1 Introduction

Terminology

Cryptography: the act or art of writing in secret characters.

Cryptanalysis: the analysis and deciphering of secret writings.

Cryptology: the scientific study of cryptography and cryptanalysis.

Encryption: method for encoding messages.

Decryption: method for decoding messages.

Plaintext: unencrypted message (in the clear).

Ciphertext: encrypted message.

Applications of Cryptography

Example applications:

1. Secure communications
2. Digital Signatures
3. End-to-end encryption
4. Protecting data
5. Storing passwords
6. Online payment
7. Online auctions
8. Electronic voting
9. Digital cash
10. Blockchain

Encryption/Decryption

Encryption is a means of transforming plaintext into ciphertext

- Under the control of a secret key

We write $c = e_k(m)$, where

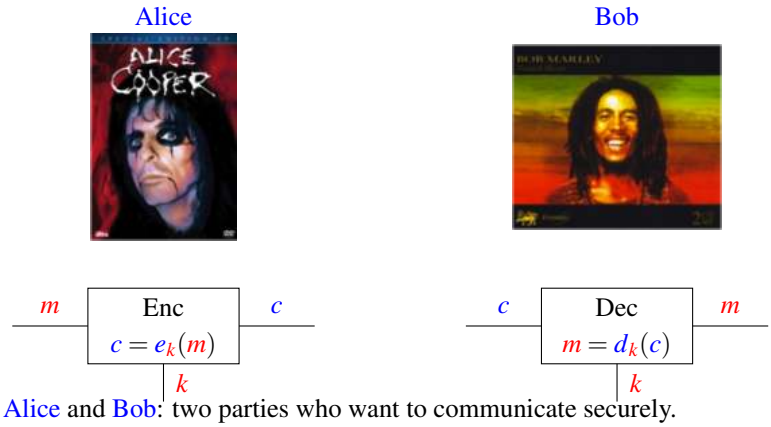
- m is the plaintext
- e is the encryption function
- k is the secret key

- c is the ciphertext

Decryption $m = d_k(c)$, where

- d is public
- the secrecy of m given c depends totally on the secrecy of k
- each party needs access to the secret key
- This needs to be known to both sides, but needs to be kept secret

Participants



- Alice and Bob: two parties who want to communicate securely.
- Eve: an eavesdropper who wants to listen/modify their communication.

Adversarial Model
The number of keys must be large to prevent exhaustive search
Worst case assumptions - assume attacker has:

- Full knowledge of the cipher algorithm
- A number of plaintext/ciphertext pairs associated to the target key

Kerchoff's Principle (1883)
System should be secure even if algorithms are known, as long as key is secret.

The cipher designer must play the role of the cryptanalyst:

- In practice ciphers are used which are believed to be strong
- All this means is that the best attempts of experienced cryptanalysts cannot break them.

Attacks
There are two basic types of attack:

- Passive
- Active

With a passive attack, information is accessed but not modified.

- An administrator reading mail messages being sent across the Internet.
- A hacker gaining access to information contained in bank accounts.

With an active attack, information or the system is modified.

- An administrator modifying mail messages.
- A hacker withdrawing money from a bank account.

Attacks
Some example types of attack:

Ciphertext only attack: ciphertext known to the adversary (eavesdropping)

Known plaintext attack: plaintext and ciphertext are known to the adversary

Chosen plaintext attack: the adversary can choose the plaintext and obtain its encryption (for example, has access to the encryption system)

Chosen ciphertext attack: the adversary can choose the ciphertext and obtain its decryption

Dictionary attack: the adversary builds a dictionary of ciphertexts and corresponding plaintexts

Brute force attack: the adversary tries to determine the key by attempting all possible keys

What is a secure system?

- Every system is susceptible to attack.
- Security is about ensuring that attacks will not be successful.
- A security mechanism prevents an attack from being successful.
  - A password can prevent unauthorized access to a computer.
  - A hand-written signature can prevent someone denying that they entered into a contract.
  - Watermarking in bank notes can prevent forgery.
- A security mechanism detects, prevents, or recovers from a attack.
- A secure system is one in which known threats have been considered and suitable security mechanisms have been incorporated to prevent successful attacks.

Trust

- In any secure system, certain components need to be trusted.
- A trusted component is assumed to behave correctly, i.e., we do not need security mechanisms to prevent it misbehaving.
  - It is common to trust operators of secure systems.
  - It is common to trust software within secure systems.
  - Of course, such trust is based on operators being vetted and software having been assured.
- In general, the number of trusted components in a system should be as small as possible.
- It is common to have components that have limited trust.
  - For example, they may be trusted within a limited part of a system.
  - In addition, their actions may be audited.
- It is also common to divide trust between a number of components.
  - Certain actions may require a number of individuals to agree.
  - For example, cheques may require two signatures.

Security Policies

- To build a secure system we need to:
  - Assess threats.
    - What threats exist?
    - What is the cost if there is a successful attack?
  - Identify trusted components.
  - Determine appropriate security mechanisms to counter threats.
    - What mechanisms will work and what will they cost?
    - How will these various mechanisms work together?
  - Define procedures to ensure the correct operation of the system.
  - Define review and audit mechanisms.
- All this requires a security policy.
- A system is only secure relative to the security policy that it enforces.

Security Objectives
These were originally summarised as the CIA triad:

- Confidentiality: keeping information secret from those not entitled to see it.
- Integrity: ensuring that information has not been altered.
- Availability: ensuring that information can be accessed in an appropriate time-frame.
  - This includes preventing denial of service (DoS) attacks.

The following security objectives are also important:

- Authentication:
  - Entity Authentication: ensuring that the purported identity of an entity is correct.
  - Message Authentication: ensuring that the purported source of information is correct.
- Non-repudiation: ensuring that an entity cannot deny a previous action.

Depending on the particular system, these security objectives can be met by using a combination of cryptographic and non-cryptographic security mechanisms.

Types of Security

- Physical Security: most security is based on ensuring that the physical access to resources is restricted.
- Secrecy: by keeping the existence or details of a system secret, then it may be more secure.
- Personnel Security: personnel who build and operate secure systems need to be trusted.
- IT Security: non-cryptographic mechanisms used in computers, networks, etc.
- Cryptographic Security: mechanisms based on the use of cryptography.

Perfect Security
Is perfect security possible?

- The security of a system is a negative attribute.
  - In general, it is impossible to demonstrate absolute security.
- Security mechanisms have limited applicability.
  - A security mechanism will only prevent a limited number of possible attacks.

- Security mechanisms have associated costs.
  - There is no point using security mechanisms that cost more than the outcome of a successful attack.
- In many circumstances, security requirements evolve.
  - Security is not a static attribute of a system and typically, security must be ”tightened” as attacks occur or threats increase.
- Prevention verses Detection.
  - The ideal is to prevent attacks becoming successful.

Therefore, except for the most trivial of systems, there is no perfectly secure system.

What is a Security Protocol?

- Let us assume that we are operating some system in an environment consisting of a collection of entities or players.
- Some of these entities will be good guys trying to achieve one or more security objectives as part of the system.
- Others will be bad guys trying to attack the system and overcome the security objectives.
- A security protocol is a description of how the good guys should interact with each other to achieve the stated security objectives.
- A security protocol should be able to achieve the security objectives no matter what attacks are mounted by the bad guys.

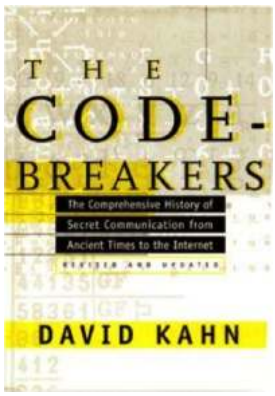
Security and Networks

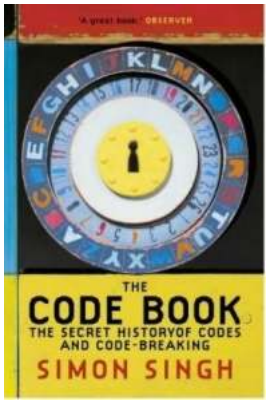
- A network is like any other system, except that it is distributed.
- In addition to being physically distributed, ownership may also be distributed.
- The internet is the prime example of the problems associated with network security.
- How can security be realized in such a chaotic environment?
  - The answer is to use security protocols based on cryptography.
- Is cryptography sufficient?
  - No - cryptography is necessary, but it is not sufficient.
  - We still need to use other forms of security.

2 Historical Ciphers

Historical Ciphers

- Shift Cipher
- Substitution Cipher
- Vigenère Cipher
- Vernam Cipher
- Rotor Machines





2.1 Shift Cipher

Shift Cipher
Identify each letter with a number

- A = 0
- B = 1
- C = 2
- ⋮
- Z = 25

The key k is a number in the range 0-25

- Encryption: add k to each letter (modulo 26).

Julius Caesar used the key $k = 3$.

- ATTACK AT DAWN becomes DWWDFN DW GDZQ

We break a shift cipher by using the statistics of the underlying language.

Geoff Hamilton

English Letter Frequencies

A	8.05	J	0.10	S	6.59
B	1.62	K	0.52	T	9.59
C	3.20	L	4.03	U	3.10
D	3.65	M	2.25	V	0.93
E	12.31	N	7.19	W	2.03
F	2.28	O	7.94	X	0.20
G	1.61	P	2.29	Y	1.88
H	5.14	Q	0.20	Z	0.09
I	7.18	R	6.03		

Most common bigrams: TH,HE,IN,ER,AN,RE,ED,ON,ES,ST,EN,AT,TO,NT,HA
Most common trigrams: THE,ING,AND,HER,ERE,ENT,THA,NTH,WAS,ETH,FOR

Shift Cipher
Take the following example cipher text:
BPMZM WVKM EIA IV COTG LCKSTQVO EQBP NMIBPMZA ITT ABCJJG IVL JZWEV IVL BPM WBPMZ JQZLA AIQL QV AW UIVG EWZLA OMB WCB WN BWEV OMB WCB, OMB WCB, OMB WCB WN BWEV IVL PM EMVB EQBP I YCIKS IVL I EILLTM IVL I YCIKS QV I NTCZZG WN MQLMZLWEV BPIB XWWZ TQBBTM COTG LCKSTQVO EMVB EIVLMZQVO NIZ IVL VMIZ JCB IB MDMZG XTIKM BPMG AIQL BW PQA NIKM VWE OMB WCB, OMB WCB, OMB WCB WN PMZM IVL PM EMVB EQBP I YCIKS IVL I EILLTM IVL I YCIKS IVL I DMZG CVPIXXG BMIZ
We need to compare the frequency distribution of this text with standard English

Letter Frequencies

A	2.59	J	1.44	S	1.73
B	10.37	K	2.59	T	3.46
C	5.48	L	6.63	U	0.29
D	0.58	M	10.09	V	8.36
E	4.61	N	2.31	W	6.63
F	0.00	O	3.46	X	1.15
G	2.59	P	4.03	Y	1.15
H	0.00	Q	4.03	Z	4.90
I	11.53	R	0.00		

Cracking the Cipher
The shift of E seems to be either 4, 8, 17, 18 or 23
The shift of A seems to be either 1, 8, 12, 21 or 22
Hence the key is probably equal to 8
We can now decrypt the cipher text to reveal:

THERE ONCE WAS AN UGLY DUCKLING WITH FEATHERS ALL STUBBY AND BROWN AND THE OTHER BIRDS SAID IN SO MANY WORDS GET OUT OF TOWN GET OUT, GET OUT, GET OUT OF TOWN AND HE WENT WITH A QUACK AND A WADDLE AND A QUACK IN A FLURRY OF EIDERDOWN THAT POOR LITTLE UGLY DUCKLING WENT WANDERING FAR AND NEAR BUT AT EVERY PLACE THEY SAID TO HIS FACE NOW GET OUT, GET OUT, GET OUT OF HERE AND HE WENT WITH A QUACK AND A WADDLE AND A QUACK AND A VERY UNHAPPY TEAR

2.2 Substitution Cipher

Substitution Cipher
The problem with the shift cipher is that the number of keys is too small.

- We only have 26 possible keys

To increase the number of keys a substitution cipher was invented.
Encryption involves replacing each letter by its permuted version.
Decryption involves use of the inverse permutation.

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
T	M	K	G	O	Y	D	S	I	P	E	L	U	A	V	C	R	J	W	X	Z	N	H	B	Q	F

Hence ATTACK AT DAWN encrypts to TXXTKE TX GTHA

Substitution Cipher
Number of keys is $26! \approx 4.03 \times 10^{26} \approx 2^{88}$

- Feasible to only run a computer on a problem which takes under 2^{80} steps.

This is far too large a number to brute force search using modern computers.
Still we can break these ciphers using statistics of the underlying plaintext language.

Example
XSO MJIWXVL JODIVA STW VAO VY OZJVCO'W LTJDOWX KVAKOAXJTXI-VAW VY SIDS XOKSAVLVDQ IAGZWXJQ. KVUCZXOJW, KVUUZAIKTXIVAW TAG UIKJVOLOKXJVAIKW TJO HOLL JOCJOWOAXOG, TLVADWIGO GIDIXTL UOGIT, KVUCZXOJ DTUOW TAG OLOKXJVAIK KVUUOJKO. TW HOLL TW SVWXIAD UTAQ JOWOTJKS TAG CJVGZKX GONOLVCUOAX KOAXJOW VY UTPVJ DLVMTL KVUCTAIOW, XSO JODIVA STW T JTCIGLQ DJVHIAD AZU-MOJ VY IAAVNTXINO AOH KVUCTAIOW. XSO KVUCZXOJ WKIOAKO GOC-TJXUOAX STW KLVWO JOLTXIVAWSICW HIXS UTAQ VY XSOWO VJDTAI-WTXIVAW NIT KVLLTMVJTXINO CJVPOKXW, WXTYY WOKVAGUOAXW TAG NIWIXIAD IAGZWXJITL WXTYY. IX STW JOKOAXLQ IAXJVGZKOG WONO-JTL UOKSTAIWUW YVJ GONOLVCIAD TAG WZCCVJXIAD OAXJOCJOAOZJITL WXZGOAXW TAG WXTYY, TAG TIUW XV CLTQ T WIDAIYIKTAX JVLO IA XSO GONOLVCUOAX VY SIDS-XOKSAVLVDQ IAGZWXJQ IA XSO JODIVA.

English Letter Frequencies

A	8.99	J	6.51	S	3.26
B	0.00	K	4.81	T	7.60
C	2.95	L	4.34	U	3.57
D	3.10	M	0.62	V	8.06
E	0.00	N	1.40	W	7.13
F	0.00	O	11.63	X	7.75
G	3.72	P	0.31	Y	2.17
H	0.78	Q	1.40	Z	2.17
I	7.75	R	0.00		

Most common bigrams: TA,AX,IA,VA,WX,XS,AG,OA,JO,JV
Most common trigrams: OAX,TAG,IVA,XSO,KVU,TXI,UOA,AXS

Analysis
Since O occurs with frequency 11.63 we can guess O = E
Common trigrams are:

- OAX = E**
- XSO = **E

Common similar trigrams in English are:

- ENT, ETH
- THE

Hence likely to have:

- X = T
- S = H
- A = N

Analysis
From now on we only look at the first two sentences:
THE MJIWTVL JEDIVN HTW VNE VY EZJVCE'W LTJDEWT KVNKENTJTTIVNW VY HIDH TEKHNVLVDQ INGZWTJQ. KVUCZTEJW, KVUU ZNIKTTIVNW TNG UIKJVELEKTJVNIKW TJE HELL JECJEWENTEG, TLVNDWIGE GIDITTL UEGIT, KVUCZTEJ DTUEW TNG ELEKTJVNIK KVUUEJKE.
This was after the changes:

- O = E
- X = T
- S = H
- A = N

Geoff Hamilton

Analysis
Since T occurs as a single letter we must have:

- T = I, or
- T = A

T has probability of 8.07, which is the highest probability left
Therefore, more likely to have:

- T = A

The most frequent

- bigram is TA = AN
- trigram is TAG = AN*

Therefore highly likely that:

- G = D

Analysis
THE MJIWTVL JEDIVN HAW VNE VY EZJVCE'W LAJDEWT KVNKENTJATIVNW VY HIDH TEKHNVLVDQ INDZWTJQ. KVUCZTEJW, KVUUZNIKATIVNW AND UIKJVELEKTJVNIKW AJE HELL JECJEWENTED, ALVNDWIDE DIDITAL UEDIA, KVUCZTEJ DAUEW AND ELEKTJVNIK KVUUEJKE.
This was after the additional changes:

- T = A
- G = D

Analysis
We now look at two letter words:

- IX = *T
- Therefore I must be one of A, I due to English plaintext
- Already have A

Hence:

- I = I
- XV = T*
- Must have, due to English, V = O

Analysis
More two letter words:

- VY = O*
- Hence Y must be one of F, N, R due to English
- Already have N
- Y has probability 1.61
- F has probability 2.28
- R has probability 6.03

Hence Y = F
We also have:

- IW = I*
- Therefore W must be one of F, N, S, T
- Already have F, N, T

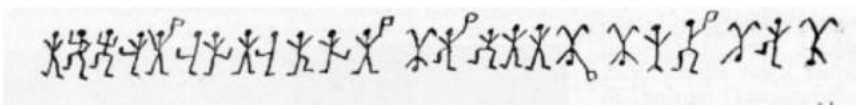
Hence W = S

Analysis
THE MJISTOL JEDION HAS ONE OF EZJOCE'S LAJDEST KONKENTJATIONS OF HIDH TEKHNOLODQ INDZSTJQ. KOUCZTEJS, KOUUZNIKATIONS AND UIKJOELEKTJONIKS AJE HELL JECJESENTED, ALONDSIDE DIDITAL UEDIA, KOUCZTEJ DAUES AND ELEKTJONIK KOUUEJKE.
This was after the additional changes:

- I = I
- V = O
- Y = F
- W = S

It is then easy to see what the underlying plaintext is.

Example Substitution Ciphers
Edgar Alan Poe's "The Gold Bug":
53++!305))6*;4826)4+.)4+);806*;48!8‘60))85;]8*;:+*8!83(88)5*!; 46(;88*96*?;8)*+(;485);5*!2:*+(;4956*2(5*-4)8‘8*;4069285);)6 !8)4++;1(+9;48081;8:8+1;48!85;4)485!528806*81(+9;48;(88;4(+?34;48)4+;161;:188;+?;
Sir Arthur Conan Doyle's "Adventure of the Dancing Men":



2.3 Vigenère Cipher

Vigenère Cipher

The problem with the shift and substitution ciphers was that each plaintext letter always encrypted to the same ciphertext letter.
Hence underlying statistics of the language could be used to break the cipher.
From the early 1800s onwards cipher designers tried to break this link between the plain and cipher texts.
The most famous cipher used during the 1800s is the Vigenère Cipher

- Invented in 1553 by Giovan Batista Belaso.
- Misattributed to the Frenchman Blaise de Vigenère in the 19th century.
- Believed to be unbreakable for a number of years.
- Actually easy to break.

Vigenère Cipher

The Vigenère cipher again identifies letters with the numbers 0-25
The secret key is a short sequence of letters (e.g. a word).
Encryption involves adding the plaintext letter to a key letter, with the key letters used in rotation.
Thus if the key is SESAME, encryption works as follows:

T	H	I	S	I	S	A	T	E	S	T	M	E	S	S	A	G	E	Message
S	E	S	A	M	E	S	E	S	A	M	E	S	E	S	A	M	E	Keystream
L	L	A	S	U	W	S	X	W	S	F	Q	W	W	K	A	S	I	Ciphertext

This is a polyalphabetic substitution cipher

- A will encrypt to a different letter depending on where it is in the message.

Vigenère Cipher

Ciphertext:

CJ UT WFCN LTTF VF AAHGKEE DNH VYC IPSPKGTMV EVLINFA, NC HXS SLGIX QNVYGEM VYYI MVG ZLUHFORRXHB-RIMRXGUZLV TBF KCAXQQD-KJGWERRXHBU ICKHZWKGDGG PFU JGRGIUPR KKCJ RHBVZLJX JKXMGHIUCW XGHQ KFT MKGERN-YWTJR. LX WPKCGTQV RLS MFCEQPVH DP BXKSEKGCZ TNFAZL CH UGVBHCC NPVYGKQ IHKCIBH XOEY MIAST KFGHIIY ANUSTJNPVS, ERPGRWPX JDOS PFRTL, RKXGITZ ERQW, TBF JCRKSV TMGICTRRT WCELKTGHU. FSG ISTJMCTZ CEB TVCPFKXV ZKMCH KSNP KDKS CEB BHFG FL DNF CSGABHA KM AXH ULAW XHJVPTTZ ERPG-BST GGVXCPJ KTWWCKC PM O FZQITBEV UWTH YV SHXR VF BD PWVY DPVS-VF-DPVS OVCIBBIJ, NPIST UMRNAGERH, TBF R DXKA JRLSLVCBC. JGTQIRJGOVVJN, MVG KCRABKTYA PWBRPSKM GEYQEWPX PTFCVV ADEZCSMGTHKFLH BG HFSCWSF FL QKCCUAPLHKEE TOSTPRWBBI RQ HX-EWVLRXG QW XTKCU RLS HBGJ RWTH QECÕH HKP UMV PCWCBCÕM FGTMVGWBV. UWTH KJ RD WWUKGCZIKJF P WWIZRPE RQCJPK KJVL XM WU RQ TTGKCW GXDTFBJVWDCC PL HJV QEHYGE UDKR? JFU SH KG TMCOSTJC EKWXRRTEM YYCC XJGIW HRZNRZAX WU SMJGQGU MUY O URRTEZKKC PGR UDCPKSF FTTK OP VLIBFG TMCMWPVLI?

Vigenère Cipher

- Finally cracked by Charles Babbage in 1854.
- First published attack by Friedrich Kasiski in 1863.

- First we need to look for repeated sequences of characters.
  - Recall English has a large repetition of certain bigrams
  - These are likely to match up to the same two letters in the key every so often
  - By looking for the distance between two repeated sequences we can guess the length of the keyword.
- Each distance should be a multiple of the keyword.
  - Taking the gcd of all distances between sequences should give the keyword length.

Vigenère Cipher
First sentence is:
CJ UT WFCN LTTF VF AAHGKEE DNH VYC IPSPKGTMV EVLINFA, NC HXS SLGIX QNVYGEM VYYI MVG ZLUHFORRXHB-RIMRXGUZLV TBF KCAXQQD-KJGWERRXHBU ICKHZWKGDGG PFU JGRGIUPR KKCJ RHBVZLJX JKXMGHI-UCW XGHQ KFT MKGERN-YWTJR.
Distance between occurrences of RR: 30
Distance between occurrences of KG: 96, 46
We have:

- gcd(30,96)=6, gcd(30,46)=2, gcd(96,46)=2

Unlikely to have a keyword of length 2

- Guess keyword is of length 6

Vigenère Cipher
Now we take every sixth letter and look at the statistics just as we did for a shift cipher to deduce the first letter of the keyword.
This gives us the following:

A	1.49	J	3.73	S	0.75
B	1.49	K	8.96	T	7.46
C	8.96	L	0.00	U	8.21
D	1.49	M	0.00	V	8.21
E	6.72	N	2.99	W	2.24
F	4.48	O	1.49	X	0.75
G	11.19	P	8.21	Y	1.49
H	1.49	Q	4.48	Z	0.00
I	2.99	R	0.75		

We look for a low value and then three high ones, corresponding to Q, R, S, T $\Rightarrow$ Key is 2 or C

Vigenère Cipher
Then we take every sixth letter starting from the second one and repeat to find the second letter of the keyword:

A	0.00	J	8.21	S	2.24
B	0.75	K	9.70	T	3.73
C	5.22	L	2.24	U	3.73
D	1.49	M	0.75	V	10.44
E	7.46	N	1.49	W	0.75
F	11.19	O	0.00	X	2.99
G	0.75	P	2.24	Y	4.48
H	0.00	Q	0.00	Z	3.73
I	4.48	R	11.94		

We look for a low value and then three high ones, corresponding to Q, R, S, T $\Rightarrow$ Key is 17 or R
And so on to determine the six letters of the keyword: CRYPTO

Vigenère Cipher
The underlying plaintext is then found to be:

AS WE DRAW NEAR TO CLOSING OUT THE TWENTIETH CENTURY, WE SEE QUITE CLEARLY THAT THE INFORMATION-PROCESSING AND TELECOMMUNICATIONS REVOLUTIONS NOW UNDERWAY WILL CONTINUE VIGOROUSLY INTO THE TWENTY-FIRST. WE INTERACT AND TRANSACT BY DIRECTING FLOCKS OF DIGITAL PACKETS TOWARDS EACH OTHER THROUGH CYBERSPACE, CARRYING LOVE NOTES, DIGITAL CASH, AND SECRET CORPORATE DOCUMENTS. OUR PERSONAL AND ECONOMIC LIVES RELY MORE AND MORE ON OUR ABILITY TO LET SUCH ETHEREAL CARRIER PIGEONS MEDIATE AT A DISTANCE WHAT WE USED TO DO WITH FACE -TO-FACE MEETINGS, PAPER DOCUMENTS, AND A FIRM HANDSHAKE. UNFORTUNATELY, THE TECHNICAL WIZARDRY ENABLING REMOTE COLLABORATIONS IS FOUNDED ON BROADCASTING EVERYTHING AS SEQUENCES OF ZEROS AND ONES THAT ONES OWN DOG WOULDNT RECOGNIZE. WHAT IS TO DISTINGUISH A DIGITAL DOLLAR WHEN IT IS A S EASILY REPRODUCIBLE AS THE SPOKEN WORD? HOW DO WE CONVERSE PRIVATELY WHEN EVERY SYLLABLE IS BOUNCED OFF A SATELLITE AND SMEARED OVER AN ENTIRE CONTINENT?

2.4 Vernam Cipher

Vernam Cipher (One-Time Pad)

- Gilbert Vernam patented this cipher in 1917 for encryption and decryption of telegraph messages.
- Used extensively during the first world war.
- To send a binary string m you need a key k as long as the message.
- Each key k is uniformly selected at random and can be used only once - hence one-time pad.
- Encryption: $c = m \oplus k$.
- Decryption: $m = c \oplus k$.
- $\oplus$ is exclusive-or (XOR):

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

Vernam Cipher (One-Time Pad)

Plaintext:	o	n	e	t	i
In binary:	01101111	01101110	01100101	01110100	01101001
Key:	01011100	01010001	11100000	01101001	01111010
Ciphertext:	00110011	00111111	10000101	00011101	00010011

Plaintext:	m	e	p	a	d
In binary:	01101101	01100101	01110000	01100001	01100100
Key:	11111001	11000110	01011010	10110001	01110011
Ciphertext:	10010100	10100011	00101010	11010000	00010111

Vernam Cipher (One-Time Pad)
Gives perfect secrecy, provided:

- Key is truly random.
- Key is at least as big as plaintext (not practical).
- Key is not reused e.g. consider a 'two-time' pad:
  - First encryption: $c_1 = m_1 \oplus k$.
  - Second encryption: $c_2 = m_2 \oplus k$.
  - $c_1 \oplus c_2 = m_1 \oplus m_2$
  - Vulnerable to frequency analysis.

2.5 Rotor Machines

Rotor Machines
With the advent of the 1920s people saw the need for a mechanical encryption device. Taking a substitution cipher and then rotating it became seen as the ideal solution.

- This had actually been used during the American Civil War.
- But now this could be done more efficiently.

The rotors could be implemented using wires and then encryption can be done mechanically using an electrical circuit.

- By rotating the rotor we obtain a new substitution cipher.
- Transmission of message still done manually using Morse Code.

Rotor Machines
To encrypt the first letter we use the substitutions:

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
T	M	K	G	O	Y	D	S	I	P	E	L	U	A	V	C	R	J	W	X	Z	N	H	B	Q	F

To encrypt the second letter we use the substitutions:

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
M	K	G	O	Y	D	S	I	P	E	L	U	A	V	C	R	J	W	X	Z	N	H	B	Q	F	T

To encrypt the third letter we use the substitutions:

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
K	G	O	Y	D	S	I	P	E	L	U	A	V	C	R	J	W	X	Z	N	H	B	Q	F	T	M

and so on.

Enigma
The most famous of these machines was the Enigma.

Enigma
We shall describe the most simple version of Enigma.
Used three such rotors chosen from a set of five:

- EKMFLGDQVZNTOWYHXUSPAIBRCJ Rotor One
- AJDKSIRUXBLHWTMCQGZNPYFVOE Rotor Two
- BDFHJLCPRTXVZNYEIWGAKMUSQO Rotor Three

Geoff Hamilton

- ESOVPZJAYQUIRHXLNFTGKDCMWB Rotor Four
- VZBRGITYUPSDNHLXAWMJQOFECK Rotor Five

The order of the rotors in the machine is important.
Number of ways of choosing the rotors is $5 \times 4 \times 3 = 60$

Enigma
Each rotor has an initial starting position

- Number of starting positions is $26^3 = 17576$

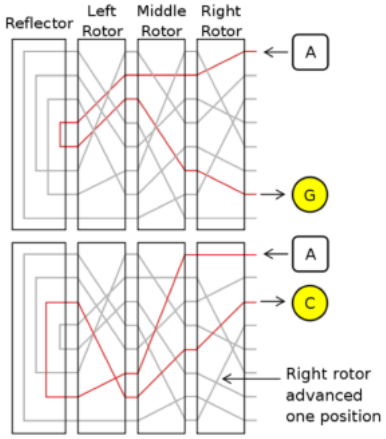
The plugboard, which maps pairs of letters to each other before the rotors encrypt them, adds a significant amount of complexity.
If the plugboard connects ten pairs of letters, this gives the following number of possible configurations:

$$\frac{26!}{6! \times 10! \times 2^{10}} = 150738274937250$$

The total possible configurations of an Enigma machine can now be obtained by multiplying all of these numbers together:

$$158962555217826360000$$

Enigma
A given plaintext letter would encrypt to a different ciphertext letter on each press of the keyboard:



Enigma
To use Enigma the Germans had a day setting of:

- Plugs to use
- Rotors to use, and their order
- Ring settings
- Initial rotor settings

However, we really need to change the key on each message.

- This is where the flaw in Enigma resulted.
- Not in the design, but in the use of the system

Enigma
So sender of message would choose a new set of rotor positions for this message (A, F, G say) and then encrypt these twice using the day settings (G, H, K, L, P, T say).
He would then set the machine to the message setting and encrypt the message.
The receiver would decrypt the message key, reset his machine and then decrypt the rest of the message.
It is the repeating of the message rotor settings which led to to the Polish and British cryptographers being able to break Enigma in WWII.

Enigma
Some weaknesses in the usage of the Enigma machine were as follows:

1. Random message keys were not truly 'random': operators often didn't bother to change the message keys between messages, or would only use a small subset of message keys.
2. Messages used common phrases (known as 'cribs'): operators frequently encrypted a certain phrase that demonstrated their appreciation for Hitler at the end of every message.
3. The Germans sent out a daily weather report: this message was sent every day at 6 A.M. and had a specific templated format.
4. The Germans had complete confidence in the Enigma: except for their Navy, they had complete trust in the device and its invincibility and didn't try to tighten up security or change their ways.

Enigma
However, it is not only bad usage which led to the breaking of Enigma.
The main problem with Enigma is that the plugboard and the rotors are orthogonal in some sense.

- In particular the plugboard acts on the permutation given by the rotors as conjugation.
- This means that many of the underlying statistics of the rotor settings are not disguised by the plugboard
- Once the rotor settings are found the plugboard settings can be found easily.

Enigma
Reflector weakens Enigma: no difference between encryption and decryption.

- Problem 1: encryption becomes involutary, i.e. if K $\rightarrow$ T, then T $\rightarrow$ K
- Problem 2: no letter is encrypted to itself (electricity can't go same way back)

Heavy reduction of encryption alphabet - violation of Kerckhoff's principle:

- Security of Enigma depended on wiring of rotors
- Wiring was part of algorithm, not part of key
- Wiring never changed from 1920s until 1945

2.6 Summary

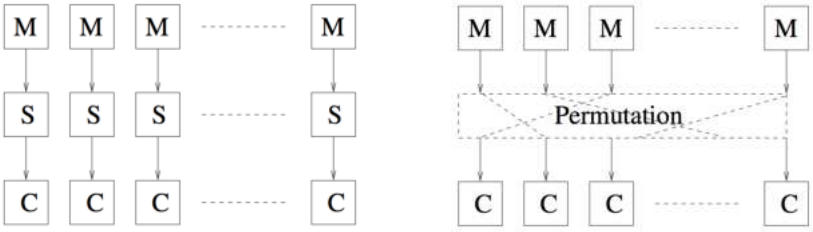
Historical Ciphers
The historical ciphers we have studied can be categorised as follows:

- monoalphabetic: shift cipher, substitution cipher
- polyalphabetic: Vigenère, Enigma

Simplest ciphers could be easily broken unless encryption rule was kept secret (security by obscurity).
Simple ciphers could easily be broken because they did not conceal the language characteristics (encryption functions were not complicated enough).
Enigma was the first step towards more mature designs (secure and efficient).

Substitution Permutation

Historical Ciphers



Geoff Hamilton

- Substitution: provides confusion, i.e. it makes the relationship between key and ciphertext complex.
- Permutation: provides diffusion, i.e. each bit (symbol) of the ciphertext depends on many (if possible all) bits (symbols) of the plaintext.

Modern Ciphers
Modern ciphers use both substitution and permutation.
They typically encrypt small blocks of plaintext at a time.
They can be efficiently implemented on different types of platforms.
Ciphers that apply an encryption function to small blocks of plaintext at a time are called block ciphers.
Ciphers that use substitution and permutation are called substitution-permutation networks.

2.7 Security

Computational Security
A system is computationally secure if the best algorithm for breaking it requires N operations.

- Where N is a very big number
- No practical system can be proved secure under this definition.

In practice we say a system is computationally secure if the best known algorithm for breaking it requires an unreasonably large amount of computer time.

Computational Security
Another practical approach is to reduce a well studied hard problem to the problem of breaking the system.

- For example, the system is secure if a given integer n cannot be factored.

Systems of this form are often called provably secure.

- However, we only have a proof relative to some hard problem.
- Not an absolute proof.

Essentially bounding the computational power of the adversary.

- Even if the adversary has limited (but large) resources they still will not break the system.

Computational Security
When considering schemes which are computationally secure:

- We need to be careful about the key sizes.
- We need to keep abreast of current algorithmic developments.
- At some point in the future we should expect our system to be broken (may be many millennia hence though).

Most schemes in use today are computationally secure.

Unconditional Security
For unconditional security we place no bound on the computational power of the adversary.
In other words, a system is unconditionally secure if it cannot be broken even with infinite computing power.

- Some systems are unconditionally secure.

Other names for unconditionally secure are:

- Perfectly secure
- Information-theoretically secure
- Semantically secure

Unconditional Security
A cryptosystem has unconditional security iff:

$$p(P = m \mid C = c) = p(P = m)$$

for all $m \in P$ and $c \in C$.
That is, the probability that the plaintext is m given that the ciphertext c is observed is the same as the probability that the plaintext is m without seeing c.
In other words knowing c reveals no information about m.
Unconditional security implies that: $|keys| \geq |messages|$.
The use of such large key spaces is impractical in practice.

Key Distribution
Perfect secrecy implies length of key is at least length of plaintext.
Key distribution becomes the major problem.
Aim of modern cryptography is to design systems where:

- one key can be used many times
- a short key can encrypt a long message.

Such systems will not be unconditionally secure, but should be at least computationally secure.

Examples
Of the ciphers we have seen or will see later on, the following are not computationally secure:

- Caesar cipher
- Substitution cipher
- Vigenère cipher

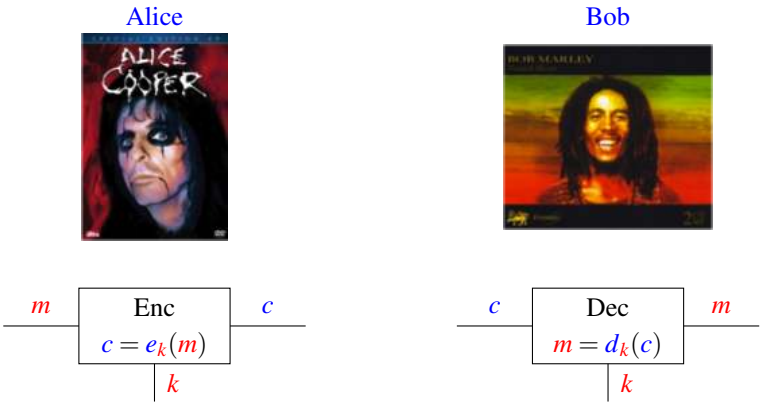
The following are computationally secure but not unconditionally secure:

- DES(?)-AES
- RSA

The Vernam cipher (one-time pad) is unconditionally secure if used correctly.

3 Symmetric Cryptography

Symmetric Cryptography

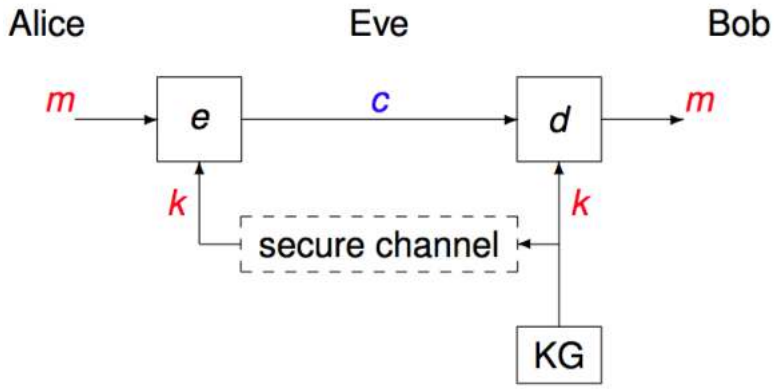


Symmetric cryptography uses the same secret key k for encryption and decryption.

Symmetric Ciphers

- Typically the same (reversible) algorithm is used for encryption and decryption.
- The algorithm(s) used may or may not be kept secret, but in either case, the strength of the algorithm should rest on the size of the keys and the design of the algorithm.
  - In theory, we can recover a symmetric key by performing an exhaustive search.
  - For a secure symmetric cipher, there should not be a more efficient algorithm for finding a key.
- There are numerous symmetric ciphers in use.
  - DES, triple-DES, IDEA, Skipjack, CAST, Blowfish, AES, ...
  - They have different key sizes (e.g., DES - 56 bits, IDEA - 128 bits, AES - 128, 192 or 256 bits).
  - Some are subject to patents in some countries.

Symmetric Key Cryptography



m = plaintext, c = ciphertext, k = key, KG = key generator.

Symmetric Key Cryptography
We write $c = e_k(m)$, where:

- m is the plaintext,
- e is the encryption function,
- k is the secret key,
- c is the ciphertext.

Decryption is given by $m = d_k(c)$.
Both sides need to know the key k, but k needs to be kept secret.

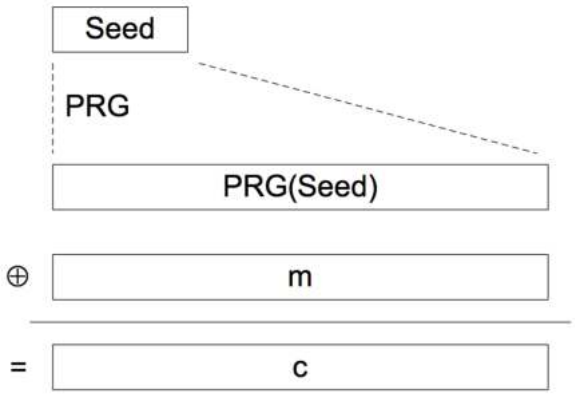
- secret-key, single-key or one-key algorithms.

Symmetric Ciphers
There are two types of symmetric cipher:

- Block ciphers that encrypt one block of data at a time.
  - Typically, blocks consist of 64 bits (8 bytes) or 128 bits (16 bytes) of data.
  - The same plaintext block always encrypts to the same ciphertext block.
  - In certain circumstances, this is a weakness and we need to use some sort of feedback to disguise patterns in the plaintext. This leads to different modes of operation.
- Stream ciphers that encrypt an arbitrary stream of data.
  - Depending on the cipher, the data may consist of a stream of bits or a stream of bytes.
  - Each plaintext bit (or byte) encrypts to a different cipher text bit (or byte) depending on what data occurred earlier in the stream.
  - It is possible to use a block cipher to implement a stream cipher.

3.1 Stream Ciphers

Stream Ciphers
Basic idea: replace the random key in the one-time pad by a pseudo-random sequence, generated by a cryptographic pseudo-random generator (PRG) that is 'seeded' with the key.



PRGs
PRG requirements:

- Randomness
  - Uniformity, scalability, consistency
- Unpredictability
  - Cannot determine what next bit will be despite knowledge of the algorithm and all previous bits.
- Unreproducable
  - Cannot be reliably reproduced.

Characteristics of the seed:

- Must be kept secret
- If known, adversary can determine output
- Must be random or pseudorandom number

Linear Congrentual Generator
Common iterative technique using:

$$X_{n+1} = (aX_n + c) \bmod m$$

Given suitable values of parameters can produce a long random-like sequence
Suitable criteria to have are:

- Function generates a full period
- Generated sequence should appear random
- Efficient implementation with 32-bit arithmetic

Note that an attacker can reconstruct sequence given a small number of values - this can be made harder in practice.

Blum Blum Shub Generator
Find two large primes p, q congruent to 3 (mod 4) where $m = p \times q$
Seed: $X_0 = k^2 \bmod m$ (k relatively prime to m)
Use least significant bit from iterative equation:

$$X_{n+1} = X_n^2 \bmod m$$

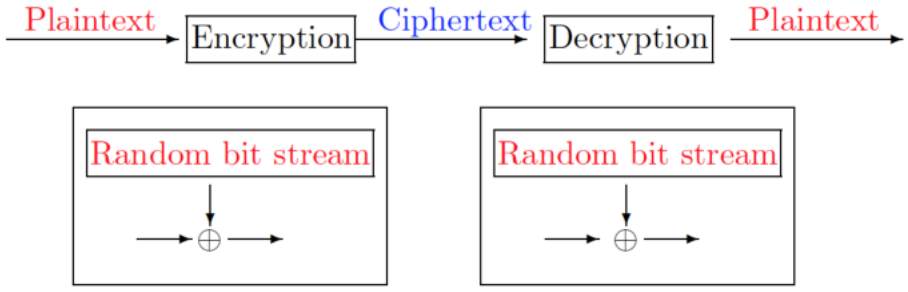
- Unpredictable given any run of bits
  - Passes next-bit test
- Security rests on difficulty of factoring m
- Slow since very large numbers must be used
  - Too slow for cipher use, but good for key generation

Real Random Numbers
In some cryptographic implementations, real random numbers are used.
Such numbers can be generated from random physical events.

- Measuring radioactive decay.
- Measuring the time to read blocks of data from a disk - this is affected by air turbulence and has a random component.
- Measuring the time between key strokes on a key board.
- Using the least significant bits of a computer's clock.

Using real random numbers is not usually a practical option.

Stream Ciphers
The stream cipher process is as follows:

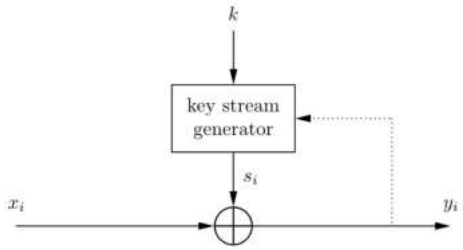


Stream Ciphers
Thus we have $c_i = m_i \oplus k_i$, where:

- m_i are the plaintext bits/bytes.
- k_i are the keystream bits/bytes.
- c_i are the ciphertext bits/bytes.

This means $m_i = c_i \oplus k_i$ and thus decryption is the same as encryption.
Encryption/decryption can be very fast.
No error propagation: one error in ciphertext gives one error in plaintext.
Loss of synchronisation means decryption fails for remaining ciphertext.
No protection against message manipulation.
Same key used twice gives same keystream.

Stream Ciphers



Stream ciphers can be synchronous or asynchronous.
In a synchronous stream cipher, the keystream depends only on the key.
In an asynchronous stream cipher, the keystream depends on the ciphertext (the dotted feedback on the diagram above).

Stream Ciphers
Synchronous stream ciphers:

- Sender and receiver must synchronise their actions for decryption to be successful.
- If digits are added or removed from the message during transmission, synchronisation is lost.
- To restore synchronisation, various offsets can be tried systematically to obtain the correct decryption.
- If a digit is corrupted in transmission only a single digit in the plaintext is affected.
- Errors do not propagate to other parts of the message.
- Useful when the transmission error rate is high.
- Very susceptible to active attacks.

Stream Ciphers
Asynchronous stream ciphers:

- Receiver will automatically synchronise with the keystream generator after receiving n ciphertext digits.
- This makes it easier to recover any digits dropped or added to the message stream.
- Single digit errors are limited in their effect, affecting only up to n plaintext digits.

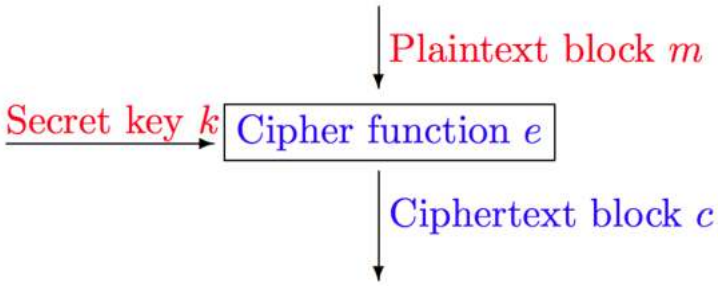
Stream Ciphers
For the stream cipher to be secure, the keystream must:

- Look random, i.e. pass pseudo-random tests.
- Be unpredictable, i.e. have a long period.
- Have large linear complexity (see textbooks).
- Have low correlation between key bits and keystream bits.

Furthermore, the keystream should be efficient to generate.
Most stream ciphers are based on a non-linear combination of LFSRs (Linear Feedback Shift Registers).

3.2 Block Ciphers

Block Ciphers
Plaintext is divided into blocks of fixed length and every block is encrypted one at a time.
A block cipher is a set of 'code books' and every key produces a different code book.
The encryption of a plaintext block is the corresponding ciphertext block entry in the code book.



Block Ciphers
We have $c = e_k(m)$ where:

- m is the plaintext block
- k is the secret key
- e is the encryption function
- c is the ciphertext

In addition we have a decryption function d such that:

$$m = d_k(c)$$

Padding
Problem: suppose length of plaintext is not multiple of block length.

- Last block m_t only contains $k < n$ bits.

Padding schemes:

- Append $n-k$ zeroes to last block. Trailing 0-bits in the plaintext cannot be distinguished from this padding, so an extra block has to be added giving the length of the message in bits.
- Append 1 and $n-k-1$ 0-bits. If $k = n$, then create extra block starting with 1 and remaining $n-1$ bits 0.

Block Size
The block size n needs to be reasonably large, $n > 64$ bits, to avoid:

- Text dictionary attacks: plaintext-ciphertext pairs for fixed key.
- Matching ciphertext attacks: uncover patterns in plaintext.

With a very large block size, the cipher becomes inefficient to operate.

- Plaintexts will then need to have a lot of padding added before being encrypted.

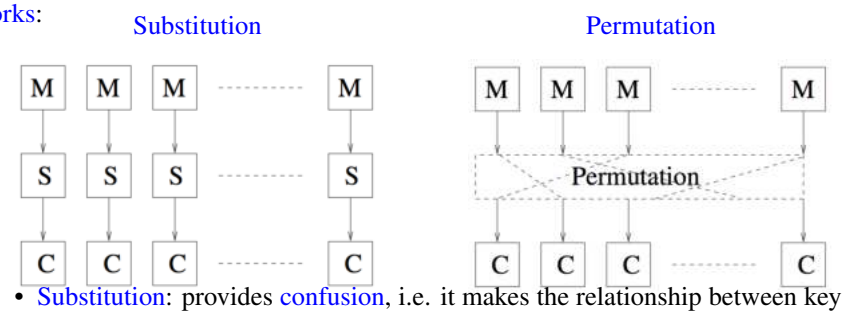
The preferred block size is a multiple of 8 bits as it is easier for implementation since most computer processors handle data in multiples of 8 bits.

Iterated Block Ciphers
An iterated block cipher involves repeated use of a round function.
The idea is to make a strong encryption function out of a weaker round function (easy to implement) by repeatedly using it.
The round function takes an n-bit block to an n-bit block.
Parameters: number of rounds r, blocksize n, keysize s.
Each use of the round function employs a subkey:

$$k_i \text{ for } 1 \leq i \leq r$$

derived from the key k.
For every subkey the round function must be invertible; if not decryption is impossible.

Substitution-Permutation Networks
Ciphers that use substitution and permutation are called substitution-permutation networks:



- Substitution: provides confusion, i.e. it makes the relationship between key and ciphertext complex.
- Permutation: provides diffusion, i.e. each bit (symbol) of the ciphertext depends on many (if possible all) bits (symbols) of the plaintext.

Substitution
A substitution box (S-box) substitutes a small block of input bits with another block of output bits.
An S-box can be considered as secure if changing one input bit will change about half of the output bits on average, exhibiting what is known as the avalanche effect.
The following is an example of a S-Box:

	00	01	10	11
0	10	01	11	00
1	00	10	01	11

For input bits $x_0x_1x_2$, x_0 gives the row, and x_1x_2 gives the column.
So, for example, if the input to this S-Box is 010, then the output is 11.

Permutation
A permutation box (P-box) is a permutation of all the bits.
It takes the outputs of all the S-boxes of the current round, permutes the bits, and feeds them into the S-boxes of the next round.
The following is an example of a P-Box:

01	15	02	13	06	17	03	19	09	04	21	11
14	05	12	16	18	07	24	10	23	08	22	20

The table gives the position the input bit is mapped onto in the output.
So, for example, input bit 01 is mapped to output bit 01, input bit 15 is mapped to output bit 02, input bit 14 is mapped to output bit 13, etc.

Data Encryption Standard (DES)
First published in 1977 as a US Federal standard.
Known as Data Encryption Algorithm DEA (ANSI) or DEA-1 (ISO).

- A de-facto international standard for banking security.
- An example of a Feistel Cipher.
- Most analyzed cryptographic algorithm.

Short history of DES:

- Work started in early 1970's by IBM.
- Based on IBM's Lucifer, but amended by NSA.
- Design criteria were kept secret for more than 20 years.
- Supposed to be reviewed every 5 years.

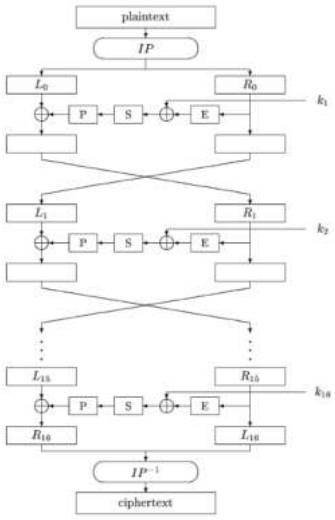
Data Encryption Standard (DES)
Block length is 64 bits, key length 56 bits.
Feistel Cipher:

- Initial permutation of bits.
- Split into left and right half.
- 16 rounds of identical operations, depending on round key.
- Inverse initial permutation.

Round transformation:

- Linear expansion: 32 bits $\rightarrow$ 48 bits.
- XOR with subkey of 48 bits (key schedule selects 48 bits of key k).
- 8 parallel non-linear S-boxes (6 input bits, 4 output bits).
- Permutation of the 32 bits.

Data Encryption Standard (DES)



Data Encryption Standard (DES)
Each DES round consists of the following six stages:
1. Expansion Permutation:

- Right half (32 bits) is expanded (and permuted) to 48 bits.

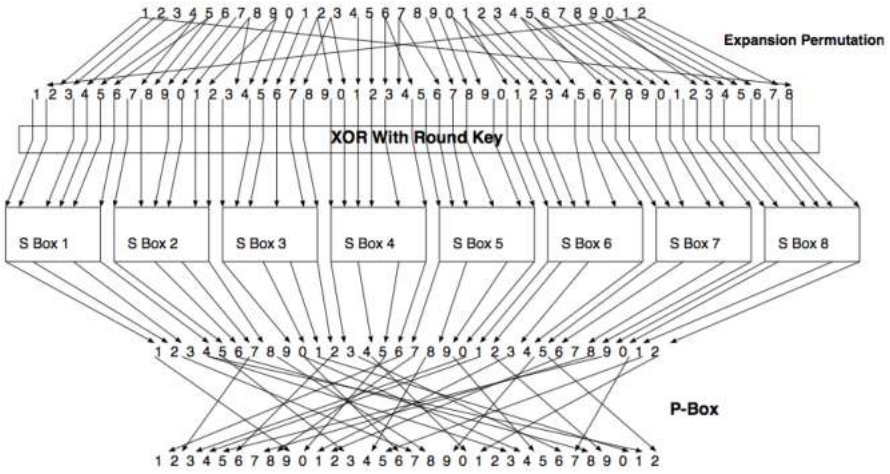
- Diffuses relationship of input bits to output bits.
- Means one bit of input affects two substitutions in output.
- Spreads dependencies.

2. Use of Round Key:
   - 48 bits are XOR-ed with the round key (48 bits).
3. Splitting:
   - Result is split into eight lots of six bit values.

Data Encryption Standard (DES)

4. S-Box:
   - Each six bit value is passed into an S-box to produce a four bit result in a non-linear way. (S = Substitution)
5. P-Box:
   - 32 bits of output are combined and permuted. (P = Permutation)
6. Feistel Part:
   - Output of f is XOR-ed with the left half resulting in new right half.
   - New left half is the old right half.

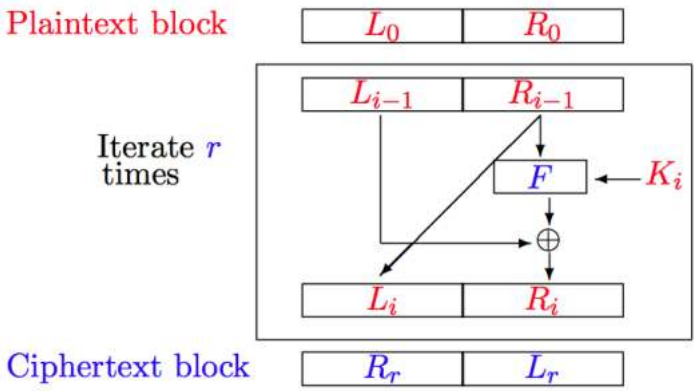
Data Encryption Standard (DES)



Data Encryption Standard (DES)
S-Boxes represent the non-linear component of DES.
There are eight different S-boxes.
The original S-boxes proposed by IBM were modified by NSA.
Each S-box is a table of 4 rows and 16 columns

- The 6 input bits specify which row and column to use.
- Bits 1 and 6 generate the row.
- Bits 2-5 generate the column.

Feistel Cipher



Feistel Cipher
The round function is invertible regardless of the choice of f .
Encryption round is:

- $L_i = R_{i-1}$
- $R_i = L_{i-1} \oplus f(R_{i-1}, k_i)$

Remark: in last step, the left half and right half are swapped:

- Decryption is achieved using the same r-round process.
- But with round keys used in reverse order, i.e. k_r first and k_1 last.

Feistel Cipher
Same algorithm can be used for decryption since we are using the Feistel structure:

$$L_{i-1} \oplus f(R_{i-1}, k_i) \oplus f(R_{i-1}, k_i) = L_{i-1}$$

Remark: for decryption the subkeys are used in the reverse order.
Note that the effect of IP^{-1} is cancelled by IP.
Also note that R_{16}, L_{16} is the input of encryption since halves were swapped and that swapping is necessary.

Security of DES
Exhaustive key search (1 or 2 known plaintext/ciphertext pairs).

- Number of possibilities for DES is $2^{56} = 7.2 \times 10^{16}$.

Software (PC with 3.2 GHz Processor): 2^{48} encryptions per year.

- 2^{23} keys per second, $2^{16.4}$ seconds per day, $2^{8.5}$ days per year.
- 1 PC: 125 years, 125 PC's: 1 year, 1500 PC's: 1 month.

Hardware (ASIC), Cost = $250,000, 'Deep Crack' (EFF, 1998).

- 1 key in 50 hours, less than $500 per key.
- Time halfed by working in conjunction with `distributed.net`
- For $1M: 1 key in 1/2 hour.

Hardware (FPGA), Cost = $10,000, 'COPACABANA', 2006

- 9 days (later reduced to 6 days)
- Reduced to less than one day by successor machine 'RIVYERA'

Security of DES
Export from US: 40-bit keys (SSL, Lotus Notes, S/MIME).

- Obviously much less secure.

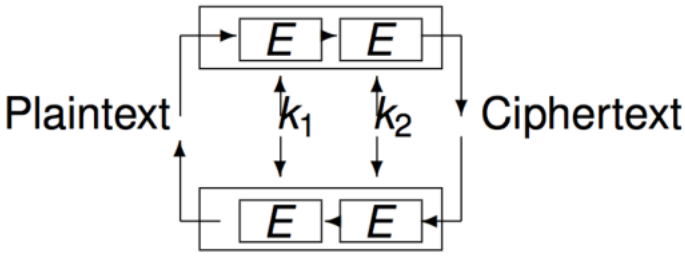
Moore's 'law':

- Computing power doubles every 18 months.
- After 21 years the effective key size is reduced by 14 bits.

Long term: key length and block length of 128 bits.

Multiple Encryption
DES is a 'standard': neither key size nor block size nor number of rounds can be changed (easily).
Remember: more rounds bring more security.
Idea: iterating the entire cipher might bring more security.
Double encryption, triple encryption, quadruple encryption, etc.
What makes most sense?

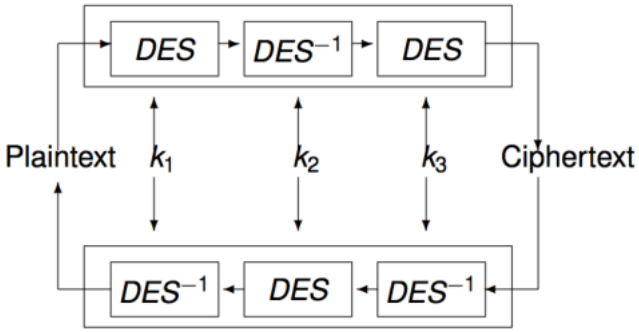
Double Encryption



Time-memory trade-off via a meet-in-the-middle attack is possible:

- Pre-compute for all keys $k_1 : C_1 = E_{k_1}(P)$.
- Given a ciphertext C: invert C and compare with list of C_1. Hit indicates k_2 and gives k_1. Check with one more C.
- Double encryption does not provide double security.

Triple DES



Invented to get around the problem of a short key.
Involves use of 2 or 3 DES keys.

Advanced Encryption Standard (AES)
January 1997: NIST call for algorithms to replace DES.

- Block cipher: 128-bit blocks, 128/192/256-bit keys.
- Strength $\approx 3 \times$ DES, much more efficient.
- Documentation, reference C code, optimised C and JAVA code, test vectors.
- Designers give up all intellectual rights.

Open process: public comments, international submissions.
Website: `http://www.nist.gov/aes/`

Advanced Encryption Standard (AES)
Standard FIPS-197 approved by NIST in November 2001.
Official scope was limited:

- US Federal Administration used AES as Government standard from 26 May 2002.
- Documents that are 'sensitive but not classified'.
- 2003: NSA has approved AES-128 also for secret information, and AES with key sizes larger than 128 for top secret information.
- Significance is huge: AES is the successor of DES.
- Major factors for quick acceptance:
  - No royalties.
  - High quality.
  - Low resource consumption.

Rijndael
Block length, key length: vary between 128 - 256 bits.
Number of rounds is 10/12/14 depending on block and key length.
Uniform and parallel round transformation, composed of:

- Byte substitution.
- Shift rows.
- Mix columns.
- Round key addition.

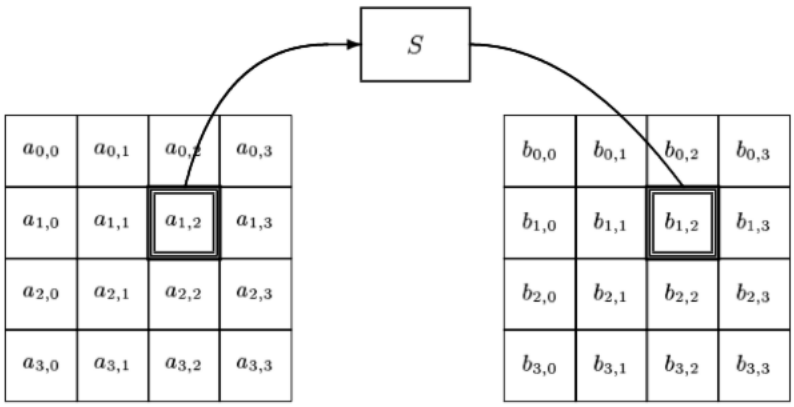
Sequential and light-weight key schedule.
No arithmetic operations.

Rijndael
Plaintext block normally is 128 bits or 16 bytes $m_0, \ldots, m_{15}$.
Key normally is 128/192/256 bits or 16/24/32 bytes $k_0, \ldots, k_{31}$.
Both are represented as rectangular array of bytes.

m_0	m_4	m_8	m_{12}
m_1	m_5	m_9	m_{13}
m_2	m_6	m_{10}	m_{14}
m_3	m_7	m_{11}	m_{15}

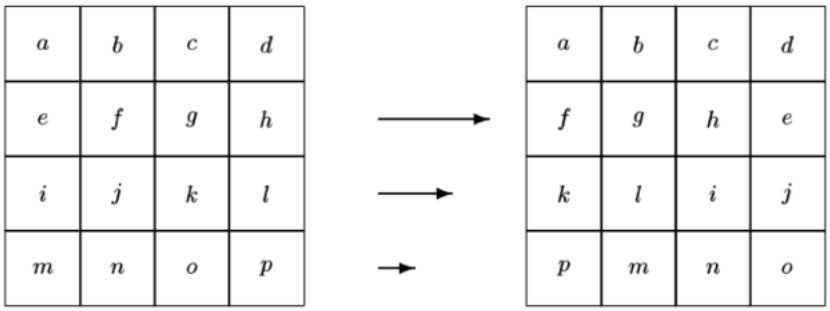
k_0	k_4	k_8	k_{12}	k_{16}	k_{20}
k_1	k_5	k_9	k_{13}	k_{17}	k_{21}
k_2	k_6	k_{10}	k_{14}	k_{18}	k_{22}
k_3	k_7	k_{11}	k_{15}	k_{19}	k_{23}

Rijndael: Byte Substitution
State matrix is transformed byte by byte.
S-box is invertible, else decryption would not work.
Only one S-box for the whole cipher (simplicity).



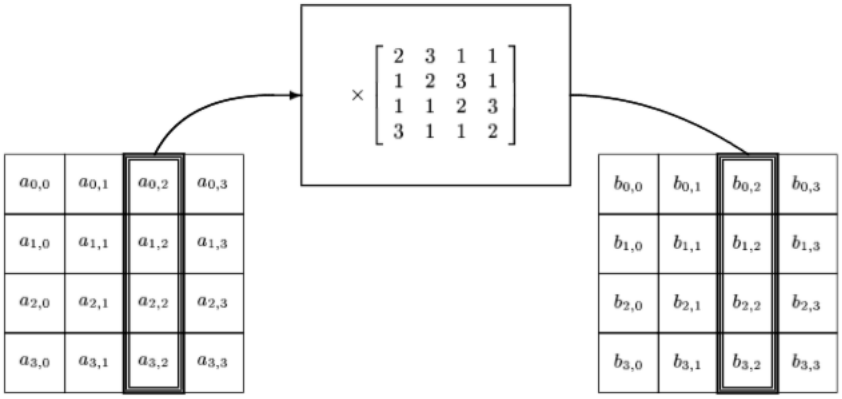
Rijndael: Shift Rows
Rows shifted over different offsets (depending on block length).

Purpose: diffusion over the columns.



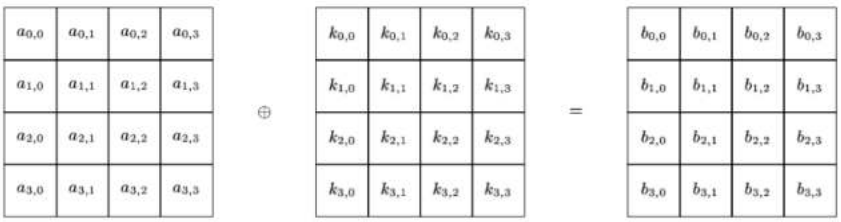
Rijndael: Mix Columns

- Bytes in columns are combined linearly.
- Good diffusion properties over rows.

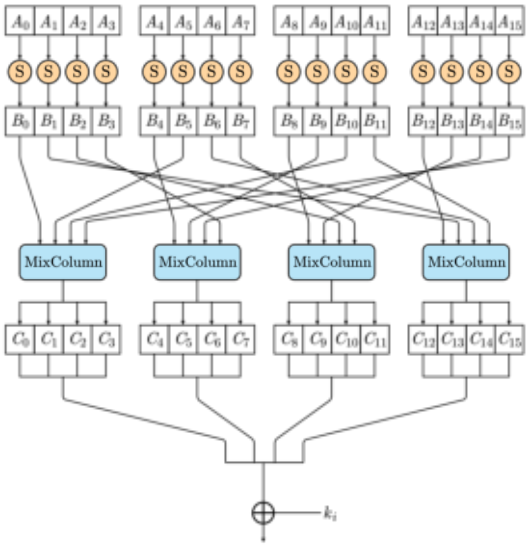


Rijndael: Round Key Addition

Round key is simply XOR-ed with state matrix.



Rijndael: Round Function

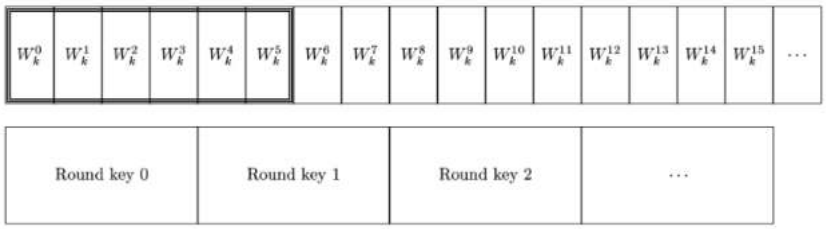


Rijndael: Pseudo-code
Rijndael with 10 rounds is described by the following code:

```
AddRoundKey(S,K[0]);
for (i = 1; i <= 9; i++)
   {
      SubBytes(S);
      ShiftRows(S);
      MixColumns(S);
      AddRoundKey(S,K[i]);
   }
SubBytes(S);
ShiftRows(S);
AddRoundKey(S,K[10]);
```

Rijndael: Key Schedule

Example: key of 192 bits or 6 words of 32 bits.



$$W_k^{6n} = W_k^{6n-6} \oplus f(W_k^{6n-1})$$
$$W_k^i = W_k^{i-6} \oplus W_k^{i-1}$$

Cipher key expansion can be done just-in-time: no extra storage required.

Comparing AES and DES

DES:

- S-P Network, iterated cipher, Feistel structure
- 64-bit block size, 56-bit key size
- 8 different S-boxes
- non-invertible round
- design optimised for hardware implementations
- closed (secret) design process

AES:

- S-P Network, iterated cipher
- 128-bit block size, 128-bit (192, 256) key size
- one S-box
- invertible round
- design optimised for byte-orientated implementations
- open design and evaluation process

Non-Linearity in Block Ciphers

Non-linearity is often achieved in block ciphers within the S-Boxes.
This can be achieved by implementing the S-Boxes as non-linear functions.
In DES, the S-Boxes were actually designed by hand. The NSA found that the original S-Boxes designed by IBM did still have a linear relationship between the inputs and outputs, so they slightly modified them to strengthen them against differential cryptanalysis.
In AES, the S-Box is implemented as a non-linear mathematical function (using modular arithmetic), thus ensuring its resistance to cryptanalysis.

Block vs Stream Ciphers
Which is best?
Block ciphers:

- More versatile: can be used as stream cipher.
- Standardisation: DES and AES + modes of operation.
- Very well studied and accepted.

Stream ciphers:

- Easier to do the maths.
- Either makes them easier to break or easier to study.
- Supposedly faster than block ciphers (less flexible).

3.3 Modes of Operation

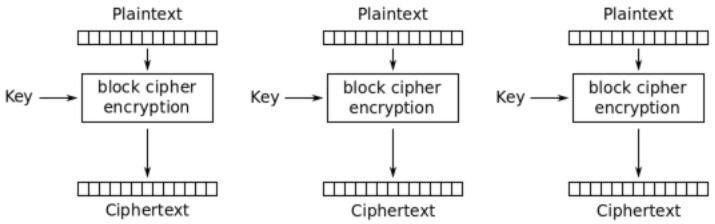
Modes of Operation
If message is longer than blocksize, block cipher can be used in a variety of ways to encrypt the plaintext.
Soon after DES was made a US Federal standard, another US standard appeared giving four recommended ways of using DES for data encryption.
These modes of operation have since been standardised internationally and can be used with any block cipher.

ECB Mode
ECB = Electronic Code Book
Simplest approach to using a block cipher.
Plaintext m is divided into t blocks of n bits $m_1, m_2, \ldots, m_t$ (the last block is padded if necessary).
Ciphertext blocks $c_1, \ldots, c_t$ are defined as follows:

$$c_i = e_k(m_i)$$

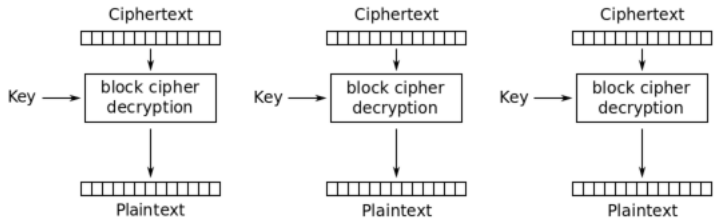
Note that if $m_i = m_j$ then we have $c_i = c_j$; thus patterns in plaintext reappear in ciphertext.

ECB Encipherment



Electronic Codebook (ECB) mode encryption

ECB Decipherment



Electronic Codebook (ECB) mode decryption

ECB Mode
Properties:

- Blocks are independent.
- Does not hide patterns or repetitions.
- Error propagation: expansion within one block.
- Reordering of blocks is possible without too much distortion.
- Stereotyped beginning/ends of messages are common.
- Susceptible to block replay.

Block Replay
Block replay:

- Extracting ciphertext corresponding to a known piece of plaintext.
- Amending other transactions to contain this known block of text.

Countermeasures against block replay:

- Extra checksums over a number of plaintext blocks.
- Chaining the cipher; this adds context to a block.

CBC Mode

CBC = Cipher Block Chaining

Plaintext m is divided into t blocks of n bits $m_1, m_2, \ldots, m_t$ (the last block is padded if necessary).

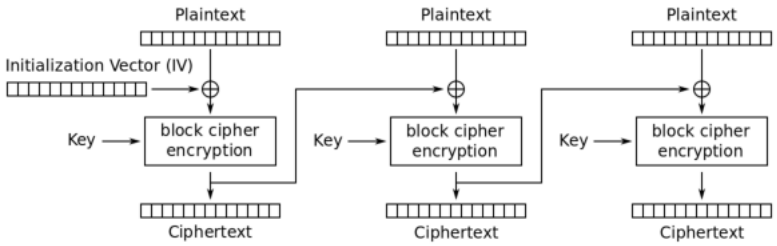
Encryption:

- $c_1 = e_k(m_1 \oplus IV)$.
- $c_i = e_k(m_i \oplus c_{i-1})$ for $i > 1$.

Decryption:

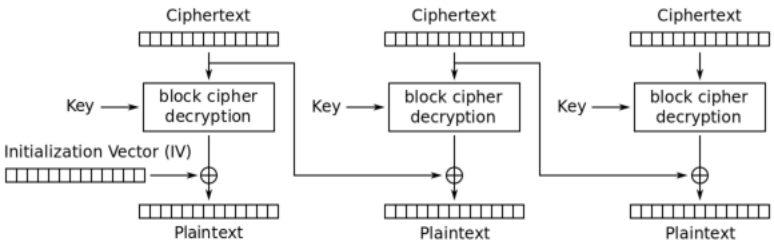
- $m_1 = d_k(c_1) \oplus IV$.
- $m_i = d_k(c_i) \oplus c_{i-1}$ for $i > 1$.

CBC Encipherment



Cipher Block Chaining (CBC) mode encryption

CBC Decipherment

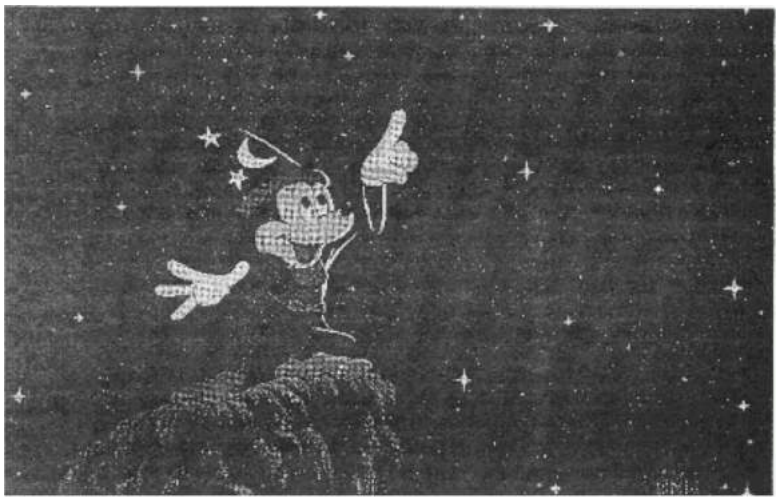


Cipher Block Chaining (CBC) mode decryption

CBC Mode
Properties:

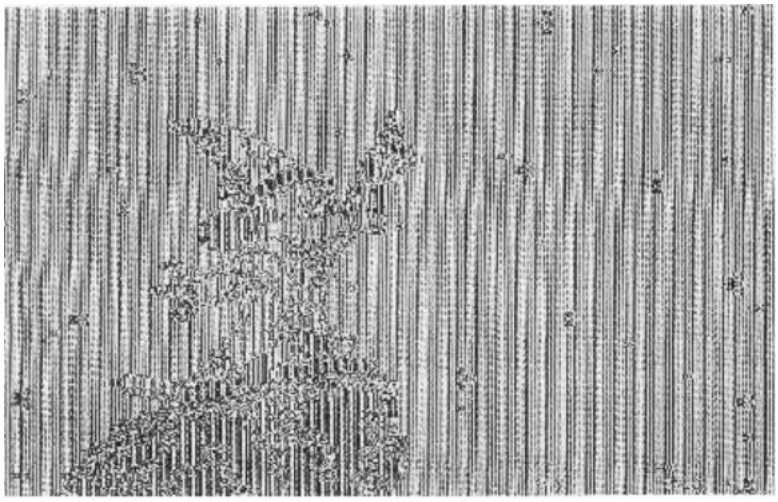
- Ciphertext depends on all previous plaintext blocks (internal memory).
- Different *IV* hides patterns and repetitions.
- Error propagation: expansion in one block, copied in next block.
- Decryption of one block requires only ciphertext of previous block, so CBC is self-synchronizing.
- Rearranging order of blocks affects decryption (not if previous ciphertext block is correct).
- Default mode to use.

Example



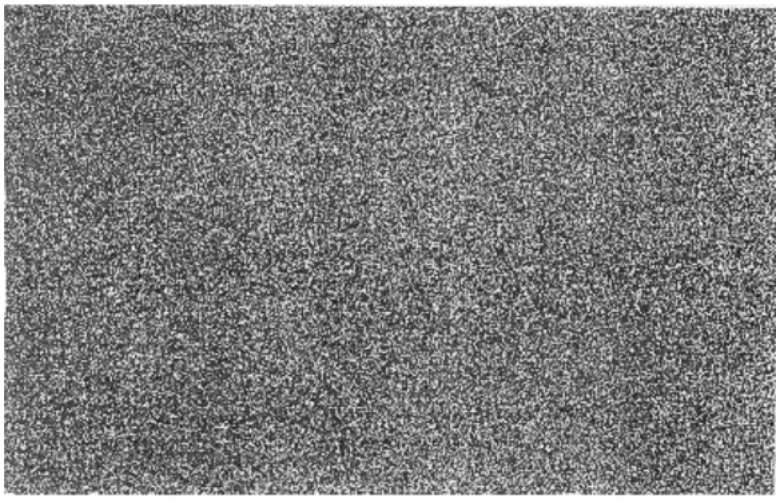
Plaintext : original picture

Example



Ciphertext: ECB Encryption

Example

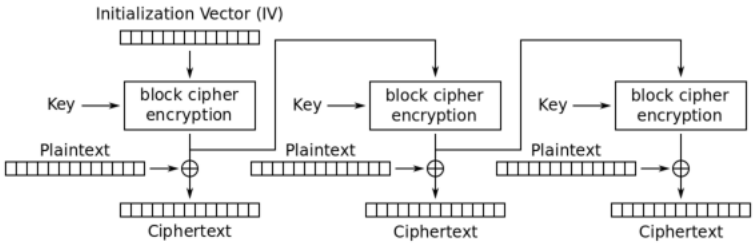


Ciphertext: CBC Encryption

OFB Mode
OFB = Output FeedBack
This mode enables a block cipher to be used as a stream cipher.

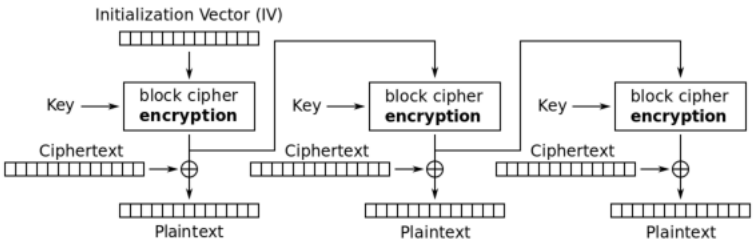
- The block cipher creates the keystream.
- Block length for block cipher is n.
- Can choose to use keystream blocks of $j \leq n$ bits only.
- Divide plaintext into a series of j-bit blocks $m_1, \ldots, m_t$.
- Encryption: $c_i = m_i \oplus e_i$, where e_i is selection of j bits of 'ciphertext' generated by block cipher, starting with IV in shift register.

OFB Encipherment



Output Feedback (OFB) mode encryption

OFB Decipherment



Output Feedback (OFB) mode decryption

OFB Mode

Properties:

- Synchronous stream cipher.
- No linking between subsequent blocks.
- Different IV necessary; otherwise insecure.

- Only uses encryption (no decryption algorithm necessary).
- If $j < n$: more effort per bit.
- Key stream independent of plaintext: can be precomputed.
- No error propagation: errors are only copied.

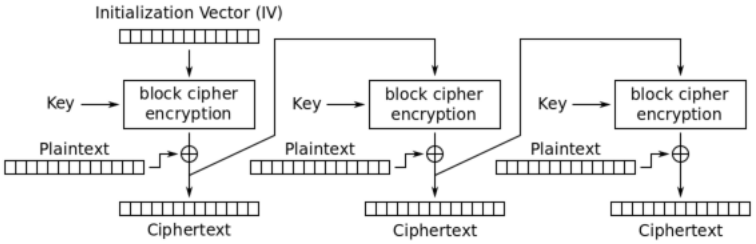
CFB Mode
CFB = Cipher FeedBack
In OFB Mode the keystream is generated by:

- Encrypting the IV.
- Encrypting the output from this encryption.

In CFB Mode the keystream is generated by:

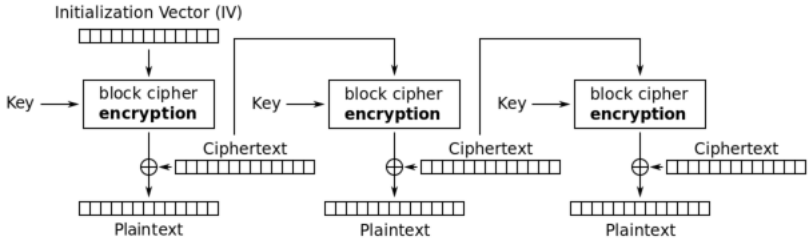
- Encrypting the IV.
- Encrypting n bits of ciphertext.

CFB Encipherment



Cipher Feedback (CFB) mode encryption

CFB Decipherment



Cipher Feedback (CFB) mode decryption

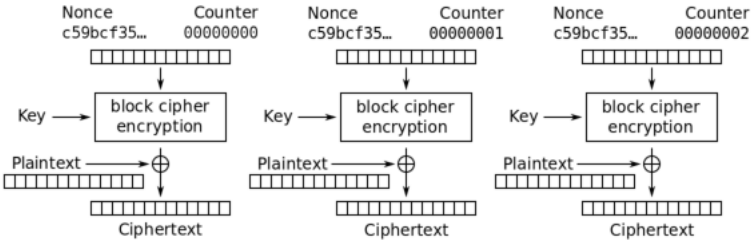
CFB Mode
Properties:

- Self-synchronizing stream cipher.
- Ciphertext depends on all previous plaintext blocks (internal memory).
- Different IV hides patterns and repetitions.
- Only uses encryption (no decryption algorithm necessary).
- If $j < n$: more effort per bit.
- Error propagation: propagates over $\lceil n/j \rceil + 1$ blocks.
- No synchronisation needed between sender and receiver; synchronisation if n bits have been received correctly.
- Often used with $j = 1, 8$ because of synchronisation.

CTR Mode
CTR = Counter

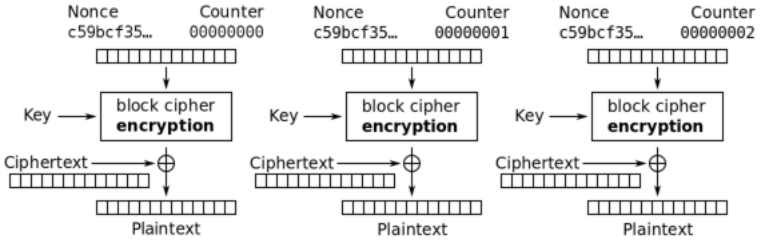
- Proposed more recently.
- Also turns the block cipher into a stream cipher.
- Enables blocks to be processed in parallel.
- Combines many of the advantages of ECB Mode, but with none of the disadvantages.
- Need to select a public IV and a different counter i for each message encrypted under the fixed key k.
- Encryption: $c_i = m_i \oplus e_k(IV + i)$
- Unlike ECB Mode two equal blocks will not encrypt to the same ciphertext value.
- Also unlike ECB Mode each ciphertext block corresponds to a precise position within the ciphertext, as its position information is needed to be able to decrypt it successfully.

CTR Encipherment



Counter (CTR) mode encryption

CTR Decipherment



Counter (CTR) mode decryption

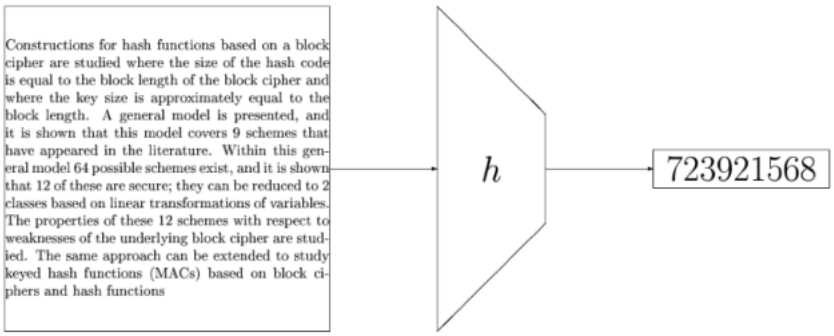
Mode Summary

	ECB	CBC	OFB	CFB	CTR
Patterns	-	+	+	+	+
Repetitions	-	IV	IV	IV	IV
Block length	n	n	j	j	n
Error prop.	1 block	1 block + 1 bit	1 bit	n bits + 1 bit	1 block
Synch.	block	block	exact	j bits	block
Parallel	enc/dec	dec	enc/dec	dec	enc/dec
Application	key enc.	default	no error prop.	synch.	parallel

4 Hash Functions

4.1 Hash Functions

Hash Functions
A hash function is an efficient function mapping binary strings of arbitrary length to binary strings of fixed length (e.g. 128 bits), called the hash-value or digest.



Hash Functions
A hash function is many-to-one; many of the inputs to a hash function map to the same digest.
However, for cryptography, a hash function must be one-way.

- Given only a digest, it should be computationally infeasible to find a piece of data that produces the digest (pre-image resistant).

A collision is a situation where we have two different messages M and M' such that $H(M) = H(M')$.

- A hash function should be collision free.
- A hash function is weakly collision-free or second pre-image resistant if given M it is computationally infeasible to find a different M' such that $H(M) = H(M')$.
- A hash function is strongly collision-free if it is computationally infeasible to find different messages M and M' such that $H(M) = H(M')$.

Hash Functions
In theory, given a digest D we can find data M that produces the digest by performing an exhaustive search.

- In fact, we can find as many pieces of such data that we want.
- With a well constructed hash function, there should not be a more efficient algorithm for finding M.

Why do we need hash functions?

- Given any data M we can determine its digest $H(M)$.
- Since it is (computationally) impossible to find another piece of data M' that produces the same digest, in certain circumstances we can use the digest $H(M)$ rather than M.
- We cannot recover M from $H(M)$, but in general, the digest is smaller than the original data and therefore, its use may be more efficient.
- We can think of the digest as a unique fingerprint of the data.

4.2 Collisions

The Birthday Paradox
What is the probability that two people have the same birthday?

People	Possibilities	Different Possibilities
2	365^2	365×364
3	365^3	$365 \times 364 \times 363$
	$\vdots$	
k	365^k	$365 \times 364 \times 363 \times \ldots \times (365-k+1)$

$$\text{P(no common birthday)} = \frac{365 \times 364 \times 363 \times \ldots \times (365-k+1)}{365^k}$$

The Birthday Paradox
With 22 people in a room, there is better than 50% chance that two people have a common birthday.
With 40 people in a room there is almost 90% chance that two people have a common birthday.
If there are k people, there are $\frac{k(k-1)}{2}$ pairs.

- The probability that one pair has a common birthday is approximately $\frac{k(k-1)}{2\times 365}$.
- If $k \geq \sqrt{365}$ then this probability is more than half.

In general, if there are n possibilities then on average $\sqrt{n}$ trials are required to find a collision.

Probability of Hash Collisions
Hash functions map an arbitrary length message to a fixed length digest.

- Many messages will map to the same digest.

Consider a 1000-bit message and 128-bit digest.

- There are 2^{1000} possible messages.

- There are 2^{128} possible digests.
- Therefore there are $2^{1000}/2^{128} = 2^{872}$ messages per digest value.

For a n-bit digest, we need to try an average of $2^{n/2}$ messages to find two with the same digest.

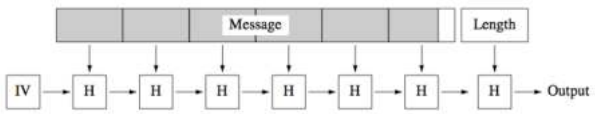
- For a 64-bit digest, this requires 2^{32} tries (feasible)
- For a 128-bit digest, this requires 2^{64} tries (not feasible)

Probability of Hash Collisions
Say B chooses 2^{32} messages M_i which A will accept that differ in 32 words, each of which has two choices:
A {will / promises to} {give / transfer to} B the amount of 100 {US / American} dollars {before / up to} April 2013. {Then / Later} B will {use / invest} this amount for ...
and 2^{32} messages M'_j which A will not accept that also differ in 32 words, each of which has two choices:
A {will / promises to} {give / transfer to} B the amount of {twenty / forty} {million / billion} {US / American} dollars {which / that} is given as a present and {should / will} not be returned ...

Probability of Hash Collisions
By the birthday paradox, there is a high probability that there is some pair of messages M_i and M'_j such that $H(M_i) = H(M'_j)$.
Both messages have the same signature.
B can claim in court that A signed on M'_j.
Alternatively, A can choose such two messages, sign one of them, and later claim in court that she signed the other message.

4.3 Merkle-Damgård Construction

Hash Functions
Most practical hash functions make use of the Merkle-Damgård construction which divides the message M into fixed-length blocks M_1, M_2, etc., pads the last block and appends the message length to the last block.
The resultant last block (after all paddings) is denoted by M_n.
Then, the hash function applies a collision-free function H on each of the blocks sequentially:



The function H takes as input the result of the application of H on the previous block (or a fixed initial value IV in the first block), and the block itself, and results in a hash value.
The hash value is an input to the application of H on the next block.

Hash Functions
The result of H on the last block is the hashed value of the message $h(M)$:

$$\begin{aligned} h_0 &= IV = \text{a fixed initial value} \\ h_1 &= H(h_0, M_1) \\ &\vdots \\ h_i &= H(h_{i-1}, M_i) \\ h_n &= H(h_{n-1}, M_n) \\ \\ h(M) &= h_n \end{aligned}$$

If H is collision-free, then h is also collision-free.

Hash Functions
Two approaches for the design of hash functions are:

1. To base the function H on a block cipher.
2. To design a special function H, not based on a block cipher.

The first approach was first proposed using DES; however the resulting hash is too small (64-bit).

- Susceptible to direct birthday attack.
- Also susceptible to "meet-in-the-middle" attack.

More modern block ciphers are suitable for implementing hash functions, but the second approach is more popular.

4.4 Commonly Used Hash Functions

Hash Functions
There are a number of widely used hash functions:

- `MD2`, `MD4`, `MD5` (Rivest).
  - Produce 128-bit digests.
  - Analysis has uncovered some weaknesses with these.
- `SHA-1` (Secure Hash Algorithm).
  - Produces 160-bit digests.
  - Analysis has also uncovered some weaknesses.

- `SHA-2` family (Secure Hash Algorithm).
  - `SHA-224`, `SHA-256`, `SHA-384` and `SHA-512`.
  - These yield digests of sizes 224, 256, 384 and 512 bits respectively.
- `SHA-3` (Secure Hash Algorithm).
  - KECCAK recently announced as winner of NIST competition.
  - Works very differently to SHA-1 and SHA-2.
- `RIPEMD`, `RIPEMD-160` (EU RIPE Project).
  - `RIPEMD` produces 128-bit digests.
  - `RIPEMD-160` produces 160-bit digests.

4.5 Applications of Hash Functions

Applications of Hash Functions
Applications of hash functions:

- Message authentication: used to check if a message has been modified.
- Digital signatures: encrypt digest with private key.
- Password storage: digest of password is compared with that in the storage; hackers can not get password from storage.
- Key generation: key can be generated from digest of pass-phrase; can be made computationally expensive to prevent brute-force attacks.
- Pseudorandom number generation: iterated hashing of a seed value.
- Intrusion detection and virus detection: keep and check hash of files on system

Information Security
Modern cryptography deals with more than just the encryption of data.
It also provides primitives to counteract active attacks on the communication channel.

- Confidentiality (only Alice and Bob can understand the communication)
- Integrity (Alice and Bob have assurance that the communication has not been tampered with)
- Authenticity (Alice and Bob have assurance about the origin of the communication)

Data Integrity
Encryption provides confidentiality.
Encryption does not necessarily provide integrity of data.
Counterexamples:

- Changing order in ECB mode.
- Encryption of a compressed file, i.e. without redundancy.
- Encryption of a random key.

Use cryptographic function to get a check-value and send it with data. Two types:

- Manipulation Detection Codes (MDC).
- Message Authentication Codes (MAC).

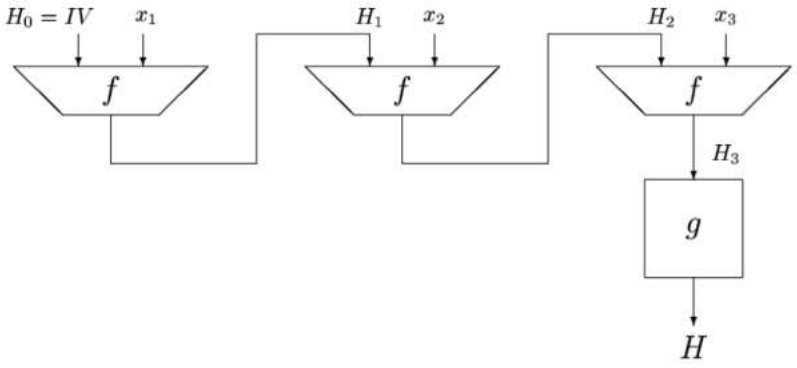
Manipulation Detection Code (MDC)
MDC: hash function without key.
The MDC is concatenated with the data and then the combination is encrypted/signed (to stop tampering with the MDC). $MDC = e_k(m||h(m))$, where:

- e is the encryption function.
- k is the secret key.
- h is the hash function.
- m is the message.
- $||$ denotes concatenation of data items.

Two types of MDC:

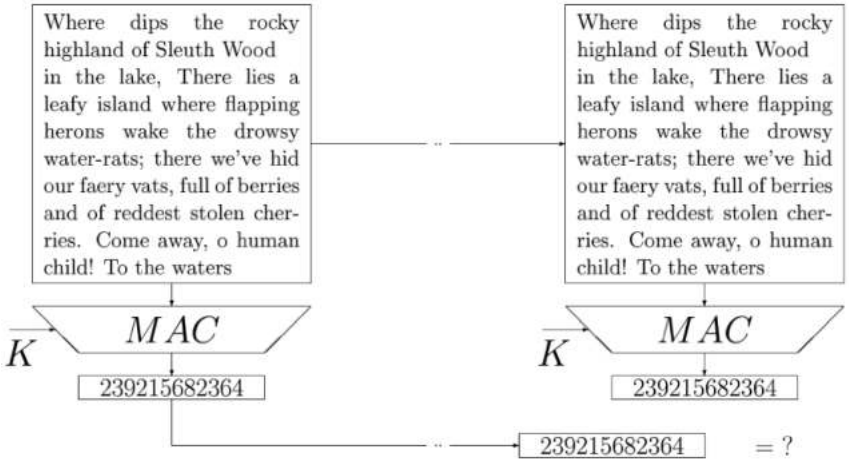
- MDCs based on block ciphers.
- Customised hash functions.

Manipulation Detection Code (MDC)
Most MDCs are constructed as iterated hash functions.



Compression/hash function f.
Output transformation g.
Unambiguous padding needed if length is not multiple of block length.

Message Authentication Code (MAC)
MAC: hash function with secret key.



Message Authentication Code (MAC)
$MAC = h_k(m)$, where:

- h is the hash function.
- k is the secret key.
- m is the message.

Transmit $m||MAC$, where $||$ denotes concatenation of data items.
Description of hash function is public.
Maps string of arbitrary length to string of fixed length (32-160 bits).
Computing $h_k(m)$ easy given m and k.
Computing $h_k(m)$ given m, but not k should be very difficult, even if a large number of pairs $\{m_i, h_k(m_i)\}$ are known.

MAC Mechanisms
There are various types of MAC scheme:

- MACs based on block ciphers in CBC mode.
- MACs based on MDCs.

- Customized MACs.

Best known and most widely used by far are the CBC-MACs.
CBC-MACs are the subject of various international standards:

- US Banking standards ANSIX9.9, ANSIX9.19.
- Specify CBC-MACs, date back to early 1980s.
- The ISO version is ISO 8731-1: 1987.

Above standards specify DES in CBC mode to produce a MAC.

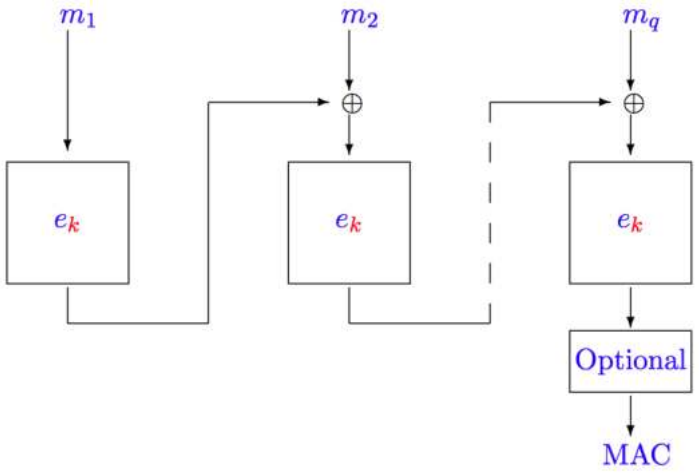
CBC-MAC
Given an n-bit block cipher, one constructs an m-bit MAC ($m \leq n$) as:

- Encipher the blocks using CBC mode (with padding if necessary).
- Last block is the MAC, after optional post-processing and truncation if $m < n$.

If the n-bit data blocks are $m_1, m_2, \ldots, m_q$ then the MAC is computed by:

- Put $I_1 = m_1$ and $O_1 = e_k(I_1)$.
- Perform the following for $i = 2, 3, \ldots, q$:
  - $I_i = m_i \oplus O_{i-1}$.
  - $O_i = e_k(I_i)$.
- O_q is then subject to an optional post-processing.
- The result is truncated to m bits to give the final MAC.

CBC-MAC



CBC-MAC: Post-Processing
Two specified optional post-processes:

- Choose a key k_1 and compute:

$$O_q = e_k(d_{k_1}(O_q))$$

- Choose a key k_1 and compute:

$$O_q = e_{k_1}(O_q)$$

The optional process can make it more difficult for a cryptanalyst to do an exhaustive key search for the key k.

MACs based on MDCs
Given a key k, how do you transform a MDC h into a MAC?
Secret prefix method: $MAC_k(m) = h(k||m)$

- Can compute $MAC_k(m||m') = h(k||m||m')$ without knowing k.

Secret suffix method: $MAC_k(m) = h(m||k)$

- Off-line attacks possible to find a collision in the hash function.

Envelope method with padding: $MAC_k(m) = h(k||p||m||k)$

- p is a string used to pad k to the length of one block.

None of these is very secure, better to use HMAC:

$$HMAC_k(m) = h(k||p_1||h(k||p_2||m))$$

with p_1, p_2 fixed strings used to pad k to full block.

MACs versus MDCs
Data integrity without confidentiality:

- MAC: compute $MAC_k(m)$ and send $m||MAC_k(m)$.
- MDC: send m and compute $MDC(m)$, which needs to be sent over an authenticated channel.

Data integrity with confidentiality:

- MAC: needs two different keys k_1 and k_2.
  - One for encryption and one for MAC.
  - Compute $c = e_{k_1}(m)$ and then append $MAC_{k_2}(c)$.
- MDC: only needs one key k for encryption.
  - Compute $MDC(m)$ and send $c = e_k(m||MDC(m))$.

Password Storage
Storing unencrypted passwords is obviously insecure and susceptible to attack.
Can store instead the digest of passwords.

- They need to be easy to remember.
- They should not be subject to a dictionary attack.

Can make use of a salt, which is a known random value that is combined with the password before applying the hash.

- The salt is stored alongside the digest in the password file: $\langle s, H(p||s) \rangle$.
- By using a salt, constructing a table of possible digests will be difficult, since there will be many possible for each password.
- An attacker will thus be limited to searching through a table of passwords and computing the digest for the salt that has been used.

Key Generation
We can generate a key at random.

- Most cryptographic APIs have facilities to generate keys at random.
- These facilities normally avoid weak keys.

We can also derive a key from a passphrase by applying a hash and using a salt.

- There are a number of standards for deriving a symmetric key from a passphrase e.g. PKCS#5.

This key generation may also require a number of iterations of the hash function.

- This makes the computation of the key less efficient.
- An attacker performing an exhaustive search will therefore require more computing resources or more time.

Pseudorandom Number Generation
Hash functions can be used to build a computationally-secure pseudo-random number generator as follows:

- First we seed the PRNG with some random data S.
- This is then hashed to produce the first internal state $S_0 = H(S)$.
- By repeatedly calling H we can generate a sequence of internal states $S_1, S_2, \ldots$, using $S_i = H(S_{i-1})$.
- From each state S_i we can extract bits to produce a random number N_i.
- This PRNG is secure if the sequence of values $S, S_0, S_1, \ldots$ is kept secret and the number of bits of S_i used to compute N_i is relatively small.

5 Number Theory

5.1 Introduction

Division
Let a and b be integers. We say that a divides b, or $a|b$ if:

$$\exists d \text{ s.t. } b = ad$$

If $b \neq 0$ then $|a| \leq |b|$.
Division Theorem: For any integer a and any positive integer n, there are unique integers q and r such that $0 \leq r < n$ and $a = qn + r$.
The value $r = a \bmod n$ is called the remainder or the residue of the division.
Theorem: If $d|a$ and $d|b$ then $d|(xa + yb)$ for any integers x, y.
Proof: $a = rd$ and $b = sd$ for some r, s. Therefore, $xa + yb = xrd + ysd = d(xr + ys)$, so $d|(xa + yb)$

Greatest Common Divisor
For integers a and b:
The greatest common divisor gcd(a,b) is defined as follows:

$$\gcd(a,b) = \max(d \text{ s.t. } d|a \text{ and } d|b) \ (a \neq 0 \text{ or } b \neq 0).$$

Note: This definition satisfies gcd(0,1) = 1.
The lowest common multiple lcm(a,b) is defined as follows:

$$\text{lcm}(a,b) = \min(m > 0 \text{ s.t. } a|m \text{ and } b|m) \text{ (for } a \neq 0 \text{ and } b \neq 0).$$

a and b are coprimes (or relatively prime) iff gcd(a,b) = 1.

Prime Numbers
An integer $p \geq 2$ is called prime if it is divisible only by 1 and itself.
Fundamental Theorem of Arithmetic: every positive number can be represented as a product of primes in a unique way, up to a permutation of the order of primes.
There are infinitely many primes

- Euclid gave simple proof by contradiction (c. 300BC).

The number of primes $\leq n : \pi(n) \approx n/\ln n$

- Even though the number of primes is infinite, the density of primes gets increasingly sparse as $n \to \infty$.

5.2 Modular Arithmetic

Modular Arithmetic
Modular arithmetic is fundamental to modern public key cryptosystems.
Given integers $a, b, n \in \mathbb{Z}$ we say that a is congruent to b modulo n:

$$a \equiv b \pmod{n} \text{ iff } n \text{ divides } b - a$$

Often we are lazy and just write $a \equiv b$ if it is clear we are working modulo n.
The modulo operator is like the C-operator %.
Example: $16 \equiv 1 \pmod{5}$ since 16-1 = 3×5

Modular Arithmetic
For convenience we define the set:

$$\mathbb{Z}_n = \{0, \ldots, n-1\}$$

which is the set of remainders modulo n.
It is clear that given n, every integer $a \in \mathbb{Z}$ is congruent modulo n to an element in the set $\mathbb{Z}_n$, since we can write:

$$a = q \times n + r$$

with $0 \leq r < n$ and $a \equiv r \pmod{n}$

Modular Arithmetic
The set $\mathbb{Z}_n$ has two operations defined on it.

- Addition
  - $11 + 13 \mod 16 \equiv 24 \pmod{16} \equiv 8 \pmod{16}$.
- Multiplication
  - $11 \times 13 \pmod{16} \equiv 143 \pmod{16} \equiv 15 \pmod{16}$.

Given integers $a, b \in \mathbb{Z}$ we have:

- $a + b \pmod{n} \equiv (a \pmod{n} + b \pmod{n}) \pmod{n}$
- $a - b \pmod{n} \equiv (a \pmod{n} - b \pmod{n}) \pmod{n}$
- $a \times b \pmod{n} \equiv (a \pmod{n} \times b \pmod{n}) \pmod{n}$

Modular Arithmetic
Properties of modular addition:

- Closure:
$$\forall a,b \in \mathbb{Z}_n : a+b \in \mathbb{Z}_n$$
- Associativity:
$$\forall a,b,c \in \mathbb{Z}_n : (a+b)+c \equiv a+(b+c)$$
- Commutativity:
$$\forall a,b \in \mathbb{Z}_n : a+b \equiv b+a$$
- Identity (0 is the identity):
$$\forall a \in \mathbb{Z}_n : a+0 \equiv 0+a \equiv a$$
- Inverse ($n-a$ is the inverse of a):
$$\forall a \in \mathbb{Z}_n : a+(n-a) \equiv (n-a)+a \equiv 0$$

Modular Arithmetic
Properties of modular multiplication:

- Closure:
$$\forall a,b \in \mathbb{Z}_n : a \times b \in \mathbb{Z}_n$$
- Associativity:
$$\forall a,b,c \in \mathbb{Z}_n : (a \times b) \times c \equiv a \times (b \times c)$$
- Commutativity:
$$\forall a,b \in \mathbb{Z}_n : a \times b \equiv b \times a$$
- Distributivity (distrubutes over addition):
$$\forall a,b,c \in \mathbb{Z}_n : (a+b) \times c \equiv a \times c + b \times c$$
- Identity (1 is the identity) :
$$\forall a \in \mathbb{Z}_n : a \times 1 \equiv 1 \times a \equiv a$$

Multiplicative Inverse
Division a/b in modular arithmetic is performed by multiplying a by the multiplicative inverse of b.
The multiplicative inverse of $b \in \mathbb{Z}_n$ is an element denoted $b^{-1} \in \mathbb{Z}_n$ with:
$$bb^{-1} \equiv b^{-1}b \equiv 1$$

Theorem: $b \in \mathbb{Z}_n$ has a unique inverse modulo n iff b and n are relatively prime i.e. gcd(b,n) = 1.
Theorem: If p is a prime then every non-zero element in $\mathbb{Z}_p$ has an inverse.

Multiplicative Inverse
Consider $\mathbb{Z}_{10}$:

- 3 has a multiplicative inverse, since gcd(3,10)=1.
  - $3 \times 7 \equiv 21 \equiv 1 \pmod{10}$.
- 5 has no multiplicative inverse, since gcd(5,10)=5.
  - We have the following table:

$$\begin{array}{ll} 0 \times 5 \equiv 0 \pmod{10} & 5 \times 5 \equiv 5 \pmod{10} \\ 1 \times 5 \equiv 5 \pmod{10} & 6 \times 5 \equiv 0 \pmod{10} \\ 2 \times 5 \equiv 0 \pmod{10} & 7 \times 5 \equiv 5 \pmod{10} \\ 3 \times 5 \equiv 5 \pmod{10} & 8 \times 5 \equiv 0 \pmod{10} \\ 5 \times 5 \equiv 0 \pmod{10} & 9 \times 5 \equiv 5 \pmod{10} \end{array}$$

Greatest Common Divisor (GCD)
We need a method to determine when $a \in \mathbb{Z}_n$ has a multiplicative inverse and compute it when it does.
We know this happens iff gcd(a,n) = 1.
Therefore we need to compute the GCD of two integers $a, b \in \mathbb{Z}$.

- This is easy if we know the prime factorization of a and b, since:

$$a = \prod p_i^{\alpha_i} \text{ and } b = \prod p_i^{\beta_i} \Rightarrow d = \gcd(a,b) = \prod p_i^{min(\alpha_i,\beta_i)}$$

- However, factoring is a very expensive operation, so we cannot use the above formula.
- A much faster algorithm to compute GCDs is Euclid's algorithm.

GCD - Euclidean Algorithm
To compute the GCD of a and b we compute:

$$\begin{array}{lcl} a & = & q_0 b + r_0 \\ b & = & q_1 r_0 + r_1 \\ & \vdots & \\ r_{k-2} & = & q_{k-1} r_{k-1} + r_k \\ r_{k-1} & = & q_k r_k \end{array}$$

If d divides a and b then d divides r_0, r_1, r_2 and so on.
Therefore: gcd(a, b) = gcd(b, r_0) = gcd(r_0, r_1) = $\cdots$ = gcd(r_{k-1}, r_k) = r_k

GCD - Euclidean Algorithm
As an example of this algorithm we want to show that:

$$3 = \gcd(21,12)$$

Using the Euclidean algorithm we compute gcd(21,12) as:

$$\begin{aligned} \gcd(21,12) &= \gcd(21 \pmod{12},12) \\ &= \gcd(9,12) \\ &= \gcd(12 \pmod 9,9) \\ &= \gcd(3,9) \\ &= \gcd(9 \pmod 3,3) \\ &= \gcd(0,3) = 3 \end{aligned}$$

XGCD - Extended Euclidean Algorithm
Using the Euclidean algorithm, we can determine when a has an inverse modulo n i.e. iff $\gcd(a,n) = 1$.

- But we do not know yet how to compute the inverse.

Solution: use an extended version of the Euclidean algorithm.
Recall that during the Euclidean algorithm we had:

$$r_{i-2} = q_{i-1}r_{i-1} + r_i$$

and finally $r_k = \gcd(a,b)$.
Now we unwind the above and write each r_i in terms of a and b.

XGCD - Extended Euclidean Algorithm
Unwinding the various steps in the Euclidean algorithm gives:

$$\begin{aligned} r_0 &= a - q_0 b \\ r_1 &= b - q_1 r_0 = b - q_1(a - q_0 b) = -q_1 a + (1 + q_0 q_1)b \\ &\vdots \\ r_k &= xa + yb \end{aligned}$$

The XGCD takes as input a and b and outputs x, y, r_k such that:

$$r_k = \gcd(a,b) = xa + yb$$

XGCD - Multiplicative Inverse
Given $a, n \in \mathbb{Z}$ we can compute d, x, y using XGCD such that:

$$d = \gcd(a,n) = xa + yn$$

Considering the above equation modulo n we get:

$$d \equiv xa + yn \pmod{n} \equiv xa \pmod{n}$$

Thus if $d = 1$ then a has a multiplicative inverse given by:

$$a^{-1} \equiv x \pmod{n}$$

Remark: the more general equation $ax \equiv b \pmod{n}$ has precisely $d = \gcd(a,n)$ solutions iff d divides b.

Modular Exponentiation
Given a prime p and $a \in \mathbb{Z}_p^*$ we want to calculate $a^x \pmod{p}$.
It does not make sense to compute $y = a^x$ and then $y \pmod{p}$.
Consider $123^5 \pmod{511} = 28153056843 \pmod{511} = 359$
There is a large intermediate result so this method takes a very long time and a great deal of space for large a, x and p.
$123^5 \pmod{511}$ could also be calculated as follows:

$$\begin{array}{lcl} a & = & 123 \\ a^2 & = & a \times a \pmod{511} = 310 \\ a^3 & = & a \times a^2 \pmod{511} = 316 \\ a^4 & = & a \times a^3 \pmod{511} = 32 \\ a^5 & = & a \times a^4 \pmod{511} = 359 \end{array}$$

This requires four modular multiplications; it is still far too slow.

Modular Exponentiation
It is much better to compute this example using the steps below:

$$\begin{array}{lcl} a & = & 123 \\ a^2 & = & a \times a \pmod{511} = 310 \\ a^4 & = & a^2 \times a^2 = 310 \times 310 \pmod{511} = 32 \\ a^5 & = & a \times a^4 = 123 \times 32 \pmod{511} = 359 \end{array}$$

This requires only 3 multiplications.
This shows that if we consider the binary representation of the exponent $x = x_{n-1}x_{n-2}\ldots x_1x_0$, then the value represented by each bit of the exponent x_i can be obtained by squaring the value represented by the previous bit x_{i-1}.
Multiplication is required for every bit which is set after the first one.
Thus for an exponent with n bits of which k bits are set, $n-1$ squarings and $k-1$ multiplications.

Modular Exponentiation
This suggests an algorithm which works through the exponent one bit at a time squaring and multiplying.
This is commonly known as the square and multiply algorithm.
Right to left variant for calculating $y = a^x \pmod{p}$:

```
y = 1
for i = 0 to n-1 do
   if x_i = 1 then y = (y*a) mod p
   a = (a*a) mod p
end
```

Left to right variant for calculating $y = a^x \pmod{p}$:

```
y = 1
for i = n-1 downto 0 do
   y = (y*y) mod p
   if x_i = 1 then y = (y*a) mod p
end
```

Chinese Remainder Theorem (CRT)
Consider $n = 15 = 3 \times 5$.
We can represent every element a of $\mathbb{Z}_n$ by its coordinates $(a \pmod 3, a \pmod 5)$.
This leads to the following table:

	0	1	2	3	4
0	0	6	12	3	9
1	10	1	7	13	4
2	5	11	2	8	14

Note that all elements in $\mathbb{Z}_n$ have different coordinates, i.e. given (a_1, a_2) with $0 \leq a_1 < 3$ and $0 \leq a_2 < 5$ we can reconstruct a.

Chinese Remainder Theorem (CRT)
Consider $n = 24 = 4 \times 6$.
We can represent every element a of $\mathbb{Z}_n$ by its coordinates $(a \pmod 4, a \pmod 6)$.
This leads to the following table:

	0	1	2	3	4	5
0	0/12		8/20		4/16	
1		1/13		9/21		5/17
2	6/18		2/14		10/22	
3		7/19		3/15		11/23

Note that a and $a + 12 \pmod{24}$ map to the same coordinates.
Therefore, given (a_1, a_2) with $0 \leq a_1 < 4$ and $0 \leq a_2 < 6$ we cannot uniquely reconstruct a.

Chinese Remainder Theorem (CRT)
The previous examples indicate that if $n = n_1 \times n_2$ with $\gcd(n_1, n_2) = 1$, we can replace computing modulo n by computing modulo n_1 and modulo n_2:

$$\mathbb{Z}_n \cong \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2} \text{ iff } \gcd(n_1, n_2) = 1$$

If $n = n_1 \times n_2$ then it is very easy to compute the coordinates of $a \in \mathbb{Z}_n$, since these are simply $(a \pmod{n_1}), a \pmod{n_2})$.
However, given the coordinates (a_1, a_2) with $0 \leq a_1 < n_1$ and $0 \leq a_2 < n_2$ how do we compute the corresponding a?

Chinese Remainder Theorem (CRT)
We can reformulate our reconstruction problem as:
Given: $n = n_1 \times n_2$ with $\gcd(n_1, n_2) = 1$
Compute: $x \in \mathbb{Z}_n$ with $x \equiv a_1 \pmod{n_1}$ and $x \equiv a_2 \pmod{n_2}$
Example: If $x \equiv 4 \pmod{7}$ and $x \equiv 3 \pmod{5}$ then we have:

$$x \equiv 18 \pmod{35}$$

How did we work this out?

CRT - Example
We want to find $x \in \mathbb{Z}_n$ with $n = 35$ such that:

$$x \equiv 4 \pmod{7} \text{ and } x \equiv 3 \pmod{5}$$

Therefore, for some $k \in \mathbb{Z}$ we have:

$$x = 4 + 7k \text{ and } x \equiv 3 \pmod{5}$$

Substituting the equality in the second equation gives:

$$4 + 7k \equiv 3 \pmod{5}$$

Therefore, k is given by the solution of:

$$2k \equiv 7k \equiv 3 - 4 \equiv 4 \pmod{5}$$

Hence we can compute k as $k \equiv 4/2 \pmod{5} \equiv 2 \pmod{5}$, so:

$$x \equiv 4 + 7k \equiv 4 + 7 \times 2 \equiv 18 \pmod{35}$$

CRT - General Case
Let $n_1, \ldots, n_k$ be pairwise relatively prime and let $a_1, \ldots, a_k$ be integers.
We want to find x modulo $n = n_1 n_2 \cdots n_k$ such that:

$$x \equiv a_i \pmod{n_i} \text{ for all } i$$

The CRT guarantees a unique solution given by:

$$x = \sum_{i=1}^{k} a_i \times N_i \times y_i \pmod{n}$$

$$N_i = n/n_i \text{ and } y_i = N_i^{-1} \pmod{n_i}$$

Note that $N_i \equiv 0 \pmod{n_j}$ for $j \neq i$ and that $N_i \times y_i \equiv 1 \pmod{n_i}$

CRT - General Case Example
We want to find the unique x modulo $n = 1001 = 7 \times 11 \times 13$ such that:

$$x \equiv 5 \pmod 7 \text{ and } x \equiv 3 \pmod{11} \text{ and } x \equiv 10 \pmod{13}$$

We compute:

$$N_1 = 143, y_1 = 5 \text{ and } N_2 = 91, y_2 = 4 \text{ and } N_3 = 77, y_3 = 12.$$

Then we reconstruct x as:

$$\begin{aligned} x &\equiv \sum_{i=1}^{k} a_i \times N_i \times y_i \pmod n \\ &\equiv 5 \times 143 \times 5 + 3 \times 91 \times 4 + 10 \times 77 \times 12 \pmod{1001} \\ &\equiv 894 \pmod{1001} \end{aligned}$$

CRT - Modular Exponentiation
Let $n = 55 = 5 \times 11$ and suppose we want to compute $27^{37} \pmod n$.
This can be done in a number of ways:

- Really stupid: using 36 multiplications modulo 55:

$$(((27 \times 27) \pmod n) \times 27 \pmod n) \cdots 27 \pmod n$$

- Less stupid: using 5 squarings and 2 multiplications modulo 55:

$$((27^{2^5} \pmod n) \times 27^{2^2} \pmod n) \times 27 \pmod n$$

- Rather intelligent: using 5 squarings and 2 multiplications modulo 5 and modulo 11 and CRT to combine both results.

Quadratic Residues
An integer q is called a quadratic residue modulo n if there exists an integer x such that:

$$x^2 \equiv q \pmod n$$

Integer x is called the square root of $q \pmod n$.
If no such integer x exists, q is called a quadratic nonresidue modulo n.

Example (n = 11):

x	0	1	2	3	4	5	6	7	8	9	10
$x^2 \pmod{11}$	0	1	4	9	5	3	3	5	9	4	1

There are five quadratic residues modulo 11: 1, 3, 4, 5, and 9.
There are five quadratic non-residues modulo 11: 2, 6, 7, 8, 10.
The trivial case $x^2 = 0$ is generally excluded from the list of quadratic residues.

Quadratic Residues
If p is a prime exactly half of the numbers in $\mathbb{Z}_p^*$ are quadratic residues.
Euler's Criterion: Given odd prime p and $q \in \mathbb{Z}_p^*$:

- q is a quadratic residue iff $q^{(p-1)/2} \equiv 1 \pmod{p}$.
- q is quadratic nonresidue, iff $q^{(p-1)/2} \equiv -1 \pmod{p}$.

A quadratic residue $q \in \mathbb{Z}_p^*$ cannot be a primitive root, since $q^{(p-1)/2} \equiv 1 \pmod{p}$ and the order of a primitive root is $p-1$.

Quadratic Residues Modulo $n = pq$
Let $n = pq$ where p and q are large primes.
If $a \in \mathbb{Z}_n^*$ is a quadratic residue modulo n, then a has exactly four square roots modulo n in $\mathbb{Z}_n^*$.
Therefore exactly a quarter of the numbers in $\mathbb{Z}_n^*$ are quadratic residues modulo n.

Legendre's Symbol
If p is a prime and a is an integer.
Legendre's symbol $\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{if } a|p \\ +1, & \text{if } a \text{ is a quadratic residue modulo } p \\ -1, & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases}$
By Euler's criterion: $\left(\frac{a}{p}\right) = a^{(p-1)/2} \pmod{p}$.

Legendre's Symbol
Properties of Legendre's symbol:

1. $a \equiv b \pmod{p} \Rightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$
2. $\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$
3. $\left(\frac{1}{p}\right) = 1$
4. $\left(\frac{-1}{p}\right) = \begin{cases} 1, & \text{if } p \equiv 1 \pmod{4} \\ -1, & \text{if } p \equiv 3 \pmod{4} \end{cases}$
5. $\left(\frac{2}{p}\right) = (-1)^{(p^2-1)/8}$
6. If p and q are odd primes: $\left(\frac{p}{q}\right) = (-1)^{((p-1)/2)((q-1)/2)}\left(\frac{q}{p}\right)$

Jacobi's Symbol
Jacobi's symbol is a generalization of Legendre's symbol to composite numbers.
If n is odd with prime factorization $n = p_1 \times p_2 \times \ldots \times p_k$ and a is relatively prime to n:
Jacobi's symbol $\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right) \times \left(\frac{a}{p_2}\right) \times \ldots \times \left(\frac{a}{p_k}\right)$
$\left(\frac{a}{n}\right) = -1 \Rightarrow a$ is a quadratic non-residue
$\left(\frac{a}{n}\right) = 1 \not\Rightarrow a$ is a quadratic residue

Jacobi's Symbol
Properties of Jacobi's symbol:

1. $a \equiv b \pmod{n} \Rightarrow \left(\frac{a}{n}\right) = \left(\frac{b}{n}\right)$
2. $\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right)$
3. $\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right)$
4. $\left(\frac{1}{n}\right) = 1$
5. $\left(\frac{-1}{n}\right) = (-1)^{(n-1)/2}$
6. $\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}$
7. If m and n are odd co-primes: $\left(\frac{m}{n}\right) = (-1)^{((m-1)/2)((n-1)/2)}\left(\frac{n}{m}\right)$

Square Roots Modulo Prime $p \equiv 3 \pmod{4}$
If the Legendre symbol is -1, then there is no solution.
If p is a prime and a is a quadratic residue modulo p then:

$$a^{(p-1)/2} \equiv 1 \pmod{p} \text{ (by Euler's criterion).}$$

Multiplying both sides by a:

$$a^{(p+1)/2} \equiv a \pmod{p}$$

Taking the square roots of both sides:

$$\pm a^{(p+1)/4} \equiv \sqrt{a} \pmod{p}$$

If $p \equiv 3 \pmod{4}$, then $(p+1)/4$ is an integer, and this can be used to calculate the square root.

Square Roots Modulo Prime $p \equiv 5 \pmod 8$
If p is a prime and a is a quadratic residue modulo p then:

$$a^{(p-1)/2} \equiv 1 \pmod p \text{ (by Euler's criterion).}$$

Taking the square roots of both sides:

$$a^{(p-1)/4} \equiv \pm 1 \pmod p$$

If $a^{(p-1)/4} \equiv 1 \pmod p$ then:

$$\sqrt{a} = \pm a^{(p+3)/8} \pmod p$$

If $a^{(p-1)/4} \equiv -1 \pmod p$ then:

$$\sqrt{a} = \pm 2a(4a)^{(p-5)/8} \pmod p$$

If $p \equiv 5 \pmod 8$ then $(p+3)/8$ and $(p-5)/8$ are integers.

Square Roots Modulo Prime $p \equiv 1 \pmod 8$
If p is a prime s.t. $p \equiv 1 \pmod 8$ and a is a quadratic residue modulo p the probabilistic Tonelli-Shanks algorithm can be used to calculate $\sqrt{a}$:

Choose a random n until one is found such that $(n/p) = -1$
Let e, q be integers such that q is odd and $p-1 = 2^e q$
$y = n^q \pmod p$
$r = e$
$x = a^{(q-1)/2} \pmod p$
$b = ax^2 \pmod p$
$x = ax \pmod p$
while $b \neq 1 \pmod p$ **do**
  Find the smallest m such that $b^{2^m} = 1 \pmod p$
  $t = y^{2^{r-m-1}} \pmod p$
  $y = t^2 \pmod p$
  $r = m$
  $x = xt \pmod p$
  $b = by \pmod p$
end
return x

Square Roots Modulo $n = pq$
If the Jacobi symbol is -1, then there is no solution.
If a is a quadratic residue and $\sqrt{a} \pmod p = \pm x$ and $\sqrt{a} \pmod q = \pm y$, then we can use the Chinese Remainder Theorem to calculate $\sqrt{a}$.
Example: Compute the square root of 3 modulo 11×13
$\sqrt{3} \pmod{11} = \pm 5$
$\sqrt{3} \pmod{13} = \pm 4$
Using the Chinese Remainder Theorem, we can calculate the four square roots as 82, 126, 17 and 61.

5.3 Group Theory

Groups
A group $(S, \oplus)$ consists of a set S and an operation $\oplus$, satisfying:

- Closure: $\forall a, b \in S : a \oplus b \in S$
- Associativity: $\forall a, b, c \in S : a \oplus (b \oplus c) = (a \oplus b) \oplus c$
- Identity element e: $\exists e \in S : \forall a \in S : a \oplus e = e \oplus a = a$
- Every element has an inverse element:

$$\forall a \in S : \exists a^{-1} \in S : a \oplus a^{-1} = a^{-1} \oplus a = e$$

- The group S is called commutative or Abelian if:

$$\forall a, b \in S : a \oplus b = b \oplus a$$

The order of a group S, denoted by $|S|$, is the number of elements in S. If a group S satisfies $|S| < \infty$ then it is called a finite group.

Groups
$(\mathbb{Z}, +)$, $(\mathbb{R}, +)$, $(\mathbb{Q}, +)$ and $(\mathbb{C}, +)$ are groups.

- the identity is 0, the inverse of x is $-x$

$(\mathbb{R} \setminus \{0\}, \times)$, $(\mathbb{Q} \setminus \{0\}, \times)$ and $(\mathbb{C} \setminus \{0\}, \times)$ are groups.

- the identity is 1, the inverse of x is $1/x$

These are all examples of infinite Abelian groups.
$(\mathbb{Z}_n, +)$ is a finite Abelian group.
Questions:

- Why is $(\mathbb{Z} \setminus \{0\}, \times)$ not a group?
- Why is $(\mathbb{R}, \times)$ not a group?

Rings
A ring $(S, \oplus, \otimes)$ consists of a set S together with two binary operators $\oplus$ and $\otimes$, satisfying:

- Closure of $\oplus$: $\forall a, b \in S : a \oplus b \in S$
- Associativity of $\oplus$: $\forall a, b, c \in S : a \oplus (b \oplus c) = (a \oplus b) \oplus c$
- Commutativity of $\oplus$: $\forall a, b \in S : a \oplus b = b \oplus a$
- Identity for $\oplus$: $\exists 0 \in S : \forall a \in S : a \oplus 0 = 0 \oplus a = a$

- Inverse for $\oplus$: $\forall a \in S : \exists -a \in S : a \oplus -a = -a \oplus a = 0$
- Closure of $\otimes$: $\forall a, b \in S : a \otimes b \in S$
- Associativity of $\otimes$: $\forall a, b, c \in S : a \oplus (b \otimes c) = (a \otimes b) \otimes c$
- Identity for $\otimes$: $\exists 1 \in S : \forall a \in S : a \otimes 1 = 1 \otimes a = a$
- Distributivity of $\otimes$ over $\oplus$: $\forall a, b, c \in S : a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$

Rings
The following are all examples of infinite rings:

- $(\mathbb{Z}, +, \times)$
- $(\mathbb{R}, +, \times)$
- $(\mathbb{Q}, +, \times)$
- $(\mathbb{C}, +, \times)$

The following is an example of a finite ring:

- $(\mathbb{Z}_n, +, \times)$

Fields
A field $(S, \oplus, \otimes)$ consists of a set S together with two binary operators $\oplus$ and $\otimes$, satisfying all the properties of a ring plus the following:

- Commutativity of $\otimes$: $\forall a, b \in S : a \otimes b = b \otimes a$
- Inverse for $\otimes$: $\forall a \neq 0 \in S : \exists a^{-1} \in S : a \otimes a^{-1} = a^{-1} \otimes a = 1$

So $(S, \oplus)$ is an abelian group with identity 0 and $(S \setminus \{0\}, \otimes)$ is an abelian group with identity 1.
$(\mathbb{R}, +, \times)$, $(\mathbb{Q}, +, \times)$ and $(\mathbb{C}, +, \times)$ are all infinite fields.
$(\mathbb{Z}_n^*, +, \times)$ is a finite field.

Finite Fields
A finite field is a field that contains a finite number of elements.
There is exactly one finite field of size (order) p^n where p is a prime (called the characteristic of the field) and n is a positive integer.
If p is a prime $\mathbb{Z}_p$ is the finite field GF(p) (note here that $n = 1$ and so is omitted).
Finite fields are of central importance in coding theory and cryptography.
GF(2^8) is of particular importance as an element can be represented in a single byte.

Euler Groups
We define the set of invertible elements of $\mathbb{Z}_n$ as:

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : \gcd(a,n) = 1\}$$

The set $\mathbb{Z}_n^*$ is always a group with respect to multiplication and is called an Euler group.
When n is a prime p we have:

$$\mathbb{Z}_p^* = \{1, \ldots, p-1\}$$

Examples:

$$\begin{array}{ll} \mathbb{Z}_1 = \{0\} & \mathbb{Z}_1^* = \{0\} \\ \mathbb{Z}_2 = \{0,1\} & \mathbb{Z}_2^* = \{1\} \\ \mathbb{Z}_3 = \{0,1,2\} & \mathbb{Z}_3^* = \{1,2\} \\ \mathbb{Z}_4 = \{0,1,2,3\} & \mathbb{Z}_4^* = \{1,3\} \\ \mathbb{Z}_5 = \{0,1,2,3,4\} & \mathbb{Z}_5^* = \{1,2,3,4\} \end{array}$$

Euler Totient Function $\phi(n)$
Euler's totient function $\phi(n)$ represents the number of elements in $\mathbb{Z}_n^*$:

$$\phi(n) = |\mathbb{Z}_n^*| = |\{a \in \mathbb{Z}_n : \gcd(a,n) = 1\}|$$

$\phi(n)$ is therefore the number of integers in $\mathbb{Z}_n$ which are relatively prime to n.
We know that an element $a \in \mathbb{Z}_n$ has a multiplicative inverse modulo n iff $\gcd(a,n) = 1$.
Therefore, there are precisely $\phi(n)$ invertible elements in $\mathbb{Z}_n$.

Euler Totient Function $\phi(n)$
Given the prime factorization of n:

$$n = \prod_{i=1}^{k} p_i^{e_i}$$

we can compute $\phi(n)$ using the following formula:

$$\phi(n) = \prod_{i=1}^{k} p_i^{e_i - 1}(p_i - 1)$$

The most important cases for cryptography are:

- If p is prime then:
$$\phi(p) = p - 1$$

- If p and q are both prime and $p \neq q$ then:
$$\phi(pq) = (p-1)(q-1)$$

Lagrange's Theorem
The order of an element a of a group $(S,\otimes)$ is the smallest positive integer k such that $a^k = 1$.
Lagrange's Theorem:

$$\text{If } S \text{ is a group of size } |S| = n \text{ then } \forall a \in S : a^n = 1$$

Corollary: the order k of an element $a \in S$ divides $n = |S|$, so if $a \in \mathbb{Z}_n^*$ then k divides $\phi(n)$.
Thus if $a \in \mathbb{Z}_n^*$ then $a^{\phi(n)} \equiv 1 \pmod{n}$, since $|\mathbb{Z}_n^*| = \phi(n)$.
This is known as Euler's Theorem.

Euler's Theorem
Euler's theroem allows us to simplify the calculation of modular exponentiation since the following holds:
$a^k \pmod{n} \equiv a^{k \pmod{\phi(n)}} \pmod{n}$
For example, to calculate $101^{108} \pmod{109}$, we can simplify this as follows:

$$\begin{aligned} & 101^{108} \pmod{109} \\ = \; & 101^{108 \pmod{\phi(109)}} \pmod{109} \\ = \; & 101^{108 \pmod{108}} \pmod{109} \\ = \; & 101^{0} \pmod{109} \\ = \; & 1 \end{aligned}$$

Fermat's Little Theorem
Not to be confused with Fermat's Last Theorem . . .
If p is a prime, Lagrange's Theorem tells us that $a^{p-1} \equiv 1 \pmod{p}$.
If we multiply both sides of this equation by a, we obtain Fermat's Little Theorem:

$$\text{if } p \text{ is a prime then } a^p \equiv a \pmod{p}$$

Fermat's Little Theorem can be used to test whether a number n is probably a prime: this will be the case if $a^{n-1} \equiv 1 \pmod{n}$.

Generators
For $a \in \mathbb{Z}_n^*$ the set $\{a^0, a^1, a^2, a^3, \ldots\}$ is called the group generated by a, denoted $\langle a \rangle$.
The order of $a \in \mathbb{Z}_n^*$ is the size of $\langle a \rangle$, denoted $|\langle a \rangle|$.
Examples for $\mathbb{Z}_7^*$:
$\langle 3 \rangle = \{1,3,2,6,4,5\}$, so the order of 3 is 6
$\langle 2 \rangle = \{1,2,4\}$, so the order of 2 is 3
$\langle 1 \rangle = \{1\}$, so the order of 1 is 1

Primitive Roots
$a \in \mathbb{Z}_n^*$ is called a primitive root of $\mathbb{Z}_n^*$ if the order of a is $\phi(n)$.
Not all groups possess primitive roots e.g. $\mathbb{Z}_n^*$ where $n = pq$ and p, q are odd primes.
If $\mathbb{Z}_n^*$ possesses a primitive root a, then $\mathbb{Z}_n^*$ is called cyclic.
If a is a primitive root of $\mathbb{Z}_n^*$ and $b \in \mathbb{Z}_n^*$ then $\exists x$ s.t. $a^x \equiv b \pmod{n}$. This x is called the discrete logarithm or index of b modulo n to the base a.
Examples for $\mathbb{Z}_7^*$:
3 is a primitive root: $\{3^0, 3^1, 3^2, 3^3, 3^4, 3^5\} = \{1, 3, 2, 6, 4, 5\} = \mathbb{Z}_7^*$
2 is not a primitive root: $\{2^0, 2^1, 2^2, 2^3, 2^4, 2^5\} = \{1, 2, 4\} \neq \mathbb{Z}_7^*$

Primitive Roots
A primitive root exists in $\mathbb{Z}_n^*$ iff n has a value 2, 4, p^k or $2p^k$ for some odd prime p and integer k.
To determine whether a is a primitive root of $\mathbb{Z}_n^*$, we need to show for all prime factors $p_1, \ldots, p_k$ of $\phi(n)$ that $a^{\phi(n)/p_i} \neq 1 \pmod{n}$ This can be determined using modular exponentiation.
For a prime p the number of primitive roots mod p is $\phi(p-1)$

5.4 Primality Testing

Prime Numbers
The generation of prime numbers is needed for many public key algorithms:

- RSA: Need to find p and q to compute $N = pq$
- ElGamal: Need to find prime modulus p
- Rabin: Need to find p and q to compute $N = pq$

We shall see that testing a number for primality can be done very fast

- Using an algorithm which has a probability of error
- Repeating the algorithm lowers the error probability to any value we require.

Prime Numbers
Before discussing the algorithms we need to look at some basic heuristics concerning prime numbers.
A famous result in mathematics, conjectured by Gauss after extensive calculation in the early 1800's, is:
Prime Number Theorem The number of primes less than X is approximately $\frac{X}{\log X}$
This means primes are quite common.
The number of primes less than 2^{512} is about 2^{503}

Prime Numbers
By the Prime Number Theorem if p is a number chosen at random then the probability it is prime is about:

$$\frac{1}{\log p}$$

So a random number p of 512 bits in length will be a prime with probability:

$$\approx \frac{1}{\log p} \approx \frac{1}{355}$$

So on average we need to select 177 odd numbers of size 2^{512} before we find one which is prime.
Hence, it is practical to generate large primes, as long as we can test primality efficiently

Primality Tests
For many cryptographic schemes, we need to generate large primes. This is usually done as follows:

- Select a random large number
- Test whether or not the number is a prime.

Naive approach to primality testing on n:

- Check if any integer from 2 to n-1 (or better: $\sqrt{n}$) divides n.

An improvement:

- Check whether n is divisible by any of the prime numbers $\leq \sqrt{n}$
- Can skip all numbers divisible by each prime number (Sieve of Eratosthenes)

These methods are too slow.

Sieve of Eratosthenes
To find prime numbers less than n:

- List all numbers $2, 3, 4, \ldots, n-1$
- Cross out all numbers with factor of 2, other than 2
- Cross out all numbers with factor of 3, other than 3, and so on
- Numbers that "fall through" sieve are prime

	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~

Primality Tests
Two varieties of primality test:

- Probabilistic
  - Identify probable primes with very low probability of being composite (in which case they are called pseudoprimes).
  - Much faster to compute than deterministic tests.
  - Examples:
    * Fermat
    * Solovay-Strassen
    * Miller-Rabin
- Deterministic
  - Identifies definite prime numbers.
  - Examples:
    * Lucas-Lehmer
    * AKS

Fermat Primality Test
Fermat's Little Theorem: if n is prime and $1 \leq a < n$, then:

$$a^{n-1} \equiv 1 \pmod{n}$$

To test if n is prime, a number of random values for a are chosen in the interval $1 < a < n-1$, and checked to see if the following equality holds for each value of a:

$$a^{n-1} \equiv 1 \pmod{n}$$

If n is composite then for a random $a \in \mathbb{Z}_n^*$:

$$\Pr[a^{n-1} \equiv 1 \pmod{n}] \leq 1/2$$

A composite number n is called a Fermat pseudoprime to base a if $a^{n-1} \equiv 1 \pmod{n}$.

Fermat Primality Test

```
Pick random a, 1 < a < n-1
if a^(n-1) (mod n)=1 then
   return PRIME
else
   return COMPOSITE
end
```

This test can be repeated k times to reduce the probability of classifying composites as primes.
If the algorithm outputs COMPOSITE at least once: output COMPOSITE; this will always be correct (a is called a Fermat witness).
If the algorithm outputs PRIME in all k trials: output PRIME (a Fermat pseudoprime); this will be an error with probability $(1/2)^k$.
Some composites always pass Fermat's test, and so are falsely identified as prime: the Carmichael Numbers.

Fermat Primality Test
Carmichael numbers are composite numbers n which fail Fermat's Test for every a not dividing n.

- Hence probable primes which are not primes at all.

There are infinitely many Carmichael Numbers

- The first three are 561, 1105, 1729

Carmichael Numbers n have the following properties:

- Always odd
- Have at least three prime factors
- Are square free
- If p divides n then $p-1$ divides $n-1$.

Fermat Primality Test
Example: consider $n = 15$.
The values computed for $a^{14} \pmod{15}$ for different values of a are as follows:

a	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$a^{14} \pmod{15}$	1	4	9	1	10	6	4	4	6	10	1	9	4	1

For $a = 1, 4, 11, 14$ the algorithm will output PRIME: these values are called Fermat liars.
For other values of a the algorithm will output COMPOSITE: these values are called Fermat witnesses.

Solovay-Strassen Primality Test
Euler's Criterion: if n is an odd prime and $a \in \mathbb{Z}_n^*$ then:

$$\left(\frac{a}{n}\right) \equiv a^{(n-1)/2} \pmod{n}$$

- $\left(\frac{a}{n}\right)$ is the Jacobi symbol.

- If n is composite then for a random $a \in \mathbb{Z}_n^*$:

$$Pr[\left(\frac{a}{n}\right) = a^{(n-1)/2}] \leq 1/2$$

Algorithm proposed by Solovay and Strassen (1973):

- A randomized algorithm.
- Never incorrectly classifies primes and correctly classifies composites with probability at least 1/2.

Solovay-Strassen Primality Test

```
Pick random a, 1 < a < n − 1
if gcd(a,n)>1 then
   return COMPOSITE
end
if (a/n) = a^((n−1)/2) then
   return PRIME
else
   return COMPOSITE
end
```

This test can be repeated k times to reduce the probability of classifying composites as primes.

- If the algorithm outputs COMPOSITE at least once: output COMPOSITE; this will always be correct (a is called an Euler witness).
- If the algorithm outputs PRIME in all the k trials: output PRIME (an Euler pseudoprime); this will be an error with probability $(1/2)^k$.

Solovay-Strassen Primality Test
Example: Consider n = 15.
For a = 3, 5, 6, 9, 10, 12 the algorithm will output COMPOSITE
For the other values of a which are relatively prime to n:

a	1	2	4	7	8	11	13	14
$(\frac{a}{15})$	1	1	1	-1	1	-1	-1	-1
$a^7 \pmod{15}$	1	8	4	13	2	11	7	14

For a = 1, 14 the algorithm will output PRIME: these values are called Euler liars.
For other values of a the algorithm will output COMPOSITE: these values are called Euler witnesses.

Miller-Rabin Primality Test

Let 2^k be the largest power of 2 dividing $n-1$.
Thus we have $n-1 = 2^k m$ for some odd number m.
Consider the sequence: $a^{n-1} = a^{2^k m}, a^{2^{k-1} m}, \ldots, a^m$.
We have set this sequence up so that each member of the sequence is a square root of the preceding member.
If n is prime, then by Fermat's Little Theorem, the first member of this sequence $a^{n-1} \equiv 1 \pmod{n}$.
When n is prime, the only square roots of 1 (mod n) are ± 1.
Hence either every element of the sequence is 1, or the first member of the sequence not equal to 1 must be -1 ($\equiv n-1 \pmod{n}$).
The Miller-Rabin test works by picking a random $a \in \mathbb{Z}_n$, then checking that the above sequence has the correct form.

Miller-Rabin Primality Test

```
Pick random a, 1 < a < n-1
b = a^m (mod n)
if b ≠ 1 and b ≠ n-1 then
    i=1
    while i < k and b ≠ n-1
        b = b^2 (mod n)
        if b = 1 then
            return COMPOSITE
        end
        i = i+1
    end
    if b ≠ n-1 then
        return COMPOSITE
    end
end
return PRIME
```

Miller-Rabin Primality Test

For any composite n the probability n passes the Miller-Rabin test is at most 1/4. On average it is significantly less.
The test can be repeated k times to reduce the probability of classifying composites as primes.

- If the algorithm outputs COMPOSITE at least once: output COMPOSITE; this will always be correct (a is called a strong witness).
- If the algorithm outputs PRIME in all the k trials: output PRIME (a strong pseudoprime); this will be an error with probability $(1/4)^k$.

Unlike the Fermat test, there are no composites for which no witness exists.

Miller-Rabin Primality Test
Example: Consider $n = 15$.
$n-1 = 14 = 2 \times 7$, so $k = 1$, $m = 7$.
For $a = 1, 14$ the algorithm will output PRIME: these values are called strong liars.
For other values of a the algorithm will output COMPOSITE: these values are called strong witnesses

Lucas-Lehmer Primality Test
A Mersenne number is an integer of the form $2^k - 1$, where $k \geq 2$.
If a Mersenne number is a prime, then it is called a Mersenne prime.
Subject of the Great Internet Mersenne Prime Search (GIMPS).
The Mersenne number $n = 2^k - 1$ ($k \geq 3$) is prime if and only if the following two conditions are satisfied:

1. k is prime
2. the sequence of integers defined by $b_0 = 4$, $b_{i+1} = (b_i^2 - 2) \pmod{n}$ ($i \geq 0$) satisfies $b_{k-2} = 0$.

This is the basis of the Lucas-Lehmer Primality Test.

Lucas-Lehmer Primality Test

```
if k has any factors between 2 and √k
    return COMPOSITE
end
b = 4
for i=1 to k-2 do
    b = (b^2 - 2) mod n
end
if b = 0 then
    return PRIME
else
    return COMPOSITE
```

AKS Primality Test
AKS algorithm discovered by Agrawal, Kayal and Saxena in 2002.
Result of many research efforts to find a deterministic polynomial-time algorithm for testing primality.
Based on the following property: if a and n are relatively prime integers with $n > 1$, n is prime iff:

$$(x-a)^n \equiv x^n - a \pmod{n}$$

where x is a variable.
Always returns correct answer.
Polynomial time algorithm, but still too inefficient to be used in practice.

AKS Primality Test

```
if n has the form a^b (b > 1) then
   return COMPOSITE
end
r = 2
while r < n
   if gcd(n,r) ≠ 1 then return COMPOSITE
   if r is a prime > 2 then
      q=largest factor of r-1
      if q > 4*√r*log n and n^((r-1)/q) ≠ 1 (mod r) then
         break
      end
      r = r+1
   end
end
for a=1 to 2*√r*log n do
   if (x-a)^n ≠ x^n - a (mod gcd(x^r - 1, n)) then return COMPOSITE
end
return PRIME
```

Primality Testing in Practice

The Miller-Rabin test is preferable to the Solovay-Strassen test for the following reasons:

- The Solovay-Strassen test is computationally more expensive.
- The Solovay-Strassen test is harder to implement since it also involves Jacobi symbol computations.
- The error probability for Solovay-Strassen is bounded above by $(1/2)^k$, while the error probability for Miller-Rabin is bounded above by $(1/4)^k$.
- From a correctness point of view, the Miller-Rabin test is never worse than the Solovay-Strassen test.

AKS is a breakthrough result: proves that PRIMES $\in$ P.

- Always gives correct results.
- No practical relevance: prohibitively slow run-times.

6 Public Key Cryptography

6.1 Introduction

Key Distribution Problem
Possible approaches to key distribution:

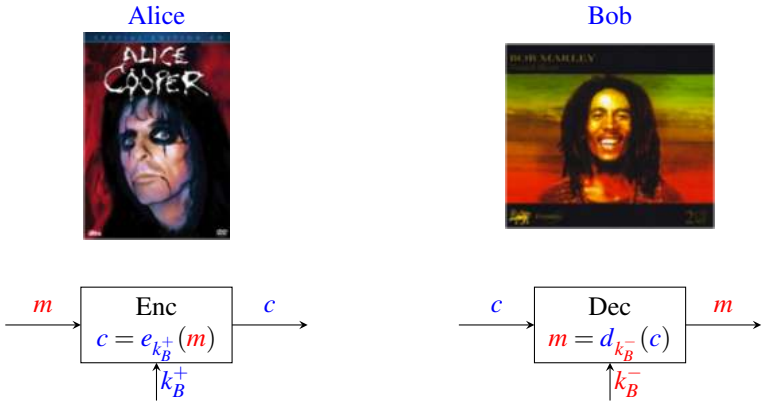
- Physical distribution: this is not scalable and the security no longer relies solely on the key.
- Distribution using symmetric key protocols.
- Distribution using public key protocols.

If we have n users each of whom wish to communicate securely with each other then we would require $\frac{n(n-1)}{2}$ secret keys.
One solution to this problem is for each user to hold only one key with which they communicate with a central authority, so n users will only require n keys.
When two users wish to communicate they generate a session key which is only to be used for that message; this can be generated with the help of the central authority and a security protocol.

Public Key Cryptography
With symmetric cryptography we can already provide confidentiality, integrity and authentication, so why invent something new?

- Key distribution problem
- Solution: instead of having a "lock" with a key that can lock and unlock, we could use a lock which has a key that can only lock and a second key that can only unlock.
  - Alice distributes locks and locking keys in public and only keeps the unlocking key private.
  - Then, everybody can grab a lock and the locking key, put a message in a box, lock the box and send it to Alice.
  - Only Alice can unlock the box and read the message.

Public Key Cryptography



Public key cryptography uses a different public key k^+ and private key k^- for encryption and decryption.
Here, Bob generates a key pair (k_B^+, k_B^-) and gives k_B^+ to Alice.

Public Key Cryptography

In public key cryptography, each user has a key pair, which consists of a public key (made public, used for encryption) and a private key (kept secret, used for decryption).

- Public key cryptography (asymmetric cryptography) realizes the idea described on the previous slide.
- The only (currently) known way to implement this idea in practice is to make use of (old) number theoretic problems.
- These problems lead to so-called one-way trapdoor functions.
  - These functions are easy to compute in one direction.
  - However, computing them in the other direction (computing the inverse) is very hard, without knowing some secret information.

Public Key Cryptography

The concept of public key cryptography was first thought of in 1976 in a paper by Diffie and Hellman: New Directions in Cryptography.
The first realisation of the concept appeared few years later: (RSA).
In the same 1976 paper, Diffie and Hellman described a method for establishing a shared secret key over an insecure communication channel: Diffie-Hellman key exchange.
It turns out that a lot of these ideas had already been developed in the UK GCHQ, but were subject to an official secrets act:

- Around 1970, James H. Ellis conceived the principles of public key cryptography.
- In 1973, Clifford Cocks invented a solution resembling the RSA algorithm.
- In 1974, Malcolm J. Williamson developed the Diffie-Hellman key exchange.

6.2 One-Way Functions

One-Way Functions
A function $f : X \to Y$ is a one-way function iff:

- For all $x \in X$ it is very easy or efficient to compute $f(x)$.
- For almost all $y \in Y$, finding an $x \in X$ with $f(x) = y$ is computationally infeasible.

A trapdoor one-way function is a one-way function $f : X \to Y$, but given some extra information, called the trapdoor information, it is easy to invert f i.e. given $y \in Y$, it is easy to find $x \in X$ such that $f(x) = y$.

One-Way Functions
Candidate one-way functions:
Multiplication:

- Given primes p and q, compute $n = pq$.
- This is very easy to compute, since we just multiply p and q.
- The inverse problem: given n find p and q (factoring).

Modular exponentiation:

- Given n and an element $a \in \mathbb{Z}_n$, compute $b \equiv a^m \pmod{n}$.
- This can be computed efficiently using squaring and multiplication.
- The inverse problem: given $n, a, b \in \mathbb{Z}_n$ find m such that $b \equiv a^m \pmod{n}$ (discrete logarithm problem).

6.3 Hard Problems

Hard Problems
Suppose you are given n but not p, q such that $n = pq$:

- Integer Factorisation Problem (IFP): Find p and q.
- RSA Problem (RSAP): Given $c \in \mathbb{Z}_n$ and integer e with $\gcd(e, \phi(n)) = 1$ find m such that $m^e \equiv c \pmod{n}$.
- Quadratic Residuosity Problem (QUADRES): Given a determine whether there is an x such that $a \equiv x^2 \pmod{n}$.
- Square Root Problem (SQROOT): Given a find x such that $a \equiv x^2 \pmod{n}$.

Hard Problems
Given an abelian group $(G, \otimes)$ and $g \in G$:

- Discrete Logarithm Problem (DLP):

  Given $y \in G$ find x such that $g^x = y$.

  The difficulty of this problem depends on the group G:

  - Very easy: polynomial time algorithm e.g. $(\mathbb{Z}_n, +)$
  - Rather hard: sub-exponential time algorithm e.g. $(GF(p), \times)$
  - Very hard: exponential time algorithm e.g. Elliptic Curve groups

- Diffie-Hellman Problem (DHP):

  Given $a = g^x$ and $b = g^y$ find $c = g^{xy}$.

- Decisional Diffie-Hellman Problem (DDH):

  Given $a = g^x$, $b = g^y$ and $c = g^z$, determine whether $z = xy$.

Hard Problems
IFP and DLP are believed to be computationally very difficult.
The best known algorithms for IFP and DLP are sub-exponential.
There is, however, no proof that IFP and DLP must be difficult.
Efficient quantum algorithms exist for solving IFP and DLP.
IFP and DLP are believed to be computationally equivalent.

Hard Problems
Some other hard problems:

- Computing discrete logarithms for elliptic curves.
- Finding shortest/closest vectors in a lattice.
- Solving the subset sum problem.
- Finding roots of non-linear multivariate polynomial equations.
- Solving the braid conjugacy problem.

6.4 Reductions

Reductions
We will reduce one hard problem to another, which will allow us to compare the relative difficulty of the two problems i.e. we can say:

Problem A is no harder than Problem B

Let A and B be two computational problems.
A is said to polytime reduce to B ($A \leq_P B$) if:

- There is an algorithm which solves A using an algorithm which solves B
- This algorithm runs in polynomial time if the algorithm for B does

Assume we have an oracle (efficient algorithm) to solve problem B.
We then use this oracle to give an efficient algorithm for problem A.

Reductions
Here we show how to reduce DHP to DLP i.e. we give an efficient algorithm for solving the DHP given an oracle for the DLP.
Given g^x and g^y we wish to find g^{xy}.
First compute y = DLP(g^y) using the oracle.
Then compute $(g^x)^y = g^{xy}$
So DHP is no harder than DLP i.e. DHP $\leq_P$ DLP.
Remark: in some groups we can show that DHP is equivalent to DLP.

Reductions
Here we show how to reduce DDH to DHP i.e. we give an efficient algorithm for solving the DDH given oracle for the DHP.
Given elements g^x, g^y and g^z, determine if $z = xy$
Using the oracle to solve DHP, compute g^{xy} = DHP(g^x,g^y).
Then check whether $g^{xy} = g^z$.
So DDH is no harder than DHP i.e. DDH $\leq_P$ DHP.
Remark: in some groups we can show that DDH is probably easier than DHP.

Reductions
Here we show how to reduce SQROOT to IFP i.e. we give an efficient algorithm for solving SQROOT given an oracle for IFP.
Given $z = x^2 \pmod{n}$ we wish to compute x:

- Using the oracle for IFP, find the prime factors p_i of n.
- Compute $\sqrt{z} \pmod{p_i}$ (can be done in polynomial time)
- Recover $\sqrt{z} \pmod{n}$ using CRT on the data $\sqrt{z} \pmod{p_i}$

We have to be a little careful if powers of p_i greater than one divide n.
So SQROOT is no harder than IFP i.e. SQROOT $\leq_P$ IFP.

Reductions
Here we show how to reduce IFP to SQROOT i.e. we give an efficient algorithm for IFP given an oracle for SQROOT.
Given $n = pq$ we wish to compute p and q:

- Compute $z = x^2$ for a random $x \in \mathbb{Z}_n^*$
- Compute $y = \sqrt{z} \pmod{n}$ using the oracle for SQROOT.

  - There are four possible square roots, since there are two factors.
  - With fifty percent probability we have $y \neq \pm x \pmod{n}$

- Factor n by computing $\gcd(x-y,n)$.

So IFP is no harder than SQROOT i.e. IFP $\leq_P$ SQROOT.
So IFP and SQROOT are computationally equivalent: SQROOT $\equiv_P$ IFP.

Reductions
Here we show how to reduce RSAP to IFP i.e. we give an efficient algorithm for solving RSAP given an oracle for IFP.
Given $c = m^e \pmod{n}$ and the integer e, find m:

- Find the factorization of $n = pq$ using the oracle.
- Compute $\phi(n)$ as $\phi(n) = (p-1)(q-1)$
- Using the XGCD compute $d = 1/e \pmod{\phi(n)}$
- Finally, recover $m = c^d \pmod{n}$

So RSAP is no harder than IFP i.e. RSAP $\leq_P$ IFP.
There is some evidence (although slight) that RSAP might be easier.

6.5 Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange
For Diffie-Hellman, we select a large prime $p(> 2^{400})$ and a generator g, then:

1. A chooses a random x such that $1 < x < p-1$.
2. $A \rightarrow B : g^x \pmod{p}$
3. B chooses a random y such that $1 < y < p-1$.
4. $B \rightarrow A : g^y \pmod{p}$
5. A computes $K = (g^y)^x \pmod{p}$.
6. B computes $K = (g^x)^y \pmod{p}$.
7. A and B now share the secret K.

Diffie-Hellman Key Exchange
A toy example ($p = 11$, $g = 5$).
Private keys:

- $x = 3$
- $y = 4$

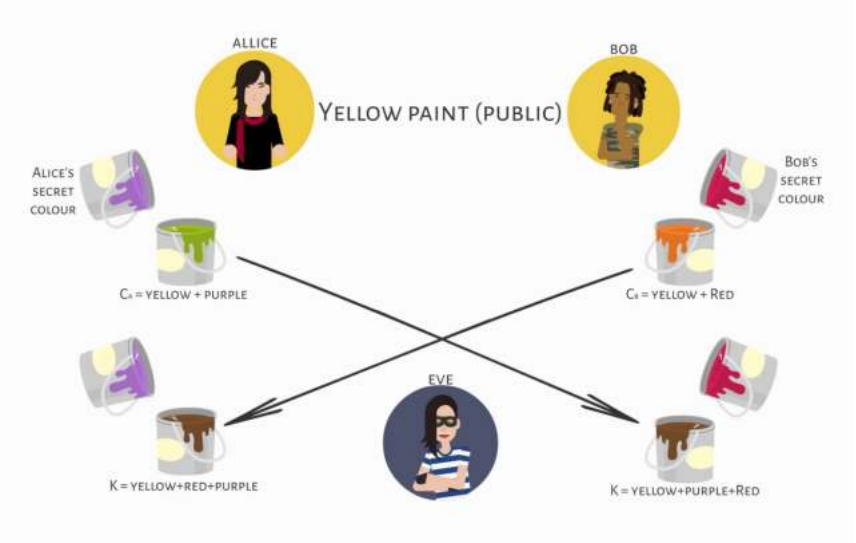
Public keys:

- $X = g^x \pmod p = 5^3 \pmod{11} = 125 \pmod{11} = 4$
- $Y = g^y \pmod p = 5^4 \pmod{11} = 625 \pmod{11} = 9$

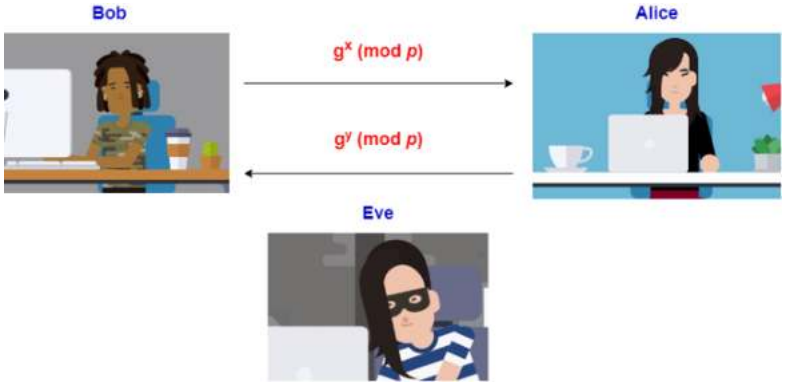
Shared secret:

- $Y^x \pmod p = 9^3 \pmod{11} = 729 \pmod{11} = 3$
- $X^y \pmod p = 4^4 \pmod{11} = 256 \pmod{11} = 3$

Diffie-Hellman Key Exchange
This is analagous to paint mixing:



Diffie-Hellman Key Exchange
Consider an eavesdropper Eve.



Diffie-Hellman Key Exchange
Bob and Alice use $g^{xy} \pmod p$ as a shared key.
Eve can see $g^x \pmod p$ and $g^y \pmod p$
Note $g^x g^y = g^{x+y} \neq g^{xy} \pmod p$.
If Eve can find x or y, system is broken.
If Eve can solve the discrete log problem, then she can find x or y.
This is a difficult problem - modular exponentiation is a one-way function.

Diffie-Hellman Key Exchange
This key exchange is susceptible to a man in the middle attack:

Eve shares secret $g^{xz} \pmod p$ with Alice and secret $g^{yz} \pmod p$ with Bob.
Alice and Bob do not know that Eve exists.

6.6 Public Key Encryption

Public Key Encryption
The basic idea of public key encryption is:

Message	+	Bob's Public Key	=	Ciphertext
Ciphertext	+	Bob's Private Key	=	Message

Anyone with Bob's public key can send Bob a secret message.
But only Bob can decrypt the message, since only Bob has the private key.
All one needs to do is look up Bob's public key in some directory.

Public Key Encryption

RSA

Rivest, Shamir, Adleman (1978): A Method for Obtaining Digital Signatures and Public Key Cryptosystems.

Key generation: Generate two large primes p and q of at least 512 bits.

- Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$
- Select a random integer e, $1 < e < \phi(n)$, such that $\gcd(e, \phi(n)) = 1$.
- Using the extended Euclidean algorithm compute the unique integer d, $1 < d < \phi(n)$ with $ed \equiv 1 \pmod{\phi(n)}$.

Public key = (e,n) which can be published.
Private key = d which needs to be kept secret.

RSA

Encryption: if Bob wants to encrypt a message for Alice, he does the following:

- Obtains Alice's authentic public key (e,n).
- Represents the message as a number $0 < m < n$.
- Computes $c = m^e \pmod{n}$.
- Sends the ciphertext c to Alice.

Decryption: to recover m from c, Alice uses the private key d to recover $m = c^d \pmod{n}$.

RSA
Recall that $ed \equiv 1 \pmod{\phi(n)}$, so there exists an integer k such that:

$$ed = 1 + k\phi(n)$$

If gcd(m, p)=1:

- By Fermat's Little Theorem we have $m^{p-1} \equiv 1 \pmod{p}$.
- Taking $k(q-1)-$th power and multiplying by m yields:

$$m^{1+k(p-1)(q-1)} \equiv m \pmod{p} \qquad (*)$$

If gcd(m, p) = p, then $m \equiv 0 \pmod{p}$ and $(*)$ is valid again.
Hence, in all cases $m^{ed} \equiv m \pmod{p}$ and by a similar argument we have $m^{ed} \equiv m \pmod{q}$.
Since p and q are distinct primes, the CRT leads to:

$$c^d = (m^e)^d = m^{ed} = m^{k(p-1)(q-1)+1} = m \pmod{n}$$

RSA: Toy Example
Choose primes $p = 7$ and $q = 11$.
Key Generation:

- Compute n =77 and $\phi(n) = (p-1)(q-1) = 6 \times 10 = 60$.
- Choose e = 37, which is valid since gcd(37,60) = 1.
- Using the extended Euclidean algorithm, compute d = 13 since $37 \times 13 \equiv 481 \equiv 1 \pmod{60}$.
- Public key = (e=37,n=77) and private key d = 13.

Encryption: suppose m = 2 then:

$$c \equiv m^e \pmod{n} \equiv 2^{37} \pmod{77} \equiv 51$$

Decryption: to recover m compute:

$$m \equiv c^d \pmod{n} \equiv 51^{13} \pmod{77} \equiv 2$$

RSA
The security of RSA relies on the difficulty of finding d given n and e.
RSAP can be reduced to IFP, since if we can find p and q, then we can compute d.
Therefore, if factoring is easy we can break RSA.

- Currently 768-bit numbers are the largest that have been factored.
- For medium term security, best to choose 1024-bit numbers.

RSA
Assume for efficiency that each user has:

- The same modulus n
- Different public/private exponents (e_i, d_i)

Suppose user one wants to find user two's d_2:

- User one computes p and q since they know d_1
- User one computes $\phi(n) = (p-1)(q-1)$
- User one computes $d_2 = (1/e_2) \pmod{\phi(n)}$

So each user can then find every other users key.

RSA
Now suppose the attacker is not one of the people who share a modulus.
Suppose Alice sends the message m to two people with public keys:

- (n, e_1),(n, e_2), i.e. $n_1 = n_2 = n$.

Eve can see the messages c_1 and c_2 where:

- $c_1 = m^{e_1} \pmod{n}$
- $c_2 = m^{e_2} \pmod{n}$

RSA
Eve can now compute:

- $t_1 = e_1^{-1} \pmod{e_2}$
- $t_2 = (t_1 e_1 - 1)/e_2$

Eve can then compute the message from:

$$\begin{aligned} c_1^{t_1} c_2^{-t_2} &= m^{e_1 t_1} m^{-e_2 t_2} \pmod{n} \\ &= m^{1+e_2 t_2} m^{-e_2 t_2} \pmod{n} \\ &= m^{1+e_2 t_2 - e_2 t_2} \pmod{n} \\ &= m^1 = m \pmod{n} \end{aligned}$$

RSA: Example
Take the public keys as:

- $n = n_1 = n_2 = 18923$
- $e_1 = 11$, $e_2 = 5$

Take the ciphertexts as:

- $c_1 = 1514$, $c_2 = 8189$
- The associated plaintext is $m = 100$

Then $t_1 = 1$ and $t_2 = 2$
We can now compute the message from: $c_1^{t_1} c_2^{-t_2} = 100 \pmod{n}$

RSA
Modular exponentiation is computationally intensive.
Even with the square-and-multiply algorithm, RSA can be quite slow on constrained devices such as smart cards.
Choosing a small public exponent e can help to speed up encryption.
Minimal possible value: $e = 3$

- 2 does not work since $\gcd(2, \phi(n)) = 2$
- However, sending the same message encrypted with 3 different public keys then breaks naive RSA.

RSA
Suppose we have three users:

- With public moduli n_1, n_2 and n_3
- All with public exponent $e = 3$

Suppose someone sends them the same message m
The attacker sees the messages:

- $c_1 = m^3 \pmod{n_1}$
- $c_2 = m^3 \pmod{n_2}$
- $c_3 = m^3 \pmod{n_3}$

Now the attacker, using the CRT, computes the solution to: $X = c_i \pmod{n_i}$
To obtain $X \pmod{n_1 n_2 n_3}$

RSA
So the attacker has: $X \pmod{n_1 n_2 n_3}$
But since $m^3 < n_1 n_2 n_3$ we must have $X = m^3$ over the integers.
Hence $m = X^{1/3}$.
This attack is interesting since we find the message without factoring the modulus.
This is evidence that breaking RSA can be easier than factoring.

RSA
Choosing a small private key d results in security weaknesses.

- In fact, d must have at least $0.3\log_2 n$ bits

The Chinese Remainder Theorem can be used to accelerate exponentiation with the private key d.

- Based on the CRT we can replace the computation of $c^{d \pmod{\phi(n)}} \pmod{n}$ by two computations $c^{d \pmod{p-1}} \pmod{p}$ and $c^{d \pmod{q-1}} \pmod{q}$ where p and q are 'small' compared to n.

RSA: Example
Take $n_1 = 323$, $n_2 = 299$, $n_3 = 341$
The attacker sees $c_1 = 50$, $c_2 = 268$, $c_3 = 1$ and wants to determine the value of m
The attacker computes via CRT: $X = 300763 \pmod{n_1 n_2 n_3}$
The attacker computes, over the integers: $m = X^{1/3} = 67$
Lessons

- Plaintexts should be randomised before applying RSA.
  - add random value to messages, so same message is not encrypted multiple times.
- Very small exponents should be avoided for RSA encryption.
  - Recommended value: $e = 65537 = 2^{16} + 1$
  - Runtime for encryption: 17 modular multiplications.
- Runtime of RSA: fast encrypt/slow decrypt

RSA: Semantic Security
Here we shall see that RSA reveals information about the message being encrypted.
Suppose the attacker knows that you only send one of two messages m_1 or m_2.
These could be buy or sell, yes or no etc.
On receiving the ciphertext c the attacker computes $c' = m_1^e \pmod{n}$.
If $c' = c$ then the attacker knows that m_1 was the message.
If $c' \neq c$ then the attacker knows that m_2 was the message.

RSA: Semantic Security
We see that RSA used naively leaks some information about encrypted messages.

- It does not have semantic security.

The problem is that the attacker has access to the (deterministic) encryption function.

- It is a public key scheme after all.

The problem could be solved if the encryption function was not deterministic.

- Randomisation is therefore required.

Raw RSA
Raw RSA is not semantically secure.
To make it semantically secure we use a padding scheme to add randomness and redundancy.
Note: Some old padding schemes are now considered weak.
Bellare and Rogaway have a scheme called OAEP (Optimal Assymetric Encryption Padding) which was believed to provide plaintext awareness when used with raw RSA (in the random oracle model).
OAEP is used in the PKCS standards and hence in most Internet protocols.

RSA
The OAEP padding scheme is as follows.
Let r denote a random value and m the plaintext s.t. $(m||0||r) < n$
Let G and H be pseudo-random functions (e.g. crypto hash functions).
We define the padding scheme: $\text{OAEP}(m||0||r) = (G(r) \oplus (m||0))||(H(m||0 \oplus G(r)) \oplus r)$
Encryption is performed by: $c = (\text{OAEP}(m||0||r))^e \pmod{n}$
Decryption is performed by: $(m||z||r) = \text{OAEP}^{-1}(c^d \pmod{n})$
Followed by verification that $z = 0$.
If $z = 0$, the decrypted message is m.
Otherwise, the ciphertext was forged, and the decrypted value should be ignored.

ElGamal
ElGamal (1985): A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.

- Domain Parameter Generation: Generate a "large prime" p (> 512 bits) and generator g of the multiplicative group $\mathbb{Z}_p^*$,
- Key Generation: Select a random integer x, $0 < x < p$ and compute $y \equiv g^x \pmod{p}$.
- Public key = (p, g, y) which can be published.
- Private key = x which needs to be kept secret.

ElGamal
Encryption: Bob encrypts a message for Alice as follows:

- Obtains Alice's authentic public key (p, g, y).
- Represents the message as an integer m where $0 \leq m \leq p-1$.
- Generates a random ephemeral key k, with $0 < k < p$.
- Computes $c_1 = g^k \pmod{p}$ and $c_2 = my^k \pmod{p}$.
- Sends the ciphertext $c = (c_1||c_2)$ to Alice.

Decryption: to recover the message, Alice does the following:

- Uses the private key x to compute $c_1^{p-1-x} \pmod{p} \equiv c_1^{-x} \equiv g^{-xk}$.
- Recovers m by computing $(c_1^{-x})c_2 \equiv m \pmod{p}$.

Proof that decryption works: $(c_1^{-x})c_2 \equiv g^{-xk} m g^{xk} \equiv m \pmod{p}$.

ElGamal: Toy Example
Toy example:
Select prime $p = 17$.
Generator $g = 6$.
Alice chooses the private key $x = 5$ and computes:

$$y \equiv g^x \pmod{p} \equiv 6^5 \pmod{17} \equiv 7$$

Alice's public key is $(p = 17, g = 6, y = 7)$, which can be published.
Alice's private key is $x = 5$ which she keeps secret.

ElGamal: Toy Example
To encrypt the message $m = 13$, Bob selects a random integer $k = 10$ and computes:

$$c_1 = g^k \pmod{p} = 6^{10} \pmod{17} = 15$$
$$c_2 = my^k \pmod{p} = (13 \times 7^{10}) \pmod{17} = 9$$

Bob then sends the ciphertext $(c_1||c_2)$ to Alice.
To decrypt, Alice first computes:

$$c_1{}^{p-1-x} \pmod{p} \equiv 15^{11} \pmod{17} \equiv 9 \pmod{17}$$

and recovers m by computing:

$$m \equiv 9 \times 9 \pmod{17} \equiv 13 \pmod{17}$$

Rabin Cryptosystem
Rabin (1979): Digitalized Signatures and Public Key Functions as Intractable as Factorization.
Breaking RSA has not been proven to be equivalent with IFP.
SQROOT and IFP are computationally equivalent.
The Rabin Cryptosystem is based on the difficulty of extracting square roots modulo $n = pq$.
The Rabin Cryptosystem was the first provably secure scheme.
Despite all its advantages over RSA it is not widely used in practice.

Rabin Cryptosystem
Key Generation:

- Generate two large primes p and q of roughly the same size.
- Compute $n = pq$.

Public key = n which can be published.
Private key = (p, q) which needs to be kept secret.
Note: we always take $p \equiv q \equiv 3 \pmod{4}$, since this makes extracting square roots easy and fast.

Rabin Cryptosystem
Encryption: to encrypt a message Bob does the following:

- Obtain Alice's authentic public key n.
- Represent the message as an integer m where $0 \leq m \leq n-1$.
- Compute the ciphertext c as $c = m^2 \pmod{n}$

Decryption: to recover the plaintext m from c Alice computes:

$$m = \sqrt{c} \pmod{n}$$

Notice at first sight this uses no private information, but we need the factorization to be able to find the square root.

Rabin Cryptosystem
The decryption operation in the Rabin Cryptosystem hides a small problem:

- n is the product of two primes p and q.
- Therefore there are four possible square roots modulo n.
- On decryption obtain four possible plaintexts.
- Need to add redundancy to plaintext to decide which one to take.

Rabin encryption is very, very fast.
Decryption is made faster by the special choice of p and q.

Rabin Cryptosystem: Toy Example
Alice chooses the primes $p = 127$ and $q = 131$.
Note that both primes are congruent to 3 modulo 4.
Alice then computes $n = pq = 16637$.
Suppose Bob wants to encrypt the message $m = 4410$, then he computes the ciphertext as:

$$c = m^2 \pmod{n} = 16084$$

Rabin Cryptosystem: Toy Example
To decrypt the ciphertext c, Alice computes $\sqrt{16084} \pmod{16637}$ by computing:

- $\sqrt{16084} \pmod{127} \equiv \pm 16084^{32} \pmod{127} \equiv \pm 35 \pmod{127}$
- $\sqrt{16084} \pmod{131} \equiv \pm 16084^{33} \pmod{131} \equiv \pm 44 \pmod{131}$

Using the CRT, we obtain four possible square roots:

$$s \equiv \pm 4410 \text{ or } \pm 1616$$

Therefore, the possible messages are:

$$s \equiv 4410 \text{ or } 12227 \text{ or } 1616 \text{ or } 15021$$

6.7 Digital Signatures

Digital Signatures
If a user encrypts data with their private key, then anyone can decrypt the data using the user's public key:

- This approach does not provide confidentiality.
- However, anyone receiving encrypted data can be sure that it was encrypted by the owner of the key pair.
- This is the basis for a digital signature.
- This allows us to obtain authentication and non-repudiation.

The basic idea of using public key cryptography for digital signatures is:

$$\begin{aligned} \text{Message} + \text{Alice's Private Key} &= \text{Signature} \\ \text{Message} + \text{Signature} + \text{Alice's Public Key} &= \text{Yes/No} \end{aligned}$$

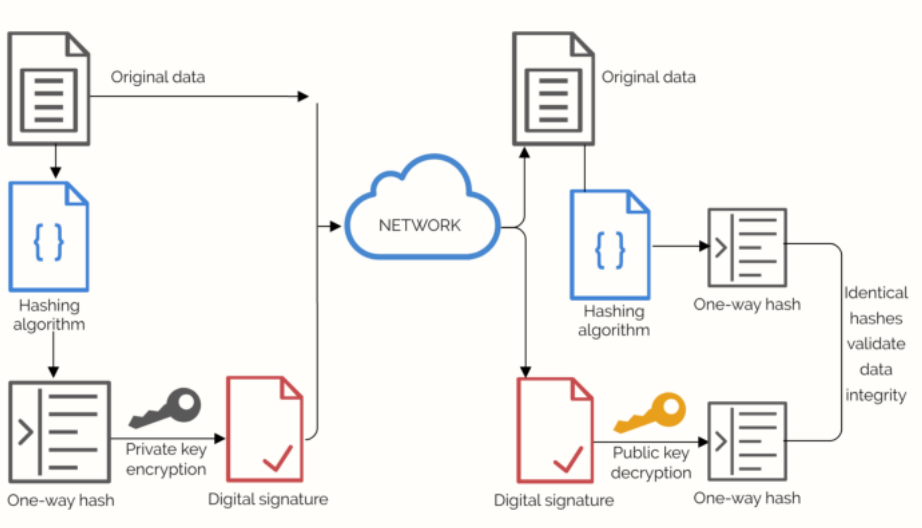
Only Alice can have encrypted the message, since only Alice has the private key.

Digital Signatures
Cryptographically secure digital signature schemes have two parts: the signing protocol and the authentication process:

- In the signing protocol, the signer generates a cryptographic hash of the data to be signed, and encrypts this with their private key to produce the digital signature. This signature is then appended to the data and sent to the verifier.
- In the authentication process, the verifier generates the same cryptographic hash of the data, decrypts the digital signature using the public key of the signer and verifies that these values are the same.

It is possible to sign the original data digitally, but this is usually done on a hash of the data instead for efficiency as the hash will generally be much smaller.

Digital Signatures



RSA Signature

Key generation: Generate two large primes p and q of at least 512 bits.

- Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$
- Select a random integer e, $1 < e < \phi(n)$, where $\gcd(e, \phi(n)) = 1$.
- Using the extended Euclidean algorithm compute the unique integer d, $1 < d < \phi(n)$ with $ed \equiv 1 \pmod{\phi(n)}$.

Public key = (e,n) which can be published.
Private key = d which needs to be kept secret.

RSA Signature

Signature: if Alice wants to sign message m for Bob, she does the following:

- Computes $s = h(m)^d \pmod{n}$
- Sends the signature s to Bob

Verification: if Bob wants to verify the signature s for message m from Alice, he does the following:

- Obtains Alice's authentic public key (e,n)
- Computes $v = s^e \pmod{n}$
- Checks that $h(m) = v$

ElGamal Signature
Domain Parameter Generation

- Generate a "large prime" p (> 512 bits) and generator g of the multiplicative group $\mathbb{Z}_p^*$

Key Generation

- Select a random integer x, $0 < x < p-1$ and compute $y \equiv g^x \pmod{p}$
- Public key = (p,g,y) which can be published.
- Private key = x which needs to be kept secret.

ElGamal Signature
Signature: if Alice wants to sign message m for Bob, she does the following:

- Generates a random ephemeral key k with $1 < k < p-1$ and gcd(k,$p-1$) = 1
- Computes $s_1 = g^k \pmod{p}$
- Computes $s_2 = k^{-1}(h(m) - xs_1) \pmod{p-1}$ where h is the hash function SHA-256 (if $s_2 = 0$ start over again).
- $(s_1||s_2)$ is the digital signature of m

Verification: if Bob wants to verify the signature $(s_1||s_2)$ for message m from Alice, he does the following:

- Obtains Alice's authentic public key (p,g,y)
- Computes $v_1 = g^{h(m)} \pmod{p}$
- Computes $v_2 = y^{s_1} s_1^{s_2} \pmod{p}$
- Checks that $v_1 = v_2$

ElGamal Signature
Proof of Correctness

$$\begin{aligned} v_2 &= y^{s_1} s_1^{s_2} \pmod{p} \\ &= g^{xs_1} g^{ks_2} \pmod{p} \\ &= g^{xs_1 + kk^{-1}(h(m) - xs_1) \pmod{p-1}} \pmod{p} \\ &= g^{xs_1 + kk^{-1}(h(m) - xs_1)} \pmod{p} \text{ (by Fermat's Little Theorem)} \\ &= g^{xs_1 + (h(m) - xs_1)} \pmod{p} \\ &= g^{h(m)} \pmod{p} \\ &= v_1 \end{aligned}$$

Digital Signature Standard (DSS)
FIPS PUB 186 by NIST, 1991 (final announcement 1994)

- Secure Hashing Algorithm (SHA) for hashing
- Digital Signature Algorithm (DSA) for signature
- The hash code is set as input of DSA
- The signature consists of two numbers

DSA:

- Based on the difficulty of discrete logarithm problem
- Based on ElGamal and Schnorr system

DSA
Domain Parameter Generation

- Generate a "large prime" p (>512 bits) with prime divisor q of $p-1$ (160 bits)
- Select a random integer h, $1<h<p-1$ and compute $g \equiv h^{(p-1)/q} \pmod{p}$ (if $g=1$ start over again)

Key Generation

- Select a random integer x, $0<x<q$ and compute $y \equiv g^x \pmod{p}$
- Public key = (p,q,g,y) which can be published.
- Private key = x which needs to be kept secret.

DSA
Signature: if Alice wants to sign message m for Bob, she does the following:

- Generates a random ephemeral key k with $0<k<p-1$ and gcd(k,$p-1$) = 1
- Computes $s_1 = (g^k \pmod{p}) \pmod{q}$
- Computes $s_2 = k^{-1}(h(m)+xs_1) \pmod{q}$
- $(s_1||s_2)$ is the digital signature of m

Verification: if Bob wants to verify the signature $(s_1||s_2)$ for message m from Alice, he does the following:

- Computes $w = s_2^{-1} \pmod{q}$
- Computes $u_1 = h(m)w \pmod{q}$
- Computes $u_2 = s_1 w \pmod{q}$
- Computes $v = (g^{u_1} y^{u_2} \pmod{p}) \pmod{q}$
- Checks that $v = s_1$

DSA

Proof of Correctness

$$\begin{aligned} v &= (g^{u_1} y^{u_2} \pmod p) \pmod q \\ &= (g^{h(m)w \pmod q} y^{s_1 w \pmod q} \pmod p) \pmod q \\ &= (g^{h(m)w \pmod q} g^{x s_1 w \pmod q} \pmod p) \pmod q \\ &= (g^{h(m)w + x s_1 w \pmod q} \pmod p) \pmod q \\ &= (g^{(h(m)+x s_1)w \pmod q} \pmod p) \pmod q \\ &= (g^{(h(m)+x s_1)k(h(m)+x s_1)^{-1} \pmod q} \pmod p) \pmod q \\ &= (g^k \pmod p) \pmod q \\ &= s_1 \end{aligned}$$

RSA vs DSA Signature

RSA

- Deterministic signatures: for each message, one valid signature exists
- Faster verifying than signing

DSA

- Non-deterministic signatures: for each message, many valid signatures exist
- Faster signing than verifying

Message may be signed once, but verified many times: this prefers the faster verification
Signer may have limited computing power (e.g. smart card): this prefers the faster signing

Blind Signature

Suppose Alice has a message m that she wishes to have signed by Bob, and she does not want Bob to learn anything about m.

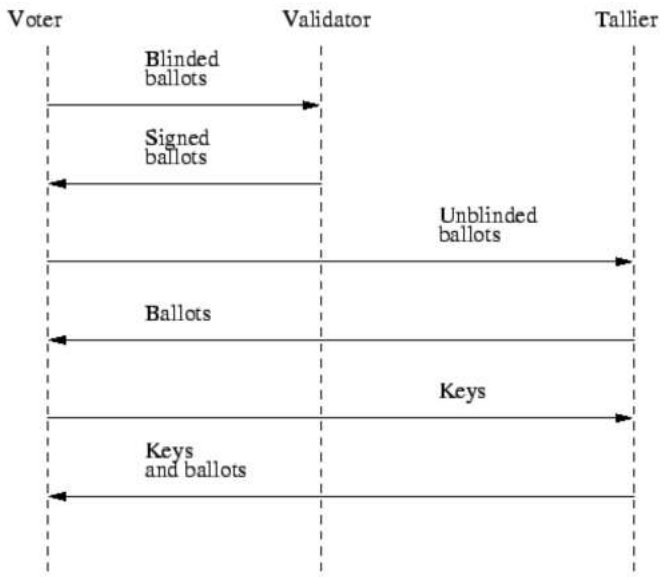
- Let (n, e) be Bob's public key and d be his private key.
- Alice generates a random value b such that $\gcd(b, n) = 1$ and sends $x = (b^e \cdot m) \pmod n$ to Bob.
- The value x is blinded by the random value b; hence Bob can derive no useful information from it.
- Bob returns the signed value $t = x^d \pmod n$ to Alice.
- Since $x^d \equiv (b^e \cdot m)^d \equiv b \cdot m^d \pmod n$, Alice can obtain the true signature s of m by computing $s = b^{-1} \cdot t \pmod n$

Blind Signature

In an online election context a blind signature can be used as follows:

- Voter encrypts their ballot with a secret key and then blinds it.
- Voter then signs the encrypted vote and sends it to the validator.
- The validator checks to see if the signature is valid and if it is the validator signs it and returns it to the voter.
- The voter removes the blinding encryption layer, which then leaves behind an encrypted ballot with the validator's signature.
- This is then sent to the tallier who checks to make sure the validator's signature is present on the votes.
- He then waits until all votes have been collected and then publishes all the encrypted votes so that the voters can verify their votes have been received.
- The voters then send their keys to the tallier to decrypt their ballots.
- Once the vote has been counted the tallier publishes the encrypted votes and the decryption keys so that voters can then verify the results.

Blind Signature



Blind Signature

- This protocol has been implemented and used in reality and it has been found that the entire voting process can be completed in a matter of minutes despite the complex nature of the voting procedure.

- Most of the tasks can be automated with the only user interaction needed being the actual vote casting.
- Encryption, blinding and all the verification needed can be performed by software in the background.
- Of course we would have to trust this software to handle the voting procedures correctly and accurately and to assume it has not been compromised in some way.

6.8 Other Public Key Algorithms

Other Public Key Algorithms

Encryption algorithm	Security depends on
Goldwasser-Micali encryption	QUADRES
Blum-Goldwasser encryption	SQROOT
Chor-Rivest encryption	Subset sum problem
XTR	DLP
NTRU	Closest vector problem in lattices
Schnorr signature	DLP
Nyberg-Rueppel signature	DLP
Elliptic Curve DSA (ECDSA)	DLP in elliptic curves

6.9 Public Key Cryptography in Practice

Public Key Cryptography in Practice
Main drawback of public key cryptography is the inherently slow speed.

- A few schemes are faster, but require huge keys, so are impractical.

Therefore, public key schemes are not used directly for encryption.
Instead, they are used in conjunction with secret key schemes.

- Encryption is performed by secret key schemes (e.g. AES).
- Key agreement is performed by public key schemes (e.g. RSA or Diffie-Hellman).

Public Key Cryptography in Practice
In secret key schemes key sizes have increased from 56-64 bits to 128 bits to provide sufficient security.

- Keys of 128 bits are large enough to thwart any practical attack, as long as the cipher does not have weakness due to its design.

In public key schemes, considerably longer keys are required (e.g. 1024 bits for RSA, 512 bits for ElGamal).

- Keys are not uniformly selected from all the possible keys with the same length.

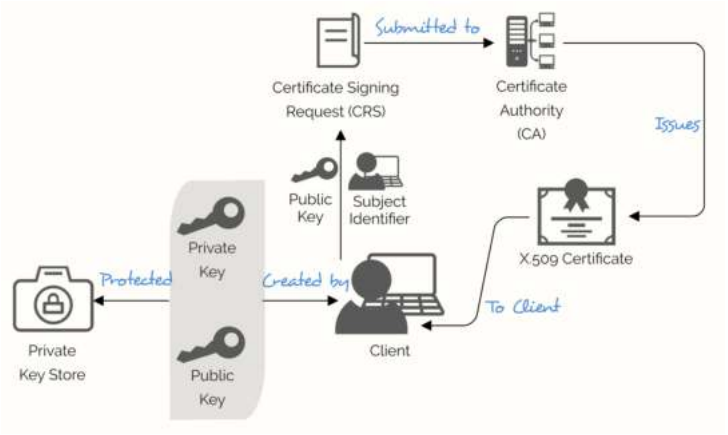
- The number of keys is (slightly) smaller than the number of values of the same length as the keys.
- The key inherits information due to the properties of the cipher.

6.10 Public Key Infrastructures

Public Key Infrastructures (PKIs)
Public key cryptography provides a tool for secure communication between parties by letting them trust messages encrypted or signed by the already known public keys of the other parties.
However, no algorithmic scheme can solve the original trust problem of accepting the identity of a party that you never met.
The usual face-to-face identification is by a trusted third party (e.g. a friend) who presents the two parties to each other.
Such a presentation protocol is also required for cryptographic protocols.
The presenting party in the cryptographic environment is called a certification authority (CA).
The management of the CAs requires a public key infrastructure (PKI).

Public Key Infrastructures (PKIs)
During face-to-face presentation, the presenter gives the relation between the name and the face of the presented party, together with some side information (e.g. they are a friend).
For cryptographic use the certification authority should give the relation between the public key and the identity of the owner.
This information should be transmitted authenticated from the CA to the receiver, e.g. signed under the widely known public key of the CA.
Thus, the receiver should only verify the signature of the CA, rather than communicate with the CA to verify every new key.
Such a CA signature is called a certificate.

Public Key Infrastructures (PKIs)



Certificates
Before using a public key, we need at least the following information:

- The name of the entity that owns the key.
- The dates during which the key is valid.
- An indication as to whether or not the private key has been compromised or is no longer in use for some other reason.

The first two pieces of information are static and can be assigned when a key pair is generated.
This can be done by some trusted entity issuing a certificate that certifies the relationship between a public key and its owner.

Certificates
A certificate should contain (at least) the following information:

- A public key.
- The name of the owner of the public key (the subject).
- The dates during which the public key is valid.
- The identity of the issuer of the certificate.

To prevent a certificate from being modified, it is digitally signed by its issuer.
If we have an issuer's public key, then we can check the integrity of a certificate.
If we trust the issuer, we know that the given public key belongs to the subject of the certificate.

Certificates
Information about compromised or invalid keys required before a public key is used is more difficult to obtain.
By its nature, this information cannot be included in a certificate.
There are various ways of supplying the information, but in all cases, we need to revoke a certificate.
The issuer of a certificate needs some way of indicating that the certificate has been revoked and that the certificate should no longer be used as justification for believing that the public key is valid.
Note that the public key could still be used if there was another valid certificate.

Certificates
All PKIs use some form of certificate. However, PKIs differ in a number of ways:

- The format for certificates and what information in addition to public keys can be certified: some certificates contain serial numbers, some contain details of the certification policy and some contain details of the uses to which a public key can be put.
- How entities are named (identified).
- How a user can determine if a certificate has been revoked: some PKIs maintain lists of revoked certificates, while others allow a user to query a server.
- How trust is handled: this is the most important difference between PKIs: some PKIs have a centralised view of trust in which trusted third parties (TTPs) act as CAs e.g. X.509 PKI, others have a more distributed view in which users trust each other e.g. PGP PKI.

The `X.509` PKI

- `X.509` is the most widely used technology for implementing PKIs.
  - In fact, when most people talk about PKIs, they are talking about `X.509` PKIs.
- An `X.509` certificate contains the following fields:

Field	Description
Serial Number	The serial number of this certificate. Each CA gives their certificates a unique serial number.
Issuer	The name of the CA that issued of the certificate.
Subject	The name of the owner of the public key being certified.
Validity Dates	`from` and `to` dates defining the period of validity of the certificate.
Public Key	The public key being certified.
...	...
Signature	Issuer's signature of the certificate.

The `X.509` PKI

- We will represent a certificate as $\mathscr{C}[N, I, S, (F, T), K]$ where

  N is the serial number of the certificate.

  I is the name of the issuer.

  S is the name of the subject.

  (F, T) are the validity dates.

  K is the public-key being certified.

The `X.509` PKI

- `X.509` certificates are issued by entities known as Certification Authorities (CAs).
- Normally a CA is a trusted party or in some cases, a trusted third party.
  - They can be trusted to bind a subject name to the proper public key.
  - The precise manner in which a CA checks these relationships is not standardized.
  - Some CAs do simple checks, while others require legally notified documents etc.
- When a certificate is obtained its signature must be checked by using the issuing CA's public key.
  - For this to work, we must have secure access to the CA's public key.
  - This requires us to get the CA's certificate.
  - The name of the CA is given by the issuer field of the certificate so, if necessary, the CA's certificate can be obtained from a directory.
- In general, the CA's certificate will have been issued by a higher-level CA and we will need to repeat the validation process.

The `X.509` PKI

- A certificate path is a list of certificates $\langle \mathscr{C}_0, \mathscr{C}_1, \cdots \rangle$ such that the signature for $\mathscr{C}_i$ was generated by the private key corresponding to the public key certified by $\mathscr{C}_{i+1}$
- For example:

$\langle \mathscr{C}[\texttt{23}, \texttt{DCU CA}, \texttt{Geoff Hamilton}, (\texttt{Jan 2020}, \texttt{Dec 2021}), k^+_{GH}],$
$\mathscr{C}[\texttt{3452}, \texttt{VeriSign}, \texttt{DCU CA}, (\texttt{Jan 2015}, \texttt{Dec 2024}), k^+_{DCU}], \cdots \rangle$

- To be practical, certificate paths must be finite.
- This is achieved by having a Root CA.

  - A root CA issues a special self-signed root certificate that does not need to be checked.

- For example:

$$\langle \mathscr{C}[23, \texttt{DCU CA}, \texttt{Geoff Hamilton}, (\texttt{Jan 2020}, \texttt{Dec 2021}), k^+_{GH}],$$
$$\mathscr{C}[3452, \texttt{VeriSign}, \texttt{DCU CA}, (\texttt{Jan 2015}, \texttt{Dec 2024}), k^+_{DCU}]$$
$$\mathscr{C}[2735, \texttt{VeriSign}, \texttt{VeriSign}, (\texttt{Jan 2010}, \texttt{Dec 2029}), k^+_{VS}]\rangle$$

The `X.509` PKI

- Consider a certificate path $\mathscr{P} = \langle \mathscr{C}_0, \cdots, \mathscr{C}_k \rangle$ where

$$\mathscr{C}_i = \mathscr{C}[N_i, I_i, S_i, (F_i, T_i), K_i]$$

- We can validate this path and extract the public key for S_0 using the following algorithm:

if $(now < F_k)$ **or** $(now > T_k)$ **then fail** ;
if $revoked(I_k, N_k)$ **then fail** ;
for $i := k-1$ **downto** 0 **do**
begin
  if $I_i \neq S_{i+1}$ **then fail** ;
  if $(now < F_i)$ **or** $(now > T_i)$ **then fail** ;
  if $revoked(I_i, N_i)$ **then fail** ;
  if not $validSig(\mathscr{C}_i, K_{i+1})$ **then fail** ;
end
return K_0 ;

The `X.509` PKI

- Root certificates are inherently insecure.
  - Anyone can simply generate a key-pair and issue a root certificate for any CA.
  - Therefore, they need to be protected and distributed in a secure manner.
- There are many `X.509` root CAs.
  - Different users of `X.509` have different CA Hierarchies, i.e., we have different `X.509` PKIs that use the same technology, but are managed and controlled by different organizations.
  - These hierarchies have different root certificates and have different policies.
- In theory, anyone can build a CA hierarchy and set themselves up as a CA.
  - However, in practice, a user will only be prepared to trust known CA hierarchies.

The `X.509` PKI

- Each `X.509` certificate contains a validity period.
  - Certificates should not be used before or after their validity period.
  - Once a certificate has expired, it is possible to issue a new certificate that binds the same public key to the subject.
- To handle revoked certificates, each CA maintains a Certificate Revocation List (CRL).
  - This is a list of the serial numbers of the certificates issued by the CA that have been revoked.
  - Only the serial numbers of revoked certificates that are otherwise within their validity period need to be held on a CRL.
  - When validating a certificate, a user should check that it does not appear on the issuer's CRL.
  - Typically, CRLs are updated periodically, e.g., every week.
- CRLs are probably the weakest aspect of `X.509`.
  - CRL can become extremely large.
  - An arguably better approach is to have servers which can be queried about the status of a certificate.

PGP PKI

- Pretty Good Privacy (PGP)
- Originally developed by Phil Zimmermann in 1991.
- For three years, Philip Zimmermann, was threatened with federal prosecution in the United States for his actions. Charges were finally dropped in January 1996.
- At the end of 1999, granted a full license by the U.S. Government to export PGP world-wide, ending a decade-old ban.
- Selected best available cryptographic algorithms and integrated into a single program.
- Works across a variety of platforms.
- Originally free; now also have commercial versions available but free for non-commercial use.

PGP PKI

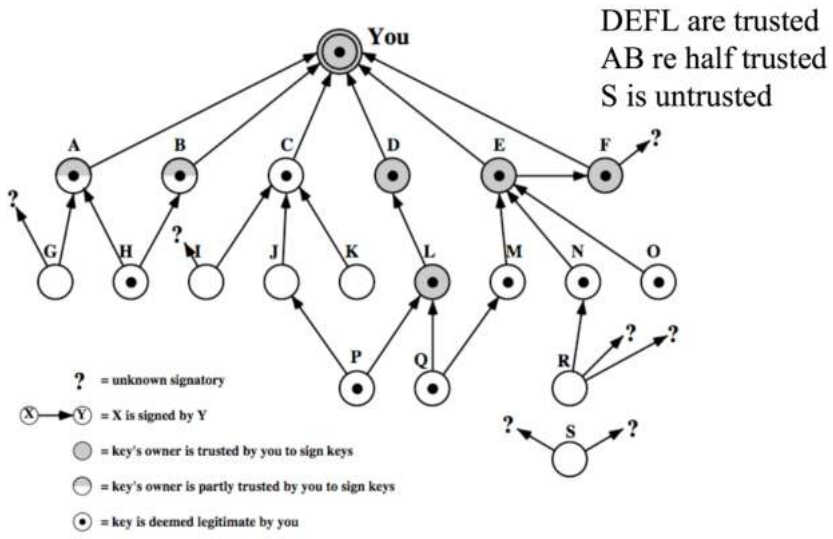
The drawback of the X.509 hierarchy is that every user must trust all the CAs.
In the PGP hierarchy every user is also a CA, and users can select which CAs they trust, and which they do not trust.

- As a CA, a user signs certificates to their recognised friends; this does not mean that the friend is trustworthy.
- Each user then asks for certificates from many other users, and collects as many as desired.
- When a user needs to prove their identity, they publish (or send) the certificates they have collected to the other user, and the other user verifies them.
- The receiver can decide to trust certificates signed by some CAs unconditionally, and not trust certificates signed by other CAs.

PGP PKI

- There is no need to buy certificates from companies
- A user can sign other user's certificates
- If you trust someone, you can trust users that they sign for
- You can assign a level of trust to each user and hence to the certificate they sign for
- For example:
  - A certificate that is signed by a fully trusted user is fully trusted
  - A certificate signed by two half trusted users is fully trusted
  - A certificate signed by one half trusted user is half trusted
  - Some certificates are untrusted.
- Owners can revoke public key by issuing a revocation certificate signed with the revoked private key
- New web-of-trust certificates have expiry dates

PGP PKI


DEFL are trusted
AB re half trusted
S is untrusted
You
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
? = unknown signatory
X Y = X is signed by Y
= key's owner is trusted by you to sign keys
= key's owner is partly trusted by you to sign keys
= key is deemed legitimate by you

7 Key Exchange and Authentication Protocols

7.1 Introduction

Introduction
In this section, we are going to explore protocols that solve two general problems:

- Entity Authentication
- Key-Exchange

An entity authentication protocol allows an entity to ensure that 1 or more other entities are, in fact, who they say they are.
A key exchange protocol allows 2 or more entities to agree on a common key or keys that can used for some communication.
We will describe security protocols in isolation, but in practice they will be part of larger communications protocols that achieve some useful function (e.g., share files).

Introduction
In a typical scenario we might have 2 entities who wish to use a communications protocol to perform some task.
This communications protocol would have a number of phases:

1. Authentication Phase during which each entity securely establishes the identity of the other entity.
2. Key-Exchange Phase during which the entities agree on a symmetric session key to be used to encrypt the data during the remainder of the communication.
3. Data Transfer Phase in which the application data is encrypted using the agreed key(s) and exchanged between the 2 entities.

Introduction
To define a security protocol we need:

1. A description of the objectives of the protocol, including:
    - The entities involved.
    - The types of cryptography to be used and the details of existing keys that entities may possess.
    - The security objectives to be achieved.
2. A rigorous description of the steps making up the protocol.
    - For each step we will define the entities involved and the data exchanged.
    - We will also describe what computations the entities need to perform.
3. An analysis of the security of the protocol, including:

- The general behaviour of the protocol.
  - Here we will assume perfect cryptography.
- The key usage properties.
- What type(s) of cryptographic attacks does the protocol permit.
  - We will not look at these attacks in any detail, but they may be important when we are picking an actual cipher.

7.2 Notation

Notation
When describing protocols, we use the following abstract notation to represent data in messages sent during protocol exchanges:

$A, B, \ldots$	The parties or entities taking part in a protocol.
$K_{A,B}$	Symmetric Key shared between A and B
$d_1, d_2, \ldots, d_n$	Concatenated Data
$\{d_1, d_2, \ldots, d_n\}_K$	Encrypted Data
$H(d_1, d_2, \ldots, d_n)$	Hashed Data
N_A	Nonce for entity A
T_A	Timestamp for entity A

Notation

P_A	Password for entity A
K_A^-	Private Key for entity A
K_A^+	Public Key for entity A
$\{\lvert d_1, d_2, \ldots, d_n \rvert\}_{K_A^-}$	Signature by entity A

Each step in a protocol is represented as a message sent from one entity to another:

$$N.\ A \rightarrow B : Message$$

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, N_B, \{N_A\}_{K_{A,B}}$

7.3 Confidentiality

Confidentiality
Before looking at authentication and key-exchange protocols, we will look at some simple protocols for the confidential transfer of data.
Protocol 1
Objectives:

- A (Alice) wishes to send a message M confidentially to B (Bob) using a symmetric cipher.
- We assume that A and B share a secret key $K_{A,B}$.

Protocol Steps:

1. $A \rightarrow B : \{M\}_{K_{A,B}}$

Confidentiality
General Analysis:

- In addition to A and B, we must consider other entities:
  - E (Eve) - a malicious active attacker who is both reading and modifying messages being transferred.
  - T (Trevor) - a trusted third party not directly involved in the exchange.
- The only way that E can read or modify messages is by knowing the key $K_{A,B}$.
- However, E can attack the protocol in a number of other ways:
  - She can delete messages - denial of service.
  - She can replay messages, i.e., she can capture a message and send it again.
  - If the plaintext of messages does not contain a certain amount of redundancy, she can send arbitrary data as a message.
- If B receives ciphertext $\{M\}_{K_{A,B}}$, and B did not produce it, then it must have been produced by A.

Confidentiality
Does this provide entity authentication?

- It does provide message origin authentication.
- It does not provide entity authentication as E can replay messages.

Does this provide message non-repudiation?

- There is no way that B can prove to T that a particular message came from A.
- B could always have constructed the message and encrypted it himself.

Confidentiality
Key Usage Analysis:

- We assume that Alice and Bob have a shared key $K_{A,B}$.
  - How exactly, do Bob and Alice agree on a shared key?
  - They need to use an out-of-band exchange or use some suitable key-exchange protocol.
- If the key $K_{A,B}$ is compromised then all messages encrypted with $K_{A,B}$ can be decrypted and messages can be falsified.
  - This includes messages that were sent in the past.
- Protocols that use symmetric keys normally require separate keys for each pair of users.
  - If there are N users, we need $\frac{N(N-1)}{2}$ keys.
  - As N grows, it becomes extremely difficult to manage all these keys.

Confidentiality
Cryptographic Analysis:

- This protocol allows a ciphertext only attack.
- However, if Eve can persuade Alice to reveal some messages and/or send particular messages, then various plaintext attacks may be possible.

Confidentiality
Protocol 2
Objectives:

- A (Alice) wishes to send a message M confidentially to B (Bob) using an asymmetric cipher.
- We assume that B's public key K_B^+ is available to A.

Protocol Steps:

1. $A \rightarrow B : \{M\}_{K_B^+}$

Confidentiality
General Analysis:

- The only way that Eve can read messages is by knowing Bob's private key K_B^-.
  - Even though Eve knows Bob's public key K_B^+, she still cannot read messages.

- As before, Eve can delete messages, replay messages etc..
- However, Eve can create new messages using Bob's public key K_B^+.
  - Therefore, we have no authentication properties as anyone can produce valid messages.
- It should also be noted that asymmetric cryptography is orders of magnitude slower that symmetric cryptography.

Confidentiality
Key Usage Analysis:

- Public keys are intended to be known by everyone.
- However, we still need to ensure that the correct public key is associated with an entity.
  - If we don't, E may be able to mount a man-in-the-middle attack.
  - We can avoid this by using a PKI.
- With this protocol each entity must have a key-pair, but since every other entity can use the same public key to communicate with (say) B, the key management problem is much simpler.
  - If there are N users we only need N public keys.

Cryptographic Analysis:

- Since we are using an asymmetric cipher, this protocol allows an adaptive chosen plaintext attack.

7.4 Entity Authentication (Identification)

Entity Authentication (Identification)
The problem is for one entity (the verifier) to be sure that another entity (the claimant) is who they say they are and hence prevent impersonation.
With Message Origin Authentication, a user can simply digitally sign data so that its origin can be determined, but there is no way of knowing when the data was signed.
Here, however, we are interested in real-time communications and ensuring that we have a valid channel to the other entity as we execute the protocol.

- In particular, we require the entity that is authenticated to be an active participant in the protocol as it is executed.

Entity Authentication (Identification)
In general, authentication is based on the claimant supplying information to the verifier which only the claimant knows or can compute.

- The obvious approaches would be for the claimant to use a password or to compute information using a cryptographic key.

One-way or unilateral authentication is when only one entity authenticates itself to the other entity.
With two-way or mutual authentication, each entity authenticates itself to the other entity.

Entity Authentication (Identification): Goals
Consider an honest entity A authenticating itself to B and a dishonest entity E trying to impersonate A:

1. A should be able to authenticate itself to B, i.e. B should accept A's claim to be A.
2. It should be computationally infeasible for E to have B accept its claim to be A, i.e. B should reject the claim and terminate the protocol.
3. No matter how many exchanges between A and B that E witnesses, it should remain computationally infeasible for E to impersonate A.
4. The information supplied to B by A should not allow B to impersonate A to a third party T, i.e. authentication should not be transferable.
5. Any authentication protocol needs to exhibit computational and communication efficiency.

Attacks on Authentication Protocols
The main purpose of attacking an authentication protocol is to allow a dishonest entity to impersonate a legitimate user.
We will consider a number of different kinds of attacks and see examples of such attacks on the protocols we describe.

1. Replay attacks in which information used in one instance of the protocol is captured and replayed at a later stage.
2. Reflection attacks in which an adversary sends information back to the originator of the information.
3. Interleaving attacks in which an adversary combines information from a number of interleaved instances of the protocol.
4. Chosen Plaintext cryptographic attacks in which an adversary can get one of the entities to encrypt arbitrary pieces of text. This gives a cryptanalyst more opportunity to discover a key.
5. Man-in-the-Middle attacks in which an adversary sits and (potentially) manipulates all communications.

Attacks on Authentication Protocols
There are a number of other issues that we need to consider.

1. Typically, authentication occurs as the first steps of a larger protocol and is only valid at a given instant of time. If ongoing assurance of identity is required, then additional techniques (e.g. encrypting all subsequent data with a session key) are required.
2. Since entity authentication takes place in real-time, the amount of computing power available to an attacker is more limited than with an off-line attack. However, an adversary might be able to gather information during a successful authentication and compute information off-line that can be used later.
3. An attacker may be able to mount a denial of service attack by simply delaying protocol messages.

Weak Authentication using Passwords
The simplest way to implement authentication is to use passwords, i.e. knowledge of the password is proof of identity.
A claimant simply supplies their identity and password, and the verifier compares this with a stored copy of the password to authenticate the claimant.
Protocol 3
Objectives:

- A wishes to authenticate herself to B using a shared password P_A.

Protocol Steps:

1. $A \rightarrow B : A, P_A$

Weak Authentication using Passwords
Analysis: This protocol is extremely weak.

1. The password is passed as plaintext, therefore, unless we use a secure channel, anyone can observe the password and capture it for future use.
2. Therefore, this protocol is subject to replay attacks.
3. B must maintain a table of passwords and these need to be protected from disclosure. This is a problem with all password-based schemes.

Weak Authentication using Passwords
There are situations where simple passwords are widely used:

1. For logging onto computers.
2. For logging onto websites; typically using a confidential SSL connection.

Both of these uses are acceptable, provided some care is taken.

1. Passwords need to be chosen so as to avoid dictionary attacks.
2. The computer or website should only permit a small number of logon attempts or take progressively longer to respond to logon attempts.
3. Users should change their passwords frequently.
4. Users should not use the same password with different computers and websites.
    - If a user uses the same password with a number of computers or sites, then a dishonest administrator could impersonate the user.

Weak Authentication using Passwords
Typically, passwords are used as one-way authentication mechanisms.

- A user (client) authenticates itself to a computer (server) and they are then granted access rights on that computer (server).

In theory, we could extend the use of passwords to supply two-way authentication.

1. $A \rightarrow B : A, P_A$
2. $B \rightarrow A : B, P_B$

In situations where passwords are used, it is normally too difficult for both parties to maintain lists of passwords.

Weak Authentication using Passwords
Instead of exchanging a password as plaintext, it is possible for the claimant to send a digest of the password.
The verifier can then compute its own copy of the digest and compare it to the value sent from the claimant.
Protocol 4
Objectives:

- A wishes to authenticate herself to B using a shared password P_A.

Protocol Steps:

1. $A \rightarrow B : A, H(P_A)$

Of course, this is no more secure as an adversary could copy $H(P_A)$ and mount a replay attack.

Nonces and Timestamps
A nonce is a value that is used only once.
The main use of a nonce is to distinguish one instance of a protocol message from another.
In particular, if a message does not naturally contain information that varies from instance to instance, then a nonce can be added as an extra field.
The value of a nonce does not matter; it only needs to be different from instance to instance.
However, we may need some way of testing that a nonce has not been used before. In particular, when we use a nonce to prevent a replay attack, we need some way of knowing that we have handled a message with a particular nonce value already.

Nonces and Timestamps
There are a number of different ways of generating nonces:

1. Picking a random number.
   - If we pick random numbers from a sufficiently large space (e.g. $0..2^{64}$), then the chances of using the same value twice are very small.
   - However, if we need to detect reuse, then all previous nonces would need to be stored.
2. Using a monotonically increasing sequence number, e.g. $0, 1, 2, \ldots$.
   - Monotonically increasing ensures that each new nonce is greater than the previous nonce.
   - To detect reuse, we only need to remember the last sequence number used.
3. Using the time of day to a suitable accuracy.
   - This requires the entities to have synchronized clocks.
   - Times are monotonically increasing and detecting replays is relatively easy.

Nonces and Timestamps
In addition to nonces, we sometimes use timestamps.
Unlike a nonce (and indeed time used as a nonce), timestamps convey information about when an event occurred, so the value is important.
Protocol 5: `X.509` Protected Password Authentication
Objectives:

- A wishes to authenticate herself to B using a shared password P_A.

Protocol Steps:

1. $A \rightarrow B : A, T, N, H(A, T, N, P_A)$

Nonces and Timestamps
Analysis:

- Given this exchange, B computes the digest $H(A,T,N,P_A)$ using its copy of P_A and compares it with the value sent in the protocol message. In particular, B uses the values T and N sent in the protocol message when computing its copy of the digest.
- This protocol uses both a timestamp T and nonce N. Using only a timestamp requires accurately synchronized clocks and using only nonces may require B to store large numbers of old nonces.
- A must ensure that the timestamp T and nonce N are chosen in such a way as to avoid the possibility of a replay.
- An adversary capturing the digest $H(A,T,N,P_A)$ is unable to replay the message as B will detect the duplicated nonce or a timestamp that is too old.
- This protocol can easily be extended to support two-way authentication.

Authentication Using Cryptography
In the previous protocols a claimant's ability to produce a password was used to authenticate the claimant.
An alternative approach is to have the claimant demonstrate that they have access to a cryptographic key. This is achieved by requiring the claimant to compute some value using the key.
This can be done using a challenge-response protocol, which has the following structure.
Objectives:

- A wishes to authenticate B.

Protocol Steps:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, \{N_A\}_K$

Authentication Using Cryptography
Analysis:

- A sends a challenge (nonce) N_A to B and B returns $\{N_A\}_K$, i.e. the challenge encrypted with some key K.
  - Provided that B is the only user with access to K, then A knows that B is participating in the protocol.
- Is there any need for A and/or B to record nonces so as to prevent a replay attack?
- How can the nonce be generated?

- Is this protocol subject to a man-in-the-middle attack?
- This basic challenge-response protocol can be extended to implement two-way authentication.

Authentication Using Symmetric Ciphers
Protocol 6
Objectives:

- A wishes to authenticate B using a shared secret key $K_{A,B}$.

Protocol Steps:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, \{N_A\}_{K_{A,B}}$

Authentication Using Symmetric Ciphers
Analysis:

- Since A shares a key with B, she can decrypt the ciphertext received from B and ensure it is valid.
- Like password-based schemes, A may need to maintain a database of shared keys. As with passwords, this is vulnerable to disclosure.
- Again, like password-based schemes, if B was foolish enough to use the same shared key with a number of entities, then A could impersonate B to other entities.

Authentication Using Symmetric Ciphers
Cryptographic Analysis:

- This protocol exposes the cipher to an adaptive chosen plaintext attack.
- Adaptive chosen plaintext attacks are a problem for challenge-response protocols, i.e. the verifier can pick any nonce N that they want.
  - An attacker could repeatedly perform steps 1 and 2 of the last protocol above.
  - Each time they would pick a value for N that would assist in attacking the shared key.

Authentication Using Symmetric Ciphers
There are two ways of reducing this vulnerability:

- B can add additional information to the data to be encrypted. This means that A has only partial control over what data is encrypted.
  1. $A \rightarrow B : A, N_A$

2. $B \rightarrow A : B, \{N_A, N_B\}_{K_{A,B}}$

- B can select the nonce. For example, the protocol presented earlier for a protected password can be extended to use symmetric keys.

1. $B \rightarrow A : B, N_B, \{B, N_B\}_{K_{A,B}}$

Authentication Using Symmetric Ciphers
We can extend this protocol to allow mutual authentication.
Protocol 7
Objectives:

- Entities A and B wish to mutually authenticate each other using a shared secret key $K_{A,B}$.

Protocol Steps:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, N_B, \{N_A\}_{K_{A,B}}$
3. $A \rightarrow B : \{N_B\}_{K_{A,B}}$

Authentication Using Symmetric Ciphers
Analysis:

- Unfortunately, this protocol is subject to a reflection attack.
- Consider, a dishonest entity E who wishes to impersonate A to B.
  - First E starts one instance of the protocol with B.
    1. $E(A) \rightarrow B : A, N_A$
    2. $B \rightarrow E(A) : B, N_B, \{N_A\}_{K_{A,B}}$
  - At this stage, E is stuck as she cannot encrypt N_B.
  - However, E can start a second instance of the protocol and get B to encrypt N_B.
    3. $E(A) \rightarrow B : A, N_B$
    4. $B \rightarrow E(A) : B, N'_B, \{N_B\}_{K_{A,B}}$
  - E abandons the second instance and resumes the first instance.
    5. $E(A) \rightarrow B : \{N_B\}_{K_{A,B}}$
- It is important to note that this is an attack on the protocol and not the cryptography.

Authentication Using Symmetric Ciphers
There are a number of ways of preventing this attack.

- A and B could use different keys.
  - Therefore, a nonce encrypted by B could not be sent back to B as it would be encrypted with the wrong key.
- The initiator could prove its identity first:
  1. $A \rightarrow B : A$
  2. $B \rightarrow A : B, N_B$
  3. $A \rightarrow B : N_A, \{N_B\}_{K_{A,B}}$
  4. $B \rightarrow A : \{N_A\}_{K_{A,B}}$

  In this case B does not encrypt anything until it has authenticated A.

Authentication Using Symmetric Ciphers

- The nature of the challenges produced by A and B could be different, i.e. a challenge encrypted by B could not be confused as a challenge to be encrypted by A.
- For example, this can be achieved by including different information in the ciphertext sent between the parties:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, N_B, \{N_A, A\}_{K_{A,B}}$
3. $A \rightarrow B : \{N_B\}_{K_{A,B}}$

Authentication Using Asymmetric Ciphers
A challenge-response protocol can also be used with asymmetric ciphers. There are two basic modes of operation:

- The claimant B encrypts the challenge with their private key.
- The verifier A encrypts a challenge using the claimant's public key K_B^+ and the claimant proves that they have the private key by decrypting the challenge and returning it to the verifier.

Protocol 8
Objectives:

- B wishes to authenticate himself to A using his private key K_B^-.

Protocol Steps:

1. $A \rightarrow B : A, N_A$
2. $B \rightarrow A : B, \{N_A\}_{K_B^-}$

Authentication Using Asymmetric Ciphers
Analysis:

- Since A knows B's public key K_B^+, she can decrypt the ciphertext received from B and ensure it is valid.
- If we use a PKI, then key management becomes relatively simple, i.e. A just needs to obtain a certified copy of B's public key from the PKI.
- If required, certificates can be exchanged as part of the protocol.
- It is unwise to use key-pairs for more than one purpose.
  - For this protocol, if A challenges B with a value $H(m)$, then B will encrypt it with his private key and hence produce a digital signature for m.
  - Thus, if the same key-pair is used for producing digital signatures, B has unwittingly signed some unknown document m.

Authentication Using Asymmetric Ciphers
Cryptographic Analysis:

- This protocol would allow an attacker E to mount an adaptive chosen plaintext attack against encryption with a private key.
- We can use some of the techniques as described for symmetric ciphers to prevent such attacks.

Authentication Using Asymmetric Ciphers
Protocol 9
Objectives:

- B wishes to authenticate himself to A using his public key K_B^+.

Protocol Steps:

1. $A \rightarrow B : A, \{N_A\}_{K_B^+}$
2. $B \rightarrow A : N_A$

Authentication Using Asymmetric Ciphers
Analysis:

- A encrypts the nonce N_A with B's public key. Therefore if N_A is returned in step 2, A knows that B must have participated in the protocol.

Cryptographic Analysis:

- This protocol would allow an attacker E to mount an adaptive chosen ciphertext attack against encryption with a public key.

Needham-Schroeder Public-Key Protocol
There are two protocols due to Needham-Schroeder; this one is based on the use of asymmetric encryption, the other one (which we will look at later) is based on symmetric encryption.
Protocol 10: Needham-Schroeder Public-Key (NSPK)
Objectives:

- Entities A and B wish to mutually authenticate each other using their public keys K_A^+ and K_B^+.

Protocol Steps:

1. $A \rightarrow B : \{N_A, A\}_{K_B^+}$
2. $B \rightarrow A : \{N_A, N_B\}_{K_A^+}$
3. $A \rightarrow B : \{N_B\}_{K_B^+}$

Needham-Schroeder Public-Key Protocol
Analysis:

- Since the ciphertext sent in step 1 is encrypted by B's public key, only B can discover N_A.
- Therefore, if A finds N_A in the message sent in step 2, she must be communicating with B.
- Similarly, only A can discover N_B and return it in step 3.
- If necessary, this protocol can be extended so that A and B exchange certificates.

Needham-Schroeder Public-Key Protocol
Cryptographic Analysis:

- In 1995, Gavin Lowe discovered an interleaving attack on the NSPK Protocol by using a model checking tool called FDR.
- If A and B can be trusted, this protocol appears to be secure.
- However, E can interleave two runs (instances) of this protocol and transfer authentication, i.e. she can impersonate A to B.

1.	$A \rightarrow E :$	$\{N_A, A\}_{K_E^+}$			
			$1'.$	$E(A) \rightarrow B :$	$\{N_A, A\}_{K_B^+}$
			$2'.$	$B \rightarrow E(A) :$	$\{N_A, N_B\}_{K_A^+}$
2.	$E \rightarrow A :$	$\{N_A, N_B\}_{K_A^+}$			
3.	$A \rightarrow E :$	$\{N_B\}_{K_E^+}$			
			$3'.$	$E(A) \rightarrow B :$	$\{N_B\}_{K_B^+}$

Needham-Schroeder Public-Key Protocol

- So at this point, A believes that she is communicating with E and that only she and E share the nonces N_A and N_B.
- In fact, she is communicating with E, but the nonces N_A and N_B are also known to B.
- B believes he is communicating with A, but he is in fact communicating with E.
- Note that this attack was achieved without attacking the cryptography, it is achieved by simply manipulating the protocol.

Needham-Schroeder Public-Key Protocol
This protocol is fixed by Lowe to prevent this attack as follows:

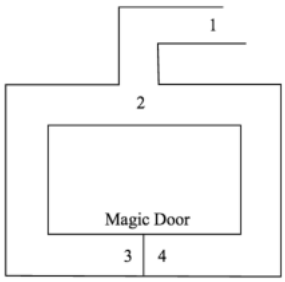
1. $A \rightarrow B : \{N_A, A\}_{K_B^+}$
2. $B \rightarrow A : \{N_A, N_B, B\}_{K_A^+}$
3. $A \rightarrow B : \{N_B\}_{K_B^+}$

- The message modified by this new version of the protocol is the only one which cannot be decrypted by E in the previous attack.
- So E will not be able to change B to E.
- A will now be able to see that the received message does not come from E but from B, so will not send N_B to E.
- Thus, E is unable to discover the value of N_B.

Zero Knowledge Proof (ZKP)

- Alice wants to prove that she knows a secret without revealing any information about it
- Bob must verify that Alice knows secret
  - But, Bob gains no information about the secret
- Process is probabilistic
  - Bob can verify that Alice knows the secret to an arbitrarily high probability
- An interactive proof system

Zero Knowledge Proof (ZKP)
Bob's Cave: at the bottom of Bob's cave, there is a magic door that can only be opened using a secret password.



Can Alice convince Bob that she knows the secret without revealing the password?

Zero Knowledge Proof (ZKP)

1. Bob stands at Point 1.
2. Alice enters the cave and stands either at Point 3 or 4
3. After Alice disappears, Bob walks to Point 2.
4. Bob calls to Alice, asking her to come out either the left or the right passage.
5. Alice complies, using the secret password if necessary.
6. Alice and Bob repeat Steps 1-5 k times.

- If Alice does not know the secret, then she could come out from the correct side with probability $1/2$.
- If this is repeated k times, then Alice can only fool Bob with probability $1/2^k$.

Fiat-Shamir Protocol

- Cave-based protocols are inconvenient
  - Can we achieve the same effect without the cave?
- Finding square roots modulo composite n is difficult
  - Equivalent to factoring
- Suppose $n = pq$, where p and q are prime
- Alice has a secret s
- n and $v = s^2 \pmod{n}$ are public, s is secret
- Alice must convince Bob that she knows s without revealing any information about s

Fiat-Shamir Protocol
Protocol 11: Fiat-Shamir
Objectives:

- A wishes to prove that she knows secret s without revealing any information about s to B.

Protocol Steps:

1. $A \rightarrow B : x = r^2 \pmod{n}$
2. $B \rightarrow A : e \in \{0,1\}$
3. $A \rightarrow B : y = r \cdot s^e \pmod{n}$

- Alice commits to random r by sending $x = r^2 \pmod{n}$ to Bob.
- Bob sends challenge $e \in \{0,1\}$ to Alice.
- Alice responds with $y = r \cdot s^e \pmod{n}$.
- Bob verifies that $y^2 = x \cdot v^e \pmod{n}$.
  - This will be the case since $y^2 = r^2 \cdot s^{2e} = r^2 \cdot (s^2)^e = x \cdot v^e \pmod{n}$.

Fiat-Shamir Protocol
If $e = 0$:

1. $A \rightarrow B : x = r^2 \pmod{n}$
2. $B \rightarrow A : e = 0$
3. $A \rightarrow B : y = r \pmod{n}$

- Public: modulus n and $v = s^2 \pmod{n}$
- Alice selects random r, Bob chooses $e = 0$
- Bob must check whether $y^2 = x \pmod{n}$
- Alice does not need to know s in this case

Fiat-Shamir Protocol
If $e = 1$:

1. $A \rightarrow B : x = r^2 \pmod{n}$
2. $B \rightarrow A : e = 1$
3. $A \rightarrow B : y = r \cdot s \pmod{n}$

- Public: modulus n and $v = s^2 \pmod{n}$

- Alice selects random r, Bob chooses $e = 1$
- If $y^2 = x \cdot v \pmod{n}$ then Bob accepts it
  - So Alice passes this iteration of the protocol
- Note that Alice must know s in this case

Fiat-Shamir Protocol

- Can Eve convince Bob she is Alice?
  - If Eve expects $e = 0$, she sends $x = r^2 \pmod{n}$ in message 1 and $y = r \pmod{n}$ in message 3 (i.e., follow the protocol)
  - If Eve expects $e = 1$, she sends $x = r^2 \cdot v^{-1} \pmod{n}$ in message 1 and $y = r \pmod{n}$ in message 3
- If Bob chooses $e \in \{0, 1\}$ at random, Eve can only trick Bob with probability 1/2
  - After k iterations, the probability that Eve can convince Bob that she is Alice is only $1/2^k$
  - Just like Bob's cave
- Bob's $e \in \{0, 1\}$ must be unpredictable
- Alice must use new r on each iteration, or else:
  - If $e = 0$, Alice sends $r \pmod{n}$ in message 3
  - If $e = 1$, Alice sends $r \cdot s \pmod{n}$ in message 3
  - Anyone can find s given $r \pmod{n}$ and $r \cdot s \pmod{n}$

Zero Knowledge Proof (ZKP)

- Zero knowledge means that nobody learns anything about the secret s
  - Public: $v = s^2 \pmod{n}$
  - Eve sees $r^2 \pmod{n}$ in message 1
  - Eve sees $r \cdot s \pmod{n}$ in message 3 (if $e = 1$)
- If Eve can find r from $r^2 \pmod{n}$, then she can get s
  - But that requires modular square root
  - If Eve could find modular square roots, she could get s from public v
- Protocol does not seem to help to find s

Zero Knowledge Proof (ZKP)

- Public key certificates identify users
  - No anonymity if certificates sent in plaintext
- ZKP offers a way to authenticate without revealing identities
- ZKP supported in Microsoft's Next Generation Secure Computing Base (NGSCB)
  - ZKP used to authenticate software without revealing machine identifying data
- ZKP is not just pointless mathematics!

7.5 Key-Exchange

Key-Exchange
The Problem:

- Symmetric cryptography is much more efficient than asymmetric cryptography.
  - Therefore, when exchanging large amounts of data we prefer to use symmetric cryptography to provide confidentiality.
- Handling large numbers of symmetric keys is problematic.
- Therefore, we would like some way of agreeing or exchanging session keys that we can use for a limited period of time and then destroy.
  - With real-time protocols we will not even want to store the session keys.
- Clearly we will want to protect the key agreement or exchange so that an observer cannot determine the session keys.

Key-Exchange
Protocol 12: Diffie-Hellman Key Exchange
Objectives:

- A and B wish to agree a symmetric key $K_{A,B}$.

Public values: Large prime p and generator g.
Private values:
A has private value x such that $1 < x < p-1$.
B has private value y such that $1 < y < p-1$.
Protocol Steps:

1. $A \rightarrow B : g^x \pmod{p}$
2. $B \rightarrow A : g^y \pmod{p}$

A now computes $K_{\langle A,B \rangle}$ as $(g^y)^x$ and B computes $K_{\langle A,B \rangle}$ as $(g^x)^y$.

Perfect Forward Secrecy
Consider the following scenario:

- A encrypts message with key $K_{A,B}$ and sends ciphertext to B
- E records ciphertext and later attacks A's (or B's) computer to recover $K_{A,B}$
- Then E decrypts recorded messages

Perfect forward secrecy (PFS): E cannot later decrypt recorded ciphertext

- Even if E gets key $K_{A,B}$ or other secret(s)
- For PFS, A and B cannot use $K_{A,B}$ to encrypt
- Instead they must use a session key and forget it after it is used
- Can A and B agree on a session key in a way that ensures PFS?

Ephemeral Diffie-Hellman: A forgets x and B forgets y so neither can later recover $K_{A,B}$

Key Agreement

- The Diffie-Hellman key exchange protocol is, however, susceptible to a man-in-the-middle attack.
- The attacker E can insert herself in the middle of the communication between A and B, and masquerade as A to B and as B to A:

$$\begin{array}{llll} 1. & A \to E\text{: } g^x \pmod p & 1'. & E \to B\text{: } g^z \pmod p \\ 2. & E \to A\text{: } g^z \pmod p & 2'. & B \to E\text{: } g^y \pmod p \end{array}$$

- A and E therefore share the secret value g^{xz} and B and E share the secret value g^{yz}.
- A and B do not know that E exists.
- However, they will both communicate with her assuming she is the other party.
- A mechanism to ensure that both parties have authenticated each other before performing the key exchange is therefore required.

User-Generated Keys
An alternative to key agreement, is for one entity A to generate a session key K, which is then transferred confidentially to another entity B.

- Of course, for this to work there must be existing keys that can be used to encrypt this session key.

With an asymmetric cipher A can simply encrypt K with B's public key.

1. $A \to B : \{K\}_{K_B^+}$

User-Generated Keys
With symmetric ciphers the situation is somewhat more complex.

- In particular, each entity will need existing symmetric keys and we have a potential key management problem.
- As we will see, this problem can be overcome by using a trusted server.

Allowing one entity to generate the session key is a potential weakness.

- The entity may use an existing key or generate a weak key.
- With the symmetric protocols we will consider, this problem is overcome by having the session key generated by a trusted server.

7.6 Authentication and Key-Exchange

Authentication and Key-Exchange
Earlier we noted that an authentication protocol gave assurance of identity at a single instant in time.
For a communications protocol we would like some sort of ongoing authentication that lasts for the lifetime of a connection.
An obvious approach is to combine authentication and key exchange into a single protocol.

- Not only are the entities authenticated, but the session key is bound to the authentication process.
- Then, provided future exchanges are encrypted with the session key, we have ongoing authentication.

Symmetric Ciphers
Earlier we looked at how two entities A and B could perform mutual authentication using a shared secret key.
This approach requires each distinct pair of users to share a separate key and with a large community of users this becomes unmanageable. An alternative approach is to use a trusted server T to authenticate users and issue session keys. With this approach:

- Each user need only maintain a single shared key with the server.
- The server generates the session key and is trusted to behave properly.

There a number of protocols that are based on this basic structure, we will consider the Needham-Schroeder Secret-Key Protocol and Otway-Rees.
We will also look briefly at Kerberos, a complete system based on the Needham-Schroeder Secret-Key Protocol.

Needham-Schroeder Secret-Key Protocol
Let us assume that:

- We have a trusted server T.
- Two users A and B who share secret keys $K_{A,T}$ and $K_{B,T}$ respectively with T.

Protocol 13: Needham-Schroeder Secret-Key (NSSK)
Objectives:

1. To allow A and B to be authenticated.
2. To allow T to generate a session key $K_{A,B}$ that is bound to the authentication process and is only known to A, B and T.

Needham-Schroeder Secret-Key Protocol
Protocol Steps:

1. $A \rightarrow T : A, B, N_A$
2. $T \rightarrow A : \{N_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,T}}\}_{K_{A,T}}$
3. $A \rightarrow B : \{K_{A,B}, A\}_{K_{B,T}}$
4. $B \rightarrow A : \{N_B\}_{K_{A,B}}$
5. $A \rightarrow B : \{N_B - 1\}_{K_{A,B}}$

Needham-Schroeder Secret-Key Protocol

1. A sends her identity, the identity B of the entity she wishes to communicate with and a nonce N_A to the server T.
2. T performs the following steps:
   - He generates a session key $K_{A,B}$ for use by A and B.
   - He constructs B's token $\{K_{A,B}, A\}_{K_{B,T}}$ that only B can decrypt.
   - He constructs A's token $\{N_A, B, K_{A,B}, \{K_{A,B}, A\}_{K_{B,T}}\}_{K_{A,T}}$ that contains B's token and that only A can decrypt.
   - He sends A's token to A.
3. A performs the following steps:
   - She decrypts her token received from T using the key $K_{A,T}$ that she shares with T.
   - She checks the nonce N_A and name B in the token are the same as those sent to T in step 1.
   - She records the session key $K_{A,B}$ for future use.
   - She relays B's token to B.

Needham-Schroeder Secret-Key Protocol

4. B performs the following steps:
   - He decrypts his token received from T via A using the key $K_{B,T}$ that he shares with T.
   - He records the session key $K_{A,B}$ for future use.
   - He challenges A by sending a nonce N_B encrypted using the session key $K_{A,B}$.
5. A performs the following steps:
   - She decrypts the message sent from B using the session key $K_{A,B}$ and checks that the result is the nonce N_B. Note that this requires N_B to be redundant.
   - She then encrypts the value $N_B - 1$ using the session key and sends it to B.

Finally, B decrypts the message sent by A in step 5 and checks that the result is $N_B - 1$.

Needham-Schroeder Secret-Key Protocol
Analysis:

- This protocol provides mutual authentication of A and B, and allows a session key generated by T to be shared by A and B.
- The server T must be trusted; he has access to all keys.
- In step 1 we use a nonce N_A to prevent a replay attack i.e. A checks that N_A appears in the token returned by T in step 2.
- In step 1, A identifies herself and the entity B she wishes to communicate with. Since this message is not protected in any way, an attacker E could substitute her own identity for B. Provided the attacker E could also intercept messages 3-5, she could impersonate B to A.
  - This attack is prevented by T including the identity of the other party in A's token i.e. A checks that T has supplied information that allows A to authenticate B.
- After step 2, A knows that T has participated in this run of the protocol and that $K_{A,B}$ is fresh.

Needham-Schroeder Secret-Key Protocol

- After step 3, A and B have a shared session key $K_{A,B}$. Clearly, T knows this key, but because it was protected using $K_{A,T}$ and $K_{B,T}$, no other entity knows the key. We trust T not to misbehave.
- After step 3, B knows that $K_{A,B}$ is a key shared by A and himself. However, B does not know if $K_{A,B}$ is fresh.

- In steps 4 and 5, A and B perform mutual authentication i.e. they prove to each other that they have the session key and that they are participating in this run of the protocol.
  - The nonce N_B needs to be redundant so an attacker cannot replace the message in step 4 with some arbitrary ciphertext.
  - In step 5, A encrypts $N_B - 1$ since the encryption of N_B is already available.
- Note that A can repeat steps 3-5 as often as she wants i.e. she can use the same session key for multiple communications.

Needham-Schroeder Secret-Key Protocol
The Needham-Schroeder Secret Key Protocol was first published in 1978.
A significant feature is that it does not use timestamps and it is the basis for many of the server-based authentication and key distribution protocols developed after 1978.
It has been shown that the double encryption of B's token in step 2 is unnecessary. However, this is not a big issue.
However, serious weaknesses with the freshness of keys have been uncovered.

Needham-Schroeder Secret-Key Protocol
Attack 1: Compromised Session Keys

- Assume that an adversary E observes a legitimate protocol exchange between A and B.
- E can capture B's encrypted token in step 3 of the protocol.
- If E can compromise the session key $K_{A,B}$, then she can follow steps 3-5 of the protocol and impersonate A to B.
  - Note that E can take as long as she wants to compromise the session key.
  - So even though determining the session key may be computationally hard, the design of this protocol gives an adversary an unnecessarily long time to attempt to find the key.

Needham-Schroeder Secret-Key Protocol
Attack 2: Compromised Server/User Keys

- If the key $K_{A,T}$ that a user A shares with the server T is compromised, then an adversary E can use steps 1-2 of the protocol to obtain session keys and tokens for as many other users as she wishes.
- E can then impersonate A to one of these users at any time by completing steps 3-5 of the protocol.
- Even if A determines that the key $K_{A,T}$ has been compromised and generates a new key, E can still use the session keys and tokens that she obtained earlier.

Needham-Schroeder Secret-Key Protocol

- The problem with this protocol is that B has no way of knowing if the session key is fresh.
- There are two ways of resolving this problem:
  1. Have B interact with the server T so that he knows that the server is participating in the protocol.
     - In the existing protocol, steps 1-2 effectively authenticate T to A since T is encrypting fresh data with the shared secret key $K_{A,T}$.
     - This was done in an expanded version of the Needham-Schroeder protocol. However, we will look at an improved protocol due to Otway and Rees.
  2. Include timestamps so that B can ascertain the freshness of the session key.
     - Kerberos does this by adding a validity period or lifetime for each session key.

The Otway-Rees Protocol
Otway-Rees gives freshness guarantees without using synchronised clocks.
Again, let us assume that:

- We have a trusted server T.
- Two users A and B who share secret keys $K_{A,T}$ and $K_{B,T}$ respectively with T.

Protocol 14: Otway-Rees
Objectives:

1. To allow A and B to be authenticated.
2. To allow T to generate a session key $K_{A,B}$ that is bound to the authentication process and is only known to A, B and T.

The Otway-Rees Protocol
Protocol Steps:

1. $A \rightarrow B : N, A, B, \{N_A, N, A, B\}_{K_{A,T}}$
2. $B \rightarrow T : N, A, B, \{N_A, N, A, B\}_{K_{A,T}}, \{N_B, N, A, B\}_{K_{B,T}}$
3. $T \rightarrow B : N, \{N_A, K_{A,B}\}_{K_{A,T}}, \{N_B, K_{A,B}\}_{K_{B,T}}$
4. $B \rightarrow A : N, \{N_A, K_{A,B}\}_{K_{A,T}}$

Otway-Rees Protocol
Analysis:

- In this protocol both A and B perform mutual authentication with the server T and T returns the session key $K_{A,B}$ to each.
- In the case of A we have:
  - T can authenticate A by decrypting $\{N_A, N, A, B\}_{K_{A,T}}$ he receives in step 2.
  - A can authenticate T and obtain her copy of the session key by decrypting $\{N_A, K_{A,B}\}_{K_{A,T}}$ in step 4.
  - How does T ensure that the message from step 1 is not a replay?
    - Actually he can't!
    - However, this is okay, because an attacker replaying the message will be unable to recover the session key $K_{A,B}$ in step 4.
  - In this protocol, B relays messages to/from A. However, since these messages are encrypted using $K_{A,T}$, B cannot decrypt or modify them.
- In the case of B we have a similar argument with $\{N_B, N, A, B\}_{K_{B,T}}$ and $\{N_B, K_{A,B}\}_{K_{B,T}}$

Otway-Rees Protocol

- In step 1, A generates two nonces:
  - N_A which is used to prevent replay of T's message back to A and guarantee the freshness of the session key received from T (the nonce N_B serves the same purpose for B).
  - N which acts as a transaction identifier that binds the information sent by A and B together:
    - T can check that $\{N_A, N, A, B\}_{K_{A,T}}$ and $\{N_B, N, A, B\}_{K_{B,T}}$ have the same value for N and therefore belong to the same transaction.
  - After the protocol has completed, A and B will share the key $K_{A,B}$.
    - If A and B use $K_{A,B}$ to encrypt a communication, then they both have ongoing mutual authentication.

Otway-Rees Protocol
The Otway-Rees protocol is subject to a type flaw attack.
Attack:

1. $A \rightarrow E(B): \quad N, A, B, \{N_A, N, A, B\}_{K_{A,T}}$
4. $E(B) \rightarrow A: \quad N, \{N_A, N, A, B\}_{K_{A,T}}$

This works if A can be fooled into believing that the triple N, A, B is a key.

Kerberos
Kerberos is a complete server-based authentication and key distribution system based on the Needham-Schroeder protocol. The main differences lie in the use of timestamps and lifetimes.
It was developed as part of project Athena at MIT and has been widely used in Unix networks.
It has been incorporated into Windows.
As with Needham-Schroeder, entities share secret keys with a trusted Kerberos server.

Kerberos
Kerberos uses its own terminology:

- The trusted server T is known as a Key Distribution Centre (KDC).
- When a client A wishes to use a computing service B, the KDC issues a ticket that can be used to access the services of B.
  - As we will see, this is essentially B's token in the original Needham-Schroeder protocol.
- Tickets have a specific lifetime, but they can be renewed, post-dated, etc.

A significant use of Kerberos is to allow human users to logon to computer networks and access computing services distributed throughout that network.
Kerberos also defines the precise syntax of the various protocols, but since we are only interested in an abstract view of Kerberos, we will ignore such detail.

Kerberos
The following is a much simplified view of the basic Kerberos protocol.
Here A is a client, B is a computer service and S is the Kerberos Server.
The client A wishes to use the service B. It does so by getting a ticket from the Kerberos server S. This ticket is then presented to the service B.
Protocol 15: Kerberos
Protocol Objectives:

1. To allow A and B to be authenticated.
2. To allow T to generate a session key $K_{\langle A,B\rangle}$ that is bound to the authentication process and is only known to A, B and T.

Kerberos
Protocol Steps:

1. $A \rightarrow T : A, B, N_A$
2. $T \rightarrow A : \{K_{A,B}, A, L\}_{K_{B,T}}, \{K_{A,B}, N_A, L, B\}_{K_{A,T}}$
3. $A \rightarrow B : \{K_{A,B}, A, L\}_{K_{B,T}}, \{A, T_A\}_{K_{A,B}}$

4. $B \rightarrow A : \{T_A\}_{K_{A,B}}$

- L is the validity period (lifetime) of the key $K_{A,B}$
- A's ticket is $\{K_{<A,B>}, A, L\}_{K_{<B,T>}}$

Kerberos
Analysis:

1. This step is the same as Needham-Schroeder.
    - As with Needham-Schroeder, T generates a session key $K_{A,B}$ for use by A and B.
    - He encrypts a copy of this key for A using their shared key.
    - He also encrypts a copy for B and this produces a so-called ticket. This is essentially the token used in Needham-Schroeder.
    - Note that the shared key has a lifetime L which is included in the encrypted information sent to both A and B.

Kerberos

2. This step is similar to Needham-Schroeder except that the ticket is conveyed separately and not as part of the ciphertext sent to A. A performs the following steps:
    - She decrypts her token using the key $K_{A,T}$.
    - She checks that the nonce N_A and name B in the token are the same as those sent to T in step 1.
    - She records the session key $K_{A,B}$ for future use.
    - She records the ticket so that she can send it to B whenever she wants to use the service offered by B.
    - A can use this ticket multiple times, but only during its lifetime L.
3. A then presents her ticket and an authenticator $\{A, T_A\}_{K_{<A,B>}}$ to B.
    - Since the ticket is encrypted by $K_{B,T}$, B can decrypt it, recover the session key $K_{A,B}$, the lifetime L during which the session key can be used and the identity A of the owner of the ticket.
    - The authenticator $\{A, T_A\}_{K_{<A,B>}}$ allows B to authenticate A, i.e. that the entity has access to the shared key $K_{A,B}$.
    - The timestamp provides a freshness guarantee.
4. Finally, A can authenticate B.

Kerberos
Notes:

- After completing this exchange, A and B use the session key $K_{A,B}$ to keep their communications confidential.
- Since Kerberos is using timestamps and lifetimes, there is a need for secure, synchronized clocks. On a local area network (Unix, Windows, Netware, ...) this is relatively easy to implement.
- As you would expect, the authenticator in step 3 and the ciphertext in step 4 have different formats.

Asymmetric Ciphers
We now look at protocols that combine authentication and key exchange using asymmetric encryption.
The Needham-Schroeder Public-Key Protocol can be used to achieve both mutual authentication and key agreement by exchanging key material instead of nonces.
Protocol 16: Diffie-Hellman Embedded in NSPK
Objectives:

1. To allow entities A and B to authenticate each other.
2. To allow entities A and B to agree a symmetric key $K_{\langle A,B\rangle}$.

Asymmetric Ciphers
Public values: Large prime p and generator g.
Private values:
A has private value x such that $1 < x < p-1$.
B has private value y such that $1 < y < p-1$.
Protocol Steps:

1. $A \rightarrow B : \{g^x \pmod p, A\}_{K_B^+}$
2. $B \rightarrow A : \{g^x \pmod p, g^y \pmod p\}_{K_A^+}$
3. $A \rightarrow B : \{g^y \pmod p\}_{K_B^+}$

A now computes $K_{\langle A,B\rangle}$ as $(g^y)^x$ and B computes $K_{\langle A,B\rangle}$ as $(g^x)^y$.

Asymmetric Ciphers
An obvious approach is to use digital signatures for authentication and public-key encryption to protect session keys.
There are a number of ways this can be achieved.

i. The session key can be signed and then encrypted.

   1. $A \rightarrow B : \{K_{A,B}, T_A, \{|B, K_{A,B}, T_A|\}_{K_A^-}\}_{K_B^+}$

The timestamp T_A ensures freshness and including B in the signed data prevents B from impersonating A to a third party.

ii. The session key can be encrypted and signed separately.

1. $A \rightarrow B : \{K_{A,B}, T_A, \}_{K_B^+}, \{|B, K_{A,B}, T_A|\}_{K_A^-}$

This only works if the signature scheme does not allow recovery, e.g. it uses a hash function.

iii. The session key can be encrypted and then signed.

1. $A \rightarrow B : T_A, \{K_{A,B}\}_{K_B^+}, \{|B, T_A, \{K_{A,B}\}_{K_B^+}|\}_{K_A^-}$

These protocols can be extended to implement two-way authentication.

X.509

- X.509 defines a number of strong authentication protocols.
- X.509 uses slightly different terminology. In particular, it uses the terms one-way, two-way and three-way to specify the number of protocol steps.
- Like Kerberos, X.509 defines the detailed syntax of the protocol using ASN.1. We will ignore such detail here.
- The X.509 protocols support the optional exchange of certificates, but we will ignore this feature.

X.509 One-Way Authentication

Protocol 17: X.509 One-Way Authentication
Objectives:

1. To allow B to authenticate A, i.e. provide unilateral authentication.
2. To allow A to share a session key $K_{A,B}$ with B.

Protocol Steps:

1. $A \rightarrow B : token_A, \{|token_A|\}_{K_A^-}$

   where $token_A = T_A, N_A, B, data_A, \{K_{A,B}\}_{K_B^+}$

Notes:

- $data_A$ is some optional signed data that A can send to B.
- This protocol uses both timestamps and nonces to ensure freshness and prevent replays.

X.509 Two-Way Authentication
Protocol 18: X.509 Two-Way Authentication
Objectives:

1. To allow A and B to mutually authenticate each other.
2. To allow A and B to establish or exchange session keys.

Protocol Steps:

1. $A \rightarrow B : token_A, \{|token_A|\}_{K_A^-}$

   where $token_A = T_A, N_A, B, data_A, \{X_A\}_{K_B^+}$
2. $B \rightarrow A : token_B, \{|token_B|\}_{K_B^-}$

   where $token_B = T_B, N_B, A, N_A, data_B, \{X_B\}_{K_A^+}$

X.509 Two-Way Authentication
Notes:

- This protocol allows two shared values X_A and X_B to be exchanged.
  - These could be symmetric keys to be used in different directions on the connection.
  - They could be combined into a larger key.
  - They could be used with Diffie-Hellman.
- Note that $token_B$ contains the nonce N_A sent by A.

X.509 Three-Way Authentication
Protocol 19: X.509 Three-Way Authentication
Objectives:

1. To allow A and B to mutually authenticate each other.
2. To allow A and B to establish or exchange session keys.

Protocol Steps:

1. $A \rightarrow B : token_A, \{|token_A|\}_{K_A^-}$

   where $token_A = T_A, N_A, B, data_A, \{X_A\}_{K_B^+}$
2. $B \rightarrow A : token_B, \{|token_B|\}_{K_B^-}$

   where $token_B = T_B, N_B, A, N_A, data_B, \{X_B\}_{K_A^+}$
3. $A \rightarrow B : token'_A, \{|token'_A|\}_{K_A^-}$

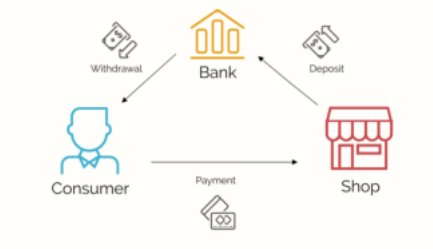
   where $token'_A = N_B, B$

X.509 Three-Way Authentication
Notes:

- This also provides mutual authentication, but does so without the need for timestamps.
- Steps $1-2$ are the same as for two-way authentication but the timestamps may be set to zero and ignored.
- This protocol is now essentially a challenge-response protocol.
  - In particular, both entities exhibit ownership of a private key by signing data containing a nonce produced by the other entity.

8 Digital Cash

Digital Cash



Transactions in a digital cash scheme typically include:

- Withdrawal of coins by customer from the bank
- Payment of coins by customer to merchant
- Deposit of coins by merchant into bank

Online scheme: bank participates in payment transaction
Offline scheme: bank does not participate in payment transaction

Digital Cash
"Blind signatures for untraceable payments" by David Chaum in 1982 first introduced the idea of digital cash.

- Security was ensured by using blind signatures to achieve unlinkability between withdrawal and spend transactions.
- Unfortunately, this idea was ahead of its time and was also mismanaged and did not catch on.

Requirements for a digital cash scheme:

1. Security
2. Anonymity
3. No forgery
4. Can be transferred
5. No double spending
6. Can be subdivided

Digital Cash
Each of the requirements for digital cash can be satisfied as follows:
1. Security

- This can be achieved by encrypting all information using public key cryptography.
- All coins can be digitally signed by a bank.

2. Anonymity

- This can be achieved using blind signatures.
- Coins can be created by a customer and blinded before being sent to the bank for signing.

Digital Cash
3. No Forgery

- The person wanting to create the coin does so using a cut and choose protocol in which they present a bunch of potential coins to the bank, with all the coins blinded.
- The bank demands the unblinding factor for all but one of the coins (randomly choosing which).
- The bank verifies that these coins are correct.
- The bank then signs the remaining coin without seeing it.
- For k coins the chance that somebody gets away with fraud is $1/k$.
- There is a more secure algorithm in which the bank only has to see the unblinding factor for half of the coins, and puts a blind signature on all the rest, concatenated together. When the coin is spent, the merchant checks that all the concatenated coins have the same value. This verification results in an attacker having a $1/2^{k/2}$ chance of getting away with fraud for k coins.

Digital Cash
4. Can be Transferred

- Each coin contains several copies of data identifying its' creator.
- Each copy of this data is split into two halves in such a way that having one half gives no information, and having both halves fully identifies the person (secret splitting).
  - For example, if the identity of the person is given by i, then we can split this secret using a random value i' to produce the values $i \oplus i'$ and i'.

- When the person spends the money, the merchant also uses a cut and choose protocol and requests the unblinding factors for one of the halves of each split identity, randomly choosing which half (using a bit commitment protocol).
- The merchant sends these unblinding factors along with the coin to the bank when they redeem the coin.

Digital Cash
5. No Double Spending

- If a coin is spent twice, the different merchants are likely to request different halves of at least one copy of the identity.
- The bank thus gets the complete identity of the double spender (even with only one copy of the split identity, the likelihood of getting caught is 50%.)

6. Can be Subdivided

- This can be achieved by using coins of different denominations.

Bitcoin
Most proposals use a centralized system:

- Tied to a traditional currency.
- No real gain over existing systems.

It would be better to create a system with distributed consensus:

- A currency that is peer-to-peer.
- All functions of a bank can be taken over by the network.

Bitcoin
The Bitcoin protocol was proposed by a person or group of people with the alias "Satoshi Nakamoto" in late 2008.

- Creation of new currency
- Secure transactions
- Protection against double-spending
- Anybody can be a "merchant" or a "customer"
- Pseudo-anonymity

Bitcoin
Makes use of a shared list of all transactions ever made: the blockchain.
Bob checks his blockchain before accepting the transaction

- If he sees that the Bitcoin in question is owned by Alice, he accepts it.
- After the transaction is complete, Bob broadcasts his acceptance.
- As soon as the other peers hear this broadcast, they will not allow double-spending.

Alice can perform a double-spend before the acceptance broadcast is heard by enough peers

- To solve this problem, we make Bob ask everybody else if a transaction is valid.
- Double-spending will be noticed before payment is accepted.

Bitcoin
How many answers should Bob require? How can the answers be trusted?

- A "majority vote" is impossible, if Alice spams Bob with false confirmations.
- There is no way to perform traditional authentication.
- But Bitcoin won't work if transactions can't be reliably verified...

The finished Bitcoin protocol uses Proof of Work (PoW).

- Basic idea: We only trust solutions that are accompanied by a proof of someone having committed a large amount of resources to a problem.
- That is, we don't authenticate a user, but we authenticate the fact that time/money/energy/etc. has been spent.
- In order for Alice to make a double-spend, she first has to spend energy before Bob trusts her.

Bitcoin
We want a problem with the following properties:

- is difficult to solve
- has solution(s) that are easy to verify
- has scalable difficulty

A cryptographic hash function is pre-image resistant, so finding pre-images is the perfect proof of work.
The verifications are done by miners:

- For transaction message m, a miner selects a random k and computes $h(m+k)$.

- If $h(m+k) > T$ (where T is the threshold), the miner chooses a new k and tries again.
- After a long time we get $h(m+k) < T$ and the miner broadcasts k.
- Bob receives k and checks that $h(m+k) < T$.

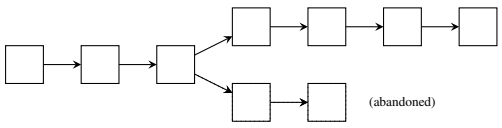
Bitcoin
Let the threshold T be so that the hash value $h(m+k)$ needs five leading zeros and let m ="AAA"

$m+k$	$h(k+m)$
AAA0	802dbe2e69...
AAA1	bbfce0d522...
AAA2	7bb4db476f. .
...	...
AAA770239	00000921ac...
k = 770239 is a valid solution	

Note that in the normal case, k is chosen randomly.
There are several solutions k to the problem $h(m+k) < T$

Bitcoin

- A block is a large number of transactions.
- The process of turning transactions into blocks is mining.
- Mining is a competition to find a solution.
- The blocks are numbered and form a long chain, the blockchain:



If two miners find a valid block simultaneously, the resolution strategy is to randomize and then work on the longest chain.

Bitcoin
The only way for Alice to cheat is the following:

1. Buy a supercomputer
2. Save up money for the electric bill
3. Broadcast an invalid transaction m to Bob
4. Let the supercomputer search for a block containing m.

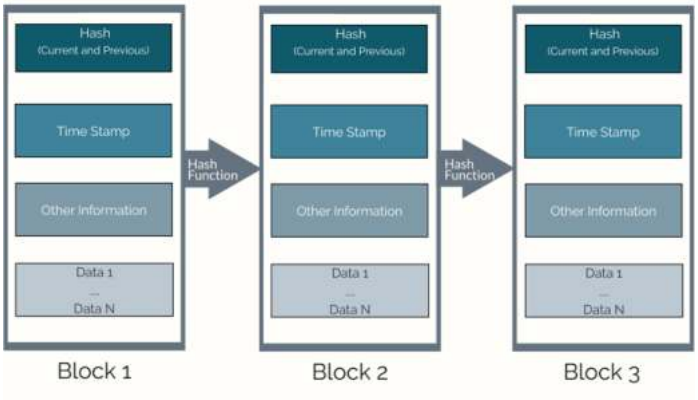
5. The computer must be faster than everybody else's, combined.
6. Even if she manages to solve an "illegal" block, no other miner will accept it.

Alice has a hard time cheating Bob.
Even if she has 1% of the hashing power, the chance of mining six blocks in a row is $(0.01)^6 = 1 \times 10^{-12}$.

Bitcoin
Bitcoin is quite robust against man-in-the-middle attacks:

- Assume that Alice and Bob have a good copy of the blockchain.
- Eve cannot intercept the transaction and take the money, since Alice and Bob require a proof of work.
- Eve would have to spend a very long time finding a block, so Alice and Bob would notice.

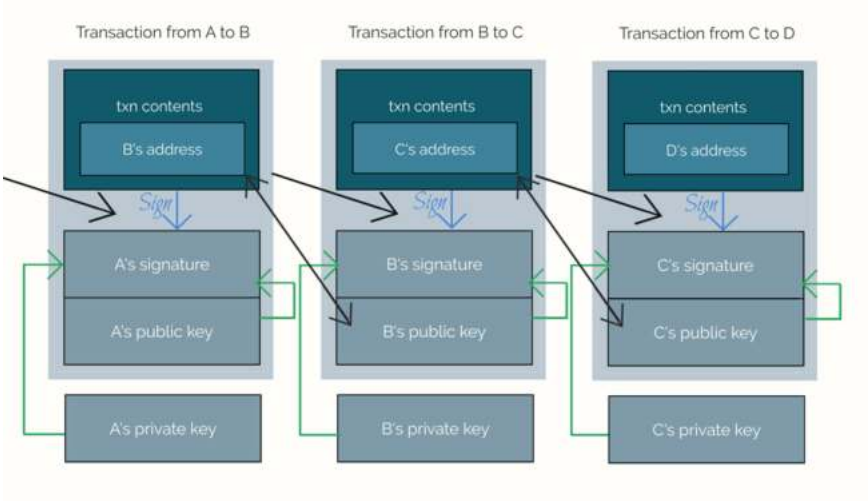
Bitcoin
Each block gives security to the previous ones:



Bob waits a number of blocks before accepting Alice's transaction.

Bitcoin

Detailed view of a transaction:



Bitcoin

- Digital signatures initiate the transaction
- Miners verify the transactions
- Bob accepts the transaction after six successive blocks (takes one hour)
- New currency is created by rewarding miners
- All transactions are in the blockchain
- The threshold T provides a way to adjust the difficulty of the proof of work.
- Bitcoin also has built-in inflation control: every fourth year the mining reward is halved.
- In Bitcoin, the users only need to trust the algorithm, nothing else.
- Transactions are safe, storage is not.

Blockchain
Three levels of blockchain:

1. Storage for digital records
2. Exchanging digital assets (tokens)
3. Executing smart contracts

Geoff Hamilton

- Ground rules: Terms and conditions recorded in code
- Distributed network executes contract and monitors compliance
- Outcomes are automatically validated without third party

9 Conclusion

Cryptography
Security objectives addressed by cryptography:

- Confidentiality
  - symmetric cryptography
  - asymmetric cryptography
- Integrity
  - hashing
- Authentication and non-repudiation
  - digital signatures

Cryptography
Types of attack:

Ciphertext only attack: ciphertext known to the adversary (eavesdropping)

Known plaintext attack: plaintext and ciphertext are known to the adversary

Chosen plaintext attack: the adversary can choose the plaintext and obtain its encryption (for example, has access to the encryption system)

Chosen ciphertext attack: the adversary can choose the ciphertext and obtain its decryption

Historical Ciphers
Monoalphabetic:

- Shift Cipher
- Substitution Cipher

Polyalphabetic:

- Vigenère Cipher
- Vernam Cipher (one-time pad)
- Rotor Machines

Security of Ciphers
Computational security: best known algorithm for breaking cipher requires an unreasonably large amount of computer time.

- Shift cipher, substitution cipher and Vigenère Cipher are not computationally secure.
- Block ciphers such as AES and public-key ciphers such as RSA are computationally secure.

Unconditional security (perfect secrecy, information-theoretical security, semantic security): no bound on the computational power of the adversary.

- Block ciphers such as AES and public-key ciphers such as RSA are not unconditionally secure.
- Vernam cipher (one-time pad) is unconditionally secure if used correctly.

Stream Ciphers
Encrypt an arbitrary stream of data by combining it with a keystream.
Examples: A5/1, RC4.
Keystream can be generated by a cryptographically secure pseudo-random generator (PRG) seeded with the key and should:

- Look random
- Be unpredictable
- Have large linear complexity
- Have low correlation between key bits and keystream bits.

The linear congrentual generator is not cryptographically secure, but the Blum Blum Shub generator is.
Most stream ciphers are based on a non-linear combination of LFSRs (Linear Feedback Shift Registers).

Block Ciphers
Plaintext is divided into blocks of fixed length and every block is encrypted one at a time.
DES:

- S-P Network, iterated cipher, Feistel structure
- 64-bit block size, 56-bit key size
- 8 different S-boxes
- non-invertible round
- design optimised for hardware implementations

AES:

- S-P Network, iterated cipher
- 128-bit block size, 128-bit (192, 256) key size
- one S-box
- invertible round
- design optimised for byte-orientated implementations

Block vs Stream Ciphers
Block ciphers:

- More versatile: can be used as stream cipher.
- Standardisation: DES and AES + modes of operation.
- Very well studied and accepted.

Stream ciphers:

- Easier to do the maths.
- Either makes them easier to break or easier to study.
- Supposedly faster than block ciphers (less flexible).

Modes of Operation
Block ciphers can be used in different modes of operation:

- Electronic Code Book (ECB mode)
- Cipher Block Chaining (CBC mode)
- Output Feed Back (OFB mode)
- Cipher Feed Back (CFB mode)
- Counter (CTR mode)

Hash Functions
A hash function is an efficient one-way function mapping binary strings of arbitrary length to binary strings of fixed length called the hash-value or digest.
A hash function should have the following properties:

- Pre-image resistant: given a digest, it should be computationally infeasible to find a piece of data that produces the digest.
- Weakly collision-free or second pre-image resistant: given message M it is computationally infeasible to find a different message M' such that $H(M) = H(M')$.
- Strongly collision-free: it is computationally infeasible to find different messages M and M' such that $H(M) = H(M')$.

Hash Functions
The birthday paradox: if there are n possibilities then on average $\sqrt{n}$ trials are required to find a collision.
Most practical hash functions make use of the Merkle-Damgård construction.
Applications of hash functions: message authentication, digital signatures, password storage, key generation, pseudorandom number generation, intrusion detection, virus detection.
Example hash functions: MD2, MD4, MD5, SHA-1, SHA-2, SHA-3, RIPEMD, RIPEMD-160.

MDCs and MACs
Used to ensure integrity of data.
Manipulation Detection Code (MDC): hash function without key.
Message Authentication Code (MAC): hash function with secret key.
Types of MAC:

- MACs based on block ciphers in CBC mode (CBC-MAC).
- MACs based on MDCs (e.g. HMAC).
- Customized MACs.

Public Key Cryptography
Each user has a key pair, which consists of a public key (made public, used for encryption) and a private key (kept secret, used for decryption).
Make use of number-theoretic problems to implement trapdoor one-way functions.
Example one-way functions:

- Multiplication
  - Inverse problem: factoring
- Modular exponentiation
  - Inverse problem: discrete logarithm

Public Key Cryptography
Hard Problems:

- Integer factorisation problem
- RSA problem
- Quadratic residuosity problem
- Square root problem
- Discrete logarithm problem
- Diffie-Hellman problem

- Decisional Diffie-Hellman problem

We can compare the relative difficulty of some of these problems using reductions.

Public Key Cryptography
Public key algorithms:

- Key exchange: Diffie-Hellman
- Encryption: encrypt using public key, decrypt using private key.
- Digital signature: encrypt using private key, decrypt using public key.
  - Schemes with message recovery.
  - Schemes with appendix.

Public Key Cryptography: Encryption
RSA
Encryption and decryption using modular exponentiation.
Encryption can be made more efficient using the square and multiply algorithm, and selecting an encryption exponent with very few bits set e.g. $2^{16}+1$.
Decryption can be made more efficient by making use of the knowledge of prime factors of the modulus and using the Chinese Remainder Theorem.
The security of RSA relies on the difficulty of the integer factorisation problem.
Raw RSA is not semantically secure, so a padding scheme should be used to add randomness and redundancy.

Public Key Cryptography: Encryption
Other public key algorithms:

- ElGamal
  - Security relies on the difficulty of the discrete logarithm problem.
  - Used as the basis of the Digital Signature Algorithm (DSA).
- Rabin
  - Security relies on the difficulty of the square root problem with a composite modulus.
  - Can be viewed as similar to RSA with an encryption exponent of 2.

Main drawback of public key cryptography is the inherently slow speed.
Therefore, public key schemes are not used directly for encryption.
Instead, they are used in conjunction with secret key schemes.

- Encryption is performed by secret key schemes (e.g. AES).
- Key agreement is performed by public key schemes (e.g. RSA or Diffie-Hellman).

Public Key Cryptography: Digital Signatures
Public key algorithms for digital signatures:

- RSA
  - Deterministic signatures: for each message, one valid signature exists
  - Faster verifying than signing
- ElGamal
- DSA: based on ElGamal
  - Non-deterministic signatures: for each message, many valid signatures exist
  - Faster signing than verifying
- Blind signatures
  - Signing of message without learning anything about it
  - Can be used in online elections

Key Exchange and Authentication Protocols
Secure communications protocols normally have a number of phases:

1. Authentication Phase
2. Key-Exchange Phase
3. Data Transfer Phase

Authentication techniques:

- Passwords
- Symmetric ciphers
- Asymmetric ciphers
- Zero knowledge Proof

Key exchange techniques:

- Diffie-Hellman Key Agreement
- Symmetric ciphers
- Asymmetric ciphers

Key Exchange and Authentication Protocols
Protocols for authentication:

- Needham-Shroeder Public-Key Protocol
- Fiat-Shamir Protocol

Protocols for authentication and key exchange:

- Needham-Shroeder Secret-Key Protocol
- Otway-Rees Protocol
- Kerberos
- Diffie-Hellman Key-Agreement embedded in Needham-Shroeder Public-Key Protocol
- X.509 Authentication Protocols

Key Exchange and Authentication Protocols
Attacks on protocols:

- Replay attacks
- Reflection attacks
- Interleaving attacks
- Chosen plaintext attacks
- Man-in-the-middle attacks

Digital Cash
Digital Cash

- Authentication: digital signature
- Anonymity: blind signature

Offline Digital Cash

- Authentication: zero knowledge proof
- Preventing double spending: cut and choose

Bitcoin

- Secure transactions
- Protection against double-spending
- Pseudo-anonymity
- Digital signatures initiate the transaction
- All transactions are in the blockchain
- Miners verify the transactions