Secret Writing 2

Images in these slides were, in some cases, taken from support material and books available from W. Stallings, M. Kaeo and B. Schneier

Last Time

Terminology and Taxonomy

Some History

Goals of all ciphers

- *Confidentiality*
- *Authentication*
- *Integrity*
- *Non-repudiation*

Ciphers generally

- $P = D_k,[E_k[P]]$

Last Time

Examples

- *Ceasar transposition cipher*
- *Monoalphabetic substitution*
- *Polyalphabetic substitution*

Concepts

- *Stream ciphers*
- *Block ciphers*
- *Transposition and substitution*
- *Side information*
- *Confusion and diffusion*
- *Block chaining*

Last Time

Symmetric (private, private) key encryption

- *Details of DES*
- *Details of AES*
- *Mentioned a few others*
  - Triple DES
  - IDEA

Last Time

What is a major issue with private (symmetric) key encryption?

Key distribution

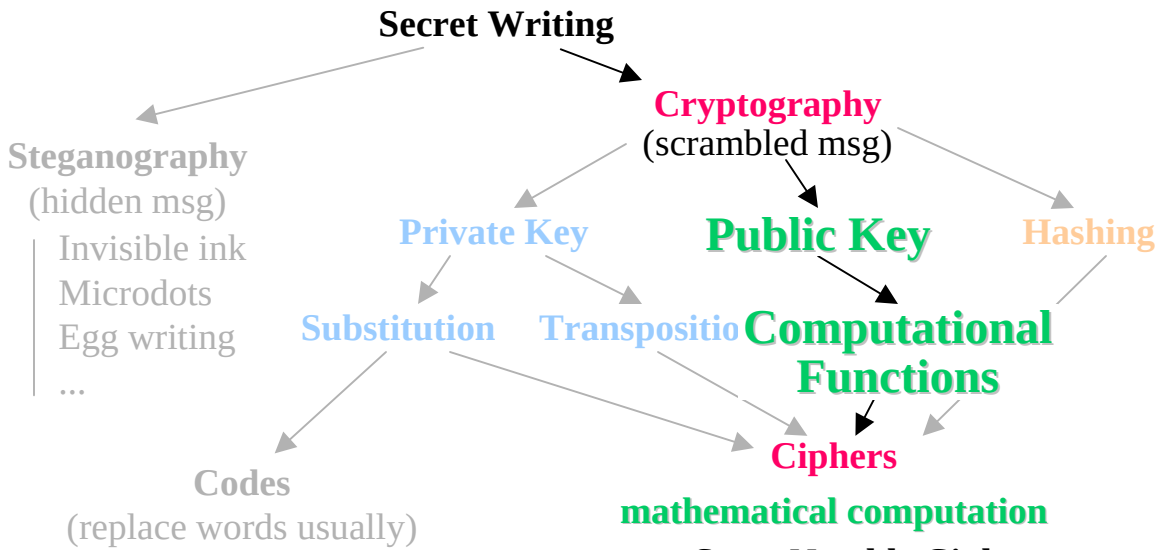
How the recipient(s) get the key

Now we will consider public key encryption

Provides a solution to key distribution

Asymmetric (Public Key) Ciphers

Taxonomy Again



Cryptography and **Ciphers** usually refer to the same things, with Cryptography being the science and Ciphers being the encoded entities. The only exception is **Codes.**

Private key ciphers use S and T.

Public key systems use CF.

S = substitution
T = transposition
H = hashing
CF = computational functions

Some Notable Ciphers:

Caesar (T)	Monoalphabetic (S)
Polyalphabetic (T&S)	Vigenère (S)
One-time pad (S)	Enigma (T&S)
DES (T&S)	RSA (T&S)
IDEA	MD5 (H)
RSA Pub (CF)	Diffie-Hellman (CF)
Elliptical (CF)	SHA-1 (H)
AES (T&S)	

Public Key Background

A major problem with private key systems is the distribution of keys in a secure manner

This has plagued cryptographers for millennia

A solution finally arrived in 1976

Diffie and Hellman (Stanford) published the 1st paper describing a public key system

Actually Whitfield Diffie figured it out

It's one of these ideas that is so simple that it's a wonder that it was not thought of centuries ago

It's a good example of how "thinking out of the box" is very important

Back to Private Keys for a Moment

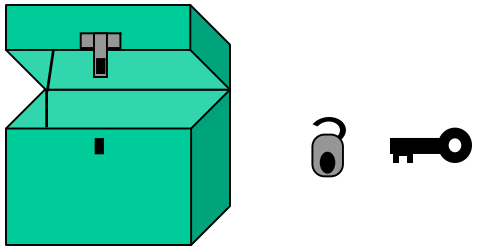
Before we go on to public key schemes, let’s go back to private keys in order to give some context

Let's look at an analogy to encryption using private keys

Private Key Box Analogy

Alice and Bob wish to privately communicate by putting their messages in a box and sending the box back & forth

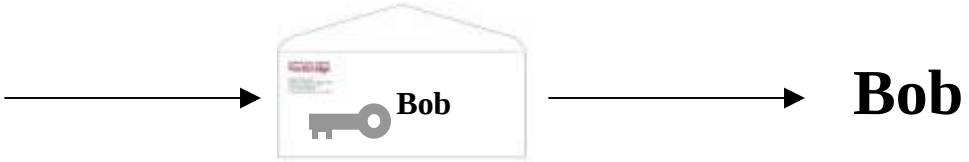
The box has a hasp, padlock and key



Alice has a 2nd key made for the padlock

She keeps one

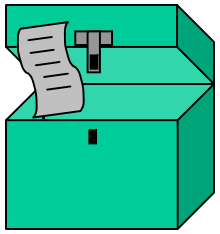
She sends the other key to Bob in a way the ensures that no one can intercept the key & make a copy of it



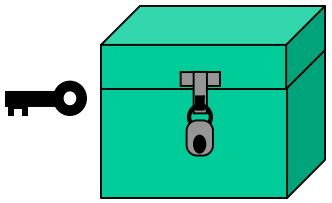
This of course is the key distribution issue. **It's a problem!**

Private Key Box Analogy

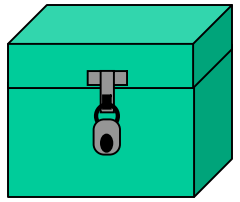
Alice puts a message in the box



Closes and locks it with her key

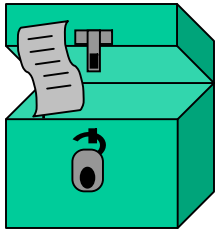


Sends the locked box to Bob



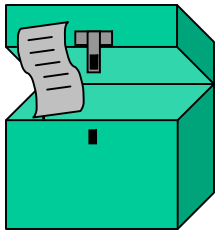
Bob

Bob unlocks the box with his key and reads the message

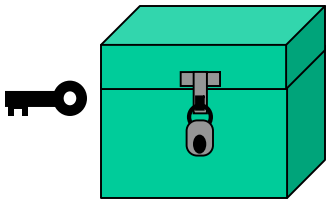


Private Key Box Analogy

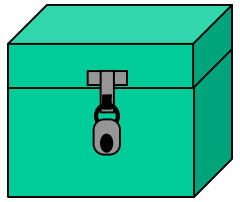
Alice puts a message in the box



Closes and locks it with her key



Sends the locked box to Bob



Bob

Bob unlocks the box with his key and reads the message

Private Key Box Analogy

Bob is sure that the message really came from Alice only if

The key was distributed in a secure fashion

Ditto for Alice receiving messages from Bob

Issues With One Key Solution

Eve might pick the lock on its way to Bob

Eve could then read the message and then reclose the box & forward it to Bob

Bob would not know that the message had been read

Alternately Eve could read the message and then discard the box and message

Bob would never get the message

Maybe Bob and Alice would eventually decide that the message was either lost or intercepted

Issues With One Key Solution

Eve might break the padlock or the box

Bob might then receive the box or the message or both but would know that it might have been read

Alternately Eve might discard the broken box

Bob might never get the box or the message or both

Eve might simply discard the locked box without ever opening it

Bob would never receive the message

Issues With One Key Solution

But even if none of the previous things ever happen, there would still be one major issue

Eve might intercept the key on its way to Bob, make a copy and then forward the key on to Bob

Bob would receive the key and not know that Eve had made a copy

Key distribution is a MAJOR problem with private keys

How to Securely Send the Key?

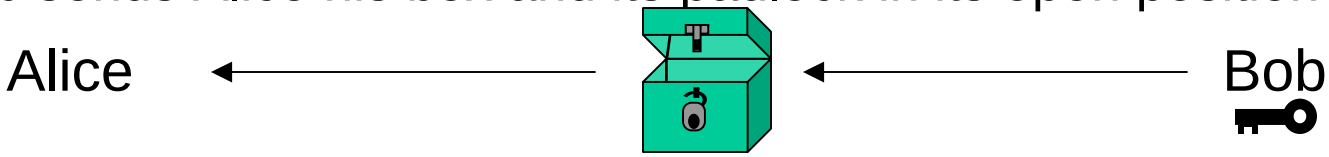
Sending the key securely perplexed Whitfield Diffie

How can Alice send Bob a secure message without first sending the key?

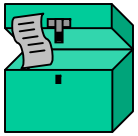
Diffie's answer: **Don't send the key!**

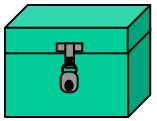
Don’t Send the Key? How?

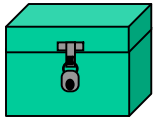
Bob sends Alice his box and its padlock in its open position



Alice has no Bob key. Bob keeps the only padlock key

Alice puts the message in the box 

Alice then locks the box with the padlock 

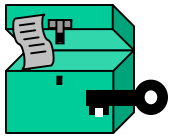
Returns the locked box to Bob  Bob

Bob receives the locked box, unlocks it and reads Alice's message

This works!

But there’s a problem. What is it?

Bob can’t send Alice messages



How Can We Send Messages in Both Directions without Distributing a Key?

What if the there were two locks each with a different key?

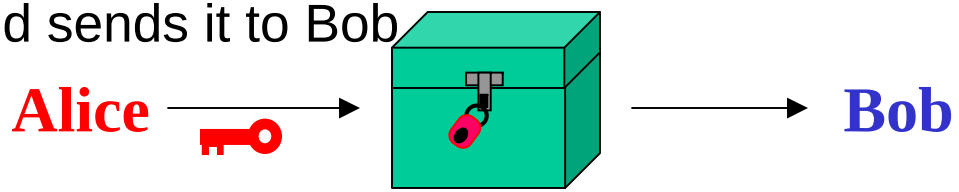
Alice and Bob each have a different lock with a key

Each keeps their own key in a safe place

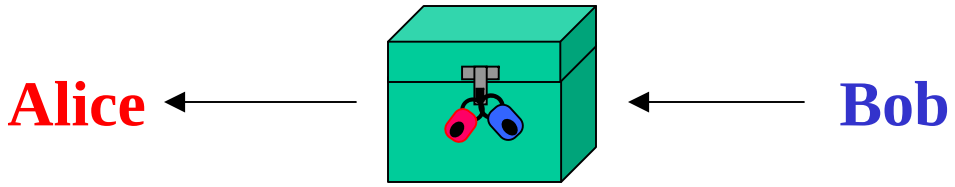
They never send any keys to anyone

Two Locks & Two Keys

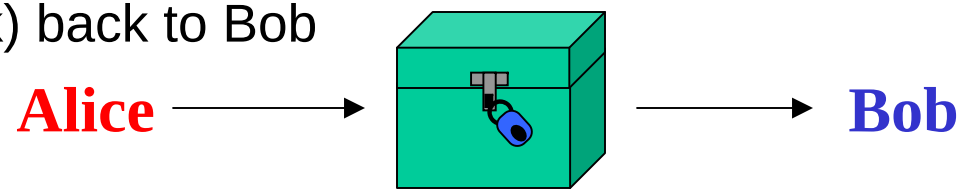
Alice puts the message into the box, locks it with her padlock & key, and sends it to Bob



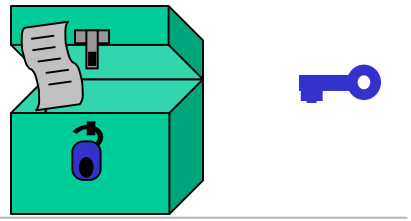
Bob puts his padlock on the box & sends it back to Alice



Alice removes her padlock and sends the box (still locked with Bob's padlock) back to Bob



Bob opens the box and reads the message



Issues With Two Key Solution

Box must be sent three times

Locks might still be picked or broken

But a major problem has been overcome.

Diffie & Hellman's Contribution

Although there were problems, the implications of the two key scenario were **enormous!**

It said that two people can securely exchange messages without exchanging keys

*For the **very first time** ever recorded, it suggested that key distribution was not inevitable*

Rather obvious when one thinks about it

Amazing that it took thousands of years to discover it

But how can one implement such schemes?

Diffie & Hellman investigated a branch of mathematics called one-way functions

Asymmetric (Public Key) Ciphers

Encrypt by treating the message as numbers and by transforming it using a one-way math function

Two keys, one public (Kpu) and one private (Kpr)

Math function requirements:

1. *Easy to generate a key pair, Kpr & Kpu for E & D*
2. $C = E_{Kpu}(P)$ *is easy*
   E_{Kpu} is encryption algorithm using the public key
3. $P = D_{Kpr}(C)$ *is easy*
   D_{Kpr} is decryption algorithm using private key
4. $P = D_{Kpu}(C)$ *is* ***computationally infeasible***
   D_{Kpu} is decryption algorithm using public key
5. *Infeasible to determine Kpr from Kpu, D and E*

One-way Functions*

These are mathematical functions that are easy to calculate in one direction but hard to calculate in the reverse direction

Modular mathematics was the basis for Diffie's and Hellman's approach

Hellman made the modular math discovery

Diffie, Hellman, and Merkle (who later joined them in the work) refined the work

One-way Functions We All Know About*

Large Product

Let the number 5280 = A x B. Find A and B?

How can we easily determine the factors A & B?

Use Iterative factorization:

5280/2 = 2640; 2640/2 = 1320; 1320/2 = 660; 660/2 = 330; 330/2 = **165**

Five divisions by 2 is the same as dividing 5280 by 2^5 or **32**

Thus A = 32 and B = 165

But it's also true that 5280 = 33 x 160

Actually there are several answers

So this doesn't tell us much

One-way Functions We All Know About*

Large Product (continued)

Now let the number 3287 = A x B. Find A and B?

A = 173 and B = 19

This **is** the only answer! Why?

*A and B are both prime numbers**

Because A and B are both prime, it's more difficult to factor. Why?

Can't perform iterative factorization.

But its easy to multiply 173 x 19 to get 3287

Hence we have a simple one-way function

*Products of numbers whose factors are relatively prime** are a type of one-way function*

*A prime number is only divisible by itself and the number 1.

** Two numbers are relatively prime if their greatest common divisor (gcd) is 1. Example? e.g., 8 and 15 are relatively prime, but neither is a prime number

Another One-way Function We Know About*

Factoring Polynomials

Find roots of the 3rd order polynomial $x^3+2x^2-45x-126$

Find the factors

$x^3+2x^2-45x-126 = (x^2-4x-21)(x+6) = (x+3)(x-7)(x+6)$

Sort of difficult; isn't it?

And suppose that we have a 50^{th} order polynomial.

But its easy to multiply several polynomials together

$(x+3)(x-7)(x+6) = (x^2-4x-21)(x+6) = x^3+2x^2-45x-126$

So polynomial factorizations are one-way functions

Modular Arithemtic

Modular arithmetic concerns itself with the remainders of division operations

13/3 = 4 with remainder = 1

13mod3 = 1

127/8 = 15 with remainder = 7

127mod8 = 7

A = 17mod3 A = ?

B = 121mod8 B = ?

C = 63mod7 C = ?

One Way Function Example*

x	1	2	3	4	5	6
5^x	5	25	125	625	3125	15625
$5^x(\text{mod}7)$	5	4	6	2	3	1

Given the value of x, it's not too hard to calculate
$5^x(\text{mod}7)$ using a computer

But try to determine x in the equation $5^x(\text{mod}7) = 6$

Next, solve for **x** in these equation examples

$137^x(\text{mod}1673) = 1625$ x = ???

$231{,}457{,}859^x(\text{mod}179{,}531) = 53{,}579$ x = ???

Example: Exchanging a Private Key Without Exchanging It

How can Alice and Bob securely distribute a private key using one-way functions?

Alice and Bob agree on the one-way function $7^x(\text{mod}11)$

They agree about this in public so Eve can know it

ALICE	***BOB***
1. Alice chooses a number A *Let A = 3* *Alice keeps her number secret*	1. Bob chooses a number B *Let B = 6* *Bob keeps his number secret*
2. Alice calculates $a = 7^A(\text{mod}11) = 343(\text{mod}11) = \mathbf{2}$	2. Bob calculates $b = 7^B(\text{mod}11) = 117649(\text{mod}11) = \mathbf{4}$

Example: Exchanging a Private Key Without Exchanging It*

3. Alice sends (a = 2) to Bob

   Transmissions are in the clear.

3. Bob sends (b = 4) to Alice

   Anyone can see these numbers.

But is very difficult to determine anything even if you know 2, 4, and $7^x(mod11)$ because you don't know the keys A & B that Alice & Bob kept to themselves.

ALICE

Alice now has A=3 & b=4

4. Using b & A, Alice calculates

$$b^A(mod11) = 4^3(mod11)$$

$$= 64(mod11)$$

$$= \mathbf{9}$$

BOB

Bob now has B=6 & a=2

4. Using a & B, Bob calculates

$$a^B(mod11) = 2^6(mod11)$$

$$= 64(mod11)$$

$$= \mathbf{9}$$

The number *9* is the symmetric (private) key.

Learning A Symmetric Key Without Exchanging It

Diffie, Hellman, and Merkle had developed a key "exchange" process that could be done publicly but still be secure

But it exchanged symmetric (not asymmetric) keys

Also it was not really an exchange of keys

A way for two sides to determine the same key from

Public information that they exchanged

Private information that they alone knew

Asymmetric Keys

Diffie was not satisfied because:

Both Alice and Bob needed to be available

Alice needed info from Bob

Bob needed info from Alice

The private key couldn't be determined by either of them without the info from the other

With symmetric keys, the same key is used to encrypt and decrypt the message

Diffie developed the idea of asymmetric keys

Two related keys: one to encrypt and one to decrypt

Back To Padlocks

Padlocks are essentially asymmetric

Don't need a key to lock it (sort of like a trivial key)

Need a key to open it

Now suppose that Alice manufactures 1000s of her padlocks but only 1 key

Bob does the same for his padlocks and key

They distribute their respective padlocks far and wide to friends, Parcel Post, UPS, FedEx, Wallgreen's, CVS, Jewel Foods, Etc.

They each keep their single key in a safe place

To send a message to Bob, Alice puts the message in a box, get a "Bob Lock" from Osco, locks the box, and sends it to Bob.

Bob gets the box, unlocks it with the only existing key and reads the message

From Padlocks to Math

What Diffie needed was a one-way math function that is

Reversible

But the number used to transform is different than the number used to reverse the transform

The RSA algorithm does this

Based upon large relatively prime numbers, modular functions and factorization

Remember our Large Product example?

$231{,}457{,}859^{x} (\text{mod} 179{,}531) = 53{,}579 \qquad x = ???$

RSA Algorithm

RSA stands for Rivest, Shamir and Adleman

MIT

Published in 1978

It is widely used and implemented today

RSA Process Simplified

Alice picks 2 large prime numbers **g** & **h**

Alice calculates ***n*** & ***q***, chooses the number ***u***, and then calculates ***r***

$n = g \times h$

$q = (g-1)(h-1)$

Chooses ***u*** *such that* ***q*** *and* ***u*** *are relatively prime*

$r = u^{-1} \pmod{q}$ [i.e. **r** x **u** ≡ 1mod **q** and **r** < **q**]

The keys are then:

K_{Apu} : *Alice's public key* $= \{u,n\}$

K_{Apr} : *Alice's private key* $= \{r,n\}$

Block lengths of P and C must be $< n$

RSA Process Simplified

Alice publishes $\mathbf{K_{Apu}} = \mathbf{\{u,n\}}$

Now Bob wants to send a message to Alice

*He breaks up his plaintext message to Alice into blocks P_{Bi} (of lengths Bi < **n**) and encrypts each one with **u***

Bob treats each block as a number & calculates

$C_i = (P_{Bi})^{\mathbf{u}} \pmod{\mathbf{n}}$

*This is how Bob encrypts using Alice's public key {**u**,**n**}*

Bob concatenates all his encrypted blocks and sends them to Alice

$C = C_1 \,||\, C_2 \,||\, \ldots$

(Note that Eve does not know the block size; only its upper bound.)

RSA Process Simplified

Alice

Receives the cyphertext message C

Breaks it into a series of blocks C_i

Calculates (decrypts) P_i *for each block using her private key* $\{\mathbf{r},\mathbf{n}\}$ *as follows:*

$$P_i = (C_i)^r \pmod{\mathbf{n}}$$

Simple RSA Example *1 of 3*
Setup & Selection of Keys

Let's choose

$g = 3, h = 11, u = 7$

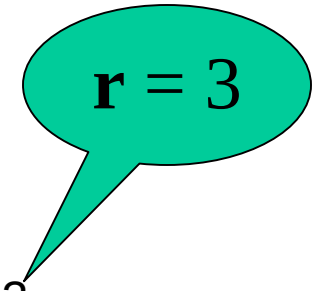
Calculate

$n = g \times h = 3 \times 11 = 33$

$q = (g-1)(h-1) = 2 \times 10 = 20$

Are **u** and **q** relatively prime?

$r = u^{-1} \pmod{q} = ??$ (Note: $r \times u \equiv 1 \bmod 20$)



If ru = 21 = 1 mod 20 then 21 ÷ u = 21 ÷ 7 = 3

If ru = 41 = 1 mod 20 then 41 ÷ u = 41 ÷ 7 = 5.857…

If ru = 61 = 1 mod 20 then 61 ÷ u = 61 ÷ 7 = 8.714…

…

What's the value of **r** ?

Simple RSA Example *2 of 3*
Enciphering

So far

$K_{pu} = \{u,n\} = \{7, 33\}$ *and* $K_{pr} = \{r,n\} = \{3, 33\}$

Suppose that the plaintext is the number "5"

$P = 5$

Then the ciphertext C is

$C = P^u \bmod n = 5^7 \bmod 33 = (78{,}125) \bmod 33 = ???$

78,125 ÷ 33 = 2,367.424242…

The fractional part of the product is the modular value

0.424242 x 33 = 13.999986 (on my calculator) ≅ 14 (dangerous)

Better way: 2,367 x 33 = 78,111 and 78,125 – 78111 = 14

$C = 14$

Simple RSA Example *3 of 3*
Deciphering

Now determine **P** from **C** = 14 and $\mathbf{K_{pr}}$ = {**r**,**n**} = {3, 33}

$P = C^r \bmod \mathbf{n} = 14^3 \bmod 33 = (2744) \bmod 33 = ???$

2744 ÷ 33 = 83.151515…

The fractional part of the product is the modular value

0.151515 x 33 = 4.999995 (on my calculator) ≅ 5 (dangerous)

Better way: 83 x 33 = 2739 and 2744 – 2739 = 5

P = 5 *Q.E.D.*

Other Uses for Public Keys

We now understand how to send secure messages using a public key system where we don't need to deal with key management as with private key systems

But originally we said that we needed encryption to achieve 4 things -- or maybe 3

Confidentiality (i.e., security)

Integrity

Authentication

Non-repudiation?

Other Uses for Public Keys

Private key systems achieve

Confidentiality

Authentication

But only if Alice and Bob can really be sure that they are the only two possessors of the key

On the other hand, public key systems can achieve all 4 of the goals without the above constraint

Or the 3 goals depending on how you count

What were the 4 goals?

The Goals

So up to now we can achieve:

Confidentiality: Ensures that meaning of message is hidden for outsiders.

Authentication: Establishes the identity of the sender of the message.

We still need to achieve:

Integrity: Ensures that the message and its authentication have not been changed.

Non repudiation: The sender cannot deny that she or he sent the message. Closely related to Authentication.

Let's go over *Confidentiality* and *Authentication* again

Asymmetric Encryption

Confidentiality but No Authentication

1: Alice & Bob each create key pairs

2: They exchange their public keys

3: Alice sends message to Bob encrypted with Bob's public key

4: Bob receives Alice's message and decrypts it with his private key

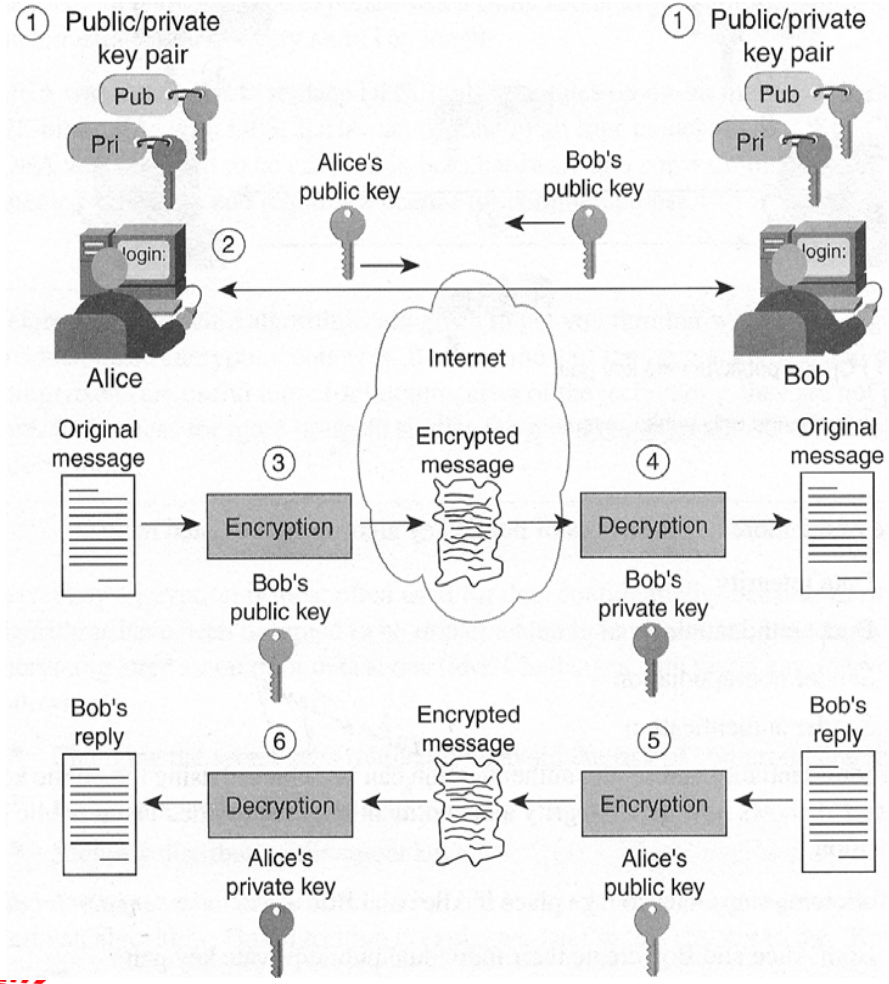
5: Bob encrypts a reply to Alice with her public key

6: Alice receives Bob's message and decrypts it with her private key

Confidentiality achieved!

Problem: No authentication!

Easy for Eve to send an encrypted message to Bob claiming that she is Alice



Asymmetric Encryption

Authentication but No Confidentiality

1: Alice & Bob each create key pairs

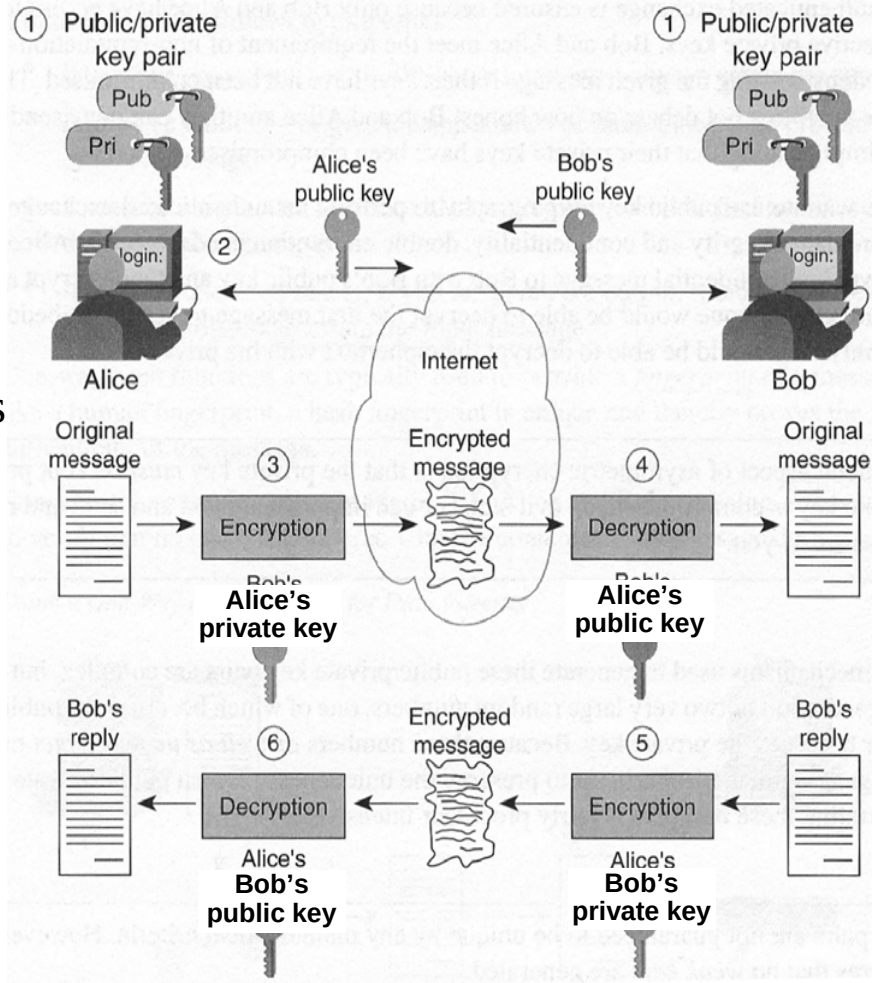
2: They exchange their public keys

3: Alice sends message to Bob encrypted with Alice's private key

4: Bob receives Alice's message and decrypts it with Alice's public key

5: Bob encrypts a reply to Alice with his private key

6: Alice receives Bob's message and decrypts it with his public key



Authentication achieved!

Problem: No confidentiality!

Easy for Eve to decrypt either Alice's or Bob's messages because she (and everyone else) has their public keys

How Do We Get Both?

To get both, we must double encrypt

For Alice to send a confidential and authenticated message to Bob

- *Alice encrypts the message to Bob with Bob's public key*
- *Alice then encrypts the encrypted message with Alice's private key*

Bob receives the doubly encrypted message

- *Bob first decrypts the message with Alice's public key*
- *Next Bob decrypts the message with his private key*

If the message is understandable plaintext

- *It is secure and it did come from Alice*

How Do We Get Both?

Alice

$C_s = E_{Bpu}[P]$

$C_{sa} = E_{Apr}[C_s]$

C_{sa} ⟶

Bob

C_{sa}

$D_{Apu}[C_{sa}] = C_s$

$D_{Bpr}[C_s] = P$

If the message is readable and makes sense, it really did come from Alice, and it is confidential.

Still A Problem!

Public (asymmetric) key systems are very computationally intensive

Private (symmetric) key systems are much more efficient with respect to computation efficiency

And what about message integrity?

An efficient alternative to public key encryption for authentication that also deals with integrity is hashing

Hashing

Hash Functions

A hash function ***H***

Takes as input a data block ***X*** *of variable length*

X can be a message, file image or other block of data

Outputs a relatively short fixed length value or code, ***h***

$$h = H(X)$$

The hash, ***h,*** is in some sense an **identifier** of the message, file, image or other block of data

H is the hash function

H *creates* ***h*** *from* ***X***

Sometimes called a “message digest”

Uses of Hash Functions

Hash functions are used for message integrity

Allows receiver to determine if message has been corrupted

While a hash function can also be used for message authentication or digital signatures, it is not as secure as if it were combined with encryption

Hashes can be used with keys for these purposes

Called message authentication codes (MACs)

More on MACs later

Seven General Properties of a Hash Function

1. Given X, the hash function H generates a fixed length hash, $h = H(X)$
2. The data block, X, used to create the hash, can be of any length
3. $h = H(X)$ is reasonably easy to compute

   Allows fast practical software and hardware implementations
4. For a given X, h will always be the same

Computationally Infeasible Properties of a Hash Function

5. Given X, it is computationally infeasible to find a block of data $Y \neq X$ such that $H(Y) = H(X)$
   - *"Weak collision" criteria! Sort of means that* $h = H(X)$ *can be considered unique for a chosen* X
6. Computationally infeasible to determine X, given h
   - *One-way! Given the hash of a message, computationally infeasible to construct the original message*
7. Computationally infeasible to find any pair of data blocks (X,Y) such that $H(Y) = H(X)$
   - *"Strong collision" criteria! Low probability of two data blocks generating the same hash. (Birthday attack resistance.)*

General Scheme for Hashing

All hash functions follow the same general scheme

The data block X *is partitioned into a sequence of* L *fixed-bit-length sub-blocks,* Y_i *, each of length* b

$X = Y_0\ Y_1\ \ldots\ Y_{L-1}$

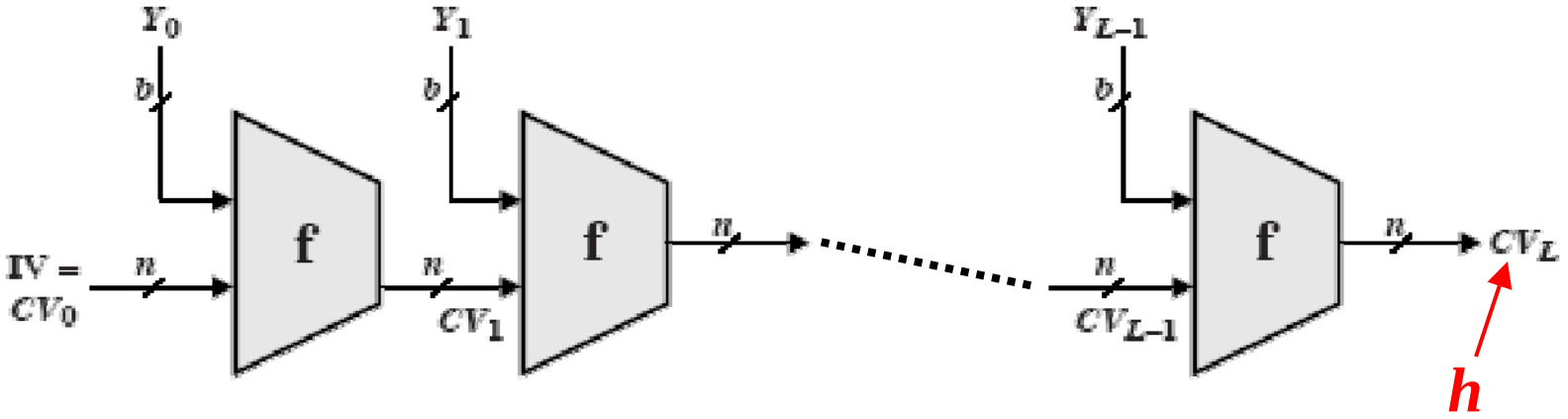
$X =$ $b_{01}\ b_{02}\ b_{03}\ b_{04}\ \ldots\ b_{0b}$ $b_{11}\ b_{12}\ b_{13}\ b_{14}\ \ldots\ b_{1b}$

$\ldots$ $b_{L-1,1}\ b_{L-1,2}\ b_{L-1,3}\ b_{L-1,4}\ \ldots\ b_{L-1,b}$

Then the sub-block are processed one at a time and processed outputs are chained together to create the n-bit hash value h

"Block chaining"

General Scheme for Hashing



IV = CV_0 = initial value
CV_i = chaining variable
Y_i = ith input block
f = chaining algorithm
L = number of sub blocks in data block X
b = length of each sub block in bits
n = length of hash h in bits
CV_L = h

Some simple Hash Functions

Some Simple Hash Functions

Hash functions used today are necessarily complex

In order to meet the computationally infeasible requirements

It's instructive to first look at some simple hash functions

These are too simple for effective use but they demonstrate principles hard to see when one examines more complex hash functions

Once we look at these simple functions, we'll consider hashes of more complexity

Simple Hash Functions
Bit-by-Bit Exclusive OR (XOR, ⊕)

Let

X = $b_{01}\ b_{02}\ b_{03}\ b_{04}\ \ldots\ b_{0b}\ b_{11}\ b_{12}\ b_{13}\ b_{14}\ \ldots\ b_{1b}$

... $b_{L-1,1}\ b_{L-1,2}\ b_{L-1,3}\ b_{L-1,4}\ \ldots\ b_{L-1,b}$

Then the individual bits of the hash value are

$h_0 = b_{01} \oplus b_{11} \oplus \ldots \oplus b_{L-1,1}$ (1st bit of hash)

$h_1 = b_{02} \oplus b_{12} \oplus \ldots \oplus b_{L-1,2}$ (2nd bit of hash)

...

$h_i = b_{0i} \oplus b_{1i} \oplus \ldots \oplus b_{L-1,i}$ (ith bit of hash)

...

$h_b = b_{0b} \oplus b_{1b} \oplus \ldots \oplus b_{L-1,b}$ (bth bit of hash)

And the hash value is

$h = h_0\, h_1 \ldots h_i \ldots h_n$

XOR Example

My last name is *Lidinsky*

In ASCII,

Lidinsky = 01001100|01101001|01100100|01101001
|01101110|01110011|01101011|01111001

Find the bit-by-bit exclusive-OR hash of my last name

XOR Example
Answer

L	0 1 0 0 1 1 0 0	
i	0 1 1 0 1 0 0 1	
	0 0 1 0 0 1 0 1	
d	0 1 1 0 0 1 0 0	
	0 1 0 0 0 0 0 1	
i	0 1 1 0 1 0 0 1	
	0 0 1 0 1 0 0 0	
n	0 1 1 0 1 1 1 0	
	0 1 0 0 0 1 1 0	
s	0 1 1 1 0 0 1 1	
	0 0 1 1 0 1 0 1	
k	0 1 1 0 1 0 1 1	
	0 1 0 1 1 1 1 0	
y	0 1 1 1 1 0 0 1	
	- - - - - - - - - - - - - - -	
	0 0 1 0 0 1 1 1	(ASCII single quote ‘)

XOR Truth Table

	0	1
0	0	1
1	1	0

Simple Hash Functions
Bit-by-Bit Exclusive OR

Pros

Very easy to implement in either hardware or software

Cons

Not useful for data integrity

- Each bit of the hash value is a function only of the correspondingly positioned bits in $\mathbf{h}$. (No avalanching!)
- For random data $\boldsymbol{X}$ the probability that a single bit change will not change the value of $\boldsymbol{h}$ is 2^{-n}
- For more predictable data such as ASCII code where the high order bit is always 0, the probability that a single bit change will not change the value of $\boldsymbol{h}$ is $< 2^{-(n-1)}$ *(It's halved or worse.)*

Simple Hash Functions
1s Complement Check Sum

This is the same checksum that is used in IP, UDP and TCP headers

Treat each sub-block Y_i of the data block X as an b-bit number

Sum (1s complement) the sub-blocks

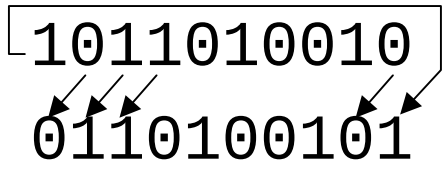
$$h = \sum_{i=0}^{L-1} Y_i = Y_0 + Y_1 + \ldots + Y_{L-1}$$

Because of the carries in the 1s complement addition, a bit in the hash value h is likely to be influenced by other bit positions in the sub-blocks

Avalanching!

Simple Hash Functions
XOR with a Barrel Shift

A left barrel shift moves each bit to the left and moves the leftmost bit to the right end of the number



A right barrel shift just shifts to the right and moves the rightmost bit to the left end of the number

Also called a ***circular shift*** or ***rotation***

This hash function does a barrel shift after each sub-block Y_i is $\oplus$ with the previous result

Simple Hash Functions
XOR with a Barrel Shift

Pros

- *Almost as simple to implement as the bit-wise ⊕*
- *Effective for data integrity*
- *Good avalanche effect*
  - Each bit of hash **h** is a function of many of the bits in **X**

Cons

- *Not useful for authentication of a plaintext message*
  - It is easy to create an alternate message with the same hash
  - How?
    - *Append a final block to the alternate message that forces the hash to be the same as that of the original message*

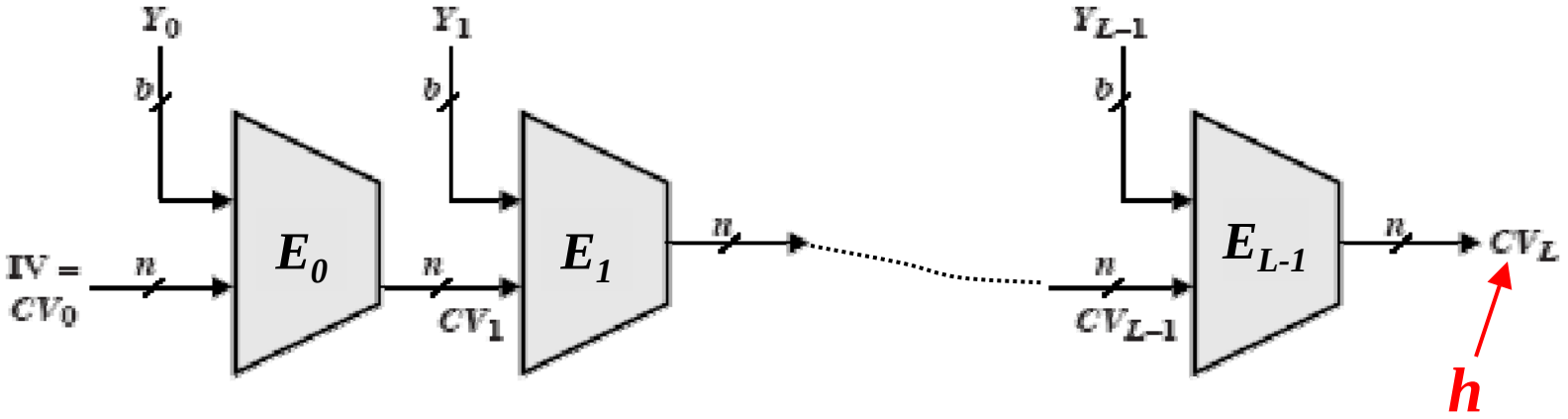
More Useful (and Complex) Hash Functions

Overview

Several useful hash functions have been developed based upon block chaining used in symmetrical encryption algorithms

Sans a secret key

Overall Block Diagram for Hashing



IV = CV_0 = initial value
CV_i = chaining variable
Y_i = ith input block
E_i = block chain encryption algorithm
L = number of sub blocks in data block X
b = length of each sub block
n = length of hash h
CV_L = h (the hash itself)

Commonly Used Hash Algorithms

MD4

MD5

SHA-1

SHA-256

RIPEMD-160

HMAC

MD4 and MD5

MD4

MD4: Message Digest version 4

Developed by Ron Rivest at MIT (of RSA fame)

RFC1186 (1990) and RFC1320 (1992)

Hash length = 128 bits

Basic unit of processing = 512 bits

Maximum message size = ∞

Fast and efficient

MD4 is optimized for little endian CPUs

Rivest felt that it was not complex enough to thwart more sophisticated cryptanalysis

MD5

MD5: Message Digest version 5 (RFC 1321, 1992)

Developed by Ron Rivest to replace MD4

Based closely upon MD4

Hash length = 128 bits

Basic unit of processing = 512 bits

Maximum message size = ∞

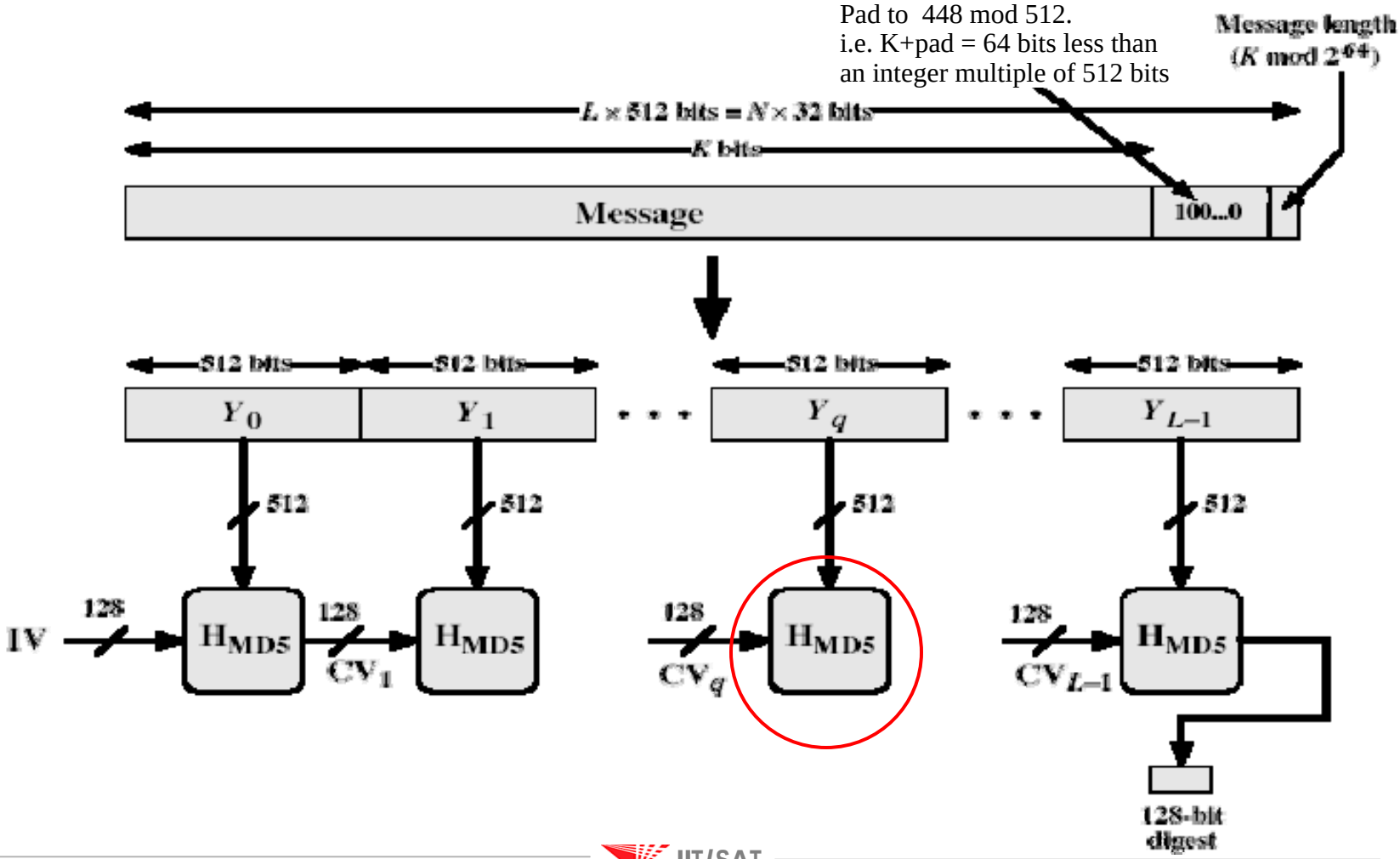
Every bit of $\boldsymbol{h}$ is a function of every bit of $\boldsymbol{X}$

Complete avalanching

Somewhat less speedy than MD4

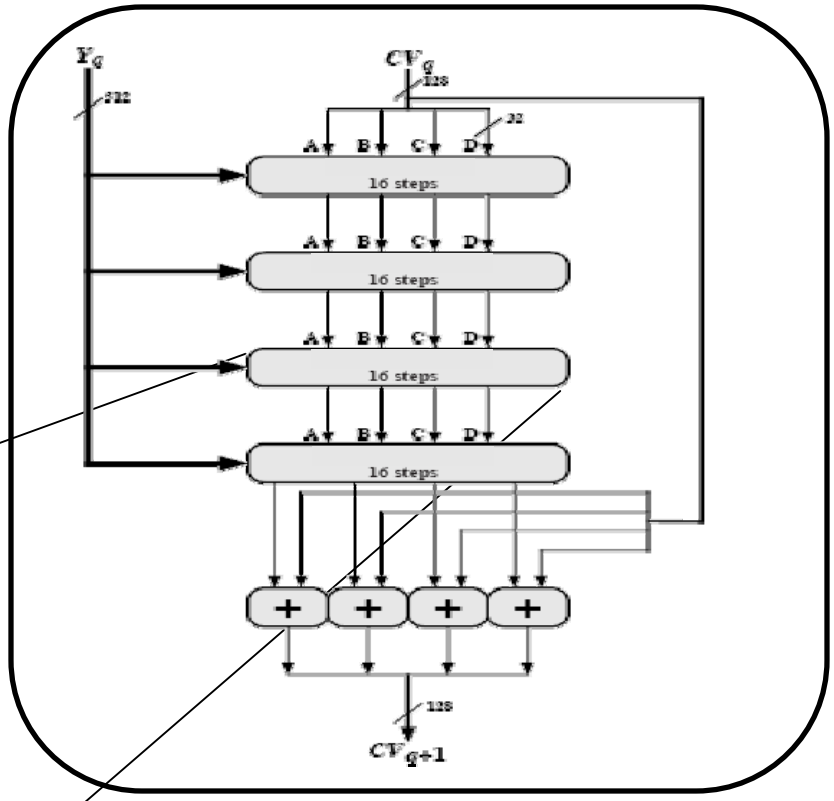
But processing was getting faster

MD5

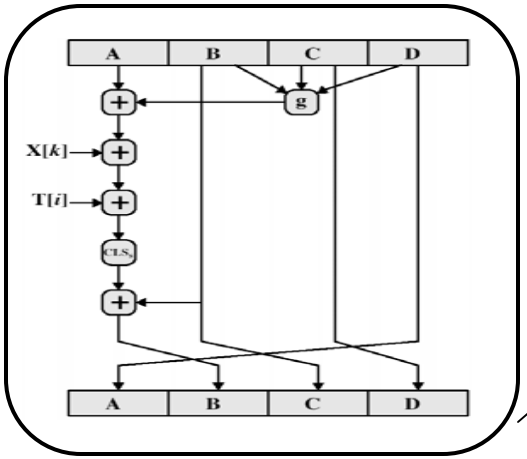


H_{MD5} Box

This diagram of the operation of a H_{MD5} box shows the complexity of the function.
All additions are mod 2^{32}.



X[k] = kth 32-bit word in the qth 512 bit block of the message
T[i] = ith 32-bit word in a intermediate matrix constructed from a sine function



1. ***g*** executes Logical Equation 1 on B,C,D.
2. The register ABCD then does a 32-bit right barrel shift. (A->B, B->C etc.)
3. ***g*** again executes Logical Equation 1 on B,C,D.
4. For each barrel shift ***g*** executes Logical Equation 1 for a total of four times

Then ***g*** changes to Logical Equation 2 and steps 1-4 are repeated. ***g*** changes its logical equation 4 times.

MD5 Vulnerability

The bad guys have become more capable

Advances in cryptanalysis

Increasing power of computers in the hands of black hats

Allows brute force attacks to succeed in a reasonable time

MD5 has become somewhat vulnerable

Security of 128-bit hash length has been questioned

MD5 has been shown to be vulnerable to birthday attacks of the order of effort of 2^{64} attempts

But it's still quite useful for many hash applications

Used a lot

Secure Hash Functions
SHA

SHA Overview

SHA developed by NIST in 1993

SHA-1 was developed by NIST in 1995

Based closely on MD4 and MD5 algorithms

But with key differences

SHA-1 standards documents: FIPS180-1 and RFC3174

Hash length = **160** bits

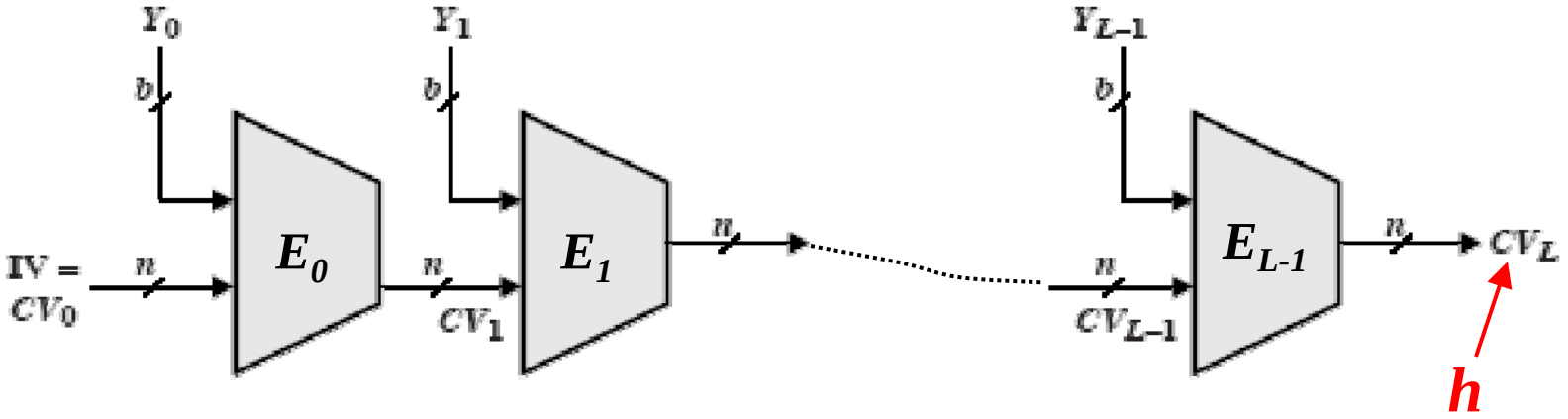
Basic unit of processing = 512 bits

Maximum message size < 2^{64} bits (1.8×10^{18} bytes)

Overall diagram is very similar to MD5

Used extensively today

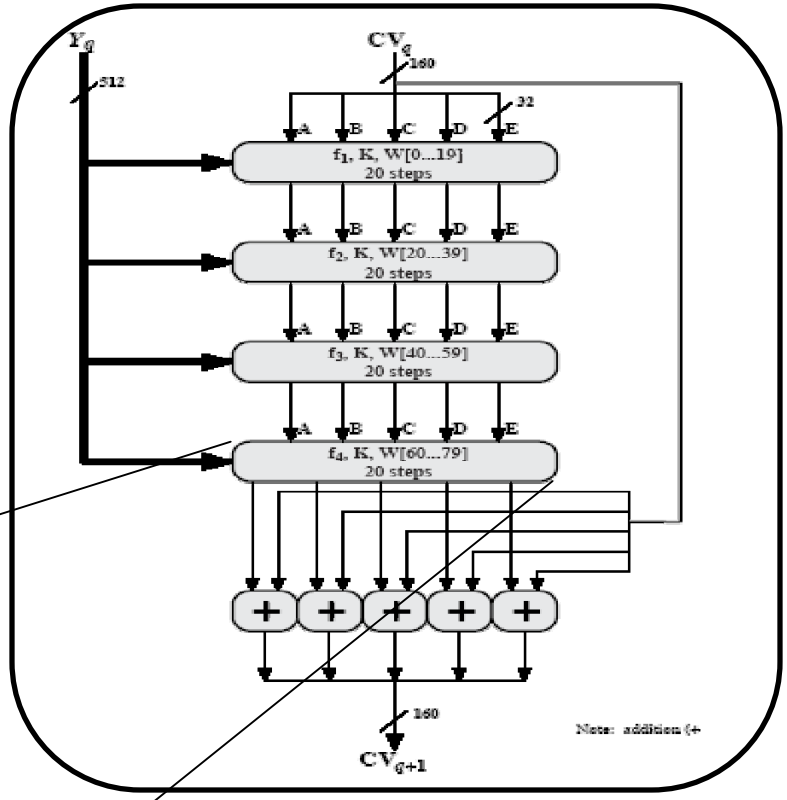
SHA-1 Overall Block Diagram



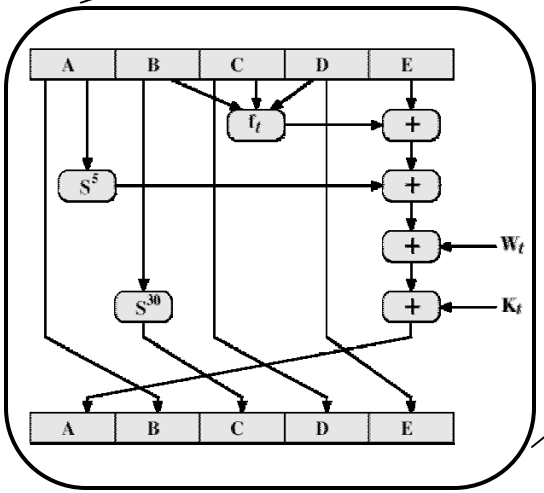
IV = CV_0 = initial value
CV_i = chaining variable
Y_i = ith input block
E_i = block chain encryption algorithm
L = number of sub blocks in data block X
b = length of each sub block
n = length of hash h
CV_L = h

SHA-1 Box

This diagram of the operation of a **SHA-1** box shows the complexity of the function.
All additions are mod 2^{32}.



W_i is the ith word in the message
$Wt = S^1(W_{t-16} \oplus W_{t-14} \oplus W_{t-8} \oplus W_{t-3}$
K[t] is one of 4 constants that are pre calculated



1. f_t executes Logical Equation 1 on B,C,D.
2. The register ABCDE then does a 32-bit right barrel shift.
   (A->B, B->C etc.)
3. f_t again executes Logical Equation 1 on B,C,D.
4. For each barrel shift f_t executes Logical Equation 1 for a total of **five** times

Then f_t changes to Logical Equation 2 and steps 1-4 are repeated.
f_t changes its logical equation 4 times.
S^i is a left barrel shift of i bits

SHA-1 & MD5 Comparison

Brute force attack is harder

160 vs. 128 bits for MD5

Not vulnerable to any known attacks (compared to MD5)

A little slower than MD5 (80 vs 64 steps)

Both designed as simple and compact

Optimised for big endian CPU's

MD5 is optimised for little endian CPUs

Revised Secure Hash Standard

NIST issued a revision FIPS 180-2

Adds 3 additional hash algorithms

SHA-256, SHA-384, SHA-512

Designed for compatibility with increased security provided by the AES cipher

Structure & detail is similar to SHA-1

Analysis should be similar to SHA-1

SHA Hash Algorithms

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)	Security[2] (bits)
SHA-1	$< 2^{64}$	512	32	160	80
SHA-256	$< 2^{64}$	512	32	256	128
SHA-384	$< 2^{128}$	1024	64	384	192
SHA-512	$< 2^{128}$	1024	64	512	256

The longer MD sizes are targeted at working with AES

As of late 2007 only SHA-1 was formally approved as a FIPS

RIPEMD-160

RIPEMD-160 Overview

Developed in Europe as part of RIPE project in 1996 by researchers involved in attacks on MD4/5

Initial proposal strengthen following analysis to become RIPEMD-160

Similar to MD5/SHA

But uses 2 parallel lines of 5 rounds of 16 steps

Hash length = 160 bits (same as SHA-1)

Basic unit of processing = 512 bits

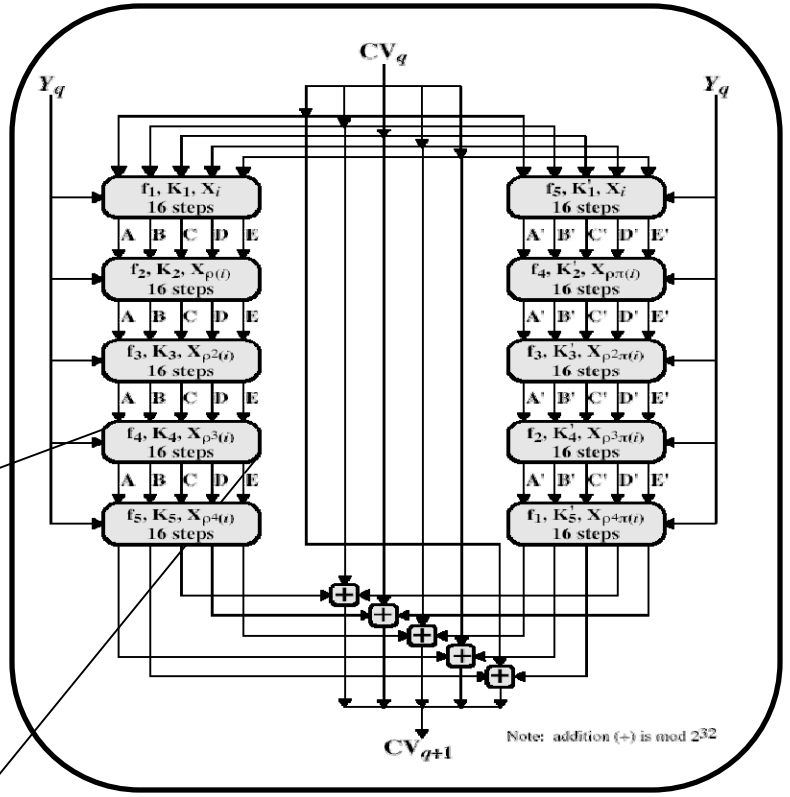
Maximum message size < 2^{64} bits (1.8 x 10^{18} bytes)

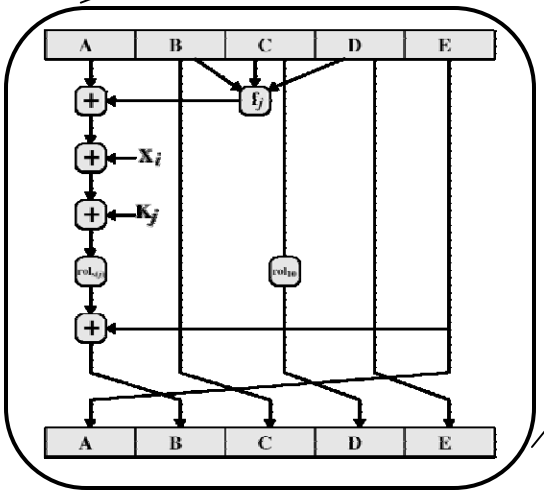
Slower but probably more secure than SHA-1

RIPEMD-160 Box

This diagram of the operation of a **RIPEMD-160** box shows the complexity of the function.

All additions are mod 2^{32}.





1. f_j executes Logical Equation 1 on B,C,D.
2. The register ABCDE then does a 32-bit right barrel shift.
   (A->B, B->C etc.)
3. f_j again executes Logical Equation 1 on B,C,D.
4. For each barrel shift ***g*** executes Logical Equation 1 for a total of **five** times

Then f_j changes to Logical Equation 2 and steps 1-4 are repeated.
f_j changes its logical equation 4 times.
$rol_{s(j)}$ is a left barrel shift of 32 bits where $s(j)$ varies from step to step

Message Authentication Codes

MAC Overview

As we mentioned earlier, while a hash function can be used by itself for complete authentication it may be possible to provide messages with duplicate hashes

$$h = H(X)$$

But a hash in combination with either symmetric or asymmetric keys can provide authentication with almost no chance of duplication

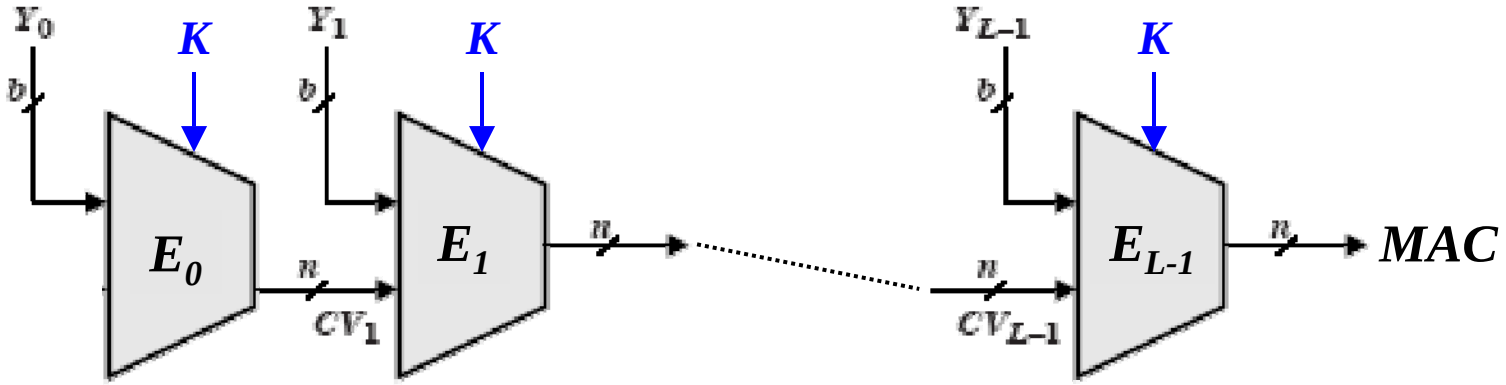
$$MAC = H_K(X)$$

Message X is divided into a number of sub blocks Yi

Each sub block is encrypted with key K and used to modify the next sub block

The output of the final sub block is the MAC

Generalized MAC Diagram



CV_i = chaining variable
Y_i = ith input sub block of data block X
E_i = block chain hashing algorithm
L = number of sub blocks in data block X
b = length of each sub block
n = length of hash h
MAC = Message Authentication Code
K = key
Note: No initial vector

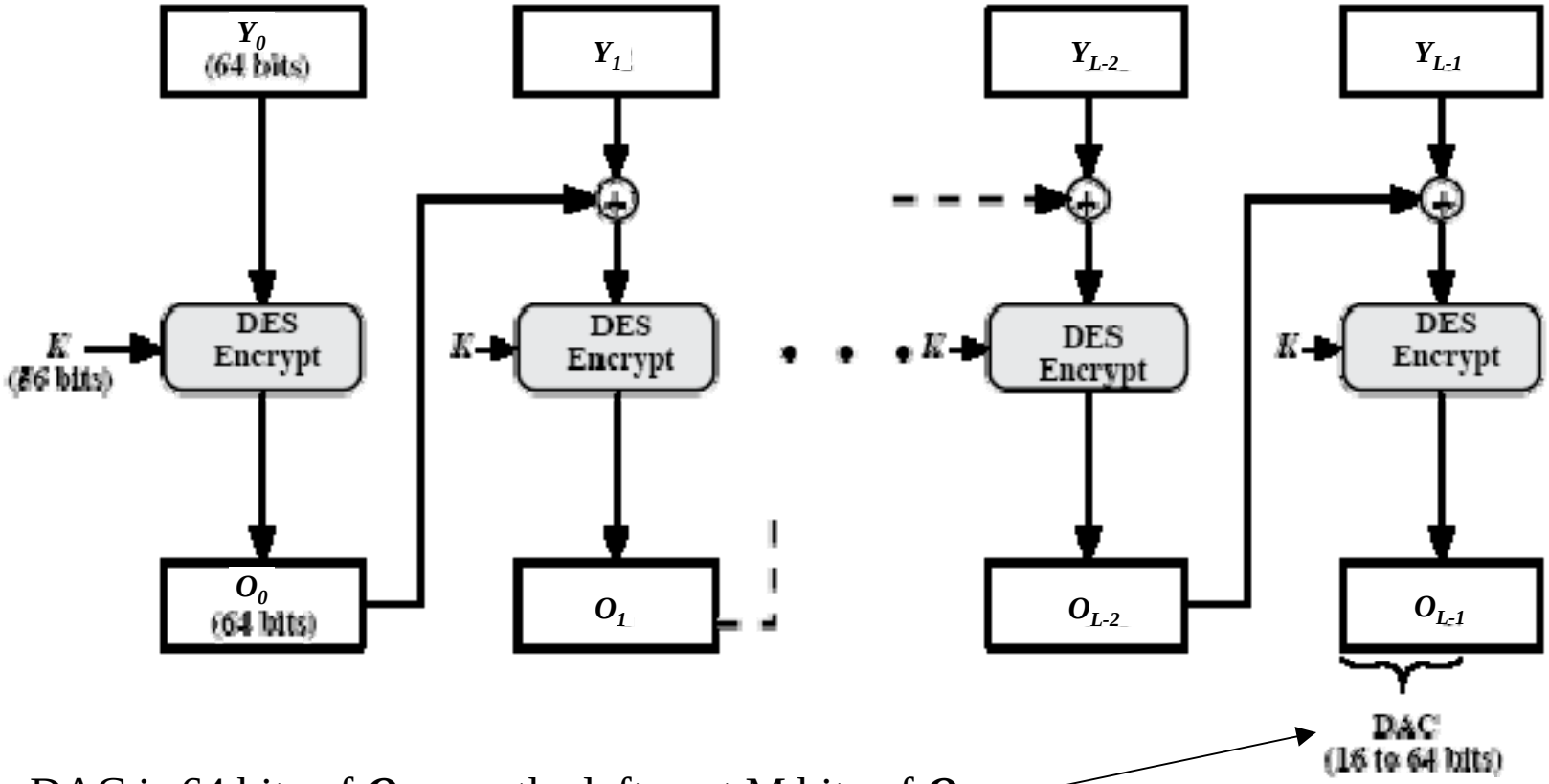
DAC Overview

A widely used MAC scheme based upon DES is called the DAC (Data Authentication Code)

Based upon the DAA (Data Authentication Algorithm)

Uses cipher block chaining mode of DES

DAC Block Diagram



DAC is 64 bits of O_{L-1} or the leftmost M bits of O_{L-1}

HMAC Overview

One MAC scheme is specifically designated “HMAC”

RFC 2104

Mandatory for IPsec

Used in SSL

HMAC Block Diagram

Hash = MD5, SHA-1…

K^+ = Symmetrical key padded left with 0s so that it is ***b*** bits long

ipad = 00110110 repeated b/8 times

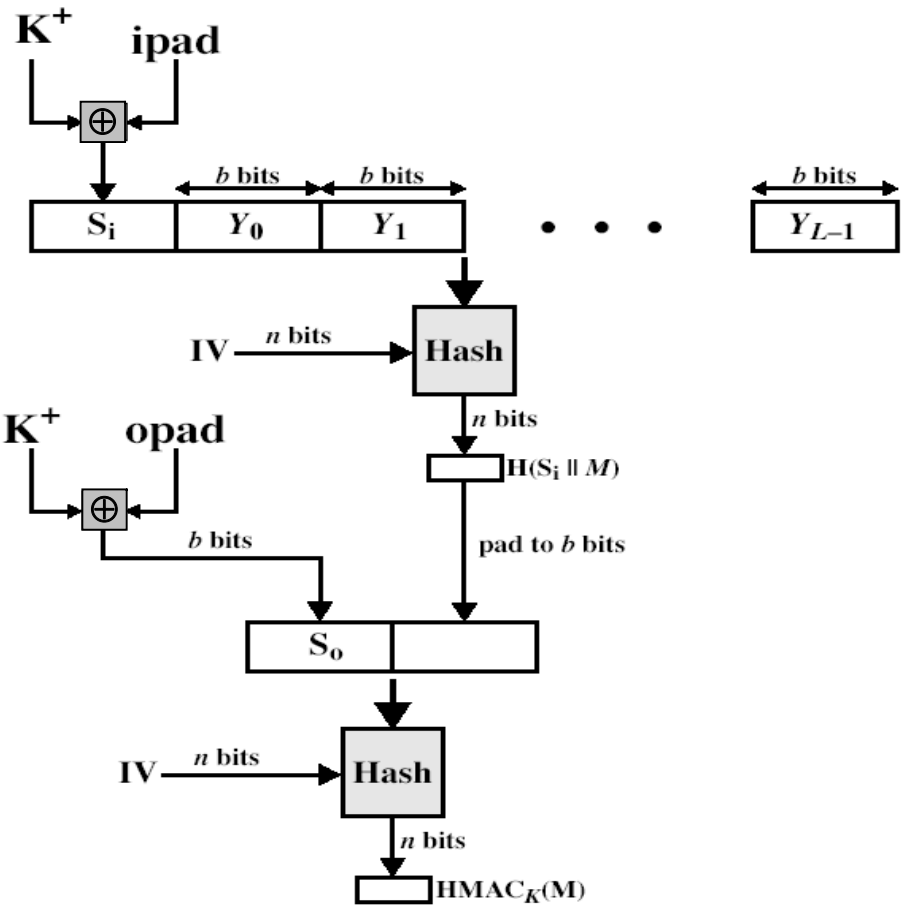
opad = 01011100 repeated b/8 times

IV = Hash function seed

$\boldsymbol{Y_i}$ = The ith block of the block ***X***

Half of the bits of K are reversed by XOR with ipad

Different half of the bits of K are reversed by XOR with opad



HMAC Security

The security of HMAC relates to that of the underlying hash algorithm

Attacking HMAC requires either:

- *Brute force attack on key used*
- *Birthday attack (but since it is keyed,Eve would need to observe a very large number of messages)*

Choose hash function based on speed verses security constraints

Hashing to Achieve Integrity

Alice **Bob**

H[P] is the hash

P, H[P] ----------> P, H[P]

Using P, calculate H[P]'

If H[P]' = H[P], P has not been changed

But Eve might send Bob a message and claim that it came from Alice

The Whole Thing

Needed to Achieve All Three Things

To achieve ***confidentiality***, ***authentication***, and ***integrity***, need double encryption plus hashing

Encryption efficiency

- *Message is likely to be much longer than hash or symmetric key*
- *For efficiency use symmetric (private key) encryption for the message*
- *Use asymmetric (public key) encryption to achieve authentication and integrity*

The Whole Enchilada*
Using Hashes

Alice

$C_1 = SE_k[P]$

$C_2 = AE_{Bpu}[k]$

$C_3 = AE_{Apr}[C_2, H[P]]$

$C_1 \| C_3$ ----------> $C_1 \| C_3$

Bob

$AD_{Apu}[C_3]$ yields **H[P] & C_2**

$AD_{Bpr}[C_2]$ yields **k**

$SD_k[C_1]$ yields **P'**

Calc **H[P'];** if H[P'] = H[P], then:

(1) P has not been changed, and

(2) Alice did send the message

(3) The message is confidential

SE: Symmetric Encryption

AE: Asymmetric encryption

The Whole Enchilada*
Using Msg. Authentication Codes

Alice

$C_1 = SE_k[P]$

$C_2 = AE_{Bpu}[k]$

$C_3 = AE_{Apr}[C_2, H_k[P]]$

$C_1 \| C_3$ ⟶ $C_1 \| C_3$

Bob

$AD_{Apu}[C_3]$ yields $H_k[P]$ & C_2

$AD_{Bpr}[C_2]$ yields k

$SD_k[C_1]$ yields P'

Calc $H_k[P']$; if $H_k[P'] = H_k[P]$, then:

(1) P has not been changed, and

(2) Alice did send the message

(3) The message is confidential

SE: Symmetric Encryption

AE: Asymmetric encryption

Information Entropy

Definition of Information Entropy

The amount of uncertainty in some pseudo random variable *(Increased uncertainty => increased entropy)*

A measure of the uncertainty about an event associated with a given probability distribution

A measure of the amount of information in a message that is based upon the logarithm of the number of possible equivalent messages

Webster's New Collegiate Dictionary

The above definition is based upon Shannon's definition of information entropy in his seminal 1948 paper, "A Mathematical Theory of Communication"

Some Intuition re Information Entropy*

Let bowl #1 contain a very large number of marbles in several different colors

You are told that there are an equal number of marbles of each color

Let bowl #2 contain a very large number of marbles in several different colors

You are told that there are more blue marbles than marbles of any other color

You have no information about the number of marbles of other colors

Without looking draw a marble from each bowl

From which bowl is the information the greatest regarding color of a marble that you drew?

Justify your answer

One Time Pads

One Time Pad Overview

Completely Secure polyalphabetic substitution cipher

- *Proved mathematically to be secure*
- *Invented 1918 by Gilbert Vernam of Bell Labs*
- *Enhanced by Joseph Mauborgne of U.S. Army Signal Corp*

What is the **key**

- *Very long string of randomly sequenced characters or bits*
- *Use as many characters or bits from the key as the message is long to both encrypt and decrypt*
  - That part of the key used only once; never used again

One Time Pad Issues

Information + Entropy = Constant

i.e., Order + Disorder = Constant

CA (cryptanalysis) is possible only because of SI (side information)

SI includes

Source information redundancy

Possible periodicity injected by the encryption process

But a random key structure introduces randomness at the same rate as the regularity of the plaintext

*It causes **diffusion** and **confusion***

Claude Shannon (Bell Laboratories)

One Time Pad Concepts

Information entropy for the CA

As encrypted message length increases, entropy decreases

=> Probability that CA can decrypt increases because information increases

Information increases because the message is not random

As encrypted message length decreases, entropy increases

=> Probability that CA can decrypt decreases because information decreases

This is sort of intuitive

One Time Pad Concepts

But suppose that the key length increases as message length increases

Then entropy is constant

=> information to the CA is constant

A key that increases in this way appears to be infinite in length

As long as you don't run out of key before you run out of message

Character-based Processing Using One Time Pads

To encrypt:

XOR one letter of the one time pad key to each plaintext letter to determine encrypted letter modulo the length of the alphabet

Do the same to decrypt using same part of the key

Basic byte-based process

Key is an apparently infinite

$c_i = p_i \oplus k_i$

where c_i *= encrypted byte* p_i *= plaintext byte*

k_i *= key byte* $\oplus$ *= Exclusive OR*

Unicity Distance & Language Redundancy

In 1940s Claude Shannon (Bell Labs) proved mathematically that one-time pads are completely secure

Shannon famous for his work in information theory

The "father" of information theory

He addressed cryptography & set forth 2 basic ideas:

*Cryptanalysis is possible **only** because of redundancy in plaintext P*

*Concept of the **Unicity Distance**, N_0*

Language Redundancy

Cryptanalysis is possible only because of redundancy in plaintext P

If there is no redundancy then there is no *Side Information* (SI)

SI consists of

Natural language frequency information, and

Periodicity introduced by the encryption process

Thus if an encrypted message is long enough and the key length short enough

Then it is possible to extract redundancy

Unicity Distance*

Ciphertexts longer than N_0 are reasonably certain to have only one meaningful plaintext

Good for the CA

Ciphertexts shorter than N_0 are reasonably certain to have multiple meaningful plaintexts

Thus making it difficult for the CA (cryptanalyst) to determine the key K

Unicity Distance*

Among other things, the unicity distance measures the amount of ciphertext required such that there is only one reasonable (i.e., meaningful) plaintext

This number depends upon two factors:

Side information regarding the characteristics of the plaintext and

The key length of the encryption algorithm

Unicity Distance

Let L_c be the length of the ciphertext and L_k the length of the key in bits

If $L_c \geq N_0$, then it becomes possible for a CA (cryptanalyst) to determine the key K

If $L_c < N_0$ then it may still be possible for a CA to determine the key K

But it's much much more difficult

$N_0 \cong L_k / R_c$

where R_c = per character language redundancy (which is dimensionally [bits/character])

e.g, after a letter another letter occurs with a certain freq., language rules…

Unicity Distance

Fortunately N_0 grows with the length of the key

If N_0 is infinite or at least very large relative to L_c, then

There are a huge number of possible and reasonable plaintexts and

As a practical matter, the CA can't decipher the message

Unicity Distance

For the English language represented in ASCII

$R_c \cong 6.8$ *bits per character (in ASCII)*

DES key = 56 bits

Therefore, $N_0 = 56/6.8 = 8.2$ *characters or about 66 bits*

Thus if L_c> 66 bits, C is theoretically vulnerable

N_0, DES, and English Text

For DES encrypting English text

*If $L_c \geq 8.2$ characters, then it **IS** possible to determine K*

Fortunately it is still hard to do

However if the key were 10,000 characters or 80, 000 bits, then

N_0 = 80,000/6.8 = 11,765 characters

It is uncertain as to whether the CA could be successful - probably not

If the key is infinite, then

$N_0 = \infty/6.8 = \infty$

And C is not decipherable

Recent One Time Pad Key Distribution Proposal

Proposed by Michael Rabin (Prof.) and Yan Zong Ding (doctorial student) of Harvard Univ.

Called "Hyper-Encryption“

Proposal Process

A huge one-time pad key is broadcast worldwide continually

- *As an infinite stream of elements (i.e., bits)*
- *Perhaps by satellites*
- *Huge never-ending stream of random bits*

Alice and Bob want to communicate securely

- *They both copy bits from the one-time pad according to some pre-arranged algorithm*
- *They now both have the same bit sequence which becomes their key*
- *Only Alice and Bob know the manner in which they selected the bits*
- *Therefore only the two of them know the key*

Proposal Process

Alice and Bob now can communicate in secret

Using any reasonable symmetric key algorithm

If they use the same key repeatedly

Effectively they are using standard symmetric key schemes

If they

Use a key that is longer than the message and

Use parts of the key progressively, discarding the used parts of the key as they are used

Then they are using a one-time pad scheme

The Proposal & The Cryptanalyst

Suppose a CA wishes to decipher Alice <--> Bob msgs.

- *Must get the key (which is gone because Alice & Bob discarded it), or*
- *Must get the algorithm for selecting the key*
  - But even if the CA gets this, it is of no use after the fact because the bit stream is gone

CA might try to copy the one-time pad broadcast bit stream

- *CA can use it later after she gets the key selection algorithm*
- *But the amount of data that the CA would need to store in infeasible - too big*

One Time Pad Conclusions*

One Time Pads are essentially private key systems with infinite keys

The key is not really infinite but it grows at the same rate as the encrypted message length, L_c

Thus it appears infinite as long as $L_k \geq L_c$

Entropy never decreases

Assign07a, slide 1 of 5

Problem 07a-1

Find the bit-by-bit XOR hash of my last name backwards in ASCII: **yksnidiL** . *Critically compare this hash with the answer to the XOR of my name spelled forward (see the example in these slides).*

Problem 07a-2

Determine a character string that yields the same XOR hash as the XOR of my last name (spelled forward).

Assign07a, slide 2 of 5

Problem 07a-3

Given: $\boldsymbol{g} = 11$, $\boldsymbol{h} = 13$, $\boldsymbol{u} = 11$.

Use RSA encryption to send the encrypted value of the plaintext number "7".

Determine:

$\mathbf{K_{pu}}$, $\mathbf{K_{pr}}$ and **C**

Then decrypt C to determine P as a check on your work.

Assign07a, slide 3 of 5

Problem 07a-3 (continued)

*Show your answers at the very end of your problem submission **exactly** as follows, substituting actual values for the "??". (If you don't do this, your homework grade will be reduced by 20%.)*

P = 7

r = ??

n = ??

K_{pr} = ??

K_{pu} = ??

C = ??

P' = ?? *(Note: P' is the value that you get after decrypting C)*

Assign07a, slide 4 of 5

You may need this equality to do the assignment:

Modulus Arithmetic Equality:

$$x^{(A+B+C)} \bmod z = \left[(x^A \bmod z)(x^B \bmod z)(x^C \bmod z)\right] \bmod z$$

Example:

$2^{10} \bmod 60 = 1024 \bmod 60 = \mathbf{4}$ *(i.e., 17 x 60 = 1020)*

$2^{10} \bmod 60 = \left[(2^7 \bmod 60)(2^2 \bmod 60)(2^1 \bmod 60)\right] \bmod 60$

$= \left[(128 \bmod 60)(4 \bmod 60)(2 \bmod 60)\right] \bmod 60$

$= \left[(8)(4)(2)\right] \bmod 60 = 64 \bmod 60 = \mathbf{4}$

Assign07a, slide 5 of 5

Read

S&B, Chapter 21

Note: S&B use somewhat different notation that these notes. For instance our "P" is S&B's "M", our "u" is S&B's "e"…

Review Questions p 677

S&B Review Questions 21.1 – 21.5

Problems p 677

Problem 21.1

Note:

Submit as a single **.doc** *document.*