

Authentication, Data Integrity and Cryptographic Hash Functions

Dr. Dhiren Patel

Authentication of...

- Of users (Human, Bot)
- Of entities (objects, persons, software module, website, machines..)
- Of things (IoT and IIoT)

Means of authentication

- in terms of “factors” of proof, such as:
- Something you *know* to prove your identity (e.g., a password)
- Something you *have* to prove your identity (e.g., a certificate)
- Something you *are* to prove your identity (e.g., a fingerprint)

Authentication of...

- Of cars? Of aircrafts? Of UAVs?
- Of messages
- Of documents
- Of computers, servers, websites
- Of programs
- Detection of malware
- Signature based? Anomaly based? Behavior?
- Markers in software?

Authentication

- An individual providing any one of these factors on its own as proof of their identity is said to be providing a “single-factor” authentication.
- Any two separate factors would be called “two-factor” authentication and so on.

One time password – via SMS/mail

- 6 to 8 digits
- Validity of about 60 seconds to 30 minutes!
- Used by Bank, Credit Cards, Online shopping

Two factor authentication

- E.g. India – two factor authentication
- using site authentication, card/account details and one time password (OTP)
- Use of two different channels (http and SMS) for Internet Banking
- VISA 3D Secure – India – use of pin/password or OTP through SMS or e-mail

2-factor authentication

- Two of the most common approaches to providing two-factor authentication for on-line access are:
- Time synchronous tokens (later slide)
- and Combining digital identities with USB tokens, smartcards, or biometric devices

Personal identification numbers (PINs)

- fixed (time-invariant) passwords
- most often used in conjunction with “something possessed”, typically a physical *token* such as a plastic banking card with a magnetic stripe, or a chipcard.

PIN

- entry of the correct PIN is required when the token is used. This provides a second level of security if the token is lost or stolen.
- For user convenience and historical reasons, PINs are typically short and numeric
- A common technique is for the PIN to serve to verify the user to the token, while the token contains additional independent information allowing the token to authenticate itself to the system

Single-Use-Password Tokens

- Some passcode generators omit the user keypad, and use as an implicit challenge - a time value (with a typical granularity of one minute) defined by a time clock loosely synchronized automatically between the system and the passcode generator.



Single-Use-Password Tokens

- The most common form of time varying pass code solution is a small keychain fob (Secure ID device) that displays a six-character password that changes every 60 seconds. Many organizations are using this type of solution to control remote employee access and as a stronger authentication solution for large corporate clients.



CAPTCHA

- *CAPTCHA =*
- *Completely Automated Public Turing test to tell Computers and Humans Apart*

CAPTCHA

- It is a type of challenge-response test used in computing to determine whether or not the user is human.
- A common type of CAPTCHA requires that the user types the letters of a distorted image, sometimes with the addition of an obscured sequence of letters or digits that appears on the screen.
- Because the test is administered by a computer, in contrast to the standard Turing test that is administered by a human, a CAPTCHA also called as **reverse Turing test**.



E.g. Spam bot

- automatic program by computer
- “Password-list attacks” which repeatedly attempt logins with illegally obtained IDs and passwords
- **Captcha prevents Spam bots**
- Easy for humans but hard for computers.
Adopting a mechanism like that made possible only humans can login.

Why Captcha?

- Authentication
- Search Engine
- Google/Yahoo/MSN
- AdSense
- EyeBall count
- Interactive Forms on Web
- Automatic v/s Human Authentication
- Account (loginID), IP based, Services based

CAPTCHA's idea - simple enough

- It presents users with an image showing an obfuscated string of letters
- that they must type in to get an e-mail or social networking account, for instance, or to enter a comment on an online forum
- The theory is that only humans can decipher the letters hidden in the image and type in the correct code, and for a time it was an effective tool to keep the bots out
- A basic CAPTCHA



Captcha evolution

- Because the test is administered by a computer, in contrast to the standard Turing test that is administered by a human, a CAPTCHA is sometimes described as a reverse Turing test.
- CAPTCHA was created in 2000 by researchers at Carnegie Mellon University, and by 2007, the technology was being used almost everywhere on the Web
- How does reCAPTCHA know that the human got a word right? By using a control word, where the system already knows the correct spelling, along with the unknown word.
- Unfortunately, beginning in early 2008, crackers started getting the better of the CAPTCHA systems.

Completely Automated Public Turing test to tell Computers and Humans Apart

- Text-based CAPTCHA
- Image-based CAPTCHA
- Audio-based CAPTCHA
- Video-based CAPTCHA
- Puzzle-based CAPTCHA
- regional language based CAPTCHA

- Protecting Registration forms in website
- Protecting Email Accounts
- Protecting Online Shopping
- Protecting worms and spam comments
- Protecting online polls
- Phishing Attacks
- Protecting (from) Search Engine Bots

Breaking Captcha

- These programs work by using OCR (optical character recognition) software to try to make sense of CAPTCHA's disguised text.
- If they fail, they try again.
- They take advantage of the fact that some CAPTCHA systems don't automatically give users a new CAPTCHA image to puzzle out.
- Instead, they'll let you, or a cracker program, keep working at the hidden text until it's solved.

Breaking Captcha

- Another way to crack a badly designed CAPTCHA program is to reuse the session identification URL of a solved CAPTCHA image.
- In this case, either the cracker, or more likely a cracking program, first gets the right answer to a CAPTCHA. It then reconnects to the Web site with a URL containing the solved session identification information with a new username.
- 100% success rate until the session ID eventually expires.

can CAPTCHA be saved?

- the [Google Books Project](#) and the [Internet Archive](#), two projects that are converting paper books to digital format using OCR software. As explained above, OCR software often doesn't read words accurately.
- When the projects' OCR programs flag a word as unreadable, it's saved as an image and used on the Web as a CAPTCHA test.
- This has positive results - These CAPTCHAs are already known to be resistant to OCR attacks, making Web sites that use reCAPTCHA less vulnerable to CAPTCHA crackers.

Image based Captcha

- CMU's ESP-PIX requires users to pick a word that describes all four objects in an image.
- SQ-PIX requires users to first pick out the right image from three and then trace the outline of the object within the image.
- For example, you might see an image of a cat, one of a flower and one of a balloon, with the instruction "Trace all balloons."

E.g. The SQ-PIX image-based CAPTCHA



Imagination CAPTCHA

- Penn State developers came up with Imagination CAPTCHA.
- In this system, a user must first pick out the geometric center of a distorted image from a page that's filled with similar overlapping pictures.
- If you get that right, you're presented with another carefully distorted image and asked to pick a word to describe what you're seeing.
- The Imagination system is based on ALIPR (Automatic Linguistic Indexing of Pictures), an automated image-tagging and searching technology.
- The core idea, as the developers explain, is that image recognition is a harder problem for computers to solve than text recognition, making the Imagination system more secure than text-based CAPTCHAs.

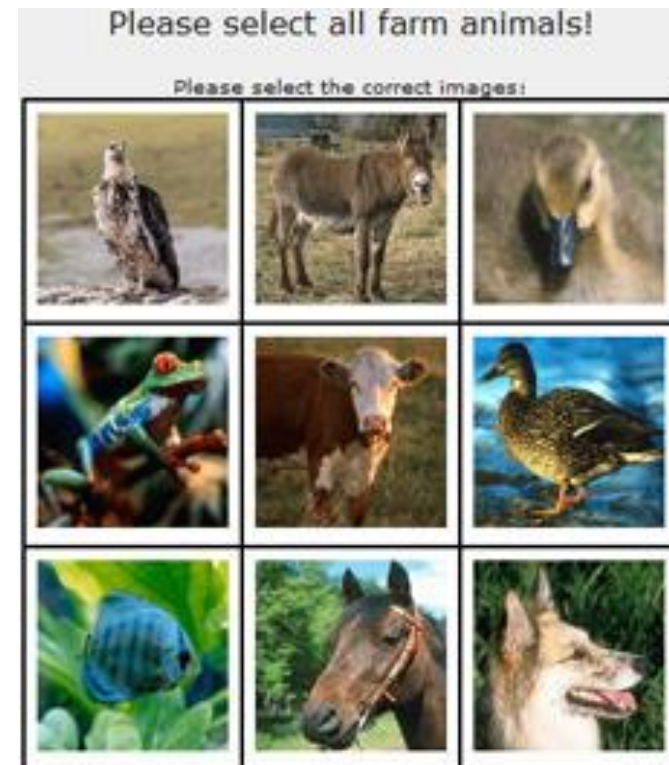
Imagination Captcha

- Unfortunately, color-blind users are likely to face problems with the Imagination system. (Blind and hard-of-sight people, of course, will have problems with all image-based CAPTCHAs.)



Image based captcha - KittenAuth

- With KittenAuth, users are presented with a grid of 12 pictures of animals and then asked to pick out, for example, the ones containing -- you guessed it -- kittens.
- Microsoft Research has taken the same idea for its ASIRRA (Animal Species Image Recognition for Restricting Access) technology. ASIRRA uses a larger pool of images from PetFinder.com, but otherwise this Web service CAPTCHA is essentially a KittenAuth clone.



More CAPTCHA

N D R

33FFAC

9Ch2UP

ESPEU8

2900

1084

irctc

Login

User ID :

Password

Captcha

Captcha letters are case sensitive and to be entered in Upper Case only

HPRJX 

☐ Request OTP

[Forgot Password](#)

[Sign up](#)

[NCT Assistant Login](#)



[Login](#)

Captcha and other details

Type the characters in the box below

F M H B

Type Here:

  ? [NLP](#)Captcha

Captcha and other details



THIS FESTIVE SEASON

WIN A NIKON CAMERA OF YOUR CHOICE



PARTICIPATE NOW 

Type "Nikon" in the box below

Type Here:

  ? [NLP](#)Captcha

[View All](#)

Login

User ID :

Password

Captcha

KDF4Q 

Captcha letters are case sensitive

☐ Request OTP

[Forgot Password](#)

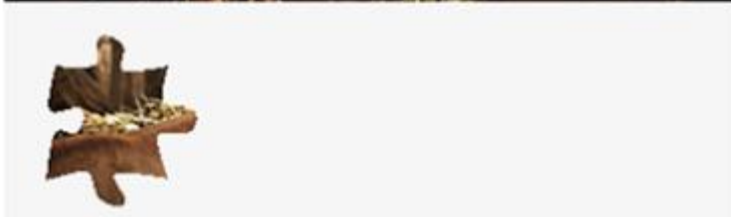
[Sign up](#)

[NCT Assistant Login](#)

[Login](#)

Puzzle captcha

- Puzzle captcha = $(10 + 10) = ?$



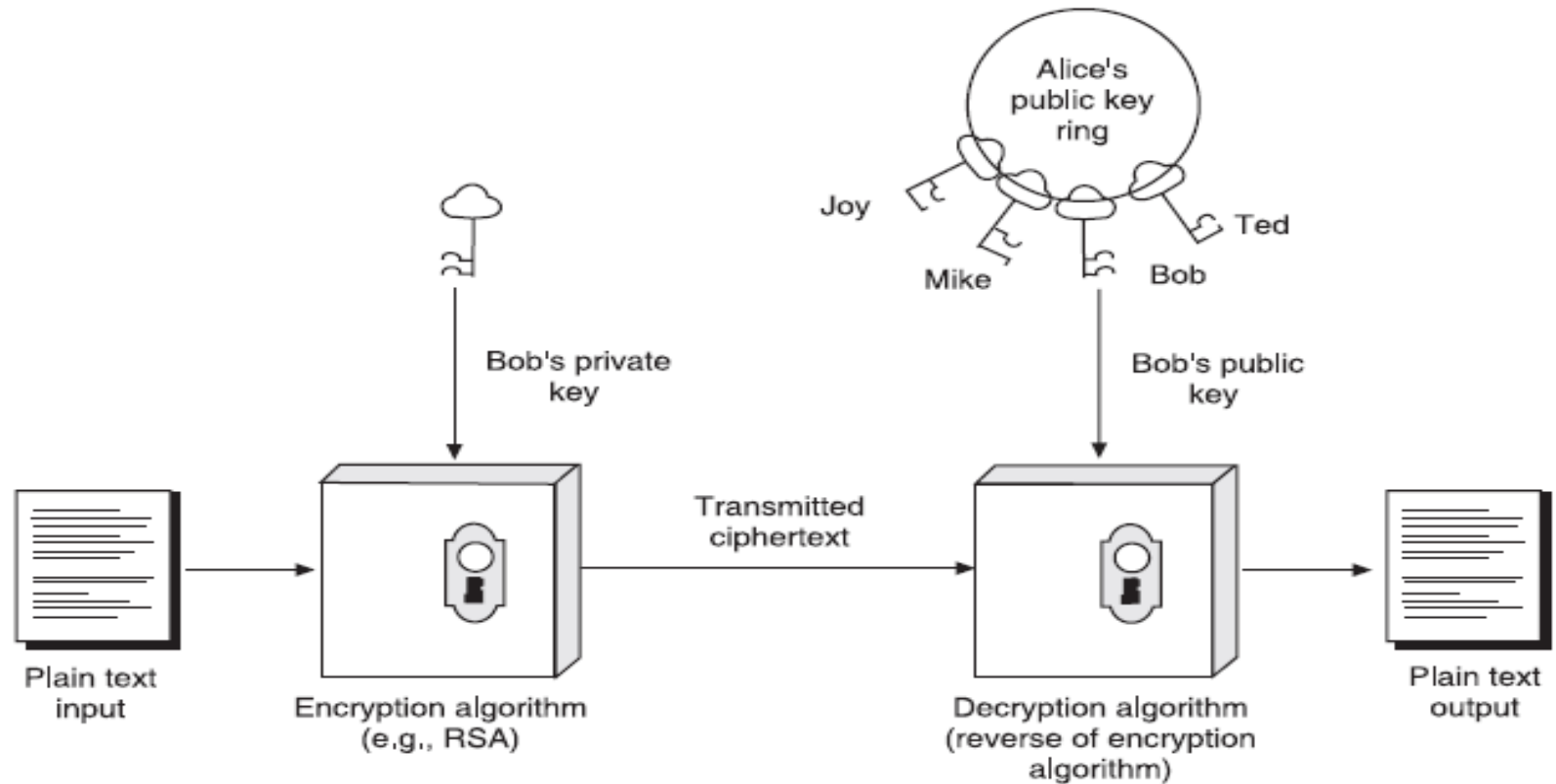
Defense - Intelligence

- way of stopping CAPTCHA bot attackers: incorporate a hidden field with CSS (Cascading Style Sheets).
- The field is coded so that human users never see it. Bots, however, read the page's code and note that there is a field to be filled in, and proceed to do so. That, of course, is enough to mark the visitor as a potential cracking program rather than an actual user.
- The bots should fill it in, and if you compare the inputted value to the value you start with, you can quit execution right there

Captcha Conclusion

- computers will be bright enough to solve any form of CAPTCHA
- any CAPTCHA system, or any other security measure, can really do is slow down would-be crackers
- Web security must be concerned not only with keeping out attackers, but with minimizing the damage they can cause when they have broken into a site

Authentication (using PKC)



Authentication using Biometric techniques

- Biometric techniques are based on the features that cannot be lost or forgotten. It benefits users as well as system administrators because it avoids the problems and cost associated with lost, reissued, or temporary tokens, cards, and passwords
- It is almost impossible to lose or forget biometrics, since they are an intrinsic part of each person, and this is an advantage which they hold over keys, passwords or codes.

Biometric Technology (will be discussed later)

- Biometric technologies may be used in three ways:
- (a) to verify that people are who they claim to be,
- (b) to discover the identity of unknown people, and
- (c) to screen people against a watch-list.
- Biometric identification works in four stages: *enrolment*, *storage*, *acquisition* and *matching*.
- It cannot rely on secrecy, since most biometric features are either self-evident or easily obtainable.

Cryptographic Hash Functions

- **HF Applications**
- very popular tools for cryptographic applications such as
- message integrity check,
- message authentication,
- digital signature,
- protection scheme for pass-phrases or passwords
- electronic funds transfer, data storage, software distribution, and other applications where data integrity is very important

Commitment Protocol

- Goal: A and B wish to play “odd or even” over the network
- Naive Commitment Protocol
- A picks a number X and sends it to B
- B picks a number Y and sends it to A
- A wins if $X+Y$ is odd
- B wins if $X+Y$ is even
- Problem: How can we guarantee that B doesn't cheat?

Commitment Protocols with Hash

- A picks a number X and sends value of $Z = H(X)$ to B
- B picks a number Y and sends value of Y to A
- A now sends value of X to B
- B checks if X complies with Z that was sent before
- A wins if $X+Y$ is odd
- B wins if $X+Y$ is even
- Solution: In this protocol B cannot cheat
- Hash function does two things in the protocol:
- Hides the number X from B at the beginning of the game
- Makes A commit to the number X until the end of the game
- Question: What if A always picks small numbers so that B can make a list of all the hash values?
- Answer: A should select random values for the protocol:
- Select the number X from a very large space of numbers
- Mask the number X with a random noise from a very large space

Block Ciphers v/s Hash Functions

- ❑ A block cipher (is a function which maps) n -bit plaintext blocks to n -bit ciphertext blocks; n is called the *block length*.

- $E: \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$

- ❑ To allow unique decryption, the encryption function must be one-to-one (i.e., invertible)

- ❑ Hash Function - takes input (of variable-length) and returns a fixed size output string h (usually much smaller than input)

$$H: \{0,1\}^* \rightarrow \{0,1\}^n, \quad h = H(M)$$

- ❑ One way

Demo – HF (MD5, SHA-1)

How Hash functions are useful in Data integrity applications?

Variable length
original data

Fixed length
“digest” of data



- Plain text, Hash (MITM - Attack)
- Plain text, Shared Secret, Hash (MAC)
- HF and Keyed HF?
- Digital Signature (later)

Building Cryptographic Hash Function

- *Operability:*
- $H()$ should work on any input length
- $H()$ should produce output of fixed size
- $H()$ should be easy to compute

Additional Properties

- Compression → collisions
- Sparse over large input space
- More bits – output lookup table too large
- Weak collision resistance
- Strong collision resistance

- 00
- 11
- 10
- 01

Checksum (CRC) as a HF --- too weak?

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	:	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

- Anil has mo g/p e/a t/y.

Building a hash function - example

- Simple Hash Function - based on XOR of message blocks

Block number	Original Blocks	Modified Blocks
1	0 1 0 1 1 0	1 0 0 1 1 0
2	1 0 1 1 0 1	0 1 1 1 1 1
3	1 1 1 1 1 1	1 1 1 0 1 1
4	1 1 1 0 0 0	1 1 1 0 0 0
Hash code obtained by XOR	1 1 1 1 0 0	1 1 1 0 1 0

not secure as manipulation is easy for any message

Block number	Original Blocks	Modified Blocks
1	0 1 0 1 1 0	1 0 0 1 1 0
2	1 0 1 1 0 1	0 1 1 1 1 1
3	1 1 1 1 1 1	1 1 1 0 1 1
4	1 1 1 0 0 0	1 1 1 0 0 0
Hash code obtained by XOR	1 1 1 1 0 0	1 1 1 0 1 0

a. block 4 is 1 1 1 0 0 0

b. Correction required 0 0 0 1 1 0

Manipulation-hiding 1 1 1 1 1 0

block 4 = (a) \oplus (b)

Resulting hash code \rightarrow

The Manipulation-hiding message is

1 0 0 1 1 0

0 1 1 1 1 1

1 1 1 0 1 1

1 1 1 1 1 0

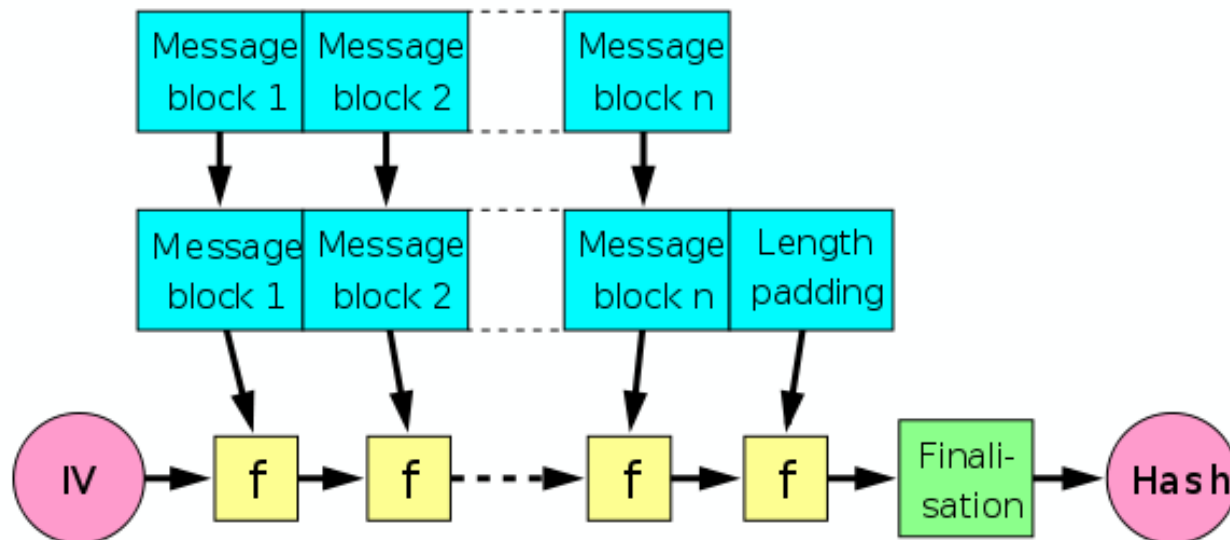
1 1 1 1 0 0

Hash Function construction

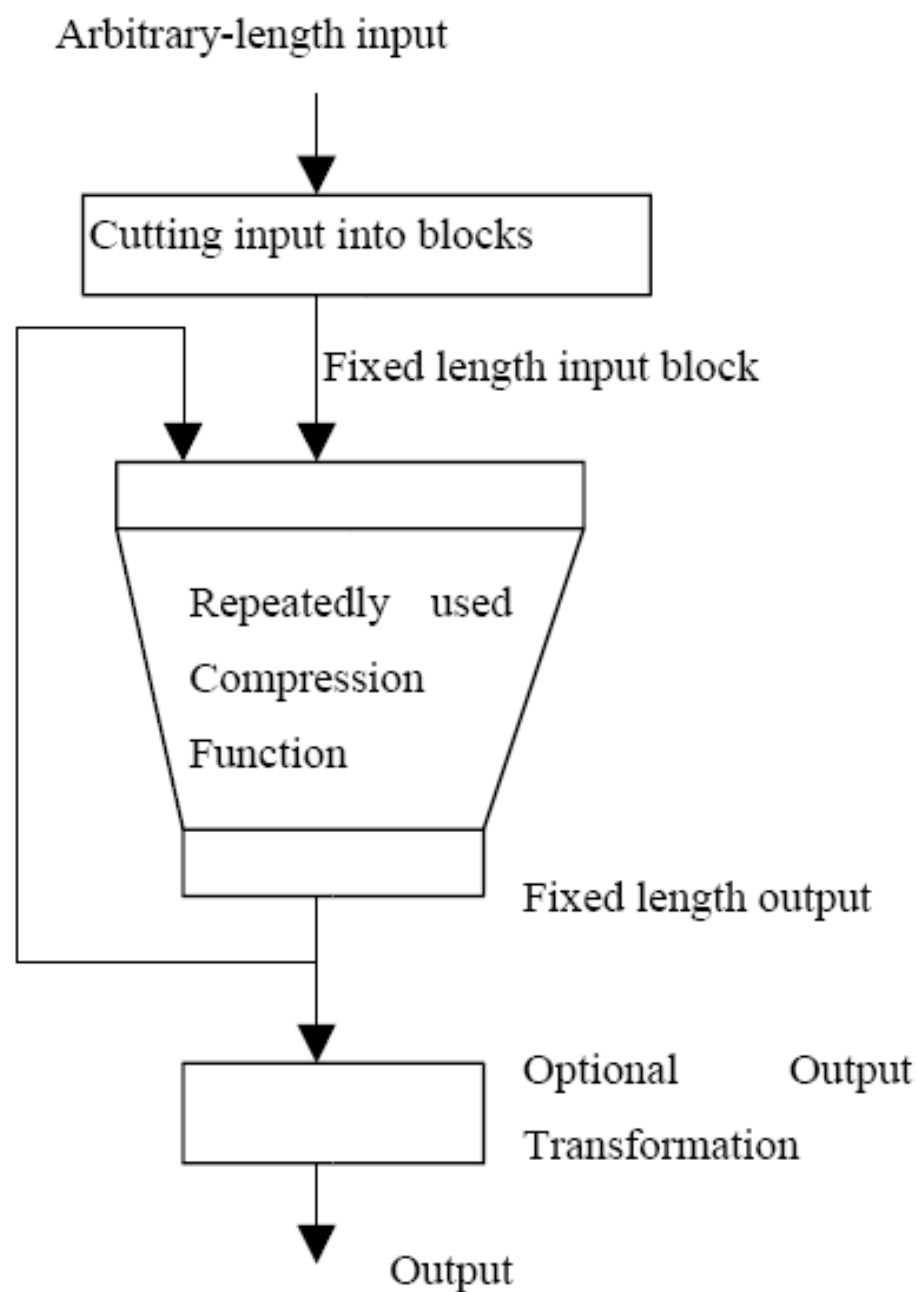
Merkle-Damgard:

iterative application of compression function

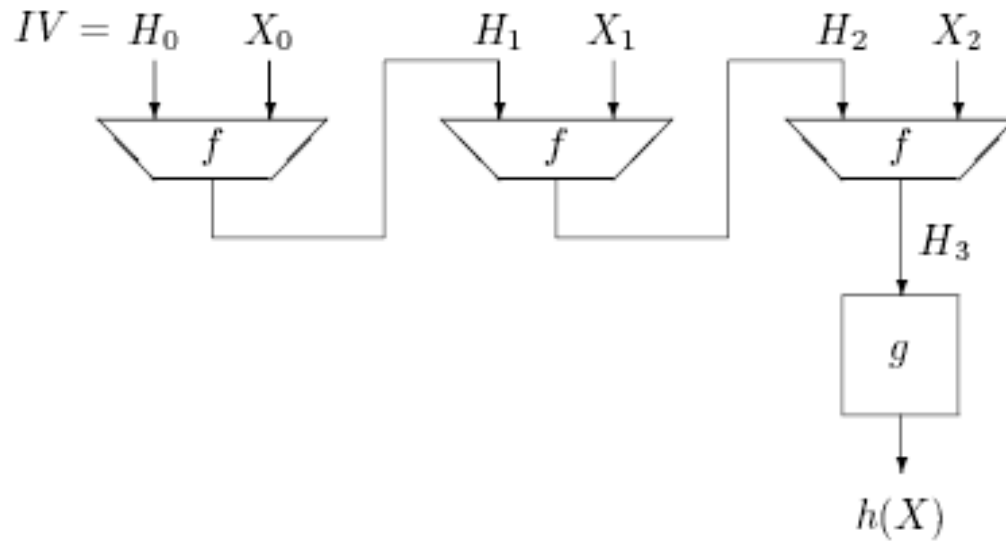
- MD-strengthening → The procedure of fixing the *IV* and adding a representation of the length of input.



- Padding
- 1111111111
- 1000000000

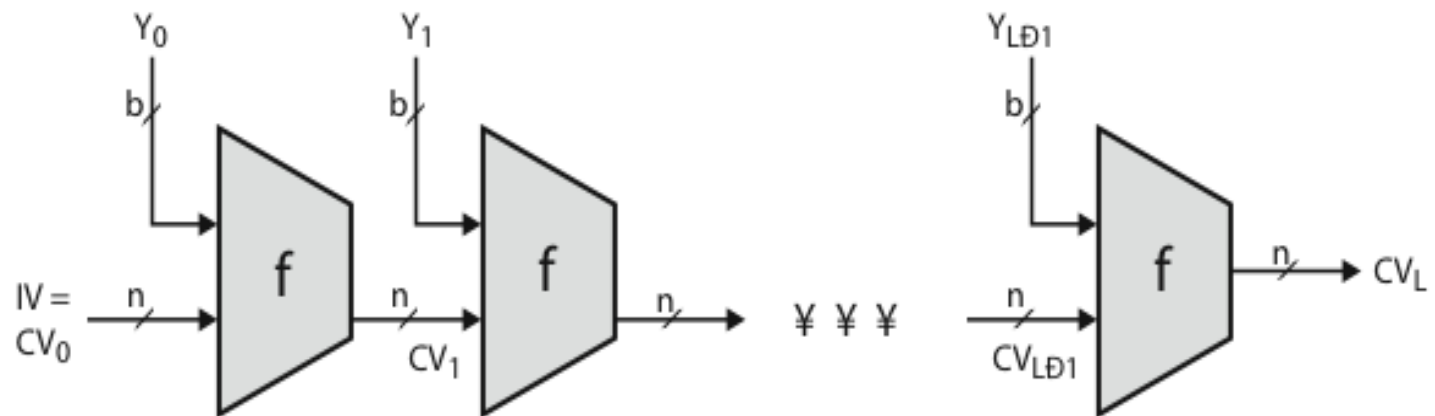


HF



$$\begin{aligned} H_0 &= IV , \\ H_{i+1} &= f(H_i, X_i) \quad \text{for } 0 \leq i < t , \\ h(X) &= g(H_t) . \end{aligned}$$

HF

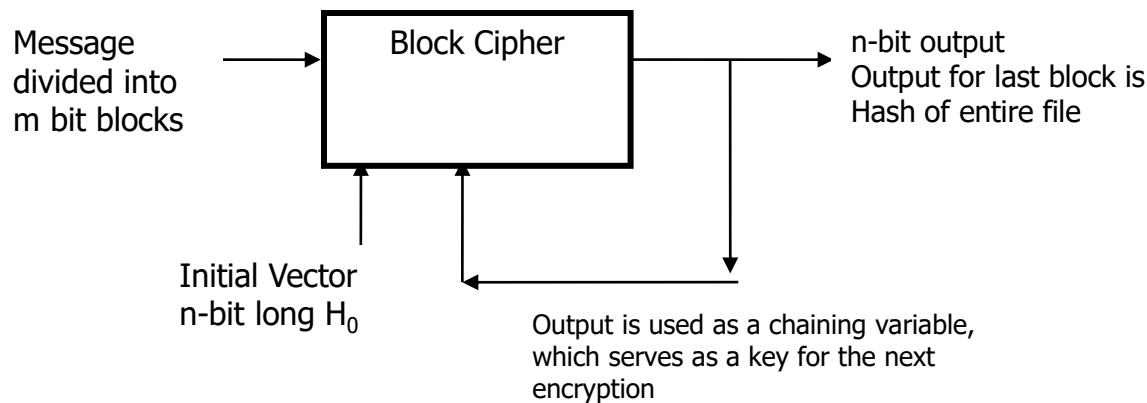


IV = Initial value
 CV_i = chaining variable
 Y_i = i th input block
 f = compression algorithm

L = number of input blocks
 n = length of hash code
 b = length of input block

HF construction — *using Block Cipher*

- Block cipher (standard or dedicated) in CBC
- Hash function $H: \{0,1\}^* \rightarrow \{0,1\}^n$
- Block cipher encryption $E: \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$
- $H_i = H_{i-1} \oplus M_i$

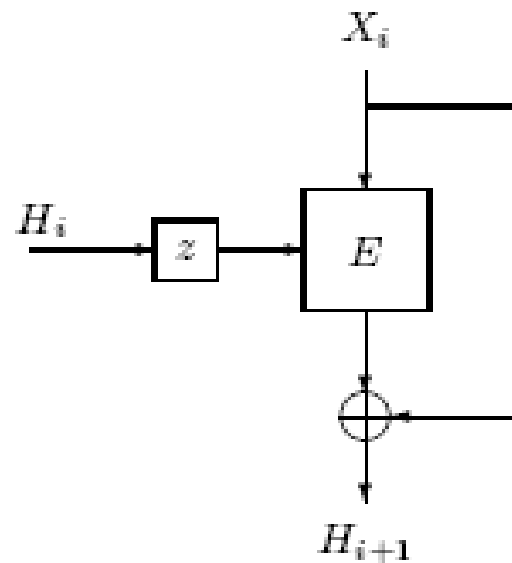


$$H_i = E_{H_{i-1}}(B_i),$$

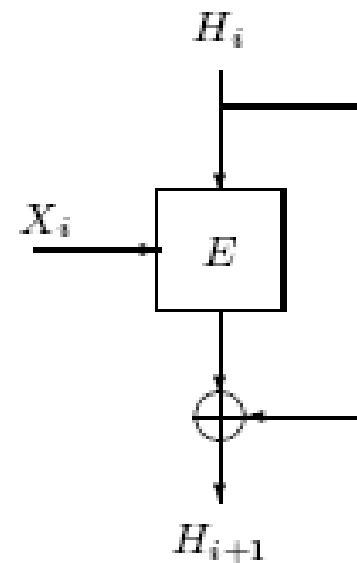
$$H(M) = E_{H_{(n-1)}}(B_n)$$

Hash Function Using A Block Cipher

Using Block cipher



(a) Matyas-Meyer-Oseas



(b) Davies-Meyer

HF construction

- Dedicated function with optimized performance
- Operates on iterative compression function
- based on 32-bit Registers/buffers, S-Boxes
- with multiple rounds of computations

Essential parameters

- Message pre-processing
- Chaining variable and hash output
- Collision-resistance of the compression function
- Word-orientation - Little-endian or big-endian conversion
- Sequential structure
- Message expansion

Hash Functions Security

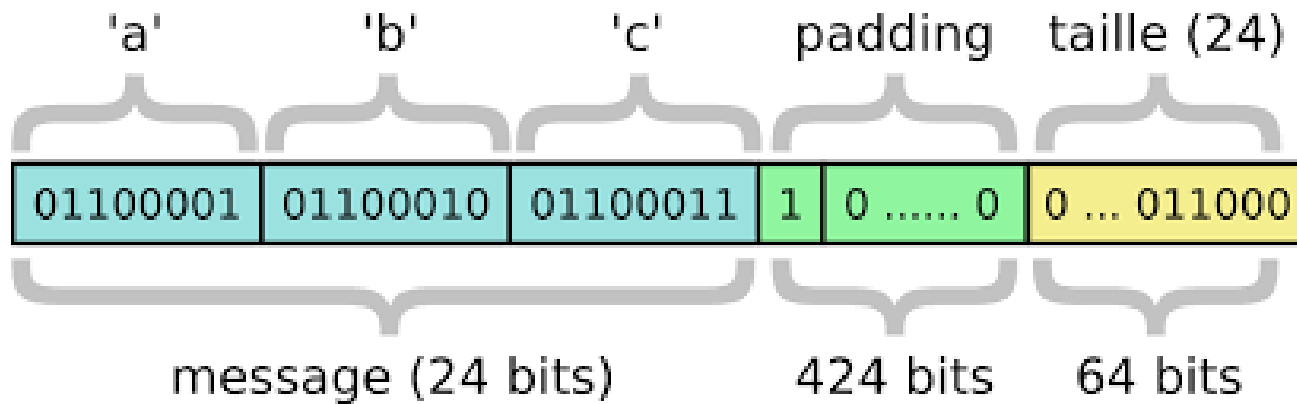
- **cryptanalytic attacks** exploit structure
- analytic attacks on iterated hash functions
 - typically focus on collisions in compression function f
 - like block ciphers, HF is often composed of rounds
 - attacks exploit properties of round functions

Some Cryptographic hash functions

- MD5 (digest of 128-bit) -- Rivest
- SHA-1 (160-bit), NIST FIPS 180-1
- National Institute for Standard & Technology
FIPS 180-2 → SHA-256, SHA-384, SHA-512
- SHA3 – new NIST selection (slide will come later)
- RIPE-MD (160-bit) – (Dobbertin, Preneel)
- MASH (based on modular arithmetic) – (Girault)
- Whirlpool (up to 512 bit, based on a dedicated block-cipher), project NESSIE (Barreto, Rijmaen)
- NESSIE - New European Schemes for Signatures, Integrity and Encryption
- HMAC (with embeded hash functions (MD5, SHA-1) and a secret key K), FIPS

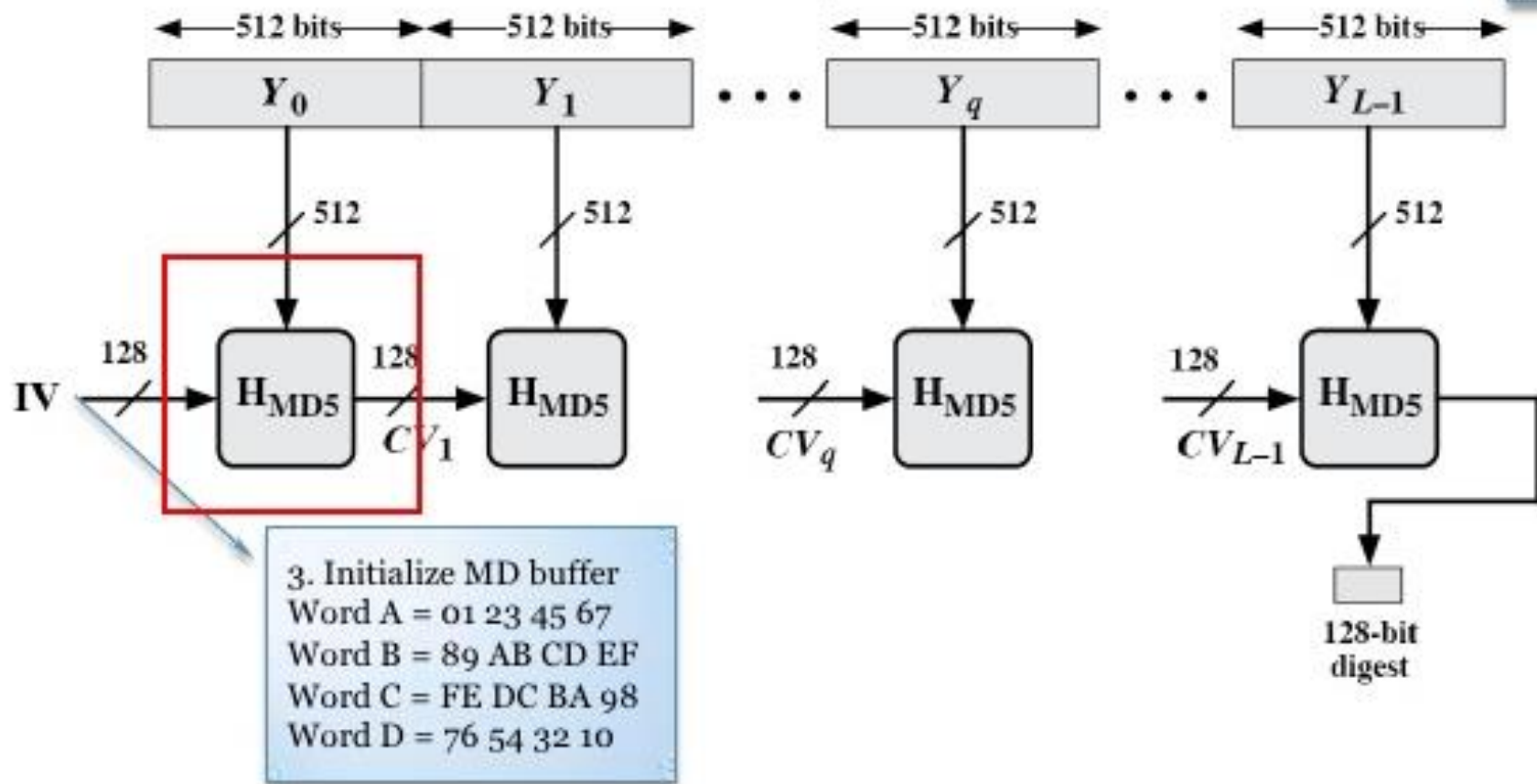
MD-5 Hash function [Rivest 1992 – RFC 1321]

- MD5 Produces hash of length 128 bits
- **Padding**
 - The message is "padded" (extended) so that its length (in bits) is being a multiple of 512 bits long.
 - Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512.
 - Last 64 bits are added as binary representation of msg length.



MD-5

- Let message has a length (after padding) that is an exact multiple of 16 (32-bit) words.
- Let $M[0 \dots N-1]$ denote the words of the resulting message, where N is a multiple of 16.
- A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register.
- These registers are initialized to the following values in hexadecimal, low-order bytes first):
 - word A: 01 23 45 67
 - word B: 89 ab cd ef
 - word C: fe dc ba 98
 - word D: 76 54 32 10



MD5 - Message Digest Computation

- **Four auxiliary functions** - Four auxiliary functions are used, each take as input three 32-bit words and produce as output one 32-bit word. //F,G,H,I

$$F(x,y,z) = (x \wedge y) \vee (\sim x \wedge z)$$

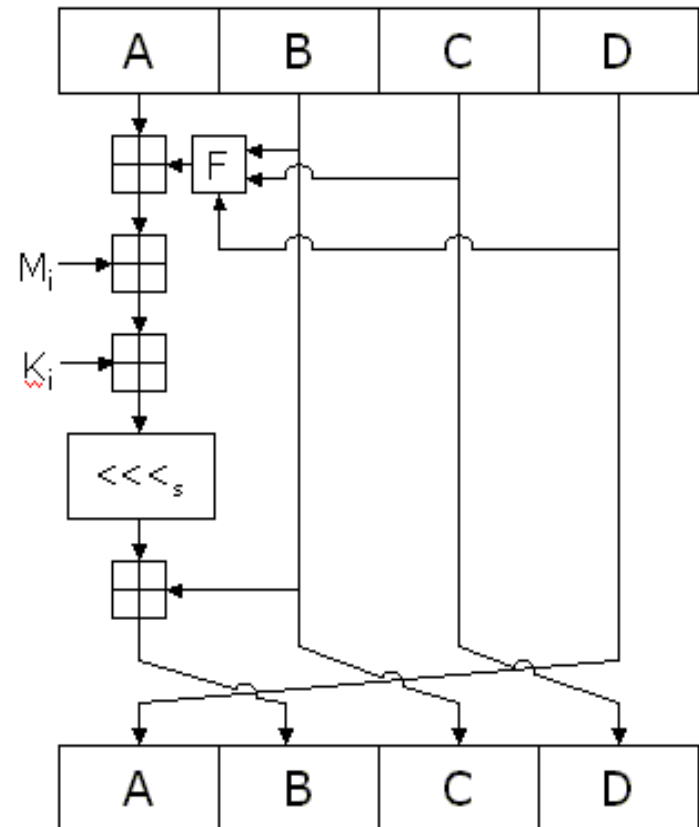
$$G(x,y,z) = (x \wedge z) \vee (y \wedge \sim z)$$

$$H(x,y,z) = x \oplus y \oplus z$$

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

MD5 – 1 operation

- F – non linear function (F,G,H,I)
- Total such 64 operations
- Grouped in 4 rounds (of 16 each)



MD-5

- Let $T[i]$ denote the i -th element of the table, computed using sine function.
- Do the following: `/* Process each 16-word block. */`
 for $i = 0$ to $N/16 - 1$ do `/* Copy block i into X . */`
 for $j = 0$ to 15 do
 Set $X[j]$ to $M[i \cdot 16 + j]$.
 end `/* of loop on j */`
- `/* Save A as AA , B as BB , C as CC , and D as DD . */`
 $AA = A$ $BB = B$ $CC = C$ $DD = D$
`/*prog cont.*/`

64 values – from sine function

- **for**
- **i from 0 to 63**
- **k[i] := floor(abs(sin(i + 1)) × (2 pow 32))**
- **end for**

64 values – from sine function

```
k[ 0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee }
k[ 4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
k[ 8..11] := { 0x698098d8, 0x8b44f7af, 0xfffff5bb1, 0x895cd7be }
k[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }
k[16..19] := { 0xf61e2562, 0xc040b340, 0x265e5a51, 0xe9b6c7aa }
k[20..23] := { 0xd62f105d, 0x02441453, 0xd8a1e681, 0xe7d3fbc8 }
k[24..27] := { 0x21e1cde6, 0xc33707d6, 0xf4d50d87, 0x455a14ed }
k[28..31] := { 0xa9e3e905, 0xfcefa3f8, 0x676f02d9, 0x8d2a4c8a }
k[32..35] := { 0xffffa3942, 0x8771f681, 0x6d9d6122, 0xfde5380c }
k[36..39] := { 0xa4beea44, 0x4bdecfa9, 0xf6bb4b60, 0xbebfbcb70 }
k[40..43] := { 0x289b7ec6, 0xeaa127fa, 0xd4ef3085, 0x04881d05 }
k[44..47] := { 0xd9d4d039, 0xe6db99e5, 0x1fa27cf8, 0xc4ac5665 }
k[48..51] := { 0xf4292244, 0x432aff97, 0xab9423a7, 0xfc93a039 }
k[52..55] := { 0x655b59c3, 0x8f0ccc92, 0xffefff47d, 0x85845dd1 }
k[56..59] := { 0x6fa87e4f, 0xfe2ce6e0, 0xa3014314, 0x4e0811a1 }
k[60..63] := { 0xf7537e82, 0xbd3af235, 0x2ad7d2bb, 0xeb86d391 }
```

MD-5 – round 1 – F function

- `/* Round 1. */` `/* Let [abcd k s i] denote the operation`
`a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */`
- `/* Do the following 16 operations. */`
[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3]
[BCDA 3 22 4] [ABCD 4 7 5] [DABC 5 12 6]
[CDAB 6 17 7] [BCDA 7 22 8] [ABCD 8 7 9]
[DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12] [ABCD 12 7 13]
[DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

A B C D are updated.....

Round 2 – G function

- `/* Round 2. */`
- `/* Let [abcd k s i] denote the operation`
- `a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */`
- `/* Do the following 16 operations. */`
[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]
[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20
24] [ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8
20 28] [ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA
12 20 32]

Round 3 – H function

- `/* Round 3. */`
- `/* Let [abcd k s t] denote the operation
 $a = b + ((a + H(b,c,d) + X[k] + T[i]) \lll s).$ */`
- `/* Do the following 16 operations. */`
[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23
36] [ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10
23 40] [ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43]
[BCDA 6 23 44] [ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16
47] [BCDA 2 23 48]

Round 4 – I function

- `/* Round 4. */`
- `/* Let [abcd k s t] denote the operation
a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */`
- `/* Do the following 16 operations. */`
[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21
52] [ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA
1 21 56] [ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59]
[BCDA 13 21 60] [ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15
63] [BCDA 9 21 64]

MD-5 - result

- `/* Then perform the additions – update with the original vlaues*/`
- `A = A + AA B = B + BB C = C + CC D = D + DD`
- `/* end of loop on i */`
- `Output = ABCD /*concatenated – 128 bit*/`

Secure Hash Algorithm (SHA)

- SHA originally designed by NIST (National Institute of standards and technology) and published as a Federal Information Processing Standard (FIPS 180) in 1993.
- revised in 1995 as FIPS 180-1 and referred to as SHA-1, also Internet RFC3174
- The algorithm is SHA, the standard is SHS – secure hash standard

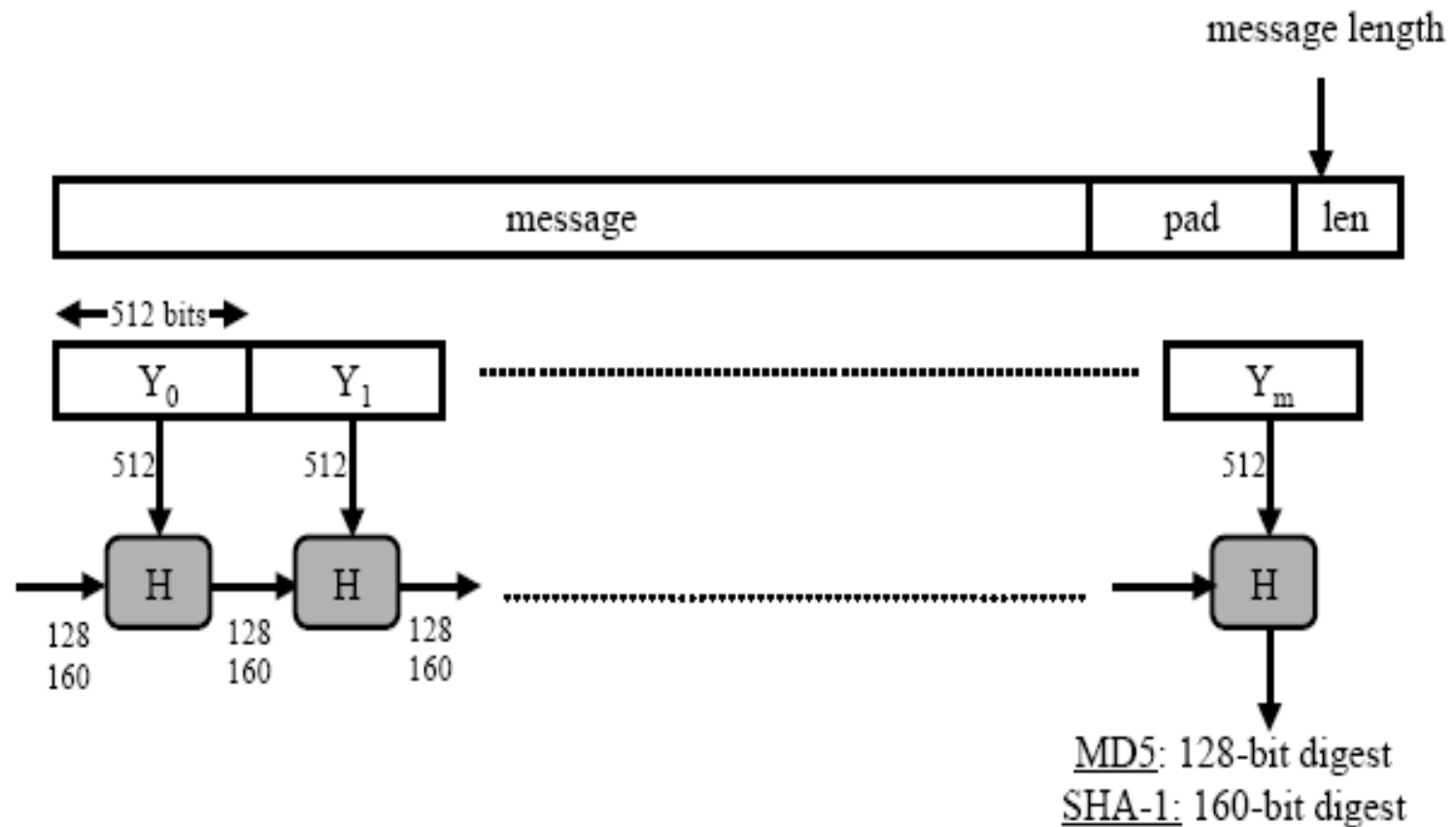
SHA-1 (Secure Hash Algorithm)

- an iterated hash function with a 160-bit message digest. (5 registers)
- Padding same as MD-5
- SHA-1 is built from word-oriented operations on bitstrings, where a word consists of 32 bits (or eight hexadecimal characters (nibble)).
- The operations used in SHA-1 are as follows:
 - $X \wedge Y$ bitwise “and” of X and Y
 - $X \vee Y$ bitwise “or” of X and Y
 - $X \text{ xor } Y$ bitwise “xor” of X and Y
 - $\neg X$ bitwise complement of X
 - $X+Y$ integer addition modulo 2^{32}
 - $\text{ROTLs}(X)$ circular left shift of X by s position ($0 \leq s \leq 31$)

Padding and Blocks

- Same as MD5
- $y = M1 \parallel M2 \parallel \dots \parallel M_n$.

MD-5 and SHA-1



SHA-1

- Four constants are used :
 - $K_t = 0x5a827999$, for $t = 0$ to 19
 - $K_t = 0x6ed9eba1$, for $t = 20$ to 39
 - $K_t = 0x8f1bbcdc$, for $t = 40$ to 59
 - $K_t = 0xca62c1d6$, for $t = 60$ to 79
- Message block is transferred from 16 blocks to 80 blocks:
 - $W_t = M_t$, for $t=0$ to 15
 - $W_t = (W_{t-3} + W_{t-8} + W_{t-14} + W_{t-16}) \lll 1$, for $t=16$ to 79

SHA-1 functions

- Define the function f_0, \dots, f_{79} as follows :

$f_t(B, C, D) =$

- $(B \wedge C) \vee ((\neg B) \wedge D)$ if $0 \leq t \leq 19$
- $B \text{ xor } C \text{ xor } D$ if $20 \leq t \leq 39$
- $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ if $40 \leq t \leq 59$
- $B \text{ xor } C \text{ xor } D$ if $60 \leq t \leq 79$.
- Each function f_t takes three words B , C and D as input, and produces one word as output.

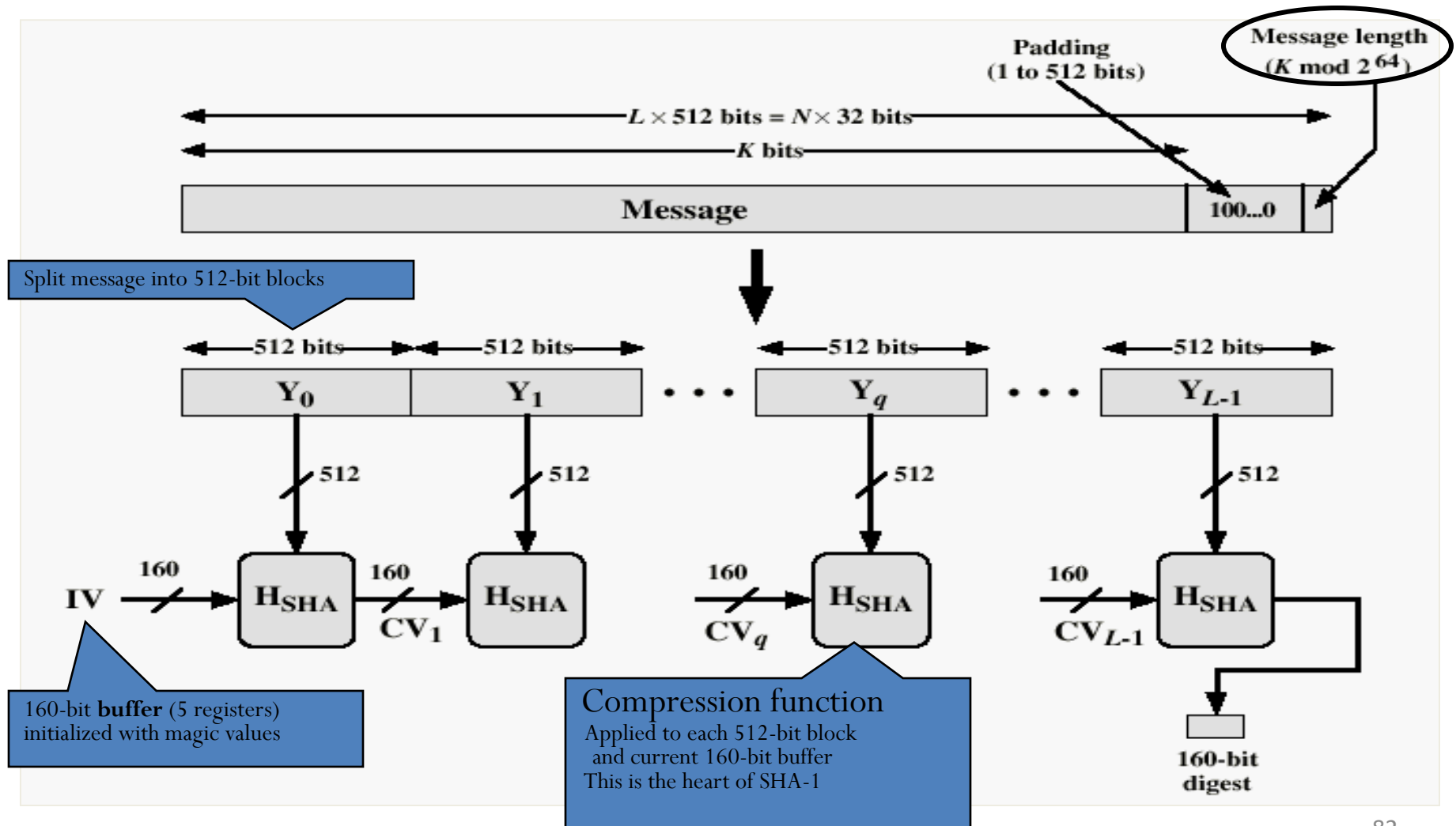
SHA-1

- $H_0 \leftarrow 67452301$
- $H_1 \leftarrow \text{EFCDAB89}$
- $H_2 \leftarrow 98\text{BADCFE}$
- $H_3 \leftarrow 10325476$
- $H_4 \leftarrow \text{C3D2E1F0}$
- Define the word constants K_0, \dots, K_{79} , which are used in the computation of $\text{SHA-1}(x)$, as follows:
 - $K_t =$
 - 5A827999 if $0 \leq t \leq 19$
 - 6ED9EBA1 if $20 \leq t \leq 39$
 - 8F1BBCDC if $40 \leq t \leq 59$
 - CA62C1D6 if $60 \leq t \leq 79$

SHA-1

- for $i \leftarrow 1$ to n
- do
- denote $M_i = W_0 || W_1 || \dots || W_{15}$, where each W_i is a word
- for $t \leftarrow 16$ to 79
- do $W_t \leftarrow \text{ROTL1}(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$
- $A \leftarrow H_0$
- $B \leftarrow H_1$
- $C \leftarrow H_2$
- $D \leftarrow H_3$
- $E \leftarrow H_4$

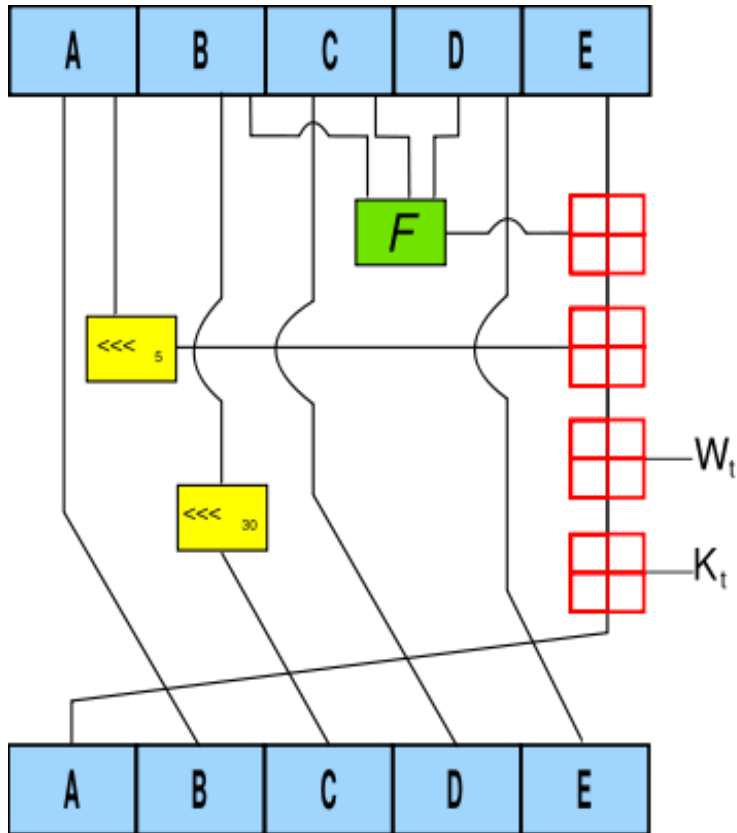
Description of SHA-1



SHA-1

- For $t \leftarrow 0$ to 79
- do
- $\text{Temp} \leftarrow \text{ROTL5}(A) + f_t(B, C, D) + E + W_t + K_t$
- $E \leftarrow D$
- $D \leftarrow C$
- $C \leftarrow \text{ROTL30}(B)$
- $B \leftarrow A$
- $A \leftarrow \text{temp}$
- $H_0 \leftarrow H_0 + A$
- $H_1 \leftarrow H_1 + B$
- $H_2 \leftarrow H_2 + C$
- $H_3 \leftarrow H_3 + D$
- $H_4 \leftarrow H_4 + E$
- return ($H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4$)

One SHA-1 operation



If t is the operation number (from 0 to 79), W_t represents the t th sub-block of the expanded message, and $\lll s$ represents a left circular shift of s bits, then the main loop looks like:

For $t=0$ to 79

$$\text{TEMP} = (a \lll 5) + f_t(b, c, d) + e + W_t + K_t$$

$$e = d$$

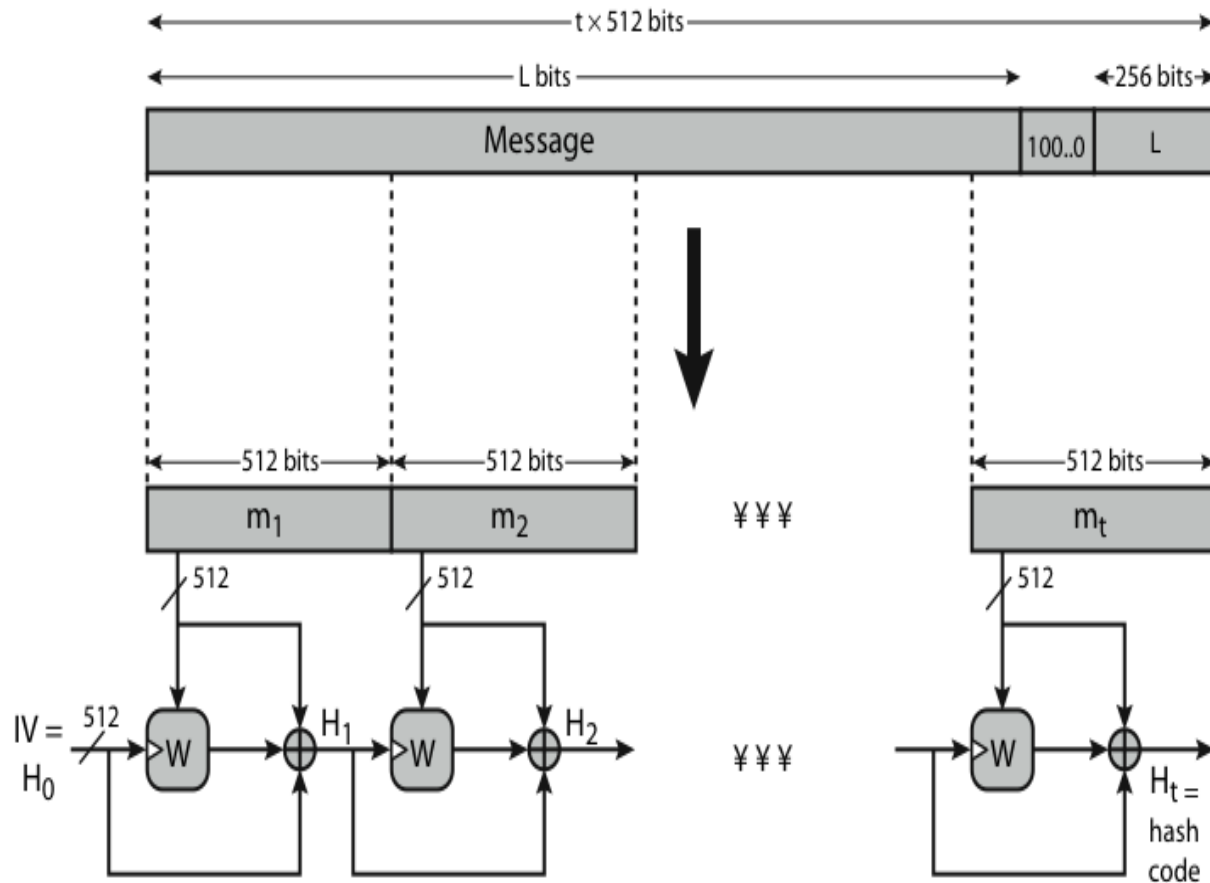
$$d = c$$

$$c = b \lll 30$$

$$b = a$$

$$a = \text{TEMP}$$

Whirlpool – 512 bit HF – using AES internals - endorsed by European NESSIE project – updating 512 bit buffer



Note: triangular hatch marks key input

SHA-3

- NIST has initiated an effort to develop one or more additional hash algorithms through a public competition – 2007
- to develop a new cryptographic hash algorithm, which converts a variable length message into a short "message digest" that can be used in generating digital signatures, message authentication codes, and many other security applications in the information infrastructure

SHA-3

- a tunable security parameter, such as the number of rounds, which would allow the selection of a range of possible security/performance tradeoffs
- a recommended value for each digest size

SHA-3

- First round submissions (64 entries) and conf – Dec 2008 – 51 candidates
- Second round – 14 candidates – July 2009
- Third round – 5 finalists – Dec 2012

5 finalists

- BLAKE
- Grøstl
- JH
- Keccak
- Skein

SHA3

- Performance, Security, CryptAnalysis, Diversity
- Winner – Keccak (Sponge functions)!!!

HF design and security issues

- Known Answer Tests (KATs) and Monte Carlo Tests (MCTs)
- Known Answer Test values must be provided with submissions, which demonstrate operation of the SHA-3
- candidate algorithm with varying length inputs, for each of the minimum required hash length values (224, 256, 384, and 512-bits).
- There are three types of KATs that are required for all submissions: 1) Short Message Test, 2) Long Message Test, and 3) Extremely Long Message Test (2^{32} bits).

MCT

- The Monte Carlo Test provides a way to stress the internal components of a candidate algorithm.
- A seed message will be provided. This seed is used by a pseudorandom function together with the candidate algorithm to generate 100,000 message digests.
- 100 of these 100,000 message digests, i.e. once every 1,000 hashes, are recorded as checkpoints to the operation of the candidate algorithm

HF Security issues

- collision-finding, first-preimage-finding, second-preimage-finding, length-extension attack, multicollision attack
- How these hash codes satisfy essential properties of a hash function?
 - 1. All outputs are equally probable.
 - 2. HF should provide security with some computational complexity proof.
- B. Changing a single bit in input text will affect many output bits, therefore manipulating the effect of modification by complementing other bits in input is not easy.
- Compression function f should be fairly secure.

Security tests - compression function

- 1-bit sensitivity test – change only a single bit in an input block (one after another and look for output collisions)
- 2-bit sensitivity test (change two bits – balancing)
- Exhaustive collision detection test (take input message of 1 block, no padding, compute hash, modify input block exhaustively and compute hash, look for collisions)
- at input, any regular pattern)

Tests

- Statistical tests (correlations between input-output) any hash bit is
 - → influenced by no. of 0's or no. of 1's in the input?
 - → influenced by any particular group of input bits?
- Avalanche effect (changing a 1-bit at input changes how many bits in output?)
- Security - **input discovery**, collisions, Any functional weakness
 - specific input patterns (for all 0's, all 1's)

Summary

- Hash function generates a small identifier of a large object (number, message, file, document, etc.)
- Speed of computation – fast (simple Mathematical functions used)
- Computational Complexity increases linearly with digest size and input length (no. of blocks processed)
- Memory requirement (for storing temporary values)
- Published material (IV, Padding technique (bit sequence, byte count))
- Scalability (by increasing block length)
- Security issues more rounds of computation remove predictable output behavior for specific input patterns

Application use case: Software licensing

- Software distribution kit (Key references)
- MAC address of Network Card or CPU ID of intended Computer where software is to be installed
- Hash of the above two
- Generated - License key for installation
- Installation - keep this at secure location of computer
- Check – read MAC address or CPU ID, compute Hash along with key references, compare result with txt stored at secure location – if matches; allow software to run