

CS 6v81 - Network Security

Cryptography

Intro to cryptography

- ▶ The uses of cryptography
 - ▶ Transmitting secret data over an insecure channel
 - ▶ Storing secret data on an insecure media
 - ▶ Message integrity checksum/authentication code
 - ▶ Privacy, authentication, data integrity



- ▶ Cryptography vs cryptanalysis
 - ▶ Cryptography – involves both algorithms and secret value (key)
 - ▶ Cryptanalysis – involves algorithms, sample plain/crypto text

2

Intro to cryptography

- ▶ How secure a crypto algorithm is?
 - ▶ Not impossible to break
 - ▶ Depends on how much effort it takes to break
 - ▶ Can be made more secure by using longer keys
- ▶ Crypto algorithms – to publish or not to publish?
 - ▶ Keeping it secret helps – if others don't know it !!!
 - ▶ Publishing it helps
 - ▶ If good people breaks it, you get to know about it (free testing service)
 - ▶ Bad guys will learn the algorithm anyway
 - ▶ Today
 - ▶ most commercial algorithms are published and
 - ▶ most military algorithms are not

3

Breaking an encryption scheme

- ▶ Ciphertext only attacks
 - ▶ Try out all keys to decrypt the ciphertext
 - ▶ Need to be able to distinguish cleartext from gibberish
- ▶ Known plaintext attacks
 - ▶ Knowing a few <plaintext, ciphertext> pairs can help mapping of plaintext letters to ciphertext letters enabling decryption of messages
- ▶ Chosen plaintext attacks
 - ▶ Attacker chooses a plaintext and system tells him back the corresponding ciphertext

- ▶ A cryptosystem should protect against all of these types of attacks

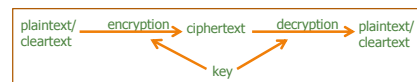
4

Classification of crypto systems

- ▶ Type of operations during transformation
 - ▶ Substitutions vs permutations
- ▶ Number of keys used
 - ▶ Symmetric / single key / secret key systems
 - ▶ Asymmetric / two-key / public key systems
- ▶ Mode of plaintext processing
 - ▶ Block cipher – one block (e.g., 64-bits) at a time
 - ▶ Stream cipher – continuous, no block definition
- ▶ Types of cryptographic functions
 - ▶ Secret key functions – use one key
 - ▶ Public key functions – use two keys
 - ▶ Hash functions – use zero or one keys

5

Secret key cryptography



- ▶ Security uses of secret key crypto
 - ▶ Transmitting over an insecure channel
 - ▶ Storing on an insecure media
 - ▶ Authentication
 - ▶ Alice and Bob shares a secret key K_{AB} to verify each other
 - ▶ Integrity check



6

Public key cryptography

- Two keys per user: **private** and **public** keys

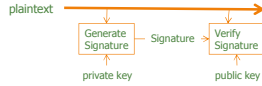
- Reverse the effect of each other

- Security uses of public key crypto

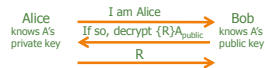
- Encryption for privacy



- Digital signature (integrity)



- Public key authentication



- Non-repudiation**

- A msg signed by Alice's private key proves that it is from Alice
- Difficult to achieve with secret key crypto

7

Hash algorithms (msg digest functions)

- A mathematical transformation from an arbitrary size message to a fixed-length (short) number

$$m \xrightarrow{\text{digest}} d(m)$$

- Easier to compute $m \rightarrow d(m)$
- Very difficult to compute $d(m) \rightarrow m$
- For $m_1 \rightarrow d(m_1)$, difficult to find m_2 s.t. $d(m_1)=d(m_2)$

- Message integrity using hash functions



8

Combining crypto funct's for performance

- Public key crypto too slow compared to hashes and secret key crypto
- Public key crypto more convenient and secure in setting up keys
- Algorithms can be combined to improve performance
 - Hybrid encryption



- Hybrid signature



- Signed and encrypted message – how to do it???

9

Some basic terminology

- Plaintext:** original message
- Ciphertext:** coded message
- Cipher:** algorithm for transforming plaintext to ciphertext
- Key:** info used in cipher known only to sender/receiver
- Encipher (encrypt):** converting plaintext to ciphertext
- Decipher (decrypt):** recovering ciphertext from plaintext
- Cryptography:** study of encryption principles/methods
- Cryptanalysis (codebreaking):** study of principles/ methods of deciphering ciphertext *without* knowing key
- Cryptology:** field of both cryptography and cryptanalysis

10

Cryptography

- Characterize cryptographic system by
 - Type of encryption operations used
 - Substitution / transposition / product
 - Number of keys used
 - Single-key or private / two-key or public
 - Way in which plaintext is processed
 - Block / stream

11

Cryptanalysis

- Objective is to recover key not just message
- General approaches
 - Cryptanalytic attack
 - Brute-force attack

12

More definitions

- ▶ **Unconditional security**
 - ▶ no matter how much computer power or time is available, the cipher cannot be broken since the ciphertext provides insufficient information to uniquely determine the corresponding plaintext
- ▶ **Computational security**
 - ▶ given limited computing resources (eg time needed for calculations is greater than age of universe), the cipher cannot be broken

13

Brute force search

- ▶ Always possible to simply try every key
- ▶ Most basic attack, proportional to key size
- ▶ Assume either know / recognise plaintext

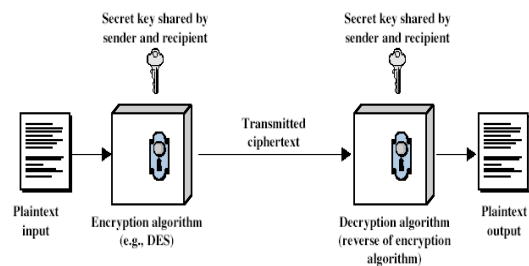
Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/ μ s	Time required at 10^6 decryptions/ μ s
32	$2^{32} = 4.3 \times 10^9$	$2^{31} \mu$ s = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55} \mu$ s = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127} \mu$ s = 5.4×10^{23} years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167} \mu$ s = 5.9×10^{36} years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu$ s = 6.4×10^{12} years	6.4×10^6 years

14

Secret (Symmetric) Key Crypto

15

Secret (symmetric) key crypto



16

Requirements

- ▶ Two requirements for secure use of symmetric encryption:
 - ▶ A strong encryption algorithm
 - ▶ A secret key known only to sender / receiver
- ▶ Mathematically have:
 - $Y = E_K(X)$
 - $X = D_K(Y)$
- ▶ Assume encryption algorithm is known (by all)
- ▶ Implies a secure channel to distribute key

17

Classical substitution ciphers

- ▶ Where letters of plaintext are replaced by other letters or by numbers or symbols
- ▶ Or, if plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns

18

Caesar cipher

- ▶ Earliest known substitution cipher
- ▶ By Julius Caesar
- ▶ First attested use in military affairs
- ▶ Replaces each letter by 3rd letter on
- ▶ Example:
meet me after the toga party
PHHW PH DIWHU WKH WRJD SDUWB

Caesar cipher

- ▶ Can define transformation as:
a b c d e f g h i j k l m n o p q r s t u v w x y z
D E F G H I J K L M N O P Q R S T U V W X Y Z A B C
- ▶ Mathematically give each letter a number
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
- ▶ Then have Caesar cipher as:
 $c = E(p) = (p + k) \bmod (26)$
 $p = D(c) = (c - k) \bmod (26)$

Cryptanalysis of Caesar cipher

- ▶ Only have 26 possible ciphers
 - ▶ A maps to A, B, ..., Z
- ▶ Could simply try each in turn
- ▶ A brute force search
- ▶ Given ciphertext, just try all shifts of letters
- ▶ Do *need to recognize when have plaintext*
- ▶ E.g., break ciphertext "GCUA VQ DTGCM"

Monoalphabetic cipher

- ▶ Rather than just shifting the alphabet
 - ▶ Could shuffle (jumble) the letters arbitrarily
 - ▶ Each plaintext letter maps to a different random ciphertext letter
 - ▶ Hence key is 26 letters long
- Plain: abcdefghijklmnopqrstuvwxyz
Cipher: DKVQFIBJWPESCXHTMYAUOLRGZN
- Plaintext: ifwewishtoreplaceletters
Ciphertext: WIRFRWAJUHYFTSDVFSFUUFYA

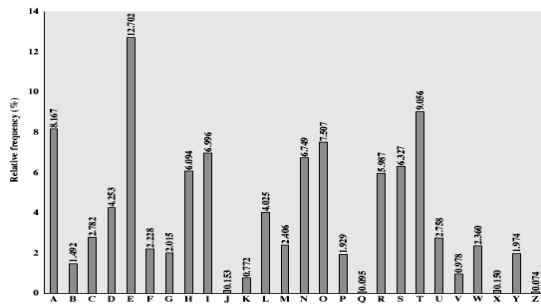
Monoalphabetic cipher security

- ▶ Now have a total of $26! = 4 \times 10^{26}$ keys
- ▶ With so many keys, might think is secure
- ▶ But would be **!!!WRONG!!!**
- ▶ Problem is...
 - ▶ language characteristics

Language redundancy and cryptanalysis

- ▶ Letters in an alphabet are not equally commonly used
- ▶ In English, E is by far the most common letter
 - ▶ Followed by T, R, N, I, O, A, S
- ▶ Other letters like Z, J, K, Q, X are fairly rare
- ▶ Have tables of single, double & triple letter frequencies for various languages

English letter frequencies



25

Use in cryptanalysis

- ▶ **Key concept:**
 - ▶ monoalphabetic substitution ciphers do not change relative letter frequencies
- ▶ Discovered by Arabian scientists in 9th century
- ▶ Calculate letter frequencies for ciphertext
- ▶ Compare counts/plots against known values
- ▶ If Caesar cipher, look for common peaks/troughs
 - ▶ Peaks at: A-E-I triple, NO pair, RST triple
 - ▶ Troughs at: JK, X-Z
- ▶ For monoalphabetic, must identify each letter
 - ▶ Tables of common double/triple letters help

26

Example cryptanalysis

- ▶ **Given ciphertext:**
 UZQSOVUOHXMOPVPGPOZPEVSGZWSZOPFPESXUDEMETSXAI Z
 VUEPHZHMDSHZOWSFPAPDTSVPQZWMYXUZHSX
 EPYEPDPDZSZUPPOMBZWPFPUPZHMDJUDTMOHMQ
- ▶ Count relative letter frequencies
- ▶ Guess P & Z are e and t
- ▶ Guess ZW is th and hence ZWP is the
- ▶ Proceeding with trial and error finally get:
 it was disclosed yesterday that several informal but
 direct contacts have been made with political
 representatives of the viet cong in moscow

29

Monoalphabetic ciphers

- ▶ Several important monoalphabetic ciphers introduced and used in the past
 - ▶ Fairplay cipher (1854) (used in WW I and WW II)
 - ▶ Vigenere cipher
 - ▶ Autokey cipher
- ▶ All are vulnerable due to possibility of using frequency characteristics of letters/words in natural languages

28

One-time pad

- ▶ If a truly random key as long as the message is used, the cipher will be secure
- ▶ Called a **one-time pad**
- ▶ Is unbreakable
 - ▶ Ciphertext bears no statistical relationship to the plaintext
- ▶ Problems in generation & safe distribution of key

29

Transposition Ciphers

- ▶ Now consider classical **transposition** or **permutation** ciphers
- ▶ These hide the message by rearranging the letter order
- ▶ Without altering the actual letters used

30

Rail Fence cipher

- Write message letters out diagonally over a number of rows
- Then read off cipher row by row
- E.g., write message out as:


```
m e m a t r h t g p r y
e t e f e t e o a a t
```
- Giving ciphertext


```
MEMATRHTGPRYETEFETEOAAT
```

31

Product Ciphers

- Ciphers using substitutions or transpositions are not secure because of language characteristics
- Hence consider using several ciphers in succession to make harder, but:
 - Two substitutions make a more complex substitution
 - Two transpositions make more complex transposition
 - But a substitution followed by a transposition makes a new much harder cipher
- This is bridge from classical to modern ciphers

32

Modern Secret Key Crypto

33

Modern secret key crypto techniques

- Block vs. stream ciphers**
 - Block ciphers:** process messages in blocks, each of which is then en/decrypted like a substitution on very big characters
 - 64-bits or more
 - Stream ciphers:** process messages a bit or byte at a time when en/decrypting
- Many current ciphers are block ciphers
- Broader range of applications

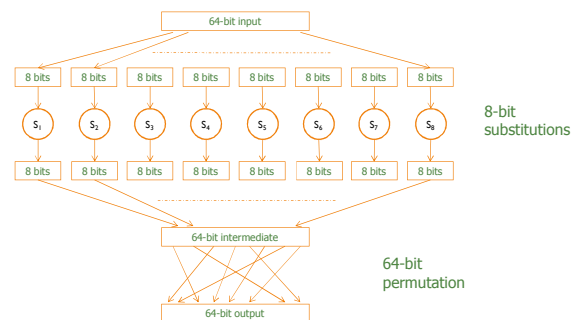
34

Block cipher encryption algorithms

- Key size**
 - If too short, then easy to guess
- Block size**
 - If too short, easy to build <plaintext, ciphertext> table by attacker
 - Reasonable size: 64-bit blocks
- Provides one-to-one mapping that looks like random w/o knowing the key
- Implemented using**
 - Substitutions** – specifies for each of 2^k possible input, k-bit output
 - Takes about $k \cdot 2^k$ bits
 - Permutations** – specifies, for each of k input bit, the output bit it maps to
 - Takes about $k \cdot \log_2 k$ bits

35

A sample block encryption



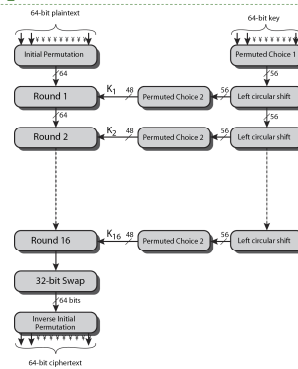
36

Data Encryption Standard (DES)

- ▶ Most widely used block cipher in the world
- ▶ Adopted in 1977 by NBS (now NIST)
 - ▶ As FIPS PUB 46
- ▶ Encrypts 64-bit data using 56-bit key
- ▶ Has widespread use
- ▶ Has been considerable controversy over its security

37

DES encryption overview



38

Avalanche effect

- ▶ Key desirable property of an encryption algorithm
- ▶ Where a change of **one** input or key bit results in changing approx **half** output bits
- ▶ Making attempts to “home-in” by guessing keys impossible
- ▶ DES exhibits strong avalanche

39

DES controversies

- ▶ Design process was not made public
 - ▶ Any hidden trapdoors?
- ▶ 56-bit keys too short
 - ▶ Is it so that NSA can break it?
- ▶ Designed for hardware and slow in software
 - ▶ Independently developed, fast-in-software cipher needed (in late 1980s)

40

Strength of DES – Key size

- ▶ 56-bit keys have $2^{56} = 7.2 \times 10^{16}$ values
- ▶ Brute force search looks hard
- ▶ Recent advances have shown that it is possible
 - ▶ In 1997 on Internet in a few months
 - ▶ In 1998 on dedicated h/w (EFF) in a few days
 - ▶ In 1999 above combined in 22hrs!
- ▶ Still must be able to recognize plaintext
- ▶ Must now consider alternatives to DES

41

AES (Advanced Encryption Standard)

- ▶ Clear a replacement for DES was needed
 - ▶ Have theoretical attacks that can break it
 - ▶ Have demonstrated exhaustive key search attacks
- ▶ Can use Triple-DES – but slow, has small blocks
- ▶ US NIST issued call for ciphers in 1997
- ▶ 15 candidates accepted in Jun 98
- ▶ 5 were shortlisted in Aug-99
- ▶ Rijndael was selected as the AES in Oct-2000
- ▶ Issued as FIPS PUB 197 standard in Nov-2001

42

AES requirements

- ▶ Secret key symmetric block cipher
- ▶ 128-bit data, 128/192/256-bit keys
- ▶ Stronger & faster than Triple-DES
- ▶ Provide full specification & design details
- ▶ Both C & Java implementations
- ▶ NIST have released all submissions & unclassified analyses

43

The AES cipher - Rijndael

- ▶ Designed by Rijmen-Daemen in Belgium
- ▶ Has a variety of block and key sizes
 - ▶ 128/160/192/224/256 bit keys and data
- ▶ An **iterative** rather than **feistel** cipher
 - ▶ Processes data as block of 4 columns of 4 bytes
 - ▶ Operates on entire data block in every round
- ▶ Designed to be:
 - ▶ Resistant against known attacks
 - ▶ Speed and code compactness on many CPUs
 - ▶ Design simplicity

44

Multiple Encryption & DES

- ▶ Clear a replacement for DES was needed
 - ▶ Theoretical attacks that can break it
 - ▶ Demonstrated exhaustive key search attacks
- ▶ AES is a new cipher alternative
- ▶ Prior to AES, the alternative was to use multiple encryption with DES implementations
- ▶ Triple-DES is the chosen form

45

Double-DES?

- ▶ Could use 2 DES encrypts on each block
 - ▶ $C = E_{k_2}(E_{k_1}(P))$
- ▶ Issue of reduction to single stage
- ▶ And have “meet-in-the-middle” attack
 - ▶ Works whenever use a cipher twice
 - ▶ Since $X = E_{k_1}(P) = D_{k_2}(C)$
 - ▶ Attack by encrypting P with all keys and store
 - ▶ Then decrypt C with keys and match X value
 - ▶ Can show takes $O(2^{56})$ steps
 - ▶ A similar attack on DES takes 2^{55} **steps only**

46

Triple-DES with two-keys

- ▶ Hence must use 3 encryptions
 - ▶ Would seem to need 3 distinct keys
- ▶ But can use 2 keys with E-D-E sequence
 - ▶ $C = E_{k_1}(D_{k_2}(E_{k_1}(P)))$
 - ▶ Number of encrypt & decrypt equivalent in security
 - ▶ Three times slower than DES
- ▶ Standardized in ANSI X9.17 & ISO8732
- ▶ No current known practical attacks

47

Triple-DES with three-keys

- ▶ Although there are no practical attacks on two-key Triple-DES, there are some indications
- ▶ Can use Triple-DES with Three-Keys to avoid even these
 - ▶ $C = E_{k_3}(D_{k_2}(E_{k_1}(P)))$
- ▶ Has been adopted by some Internet applications, e.g., PGP, S/MIME

48

Modes of Operation

49

Modes of Operation

How to encrypt messages larger than 64-bits?

- ▶ Block ciphers encrypt fixed size blocks
 - ▶ E.g., DES encrypts 64-bit blocks with 56-bit key
- ▶ Need some way to en/decrypt arbitrary amounts of data in practise
- ▶ ANSI X3.106-1983 Modes of Use (now FIPS 81) defines 4 possible modes
- ▶ Subsequently 5 defined for AES & DES
- ▶ Have **block** and **stream** modes

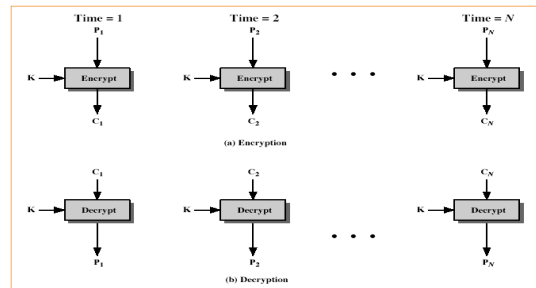
50

Electronic Codebook Book (ECB)

- ▶ Message is broken into independent blocks which are encrypted
 - ▶ Each block is a value which is substituted, like a codebook, hence the name
 - ▶ Each block is encoded independently of the other blocks
- $$C_i = \text{DES}_{K1}(P_i)$$
- ▶ Uses: secure transmission of single values

51

Electronic Codebook Book (ECB)



52

Advantages and Limitations of ECB

- ▶ Propagation error in a ciphertext block does not affect the other ciphertext blocks
- ▶ Can modify a block of ciphertext stored in a disk without needing to modify all other blocks
- ▶ Message repetitions may show in ciphertext
 - ▶ If aligned with message block
 - ▶ Particularly with data such as graphics
 - ▶ Or with messages that change very little, which become a code-book analysis problem
- ▶ Weakness is due to the encrypted message blocks being independent
- ▶ Main use is sending a few blocks of data

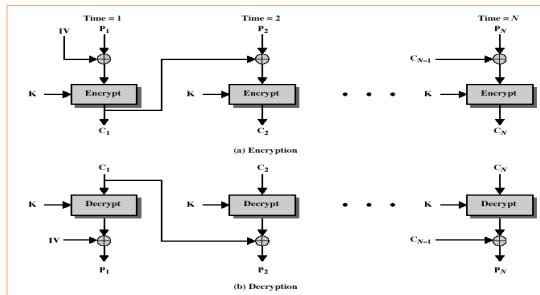
53

Cipher Block Chaining (CBC)

- ▶ Message is broken into blocks
 - ▶ Linked together in encryption operation
 - ▶ Each previous cipher block is chained with current plaintext block, hence the name
 - ▶ Use Initial Vector (IV) to start process
- $$C_i = \text{DES}_{K1}(P_i \text{ XOR } C_{i-1})$$
- $$C_{-1} = \text{IV}$$
- ▶ Uses: bulk data encryption, authentication

54

Cipher Block Chaining (CBC)



55

Advantages and Limitations of CBC

- ▶ A ciphertext block depends on **all** blocks before it
 - ▶ If you modify a plaintext block, you need to re-do CBC on the rest as it causes change to all following
- ▶ **Any change to a ciphertext during the transit affects the corresponding plaintext and the next one at the receiver**
- ▶ Need **Initialization Vector (IV)**
 - ▶ Which must be known to sender & receiver
 - ▶ If sent in clear, attacker can change bits of first block, and change IV to compensate
 - ▶ Hence IV must either be a fixed value **OR** must be sent encrypted in ECB mode before rest of message

56

An attack on CBC: Modify ciphertext

- ▶ Changing a ciphertext block C_i to C'_i would cause a change in P_{i+1} which may be desired by the attacker
 - ▶ But would also cause P_i to map to a random 64-bit string where attacker have no control over - $D(C'_i) \text{ XOR } C_{i-1}$ may be anything
- ▶ Append a 64-bit CRC to the message to detect modify attacks
 - ▶ Decrypted message should pass CRC check to verify tamper proof message content

57

Another CBC attack: Rearrange ciphertext blocks

- ▶ Assume you know $P_1, P_2, \dots, P_n, C_1, C_2, \dots, C_n, IV$
 - ▶ Using this, can find what C_i decrypts to ($C_{i-1} \text{ XOR } P_i$)
- ▶ Now can construct a ciphertext stream using any combination of C_i and would know the corresponding plaintext
- ▶ New message should pass the CRC test if 32-bit CRC used, need an average of 2^{31} ciphertext block arrangements to build a message with proper CRC
 - ▶ Changing one part of the message to your benefit would result in many other changes in other parts in order to pass CRC text
 - ▶ Using a 64-bit CRC makes it more difficult

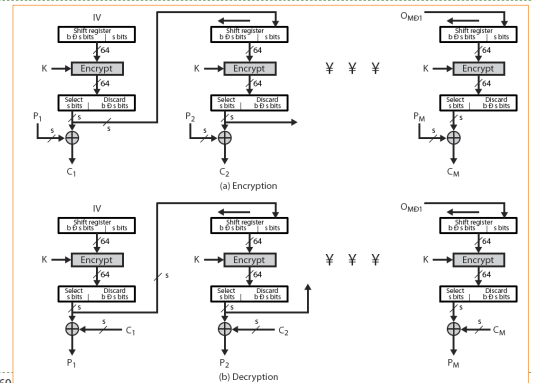
58

Output FeedBack (OFB) – stream cipher

- ▶ Message is treated as a **stream of bits**
- ▶ Output of cipher is added to message
- ▶ Output is then feed back (hence name)
- ▶ Feedback is independent of message
- ▶ Can be computed in advance
 - $C_i = P_i \text{ XOR } O_i$
 - $O_i = \text{DES}_{K1}(O_{i-1})$
 - $O_{-1} = IV$
- ▶ Uses: stream encryption on noisy channels

59

Output FeedBack (OFB)



60

Advantages and limitations of OFB

- ▶ Advantages
 - ▶ One-time pad can be generated in advance
 - ▶ Bit errors do not propagate
- ▶ Disadvantages
 - ▶ More vulnerable to message stream modification
 - ▶ If attacker knows both P and C, can change message to anything
 - ▶ Sender & receiver must remain in sync
 - ▶ If a message is lost, rest is garbled

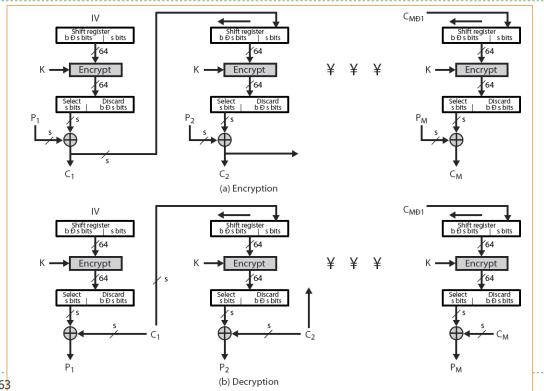
61

Cipher FeedBack (CFB)

- ▶ Message is treated as a stream of bits
- ▶ Added to the output of the block cipher
- ▶ Result is feed back for next stage (hence name)
- ▶ Standard allows any number of bit (1,8, 64 or 128 etc) to be feed back
 - ▶ Denoted CFB-1, CFB-8, CFB-64, CFB-128 etc
- ▶ Most efficient to use all bits in block (64 or 128)
 - $C_i = P_i \text{ XOR } \text{DES}_{K1}(C_{i-1})$
 - $C_{-1} = \text{IV}$
- ▶ Uses: stream data encryption, authentication

62

Cipher FeedBack (CFB)



63

Advantages and Limitations of CFB

- ▶ Appropriate when data arrives in bits/bytes
- ▶ Most common stream mode
- ▶ Limitation is – need to stall while do block encryption after every n-bits
- ▶ Note that the block cipher is used in **encryption** mode at **both** ends
- ▶ Errors propagate for several blocks after the error
 - ▶ Work on an example to observe this...
- ▶ More resistant to msg loss or corruption compared to CBC and OFB
 - ▶ In CBC or OFB, the rest will be corrupted
 - ▶ In CFB, will synchronize after some data corruption

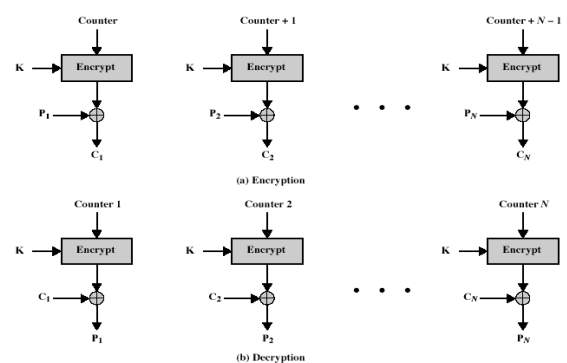
64

Counter (CTR)

- ▶ A “new” mode, though proposed early on
- ▶ Similar to OFB but encrypts counter value rather than any feedback value
- ▶ Must have a different key & counter value for every plaintext block (never reused)
 - $C_i = P_i \text{ XOR } O_i$
 - $O_i = \text{DES}_{K1}(i)$
- ▶ Uses: high-speed network encryptions

65

Counter (CTR)



66

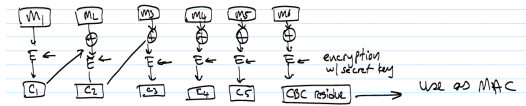
Advantages and limitations of CTR

- ▶ Efficiency
 - ▶ Can do parallel encryptions in h/w or s/w
 - ▶ Can preprocess in advance of need
 - ▶ Good for bursty high speed links
- ▶ Random access to encrypted data blocks
- ▶ Provable security (good as other modes)
- ▶ But must ensure never reuse key/counter values, otherwise could break (cf OFB)

67

Generating Message Auth. Codes (MACs)

- ▶ Message authentication codes (MACs) help protect integrity of a message
- ▶ If we need integrity only and no privacy, can use CBC as follows



- ▶ Send CBC residue along with plaintext

68

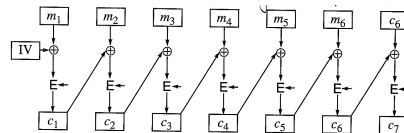
Ensuring Both Privacy & Integrity

- ▶ CBC residue along with ciphertext msg would not work
 - ▶ You could tamper w/ ciphertext msg and send the modified last cipher block as residue
- ▶ Encryption by itself cannot provide integrity
 - ▶ If the receiver side is a computer writing the content into a disk, tampered ciphertext would go undetected
- ▶ $E_{CBC}\{\text{Plaintext} \mid \text{CRC}\}$ would almost work but short CRC is a problem
- ▶ Encrypt plaintext
 - ▶ w/ K_1 for CBC for privacy
 - ▶ w/ K_2 for CBC residue for integrity
 - ▶ Double the work but it is the best we have

69

Ensuring Both Privacy & Integrity

- ▶ The approach of using last cipher and encrypting it again would not work as it gives a zero input to the encryption algorithm no matter what the ciphertext (c_6) is
 - ▶ Once capture such c_7 from a previous transmission, can use it to fake integrity



70

Stream Ciphers

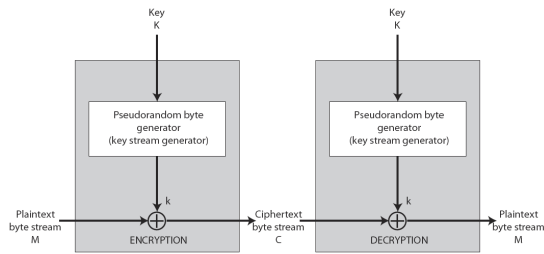
71

Stream Ciphers

- ▶ Process input one bit/byte at a time instead of in blocks
 - ▶ Good for streaming applications
- ▶ Have a *pseudo random keystream*
 - ▶ Combined (XOR) with plaintext bit by bit
- ▶ Randomness of *stream key* completely destroys statistical properties in message
 - ▶ $C_i = M_i \text{ XOR } \text{StreamKey}_i$
- ▶ But must never reuse stream key
 - ▶ Otherwise, can recover messages

72

Stream Cipher Structure



73

Stream Cipher Properties

- ▶ Some design considerations are
 - ▶ Long period with no repetitions
 - ▶ Statistically random
 - ▶ Depends on large enough key
 - ▶ Large linear complexity
- ▶ Are faster than block cipher and use far less code
- ▶ Requires that keys not be repeated as otherwise cryptanalysis would be easy
- ▶ RC4 – a simple byte oriented and fast (in s/w) cipher
 - ▶ Used in SSL/TLS, WEP, WPA

74

Hashes and Message Digests

75

Hashes and message digests

- ▶ Requirements for hash functions
 1. Can be applied to any sized message M
 2. Produces fixed-length output h ($h \ll M$)
 3. Is easy to compute $h=H(M)$ for any message M
 4. Given h , is infeasible to find x s.t. $H(x)=h$
 - One-way property
 5. Given x , is infeasible to find y s.t. $H(y)=H(x)$
 - Weak collision resistance
 6. Is infeasible to find any x, y s.t. $H(y)=H(x)$
 - Strong collision resistance
- ▶ Desirable properties
 - ▶ Given inputs M_1, \dots, M_{1000} , outputs h_1, \dots, h_{1000} look all random
 - ▶ For consecutive M_1, M_2 , outputs h_1, h_2 look random

76

Hashes and message digests

- ▶ Given a msg M_1 with m -bit digest, it takes $2^{m/2}$ randomly chosen msgs to find M_2 such that $H(M_2)=H(M_1)$
- ▶ Given m -bit msg digests, it takes $2^{m/2}$ randomly chosen msgs to find two msgs with the same digest
 - ▶ $m=128$ requires 2^{64} randomly chosen msgs \rightarrow difficult
- ▶ **Birthday Problem**
 - ▶ Given n inputs and k possible outputs
 - ▶ How many input sample needed s.t. any two input sample maps to the same output w/ a probability of 50%?
 - ▶ With n input, we have $n(n-1)/2$ tuples
 - ▶ Prob. that both input values in a tuple map to the same o/p is $1/k$
 - ▶ For 50% chance, need $k/2$ pairs $[n(n-1)/2 > k/2]$
 - Hence, n should be no less than \sqrt{k}

77

Birthday Attacks

- ▶ Might think a 64-bit hash is secure
 - ▶ but by **Birthday Paradox**, it is not
- ▶ **Birthday attack** works thus:
 - ▶ Opponent generates $2^{m/2}$ variations of a valid message all with essentially the same meaning
 - ▶ Opponent also generates $2^{m/2}$ variations of a desired fraudulent message
 - ▶ Two sets of messages are compared to find pair with same hash (probability > 0.5 by birthday paradox)
 - ▶ Have user sign the valid message hash, then substitute the forgery which will have a valid signature
- ▶ Conclusion is that need to use longer MAC/hash

78

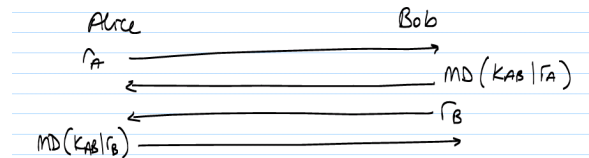
Use of crypto hash functions

- ▶ Computing a MAC w/ a hash
 - ▶ MD: message digest or a hash
 - ▶ Given a msg m , $MD(m) \rightarrow$ a MAC ???
 - ▶ No security value in $MD(m)$ as a MAC! Why?
 - ▶ Would $MD(K_{AB} || m)$ work ?
 - ▶ Alice creates m , uses K_{AB} to create $MD(K_{AB} || m)$ and sends $m, MD(K_{AB} || m)$ to Bob
 - ▶ Carol takes $m, MD(K_{AB} || m)$, changes $m \rightarrow m'$ and computes $MD(K_{AB} || m')$ starting MD with $MD(K_{AB} || m)$
- ▶ $MD(m || K_{AB})$ works
- ▶ Using half the bits in MD as MAC works, too
- ▶ $MD(K_{AB} || m || K_{AB})$ works too
- ▶ HMAC: $MD_K(x) = H(K_1 || H(m || K_2))$

79

Use of crypto hash functions

- ▶ Typically used with a secret key
 - ▶ How are they different from secret key crypto algrs?
 - ▶ Reversible vs irreversible
- ▶ Authentication



80

Use of crypto hash functions

- ▶ Encryption w/ a msg digest
 - ▶ Generate and use one-time pass to XOR with data
 - ▶ Start with $MD(K_{AB} || IV) \rightarrow b_1$: first block of bit stream
 - $MD(K_{AB} || b_1) \rightarrow b_2$
 - $MD(K_{AB} || b_{i-1}) \rightarrow b_i$
- XOR plaintext with b bit string
- ▶ **Problem:** If you correctly guess plaintext p , you can get the entire one-time pass string $s = p \text{ XOR } c$
 - ▶ Then can use s to send any message

81

Use of crypto hash functions

- ▶ Mixing in with plaintext
 - ▶ Tie one time pass string to individual msgs
 - ▶ Break msg p into MD-length chunks p_1, p_2, \dots
 - ▶ corr. ciphertext blocks will be c_1, c_2, \dots
 - ▶ each MD will use one-time pass block b_1, b_2, \dots
 - ▶ compute cipherblock as

$$\begin{aligned}
 b_1 &= MD(K_{AB} || IV) & c_1 &= p_1 \oplus b_1 \\
 b_2 &= MD(K_{AB} || c_1) & c_2 &= p_2 \oplus b_2 \\
 &\vdots & & \vdots \\
 b_i &= MD(K_{AB} || c_{i-1}) & c_i &= p_i \oplus b_i
 \end{aligned}$$

- ▶ Even if you can guess the entire plaintext, one-time pass cannot be used to send new msgs as it is computed for this particular msg only.

82

Popular hash functions

- ▶ MD2, MD4, and MD5 used to be popular
- ▶ SHA-1 taking over, HMAC also popular
- ▶ All produce 128-bit digests (SHA is 160-bits)
- ▶ MD2, MD4 broken, MD5 has vulnerabilities
- ▶ SHA-1 proposed by US government
 - ▶ produces 160, 256, 384, 512 bit digests

83

Public Key Crypto

84

Public-Key Cryptography

- ▶ Probably most significant advance in the 3000 year history of cryptography
- ▶ Uses **two** keys – a public & a private key
- ▶ **Asymmetric** since parties are **not** equal
- ▶ Uses clever application of number theoretic concepts to function
- ▶ Complements **rather than** replaces private key crypto

85

Why Public-Key Cryptography?

- ▶ Developed to address two main issues:
 - ▶ **Key distribution** – how to have secure communications in general without having to trust a KDC with your key
 - ▶ **Digital signatures** – how to verify a message comes intact from the claimed sender
- ▶ Public invention by Whitfield Diffie & Martin Hellman at Stanford Univ. in 1976
 - ▶ Known earlier in classified community

86

Private-Key Cryptography

- ▶ Traditional **private/secret/single key** cryptography uses **one** key
- ▶ Shared by both sender and receiver
- ▶ If this key is disclosed communications are compromised
- ▶ Also is **symmetric**, parties are equal
- ▶ Hence does not protect sender from receiver forging a message & claiming is sent by sender

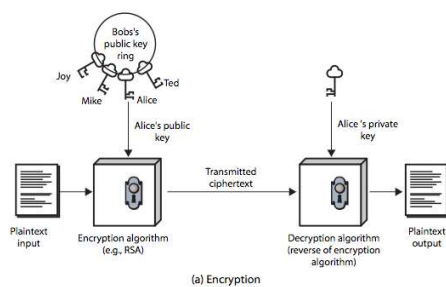
87

Public-Key Cryptography

- ▶ **Public-key/two-key/asymmetric** cryptography involves the use of **two** keys:
 - ▶ A **public-key**, which may be known by anybody, and can be used to **encrypt messages**, and **verify signatures**
 - ▶ A **private-key**, known only to the recipient, used to **decrypt messages**, and **sign (create) signatures**
- ▶ Is **asymmetric** because
 - ▶ Those who encrypt messages or verify signatures **cannot** decrypt messages or create signatures

88

Public-Key Cryptography



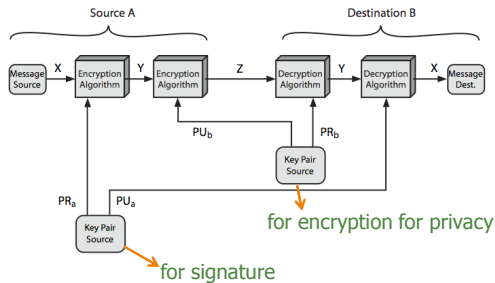
89

Public-Key Characteristics

- ▶ Public-key algorithms rely on two keys where:
 - ▶ It is computationally infeasible to find decryption key knowing only algorithm & encryption key
 - ▶ It is computationally easy to en/decrypt messages when the relevant (en/decrypt) key is known
 - ▶ Either of the two related keys can be used for encryption, with the other used for decryption (for some algorithms)

90

Public-Key Cryptosystems



91

Public-Key Applications

- ▶ Can classify uses into 3 categories:
 - ▶ Encryption/decryption (provide secrecy)
 - ▶ Digital signatures (provide authentication)
 - ▶ Key exchange (of session keys)
- ▶ Some algorithms are suitable for all uses, others are specific to one

92

Security of Public Key Schemes

- ▶ Like private key schemes brute force exhaustive search attack is always theoretically possible
 - ▶ But keys used are too large (>512 bits)
- ▶ Security relies on a large enough difference in difficulty between easy (en/decrypt) and hard (cryptanalysis) problems
 - ▶ More generally the hard problem is known, but is made hard enough to be impractical to break
- ▶ Requires the use of very large numbers
- ▶ Hence is slow compared to private key schemes

93

RSA

- ▶ By Rivest, Shamir & Adleman of MIT in 1977
- ▶ Best known & widely used public-key scheme
- ▶ Based on exponentiation in a finite (Galois) field over integers modulo a prime
 - ▶ Number exponentiation takes $O((\log n)^3)$ operations (easy)
- ▶ Uses large integers (eg. 1024 bits)
- ▶ Security due to cost of factoring large numbers
 - ▶ Number factorization takes $O(e^{\log n \log \log n})$ operations (hard)

94

RSA Key Setup

- ▶ Each user generates a public/private key pair by:
 - ▶ Selecting two large primes at random - p, q
 - ▶ Computing their system modulus $n=p \cdot q$
 - ▶ Define $\phi(n)=(p-1)(q-1)$ - called Totient function
 - ▶ $\phi(n)$: # of numbers $< n$ and relatively prime to n
 - ▶ Selecting at random the encryption key e
 - ▶ where $1 < e < \phi(n)$, $\gcd(e, \phi(n))=1$
 - ▶ Solve following equation to find decryption key d
 - ▶ $e \cdot d = 1 \pmod{\phi(n)}$ and $0 \leq d \leq n$
- ▶ Publish their public encryption key: $PU=\{e, n\}$
- ▶ Keep secret private decryption key: $PR=\{d, n\}$

95

RSA Use

- ▶ To encrypt a message m the sender:
 - ▶ Obtains public key of recipient $PU=\{e, n\}$
 - ▶ Computes: $C = M^e \pmod n$, where $0 \leq M < n$
- ▶ To decrypt the ciphertext C , the owner:
 - ▶ Uses his/her private key $PR=\{d, n\}$
 - ▶ Computes: $M = C^d \pmod n$
- ▶ Note that the message M must be smaller than n (block if needed)

96

Why RSA Works

- Arithmetic is mod n ; $n=p \cdot q$, p, q are prime
- For any x , $x^y \bmod n = x^{(y \bmod \phi(n))} \bmod n$, where
 - n is prime or product of two primes
- Thus, for any x , $x^{d \cdot e} \bmod n = x \bmod n$ where $de=1 \bmod \phi(n)$

In RSA, we have:

- $n=p \cdot q$
- $\phi(n)=(p-1)(q-1)$
- Carefully chose e & d to be inverses mod $\phi(n)$
- Then, $e \cdot d=1+k \cdot \phi(n)$ for some k

Hence :

$$C^d = M^{e \cdot d} = M^{1+k \cdot \phi(n)} = M^1 \cdot (M^{\phi(n)})^k = M^1 \cdot (1)^k = M^1 = M \dots (\text{all in mod } n)$$

97

RSA Example - Key Setup

- Select primes: $p=17$ & $q=11$
- Compute $n = pq = 17 \times 11=187$
- Compute $\phi(n)=(p-1)(q-1)=16 \times 10=160$
- Select e : $\gcd(e, 160)=1$; choose $e=7$
- Determine d : $de=1 \bmod 160$ and $d < 160$
Value is $d=23$ since $23 \times 7=161= 10 \times 160+1$
- Publish public key $PU=\{7, 187\}$
- Keep secret private key $PR=\{23, 187\}$

98

RSA Example - En/Decryption

Sample RSA encryption/decryption is:

- Given message $M = 88$ (nb. $88 < 187$)
- encryption:
 $C = 88^7 \bmod 187 = 11$
- decryption:
 $M = 11^{23} \bmod 187 = 88$

99

RSA Key Generation

- Users of RSA must:
 - Determine two primes at random p, q
 - Select either e or d and compute the other
- Primes p, q must not be easily derived from modulus $n=p \cdot q$
 - Means must be sufficiently large
 - Typically guess and use probabilistic test
- Exponents e, d are inverses, so use Euclid's inverse algorithm to compute the other
 - Finding multiplicative inverse is easy via Euclid's algr
 - Finding exponential inverse is difficult

100

Generating RSA keys

- Finding big primes p, q
 - No practical way to determine absolutely that a given large number is prime
 - But can test if the number is probably prime
- Euler's Theorem:** For any a relatively prime to n
 $a^{\phi(n)} \bmod n = 1$
- Note that if n is prime, $\phi(n)=(n-1)$.
- Fermat's Theorem:** If n is prime and $0 < a < n$,
 $a^{n-1} = 1 \bmod n$.
- Given n a randomly generated 100-digit number, probability that n is not prime but $a^{n-1} \bmod n=1$ is $1/10^{13}$ for $0 < a < n$.

101

RSA Security

- Possible approaches to attacking RSA are
 - Brute force key search (infeasible given size of numbers)
 - Mathematical attacks (based on difficulty of computing $\phi(n)$, by factoring modulus n)
 - Knowing n not easy to find p and q ($n = p \cdot q$)
 - Not knowing p and q , difficult to find $\phi(n)$
 - Not knowing $\phi(n)$, difficult to find d such that $d \cdot e \bmod \phi(n) = 1$
 - Timing attacks (on running of decryption)
 - Chosen ciphertext attacks (given properties of RSA)

102

Diffie-Hellman key exchange

- ▶ Allows two individuals to agree on a secret key even though they can only communicate in public
 - ▶ Alice chooses a private number and from that calculates a public number
 - ▶ Bob does the same
 - ▶ Each can use the other's public number and their own private number to compute the same secret
- ▶ An eavesdropper cannot reproduce it

103

Diffie-Hellman key exchange

- ▶ A public-key distribution scheme
 - ▶ Cannot be used to exchange an arbitrary message
 - ▶ Rather it can establish a common key
 - ▶ Known only to the two participants
- ▶ Value of key depends on the participants (and their private and public key information)
- ▶ Based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) – easy
- ▶ Security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

104

Diffie-Hellman Setup

- ▶ All users agree on global parameters p and g :
 - ▶ Large prime integer or polynomial p
 - ▶ g being a primitive root mod p
- ▶ Each user (eg. A) generates their key
 - ▶ Chooses a secret key (number): $s_A < p$
 - ▶ Compute their public key: $T_A = g^{s_A} \text{ mod } p$
- ▶ Each user makes public that key T_A

105

Diffie-Hellman Key Exchange

- ▶ Shared session key for users A & B is K_{AB} :
 - $K_{AB} = g^{s_A \cdot s_B} \text{ mod } p$
 - $= T_A^{s_B} \text{ mod } p$ (which B can compute)
 - $= T_B^{s_A} \text{ mod } p$ (which A can compute)
- ▶ K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- ▶ if Alice and Bob subsequently communicate, they will have the same key as before, unless they choose new public-keys

106

Diffie-Hellman

- ▶ Alice and Bob agree on a prime p and a random number g
- ▶ Alice picks s_A at random, Bob s_B at random

Alice $T_A = g^{s_A} \text{ mod } p$ → Bob $T_B = g^{s_B} \text{ mod } p$ → Alice
- ▶ Alice computes $T_B^{s_A} \text{ mod } p$, Bob $T_A^{s_B} \text{ mod } p$

$T_B^{s_A} = (g^{s_B})^{s_A} = g^{s_B s_A} = g^{s_A s_B} = (g^{s_A})^{s_B} = T_A^{s_B} \text{ mod } p$
- ▶ Knowing g , p , $g^x \text{ mod } p$ (i.e., T_A or T_B), difficult to compute x (i.e., $X = s_A \cdot s_B$)
 - ▶ Attacker needs an x , must solve discrete log

107

Diffie-Hellman Example

- ▶ Users Alice & Bob who wish to swap keys:
- ▶ Agree on prime $p=353$ and $g=3$
- ▶ Select random secret keys:
 - ▶ A chooses $s_A=97$, B chooses $s_B=233$
- ▶ Compute respective public keys:
 - ▶ $T_A = 3^{97} \text{ mod } 353 = 40$ (Alice)
 - ▶ $T_B = 3^{233} \text{ mod } 353 = 248$ (Bob)
- ▶ Compute shared session key as:
 - ▶ $K_{AB} = T_B^{s_A} \text{ mod } 353 = 248^{97} = 160$ (Alice)
 - ▶ $K_{AB} = T_A^{s_B} \text{ mod } 353 = 40^{233} = 160$ (Bob)

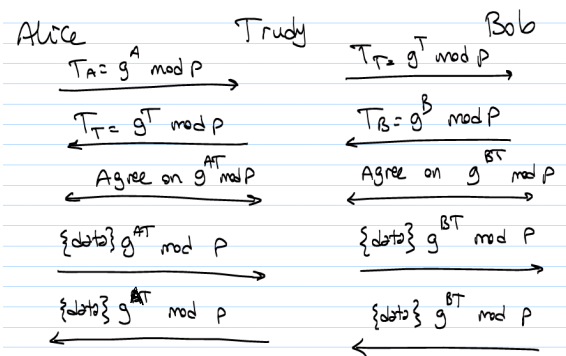
108

D-H Key Exchange Protocols

- ▶ Users could create random private/public D-H keys each time they communicate
- ▶ Users could create a known private/public D-H key and publish in a directory, then consulted and used to securely communicate with them
- ▶ Both of these are vulnerable to a meet-in-the-Middle Attack
- ▶ Authentication of the keys is needed

109

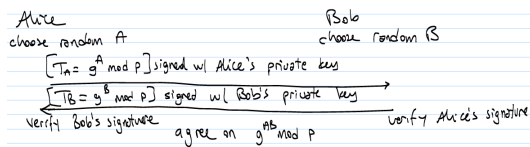
Man in the middle attack on D-H



110

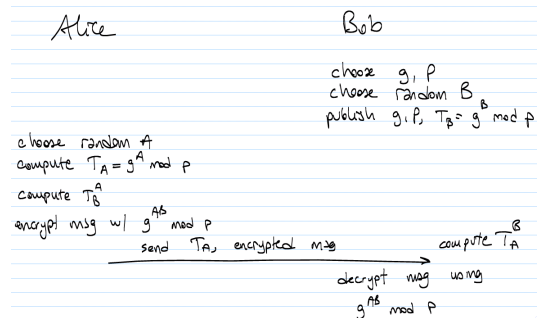
Defending against man in the middle attack

- ▶ Published D-H
 - ▶ Everyone agrees on p and g
 - ▶ Use PKI to reliably collect other party's public number T_x
- ▶ Authenticated D-H
 - ▶ Assume pre-established secret key or each others public key



111

Diffie-Hellman for encryption



112