LECTURE 5

Hash functions

Telecommunication systems department

Lecturer: assistant professor Persikov Anatoliy Valentinovich

BASIC PROPERTIES

Cryptographic hash functions map strings of different lengths to short, fixed-size, outputs. These functions, e.g., MD5, SHA-1 or SHA-2, are primarily designed to be **collision resistant**.

This means that if we represent such a hash function by F, then it should be infeasible for an adversary to find two strings x and x' such that $F(x) = F(x')$.

Notice that this cryptographic notion does not involve any secret key. Indeed, the collision-resistance property is usually attached to keyless functions. The prime motivation for such functions is to be combined with digital signatures in a way that makes these signatures more efficient and yet unforgeable. For that application it is required that the function be publicly computable and, in particular, that it involve no secret key.

BASIC PROPERTIES

In addition to the basic collision-resistance property, cryptographic hash functions are usually designed to have some randomness-like properties, like good mixing properties, independence of input/output, unpredictability of the output when parts of the input are unknown, etc. Not only do these properties help in making it harder to find collisions, but also they help to randomize the input presented to the signature algorithm (e.g., RSA) as usually required for the security of these functions.

It is the combination of these properties attributed to cryptographic hash functions that make them so attractive for many uses beyond the original design as collision-resistant functions. These functions have been proposed as the basis for pseudorandom generation, block ciphers, random transformation, and message authentication codes. We concentrate on a particular class of cryptographic hash functions, which we call iterated constructions.

REQUIREMENTS FOR A HASH FUNCTION

The purpose of a hash function is to produce a "fingerprint" of a file, message, or other block of data. To be useful for message authentication, a hash function H must have the following properties:

1) H can be applied to a block of data of any size.

2) H produces a fixed-length output.

3) $H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.

4) For any given value h, it is computationally infeasible to find x such that $H(x) = h$. This is sometimes referred to in the literature as **the one-way property**.

5) For any given block x, it is computationally infeasible to find $y \neq x$ such that $H(y) = H(x)$. This is sometimes referred to as **weak collision resistance**.

6) It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$. This is sometimes referred to as **strong collision resistance**.

REQUIREMENTS FOR A HASH FUNCTION

The **first three properties** are requirements for the practical application of a hash function to message authentication.

The **fourth property**, the one-way property, states that 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 secret value itself is not sent; however, if the hash function is not one way, an attacker can easily discover the secret value: If the attacker can observe or intercept a transmission, the attacker obtains the message M and the hash code $C = H(S_{AB}||M)$. The attacker then inverts the hash function to obtain $S_{AB}||M = H^1(C)$. Because the attacker now has both M and $S_{AB}||M$, it is a trivial matter to recover S_{AB}.

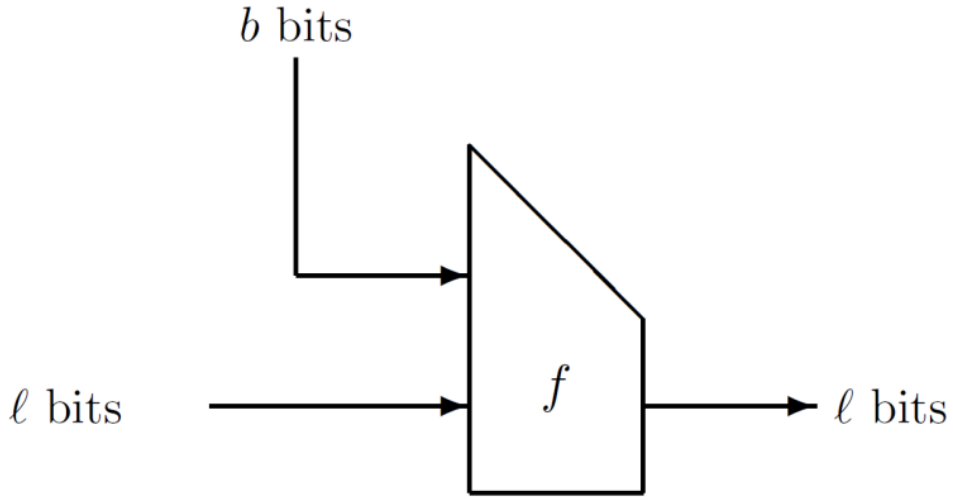
REQUIREMENTS FOR A HASH FUNCTION

The **fifth property** guarantees that an alternative message hashing to the same value as a given message cannot be found. This prevents forgery when an encrypted hash code is used. For these cases, the opponent can read the message and therefore generate its hash code. However, because the opponent does not have the secret key, the opponent should not be able to alter the message without detection. If this property were not true, an attacker would be capable of the following sequence: First, observe or intercept a message plus its encrypted hash code; second, generate an unencrypted hash code from the message; third, generate an alternate message with the same hash code.

The **sixth property** refers to how resistant the hash function is to a type of attack known as the birthday attack, which we examine shortly.

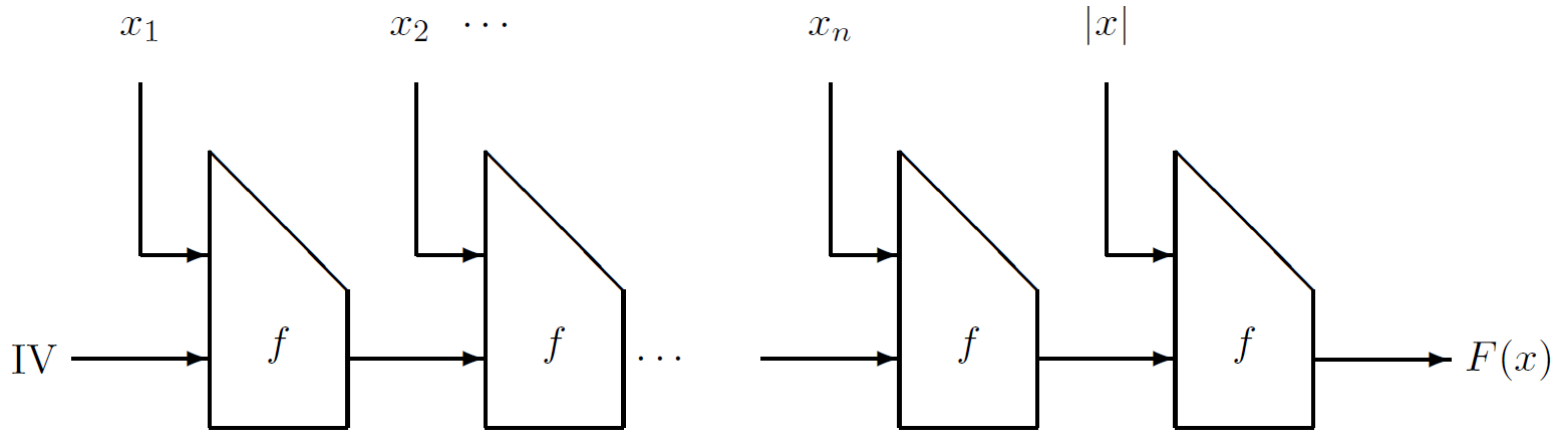
ITERATED CONSTRUCTIONS

A particular methodology for constructing collision-resistant hash function has been proposed by Merkle (and later by Damgard) This methodology forms the basis for the design of the most common cryptographic hash functions like MD5 and SHA-1. It is based on a basic component called **compression function** which processes short fixed-length inputs, and is then iterated in a particular way in order to hash arbitrarily long inputs. Such a compression function, which we denote by f, accepts two inputs: a chaining variable of length and a block of data of length b. (For MD5 and SHA-1 we have $b = 512$, while for the first $l = 128$ and for the second $l = 160$.



ITERATED CONSTRUCTIONS

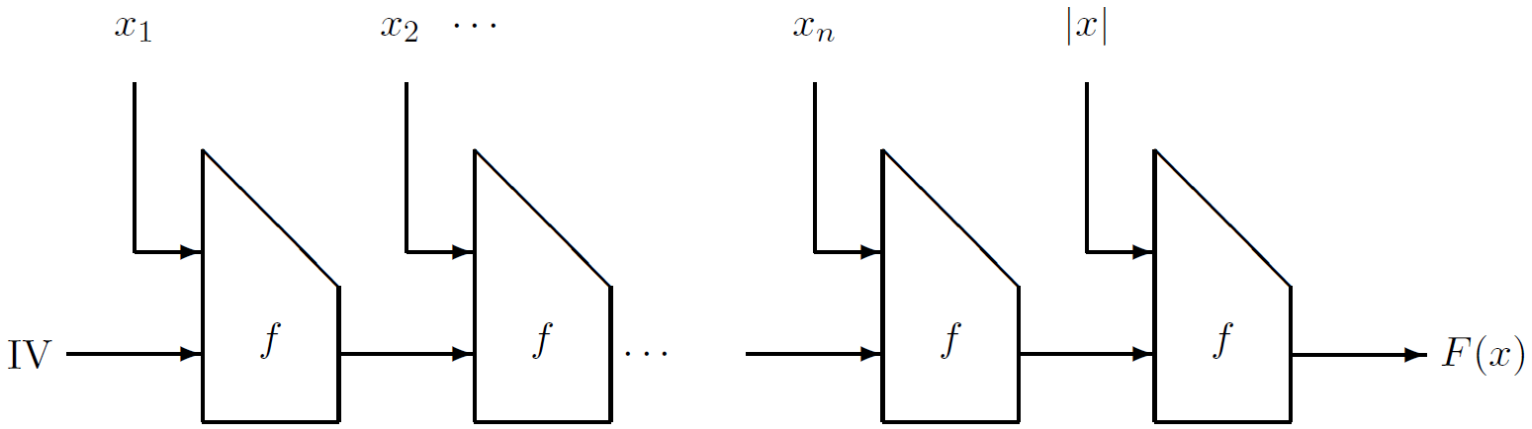
The operation of the iterated hash function is as follows. First, an b-bit value IV is fixed. Next an input is hashed by iterating the compression function. That is, if $x = x_1, x_2, \ldots, x_n$ is the input, where the x_i's are blocks of length b each and n is an arbitrary number of blocks, then let $x_{n+1} = |x|$ be the message length. The value of the iterated function F on x is h_{n+1} where $h_0 = IV$ and $h_i = f(h_{i-1}; x_i)$ for $i = 1; 2; \ldots ; n + 1$.



ITERATED CONSTRUCTIONS

Notice that a way to pad messages to an exact multiple of b bits needs to be defined, in particular, MD5 and SHA pad inputs to always include an encoding of their length.

The motivation for this iterative structure arises from the observation (of Merkle and Damgard) that if the compression function is collision-resistant then so is the resultant iterated hash function. (The converse is not necessarily true). Thus, this structure provides a general design criterion for collision resistant hash functions since. Namely, it reduces the problem to the design of a collision resistant function on inputs of some fixed size.



We will use the symbol f to denote the compression function, and F to denote the associated iterated hash which we assume to include a standard way to pad inputs to an exact multiple of b bits.

SHA-2

Standard FIPS 180-3 specifies five secure hash algorithms, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512. All five of the algorithms are iterative, one-way hash functions that can process a message to produce a condensed representation called a **message digest**. These algorithms enable the determination of a message's integrity: any change to the message will, with a very high probability, result in a different message digest. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers (bits).

Each algorithm can be described in two stages: preprocessing and hash computation.

Preprocessing involves padding a message, parsing the padded message into m-bit blocks, and setting initialization values to be used in the hash computation.

The hash computation generates a message schedule from the padded message and uses that schedule, along with functions, constants, and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest.

SHA-2 ALGORITHMS

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512

The five algorithms differ most significantly in the number of bits of security that are provided for the data being hashed. Security strengths of these five hash functions and the system as a whole when each of them is used with other cryptographic algorithms, such as digital signature algorithms and keyed-hash message authentication codes. Additionally, the five algorithms differ in terms of the size of the blocks and words of data that are used during hashing.

SHA-256 PARAMETERS

$a, b, c, \ldots, h$	Working variables that are the w-bit words used in the computation of the hash values, $H^{(i)}$.
$H^{(i)}$	The i-th hash value. $H^{(0)}$ is the initial hash value; $H^{(n)}$ is the final hash value and is used to determine the message digest.
$H_j^{(i)}$	The j^{th} word of the i^{th} hash value, where $H_0^{(i)}$ is the left-most word of hash value i.
K_t	Constant value to be used for iteration t of the hash computation.
k	Number of zeroes appended to a message during the padding step.
l	Length of the message, M, in bits.
m	Number of bits in a message block, $M^{(i)}$.
M	Message to be hashed.
$M^{(i)}$	Message block i, with a size of m bits.
$M_j^{(i)}$	The j^{th} word of the i^{th} message block, where M i is the left-most word of message block i.
n	Number of bits to be rotated or shifted when a word is operated upon.
N	Number of blocks in the padded message.
T	Temporary w-bit word used in the hash computation.
w	Number of bits in a word.
W_t	The t^{th} w-bit word of the message schedule.

SHA-256 FUNCTIONS

SHA-256 uses six logical functions, where each function operates on 32-bit words, which are represented as x, y, and z. The result of each function is a new 32-bit word.

$$Ch(x, y, z) = (x \wedge y) \oplus (\sim x \wedge z),$$

$$Maj(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z),$$

$$\Sigma_0^{\{256\}}(x) = ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x),$$

$$\Sigma_1^{\{256\}}(x) = ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x),$$

$$\sigma_0^{\{256\}}(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x),$$

$$\sigma_1^{\{256\}}(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x).$$

SHA-256 CONSTANTS

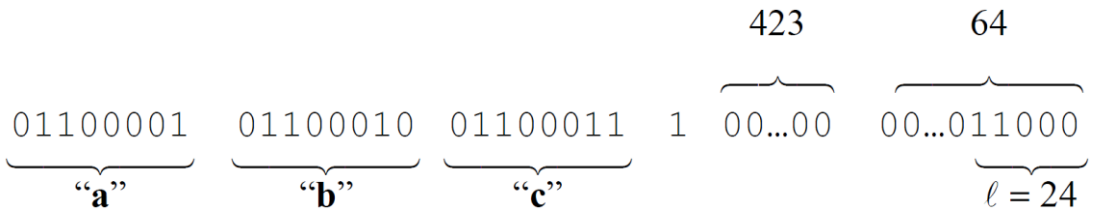
SHA-256 use the same sequence of sixty-four constant 32-bit words, $K_0^{\{256\}}$, $K_1^{\{256\}}$,…, $K_{63}^{\{256\}}$. These words represent the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers. In hex, these constant words are (from left to right):

428a2f98	71374491	b5c0fbcf	e9b5dba5	3956c25b	59f111f1	923f82a4	ab1c5ed5
d807aa98	12835b01	243185be	550c7dc3	72be5d74	80deb1fe	9bdc06a7	c19bf174
e49b69c1	efbe4786	0fc19dc6	240ca1cc	2de92c6f	4a7484aa	5cb0a9dc	76f988da
983e5152	a831c66d	b00327c8	bf597fc7	c6e00bf3	d5a79147	06ca6351	14292967
27b70a85	2e1b2138	4d2c6dfc	53380d13	650a7354	766a0abb	81c2c92e	92722c85
a2bfe8a1	a81a664b	c24b8b70	c76c51a3	d192e819	d6990624	f40e3585	106aa070
19a4c116	1e376c08	2748774c	34b0bcb5	391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
748f82ee	78a5636f	84c87814	8cc70208	90befffa	a4506ceb	bef9a3f7	c67178f2

PADDING THE MESSAGE

The message, M, shall be padded before hash computation begins. The purpose of this padding is to ensure that the padded message is a multiple of 512.

Suppose that the length of the message, M, is l bits. Append the bit "1" to the end of the message, followed by k zero bits, where k is the smallest, non-negative solution to the equation $l + 1 + k \equiv 448 \bmod 512$. Then append the 64-bit block that is equal to the number l expressed using a binary representation. For example, the (8-bit ASCII) message "abc" has length 8×3 = 24, so the message is padded with a one bit, then $448 - (24 + 1) = 423$ zero bits, and then the message length, to become the 512-bit padded message



The length of the padded message should now be a multiple of 512 bits.

PROCESSING PREPARATION

Parsing the padded message

After a message has been padded, it must be parsed into N m-bit blocks before the hash computation can begin.

For SHA-256, the padded message is parsed into N 512-bit blocks, $M^{(1)}$, $M^{(2)}$,…, $M^{(N)}$. Since the 512 bits of the input block may be expressed as sixteen 32-bit words, the first 32 bits of message block i are denoted $M_0^{(i)}$, the next 32 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$.

Setting the Initial Hash Value ($H^{(0)}$)

The initial hash value, $H^{(0)}$, shall consist of the following eight 32-bit words, in hex:

$H_0^{(0)}$ = 0x6a09e667, $H_1^{(0)}$ = 0xbb67ae85, $H_2^{(0)}$ = 0x3c6ef372, $H_3^{(0)}$ = 0xa54ff53a,
$H_4^{(0)}$ = 0x510e527f, $H_5^{(0)}$ = 0x9b05688c, $H_6^{(0)}$ = 0x1f83d9ab, $H_7^{(0)}$ = 0x5be0cd19.

These words were obtained by taking the first thirty-two bits of the fractional parts of the square roots of the first eight prime numbers.

ALGORITHM

SHA-256 may be used to hash a message, M, having a length of l bits, where $0 \leq l < 2^{64}$. The algorithm uses:

1) a message schedule of sixty-four 32-bit words,
2) eight working variables of 32 bits each, and
3) a hash value of eight 32-bit words. The final result of SHA-256 is a 256-bit message digest.

The words of the message schedule are labeled W_0, W_1,…, W_{63}. The eight working variables are labeled a, b, c, d, e, f, g, and h. The words of the hash value are labeled $H_0^{(i)}$, $H_1^{(i)}$,…, $H_7^{(i)}$ which will hold the initial hash value, $H^{(0)}$, replaced by each successive intermediate hash value (after each message block is processed), $H^{(i)}$, and ending with the final hash value, $H^{(N)}$. SHA-256 also uses two temporary words, T_1 and T_2.

SHA-256 preprocessing:

1) Pad the message, M.
2) Parse the padded message into N 512-bit message blocks, $M^{(1)}$, $M^{(2)}$, …, $M^{(N)}$.
3) Set the initial hash value, $H^{(0)}$.

ALGORITHM

After preprocessing is completed, each message block, $M^{(1)}$, $M^{(2)}$, …, $M^{(N)}$, is processed in order, using the following steps:

for $i = 1$ to N:

{

1. Prepare the message schedule, $\{W_t\}$:

$$W_t = \begin{cases} M_t^{(i)}, & 0 \le t \le 15, \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16}, & 16 \le t \le 63. \end{cases}$$

2. Initialize the eight working variables, a, b, c, d, e, f, g, and h, with the $(i-1)^{st}$ hash value:

$a = H_0^{(i-1)}$, $b = H_1^{(i-1)}$, $c = H_2^{(i-1)}$, $d = H_3^{(i-1)}$, $e = H_4^{(i-1)}$, $f = H_5^{(i-1)}$, $g = H_6^{(i-1)}$, $h = H_7^{(i-1)}$.

ALGORITHM

3. For $t = 0$ to 63:

{

$T_1 = h + \Sigma_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t;\ \ T_2 = \Sigma_0^{\{256\}}(a) + Maj(a,b,c)$

$h = g; g = f; f = e; e = d + T_1; d = c; c = b; b = a; a = T_1 + T_2.$

}

4. Compute the i^{th} intermediate hash value $H^{(i)}$:

$H_0^{(i)} = a + H_0^{(i-1)}; H_1^{(i)} = b + H_1^{(i-1)}; H_2^{(i)} = c + H_2^{(i-1)}; H_3^{(i)} = d + H_3^{(i-1)};$

$H_4^{(i)} = e + H_4^{(i-1)}; H_5^{(i)} = f + H_5^{(i-1)}; H_6^{(i)} = g + H_6^{(i-1)}; H_7^{(i)} = h + H_7^{(i-1)};$

}

After repeating steps one through four a total of N times (i.e., after processing M(N)), the resulting 256-bit message digest of the message, M, is

$$H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}H_4^{(N)}||H_5^{(N)}||H_6^{(N)}||H_7^{(N)}$$

THANKS FOR ATTENTION