to encrypt/decrypt



- Hashing is one-way. There is no 'de-hashing'

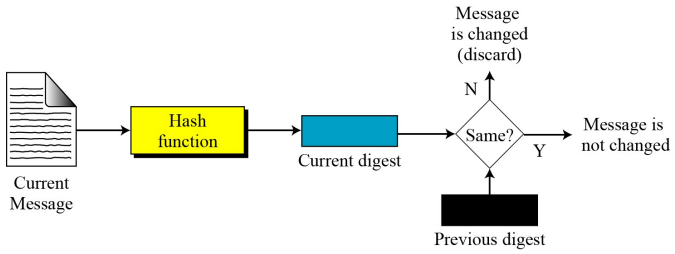
Motivation for Hash Algorithms

- Intuition
  - Limitation on non-cryptographic checksum
  - Very possible to construct a message that matches the checksum
- Goal
  - Design a code where the original message can not be inferred based on its checksum
  - such that an accidental or intentional change to the message will change the hash value

Hash Function Applications

- Used Alone
  - Fingerprint -- file integrity verification, public key fingerprint
  - Password storage (one-way encryption)
- Combined with encryption functions
  - Hash based Message Authentication Code (HMAC)
    - protects both a message's integrity and confideltaility
  - Digital signature
    - Ensuring Non-repudiation
    - Encrypt hash with private (signing) key and verify with public (verification) key

Integrity



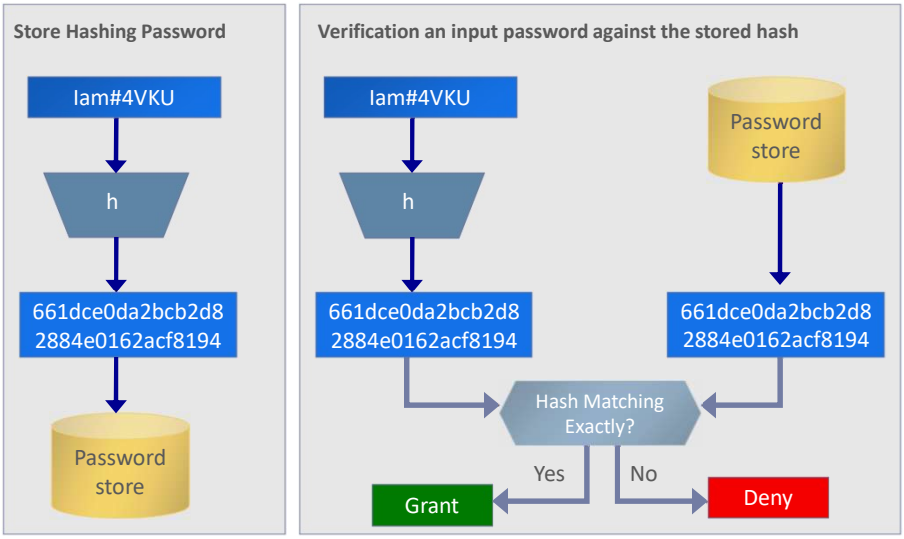
- to create a one-way password file
  - store hash of password not actual password
- for intrusion detection and virus detection
  - keep & check hash of files on system

Hash functions are commonly used to create a one-way password file. Chapter 20 explains a scheme in which a hash of a password is stored by an operating system rather than the password itself. Thus, the actual password is not retrievable by a hacker who gains access to the password file. In simple terms, when a user enters a password, the hash of that password is compared to the stored hash value for verification. This approach to password protection is used by most operating systems.

Hash functions can be used for intrusion detection and virus detection. Store H(F) for each file on a system and secure the hash values (e.g., on a CD-R that is kept secure). One can later determine if a file has been modified by recomputing H(F). An intruder would need to change F without changing H(F).

A cryptographic hash function can be used to construct a pseudorandom function (PRF) or a pseudorandom number generator (PRNG). A common application for a hash-based PRF is for the generation of symmetric keys. We discuss this application in Chapter 12.

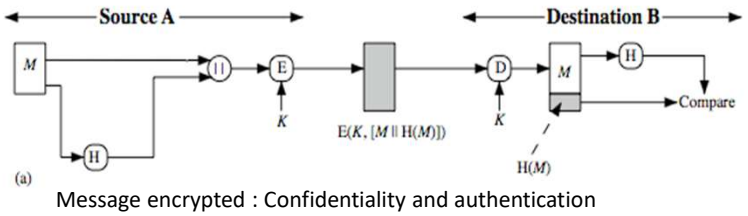
Password Verification



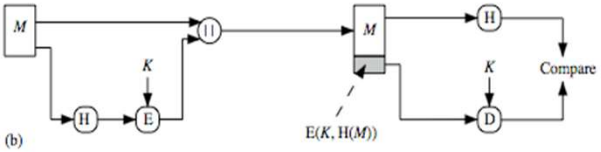
Topics

- Overview of Cryptography Hash Function
- **Usages**
- Properties
- Hashing Function Structure
- Attack on Hash Function
- The Road to new Secure Hash Standard

Hash Function Usages (I)



Message encrypted : Confidentiality and authentication



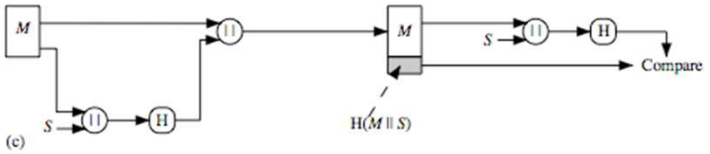
Message unencrypted: Authentication

Message authentication is a mechanism or service used to verify the integrity of a message, by assuring that the data received are exactly as sent. Stallings Figure 11.2 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows:

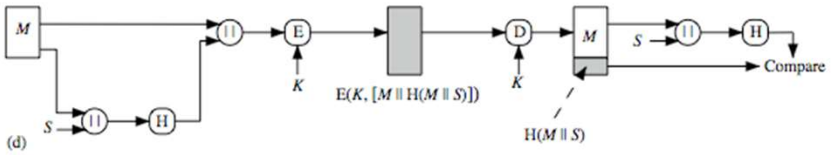
a.The message plus concatenated hash code is encrypted using symmetric encryption. Since only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication.

b.Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications not requiring confidentiality.

Hash Function Usages (II)



Message encrypted : Authentication (no encryption needed!)



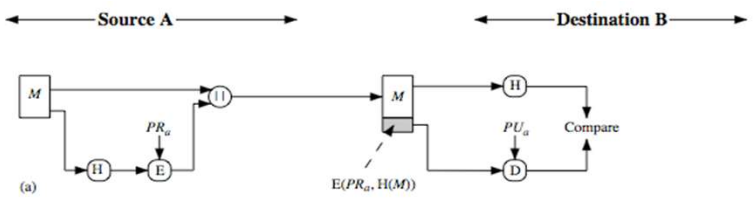
Message unencrypted: Authentication, confidentiality

(c) Shows the use of a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses S, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

(d) Confidentiality can be added to the approach of (c) by encrypting the entire message plus the hash code.

When confidentiality is not required, method (b) has an advantage over methods (a) and (d), which encrypts the entire message, in that less computation is required.

Hash Function Usages (III)



Authentication, digital signature

Authentication, digital signature, confidentiality

Another important application, which is similar to the message authentication application, is the digital signature. The operation of the digital signature is similar to that of the MAC. In the case of the digital signature, the hash value of a message is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature. In this case an attacker who wishes to alter the message would need to know the user's private key. As we shall see in Chapter 14, the implications of digital signatures go beyond just message authentication. Stallings Figure 11.3 illustrates, in a simplified fashion, how a hash code is used to provide a digital signature:

a.The hash code is encrypted, using public-key encryption and using the sender's private key. As with Figure 11.2b, this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

b.If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.

Topics

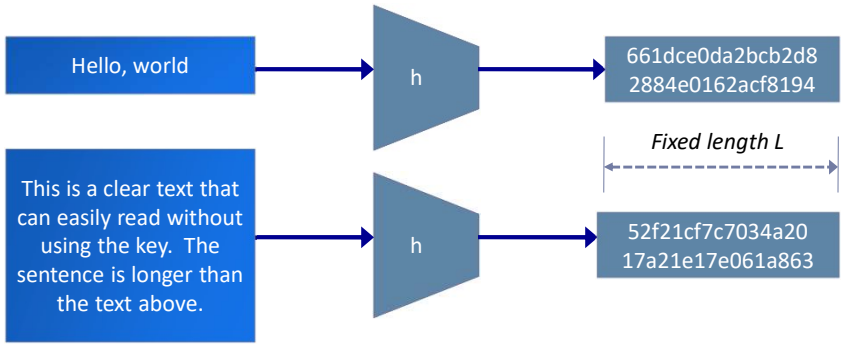
- Overview of Cryptography Hash Function
- Usages
- **Properties**
- Hashing Function Structure
- Attack on Hash Function
- The Road to new Secure Hash Standard

Hash Function Properties

- Arbitrary-length message to fixed-length digest
- Preimage resistant (**One-way property**)
- Second preimage resistant (**Weak collision resistant**)
- Collision resistant (**Strong collision resistance**)

The first properties are requirements for the practical application of a hash function. The second property, preimage (for a hash value *h = H(x),* we say that x is the **preimage** of ***h***) resistant, is the one-way property: it is easy to generate a code given a message, but virtually impossible to generate a message given a code. This property is important if the authentication technique involves the use of a secret value The fthird property, second preimage resistant, guarantees that it is impossible to find an alternative message with the same hash value as a given message. This prevents forgery when an encrypted hash code is used. If the fouth property, collision resistant, is also satisfied, then it is referred to as a strong hash function. A strong hash function protects against an attack in which one party generates a message for another party to sign. The final requirement, **pseudorandomness**, has not traditionally been listed as a requirement of cryptographic hash functions, but is more or less implied.

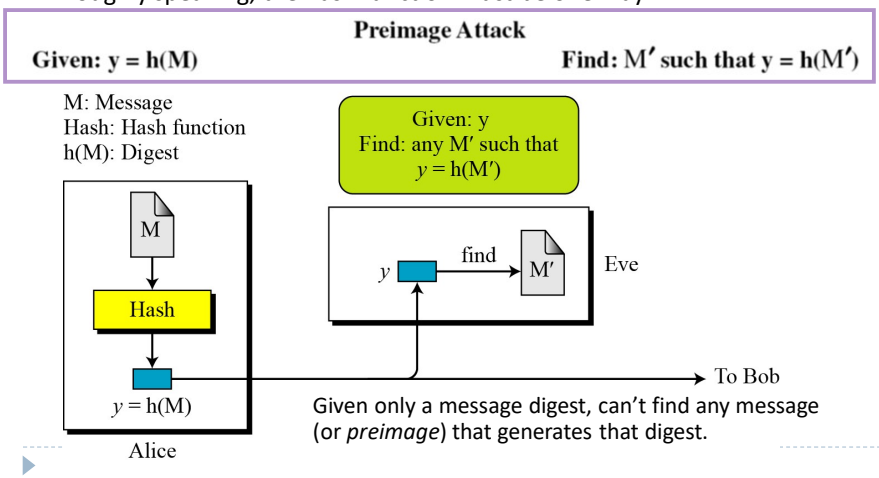
Properties : Fixed length



- Arbitrary-length message to fixed-length digest

Preimage resistant

- This measures how difficult to devise a message which hashes to the known digest
- Roughly speaking, the hash function must be one-way.



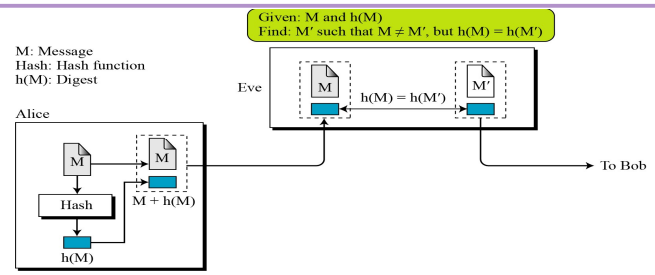
Exploiting weak preimage resistance: If we are able to "work backwards" from a hash and create some text that produces the same hash, we can use this to beat hashed passwords. We won't ever know the actual input data that was used, but that doesn't matter.Looking at the flow for validating against hashed passwords, all that matters is that the hashes match, not the passwords, so if we can find*any other text that produces the stored hash, we'll be granted access. "Collisions" mean "more than one password will be accepted".*

Second preimage resistant

- This measures how difficult to devise a message which hashes to the known digest and its message

Second Preimage Attack

Given: M and h(M) **Find: M′ ≠ M such that h(M) = h(M′)**



- 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

This seems like a somewhat easier case of the previous item: the goal is to produce a new input that generates the given hash, but this time we know the original text that created it. **Exploiting second weak preimage resistance: As with preimage resistance, we want to fool somebody into authenticating *our data as genuine, and we'd most likely use this when trying to introduce a corrupted software distribution.We saw that many other organizations publishes software and matching md5 checksums, and if we are able to maliciously modify the source code but nevertheless keep the same checksum, downloaders around the globe will accept badware as genuine.***

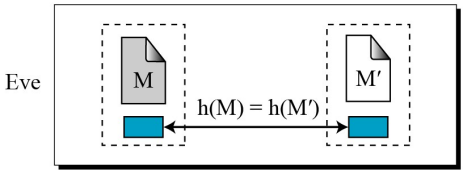
Collision Resistant

Collision Attack

Given: none **Find: M′ ≠ M such that h(M) = h(M′)**

M: Message
Hash: Hash function
h(M): Digest

Find: M and M′ such that M ≠ M′, but h(M) = h(M′)



- 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

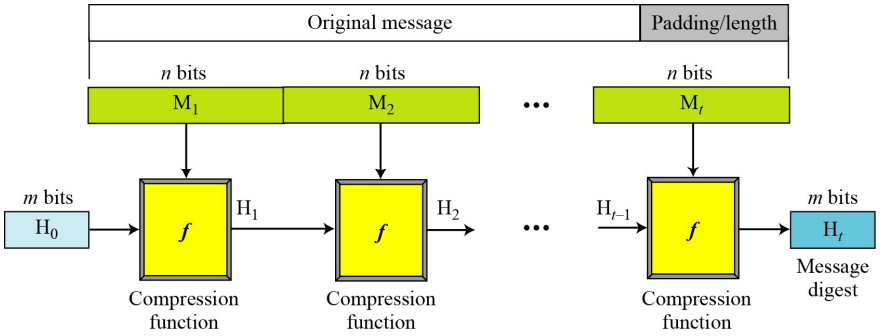
Topics

- Overview of Cryptography Hash Function
- Usages
- Properties
- **Hashing Function Structure**
- Attack on Hash Function
- The Road to new Secure Hash Standard

Two Group of Compression Functions

- The compression function is made from scratch
  - Message Digest
- A symmetric-key block cipher serves as a compression function
  - Whirlpool

Merkle-Damgard Scheme



- Well-known method to build cryptographic hash function
- A message of arbitrary length is broken into blocks
  - length depends on the compression function *f*
  - padding the size of the message into a multiple of the block size.
  - sequentially process blocks , taking as input the result of the hash so far and the current message block, with the final fixed length output

Damgard and Merkle greatly influenced cryptographic hash function design by defining a hash function in terms of what is called a *compression functio*n.

A compression function takes a fixed-length input and returns a shorter, fixed-length output.

Given a compression function, a hash function can be defined by repeated applications of the compression function until the entire message has been processed.

Take a long message, break it into blocks

$M_1, M_2, M_3 \ldots M_k$ (pad out last block)

Let *f* be a "compression function" that operates on a block and the current *h*-bit state and "mixes" the block into the state

Last output of compression function is the *h*-bit message digest.

Hash Functions Family

- **MD (Message Digest)**
  - Designed by Ron Rivest
  - Family: MD2, MD4, MD5
- **SHA (Secure Hash Algorithm)**
  - Designed by NIST
  - Family: SHA-0, SHA-1, and SHA-2
    - SHA-2: SHA-224, SHA-256, SHA-384, SHA-512
    - SHA-3: New standard in competition
- **RIPEMD (Race Integrity Primitive Evaluation Message Digest)**
  - Developed by Katholieke University Leuven Team
  - Family : RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320

MD5, SHA-1, and RIPEMD-160

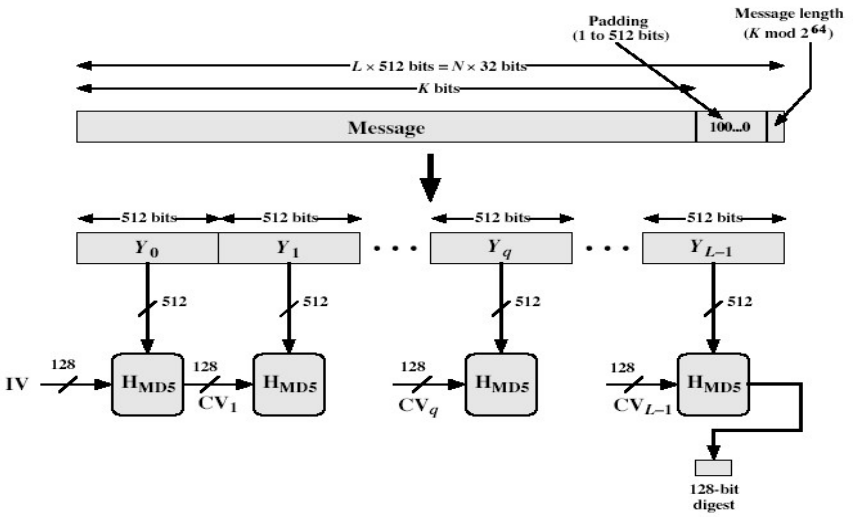
	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	∞	$2^{64} - 1$ bits	$2^{64} - 1$ bits
Primitive logical functions	4	4	5
Additive constants used	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

25

MD2, MD4 and MD5

- Family of one-way hash functions by Ronald Rivest
  - All produces 128 bits hash value
- **MD2: 1989**
  - Optimized for 8 bit computer
  - Collision found in 1995
- **MD4: 1990**
  - Full round collision attack found in 1995
- **MD5: 1992**
  - Specified as Internet standard in RFC 1321
  - since 1997 it was theoretically not so hard to create a collision
  - Practical Collision MD5 has been broken since 2004
  - CA attack published in 2007

MD5 Overview

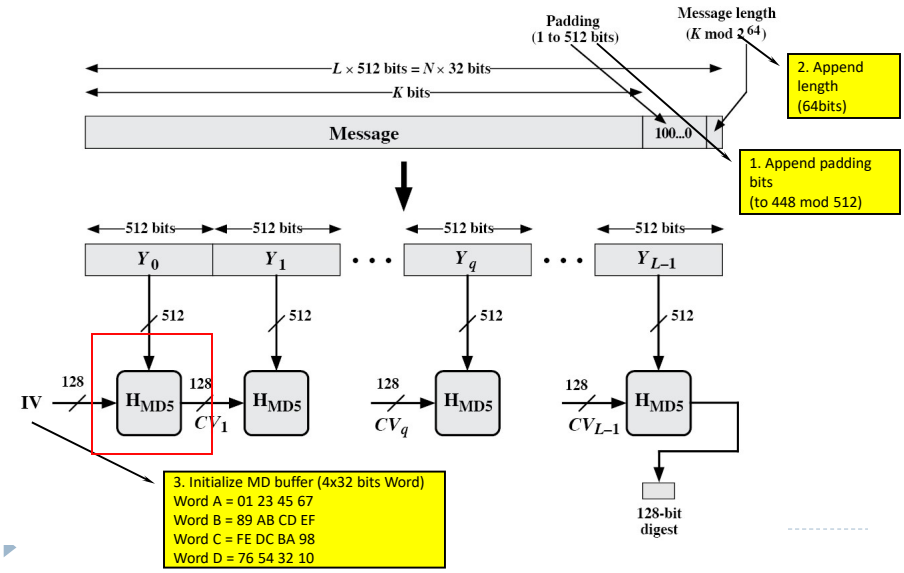


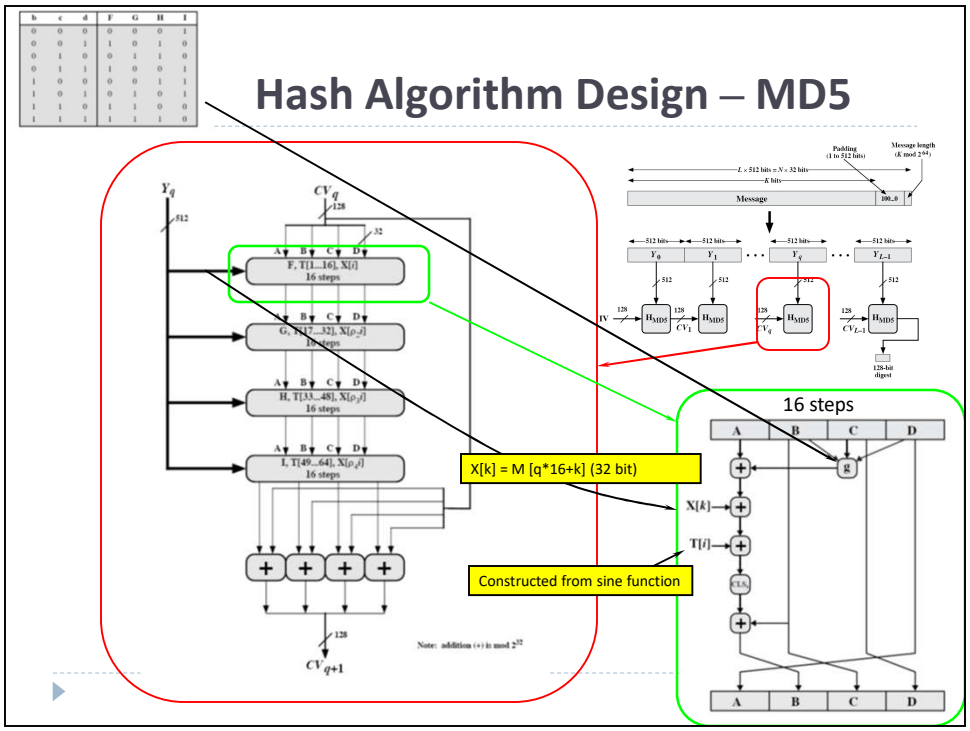
Stallings Fig 12.1

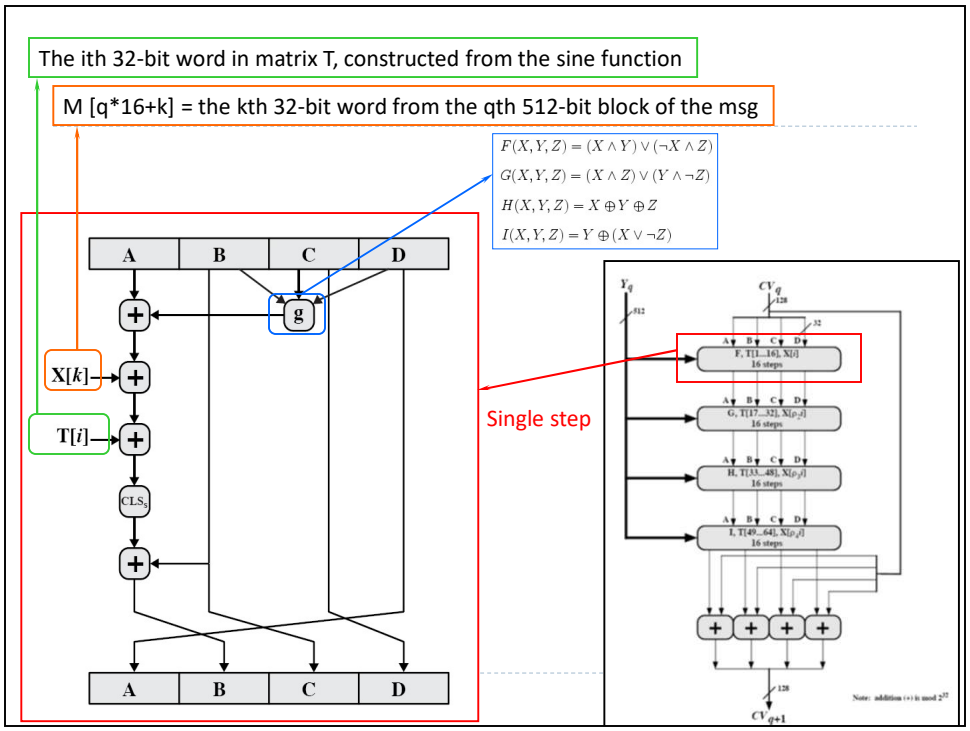
Topics

- Overview of Cryptography Hash Function
- Usages
- Properties
- **Hashing Function Structure**
  - **MD5**
  - SHA
- Attack on Hash Function
- The Road to new Secure Hash Standard

MD5 Overview

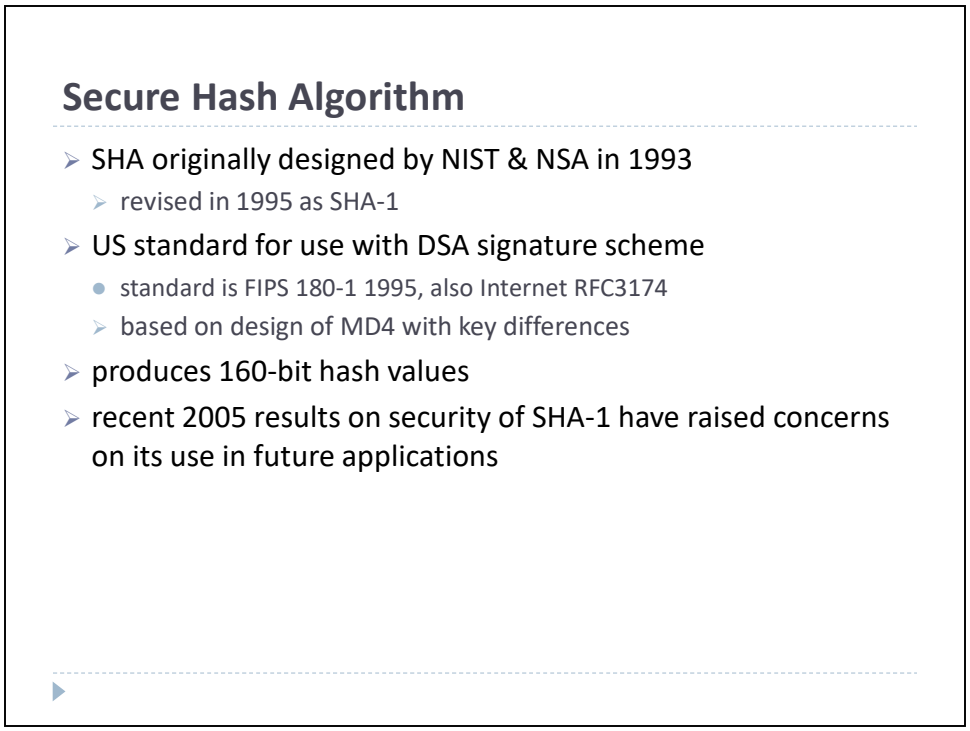



Hash Algorithm Design – MD5
X[k] = M [q*16+k] (32 bit)
Constructed from sine function
16 steps


The ith 32-bit word in matrix T, constructed from the sine function
M [q*16+k] = the kth 32-bit word from the qth 512-bit block of the msg
$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
$G(X,Y,Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$
$H(X,Y,Z) = X \oplus Y \oplus Z$
$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$
A
B
C
D
g
X[k]
T[i]
CLS
Single step

Topics

- Overview of Cryptography Hash Function
- Usages
- Properties
- **Hashing Function Structure**
  - MD5
  - **SHA**
- Attack on Hash Function
- The Road to new Secure Hash Standard



In recent years, the most widely used hash function has been the Secure Hash Algorithm (SHA). The Secure Hash Algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993; a revised version was issued as FIPS 180-1 in 1995 and is generally referred to as SHA-1. The actual standards document is entitled Secure Hash Standard. SHA is based on the hash function MD4 and its design closely models MD4. SHA-1 produces a hash value of 160 bits. In 2005, a research team described an attack in which two separate messages could be found that deliver the same SHA-1 hash using 2^69 operations, far fewer than the 2^80 operations previously thought needed to find a collision with an SHA-1 hash [WANG05]. This result has hastened the transition to newer, longer versions of SHA.

Revised SHA

- NIST issued revision FIPS 180-2 in 2002
- adds 3 additional versions of SHA
  - SHA-256, SHA-384, SHA-512
- designed for compatibility with increased security provided by the AES cipher
- structure & detail is similar to SHA-1
- hence analysis should be similar
- but security levels are rather higher

In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512. Collectively, these hash algorithms are known as SHA-2. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1, hence analyses should be similar. A revised document was issued as FIP PUB 180-3 in 2008, which added a 224-bit version. SHA-2 is also specified in RFC 4634, which essentially duplicates the material in FIPS 180-3, but adds a C code implementation.

In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010.

SHA Versions

	MD5	SHA-0	SHA-1	SHA-224	SHA-256	SHA-384	SHA-512
Digest size	128	160	160	224	256	384	512
Message size	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$	$2^{64}-1$	$2^{128}-1$	$2^{128}-1$
Block size	512	512	512	512	512	1024	1024
Word size	32	32	32	32	32	64	64
# of steps	64	64	80	64	64	80	80

Full collision found

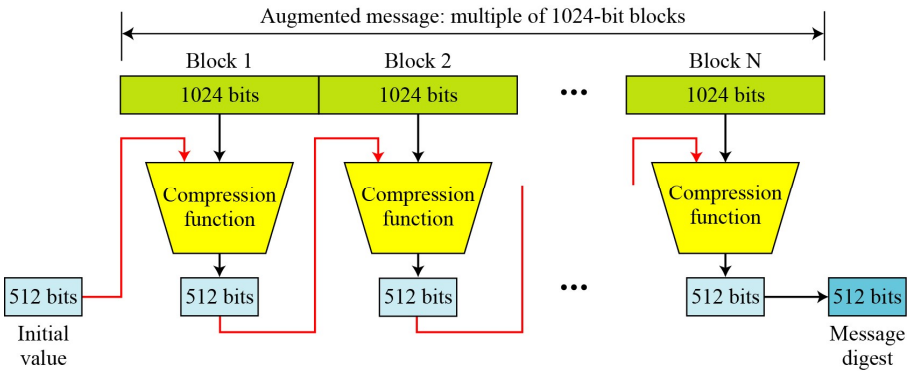
Stallings Table 11.3 provides a comparison of the various parameters for the SHA hash functions.

Sample Processing

Type	bits	data processed
MD5	128	469.7 MB/s
SHA-1	160	339.4 MB/s
SHA-512	512	177.7 MB/s

- Mac Intel 2.66 Ghz core i7
- 1024 bytes block of data

SHA-512 Overview



Now examine the structure of SHA-512, noting that the other versions are quite similar.

SHA-512 follows the structure depicted in Stallings Figure 11.8. The processing consists of the following steps:

• Step 1: Append padding bits, consists of a single 1-bit followed by the necessary number of 0-bits, so that its length is congruent to 896 modulo 1024

• Step 2: Append length as an (big-endian) unsigned 128-bit integer

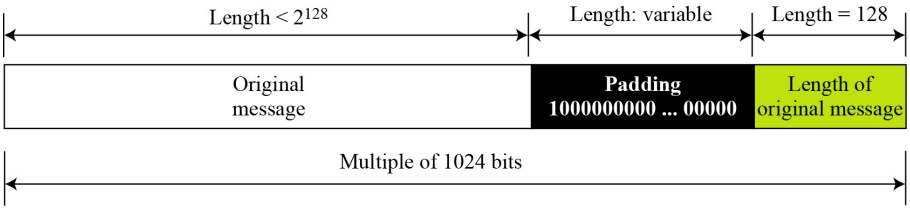
• Step 3: Initialize hash buffer to a set of 64-bit integer constants (see text)

• Step 4: Process the message in 1024-bit (128-word) blocks, which forms the heart of the algorithm. Each round takes as input the 512-bit buffer value H_i, and updates the contents of that buffer.

• Step 5: Output the final state value as the resulting hash

See text for more details.

Padding and length field in SHA-512

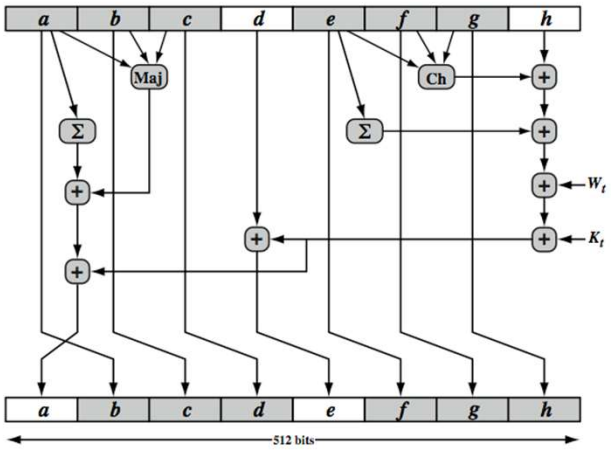


- **What is the number of padding bits if the length of the original message is 2590 bits?**
- We can calculate the number of padding bits as follows:

$$|P| = (-2590 - 128) \bmod 1024 = -2718 \bmod 1024 = 354$$

- The padding consists of one 1 followed by 353 0's.

SHA-512 Round Function



The structure of each of the 80 rounds is shown in Stallings Figure 11.10. Each 64-bit word is shuffled along one place, and in some cases manipulated using a series of simple logical functions (ANDs, NOTs, ORs, XORs, ROTates), in order to provide the avalanche & completeness properties of the hash function. The elements are:

Ch(e,f,g) = (e AND f) XOR (NOT e AND g)

Maj(a,b,c) = (a AND b) XOR (a AND c) XOR (b AND c)

∑(a) = ROTR(a,28) XOR ROTR(a,34) XOR ROTR(a,39)

∑(e) = ROTR(e,14) XOR ROTR(e,18) XOR ROTR(e,41)

+ = addition modulo 2^64

Kt = a 64-bit additive constant

Wt = a 64-bit word derived from the current 512-bit input block.

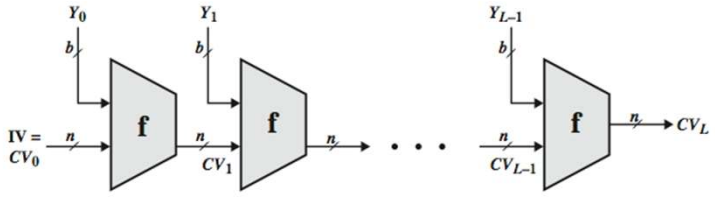
Six of the eight words of the output of the round function involve simply permutation (*b, c, d, f, g, h*) by means of rotation. This is indicated by shading in Figure 11.10. Only two of the output words (*a, e)* are generated by substitution. Word e is a function of input variables *d, e, f, g, h,* as well as the round word W t and the constant Kt. Word a is a function of all of the input variables, as well as the round word W t and the constant Kt.

Topics

- Overview of Cryptography Hash Function
- Usages
- Properties
- Hashing Function Structure
  - MD5
  - SHA
- **Attack on Hash Function**
- The Road to new Secure Hash Standard

Hash Function Cryptanalysis

- cryptanalytic attacks exploit some property of algorithm so faster than exhaustive search
- hash functions use iterative structure
  - process message in blocks (incl length)
- attacks focus on collisions in function f



As with encryption algorithms, cryptanalytic attacks on hash functions seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. In recent years, have much effort, and some successes, in developing cryptanalytic attacks on hash functions. Must consider the overall structure of a typical secure hash function, referred to as an iterated hash function, as indicated in Stallings Figure 11.7. This was proposed by Merkle and is the structure of most hash functions in use today. The hash function takes an input message and partitions it into *L* fixed-sized blocks of *b* bits each. If necessary, the final block is padded to *b* bits. The final block also includes the value of the total length of the input to the hash function. The inclusion of the length makes the job of the opponent more difficult. The hash algorithm involves repeated use of a compression function, *f*, that takes two inputs (an *n*-bit input from the previous step, called the chaining variable, and a *b*-bit block) and produces an *n*-bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Often, $b > n$; hence the term compression. The motivation for this iterative structure stems from the observation by Merkle and Damgard that if the compression function is collision resistant, then so is the resultant iterated hash function. Therefore, the structure can be used to produce a secure hash function to operate on a message of any length. Cryptanalysis of hash functions focuses on the internal structure of f and is based on attempts to find efficient techniques for producing collisions for a single execution of f. Once that is done, the attack must take into account the fixed value of IV. The attack on f depends on exploiting its internal structure. The attacks that have been mounted on hash functions are rather complex and beyond our scope here.

Attacks on Hash Functions

- brute-force attacks and cryptanalysis
  - cryptanalytic attacks exploit some property of algorithm so faster than brute-force
- a preimage or second preimage attack
  - find y such that $H(y)$ equals a given hash value
- collision resistance
  - find two messages x & y with same hash so $H(x) = H(y)$

"*md5 and sha1 are both clearly broken (in terms of collision-resistance*"
Ron Rivest

http://mail.python.org/pipermail/python-dev/2005-December/058850.html

As with encryption algorithms, there are two categories of attacks on hash functions: brute-force attacks and cryptanalysis. A brute-force attack does not depend on the specific algorithm but depends only on bit length. In the case of a hash function, a brute-force attack depends only on the bit length of the hash value. A cryptanalysis, in contrast, is an attack based on weaknesses in a particular cryptographic algorithm.

For a preimage or second preimage attack, an adversary wishes to find a value *y* such that *H(y)* is equal to a given hash value h. The brute force method is to pick values of y at random and try each value until a collision occurs. For an *m-bit* hash value, the level of effort is proportional to 2^m. Specifically, the adversary would have to try, on average, 2^{m-1} values of y to find one that generates a given hash value h.

For a collision resistant attack, an adversary wishes to find two messages or data blocks, x and *y,* that yield the same hash function: H(x) = H(y). This requires much less effort than a preimage or second preimage attack. The effort required is explained by a mathematical result referred to as the birthday paradox (next slide).

If collision resistance is required, then the value $2^{m/2}$ determines the strength of the hash code against brute-force attacks. Van Oorschot and Wiener presented a design for a $10 million collision search machine for MD5, which has a 128-bit hash length, that could find a collision in 24 days. Thus a 128-bit code may be viewed as inadequate. The next step up, if a hash code is treated as a sequence of 32 bits, is a 160-bit hash length. With a hash length of 160 bits, the same search machine would require over four thousand years to find a collision. With today's technology, the time would be much shorter, so that 160 bits now appears suspect.

Topics

- Overview of Cryptography Hash Function
- Usages
- Properties
- Hashing Function Structure
  - MD5
  - SHA
- Attack on Hash Function
- **The Road to new Secure Hash Standard**

The need of new Hash standard

- *MD5 should be considered cryptographically broken and unsuitable for further use*, US CERT 2010
- In 2004, a collision for the full SHA-0 algorithm was announced
- SHA-1 not yet fully "**broken**"
  - but similar to the broken MD5 & SHA-0
  - so considered insecure and be fade out
- SHA-2 (esp. SHA-512) seems secure
  - shares same structure and mathematical operations as predecessors so have concern

As yet, SHA-1 has not yet been "broken". That is, no one has demonstrated a technique for producing collisions in less than brute-force time. However, because SHA-1 is very similar in structure and in the basic mathematical operations used to MD5 and SHA-0, both of which have been broken, SHA-1 is considered insecure and has been phased out for SHA-2.

SHA-2, particularly the 512-bit version, would appear to provide unassailable security. However, SHA-2 shares the same structure and mathematical operations as its predecessors, and this is a cause for concern. Because it will take years to find a suitable replacement for SHA-2, should it become vulnerable, NIST decided to begin the process of developing a new hash standard. Accordingly, NIST announced in 2007 a competition to produce the next generation NIST hash function, to be called SHA-3. NIST would like to have a new standard in place by the end of 2012, but emphasizes that this is not a fixed timeline.

SHA-3 Requirements

- NIST announced in 2007 a competition for the SHA-3 next gen hash function
- Replace SHA-2 with SHA-3 in any use
  - so use same hash sizes
- preserve the nature of SHA-2
  - so must process small blocks (512 / 1024 bits)
- evaluation criteria
  - security close to theoretical max for hash sizes
  - cost in time & memory
  - characteristics: such as flexibility & simplicity

The basic requirements that must be satisfied by any candidate for SHA-3 are:

1.It must be possible to replace SHA-2 with SHA-3 in any application by a simple drop-in substitution. Therefore, SHA-3 must support hash value lengths of 224, 256, 384, and 512 bits.

2.SHA-3 must preserve the online nature of SHA-2. That is, the algorithm must process comparatively small blocks (512 or 1024 bits) at a time instead of requiring that the entire message be buffered in memory before

Beyond these basic requirements, NIST has defined a set of evaluation criteria. These criteria are designed to reflect the requirements for the main applications supported by SHA-2, and are:

• Security: The strength of SHA-3 should be close to the theoretical maximum for the different required hash sizes, and for both preimage resistance and collision resistance. SHA-3 algorithms must be designed to resist any potentially successful attack on SHA-2 functions

• Cost: be both time and memory efficient over a range of hardware platforms.

• Algorithm and implementation characteristics: such as flexibility (e.g., tunable parameters for security/performance tradeoffs, opportunity for parallelization, and so on), and simplicity (which makes it easier to analyze the security properties of the algorithm)

Timeline Competition

- **Nov 2007**: Announce public competition
- **Oct 2008**: 64 Entries
- **Dec 2008**: 51 Entries as 1st Round
- **Jul 2009**: 14 Entries as 2nd Round
- **Dec 2010**: 5 Entries as 3rd Round
- **Jan 2011:** Final packages submission and enter public comments
- **2012:** *SHA-3 winner announcement (Still in progress)*

Five SHA-3 Finalists

- BLAKE
- Grøstl
- JH
- Keccak
- Skien

http://csrc.nist.gov/groups/ST/hash/sha-3/Round3/submissions_rnd3.html

Summary

- Hash functions are keyless
  - Applications for digital signatures and in message authentication codes
- The three security requirements for hash functions are
  - one-wayness, second preimage resistance and collision resistance
- MD5 and SHA-0 is insecure
- Serious security weaknesses have been found in SHA-1
  - should be phased out
  - SHA-2 appears to be secure
  - May use SHA-512 and use the first 256 bytes
- The ongoing SHA-3 competition will result in new standardized hash functions in a next year