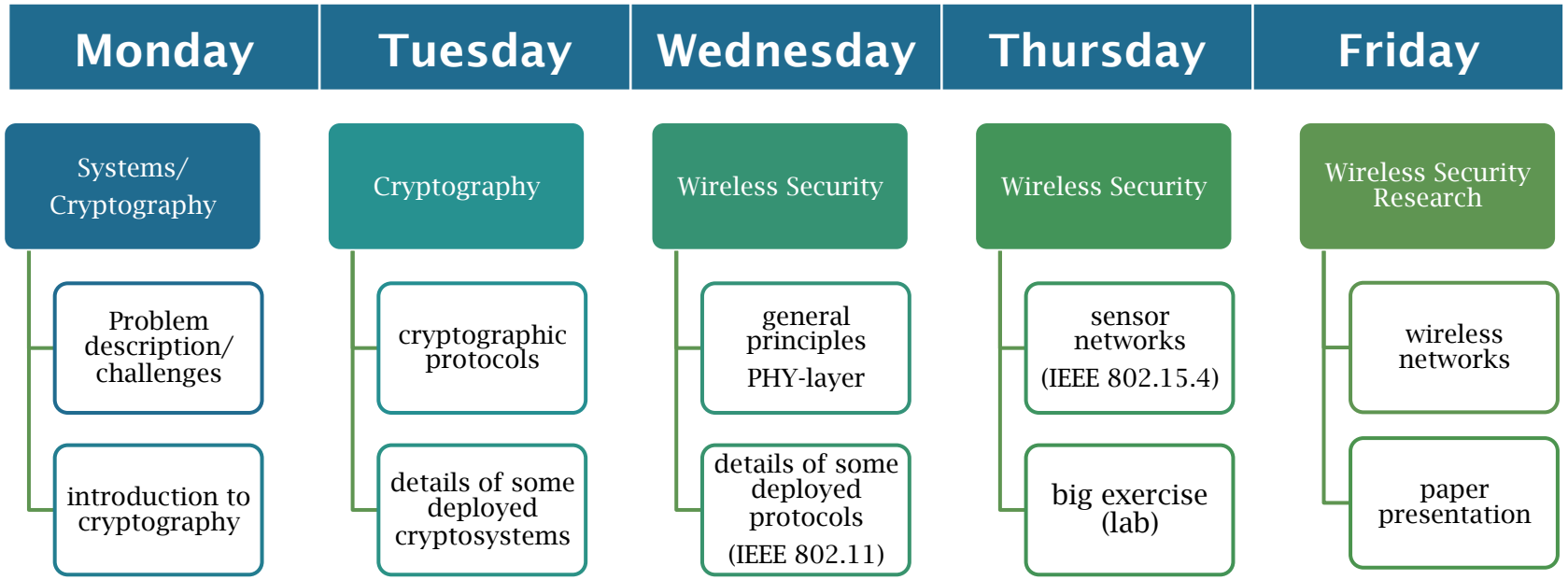
Security in Wireless Networks

Prof. Ivan Martinovic

Course Outline



Acknowledgment

- Course material for Monday and Tuesday (Cryptography) is based on the **Security Principles Course**, Software and System Security, Software Engineering Programme (SEP). Prof Andrew Martin.
- Course material for Wednesday (Wireless Security) – Friday is based on **Security in Wireless Network Course** , Software and System Security, Software Engineering Programme (SEP). Prof Ivan Martinovic

Technical Element

- cryptography and protocols
  - fairly abstract
  - mathematically deep: we will only scratch the surface
  - important controls, but not necessarily central to every security regime
  - underpin a great deal of technologies
  - indicative of the level of complexity in typical technical controls
  - ‘typical’ indication of skills and approaches needed by attackers and defenders
  - more long-lived insight than we would gain from studying particular products or systems

1

Cryptography: in principle

Prof. Ivan Martinovic

2

Acknowledgment

- Course material for Monday and Tuesday (Cryptography) is based on the **Security Principles Course**, Software and System Security, Software Engineering Programme (SEP). Prof Andrew Martin.
- Course material for Wednesday (Wireless Security) – Friday is based on **Security in Wireless Network Course**, Software and System Security, Software Engineering Programme (SEP). Prof Ivan Martinovic

3

Contents

- Cryptography
  - history and concepts
- Substitution *vs* Transposition
- Block ciphers *vs* Stream ciphers
- One-way functions
- Symmetric encryption and block modes
- Asymmetric Encryption

4

Cryptography's long history

- For centuries, cryptography has been employed for state secrets
- Apparently, *Julius Caesar* used a cipher:

	A	B	C	D	...	Y	Z
Key= D	D	E	F	G	...	B	C

- thus:

C	R	Y	P	T	O	G	R	A	P	H	Y
F	U	B	S	W	R	J	U	D	S	K	B

- 'limited' usefulness: why?

5

Vigenère (1586)

- generalizes Caesar's cipher by using a different key letter for each encryption

	C	R	Y	P	T	O	G	R	A	P	H	Y
Key = SPRA	U	G	P	P	L	D	X	R	S	E	Y	Y

- Many more possible keys
- Frequency analysis harder
- Basis of Enigma machine (key lengths in excess of 26^3)
- Related to an embarrassingly-weak cipher that's still used sometimes
- How does it improve on Caesar's cipher?

6

Transposition

- keep the same letters, but re-order them

7

Terminology

- For centuries, cryptography has been employed for state secrets
- Apparently, *Julius Caesar* used a cipher:

lookup table

	A	B	C	D	...	Y	Z
Key= D	D	E	F	G	...	B	C

plaintext (cleartext)

C	R	Y	P	T	O	G	R	A	P	H	Y
F	U	B	S	W	R	J	U	D	S	K	B

ciphertext

- 'limited' usefulness: why?

8

More terminology

cipher

- replace letters, digits, blocks, by other letters, digits, blocks in a systematic but secret way; also known as *cryptographic algorithm*

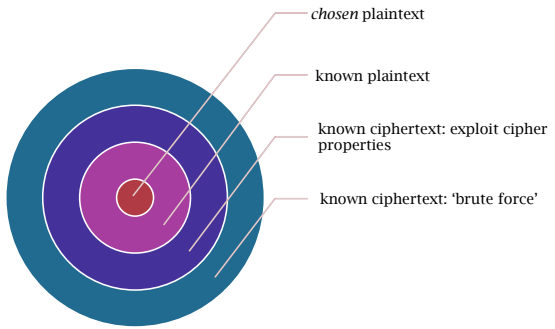
key

- just about all ciphers depend for their secrecy on a secret *key:* the size (bit length) of the key is of interest, as is the *keyspace:* the set from which all possible keys are drawn

9

Attacks

- objective is (generally) to discover the key



10

Modern cryptography

- (most) modern cryptography relies on the same principles as the techniques known in antiquity
- replace *letters* with blocks of binary data
- use a mathematical function (or, equivalently, a circuit) in place of a lookup table: *why?*

	0000	**0001**	**0010**	**0011**	**...**	**1110**	**1111**
Key= 0101	0101	0100	0111	0110	...	1011	1010

- block size: ~~4 bits~~, 64 bits, 128 bits, 256 bits: *why does this matter?*
- key size: similar order of magnitude, for block ciphers

11

Security versus Obscurity

- All *respectable* cryptography assumes that the attacker knows the *algorithm* (the cipher, and the implementation details) used for encryption (*but not the key*).
- Why is this?

12

Randomness

- The ideal cipher is indistinguishable from a random function
  - every ciphertext is equally likely
- this is known as the 'random oracle' model of cryptography
- true randomness is elusive: *pseudorandom* describes a function which passes suitable statistical tests
- we seek key-based mathematical functions which transform inputs in a pseudorandom way, when supplied with a randomly-chosen key
- if your attackers have the same pseudorandom number generator as you, they will use it to guess your keys

Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin. John von Neumann

13

Objective

- the role of the randomness is to make 'brute force' attacks difficult
- given the way that each generation of technology gives rise to faster computers (c.f. *Moore's Law*) and the increasing use of parallelism, we must build in large margins

strong secret

· something that with foreseeable technology could not be guessed within, say, many universe lifetimes, and has no known 'backdoor': for all practical purposes, 'unbreakable'.

weak secret

· something that might reasonably be guessed or brute-forced within hours or days: your password, for example

14

Entropy

entropy

· Entropy is the measure of *information content* in a message.
· It is measured in *bits.*
· The entropy of message *M* is sometimes written *H(M)*
· *(don't confuse this with a hash! – see later)*

Examples:

- a bit field holding 0 or 1 for false/true has entropy 1 bit
- a 3-bit field reporting the day of the week has entropy somewhat less than 3 bits(more precisely, $\log_2 7$ bits)
- string field holding the values "true" and "false" also has entropy 1 bit
- in order to achieve entropy of 64 bits for a password field, we need 11 randomly chosen case-sensitive alphanumeric characters
- because humans are typically bad at randomness, a freely-chosen 8 character password will typically have around merely 18 bits of entropy. [NIST SP 800.63, table A.1]

15

Question

- Why does entropy (measuring information content) matter in our discussion of randomness?

Recommended Paper:
Communication Theory of Secrecy Systems By C. E. Shannon
*netlab.cs.ucla.edu/wiki/files/**shannon**1949.pdf*

16

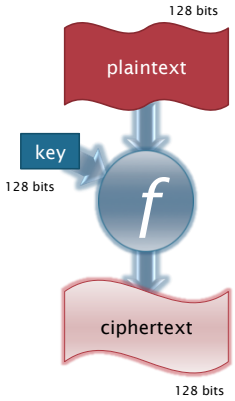
Substitution Ciphers

- 'like' Caesar's cipher, but with a much bigger 'alphabet'
  - for example, 2^{64} possible 'letters'
- *simple substitution* cipher is one-to-one plaintext to ciphertext
  - frequency analysis still applies
- *homophonic* (one to many), *polygram* (group to group), and *polyalphabetic* (multiple simple) ciphers all aim to confound frequency analysis
  - not necessarily successfully

17

Block Cipher

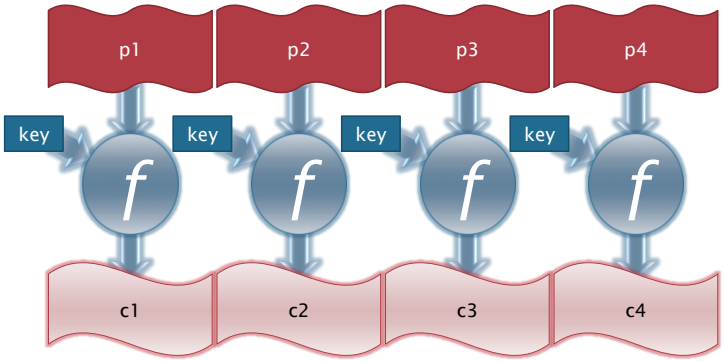
- Our general goal is to define *f*
- a *lookup table* would be ideal, but impractical (why?)
- so *f* must be a mathematical function
- bit sizes are examples
  - plaintext size will generally match ciphertext size
  - key size may differ, but will be similar order of magnitude to block size



18

Block cipher for whole message

- simplistic view: we will come back to this



19

Candidate: XOR

- Write $a \oplus b$

$0 \oplus 0 = 0$
$1 \oplus 1 = 0$
$1 \oplus 0 = 1$
$0 \oplus 1 = 1$

- Generalize for any number
  - express in binary
  - do bitwize $\oplus$

$\forall\, a : \mathbb{Z} \cdot \quad a \oplus a = 0$
$\forall a, b : \mathbb{Z} \cdot a \oplus b \oplus b = a$

We could try $f = \oplus$

cipherText = *plainText* $\oplus$ *Key*

plainText = *cipherText* $\oplus$ *Key*

block size = key size

is this any good?

20

Security Principle: A cipher should exhibit the avalanche effect: a change to a single bit of the input or the key should result in a potential change to every bit of the output.

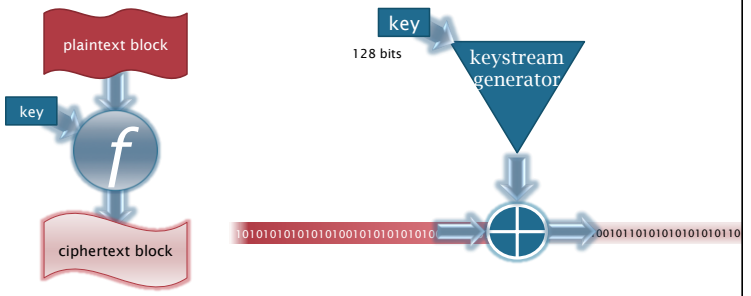
- people do have a habit of screwing this up
- 'super-fast encryption' often means ... XOR
- Windows CE (codename 'Pegasus')/ActiveSync 2.x offered to hold your NT password securely for you. It did this by storing
  - password $\oplus$ susageP

21

Stream Cipher

- We are mainly considering *block ciphers*
- an entirely different construction is a *stream cipher*
  - encrypt a 'continuous' stream of data, rather than separating into blocks
  - uses XOR at its centre – so *keystream generator* must avoid repeating sequences
  - ideally suited to streaming media
  - different algebra; different concerns; same basic strength when done right

22



block cipher

stream cipher

23

One-time pad: absolute secrecy

- classically: have a pad of randomly chosen letters; use each one once
- equivalent to Vigenère with an arbitrary long key, and a different key for each message
- *provided the pad is truly random* every ciphertext is equally likely, so without the pad it is *impossible* to recover the plaintext
- equivalent to a stream cipher with infinite key stream
- this is the only route to perfect secrecy
- truly random sources are hard to come by: and hard to share

24

Aside: transposition

- key-dependent *rearrangement* of bits has considerable strength
- limited random-access memory has curtailed its use

25

Notation

- cryptography as function application:
  - *C = encrypt(P)*
  - FUBSWRJUDSKB = *caesar* (CRYPTOGRAPHY)
  - *C = encrypt(k, P)*
  - FUBSWRJUDSKB = *caesarVariable* (D, CRYPTOGRAPHY)
  - *P = decrypt(k, C)*
  - *decrypt = encrypt* $^{-1}$
- encrypted content in protocols etc.
  - 'message *m* encrypted with key *k*'
  - $\{m\}_k$
  - crypto algorithm determined from the context!

26

Forms of encryption

- One-way functions
  - passwords
  - *hashes* and message digests
  - hash-based authentication
- Symmetric encryption
  - shared secret keys
  - bulk message encryption
- Asymmetric encryption
  - separate public-private *key pairs*
  - key distribution
  - message authentication and integrity

27

Password Security

- general principle: store encrypted passwords, only
- so the encryption *does not need to be reversible*
- compare UNIX and Windows approaches: salt or no salt
- passwords are now seen as very weak protection mechanisms
- social issues abound: see PAS module

28

Question

Why do you 'need' a long and complex password for your email login, but only a 4-digit PIN for your bank ATM card?

29

Threats to passwords

- Direct guess of an individual's password
- 'Brute force' attack
  - against an individual
  - against any/all the users of a system
- Modes of attack
  - 'online'; guessing etc. against the live interface
    - best scenario for defender
  - 'offline': guessing undertaken against a copy, not the live system
    - circumvents many controls; avoids arousing suspicion

30

Password brute force

1. encrypt every possible password, and store the results in a look-up table ('Rainbow table')
2. obtain a copy of the password file for the victim system
3. for each encrypted password, use the look-up table to discover its plaintext version

- needs substantial storage, and one-off compute power
- returns passwords in negligible time
- works well against Windows XP (and all previous versions)
  - NT Lan Man had certain other related weaknesses, too

31

Salt vs No Salt

- *e(p)*: fixed encryption/hash function for passwords
- simple password file/database:

username	encrypted password
charlotte	*e*(charlotte's password)
bob	*e*(bob's password)
alice	*e*(alice's password)

- attacker can easily pre-compute encrypted version of all the passwords of a given length (say, *n*)
- lookup table will be approximately $(n)*80^n$ bytes
  - assuming there are 80 available characters to be typed in passwords
- so for *n* characters of strength, the user must remember *n* characters

32

Salt vs No Salt

|| means string concatenation

- *e(p)*: fixed encryption/hash function for passwords
- salted password file/database:

username	salt	encrypted password
charlotte	gv	*e*(gv||charlotte's password)
bob	A%	*e*(A%||bob's password)
alice	=k	*e*(=k||alice's password)

- salt value chosen at random when password is created: stored in cleartext
- no significant overhead for legitimate normal use
  - nor for brute force approach (a)
- lookup table has to be two characters longer: size $(n+2)*80^{n+2}$ bytes
- *n* characters of 'strength' for (*n*-2) characters of Alice's memory

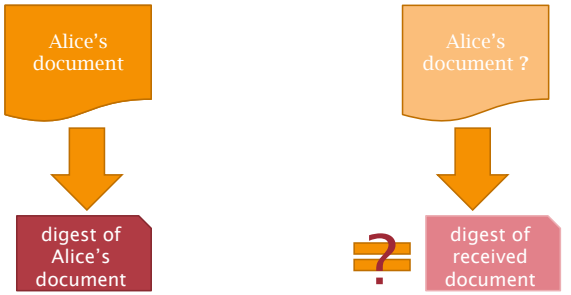
33

Passwords: summary

- offline brute-force guessing is the worst attack
  - and eventually always fatal
  - even an attacker with negligible resources can do this easily
- many legacy systems are subject to rainbow table attacks
  - so those systems are trivial to circumvent
- online attacks are harder to mount
  - but distinguishing good login attempts from bad ones is quite hard
  - as is knowing what to do about it
  - few systems implement a 'lock out' anyway

34

Message Digests



35

Question

- To construct a message digest, could we just XOR the blocks of the message together? Would a *checksum* suffice?

36

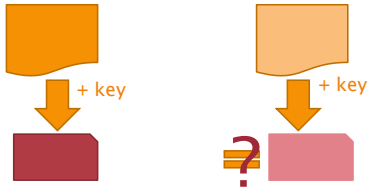
Hashing

- hash: compact representation of large amount of data
- many-to-one: there are inevitable *hash collisions*
- cryptographic hash design goal
  - **efficiency:** make it easy to compute hash from message
  - **one-way function:** make it hard (i.e. effectively impossible) to compute message from hash
  - **unpredictable collisions:** make it hard (i.e. effectively impossible) to find two messages with the same hash
  - **like a cipher**: every input bit affects every output bit; whole output space should be reachable (and equally likely?)

37

Message Authentication Code

- intending to give authentication without secrecy
- avoid running costly encryption/decryption whenever possible
- use a key-dependent one-way hash function
- key not passed with the communication: recipient knows the key, and uses it to recompute the hash, and check its value.



38

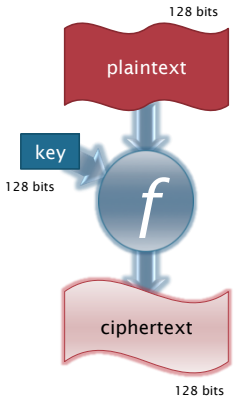
Symmetric Encryption

- historically, the only kind
- if you can do encryption, you can do decryption, and vice versa
- same key is used for both
- usually, run algorithm 'in reverse' for decryption
- sometimes (DES) same algorithm used to encrypt and decrypt
  - easy re-use in hardware
- often called 'secret key encryption'
  - essential that key is kept secret

39

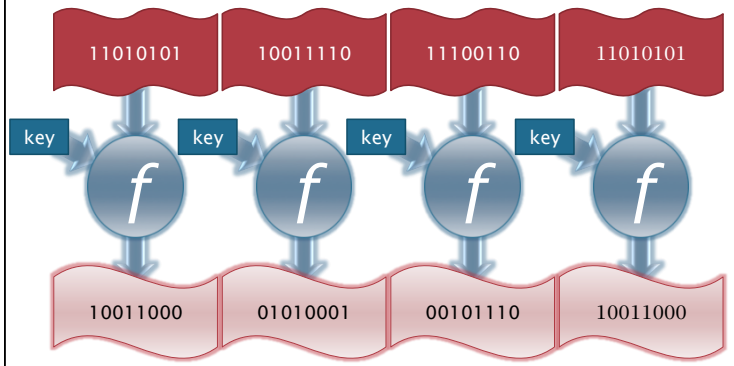
Using block ciphers

- recall this picture
- the 'obvious' way to use this to encrypt a long message is to break it into blocks, and use ***f*** to encrypt each block separately
- this is called electronic codebook mode (ECB)
- named because you *could* create a codebook (lookup) for blocks, but it would need 2^{128} entries (for a 128-bit block size)



40

ECB visualized



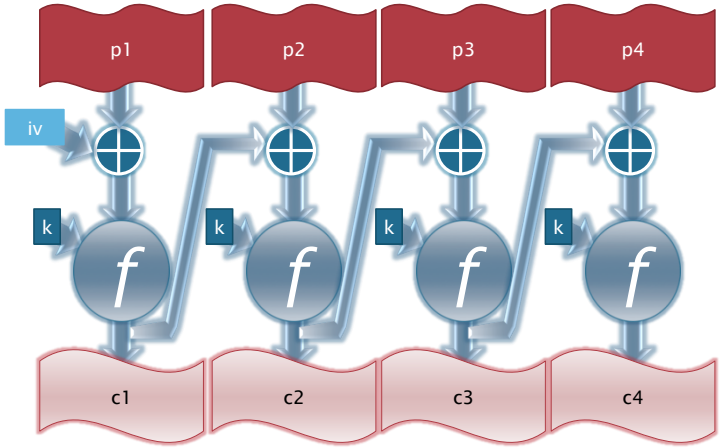
41

Attacking ECB

- cryptanalyst who has plaintext and ciphertext for a few messages, can begin to compile a code book *without knowing* ***k*** (or even ***f***, actually)
- messages tend to have standard formats
- block replay problem

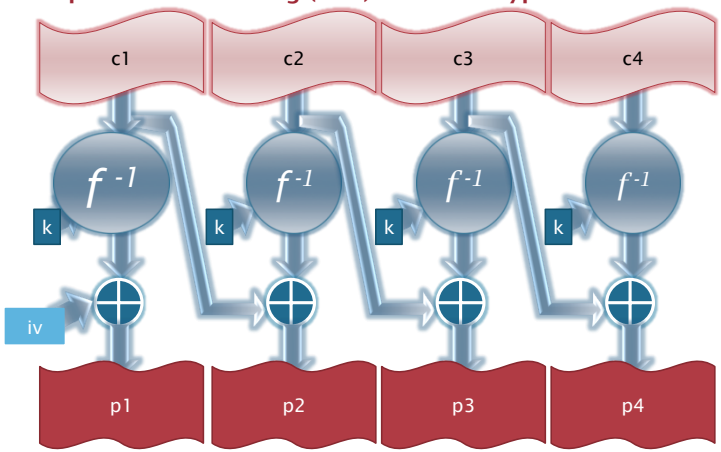
42

Cipher block chaining (CBC) mode: encryption



43

Cipher block chaining (CBC) mode: decryption



44

CBC notes

- requires initialization vector *iv*
  - (can be sent in cleartext at the start of the message)
- CBC is much 'safer'.
- problems with error propagation (not a problem with modern error-correcting channels)
  - bit errors, not too bad;
  - synchronization errors fatal
- still a basic correspondence between blocks
  - can you add/remove some at the end?: depends on message structure
- very long messages still have patterns
- does not protect *integrity* very satisfactorily
- CBC is one of a number of safer ways to use a block cipher

45

Uses of symmetric cryptography

- secrecy: bulk encryption
- authenticity, integrity
- secure storage
- key distribution problem

46

Asymmetric Encryption

- separate keys for encryption and decryption (a 'key pair')
- computationally infeasible to derive one from the other
- one key is (can be) public/published; the other is kept private
- therefore subject to many new attacks
- keys must be huge to prevent brute-force attacks, and algorithms must be resistant to chosen-plaintext attacks
- often called 'public key encryption' — it is quite safe to publish the encryption key

47

Encrypting and signing

- Call Alice's private key d_A and her public key e_A. Alice publishes e_A but keeps d_A secret.
- To send a message secretly to Alice, encrypt it with e_A: only Alice can read it, which she does using d_A. $\{m\}_{eA}$
- If Alice wants to prove she originated a message, she can encrypt it using d_A. Then anyone can get hold of e_A and read the message, and also know that it must have been encrypted using d_A — i.e. it was encrypted by Alice. $\{m\}_{dA}$
- If Alice wants to send a secret message to Bob, and have Bob know it came from her, she should first encrypt it with d_A, and then with e_B. $\{\{m\}_{dA}\}_{eB}$

48

But

- In fact, asymmetric encryption is computationally expensive, so we wouldn't do the above in practice.
- to prove Alice sent m: create a hash of the message, and sign that; $m,\{h(m)\}_{dA}$
- to send a secret message to Alice: use a symmetric session key to encrypt the message, placing this at the beginning of the message, encrypted under e_A. $\{k\}_{eA},\{m\}_k$
- How to do signing and encryption together, in practice, then?

49

Comparison

Symmetric Encryption	Asymmetric Encryption
40 – 256 bit keys	512 – 4096 bit keys
one key per two parties communicating	one 'key pair' per individual
efficient, especially in hardware	computationally expensive
DES, AES, Blowfish, Caesar, Vignere, Enigma	RSA, El Gamal, Elliptic Curves
use with ECB, CBC, ...	use sparingly, usually to encrypt other keys, or to sign hashes

50

Summary

- Cryptography
  - history and concepts
  - perennial issues
- Concepts surrounding cryptography
  - types of cipher
- One-way functions
  - passwords are problematic
  - hashes are similar to ciphers, but typically simpler
- Symmetric encryption and block modes
  - easy to use a good crypto algorithm badly
- Asymmetric Encryption
  - need a clear head; the *e*'s and *d*'s will trip you up eventually

1

Cryptography: in practice

Prof. Ivan Martinovic

2

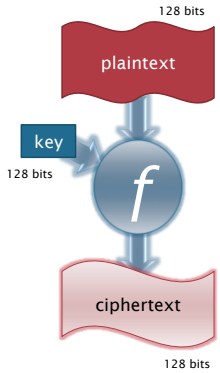
Contents

- Symmetric block ciphers
  - DES, 3DES, AES
  - cryptanalysis
- Asymmetric ciphers
  - RSA
  - ECC
- Digital Signatures
  - practical hashing (MD5, SHA-1, SHA-256, SHA-3)
  - signatures and attacks
- Quantum cryptography and quantum computing
  - impact on the future of cryptography

3

Review: Block Cipher

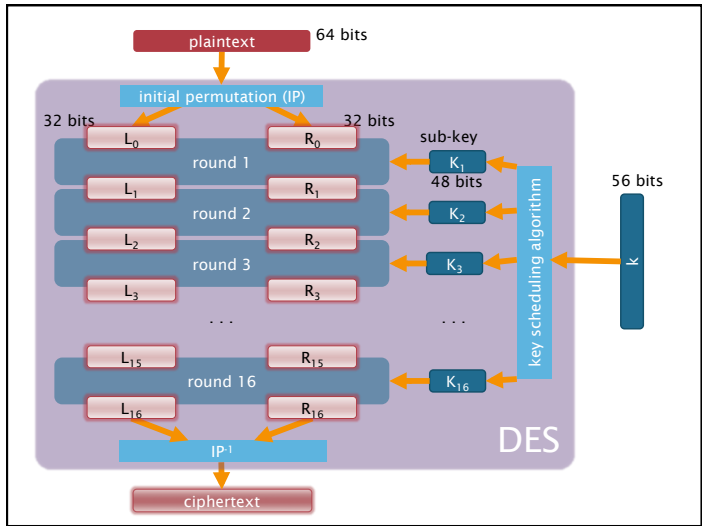
- Our general goal is to define f
- a *lookup table* would be ideal, but impractical
- so f must be a mathematical function
- we have seen how to use f in encrypting a whole message
- now we consider the design of f itself.



4

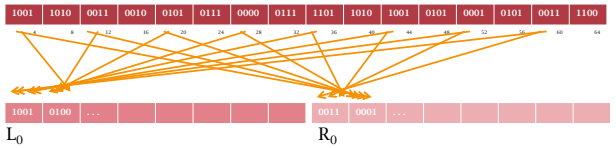
DES: *Data Encryption Standard*

- 64-bit block cipher
- 56-bit key
  - often expressed as a 64-bit number with parity checking
- baroque design
  - Mostly proposed by IBM (based on earlier work *'Lucifer'*)
  - 'approved' by NSA;
- originally designed to run on custom hardware
- eventually ISO/ANSI standard;
  - also known as DEA (*data encryption algorithm*)
- adopted 1976; ANSI standard 1981;
- NIST endorsement withdrawn, 19th May, 2005.

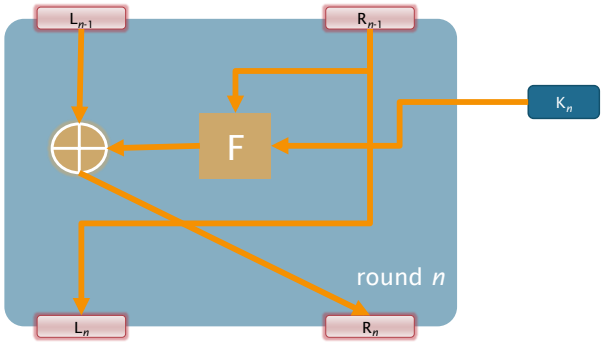


Initial permutation

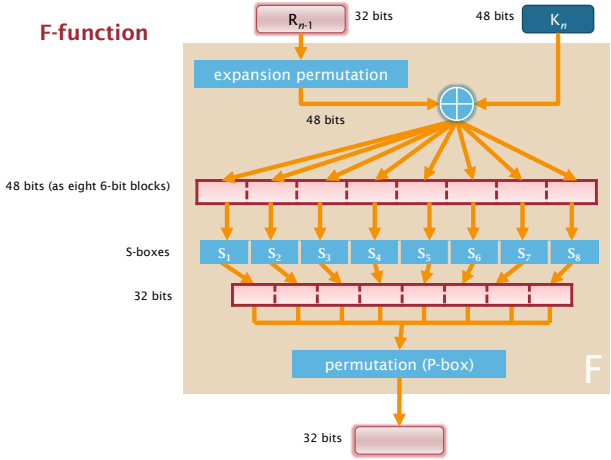
- simple re-ordering of bits
  - helps construct the algorithm; not cryptographically significant
- L_0 gets bits numbers:
  - 58,50,42,34,26,18,10,2,60,52,44,36,28,20,12, 4, 62,54,46,38,30,22,14,6,64,56,48,40,32,24,16, 8
- R_0 gets bits numbers:
  - 57,49,41,33,25,17, 9,1,59,51,43,35,27,19,11, 3, 61,53,45,37,29,21,13,5,63,55,47,39,31,23,15,7



Rounds



F-function



9

S-boxes

S_1

Row No.	Column Number															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

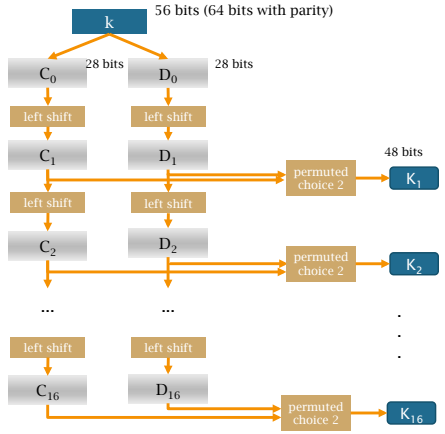
- simple fixed look-up table
  - outermost bits of 6-bit word form row, remainder form column number
- total of eight such tables/boxes

From FIPS PUB 46-3
FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION, 1999 October 25

10

Key Scheduling

- key expressed as 64-bit value passes through '*permuted choice 1*' (not shown) to yield 56 bits of real key material, *k*
- *left shift* (rotate) by one or two bits, depending on the round
- *permuted choice 2* is sometimes called a ' compression permutation'



11

Design Issues

- S-boxes are carefully designed
  - and were tweaked by NSA
- rumours of 'back-doors'
- *weak keys* exist
- *compliment key property* reduces 'brute force' search space
- clear means for 'avalanche effect'
- same algorithm works in reverse
  - just build the key schedule backwards
- much discussion about key length!

12

Breaking DES

- Brute Force
  - potentially as a 'known ciphertext attack' – the weakest kind of attack
- By cryptanalysis
  - usually as a 'chosen plaintext attack' – the strongest kind of attack

13

Brute force

- Brute force is the simplest attack
  - try every key in turn
- if you know the key generation process, try to re-run it
  - e.g. use keys based on hashes of dictionary words
  - e.g. Ubuntu OpenSSL/random number generator bug
  - otherwise, start at **0x00 0000 0000 0000** and go systematically through to **0xFF FFFF FFFF FFFF** (etc.)
- *n*-bit key gives rise to 2^n possible keys
  - comparison: 2^{25} seconds in a year; 2^{50} seconds since the big bang
  - following Moore's law gives a factor of up to 2^7 speed-up in a decade
- massive parallelism helps, of course
  - if we could make quantum computing do this kind of thing, we'd really get somewhere

14

Chinese Lottery – a hypothetical attack

- suppose you control the means of production...
- suppose you equip every receiver with a DES-breaking chip

Country	Population	number of Radios/TVs	time to break	
			56 bit	64bit
China	1 190 431 000	257 000 000	280s	20h
USA	260 714 000	739 000 000	97s	6.9h
Iraq	19 890 000	4 730 000	4.2h	44d
Israel	5 051 000	3 640 000	5.5h	58d
Wyoming	470 000	1 330 000	15h	160d

Source: [Schneier96,Table 7.2]
(million tests per second, using data from 1995 World Almanac and Book of Facts)

15

What really happened... RSA Labs Challenge

Contest	Prize	Start	End	Time for Solution
DES	$10,000	28 January 1997, 9 am PST	17 June 1997, 10:40 pm PST	140 days
RC5-32/12 /5	$1,000	28 January 1997, 9 am PST	28 January 1997, 12:30 pm PST	3.5 hours
RC5-32/12 /6	$5,000	28 January 1997, 9 am PST	10 February 1997, 10:00 am PST	313 hours
RC5-32/12 /7	$10,000	28 January 1997, 9 am PST	20 October 1997, 11:18 am PST	265 days
RC5-32/12 /8	$10,000	28 January 1997, 9 am PST	14 July 2002, 0150 UTC	1757 days Active search time as reported by winner

http://www.rsa.com/rsalabs/node.asp?id=2103

16

What really happened... RSA Labs Challenge

- *Challenge II-1:* distributed.net solved in 41 days, 1998
- *Challenge II-2:* purpose-built machine, 56 hours, July 1998
  - *machine cost $250000; prize was $10000.*
- "In 1999, the Electronic Frontier Foundation's "Deep Crack" machine, in combination with distributed.net, successfully solved RSA's DES Challenge III in 22 hours and 15 minutes."

http://www.rsa.com/rsalabs/node.asp?id=2100

17

Chosen plaintext attacks

- Two significant techniques developed to attack DES
  - work (with varying degrees of efficiency) against many similar algorithms
- Rely on
  1. analysing the algorithm structure,
  2. computing probabilities that bits of the key are a 1 or a 0,
  3. and then encrypting massive numbers of plaintexts until the accumulated data allows the probabilities to converge (so one key is 'overwhelmingly likely' – and easy to check)

18

Chosen plaintext attacks

- *Differential cryptanalysis*
  - look at XOR-difference between pairs of plaintexts and corresponding ciphertexts
  - recovers a DES key with, on average 2^{47} plaintexts
  - if the number of rounds is 17 or 18, becomes about as hard as brute force
  - 19 rounds or more, becomes impossible: needs more than 2^{64} plaintexts
  - published in 1990; transpires that DES's designers knew the technique — which is why it doesn't help too much, and why 16 rounds were chosen

19

Chosen plaintext attacks

- *Linear cryptanalysis:*
  - make linear approximations of the block cipher;
  - work by joining together 1-round linear approximations;
  - some S-boxes are easier to exploit than others
  - on average can recover key with 2^{43} plaintexts; best known attack against DES
  - DES is relatively weak against this attack — either the spooks didn't know about it, or had some other motive!

20

Double encryption

- Does encrypting twice

DES(k2,DES(k1,plaintext))

  - halve the security?
  - double the security?
  - square the security?
  - not make much difference?

21

- part of the answer has to do with the mathematical theory of *groups:*
  - Elements: ciphertext blocks
  - binary operation: composition
- if these form a group, then two encryptions are no better than one
- DES has been shown to be not at all group-like
- not the whole story …
  - 'meet in the middle attack' makes double encryption theoretically weak

22

3DES (triple DES): interim solution to DES weakness

- Use three DES encryptions in series
- Effectively 168-bit key size

  $3DES = DES(K1) ; DES^{-1}(K2) ; DES(K3)$
- Encrypt-Decrypt-Encrypt
  - Can build backward-compatible hardware this way
  - Does this affect the strength?
- Strong but slow

- DES with independent subkeys is also possible;
  - many other DES variants exist: e.g. 'export strength' 40-bit DES

23

AES: Advanced Encryption Standard

- DES has given rise to decades of research in symmetric cryptography and cryptanalysis
  - many other algorithms along the way: IDEA, RC2, RC5, Fortezza, Blowfish, Twofish, ...
- AES is designated successor to DES and 3DES
- result of open competition
  - and two years' public and private review
- winner *Rijndael*
  - two authors from Belgium (J. Daemen and V. Rijmen)
- Formal standard: Federal Information Processing Standards (FIPS) Publication 197
- Open standard
  - source code/reference implementations available from day one.

24

AES: options

- data blocks of 128 bits
- key lengths of 128, 192, and 256 bits
- corresponding number of rounds 10, 12, or 14
- algorithm allows for other options not endorsed by NIST
- intended for hardware or software

- so far, seems strong:
  - but doubt cast on the key-scheduling in the 256-bit/14-round version

25

AES: pseudocode

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
        byte state[4,Nb]
        state = in AddRoundKey(state, w[0, Nb-1]) // See Sec. 5.1.4
        for round = 1 step 1 to Nr 1
                SubBytes(state) // See Sec. 5.1.1
                ShiftRows(state) // See Sec. 5.1.2
                MixColumns(state) // See Sec. 5.1.3
                AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
        end for
        SubBytes(state)
        ShiftRows(state)
        AddRoundKey(state, w[Nr*Nb,(Nr+1)*Nb-1])
        out = state
end
```

26

Public-key algorithms

- recall that now we are looking for algorithms which use *two* keys
  - one for encryption and one for decryption
- so the keys must be related
  - but we don't want it to be possible to derive one if you know the other
- this allows us to make one key public, and keep the other secret
- so the algorithms are quite different
  - rely on 'hard' maths problems, i.e., no efficient (non-quantum) algorithms known
  - however, verifying the solution is simple
- various parts of mathematics have been proposed for this purpose; leading solutions are
  - RSA (factorisation)
  - Diffie-Hellman (discrete logarithm)
  - Elliptic Curve Diffie-Hellman (Elliptic curves)

27

Some difficult problems: factorisation

- Integer factorisation
  - Finding prime numbers of an composite number
  - Example: 15 = 3 x 5, where 3 and 5 are prime numbers
  - RSA-768 composite number:
    123018668453011775513049495838496272077285356959533479219732245215172640050726365751874520219978646938995647494277406384592519255732630345373154826850791702612214291346167042921431160222124047927473779408066535141959745985690214341 3
  - Factorisation of RSA-768 took 2 years using hundreds of machines
  - In contrast to factorisation, checking the solution is easy (just multiply the prime numbers and see if you get the composite)

28

Some difficult problems: discrete logarithm

- (Ordinary) logarithm problem
  - $Log_a(b)$: find an exponent x, such that $b = a^x$
- Discrete logarithm problem
  - $g^z \equiv c \bmod p$
    z is called the *discrete logarithm* of c modulo p to the base g
  - Example: $2^4 \equiv 1 \bmod 5$
    4 is the discrete logarithm of 1 modulo 5 to the base 2
  - The calculation of the discrete logarithm z when given g, c, and p is a computationally difficult problem and the asymptotical runtime of the best known algorithms for this problem is exponential in the bitlength of p
  - Verifying the discrete logarithm (called discrete exponentiation) is not difficult

29

RSA algorithm

- named for Rivest, Shamir, and Adelman
  - published the algorithm in 1978
- was in fact previously discovered – in secret
  - by Clifford Cocks at GCHQ (CESG), in 1973
- based on modular arithmetic

30

RSA – the main features

- Set-up
  1. choose prime numbers p and q.
  2. compute $n = p * q$
  3. select d and e, such that
     - *i.* d is relatively prime to $(p-1)*(q-1)$, and
     - *ii.* $(e*d) \bmod ((p-1)*(q-1)) = 1$
  4. discard p and q
  5. public key is the *pair* (e,n) and private key is the *pair* (d,n)
- operation (plaintext P, ciphertext C)
  - encrypt: $C = P^e \bmod n$
  - decrypt: $P = C^d \bmod n$

31

RSA issues

- large prime numbers?
  - fine: use a probabilistic prime checker (Rabin-Solway-Strassen)
- d can be selected by making it any prime larger than both p and q
- find e using Euclid's algorithm — polynomial time
- exponentiation mod n can also be done in polynomial time
- you can bias the speed by choosing e to be small, say — makes encryption faster, decryption slower; popular choice are 3, 17, 65537 (choosing $e = 3$ is vulnerable to the Low-exponent attack)
- in software, typically 100 times slower than DES; in hardware, about 1 000 times slower

32

Breaking RSA

A. Factoring n.

History suggests this is a hard problem.

if you can find p and q, then knowing e, you can easily find d

B. Finding $(p-1)*(q-1)$ without factoring n.

This is arguably about as hard.

33

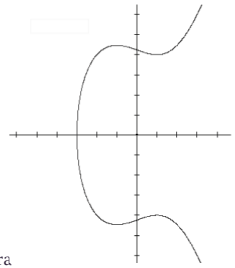
ECC: Elliptic Curve Cryptography

- Another basis for asymmetric (public key) cryptography
- Desirable because it has better scaling properties than RSA (much shorter keys)
- Elliptic curve cryptography (ECC) was proposed independently by Victor Miller and Neal Koblitz in 1985/1987.
- Rather more complex mathematically
  - many implementations
- Becoming widely adopted

34

Elliptic Curve – the main features

- Elliptic Curve E over Real Numbers ($\mathbb{R}$)
  - $y^2 = f(x)$ for a cubic polynomial f(x)
  - E(a,b): $y^2 = x^3 + ax + b$, where a, b $\in \mathbb{R}$
  - E.g.: E(-3,18): $y^2=x^3-3x+18$
- $E(a,b) = \{(x,y) : y^2 = x^3 + ax + b\} \cup \{O\}$
  - set of all points on elliptic curve and an extra point O at "infinity"
  - defines an abelian group, provided that discriminant $D \neq 0$
  - i.e., $4a^3 + 27b^2 \neq 0$

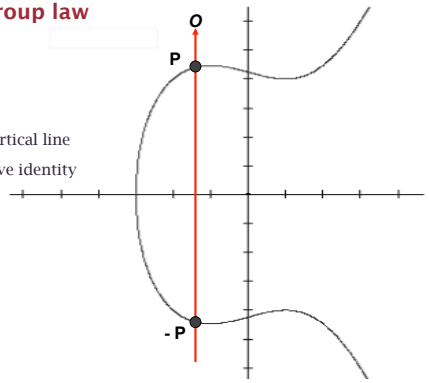


$y^2=x^3-3x+18$

35

Elliptic Curve – group law

- Point "O"
  - Point of infinity
  - Lies on every vertical line
  - Serves as additive identity
    - $P+O = P$
- Inverse
  - P = (x,y)
  - -P = (x,-y)
  - $P + (-P) = O$



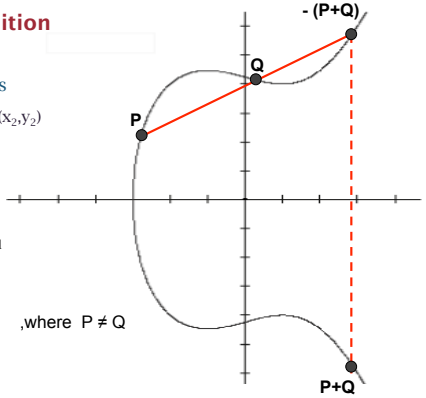
36

Elliptic Curve – addition

- Addition of two points
  - $P = (x_1,y_1)$ and $Q = (x_2,y_2)$
  - $P + Q = (x_3,y_3)$
- Algebraic

$$x_3 = \Delta^2 - x_1 - x_2$$
$$y_3 = \Delta(x_1 - x_3) - y_1$$

$$\Delta = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & \text{,where } P \neq Q \\ (3x_1^2 + a)/2y_1 & \text{,where } P = Q \end{cases}$$



37

Elliptic Curve over $\mathbb{Z}_p$ (Prime Curves)

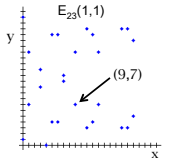
- Similar to EC over $\mathbb{R}$
- $\mathbb{Z}$ is the set of integers, $\mathbb{Z}_p = \{0,1,2,\ldots,p-1\}$, where p is a prime number
- Variables and coefficients are restricted to elements of $\mathbb{Z}_p$
- $E = \{ (x,y) : y^2 \bmod p = (x^3 + ax + b) \bmod p \} \cup \{ O \}$, where $x,y,a,b \in \mathbb{Z}_p$
  - Defines an abelian group, provided that discriminant $D \neq 0$
  - i.e., $(4a^3 + 27b^2) \bmod p \neq 0$
- Example: a=1, b=1, x=9, y=7, p=23

  $7^2 \bmod 23 = (9^3 + 9 + 1) \bmod 23$

  $49 \bmod 23 = 739 \bmod 23$

  $3 = 3$

  $\Rightarrow (9,7) \in E_{23}(1,1)$



$y^2 \bmod p = (x^3 + x + 1) \bmod 23$

- Addition remains the same, but also within modular operation

38

Elliptic Curve Discrete Logarithm Problem

- The security of ECC depends on the problem of **Elliptic Curve Discrete Logarithm Problem (ECDLP):**
  - Given a curve E and two points Q and P, find n such that $Q = n \times P$
- At the moment, the best algorithms for solving ECDLP are much less efficient than the algorithms for solving discrete logarithm or for factoring large integers.
- Many public-key protocols can be implemented using ECC
  - the Elliptic Curve Diffie-Hellman (ECDH) key agreement scheme is based on Diffie-Hellman key agreement (discrete logarithm)
  - the Elliptic Curve Digital Signature Algorithm (ECDSA) is based on the Digital Signature Algorithm (discrete logarithm)

Elliptic Curve Diffie-Hellman Key Exchange (ECDH)

- Choose
  - $E_p(a,b)$: large prime p and EC parameters a,b
  - Point $G = (x_1,y_1) \in E_p(a,b)$
    - Order of G should be very large value
    - The order n of a point G on an elliptic curve is the smallest positive integer n such that $G \times n = O$.
- ECDH Public parameters: $E_p(a,b)$ and G

Alice		Bob
Private Key: $n_a < n$		Private Key: $n_b < n$
Public Key: $P_a = G \times n_a$	$\xrightarrow{P_a}$	Public Key: $P_b = G \times n_b$
Shared Key: $K = P_b \times n_a$	$\xleftarrow{P_b}$	Shared Key: $K = P_a \times n_b$

$$P_b \times n_a = (G \times n_b) \times n_a = (G \times n_a) \times n_b = P_a \times n_b$$

40

Efficiency of ECC

- NIST recommended key sizes (in bits):

Symmetric Key Size	RSA and DH Key Size	EC Key Size
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

- E.g.: To protect a symmetric 128-bit AES key one should use a 3072-bit RSA key or a 256-bit ECDH key

Source: http://www.nsa.gov/business/programs/elliptic_curve.shtml

41

Crypto Performance: symmetric algorithms

- Benchmarks using
  - OpenSSL (using 1 of the 4 cores), Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz.

		16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
rc4		413638.26k	641908.25k	737861.80k	778967.72k	775034.20k
des	cbc	62096.76k	63910.98k	63973.37k	63590.06k	63946.75k
des	ede3	23975.98k	24349.12k	24486.23k	24405.67k	24751.35k
rc2	cbc	38653.41k	39182.72k	39425.71k	39482.37k	39531.86k
blowfish	cbc	106321.62k	111338.43k	112307.80k	112809.64k	113589.34k
cast	cbc	98649.05k	102900.35k	104184.83k	104588.97k	104445.27k
aes-128	cbc	100340.98k	108169.22k	109668.27k	110206.63k	111353.86k
aes-192	cbc	86331.45k	90609.98k	92677.97k	93334.19k	93025.62k
aes-256	cbc	73806.76k	78227.01k	79286.70k	80326.31k	80153.26k

42

Crypto Performance: asymmetric algorithms

			sign	verify	sign kb/s	verify kb/s
rsa	512	bits	0.000053s	0.000004s	18866.8	232884.8
rsa	1024	bits	0.000184s	0.000012s	5425.6	82669.3
rsa	2048	bits	0.001324s	0.000041s	755.1	24433.8
rsa	4096	bits	0.009569s	0.000153s	104.5	6528.5

43

Crypto Performance: hashing algorithms

	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
md2	0	0	0	0	0
mdc2	0	0	0	0	0
md4	81569.65k	245852.74k	567837.18k	843165.01k	983283.84k
md5	58067.51k	169312.94k	373451.18k	535094.95k	612229.12k
hmac(md5)	46428.89k	145214.39k	335519.15k	509540.35k	606142.46k
sha1	63343.55k	180989.91k	396493.40k	555289.60k	635505.32k

44

Crypto Performance: asymmetric algorithms (ECC)

			sign	verify	sign kb/s	verify kb/s
ecdsa	160	bits	0.0001s	0.0002s	15766.4	4439.2
ecdsa	192	bits	0.0001s	0.0003s	13140.8	3634.7
ecdsa	224	bits	0.0001s	0.0004s	10463.3	2662.3
ecdsa	256	bits	0.0001s	0.0004s	9015.5	2241.2
ecdsa	384	bits	0.0002s	0.0009s	4667.3	1071.1
ecdsa	521	bits	0.0004s	0.0020s	2482.5	498.7

ecdsa = elliptic curve digital signature algorithm

45

Crypto Performance: summary

- Relative Computation Costs of Diffie-Hellman and Elliptic Curves

Security Level (bits)	Ratio of DH Cost : EC Cost
80	3:1
112	6:1
128	10:1
192	32:1
256	64:1

Source: http://www.nsa.gov/business/programs/elliptic_curve.shtml

46

Side-note

- More 'exotic' asymmetric cryptography also exists.
- For example, have multiple encryption keys, and a single decryption key
  - or vice versa
  - sometimes used for 'group signatures'
  - useful for preserving privacy or anonymity
- details are out of scope for us

47

Digital Signatures

desirable signature properties:

- authenticity — the signer deliberately signed
- unforgeability
- not re-usable
- document unalterable after signature
- cannot be repudiated

none of these is entirely true of pen & paper signatures

- involve cryptography
- can include timestamps
- usually sign a message digest
- use the properties of asymmetric cryptography
- encrypt with private key

  $\{m\}_{dA}$
- anyone can verify (decrypt) with public key, ***eA***

48

Practicalities

- Asymmetric algorithms are too inefficient to do this in practice
- Sign a *hash* of the message, not the message itself

  $m,\{h(m)\}_{dA}$
- key compromise is an inherent problem
  - how do you (as a relying party) distinguish

    sign ➲ compromise ➲ repudiate (*deliberate compromise*)
  - from

    compromise ➲ sign ➲ repudiate (*accidental compromise*)
- talk of *Alice* or *Bob* is misleading
  - the signature is created *by software* not by the person
  - whether or not the signature is applied to the data *Alice* expects is entirely in the hands of the interface designer

49

Hash Algorithms in use

MD5

- processes input text in 512-bit blocks
- output is four 32-bit blocks; concatenate to a 128-bit hash value
- multiple 'rounds' like DES etc., based on bitwize , AND, OR, NOT, and left circular shift
- several groups have demonstrated serious, repeatable, hash collisions
- use of MD5 discouraged
  - but not dead yet

SHA-1

- processes input text in 512-bit blocks
- similar construction, but uses five 32-bit blocks instead of four; thus giving a 160-bit output
- some mystery in the design; NSA had a hand in it
- SHA-1 is very widely used, but looking shaky: increasingly good attacks are being found; consensus is that it needs replacing.

50

Hash Algorithms

SHA-2

- 'Next generation', sharing some details with SHA-1
- most popular variant, SHA-256 is gaining ground
- commonality with SHA-1 makes cryptographers uneasy

SHA-3

- open competition in progress to find a fresh replacement
- round one: 64 entrants, November 2008
- round two: 14 remaining candidates; started July 2009
- round three: 5 finalists announced December 2010
- final result (winner) due mid-2012

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

51

Question

- Could we use a cryptographic algorithm to create a hash implementation?

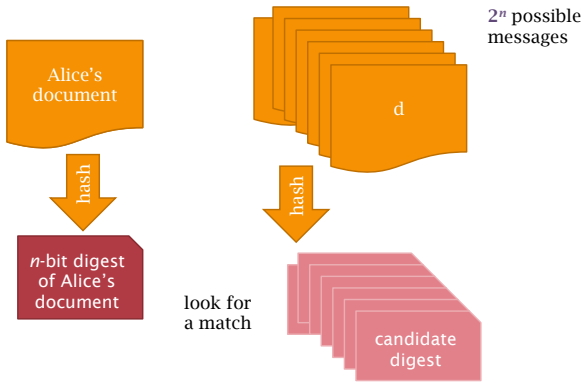
52

Attacks on Hashing and Signatures

- ideal attack:
  1) find two messages that hash to the same digest
  2) one is nice; one nasty
  3) get Alice to sign the digest, seeing that it corresponds to the nice message
  4) distribute the nasty one, and say that Alice signed it
  5) everyone can verify that this is true
- Step (1) is difficult if you are given one of the messages by Alice
  - see following slides; compare the 'birthday book paradox'

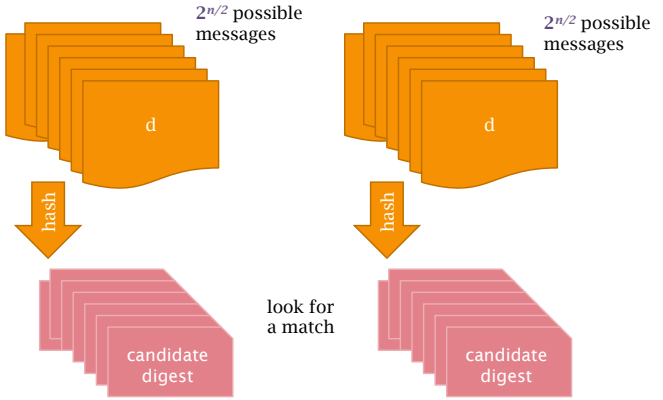
53

Attack One: Given Message



54

Attack two: choosing both messages



55

nice...

[John's | Mr Smith's] assignment was [an outstanding | a most impressive] piece of [work | writing]. He has [demonstrated | shown] a [clear | detailed] understanding of the [material | subject], and has [outclassed | eclipsed] his fellow-students.

The [answer | solution] to [Question | Q] 4 was particularly [impressive | striking], in that it used a [technique | idea] far [above | in advance of] any of the usual [methods | algorithms]. This is a [significant | key] breakthrough, and should [receive | see] publication [as soon as possible | immediately].

56

nasty...

The [assignment | work] submitted by [John | Mr Smith] was [most | very] [disappointing | poor]. He [completely | entirely] misunderstood the questions, and [demonstrated | showed] very little [understanding | knowledge] of the subject. He is [easily | clearly] the [weakest | least-able] student in [this class | our programme].

A [clear | plain] demonstration of his [shortcomings | problems] is seen in the [answer | solution] to [Question | Q] 4. The [work | material] presented is at odds with all of the [course material | literature] and cannot possibly be right.

57

Outcome

- 2^{16} possible versions of each message
- With a (hypothetical) 32 bit hash, this gives a very good chance of finding two messages with the same hash.
- If no hash collisions are found, re-run with a few more options.
- Examiner-Alice signs the nice one; nasty one is put on file
- John Smith is well and truly stitched-up
- The moral of the story: your hash needs to be twice as long as you thought

58

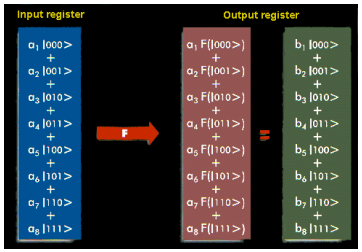
Documented Attacks on MD5

- 1996: collision attacks identified
- 2005: researchers release pair of PostScript documents which render to different texts but have identical hashes
  - and something similar for digital certificates
- 2008: fairly comprehensive attack against MD5-signed digital certificates

59

Quantum Computing

- *Oxford Centre for Quantum Computation* is a good source of information
- http://www.qubit.org/tutorials
- by using quantum effects, create a *qubit* register which holds *both* values **1** and **0** simultaneously
  - call this 'superposition'
- put together n qubits to build a register holding 2^n values simultaneously
- can think of this as a collection of probability coefficients – must add up to 1.



- apply an operation to the register – change the coefficients
- making an 'observation' destroys the superposition, and delivers a single answer
- with the 'right' operations, the desired answer becomes overwhelmingly likely

60

Quantum Computing Practicalities

- Shor's Algorithm
  - rapidly factorizes (large) numbers
  - equivalently enables us to reverse discrete logarithms
  - result: easy break of asymmetric cryptography
  - state-of-the-art appears to be 10 years old
    - $(15 = 5 \times 3$ computed in 2001 at IBM)
- Grover's Algorithm
  - rapidly search unsorted data
  - equivalently, invert a non-invertable function

61

Quantum Cryptography

- uses similar effects to quantum computing, but in a different – and so far more successful – way.
- claimed as a solution to the key distribution problem
- eavesdropping is detectable according to the laws of physics
  - no passive observers exist at the quantum level (c.f. Heisenburg)
- quantum cryptography devices are available to buy
  - apparently work over many km today

Example implementation

1. Photons polarized at 0, 45, 90, or 135 degrees
2. Recipient can measure polarization, *either* the rectilinear or the diagonal, but *not both* for a single photon
3. Sender sends photons, choosing polarizations at random
4. Recipient choses detection mode at random
5. Recipient publishes the detection mode chosen
6. Where it was the *wrong* mode, both parties throw away the bit.
7. Where it was the *right* mode, they have an un-eavesdropped (sequence of) bit(s)
8. Eavesdropper would need to detect and retransmit photons – statistically impossible to get right.

62

Summary

- Good, commercial-grade cryptography is readily available
- Implementations can be made *quite* efficient
  - but not without resource costs
- Asymmetric Algorithms are all much less efficient than symmetric ones
  - but are adequate for signing hashes, encrypting *session keys*, etc.
- Hashing is a bit of a mess right now
  - but is heading for a good outcome
- Quantum cryptography provides alternatives for establishing encrypted channels
  - few obvious use cases
- Quantum computing *might* some day defeat all asymmetric algorithms