

CSCI361
Computer Security
(Cryptography and secure applications)

Subject introduction

Lecturer & Subject Coordinator

Professor Willy Susilo

School of Computing and Information
Technology

Email: wsusilo@uow.edu.au

Tutor: Mr. Japit Sionggo

What is this subject about?

- We cover a wide range of topics in computer security.
 - We will put computer security into perspective later in this lecture.
- From understanding cryptographic algorithms ...
 - ... through security programming (assignments) ...
 - ... to applying cryptography to real-world applications.
- It is assumed that you can program:
 - In either Java or C/C++.
 - If you cannot, *you will need to learn!*
 - A significant proportion of the assignment assessment is for programming, either in C/C++/Java or using Number Theory Package.

The objectives of this subject.

- This is what we hope you will be able to do by the end of subject:
 - Understand fundamental cryptographic principles, including the types of cryptography and their properties.
 - Understand some basic building blocks of computer security (encryption, authentication, hashing ...).
 - Understand and know some of the algorithms used to provide examples of those building blocks.
 - Identify security problems in computer systems.
 - Understand how to apply security algorithms to real-world applications.

Approximate Contents

- Introduction.
- Classical cryptology, Secret key cryptography.
- Modern secret key cryptography, block ciphers.
- Modern stream ciphers, AES.
- Message integrity, Public key cryptography
(Knapsacks, RSA).
- Public key cryptography. Digital signatures.
- Digital signatures, hashing.
- Secret Sharing Schemes

Assessment

- 2 Assignments.
 - Programming may be required in all assignments, although the entire assignment need not be all programming.
- One written test.
- The final exam.

Resources

■ References:

- *Cryptography Engineering*. Niels Ferguson, Bruce Scheiner and Tadayoshi Kohno, Wiley, 2010.
- *Cryptography and Network Security: Principles and Practices*. William Stallings. 4rd edition. Prentice Hall, 2005.
- Cryptography: Theory and Practice. D. Stinson. 3nd edition. CRC Press, 2005.
- *Fundamentals of Computer Security*. J. Pieprzyk, T. Hardjono and J Seberry, Springer-Verlag, 2003

CSCI361 – Introduction
Computer Security

What, why and who?

Outline

- We are basically going to address three questions.
- **What** do we mean by (computer) security?
 - Threats, attacks and vulnerabilities.
 - Distributed systems.
 - Design principles.
- **Why** does computer security matter?
- **Who** needs computer security?

Computer security: What is it?

- Briefly at this stage:
- Computer security is about protecting computer based *assets* against possible *threats*.
 - Primarily we are interested in protecting information from *attack*.

Why does computer security matter?

- Computer security matters because people do actually attack computer systems, for various reasons.
 - For money.
 - To obtain knowledge or intellectual property.
 - Industrial sabotage.
 - For fun.
 - Because they can ...

Eugene Spafford:

“Using encryption on the Internet is the equivalent of arranging an armored car to deliver credit-card information from someone living in a cardboard box to someone living on a park bench.”

Who needs computer security?

- **Governments:**
 - To safeguard military or diplomatic communications and to protect national interests.
- **Private sector:**
 - To protect sensitive information such as health and legal records, financial transactions, credit ratings.
 - To protect information ownership.
- **Individuals:**
 - To protect sensitive information, and to protect an individual's privacy in the electronic world.
 - Allow E-commerce, internet banking and so on.

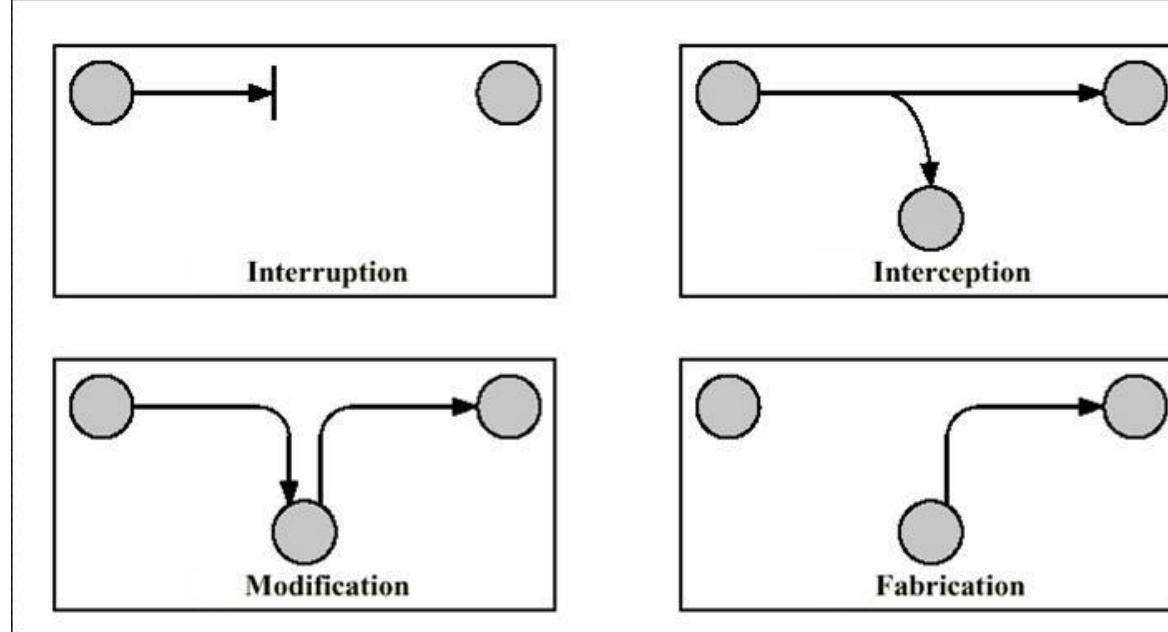
What do we mean by security?

- Security is about protecting *assets* against possible *threats*.
 - In particular we are concerned with computer security, but primarily in the sense of *information security*.
- **Assets** are anything we possess of value or use to us.
 - A computer system consists of **hardware**, **software** and **data**, each of which is an asset.
- A **threat** is something which potentially violates security. It exists when there is a circumstance, capability, action or event that could breach security and cause harm. In other words, a threat is a possible danger that might exploit a vulnerability.

- We need to be able to identify
 - Assets.
 - Threats.
 - Possible controls.

... and estimate the cost and resulting benefits of implementing the controls.
- We would also like to be able to identify when **attacks** occur.
 - We use the term **attack** to mean a deliberate attempt to evade security services and violate the security policy of a system.
- Damage to assets can be intentional or accidental.
 - We will be mainly concerned with intentional damages, i.e. where people undertake attacks.

The four basic attack types



- Interruption: an attack on availability of an asset.
 - Hardware destruction, software erasure.
- Interception: an attack on confidentiality.
 - Wiretapping network, illegal copying of files.
- Modification: an attack on integrity.
 - Of database data or transmitted communications.
- Fabrication: an attack on authenticity.
 - Pretending to be someone else.

Asset vulnerabilities

- Hardware is vulnerable to such accidental damage as coffee, dust or power surges, although neither may be accidental.
- Hardware is vulnerable to deliberate damage such as theft or tampering.

- Software/Data may be damaged due to accidental media damage, for example, accidentally dropping a cup of coffee on a CD. Or someone accidentally deleting some files.
- It can also suffer from deliberate attacks which modify the purpose or content of the software/data.

- Damage can be:
 - **Easily detectable:** For example the software crashes when it is run or data is easily observed as being corrupted. Effectively this provides a denial of service.
 - **Not easily detectable:** For example, the replaced data looks realistic or software runs correctly but has an additional hidden purpose. Detection may only be possible through side effects, if at all.

Damage to software/data

- Deletion (interruption): Erasing a file, or copying it.
- Modification:
 - *Software modification* may cause a program to crash immediately, or at a certain time (logic bomb), or it can make program do what it is not supposed to do. For example. modifying access rights while copying.
 - *Data modification* can take many forms. Replaying used data, fabrications of messages etc.
- Software interception: Stealing software (including piracy).
- Data interception: Breaching confidentiality of data by wiretapping or monitoring electromagnetic radiation.

Distributed systems

- New distributed systems have added problems:
 - Laptops, handheld devices and mobile phones can be easily lost.
 - Hardware security cannot be relied on.
 - Communication between different parts of the system exposes the data. It also provides many more attack points for intruders to attack data and other resources.
- Sharing resources and access control is a much more complex problem than for single point systems.

Controls

- Hardware controls:
 - Devices for user identification,
 - Hardware implementation of encryption.
 - Chip sets with embedded security functionality,
 - Trusted systems (Microsoft Palladium/NGSCB).
- Software controls:
 - Standards for coding, testing and maintaining, to ensure software correctness.
 - Operating system controls on accessing data and programs.
 - Internal controls, for example, data base management systems access control.
- Cryptography is a powerful tool in providing security.
It can add security to an “insecure” system.

Control policies

- The protection we can provide varies depending on the situation. Primarily we are interested in using **policy** or **protocol-based security** centred around cryptography.
- **Policies** are working procedures adopted by organisations to improve asset security.
 - Requiring frequent password changes.
- **Protocols** are agreed upon rules or standards enabling connection and interaction between parties.
 - They can specify data formats.
 - Rules of exchange, who does what when?
 - Specify termination or error rules or handling conditions.
- In this course we are only concerned with security of software and data, primarily data (information).
 - The first half of the course will be spent looking at cryptography.

Goals of security (CIA)

- ***Confidentiality***: Assets should be inaccessible to unauthorised parties.
- ***Integrity***: Assets should be unmodifiable or unforgeable, without detection, by unauthorised parties.
- ***Availability (Authenticity)***: Assets should be available to authorised people.
- There is a cost in achieving those goals, and one needs to balance this cost against what you can gain.

Security Principles: Construction and analysis

- Principle of **easiest penetration**:
 - Intruders will use any available means of penetration. This makes security assessment of security a very difficult problem because *all possible* ways of breaching security must be examined.
- Principle of **adequate protection**:
 - Also known as the *timeliness principle*, this means items should only be protected while they are valuable, and that the level of protection should be consistent with their value. This is a very practical principle which underlies a large proportion of modern computer security.
- Principle of **effectiveness**:
 - Controls must be used properly to be effective.
 - Controls should be efficient, easy to use and appropriate.
- Principle of the **weakest link**:
 - Security is only as strong as the weakest link in the system.

Other questions...

- We also want to think about **when** to use cryptography?
- ... and **which** cryptography to use in a given situation?

CSCI361

Computer Security

Classical cryptology

Outline

- What is *cryptology*?
 - Cryptography.
 - Cryptanalysis.
- Communication model.
- Secret-key cryptography.
- Epochs.
- Early ciphers:
 - Caesar.
 - Monoalphabetic.
- Statistical cryptanalysis.

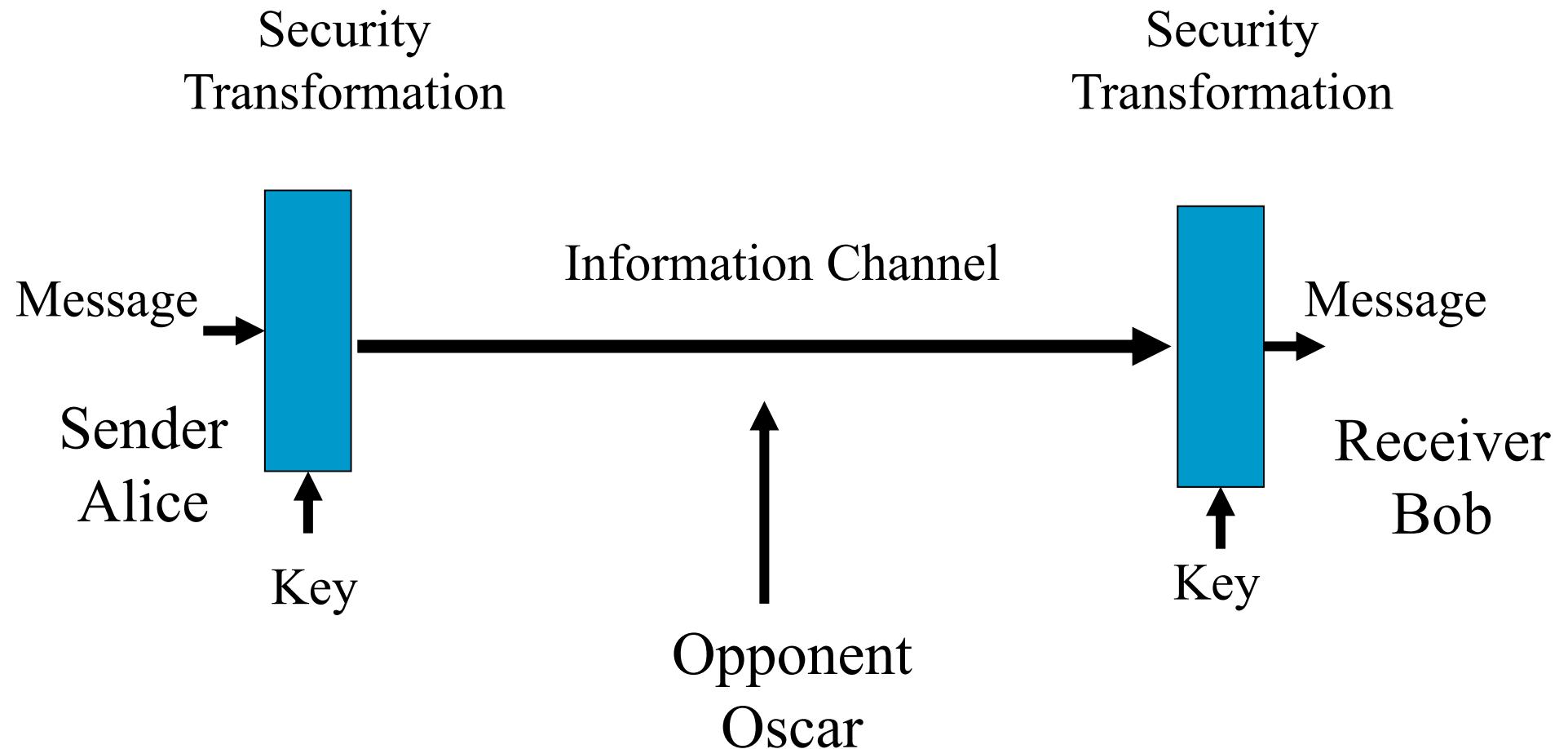
Cryptology

- From the Greek words:
 - *kryptos* meaning “hidden”
 - *logos* meaning “word”

Cryptology is the art/science of secure communication. It splits into...

- **Cryptography:** Designing algorithms to ensure security: i.e. confidentiality, integrity and authenticity.
- **Cryptanalysis:** Analysing security algorithms with the aim of breaching security.

The basic secrecy channel



The basic secrecy channel

- The channel can be a *communication channel* or a *storage channel*.
- Sender (A for Alice) wants to send a message X to the Receiver (B for Bob), through this channel, such that the opponent/enemy/intruder O (O for Oscar) cannot access X.
- Alice applies a transformation, known as **encryption**, to X, referred to as the **plaintext**, to produce a garbled message Y, referred to as the **ciphertext** (or cryptogram).
- Bob applies another transformation, known as **decryption**, to Y to obtain the plaintext again.

Key dependence

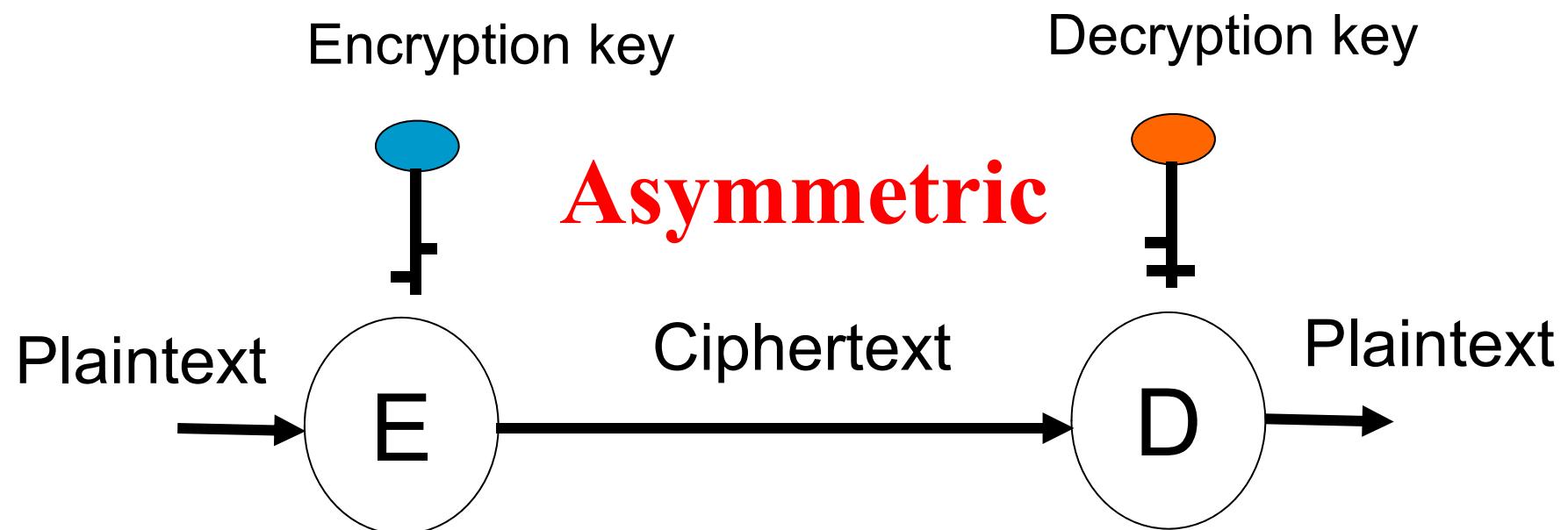
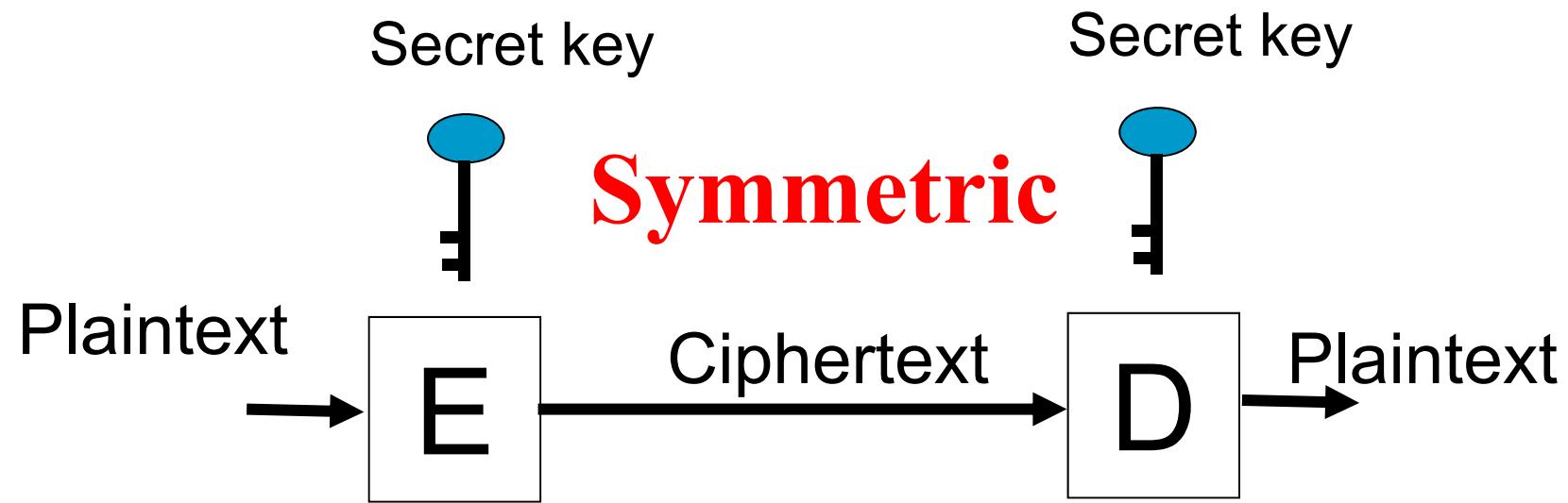
- The transformations are not fixed, they are **key** dependent. The **key K** controls the transformation and is known only by Alice and Bob. The key is secret.
- Encryption and decryption are sometimes referred to as enciphering and deciphering, respectively.
- Note that if a transformation **does not** depend on a key, it is referred to as **encoding**, with the inverse transformation being referred to as **decoding**.
 - Morse code.
 - ASCII code.

Secret-key cryptography

- In secret key cryptography the participants all have only secret key information.
- Basically this means there is no role, other than as an attacker, for anybody who doesn't hold a secret key of some sort.
- We shall later look at public-key cryptography, where persons without secret-keys can play a role (other than an attacker) in the cryptosystem.

Symmetric and asymmetric encryption

- In classical cryptography the encryption and decryption keys are the same, this is an example of a symmetric encryption scheme.
- In asymmetric encryption the encryption and decryption keys are different, this is the case in public-key encryption.
- We emphasise that the terms symmetric and private are not equivalent. It is possible to have private asymmetric key systems (not necessarily encryption), although we shall not be discussing such in this course.



Kirchoff's Law

- This is the main assumption of cryptology.

The cryptanalyst knows all the details of the encryption and decryption transformations, except for the value of the secret key or keys.

Some possible attacks

- Oscar is trying to decrypt a particular ciphertext, and possibly (although it is harder in general) to figure out the key.
- *Ciphertext only*: Oscar knows Y. Alice and Bob don't want Oscar to figure out what either X or K is.
- *Known plaintext*: Oscar knows some X-Y pairs. Alice and Bob don't want Oscar to figure out what K is, or the correspondence between other X-Y pairs.
- *Chosen plaintext*: Oscar is allowed to choose some plaintexts (X's), and receives the corresponding ciphertexts (Y's).
- *Chosen ciphertext*: Oscar is allowed to choose some ciphertexts (Y's), and receives the corresponding plaintexts (X's).
- Some combination of these.

Epochs in cryptology

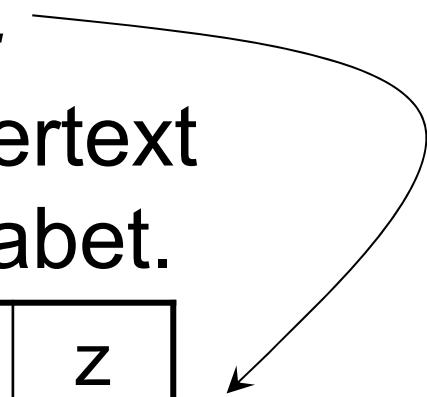
- Non-scientific cryptography: from antiquity until 1949.
Cryptology was more an “art” than a science.
- Scientific cryptography starts with Shannon's paper :
Communication theory of secrecy systems (1949).
This was based on Shannon's 1948 paper in which he had founded information theory.
- Cryptologic research really took off in 1976 with the paper
New directions in cryptography, by Diffie & Hellman.
They showed that secret communication is possible without a shared key.

Early ciphers

- Studying some early ciphers is useful because the empirical principles developed through their use are applied in the design and analysis of modern ciphers.
- Caesar cipher: (Julius Caesar 2050 years ago).
 - Every letter is replaced by the letter “three to the right” in the alphabet, where this operation is cyclic. A→D, B→E, ... X→A, Y→B, Z→C
 - For example: CABBAAGE → FDEEDJH
 - There is no key though ☹, so it isn’t a true cipher!
 - The generalised Caesar (or shift) cipher allowed the 3 to be replaced by an value between 1 and 25 inclusive.

Monoalphabetic ciphers

- Also known as simple substitution ciphers, each letter of the plaintext alphabet is replaced with an element of the ciphertext alphabet.
- The substitution alphabet is the *key*.
- Consider that the plaintext and ciphertext alphabets are both the English alphabet.



a	b	c	d	e	...	x	y	z
F	G	N	T	A	...	K	P	L

a		b	a	d		d	a	y
F		G	F	T		T	F	P

- Example: Consider the plaintext and ciphertext alphabets to be the set of binary strings of length 3.

$$K = \begin{array}{|c|c|c|c|c|c|c|c|} \hline & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ \hline & 101 & 111 & 000 & 110 & 010 & 100 & 001 & 011 \\ \hline \end{array}$$

- Plaintext: 100 101 111
- Ciphertext: 010 100 011

Unique decryption: One-to-one.

- In order for decryption to be unique, we need one-to-one mappings. Consider...

a	b	c	d	e	...	x	y	z
F	G	N	G	A	...	K	P	L

a		b	a	d		d	a	y
F		G	F	G		G	F	P

- Decryption: a bad day, a dad day, a bab day, a dab day, a bad bay, a dad bay, a bab bay, a dab bay.

Security: Monoalphabetic ciphers

- To decipher a ciphertext we need to know the substitution alphabet (key), or at least the subset of the key corresponding to those symbols which appear in the ciphertext.
- One can use an *exhaustive key search*, to find the full key. This means use each key to decipher the ciphertext and accept the one that produces a meaningful plaintext.
- For an alphabet of size N , the number of possible keys is $N!$.
- The key is N elements long.

$$N! \approx \sqrt{2\pi N} \left(\frac{N}{e}\right)^N$$

- For the English alphabet there are $26! \approx 4 \cdot 10^{26}$ keys.
- The key length is 26 symbols long, although the information content is 25 symbols long.
- We need 5 bits (log base 2 of 26) to represent each symbol.

Weak keys

- Not every substitution alphabet is suitable.
- For example, a possible substitution which doesn't hide the message very well at all is:

a	b	c	d	e	...	x	y	z
X	B	C	D	E	...	A	Y	Z

c	o	m	p	u	t	e	r
C	O	M	P	U	T	E	R

- In most ciphers there are some *weak keys*.

Properties of keys

- Keys must be easy to remember, a long random string is difficult to remember.
- The key set must be large enough so that *exhaustive key search* is not easy.
- To reduce the number of keys we may restrict ourselves to an indexed subset of all possible substitutions and use the index to identify the substitution that is used.
- In this case the cipher algorithm is the collection of substitutions, and the key is the index.
- Additive and multiplicative ciphers are examples of such indexed constructions.

Additive ciphers

- Also known as *translation ciphers*, the substitution alphabet is obtained by shifting the plaintext alphabet by a fixed value. The amount of the shift is the *key*.
- For example with the key of 3 we have

a	b	c	d	e	...	x	y	z
D	E	F	G	H	...	A	B	C

which is the substitution alphabet for the Caesar cipher.

Modular addition for additive ciphers

- Additive ciphers can be described by modular addition.

a	b	c	d	e	...	x	y	z
0	1	2	3	4	...	23	24	25

- Plaintext character X, ciphertext character Y, shift (key) Z. Each of X,Y,Z is an element of the set $\{0,1,2,3,\dots,25\}$ and they are related by $Y = X \oplus Z$, where \oplus denotes addition modulo 26. There are 26 keys, so bits are needed to represent the key.
- This is a key space, hence *exhaustive key search* is a feasible attack against additive ciphers.

Multiplicative Ciphers

- The plaintext alphabet will again be taken as the set $\{0, 1, \dots, 25\}$.
- The ciphertext alphabet can be determined using modular multiplication. We multiply each plaintext alphabet by a constant value, which is the *key* for this cipher.
- Using similar notation to that for additive ciphers we write

$$Y = X \otimes Z$$

where \otimes represents multiplication modulo 26.

- Not all possible numbers for the key Z will result in a one-to-one mapping. The set of keys is therefore a subset of $\{0, 1, \dots, 25\}$.
- Consider $Z=2$

$$1 \otimes 2 = 2 \quad b \rightarrow c$$

$$14 \otimes 2 = 2 \quad o \rightarrow c$$

- For all even numbers, and 13, the mapping not one-to-one. Hence the number of keys is 12, we need only 4 bits to represent the key.
- Again exhaustive key search can easily break the cipher (find the secret key).

Affine ciphers

- To increase the number of keys we can combine additive and multiplicative ciphers to obtain *affine* ciphers.

$$Y = \alpha \otimes X + Z$$

where $X, Y, Z \in \{0, 1, \dots, 25\}$

and $\alpha \in \{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$

Number of keys = $12 * 26 = 312$.

Still too small!

Key phrase based ciphers

- The next thing we can try and do is use a phrase as the key (index). We can also specify a starting letter for the translation alphabet.
- This increases the size of the key space but makes the key (index) easy to remember.

- Phrase: bubble bath
- Starting letter: e

a	b	c	d	e	f	g	h	i
W	X	Y	Z	B	U	L	E	A
j	k	l	m	n	o	p	q	r
T	H	C	D	F	G	I	J	K
s	t	u	v	w	x	y	z	
M	N	O	P	Q	R	S	V	

- This cipher is significantly more resistant to exhaustive key search, but ...

Statistical cryptanalysis

- ... it is insecure against statistical cryptanalysis.
- Statistical properties of the plaintext language can be used to cancel many keys in one step and enable the cryptanalyst to find the key without trying all of them.
- Statistical analysis relies on there being a relationship between the statistical properties of the plaintext and the statistical properties of the ciphertext, since we assume the attacker has only the ciphertext.

Statistics of the English language

- The letters can be grouped according to the frequency with which they occur.

I	e
II	t a o i n s h r
III	d l
IV	c u m w f g y p b
V	v k j x q z

- The frequency of pairs of consecutive letters (bigrams) and triples of consecutive letters (trigrams) are important clues to cryptanalysts, as are spaces between words.
- Frequent bigrams:
th, he, in, er, an, re, ed,
on, es, st, en, at, to
- Frequent trigrams:
the, ing, and, her, ere, ent
tha, nth, was, eth, for, dth
- It is important to realise that frequency counts only provide clues to the actual key used. The distribution will differ from sample to sample.

- What is the plaintext associated with the following ciphertext?
- According to Kirchoff's Law the encryption algorithm is known to the attacker; it is **key phrase substitution**.

YKHLBA JCZ SVIJ JZB TZVHI JCZ VHJ DR IZXKHLBA VSS
RDHEI DR YVJV LBXSKYLBA YLALJVS IFZZXC CVI
LEFHDNZY EVBLRDSY JCZ FHLEVHT HZVIDB RDH JCLI
CVI WZZB JCZ VYNZBJ DR ELXHDZSXJHDBLXI JCZ
XDEFSZQLJT DR JCZ RKBXJLDBI JCVJ XVB BDP WZ
FZHRDHEZY WT JCZ EVXCLBZ CVI HLIZB
YHVEVJLXVSST VI V HZIKSJ DR JCLI HZXZBJ
YZNZSDFEZBJ LB JZXCBDS DAT EVBT DR JCZ XLFCZH
ITIJZEI JCVJ PZH Z DBXZ XDBILYZHZY IZXKHZ VHZ BDP
WHZVMVWSZ.

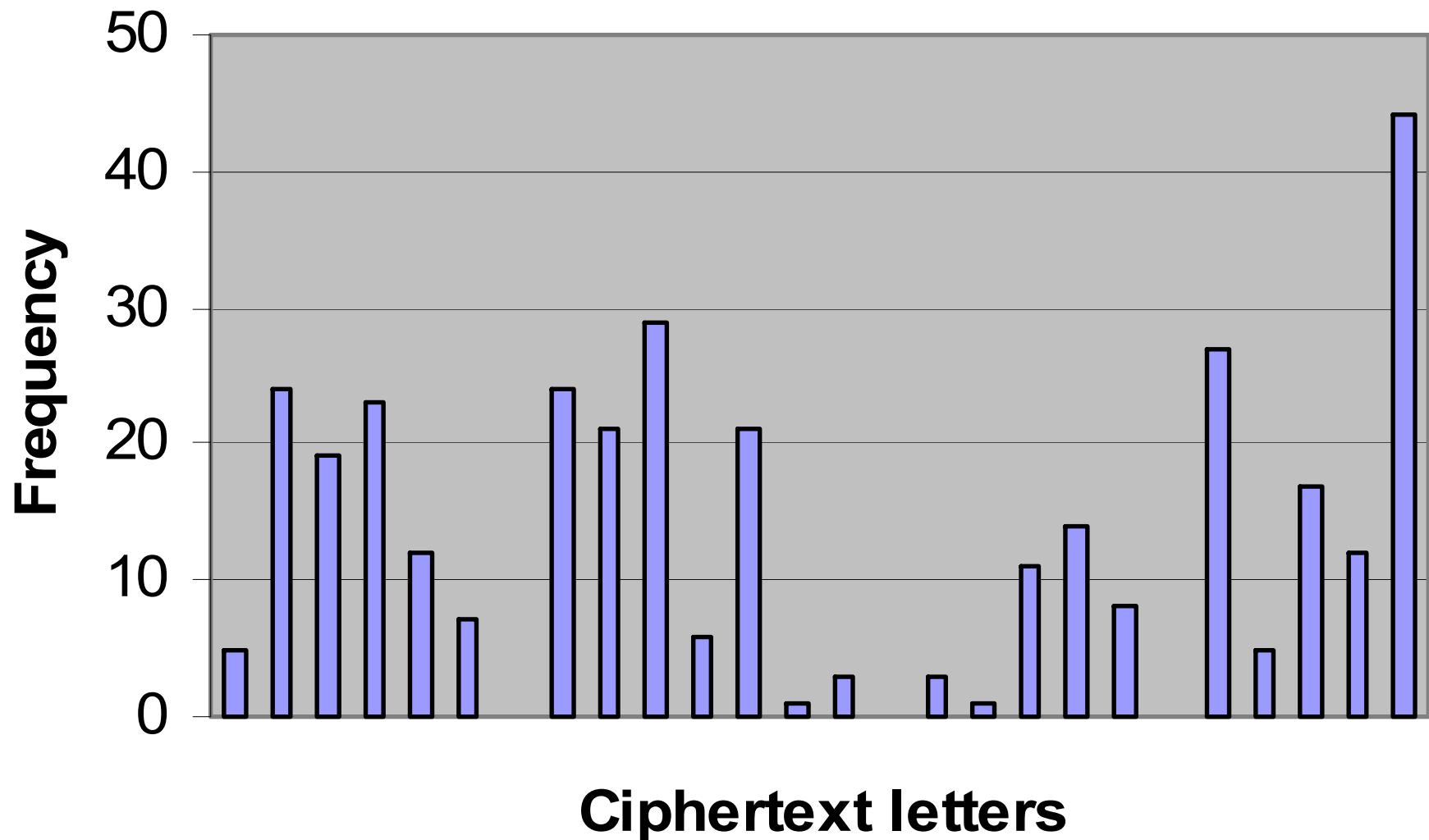
Breaking the cipher

- There are 337 letters, with frequencies:

A	B	C	D	E	F	G	H	I
5	24	19	23	12	7	0	24	21
J	K	L	M	N	O	P	Q	R
29	6	21	1	3	0	3	1	11
S	T	U	V	W	X	Y	Z	
14	8	0	27	5	17	12	44	

- This suggests we should first try **Z** being the encryption of **e**.

Ciphertext distribution



- There are 8 **JCZ** in the ciphertext, this is almost certainly **the** in the plaintext.
- The single letters will generally be **i** or **a**. In this case there is a single letter word **V** in the ciphertext.
- The word **JZB** in the ciphertext can be identified by looking at the word **teB** and noting that **B** occurs in the second frequency group for this ciphertext.
 - Some of those letters (**t a o i n s h r**) have already been identified.

- After a few of these kind of steps we can build up a preliminary mapping such as:

a	b	c	d	e	f	g	h	i
v				z			c	l
j	k	l	m	n	o	p	q	r
				B	D			H
s	t	u	v	w	x	y	z	
I	J							

- Most of the time we would probably be safe to guess f as the starting position. With that assumption we can fill b,c,d → W,X,Y!

a	b	c	d	e	f	g	h	i
v	w	x	y	z	r	A	c	L
j	k	l	m	n	o	p	q	r
O	M	S	E	B	D	F	G	H
s	t	u	v	w	x	y	z	
I	J	K	N	P	Q	T	U	

YKHLBA JCZ SVIJ JZB TZVHI JCZ VHJ DR IZXKHLBA VSS
RDHEI DR YVJV LBXSKYLBA YLALJVS IFZZXC CVI
LEFHDNZY EVBLRDSY JCZ FHLEVHT HZVIDB RDH JCLI
CVI WZZB JCZ VYNZBJ DR ELXHDZSXJHDBLXI JCZ
XDEFSZQLJT DR JCZ RKBXJLDBI JCVJ XVB BDP WZ
FZHRDHEZY WT JCZ EVXCLBZ CVI HLIZB
YHVEVJLXVSST VI V HZIKSJ DR JCLI HZXZBJ
YZNZSDFEZBJ LB JZXCBDS DAT EVBT DR JCZ XLFCZH
ITIJZEI JCVJ PZH Z DBXZ XDBILYZHZY IZXKHZ VHZ BDP
WHZVMVWSZ.

during the last ten years the art of securing all forms of data including digital speech has improved manifold the primary reason for this has been the advent of microelectronics the complexity of the functions that can now be performed by the machine has risen dramatically as a result of this recent development in technology many of the cipher systems that were once considered secure are now breakable.

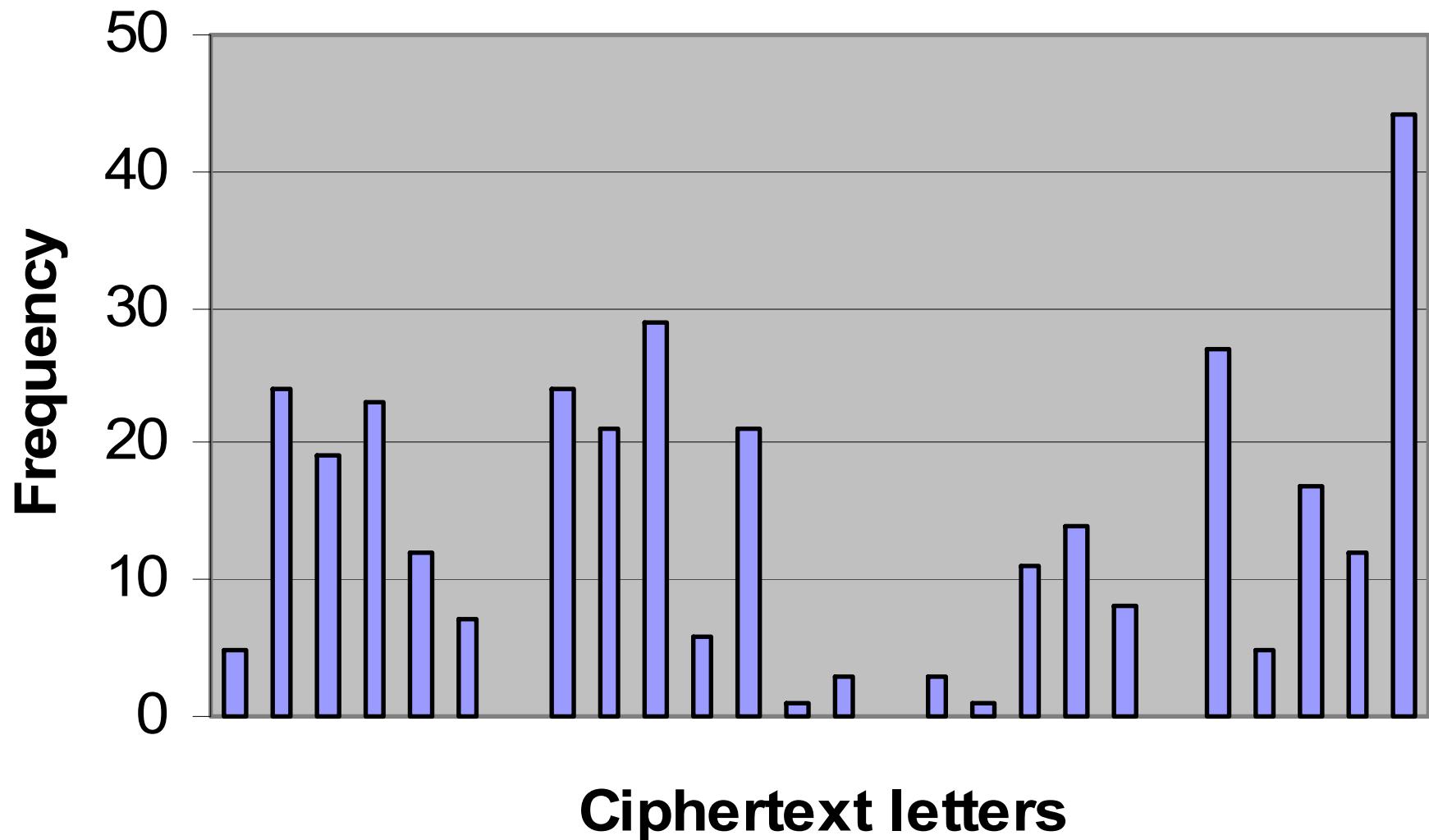
What does this tell us?

- A cipher system should not allow statistical properties of the plaintext language to pass to the ciphertext.
- The ciphertext generated by a “good” cipher system should be statistically indistinguishable from random text.

Outline

- Flattening the ciphertext frequency histogram.
- Polyalphabetic ciphers.
 - Vigenere ciphers.
- Statistical analysis of polyalphabetic ciphers.
 - Kasiski method.
 - Index of coincidence.

Ciphertext distribution



Flattening the histogram

- To avoid the basic statistical analysis of identifying letters by frequency, we want to try and flatten the histogram. That is, we want the elements of ciphertext to all occur with similar frequency.
- We are going to look at two ways of doing this.

Homophonic substitution ciphers

- A plaintext characters is mapped into a set of ciphertext characters, called *homophones*. To encipher a plaintext character, one of it's homophones is randomly chosen.

a	86	42	69	51
g	2	59	75	
l	56	23	0	4
n	24	3	98	7
o	13	9	5	
w	12	99		

Letters: {a,b,...z}

Homophones: {0,1,...99}

wollongong → 12, 9, 23, 0, 5, 98, 75, 13, 3, 2

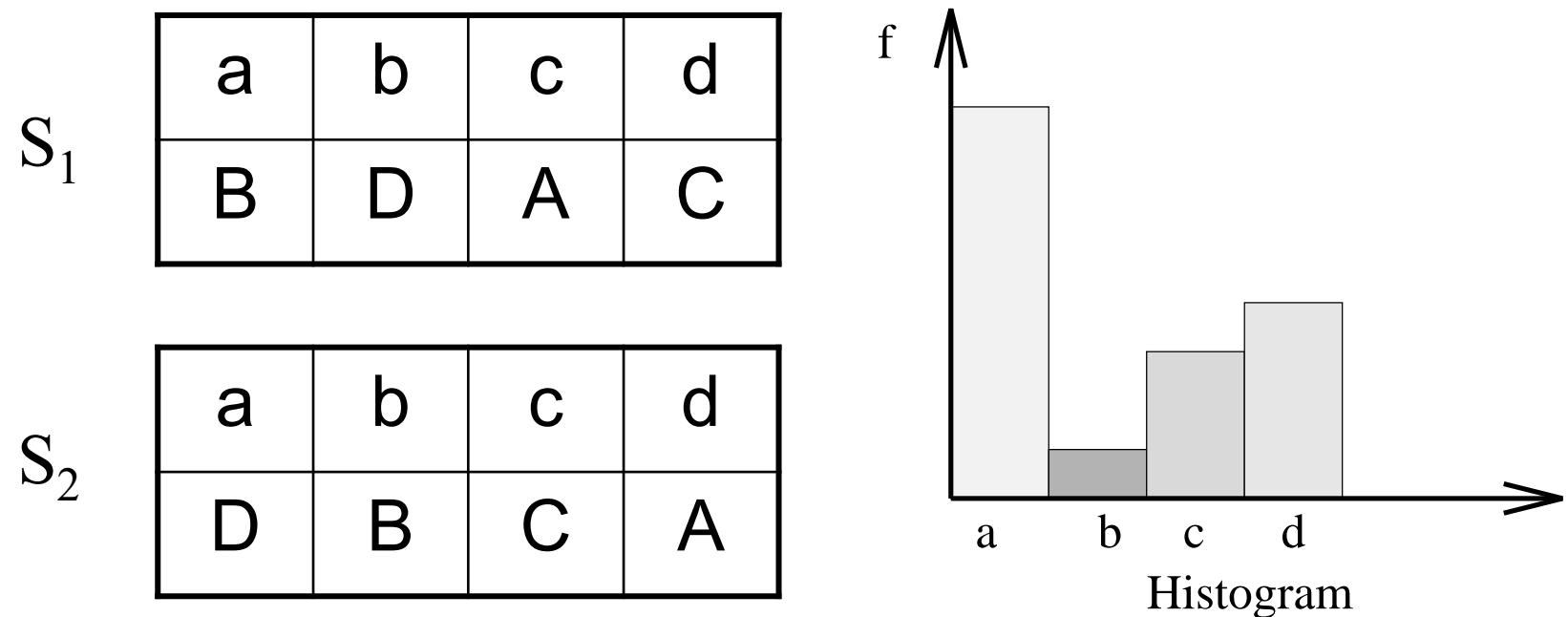
- Homophonic ciphers in which the number of homophones are proportional to the letter frequencies are hard to break by observing homophone frequency.
 - For example, the letter **e** should have more homophones.
- For large enough text, statistical analysis is still going to be relatively simple.
- A significant drawback of this method is that the cryptogram is longer than the message.
 - For the example on the previous page we lengthen a 50 bit (10^5) plaintext into a 70 bit (10^7) ciphertext.

Use more than one substitution alphabet

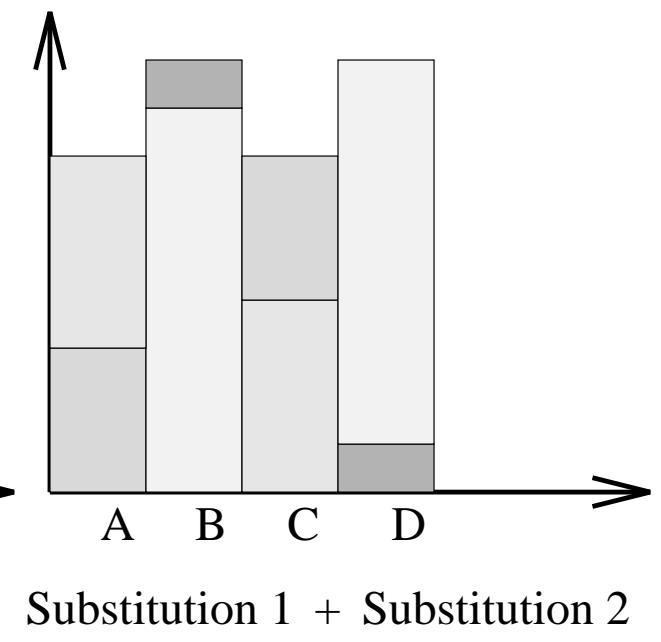
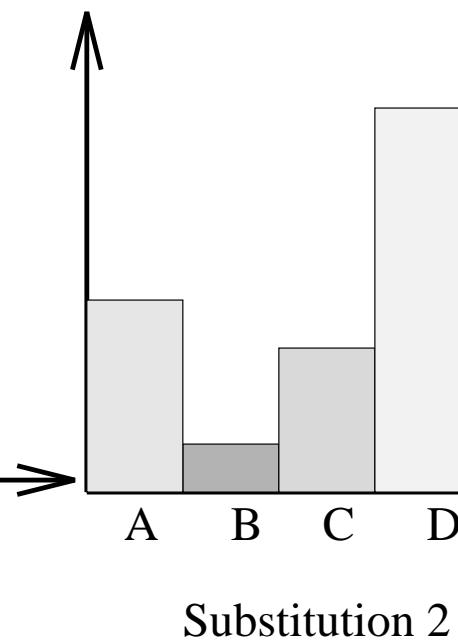
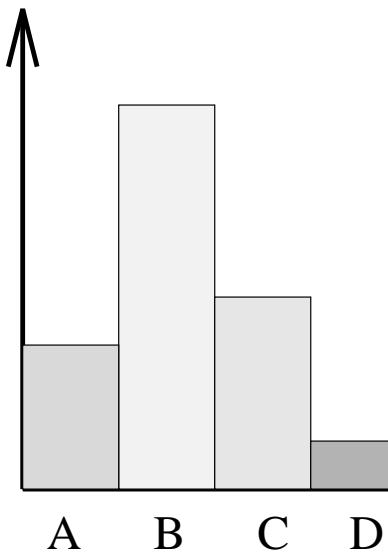
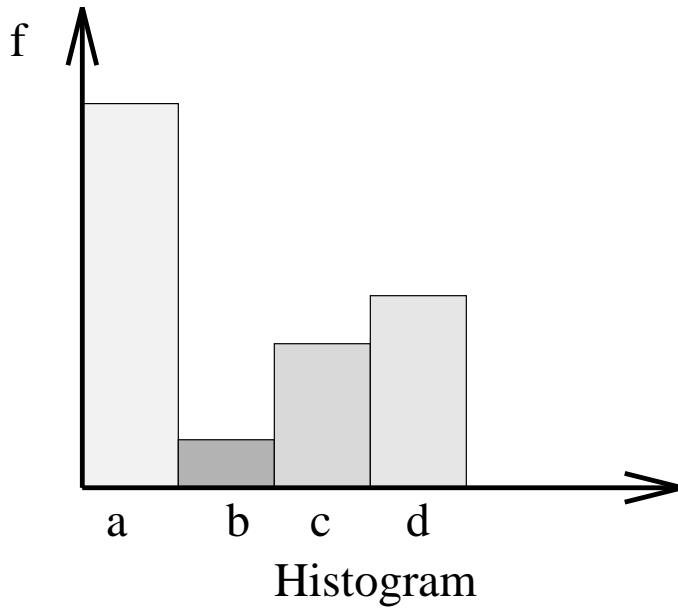
- By combining substitution alphabets, each of which produces non-flat frequency distributions for the ciphertext, we can compensate by aligning the alphabets so the effective combination has a flat frequency distribution.
- We shall consider an example using two substitution alphabets and a key sequence that determinates which substitution alphabet must be used for encrypting each plaintext letter.

■ Plaintext alphabet: $\{a,b,c,d\}$

$$P(a)=0.5, P(b)=0.05, P(c)=0.2, P(d)=0.25$$



X	a	b	a	c	a	d	a	d	c
Z	1	2	2	1	2	1	2	2	2
Y	B	B	D	A	D	C	D	A	C



Polyalphabetic ciphers

- A ciphertext character represents more than one plaintext character. This must be done in a way that allows the plaintext to be recovered.
 - For example, if **B** represents **n** and **t** we need to know when to decipher it as **n**, and when as **t**.
- A polyalphabetic cipher uses a sequence of substitution alphabets. If this sequence repeats after p characters, we say it has *period p*.

The Vigenère cipher

- This uses 26 substitution alphabets, and a *key word* or *key phrase*.
- The substitution alphabets are cyclically related, they form a Trithemius tableau.

a	b	c	d	e	...	x	y	z
0	1	2	3	4	...	23	24	25

	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
a	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
b	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	A
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
e	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
k	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
l	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
p	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
r	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
z	Z	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

X	w	o	I	I	o	n	g	o	n	g
	22	14	11	11	14	13	6	14	13	6
Z	c	a	f	e	c	a	f	e	c	a
	2	0	5	4	2	0	5	4	2	0
Y	24	14	16	15	16	13	11	18	15	6
	y	o	Q	P	Q	N	L	S	P	G

$$Y = X \oplus Z, \text{ remember addition modulo 26.}$$

Analysing Vigenère ciphers

- As we stated earlier, the frequency distribution of the ciphertext can be flattened by using multiple substitution alphabets.
- We analyse these ciphers by
 1. Finding the period.
 2. Breaking the ciphertext into components each obtained from a single substitution alphabet.
 3. Solving each component using techniques discussed for monoalphabetic ciphers, although cross component techniques are also useful.

Finding the period: The Kasiski method

- We observe that two identical plaintexts will be encrypted to the same ciphertext if their occurrence is m positions apart, where $m \equiv 0 \pmod p$, i.e. when m is a multiple of the period p .

X	w	o	I	I	w	o	I	I
Z	c	a	f	e	c	a	f	e
y	y	O	Q	P	y	O	Q	P

- We search the ciphertext for repeated segments, and measure the distances between such repeated segments. It is *likely*, but not guaranteed, that these distances will be a multiple of the period.

Finding the period: The index of coincidence.

- Consider a string of length n , where each element is a letter from the English alphabet. $X=x_1x_2\dots x_n$
 $\lambda \in \{A,B,C,\dots,Z\}$, and f_λ frequency of λ

$$IC(x) = \frac{\sum_{\lambda=A}^Z f_\lambda(f_\lambda - 1)}{n(n-1)}$$

- $IC(x)$ is an estimate of the probability that two randomly chosen elements of X are identical.
- It is also a *measure of roughness* of the histogram, that is, it indicates how uneven the histogram is.
- The Index of Coincidence can be used to estimate the period (Friedman or Kappa test).

$$\kappa = \frac{0.027n}{IC(n-1) - 0.038n + 0.065}$$

Random IC and English IC

- If X were a random string over the English alphabet we would expect the probability of each letter occurring to be the same, so that
$$IC \approx 1/26 \approx 0.038$$
- If X is an English language text then we would expect, for $p(\lambda)$ the probability of a particular letter being λ ,

$$IC(x) \approx \sum_{\lambda=A}^Z p(\lambda)^2 \approx 0.065$$

- The two values 0.065 and 0.038 are sufficiently far apart that we will often be able to determine the correct keyword length.

English language letter probabilities

A	B	C	D	E	F	G	H	I
0.082	0.015	0.028	0.043	0.127	0.022	0.020	0.061	0.070
J	K	L	M	N	O	P	Q	R
0.002	0.008	0.040	0.024	0.067	0.075	0.019	0.001	0.060
S	T	U	V	W	X	Y	Z	
0.063	0.091	0.028	0.010	0.023	0.001	0.020	0.001	

IC for monoalphabetic ciphers

- For monoalphabetic ciphers the index of coincidence is the same for the plaintext as for the ciphertext.
- The IC for the ciphertext of a English text plaintext encrypted under a monoalphabetic cipher, will therefore be approximately 0.065, the same as English text.
- We can use this test to find the period.

Example

- Suppose we know the following ciphertext was generated using the Vigenère cipher. We need to find the period.

ooon yho cshexjlg nz xfksledcl ky luw h dltqupduw jjhrolww
jshehj dwns df llwpslpchlodf plzdv yohrh gm audwwyg uyg
vvqnzuss zyghd wfirustg qp djl hbp psqcl augmsmdlguof
pgmjontrf jshehj dwnslf zihj zaav u qxds fuyjw vt jcrxlgmtrfhz
xtvupdftqwz whnomkwhr vgts ftnw soq aksyaunb zlofek
jlzuehv wfiqhkwiyv loon luw bbcbxw ac mfqq ds uch ssgi
ekw solrhka iohhjnfuoxsas wpqlif qtwz avy hlvlgn cdfns iq
sjvullpk hbx hllowh zxj usqwb xvfgpg ...

- To find the period we guess and check the IC of the components. Lets try $p=2$.

oo yo sejg z fsec k lw dtudw jrlw sej ws f lplcld pzw or g adwg y vqzs zgd
frsg p j hp sc agsdgo pmotf sej wsf ij av qd fyw t cxgtfz tudtw wnmwr gs
tw o asan zoe jzev fqkwy lo lw bbw c fq s c sg ew ork iojfoss plf tz v hvg
cfs q julk b hlw zj sw xfp uzpw t ck b dabp oh ew flwh zwhl l cqj rqfb
fvfcvop vqeg glfb ew ilawd zcr ls zaz nwqd g v aqwl sr tdund qpub sl g
yaopq wvv c s shg ybclc z fk yosr sr y ...

on h chxl n xkldl y u h lqpu jhow jhh dn d lwsphof ld yhh m uwg ug vnus
yh wiut q dl b pql ummluf gjnr jhh dnl zh za u xs uj v jrlmrh xvpfqz hokh
vt fn sq kyub lfk luh wihziv on u bcx a mq d uh si k slha hhnuxa wql qw
ay lln dn i svlp hx loh x uqb vgg vfj v uw hx flwv pb k nywz jwuco v zh h
ylnpa qladbn hbulu jqpa k ogqay wyok o mfh mluy w ay kzwo u vrvcd
zcql nd p cwtuo shh a nh jch lydph i l ersxh u u ...

Component 1: IC = .047082

Component 2: IC = .050946

p=2	.047028	.050946					
p=3	.050229	.054603	.056575				
p=4	.047511	.051445	.047226	.051843			
p=5	.045771	.045228	.048290	.045118	.043533		
p=6	.070324	.068399	.065509	.066987	.063925	.070450	
p=7	.042861	.045826	.045078	.046161	.047380	.047236	.047230

- Therefore $p=6$ seems most likely.
- The keyword is ***should***.

Notes on statistical methods

- Statistical methods work if “enough” ciphertext is available. We will discuss later what we mean by enough.
- Using long keys makes both the IC and Kasiski methods of less value.

CSCI361

Computer Security

Secret-key cryptography

Outline

- One-time-pad.
- Perfect secrecy.
- Unicity distance.
- Redundancy.
 - Rates.
- Measuring and increasing security.
- Unconditional versus practical security.
- Improving security.
- Block and stream ciphers (classical).

One-time-pad (Vernam cipher)

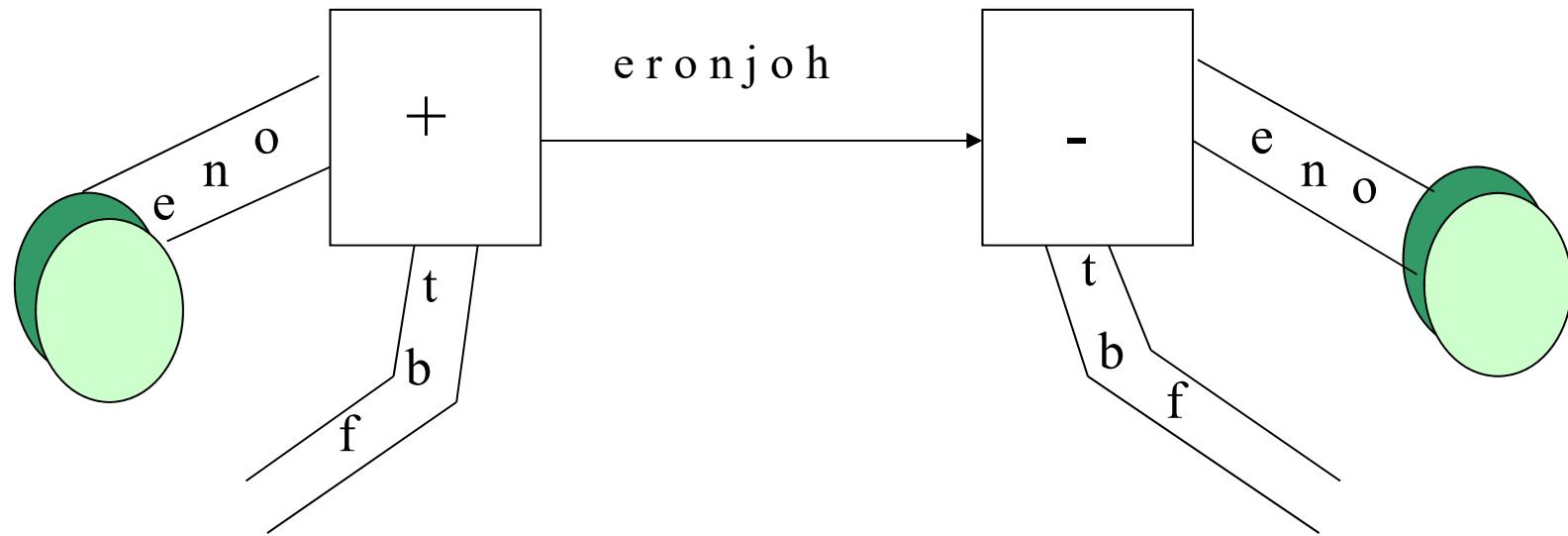
- Gilbert Vernam (1917/8) invented a cipher that was extended by Joseph Mauborgne to give a scheme which was later (Shannon 1949) proved to provide *perfect secrecy*.
 - We shall discuss later what *perfect secrecy* is.
- The cipher is called one-time-pad because the key is written on a long tape and is used only once.
- Encryption is by addition.
- To decrypt a cryptogram, the same key sequence must be used. Decryption is by subtracting the key sequence from the ciphertext.
- The key sequence is a truly random sequence, this was Mauborgne's contribution.

- In a one-time pad the encryption of each element of plaintext is independent of the encryption on any other piece of plaintext.
- Encryption:

$$Y_i = (X_i + K_i) \bmod 26 \quad \text{Letter by letter.}$$

$$Y_i = X_i \oplus K_i \quad \text{Bit by bit.}$$

- Decryption:
- $X_i = (Y_i + K_i) \bmod 26 \quad \text{Letter by letter.}$
- $X_i = Y_i \oplus K_i \quad \text{Bit by Bit}$



Message : ONETIMEPAD

Key on pad: TBFRGFARFM

Ciphertext: HOJNOREGFP

$$O + T \bmod 26 = H$$

$$N + B \bmod 26 = O$$

$$E + F \bmod 26 = J \dots$$

Perfect secrecy (Shannon 1949)

- Model: Transmitter, receiver, enemy.
- Attack: *Ciphertext only*.
- Assumption: *Enemy has unlimited power!*

- **Question 1:** Is it always possible to find the plaintext (or key)?
- **Question 2:** If it is possible, how can we measure the security?

- To find the plaintext in substitution ciphers we may use exhaustive key searches.
- For short cryptograms though, we may find multiple keys give *meaningful plaintexts*.

A	Z	N	P	T	F	Z	H	L	K	Z
m	y	s	t	e	r	y	p	l	a	y
r	e	d	b	l	u	e	c	a	k	e

- For shorter ciphertext more valid plaintext can be found.

H	J	M	T	T
y	a	h	o	o
a	t	r	e	e
m	e	t	o	o
i	w	i	l	l

- If the length of the ciphertext is 1, every decipherment is valid.
- For length around 50, a *unique* plaintext is likely to be found.

Perfect secrecy

- Using the knowledge of the plaintext languages a set of possible plaintexts are determined with certain probability.
- In a system with perfect secrecy, knowledge of the cryptogram does not help the enemy.

$$P(X=x) = P(X=x|Y=y), \quad \forall x,y$$

They are just as likely to guess the plaintext associated with a ciphertext **after** they see the ciphertext as they are **before** they see it.

Theorem (Shannon)

In a system with perfect secrecy the number of keys is at least equal to the number of messages.

- This tells us that to achieve perfect secrecy in practice, many key bits must be exchanged. This is not practical.
- How can we measure security then if we know it probably isn't perfect?
- Shannon proposed *unicity distance* as the measure of security.

Unicity distance

- N_0 is the least number of ciphertext characters needed to determine the key uniquely. If there are E keys and they are chosen with uniform probability, unicity distance is given by:

$$N_0 = \frac{\log_2 E}{d}$$

where d is the *redundancy* of the plaintext language.

Redundancy and rates

- Redundancy of a language is defined in terms of the *rates* of the language.
 $d=R-r$ bits
- The **absolute rate R** of a language is the minimum number of bits to represent each character, assuming characters are equally likely and emitted independently. For an alphabet of size A, $R=\log_2 A$.
- The **true rate r** of a language is the average number of bits required to represent characters of that language. This uses the real probability distribution of characters.
- For English; $R \approx 4.7$ bits, $r \approx 1-1.5$ bits, $d \approx 3.2$ bits.
- True rate is always smaller than absolute rate, and the difference is the redundancy.

- All natural languages are redundant:

mst ids cn b xprsd n fwr ltrs,
bt th xprnc s mst nplsnt.

- This sentence is readable because we can fill all the missing characters: that is, all the missing characters are redundant.

- Redundancy is related to structure.
- A truly random source has no redundancy.
**mmhf
fsd
acxnv
fdvvdf
pnfui
pawedka**
- Every character in this string is necessary: if one of them is omitted the information it carries is lost and cannot be recovered.
- Redundancy occurs because of the non-uniform letter frequencies, bigram (trigram ...) frequencies and other (e.g. grammatical) structures of the language.

Measuring security

- We can use unicity distance as a measure to compare the security of various ciphers.
- Recall that we presented security as being about protecting *assets* against possible *threats*.
- Recall also the security principles we described, in particular the *Principle of Adequate Protection*, or the concept of *timeliness*.

- For monoalphabetic ciphers we have:

$$E=26! \quad P(K)=1/E$$

$$\log_2 E \approx 88.4 \text{ bits}$$

$$N_0 \approx 88.4/3.7 \approx 23.9 \text{ characters.}$$

Therefore cryptograms (of English text plaintext) 24 or more letters can be deciphered (in general).

- For the one-time-pad:

X = Message space is the set of English text of length k .

Z = The key space which is the set of sequences of length k over the English alphabet.

Keys are chosen randomly with uniform distribution:

$$E=26^k, \log_2 E \approx 4.7k \text{ so that}$$

$$N_0 \approx (4.7)/(3.7)k \approx 1.27k$$

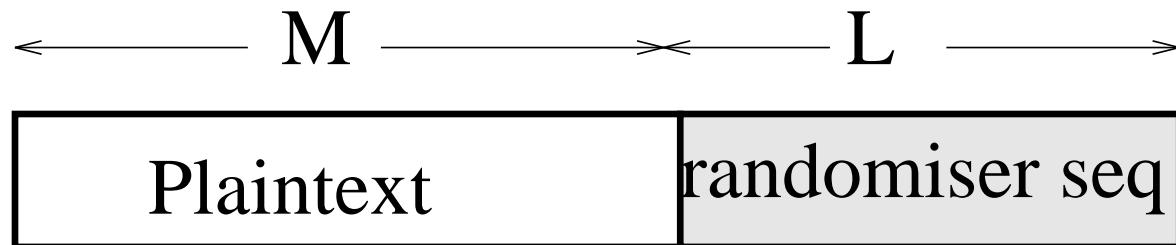
Therefore, a *unique* solution cannot be obtained even if all the characters of the ciphertext are intercepted.

Increasing security

- Studying perfect security suggests ways of increasing security of a cipher systems.
- We could *increase the size of the key space.*
 - The increase should be such that for a ciphertext Y, using a randomly chosen key Z for decryption, a random message X should be generated.

- We could *reduce the redundancy of the plaintext language.*
 - This can be achieved using data compression.
 - A perfect data compression algorithm will result in $d=0$ and hence $N_0 \rightarrow \infty$, i.e. makes the cipher unbreakable.

- We also could *use a randomiser to reduce the redundancy of the plaintext language.*



- The new redundancy is $M/(L+M)d$.

Unconditional versus practical security

- A system is *unconditionally secure* if it is secure against an enemy with unlimited resources (time, memory).
- In an unconditionally secure system, protection is provided in an *information theoretic* sense. That is, the reason that the enemy cannot break the cipher is that they do not have enough information.
- In practice however:
 - Enemies have limited resources.
 - Data has a limited life time anyway, *timeliness*.
 - The enemy may be able to access other types of attack:
 - Social attacks aimed at getting passwords.
 - Plaintext/ciphertext pair attacks.
- The one-time-pad is the simplest only encryption system with unconditional security, but it requires many key bits and security is under ciphertext only attack.

Practical or computational security

- Computational or practical security attempts to provide a more realistic model for a secure system than unconditional security.
- After the unicity distance is reached, there is a unique plaintext for a given ciphertext.
- In practical security the main objective of the cryptographer is to make sure the amount of work required to find that plaintext stays high even after the unicity distance.

A cipher is computationally secure if the difficulty of the best attack exceeds the computational capability of the cryptanalyst (attacker!).

Principles suggested by Shannon's work

- In the design of a good algorithm we must make methods of statistical analysis ineffective:
- **Diffusion:** Diffuse the statistical structure of the plaintext into long range statistics : statistics involved long combinations of letters.
- **Confusion:** The relationship between plaintext, ciphertext and key must be complex one so that knowledge of plaintext/ciphertext pairs cannot easily be used to find the key. In other words, $Y=E(X,Z)$ should be a *nonlinear* and complex function.

Diffusion

- For example: for $X=X_1X_2\dots$ we could use the relation

$$Y_n = E\left(\sum_{i=1}^s X_{n+i}\right)$$

- This diffuses by averaging s consecutive letters.
- For $s=2$, English alphabet, and additive cipher with shift 5 we have:

t	e	s	t	c	a	s	e
19	4	18	19	2	0	18	4
23	22	11	21	2	18	22	23
2	1	16	0	7	23	1	2
C	B	Q	A	H	X	B	C

The avalanche effect

- A desirable property for encryption functions is that small changes in either plaintext or key should result in significant changes in the ciphertext.
- The avalanche effect is in place if this is the case.
- For example, a cipher might be said to have the avalanche effect if changing a single bit (in either the key or plaintext) results in half the output being different.

Confusion

- A simple additive cipher is linear.

$$Y = X \oplus Z$$

- We could use a more complex relationship. In modern ciphers the complex relationship is obtained by combining simple elements.
 - We will start to look at this soon.

Some numbers

- It is useful to get an idea about how complex the relationship needs to be. To do this we need to look at the amount of work that we must produce for a cryptographic algorithm to be secure. Consider this:
 - 10^{24} operations with 280.6×10^{12} operations per second (IBM Blue Gene/L October 2005) would take 113 years.
 - A system is probably safe enough to be considered secure if this amount of computation is the required “cost” of breaking the system.

- The number of operations or storage required depends on the attack. For a secure algorithm the number remains large even for the best (known) attack.
- Monoalphabetic ciphers are insecure because:
 - Although there are many keys, statistical methods allows many keys to be quickly eliminated.

Block and stream ciphers

- All the early ciphers we have studied so far are examples of *stream ciphers*.
- In stream ciphers characters are encrypted one at a time.
- A second class of ciphers are *block ciphers*. In block ciphers plaintext characters are grouped in blocks and then each block is encrypted.
- This provides **diffusion** across the block.

Playfair cipher (1850)

- The key is specified by a 5 by 5 matrix of 25 letters, with J omitted.

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

- A pair of plaintext X_1X_2 is encrypted to Y_1Y_2 according to the following rules:
 - X_1X_2 in the same row: Y_1 and Y_2 are the two characters to the right of X_1 and X_2 (cyclic).
 - X_1X_2 in the same column: Y_1 and Y_2 are the two characters below X_1 and X_2 (cyclic).
 - $X_1=X_2$: A separating character (anything, X, Z ...) is inserted in the plaintext between the two. This is really a pre-encryption phase.
 - X_1X_2 different rows & columns: Y_1 and Y_2 are the other two corners of the rectangle with X_1 and X_2 as corners. The direction we shall take for the example is anticlockwise from the first element to the second but this is arbitrary although we must be consistent.

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

TH	IS	IS	AT	RI	AL
PM	BH	BH	NP	HO	FS

Transposition: Columnar

- Plaintext is written into a matrix by rows. The ciphertext is obtained by reading off the columns.

c	o	m	p	u
t	e	r	s	e
c	u	r	i	t
y	x	x	x	x

- plaintext : Computer Security
- ciphertext : CTCYOEUXMRRXPSIXUETX

Transposition: Periodic

- Break plaintext into blocks of size d and permutes characters in a block.

i	1	2	3	4	5
F(i)	5	1	4	3	2

compu	terse	curit	y
UCPMO	ETSRE	TCIRU	Y

- The relative frequency of ciphertext characters is the same as the plaintext.
- The cipher is breakable by *anagramming* - the process of restoring disarranged set of letters using the frequency of bigrams and trigrams.

CSCI361

Computer Security

Modern Secret-key cryptography I

Outline

- Block ciphers.
 - Design principles.
 - Iterated ciphers.
- Data Encryption Standard (DES) development.
- DES.
 - Structure.
 - Algorithms.
 - S-Boxes.
- Feistel ciphers/networks/structure.

Modern block ciphers

- In a block cipher algorithm, plaintext bits are grouped into blocks and then processed.
- We have already seen a classical block ciphers (the Playfair cipher), as well as the transposition building block.
- We are going to begin looking at a more formal presentation and analysis of block ciphers.
- Two important parameters of such algorithms are *block length* and *key size*.

Key	000	001	010	011	100	101	110	111
0	001	111	110	000	100	010	101	011
1	001	110	111	100	011	010	000	101
2	001	000	100	101	110	111	010	011
3	100	101	110	111	000	001	010	011
4	101	110	100	010	011	001	011	111

- **The block length is 3.** The cipher takes an arbitrary binary string of length 3, and uses the specified keys to produce 3 bits of output.
- We need **3 bits** to specify the key (1 of 5).
- The number of possible substitutions on an alphabet of size 8 is $8!$ (40320, about 16 bits). By using a subset of all possible substitutions we have reduced the size of the key.

- One important way of thinking about, or representing, block ciphers is the following:

A block cipher algorithm, with block size b bits, describes a mapping to an indexed subset of the set of all possible substitutions on an alphabet of size 2^b .

- The idea is to have an efficient way of generating a “nice” subset.

- We already know that for a secure cipher the indexed set should look random.
- Looking at the example we had before we see that if the enemy receives 001, they will know that the plaintext is either 000 or 101. This isn't very good.

Key	000	001	010	011	100	101	110	111
0	001	111	110	000	100	010	101	011
1	001	110	111	100	011	010	000	101
2	001	000	100	101	110	111	010	011
3	100	101	110	111	000	001	010	011
4	101	110	100	010	011	001	011	111

A secure block cipher...

- ... needs a **block length** large enough to deter statistical attacks.
 - Length 64 is usually considered large enough to make frequency analysis infeasible.
 - Generally, the larger the block length the longer the processing time. Furthermore the longer you have to wait before you can begin processing.
- ... needs a **key space** large enough to make exhaustive key searches infeasible.
 - On the other hand, keys should be short enough so that key generation, distribution and storage can be efficiently managed.

Design principles for block ciphers

1. Security principles.

- Confusion.
- Diffusion.

2. Implementation principles.

Two implementation methods are available.

- Software – flexible, low cost.
- Hardware (VLSI) – high speed.

(Very Large Scale Integration)

Design principles for software

- Cipher operations should be on a sub-block size of length *natural* to the software, generally 16 or 32 bits.
- Operations should be simple and easy to implement. They should use basic instructions of the standard processors, that is addition, multiplication, shifting ...
 - Bitwise permutations, for example, are hard to implement.

Design principles for hardware

- Encryption and decryption should be similar.
 - Ideally they should differ only in the way of using the secret key.
- The cipher should have a regular modular structure to facilitate VLSI implementation.

Iterated ciphers

- A block cipher is called an *iterated cipher* if it is based on a simple function f , repeated (or iterated) several times.
- Each iteration is called a *round*. The output of each round is a function of the output of the previous round and a sub-key derived from the full secret key by a key-scheduling algorithm.
- Decryption is performed by using the inverse of the round functions and the sub-keys in reverse order.
- A particularly interesting class of functions in this context are the involution functions. These functions are self-invertible. Employing such functions allows encryption and decryption modules to be the same, but keyed in the reverse order.

Involution example

- Consider that a function f acts on binary blocks of length 3.

$$f = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{bmatrix}$$

- $f(101)=011$, $f(f(101))=101$
- $f^{-1}=f$, $f(f(x))=x$
- Another example is $f(x) = x \oplus 11$, for x a binary block of length 2.

Towards DES: Preliminaries

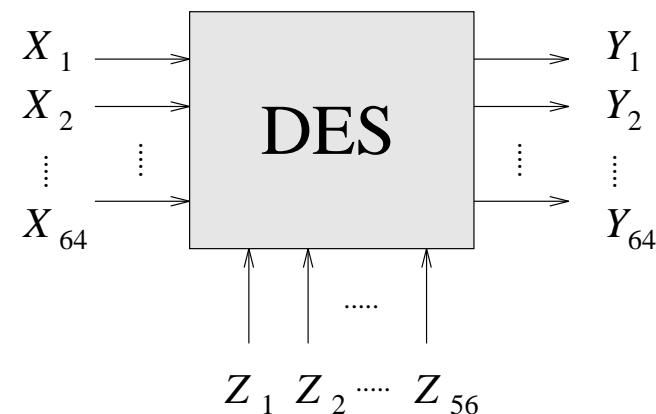
- It became clear in the 1970's that there was a need for a standard encryption algorithm. There were several reasons for this:
 - Advances in information technology and the need for security to support this. Government and Commercial parties were beginning to use computers extensively.
 - Required assurance could not be provided by ad-hoc algorithms.
 - Different devices had to be able to exchange encrypted information.

- The standard needed to have the following properties:
 - It needed a high level of security.
 - It needed to be completely specified. In accordance with Kirchoff's Law the security shouldn't rely on a hidden part of the algorithm.
 - It needed to be economical to implement.
 - It needed to be adaptable to diverse application.
- In 1973 National Bureau of Standards published a solicitation for cryptosystems in the Federal Register. This lead ultimately to the development of the ***Data Encryption Standard***, or ***DES***.
- DES was developed at IBM based on an earlier cipher system known as Lucifer.

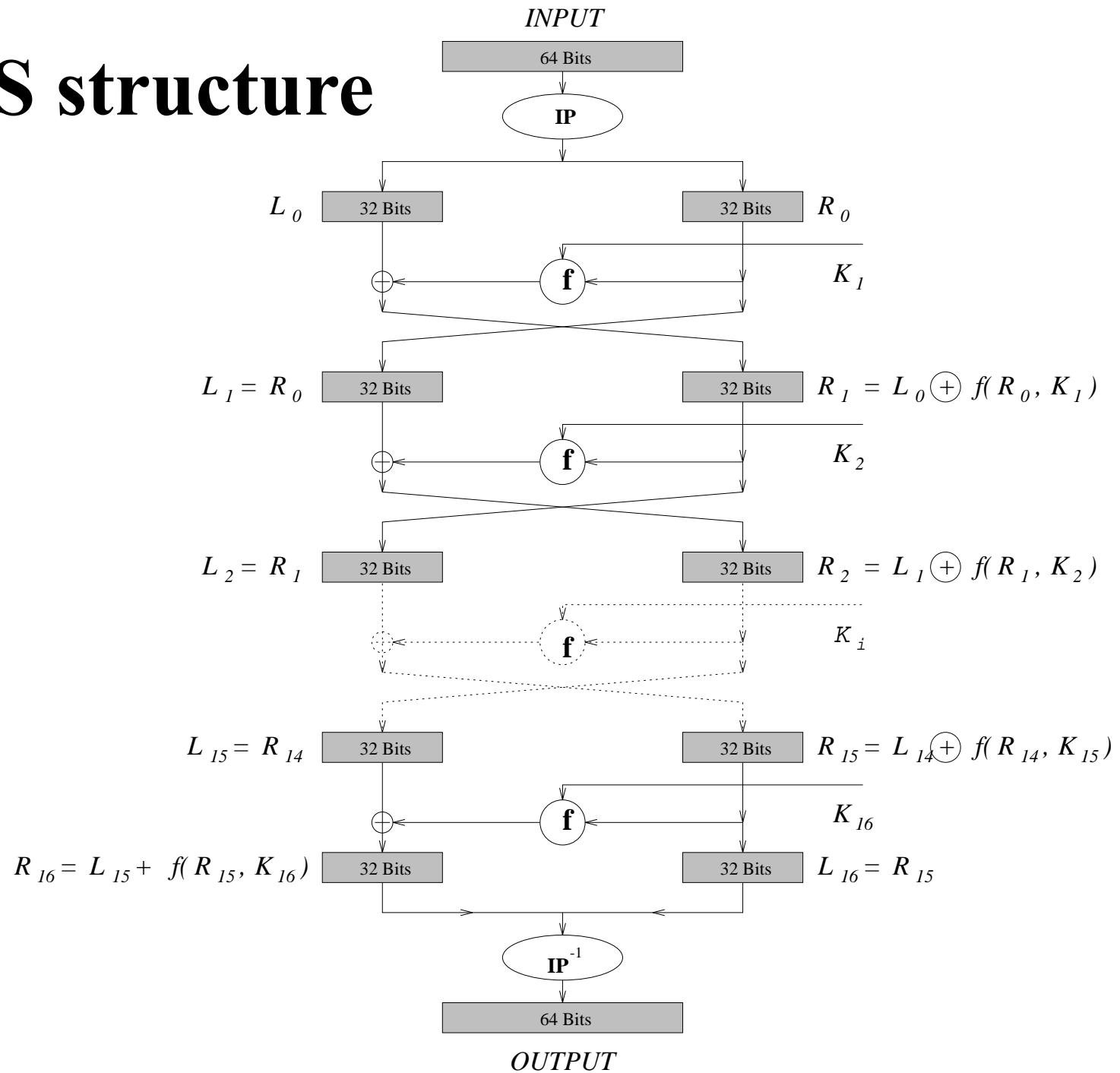
Data Encryption Standard (DES)

- DES (1977) is one of the most widely used, if not the most widely, and perhaps the most controversial cipher.
- It is a block cipher.
- It takes an input block of 64 bits and maps it into an output block of 64 bits using a key of 56 bits.

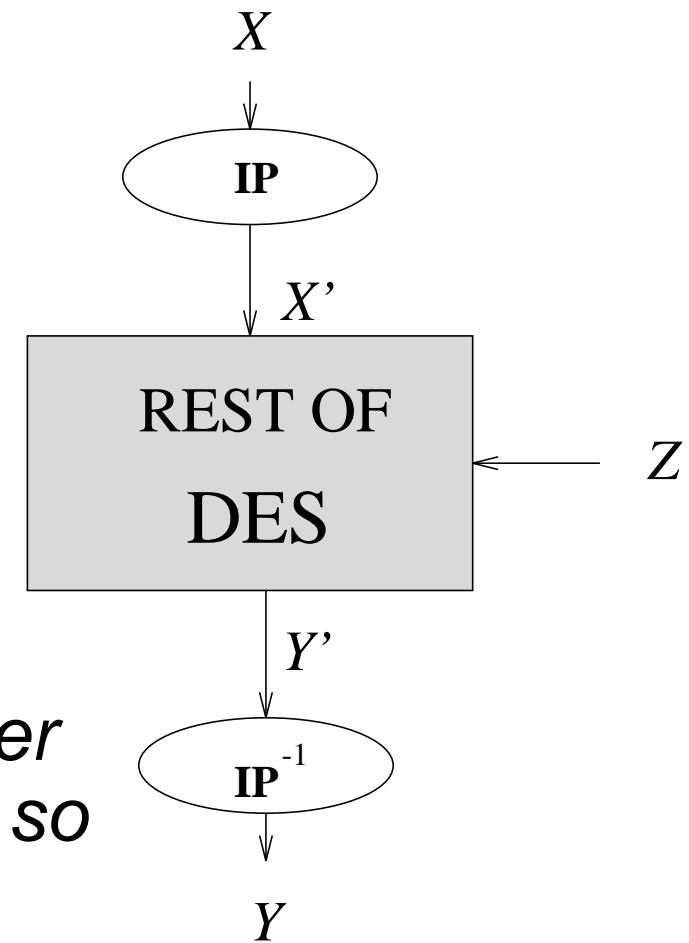
$$|X|=|Y|=2^{64}, \quad |Z|=2^{56}.$$



DES structure



- The function f is a non-linear transformation, and is the source of the cryptographic strength of DES.
- IP is the initial permutation, and has no cryptographic significance.
 - This is because if X and X' uniquely determine one another in a way known to the enemy, so do Y and Y' .*
- IP is used to facilitate getting bits onto the chip in VLSI DES.



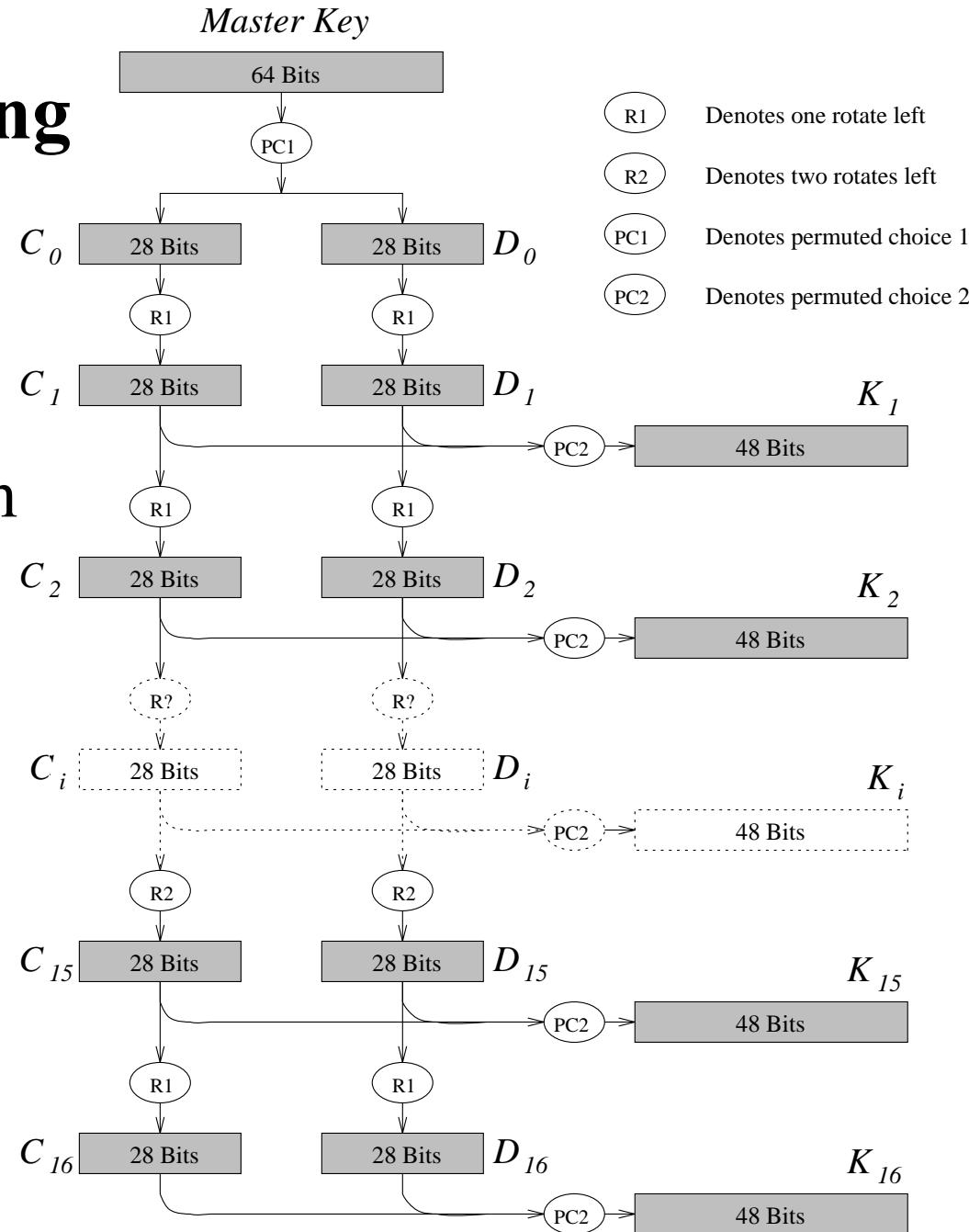
Specification of IP

- L₀ and R₀ are the initial left half and right half of the post-IP data.

L_0	{	58	50	42	34	26	18	10	2	This means: Bit 58 in the original input X becomes bit 1 of IP(X)
		60	52	44	36	28	20	12	4	...
		62	54	46	38	30	22	14	6	
		64	56	48	40	32	24	16	8	
R_0	{	57	49	41	33	25	17	9	1	
		59	51	43	35	27	19	11	3	
		61	53	45	37	29	21	13	5	
		63	55	47	39	31	23	15	7	

Key scheduling

This produces
16 subkeys from
56 bits of key.



56 bits key?

- You might have noticed the key size at the top is 64 bits. What is going on?
- DES expects 64 bits of key, but only uses 56. The remainder are generally used for parity checking.
- 56 bits isn't actually very big as keys go, actually it is too small now and was even a concern at the time.
- The size was chosen with chip implementation in mind.

Permuted choice one

- PC-1 extracts 28 bits for C_0 and 28 for D_0 .

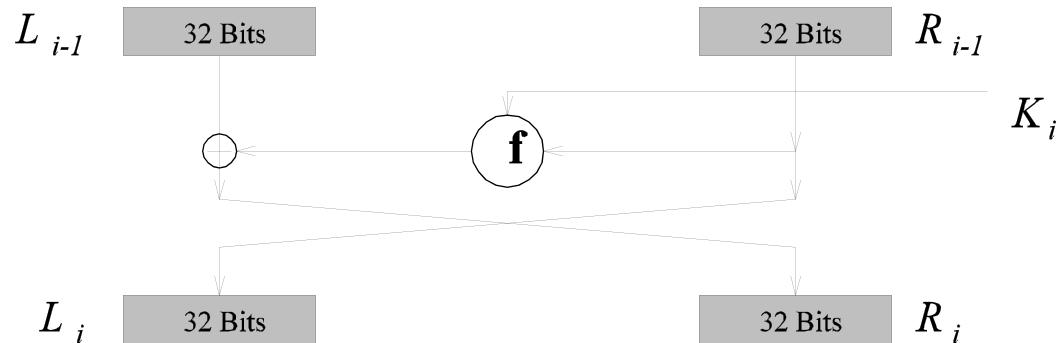
	57	49	41	33	25	17	9
C_0	1	58	50	40	34	26	18
	10	2	59	51	43	35	27
	19	11	3	60	52	44	36
	63	55	47	39	31	33	15
D_0	7	62	54	46	38	30	22
	14	6	61	53	45	37	29
	21	13	5	28	20	12	4

Permuted choice two

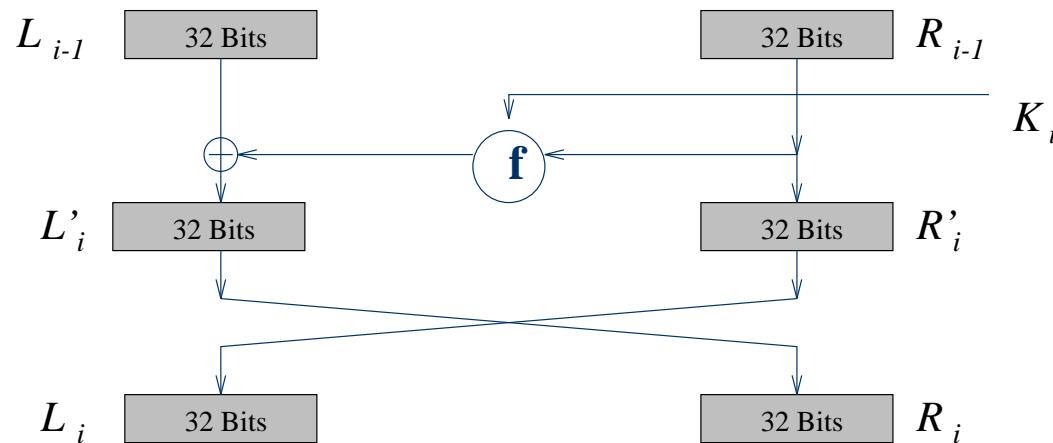
- PC-2 extracts 48 bits for a subkey.

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

The round structure of DES



Why? This can be redrawn as ...



... the Feistel structure!

The Feistel structure

- DES has a Feistel structure. This means...
 - Multiple rounds with ordered sub-keys.
 - The input is split into two parts.
 - The right hand side is transformed by a *round function* f in each round, before being combined with the left hand side using an XOR operation.
 - The left and right hand sides are switched after each round to obtain the input to the next round.
- Feistel (1973) proposed this structure as a means of combining substitution and permutation elements to provide confusion and diffusion (as described by Shannon).

Parameters in Feistel ciphers

- A Feistel cipher or network, that is a cipher with the Feistel structure, has various parameters which can characterise them:
 - Block size.
 - Key size.
 - Number of rounds. A single round of a Feistel structure is not expected to provide enough security, however the cipher should be such that multiple rounds of it provides more security.
 - Subkey generation algorithm.
 - Round function f . It doesn't have to be invertible, since it is only used in one direction.

- Each round of DES is a product cipher whose first component cipher is a substitution cipher $F(\cdot)$, and whose second component cipher is the transposition $T(\cdot)$.
- In the last round T is omitted.
- The substitution cipher is an *involution*, this is characteristic of Feistel networks.

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

$$\begin{aligned} F_i F_i(L_{i-1}, R_{i-1}) &= F_i(L_{i-1} \oplus f(k_i, R_{i-1}), R_{i-1}) \\ &= (L_{i-1} \oplus f(k_i, R_{i-1}) \oplus f(k_i, R_{i-1}), R_{i-1}) \\ &= (L_{i-1}, R_{i-1}) \end{aligned}$$

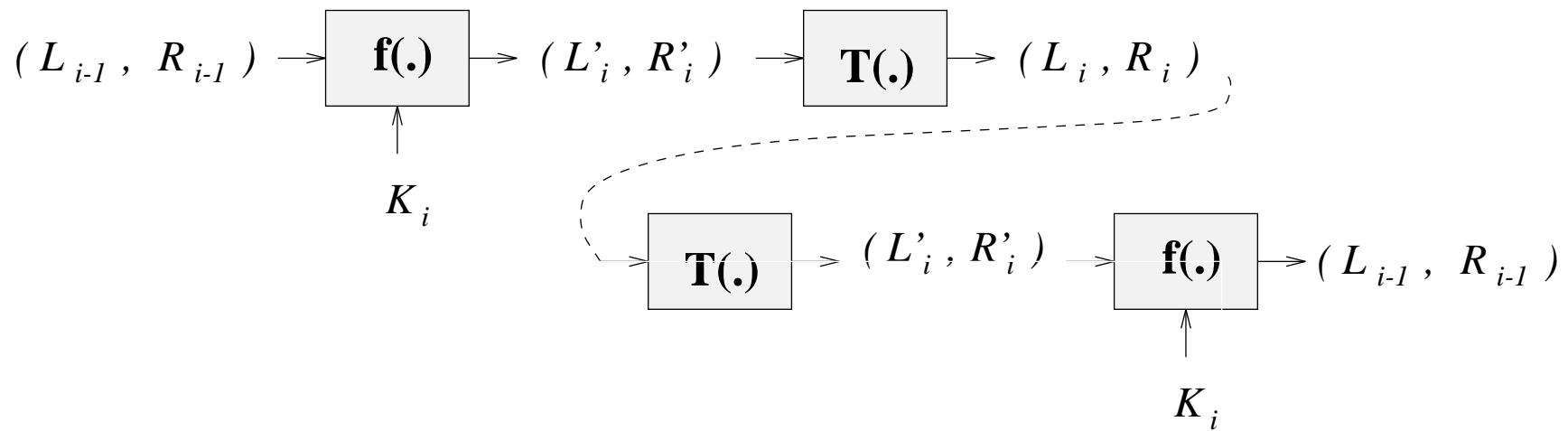
DES Decryption

- The transposition cipher $T(\cdot)$ is an involution too.

$$T(L'_i, R'_i) = (R'_i, L'_i)$$

$$T(T(L'_i, R'_i)) = T(R'_i, L'_i) = (L'_i, R'_i)$$

- To decrypt one round.



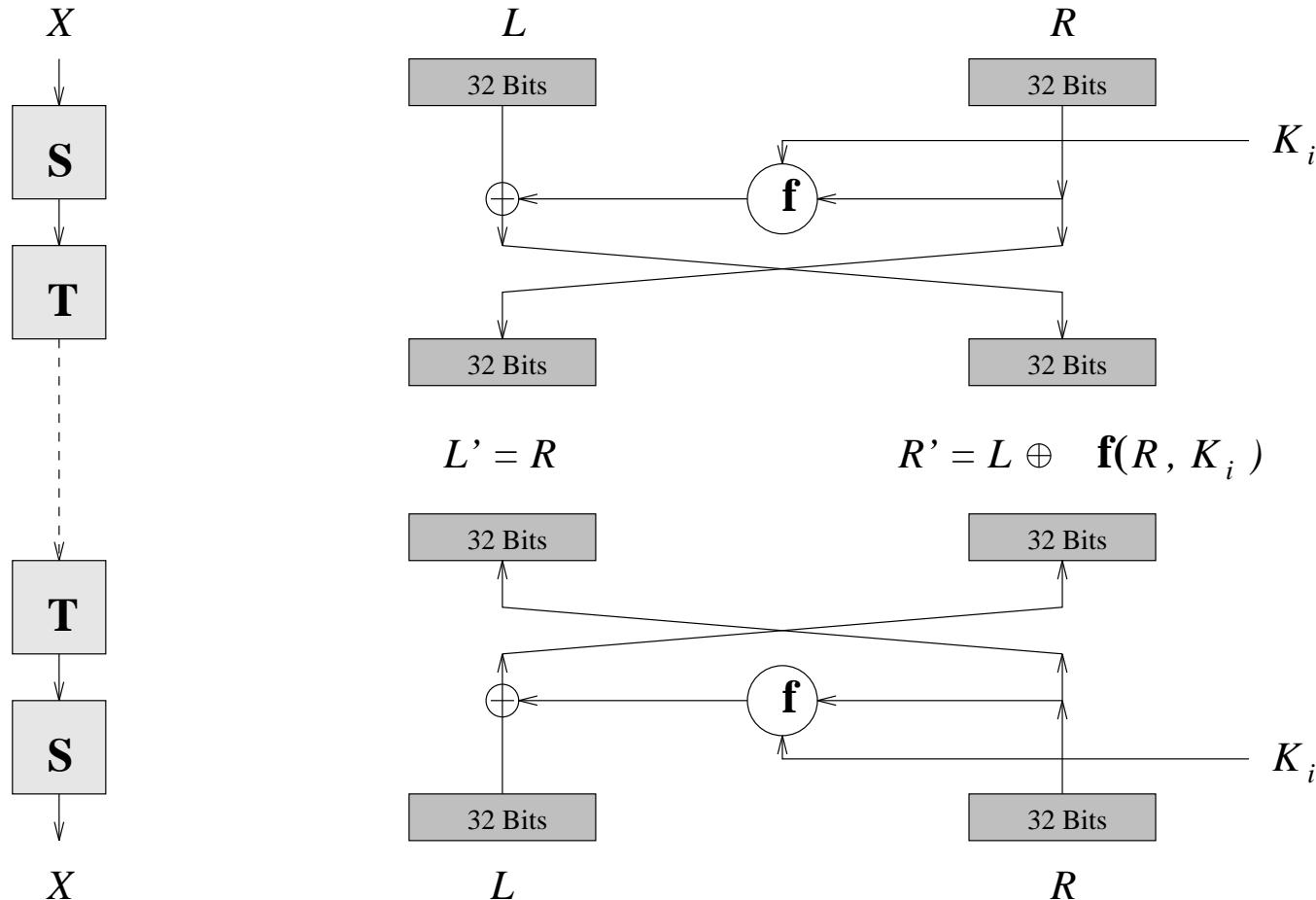
- The full 16 round DES algorithm is a product cipher composed of 16 rounds, each round a substitution and a transposition (except the last).

$$\text{DES} = (\text{IP})^{-1} F_{16} T F_{15} T \dots F_2 T F_1 (\text{IP})$$

- Decryption is similar:

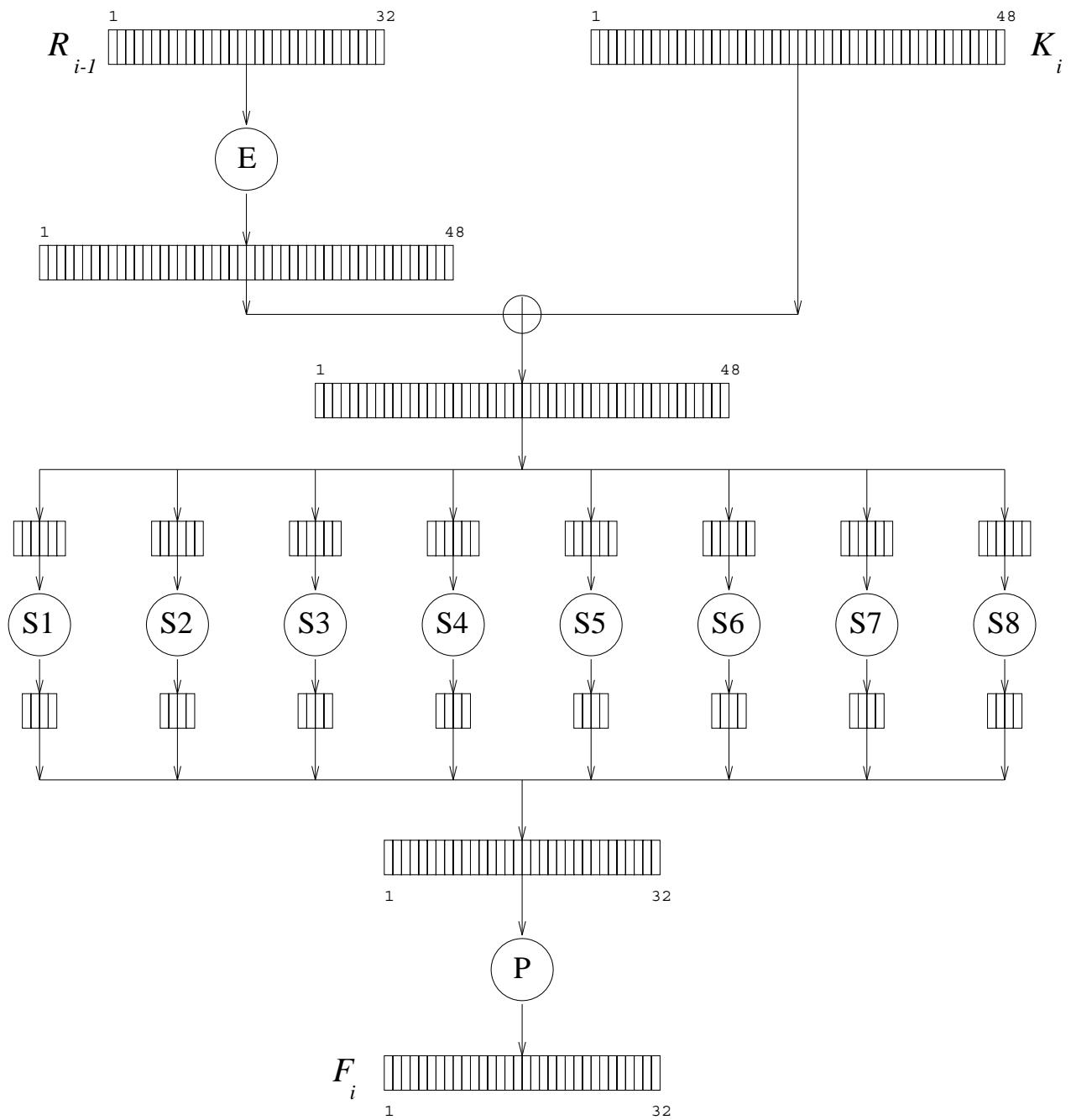
$$\text{DES}^{-1} = (\text{IP})^{-1} F_1 T F_2 T \dots F_{15} T F_{16} (\text{IP})$$

Decryption algorithm



- The decryptor for DES is identical to the encryptor, except that the 16 keys are used in reverse order.

$$f(k_i, R_{i-1})$$



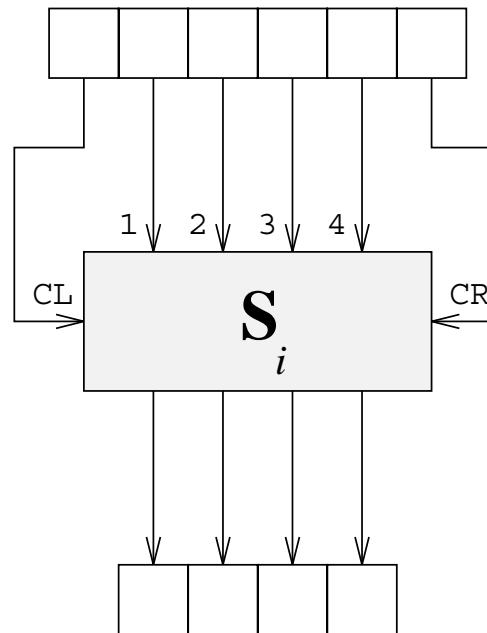
- P is the permutation

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
10	13	30	6	22	11	4	25

- E is the expansion function

32	1	2	3	4	5
8	9	10	11	12	13
16	17	18	19	20	21
24	25	26	27	28	29
4	5	6	7	8	9
12	13	14	15	16	17
20	21	22	23	24	25
28	29	30	31	32	1

Structure of S-Boxes



- CL = left control bit.
- CR= right control bit.
- The CL and CR select one row of the S-Box S_i to use.

- For each of the 4 choices of (CL,CR), S_i performs a different substitution on the 16 possible values of the 4 inner input bits.
- For example $S_1(1,0,1,1,1,0)=[1,0,1,1]$.
- In the S-box below we represent the output values by the integer value of the four binary digits.

00	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
01	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
10	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
11	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

CSCI361
Computer Security

Modern Secret-key cryptography
II

Outline

- Properties of S-Boxes.
 - Boolean functions.
- Weaknesses in DES.
 - Complements.
 - Weak keys.
 - Brute force (in parallel).
- Multiple encryption.
 - Meet-in-the-middle.
 - Triple DES.
- Differential cryptanalysis.
- Linear cryptanalysis.

Properties of S-Boxes

- The design principles used for S-Boxes in DES were classified (US) information.
 - This was one of the controversial aspects of DES.
 - One of the designers, Coppersmith revealed the three design properties of S-boxes in 1994. These properties ensure good confusion and diffusion for the algorithm. They allayed some concerns regarding DES.
1. For every choice of control bits, every output bit is a **nonlinear** function of input bits.
 - Each row of an S-box consists of 4 non-linear Boolean functions. This non-linearity is the basis of the required **confusion** for DES.

2. Changing a single input bit to an S-box changes at least two output bits of that S-box.
 - This means that the statistical properties of the input bits are spread over the output bits → **diffusion**.
 3. When a single input bit is held constant, there is a good balance of 0's and 1's in the $2^5=32$ output half-bytes as the remaining 5 input bits are varied.
 - This ensures a uniform distribution of the algorithm output, and stops cryptanalysts from making statistical inferences.
- 1, 2 and 3 imply good confusion and diffusion.

Boolean functions

- A Boolean function is *linear* if it can be written as

$$f(x_1, x_2, \dots, x_n) = \sum a_i x_i$$

where a_i and x_i take only value 0 or 1 and addition is modulo 2.

- The function $f(x_1, x_2, x_3) = x_1 + x_2 x_3 \pmod{2}$ is *non-linear*, since it cannot be written in that form.
- Linear functions are cryptographically weak because the relationship between input and output is very simple.

Number of rounds

- After 8 rounds every bit of output is affected by every bit of input and every bit of key. That is, **the input information is diffused over the whole output block!**
- Moreover, the dependency between input, output and key is very complex.
- However attacks found on DES imply that 8 rounds does not provide enough complexity.
- 16 rounds are used.

Weaknesses in DES

1. Complement property:

Let \bar{u} denote complement of u . That is,

$$\bar{u} = u + 1 \pmod{2}.$$

$$DES_z(X) = DES_{\bar{z}}(\bar{X})$$

So, if we know ciphertext Y for a plaintext X produced with a key Z , then we also know the cryptogram that key \bar{Z} produces for \bar{X} .

This effectively halves the number of keys to be tested in an exhaustive key search.

2. Not every key is a good key:

Let Z denote the original 64 bit key.

Z is a **weak** or **self-dual** key if the key scheduling algorithm applied to the key Z produces identical sub-keys
 $Z_1 = Z_2 = Z_3 = \dots = Z_{16}$.

In this case encryption and decryption are the same operation.

$$DES_z = DES_z^{-1}$$

- There are four weak keys. (With each 8-bit word having odd-parity).
 - [00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001]
 - [11111110 11111110 11111110 11111110
11111110 11111110 11111110 11111110]
 - [00011111 00011111 00011111 00011111
00001110 00001110 00001110 00001110]
 - [11100000 11100000 11100000 11100000
11110001 11110001 11110001 11110001]

v. These are the keys which make C_0 all zero or all one, **and** also make D_0 all zero or all one.

(01010101010101), (FEFEFEFEFEFEFEFE),
(IFIFIFIF0E0E0E0E), and (E0E0E0E0FIFIFI)

- Z is a **semi-weak** key, or **key with a dual**, if there is another key Z' such that

$$DES_Z^{-1} = DES_{Z'} \quad \text{or} \quad DES_{Z'}^{-1} = DES_Z$$

There are 12 semi-weak keys (with odd parity for each 8-bit word). 6 pairs.

E001E001F101F101
FE1FFE1FFEOEFEOE
E01FE01FF10EF10E
01FE01FE01FE01FE
011F011F010E010E
EOFEE0FEF1FEF1FE

01E001E001F101F1
1FFE1FFEOEFEOEFE
1FE01FE00EF10EF1
FE01FE01FE01FE01
1F011F010E010E01
FEE0FEE0FEF1FEF1

- Weak and semi-weak keys can have disastrous effects if multiple encryption strategies are used.
- However, they are too few to be of general cryptanalytic use.

3. Exhaustive key search: (brute force attack)

DES has $2^{56} \approx 10^{17}$ keys. Given a plaintext block and the corresponding ciphertext block, we can try each key to decrypt the ciphertext and compare the result with the known plaintext.

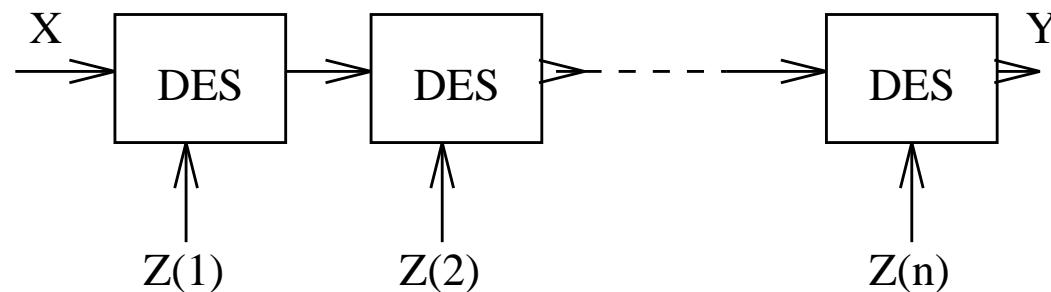
If we tried one key every 10^{-6} seconds, this would take about 10^{11} seconds, or about 3170 years!

- But we can **test keys in parallel**.
- Building a device with 10^7 DES chips would allow exhaustive cryptanalysis with only 10^{10} decryptions per chip.
- Even a few years ago chips existed with more than 10^5 encryptions/decryptions per second, proposals for 10^6 were around as long ago as 1998.
- January 1999 a DES key was broken in 22 hours and 15 minutes.
- Diffie & Hellman (1977) estimated $\frac{1}{2}$ day and \$5000 per solution on a special purpose computer costing about \$20,000,000.
- They extrapolated their result to conclude that in 1987 the cost of such a machine would be \$200,000.

- Michael Wiener (1993) gave a detailed design for a key search machine.
 - The machine was based on a key search chip which is pipelined, so that 16 encryptions take place simultaneously.
 - This chip can test $5*10^7$ keys per second, and could be built using existing technology (Field Programmable Gate Arrays) for \$10.50 per chip, at the time.
 - A frame consisting of 5760 chips could be built for \$100,000. This would allow a DES key to be found in about 1.5 days on average.
 - A machine using 10 frames would cost \$1,000,000, but would reduce the average search time to about 3.5 hours.

Increasing the key length

- There is one method of increase the key size which can be used for DES or any other cipher.
 - Multiple encryption.

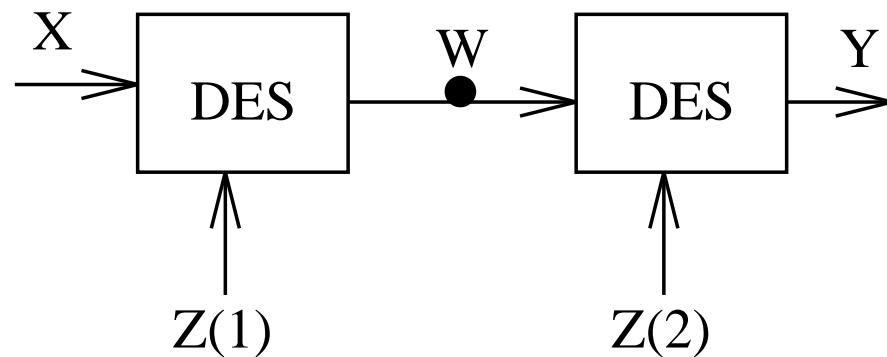


- Multiple encryption does not necessarily produce a stronger cipher. It depends on the algebraic structure of the cipher.
 - There were some concerns that DES might have a group structure, this was proven not to be the case.

- **Example:** Using double encryption with a monoalphabetic cipher does not produce a stronger cipher.
- It is important to consider what the effective key size is for double encryption.
- Naively we might consider that double encryption increases the size of the key by a factor of 2, that is the enemy must find 112 bits of key.
- Unfortunately this is not true, because the enemy can mount a **meet-in-the-middle** attack.

Meet-in-the-middle attack

- We can show that double (DES) encryption produces a cipher with effectively 57 bits of key, not 112.



- The enemy knows a pair of plaintext & ciphertext (X,Y), and a few further cryptograms.

Attack description...

1. The enemy tries all 2^{56} values of $Z(2)$ to decrypt Y . Then he produces a list (w_i, z_i) where

$$w_i = DES_{z_i}^{-1}(Y)$$

This list is sorted on w_i to facilitate later retrievals.

W	Z(2)
w_1	z_1
...	...
w_n	z_n

$$n=2^{56}$$

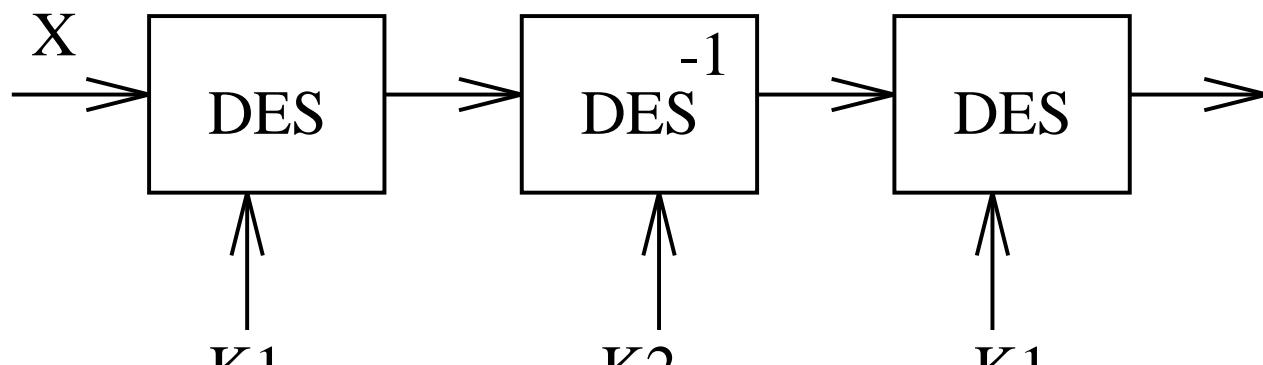
2. The enemy tries all 2^{56} values of $Z(1)$ to encrypt X .
 - After each encryption the enemy checks to see if the result is in the sorted list of w_i .
 - If $DES_{z'}(X)=w_p$, the attacker will use $Z(1)=z'$ and $Z(2)=z_p$ as the guessed keys.
 - These keys will be checked with other ciphertext, or other plaintext/ciphertext pairs if the attacker has them.
 - On the basis of this checking the guess is either confirmed, (z', z_p) is the secret key, or the attacker continues their search.

Number of operations

- The number of operations is equal to the number of encryption and decryption operations required.
- All 2^{56} decryptions for Z(2) are used to build the table.
- At most 2^{56} encryptions for Z(1) are required.
- Thus at most $2^{56}+2^{56}=2^{57}$ operations are required.
- So the effective key space is 57 bits.

Triple DES (3DES-EDE2)

- This was proposed by Tuchman (1978).
- There are various different “modes”.



- In this case meet-in-the-middle doesn't work.
- Software implementations of this cipher are relatively slow.

Differential cryptanalysis

- Biham & Shamir 1991
 - This is a **chosen plaintext attack** on iterated ciphers.
 - In differential cryptanalysis we analyse the effect of the difference in a plaintext pair on the differences of the resultant ciphertext pair. These differences are used to assign probabilities to possible keys and result in a **good estimate** of possible keys.
 - The essence of the attack is an observation made on a single round.

- Complexity of the attack is measured in terms of the number of encryptions, or the number of plaintext/ciphertext required for the analysis, and depends on the number of rounds.
- For small numbers of rounds the attack is more efficient than exhaustive key search but with increased number of rounds it becomes less efficient. The results to the right were announced in 1991:

Rounds	Complexity
4	2^4
6	2^8
8	2^{16}
10	2^{35}
12	2^{43}
14	2^{51}
15	2^{52}
16	2^{58}

- These results suggest the method does not work against 16 round DES.
 - Was choosing 16 rounds a coincidence?
 - No. DES designers at IBM knew by 1974 about differential cryptanalysis, long before it was used against FEAL by Murphy (1990) or applied against DES by Biham & Shamir (1991).
 - The S-Boxes were chosen to resists differential cryptanalysis.
- Another way of describing the effectiveness of the attack is to describe how much information is needed. A differential cryptanalysis attack against DES requires 2^{47} chosen plaintexts.
- An interesting result of this attack is that it shows **no key scheduling** can make DES more secure against this attack.

Linear cryptanalysis

- Matsui 1993. First applied to another cipher (FEAL, Matsui & Yamagishi, 1992).
- This method finds linear approximations for DS which hold with a probability $p \neq \frac{1}{2}$.

$$x_{i_1} \oplus x_{i_2} \dots y_{j_1} \oplus y_{j_2} \dots = z_{k_1} \oplus z_{k_2} \dots$$

- These kind of equations can be used to find estimates for key bits: The left hand side of the equation is calculated for many pairs of plaintext/ciphertext. The majority outcome gives an estimate for right hand side of the equation.
- It exploits the fact that knowledge of *pairs* of plaintext blocks, and the corresponding ciphertext blocks, usually allows the round key to be determined.
- This requires approximately 2^{43} known plaintexts.

CSCI361
Computer Security

Modes for block ciphers

Outline

- Modes.
 - Electronic codebook.
 - Cipher block chaining.
 - Cipher feedback.
 - Output feedback.
 - Counter.
- Advantages/Disadvantages.
- Padding.

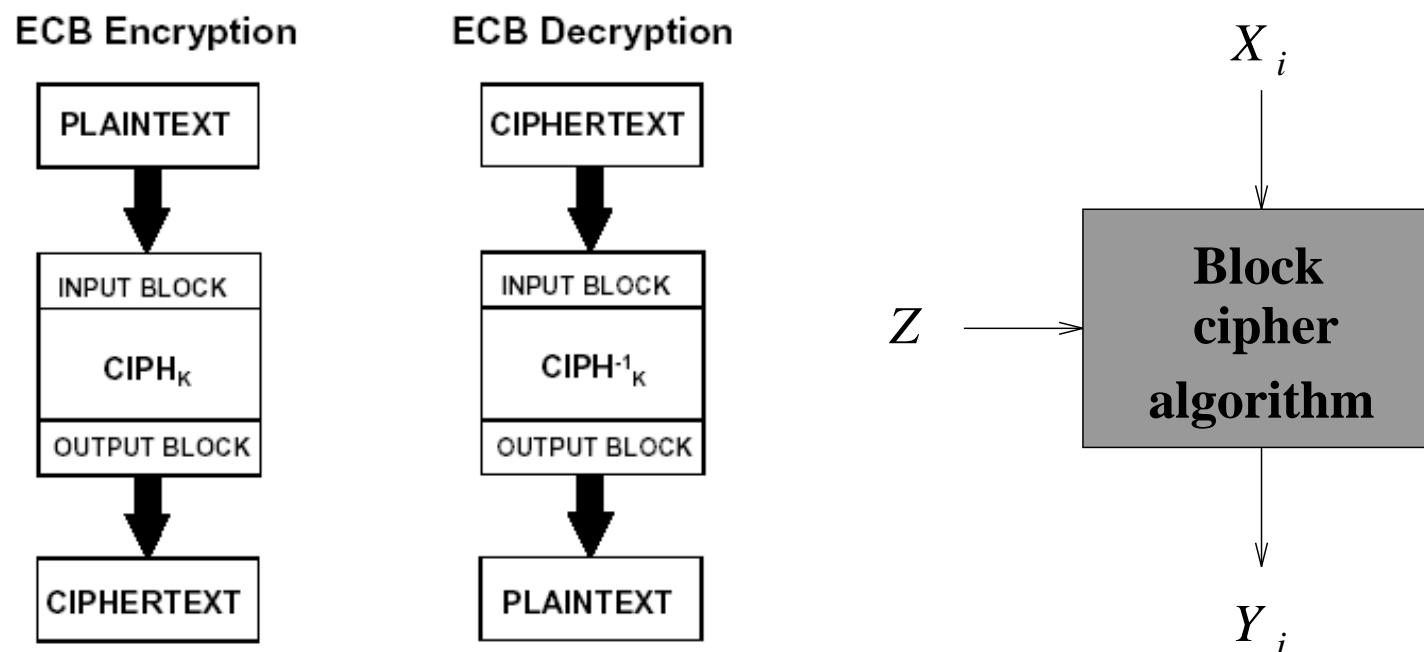
Modes of operation for block ciphers

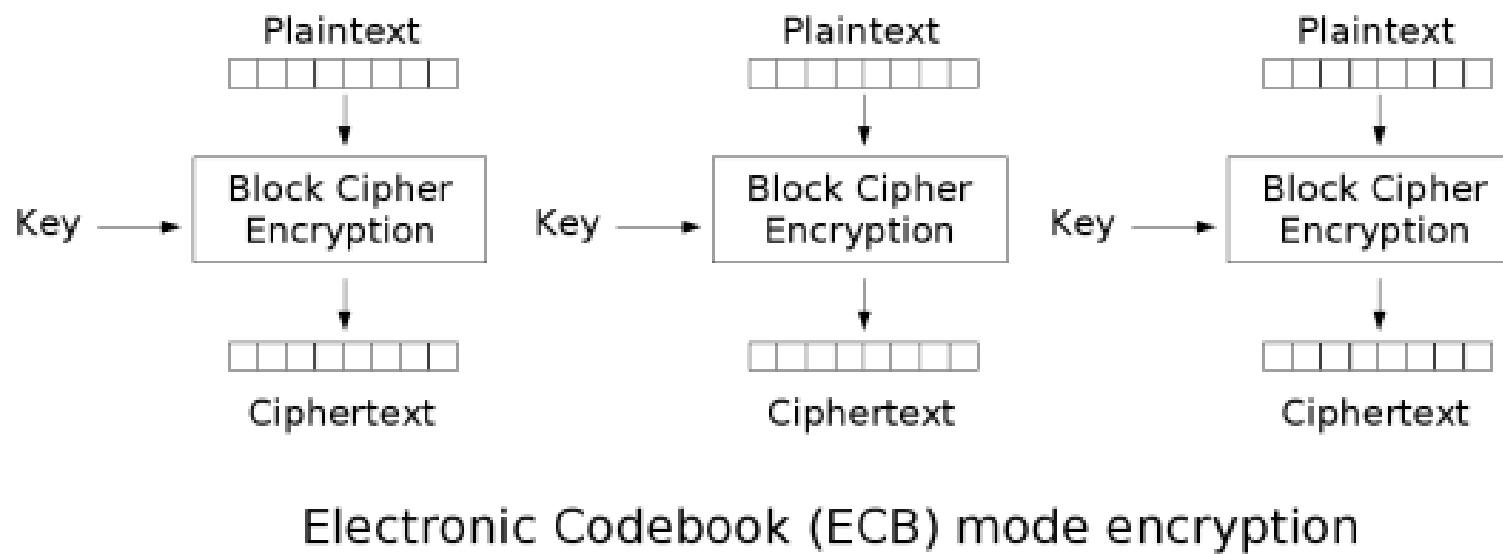
- A block cipher is defined to work on blocks of a particular size. A message will generally consist of multiple blocks. **Modes** describe the relationship between the application of the block cipher to different blocks.
- A block cipher can be used in different **modes**. The following four modes are standard modes which were recommended for DES.
 1. Electronic codebook mode (ECB).
 2. Cipher block chaining mode (CBC).
 3. Cipher feedback mode (CFB).
 4. Output feedback mode (OFB).

Although the modes can have different security properties, the length of the key remains the same, and therefore the upper bound on security stays the same also.

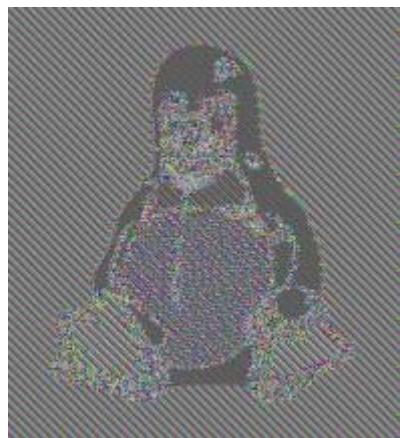
Electronic codebook mode (ECB)

- This is the basic mode of operation. A plaintext message is broken into blocks and each block is encrypted separately.

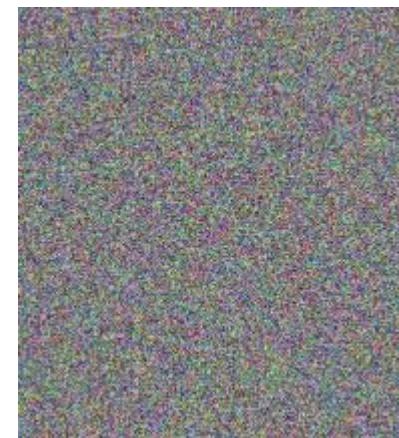




- Electronic codebook mode has the **disadvantage** that if we encrypt two identical plaintext blocks, the resulting ciphertext blocks will be the same.
- This leaks some information to an eavesdropper.
- Formatting information, such as spaces between columns or paragraphs, and other plaintext details can also leak through the cipher system.



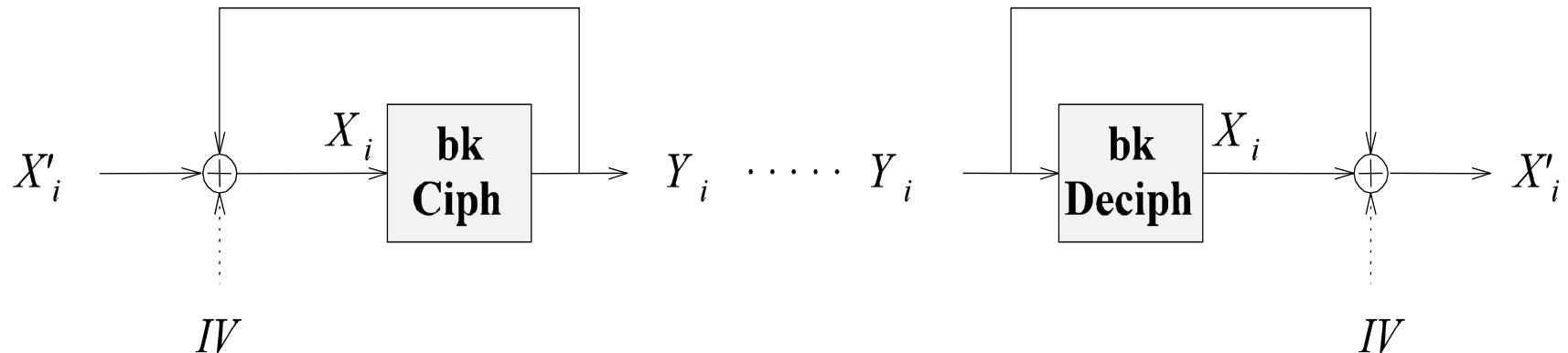
ECB



Other

http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

Cipher block chaining mode (CBC)



- In this mode a plaintext block is XORed with the ciphertext block of the previous round and then entered to the encryption algorithm.
- A strong dependency between consecutive blocks is created.

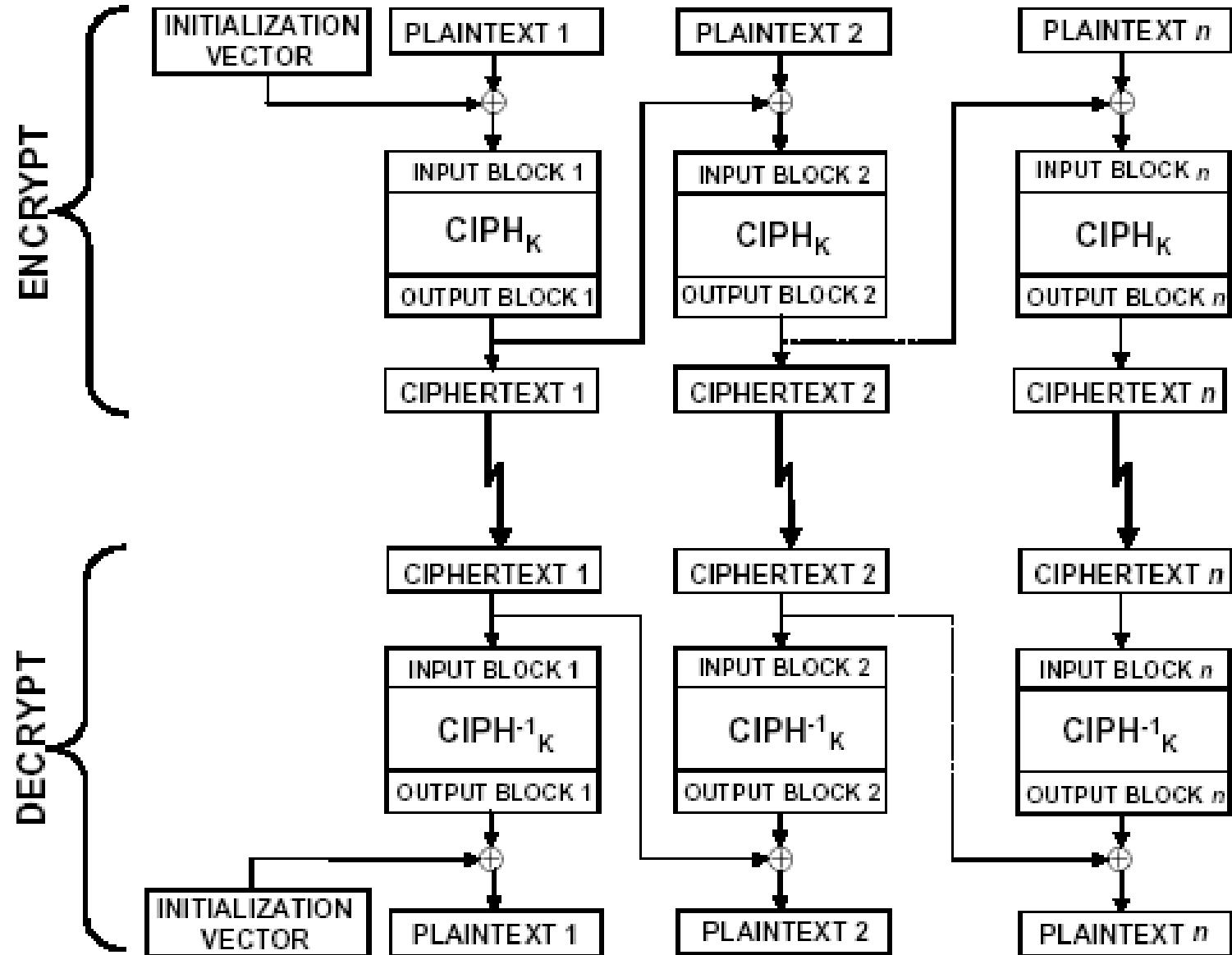
$$X_1 = X'_1 \oplus IV$$

$$X_2 = X'_2 \oplus Y_1$$

...

$$X_i = X'_i \oplus Y_{i-1}$$

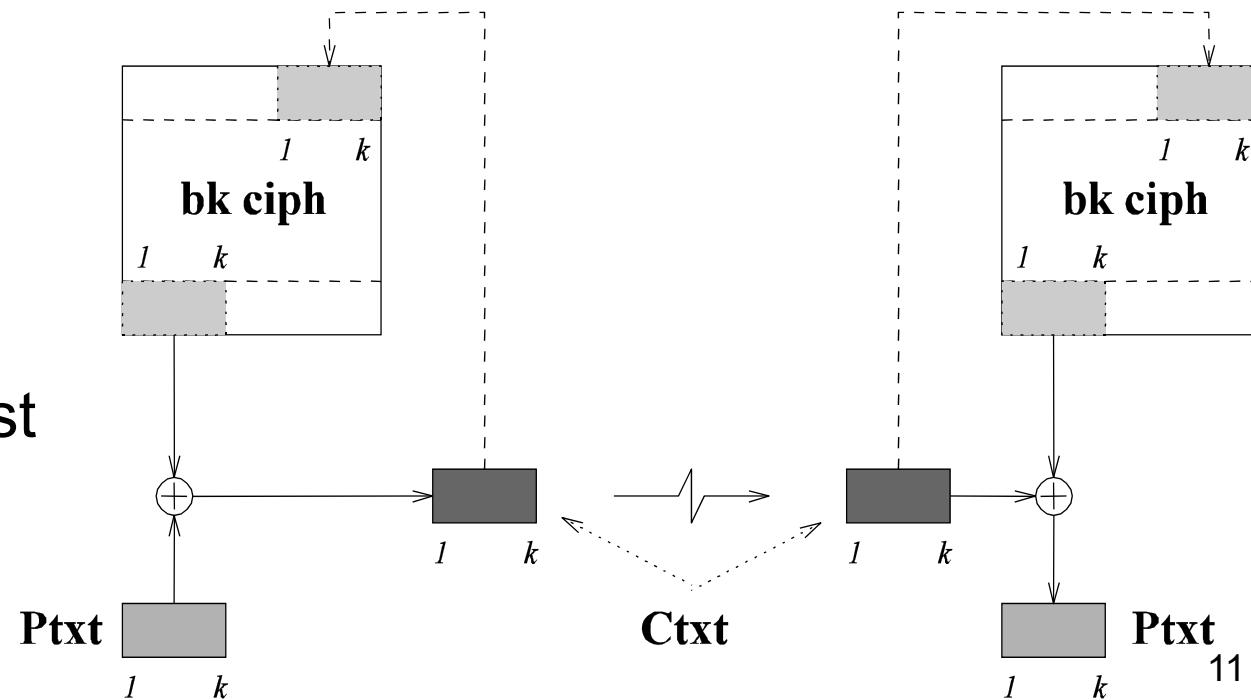
- For the first block, since ciphertext from the previous block does not exist, a random initial vector, called the IV, is used. IV is not secret but it needs to be unpredictable.
- CBC mode produces different ciphertext blocks even if plaintext blocks are the same.
- The strong dependency between blocks suggests that CBC can also be used to provide integrity.

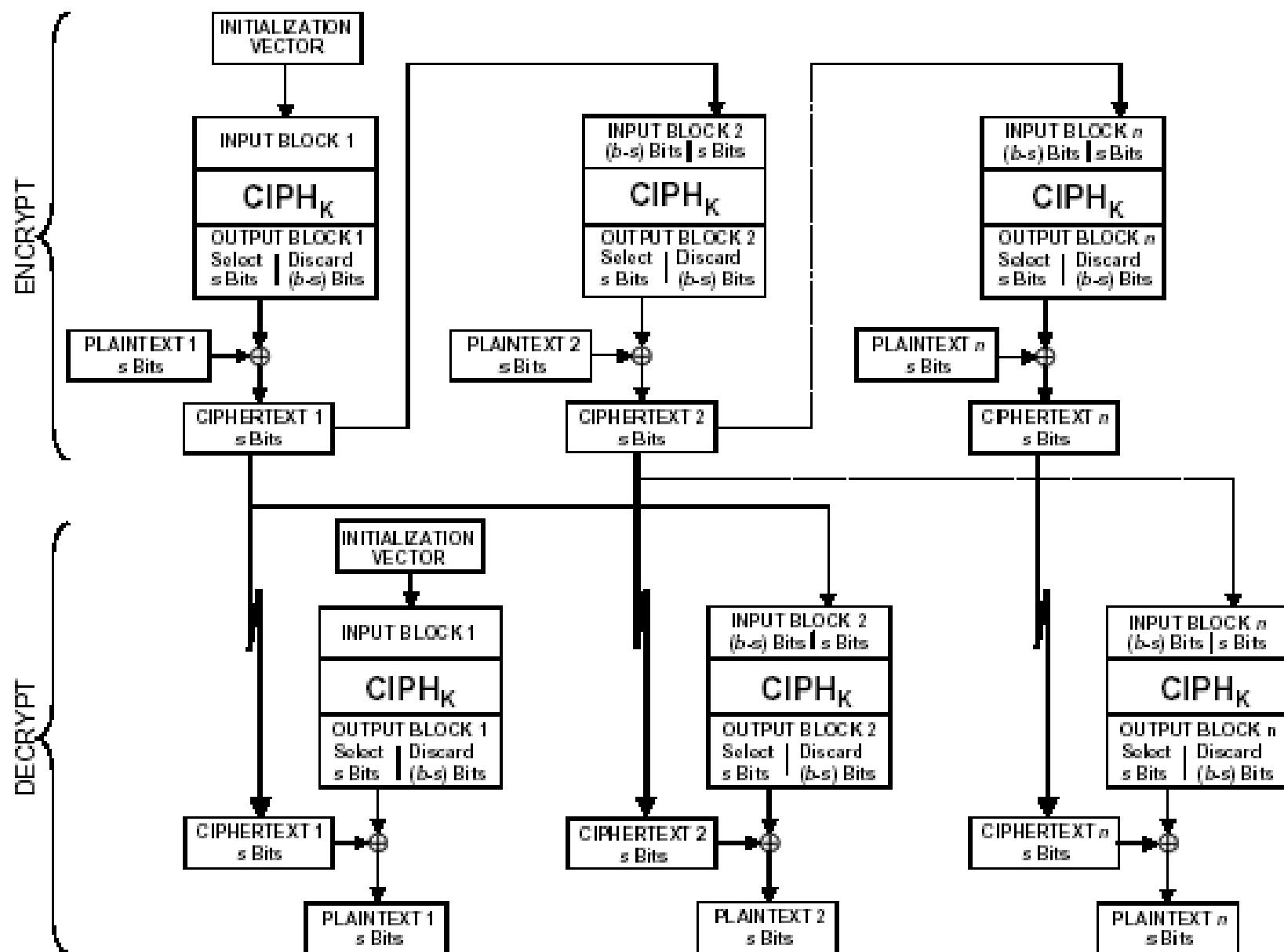


(k-bit) Cipher feedback mode (CFB)

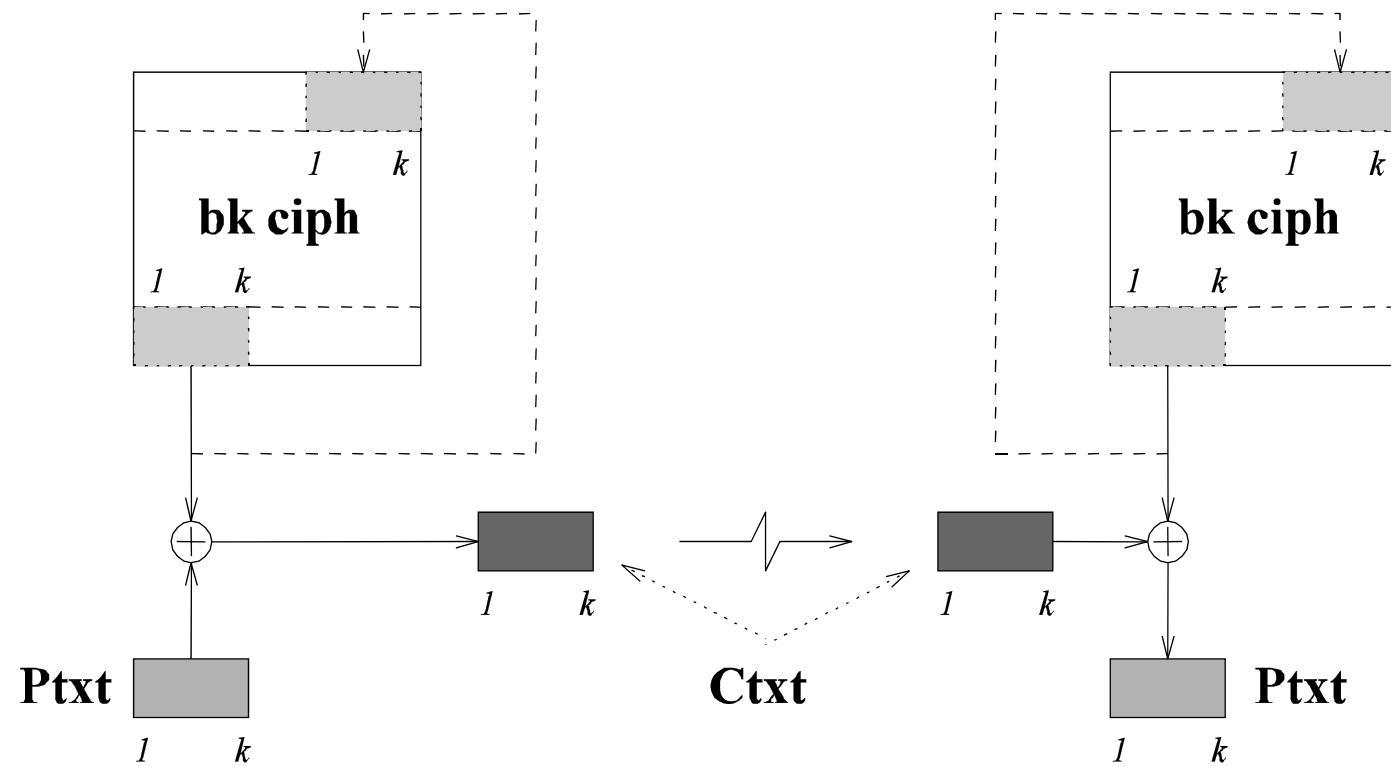
- The block cipher algorithm is effectively turned into a pseudorandom generator that produces a k -bit pseudorandom number in every execution of the algorithm.
- For decryption a similar generator is used to remove the masking pseudo-noise data.
- The input to the cipher “at the top” is the obtained by concatenating the input to the previous round, left shifted by k -bits, with the k -bit feedback.

Using smaller k in general produces a **more secure** pseudorandom stream, with the cost of lowering speed.



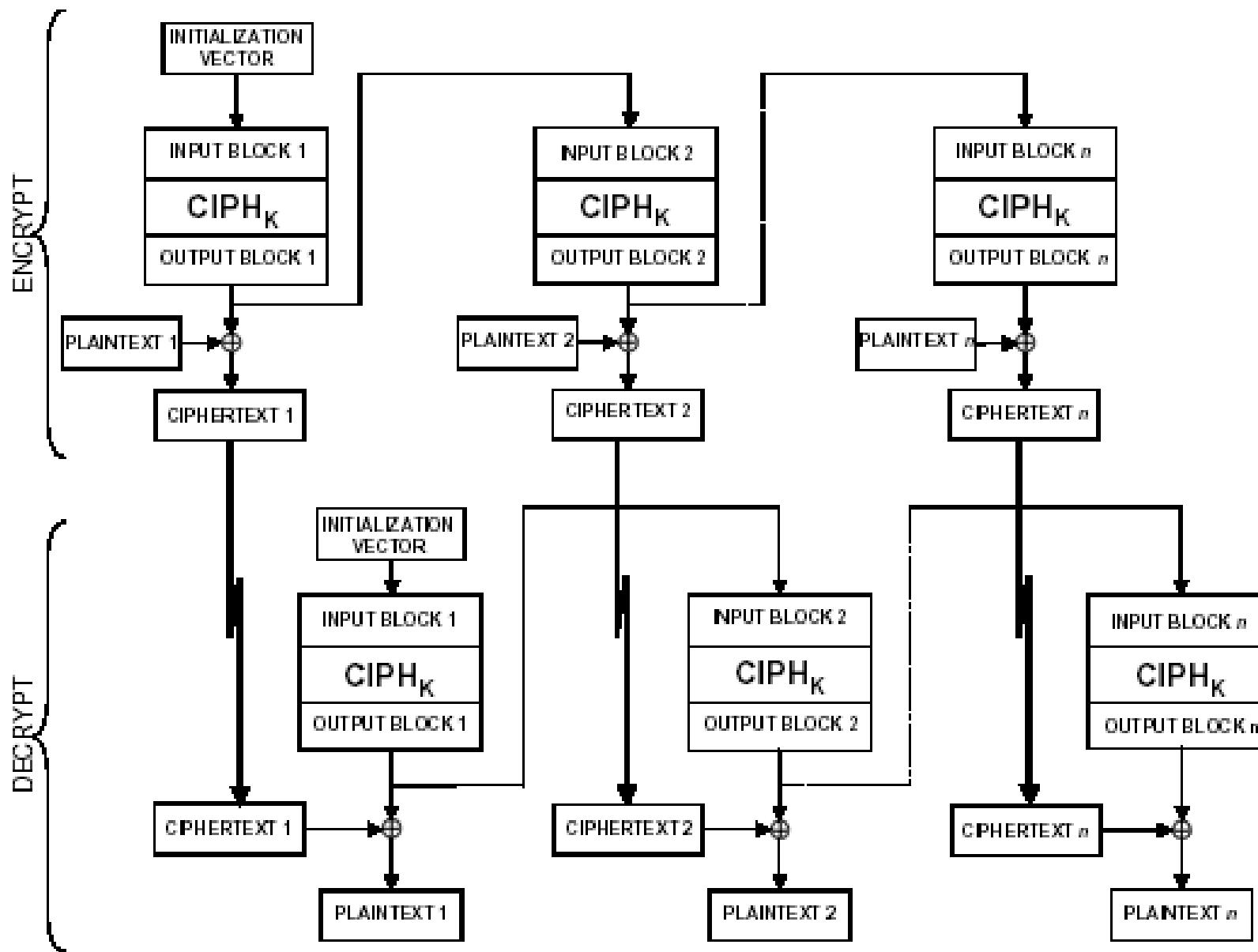


(k-bit) Output feedback mode (OFB)



- This is very similar to CFB, but the feedback is independent of the transmitted data.
- This means the pseudorandom stream can be pre-generated prior to any transmission.

A special case of OFB: k=block size.



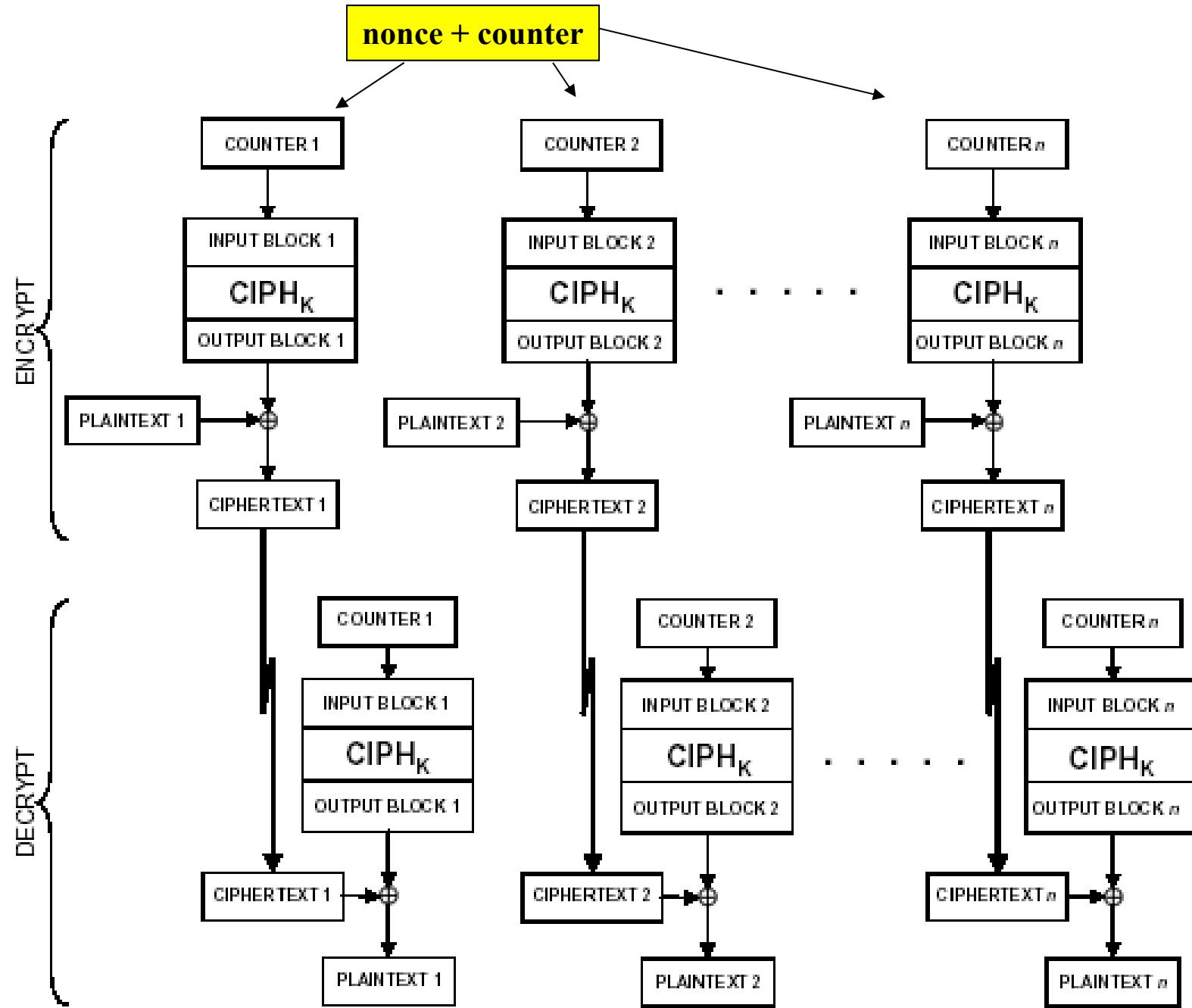
Advantages of CBC, CFB and OFB

1. All hide patterns in plaintext since $X_{i+n}=X_i$ does not imply that $Y_{i+n}=Y_i$.
 - Two identical plaintext blocks at two different locations in a ciphertext will produce two distinct ciphertexts. This will stop the leakage of information which occurs in ECB mode.
2. Prevent data tampering (CFB).
 - The dependency between ciphertext blocks can be used to detect tampering of the ciphertext. That is, a single bit change in a ciphertext block will affect decryption of all the following ciphertext blocks.
3. The k-bit ciphers (CFB and OFB) can be tuned, with respect to k, to allow for the user's data format, required level of security and resources.
4. The OFB cipher can be performed in parallel, with pre-computation of the pseudorandom stream.

Counter Mode (CTR)

- The cipher is applied to a set of input blocks, called **counters**.
- The sequence of output blocks are XORed with the plaintext to produce the ciphertext.
- The sequence of counters must satisfy the following property:

Each block in the sequence is different from every other block, across all of the messages that are encrypted under the given key.



- In both CTR encryption and decryption the ciphers can be performed in parallel.
 - The plaintext block that corresponds to any particular ciphertext block can be recovered independently from the other plaintext blocks.
 - The ciphers can be applied to the counters prior to the availability of the plaintext or ciphertext data.

Mode examples: ECB

- Let us consider a 3-bit block cipher with the following mapping:

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

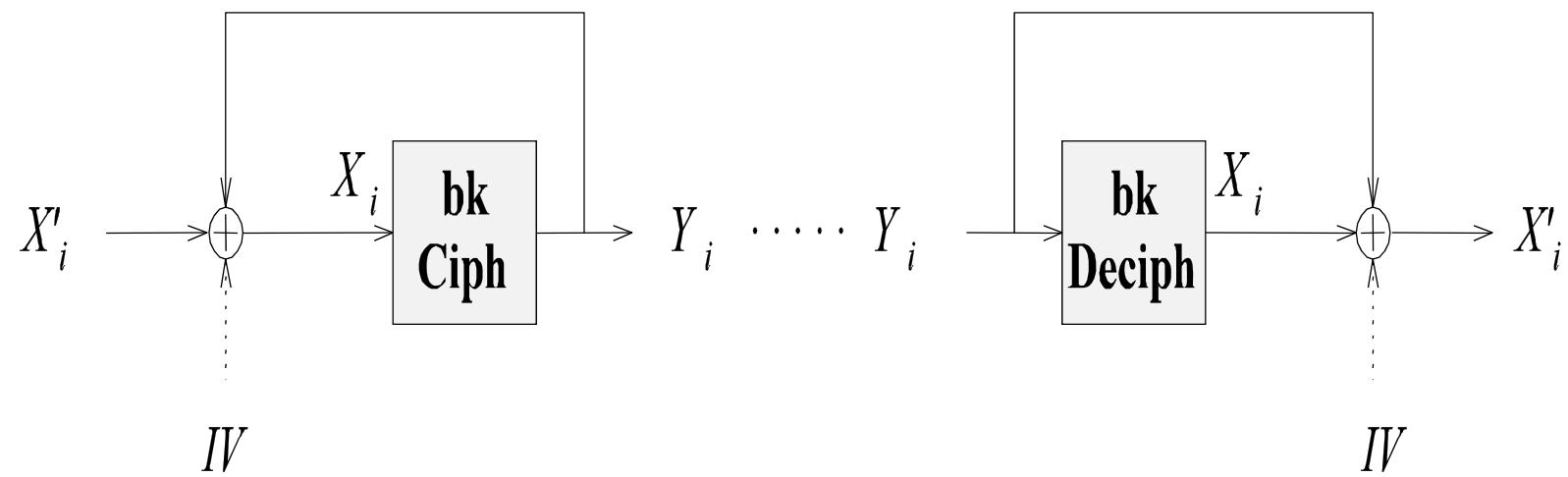
ECB

Plaintext	101	101	110	010
Ciphertext	000	000	101	011

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

$$\text{XOR}_0 = \text{Plaintext}_0 \oplus \text{IV}$$

$$\text{XOR}_i = \text{Plaintext}_i \oplus \text{Ciphertext}_{i-1}$$



CBC

IV=111

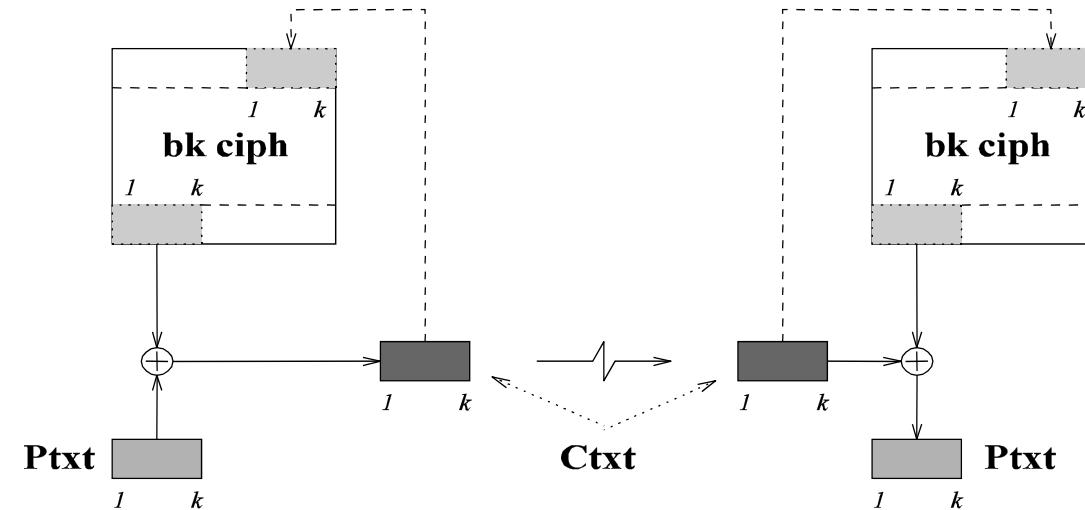
Plaintext	101	101	110	010
XOR	010	110	011	110
Ciphertext	011	101	100	101

20

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

2-bit CFB

IV=111

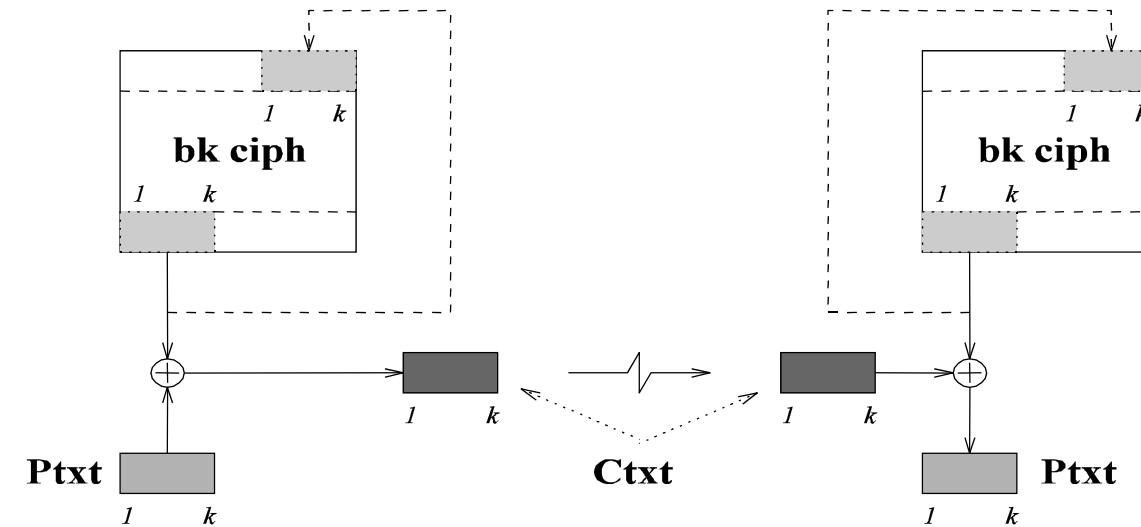


Plaintext	10	11	01	11	00	10
Input	111	111	110	011	101	100
E(Input)	010	010	101	100	000	001
Ciphertext	11	10	11	01	00	10

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

2-bit OFB

IV=111



Plaintext	10	11	01	11	00	10
Input	111	101	100	000	011	110
E(Input)	010	000	001	111	100	101
Ciphertext	11	11	01	00	10	00

Input	000	001	010	011	100	101	110	111
Output	111	110	011	100	001	000	101	010

CTR
IV=000

$$\text{Counter}_i = \text{Counter}_{i-1} + 1$$

Plaintext	101	101	110	010
IV	000	001	010	011
E(IV)	111	110	011	100
Ciphertext	010	011	101	110

Padding

- We have discussed block ciphers and described modes which tell us how to deal with multiple blocks.
- One issue we have not considered so far is that the last block of plaintext is, in general, a partial block of size u bits, this being less than the blocksize of the cipher.
- One needs to apply a **padding rule** to allow the last block to be suitably encrypted.
- That is, the block ciphers and modes mostly require full blocks.

- Padding example: Add a string 1000...0 to fill out the last block to the correct length. For decryption take the plaintext as being read back to the least significant 1.
 - Note that there are two possible plaintext values, that back to the least significant 1 or that with the full block (if padding hasn't been added).
- One could use an arbitrary method, essentially adding random data with a length for the last block included.
- For CTR mode one can simply use only the most significant u bits of the last output block for the XOR operation. This reveals the length of the message though, which is not always desirable.

CSCI361
Computer Security

Other block ciphers

Outline

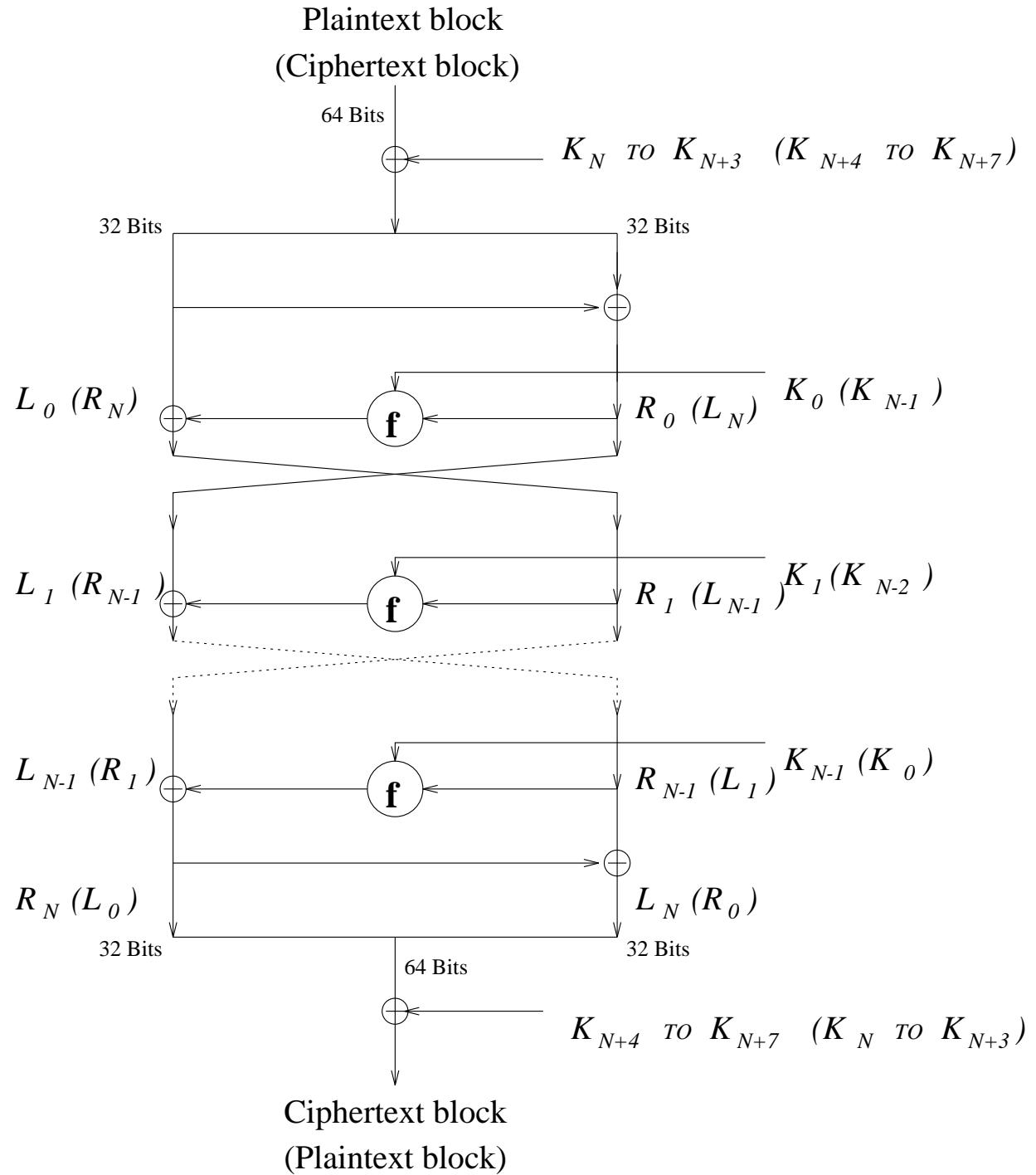
- FEAL
- LOKI91
- Blowfish
- IDEA 1990
- TEA
- RC5

FEAL

- The Fast Data Encryption ALgorithm (FEAL) was developed for high speed software implementation, at Nippon Telegraph and Telephone (NTT).
- The same algorithm is used for decryption, but the sub-keys are used in reverse order.
- FEAL-4 is a 64 bits block cipher with 64 bits key, proposed in 1987 (Miaguchi). It was successfully cryptanalysed with a chosen-plaintext attack in 1988 (den Boer).
- FEAL-8 was successfully cryptanalysed in 1989 (Biham-Shamir).
- FEAL-N/NX was developed in 1990.

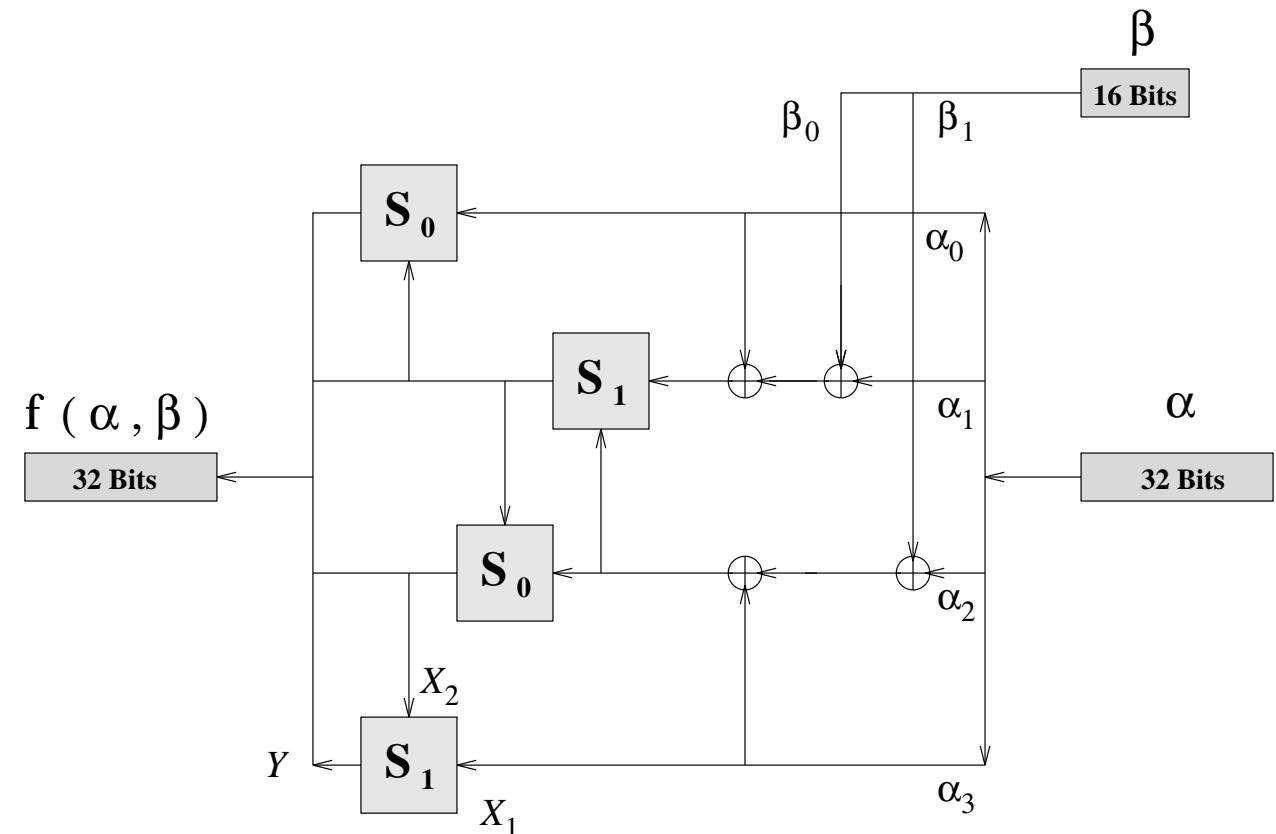
- Differential-linear cryptanalysis can break FEAL-8 with only **12 chosen plaintext blocks**.
- FEAL-N has a variable number of rounds. Biham-Shamir showed that for $N < 32$ the algorithm can be broken more quickly than 2^{64} chosen plaintext encryptions.
- FEAL-NX is a modification of FEAL that takes a 128 bit key. Biham-Shamir showed that the increase in key length does not add much to the security.

FEAL Encryption (Decryption)



FEAL

f function



The two S boxes act as follows:

$$Y = S_0(X_1, X_2) = \text{Rot2}((X_1 + X_2) \bmod 256)$$

$$Y = S_1(X_1, X_2) = \text{Rot2}((X_1 + X_2 + 1) \bmod 256)$$

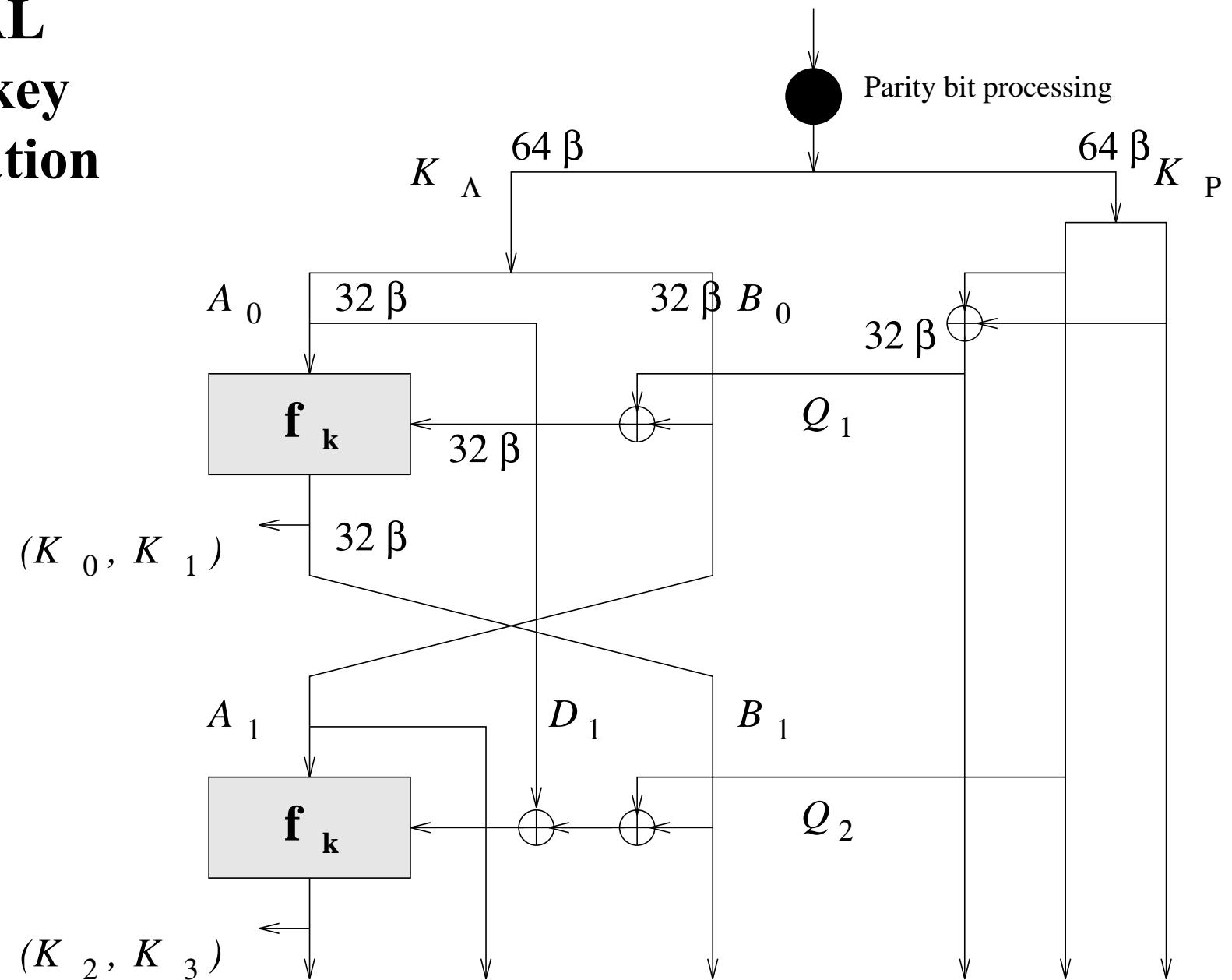
X_1 and X_2 are 8 bit inputs.

Y is an 8 bit output.

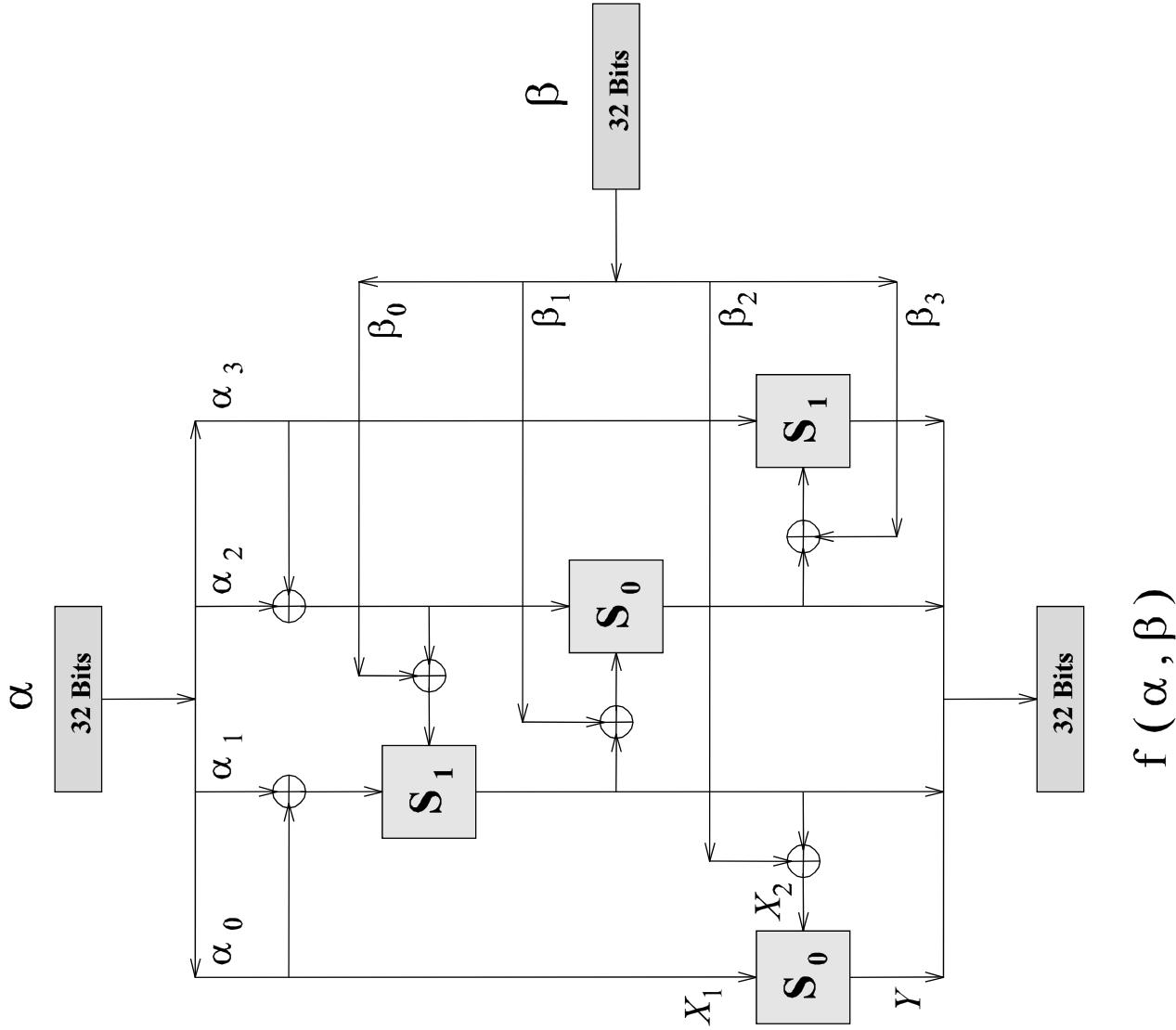
$\text{Rot2}(X)$ is a 2-bit left rotation on 8-bit data X .

FEAL Sub-key generation

Key block (K_A, K_P) : 128 bits



FEAL f_k



- The S boxes, S_0 and S_1 , are defined in the same way as for the FEAL f function.
- Note that f and f_k are necessarily different, since the key and data input sizes differ.

LOKI91

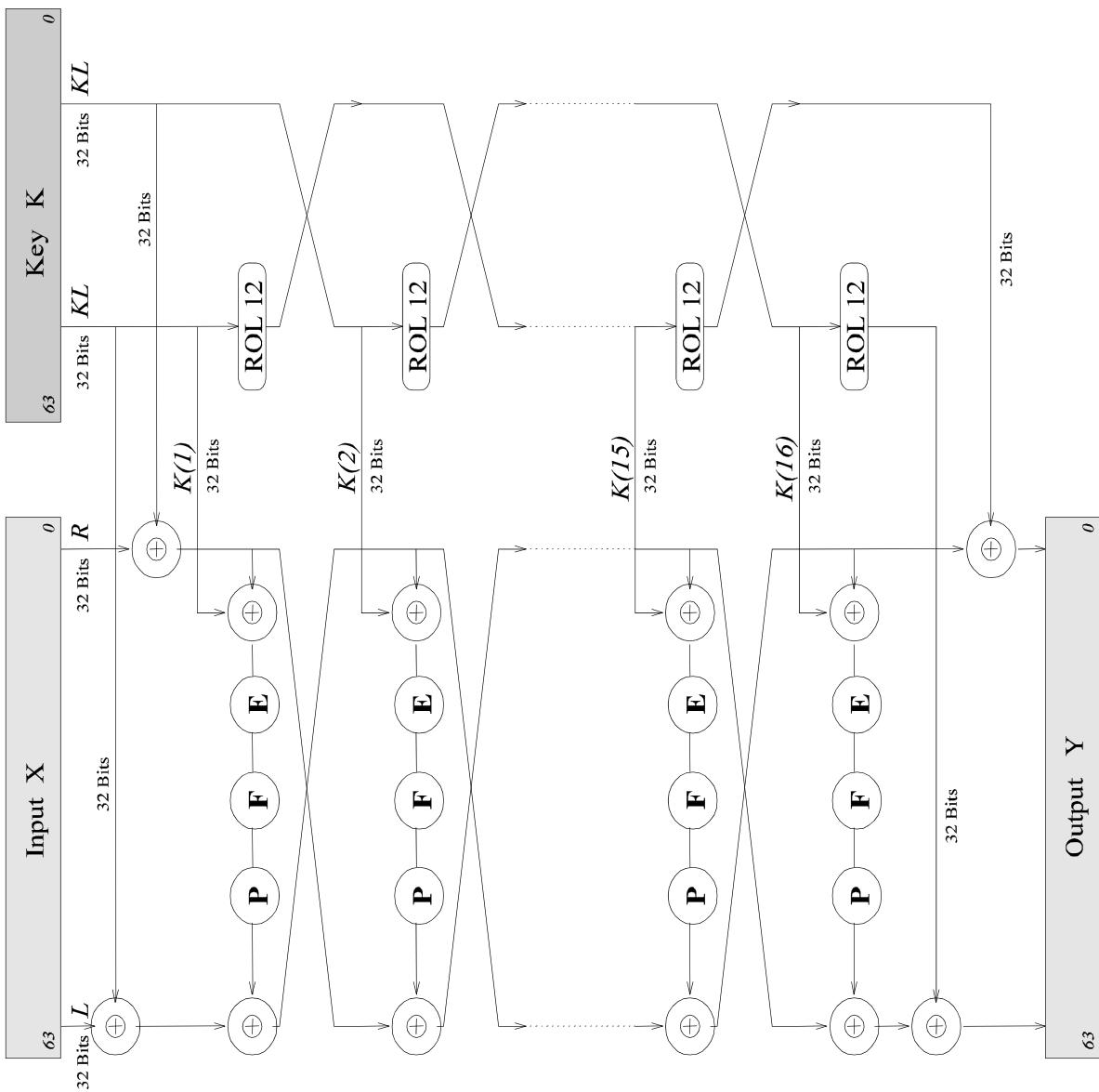
- Brown, Pieprzyk & Seberry.

- 64-bit block cipher.
- 64-bit key.
- 16 rounds.

- Secure against differential cryptanalysis.
 - LOKI89 wasn't.

- Weakness in key-scheduling allows related-key chosen-plaintext attack.

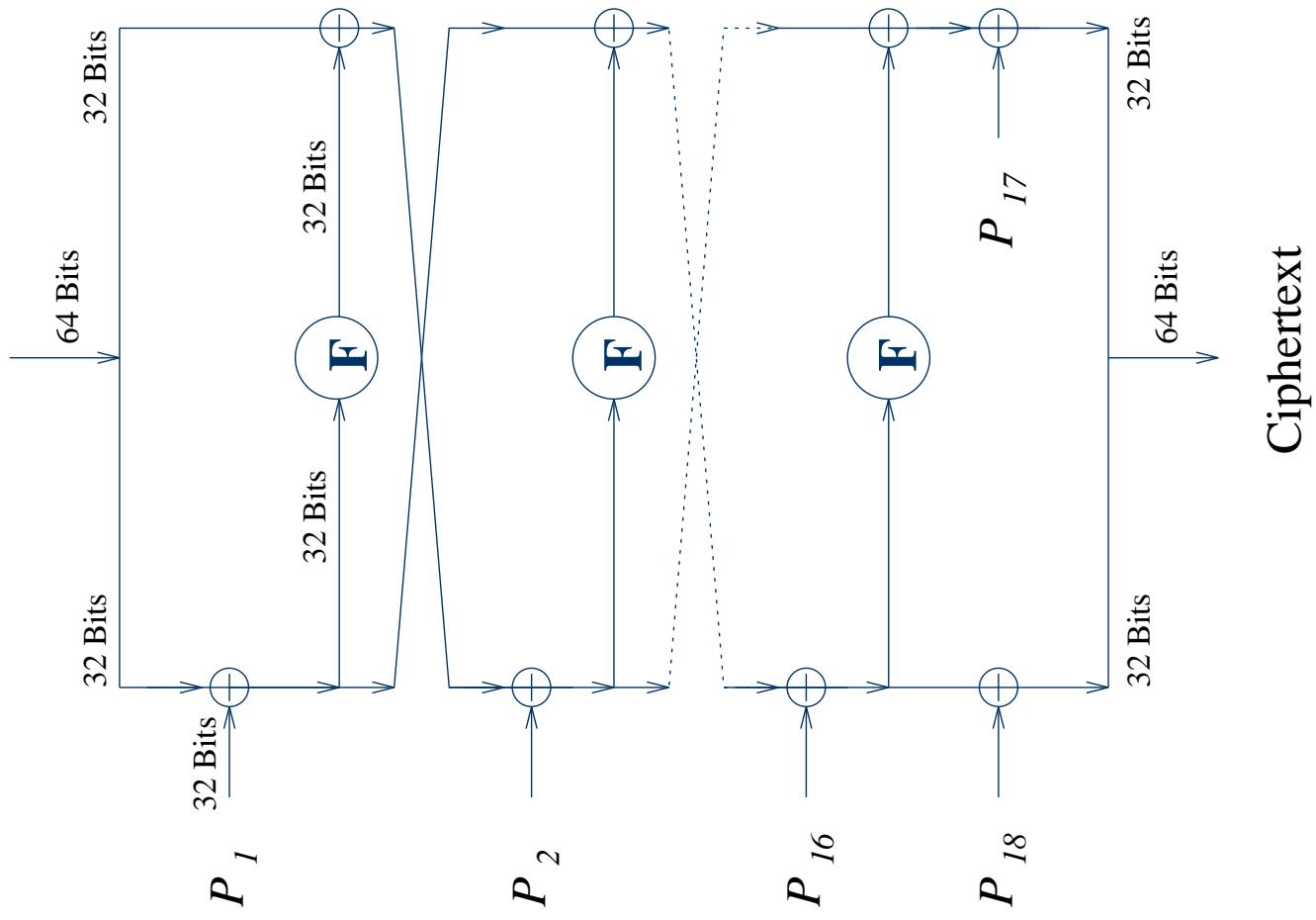
- LOKI97.



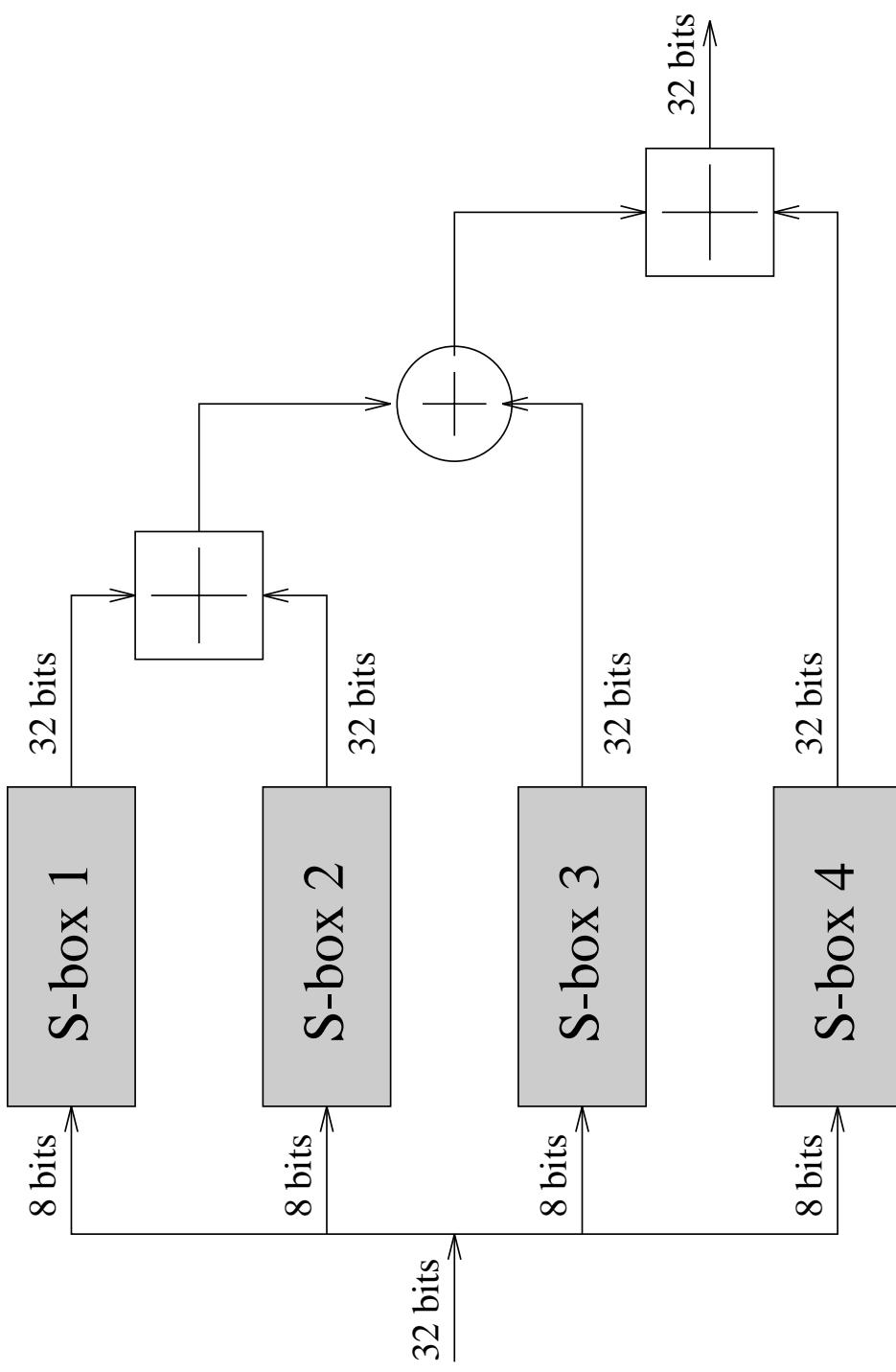
Blowfish

- Schneier (1993).
- 64-bit block cipher.
- 32→448-bit key.
- 16 round Feistel.
- One of the fastest block ciphers.
- Considered secure for key lengths>64-bits.
 - Exhaustive otherwise.
- There are attacks against reduced rounds.
- Key initialisation is relatively expensive so it isn't as useful where keys have a short lifetime.
- Unpatented.

Plaintext



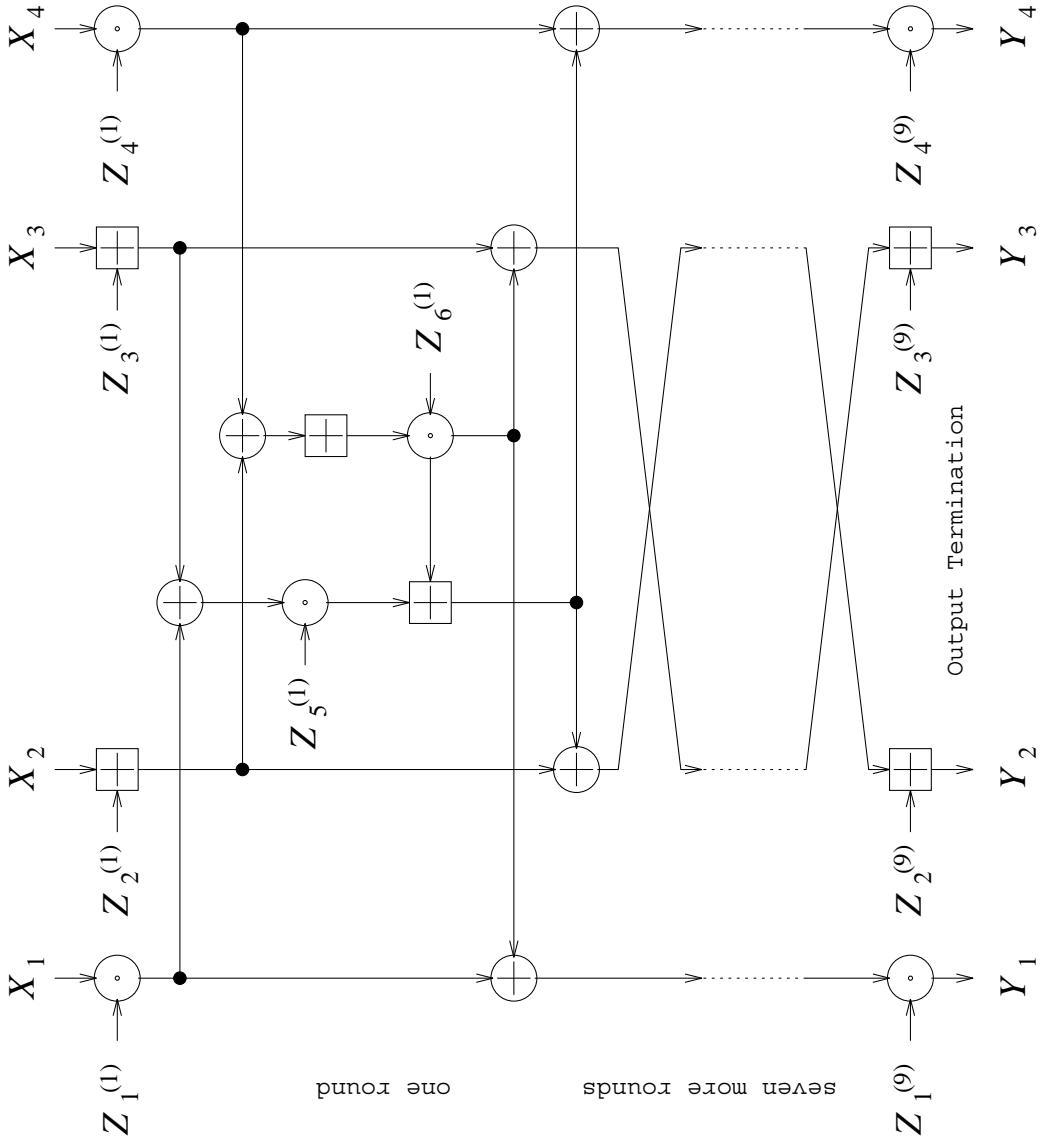
Blowfish



- Consists of key dependent permutation and key and data dependent substitution.
- The input X is divided into 8-bit blocks a, b, c and d .
- $F[X] = ((S_{1,a} + S_{2,b}) \text{ (mod } 2^{32}) \oplus S_{3,c}) + S_{4,d} \text{ mod } 2^{32}$

IDEA 1990

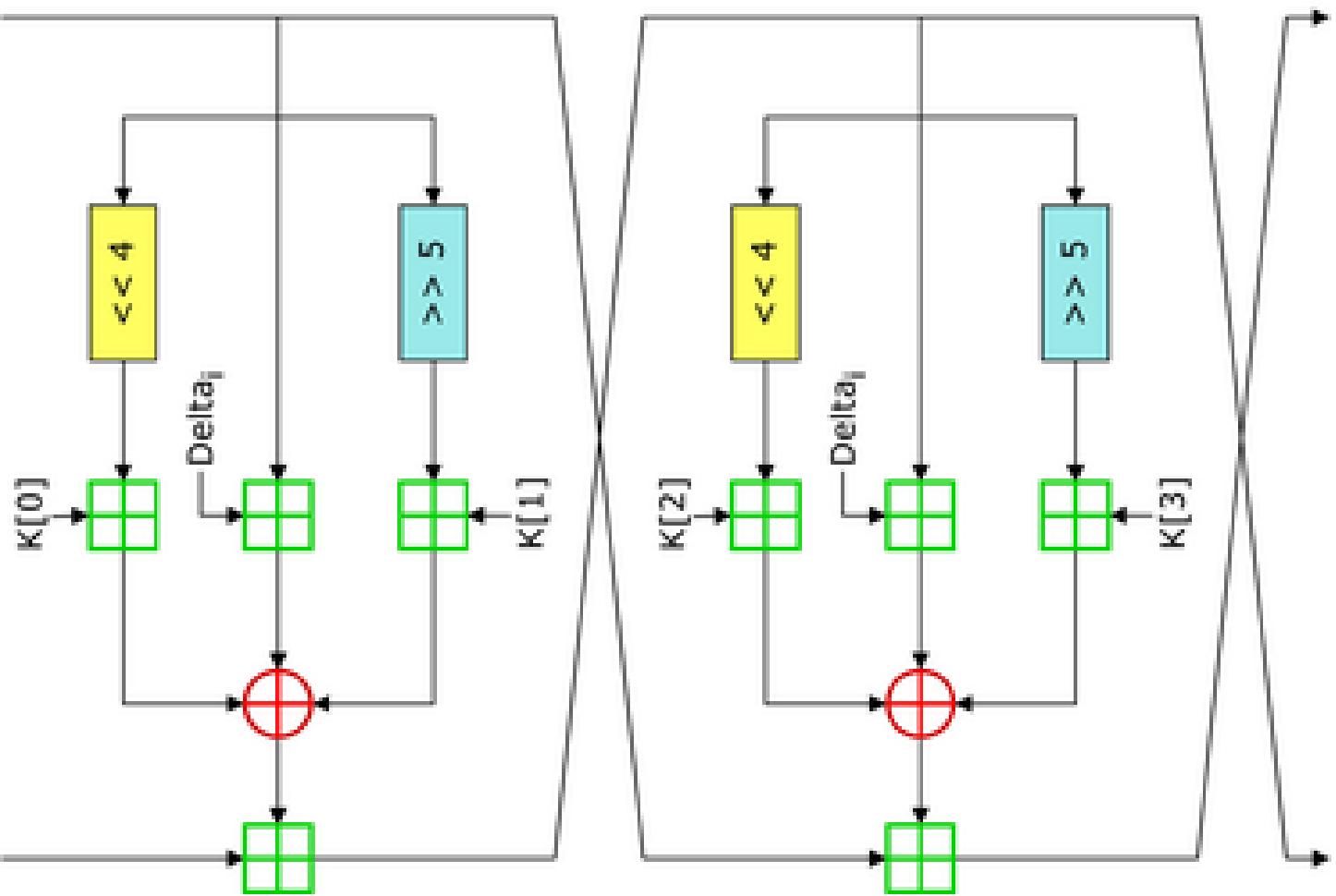
- International Data Encryption Algorithm (1990).
- Lai and Massey, ETH-Zurich (Swiss Federal Institute of Technology).
- Based on theoretical foundations.
 - It was designed to be resistant against differential cryptanalysis.
 - It can be shown that after 4 rounds it is resistant.
- 64-bit block cipher.
- 128-bit key.
- Operations used:
 - XOR
 - Addition modulo 2^{16}
 - Multiplication modulo $2^{16}+1$
- All operations are on 16-bit blocks: no bit permutation.
- IDEA with 8 rounds is considered “safe”.
 - There are fast attacks against reduced rounds (for example 5).
- Some weak keys have been identified, but they are few enough to not really be an issue.



- X_i : 16-bit plaintext sub-block \oplus : bit-by-bit exclusive-or of 16-bit sub-block
- Y_i : 16-bit ciphertext sub-block \boxplus : addition modulo 2^{16} of 16-bit integer
- $Z_i^{(r)}$: 16-bit key sub-block \circ : multiplication modulo $1+2^{16}$ of 16-bit integers with the zero sub-block corresponding to 2^{16}

TEA – Tiny Encryption Algorithm

- Designed by Needham and Wheeler (1994).
- TEA is unusual in that, as the name suggests, the description and implementation of the algorithm is relatively simple.
- 64-bit block cipher.
- 128-bit key.
- 64-round Feistel structure, with the rounds being paired into 32 “cycles”.
- Uses XOR, shift operations and modular addition.
- Simple (effectively trivial) key scheduling.



TEA

TEA Encryption

```
void encrypt(unsigned long* v, unsigned long* k) {
    unsigned long v0=v[0], v1=v[1], sum=0, i;           /* setup */
    unsigned long delta=0x9e3779b9; /* key schedule constant */
    unsigned long k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /* cache key */
    for (i=0; i < 32; i++) {                           /* basic cycle start */
        sum += delta;
        v0 += (v1<<4)+k0 ^ v1+sum ^ (v1>>5)+k1;
        v1 += (v0<<4)+k2 ^ v0+sum ^ (v0>>5)+k3; /* end cycle */
    }
    v[0]=v0; v[1]=v1;
}
```

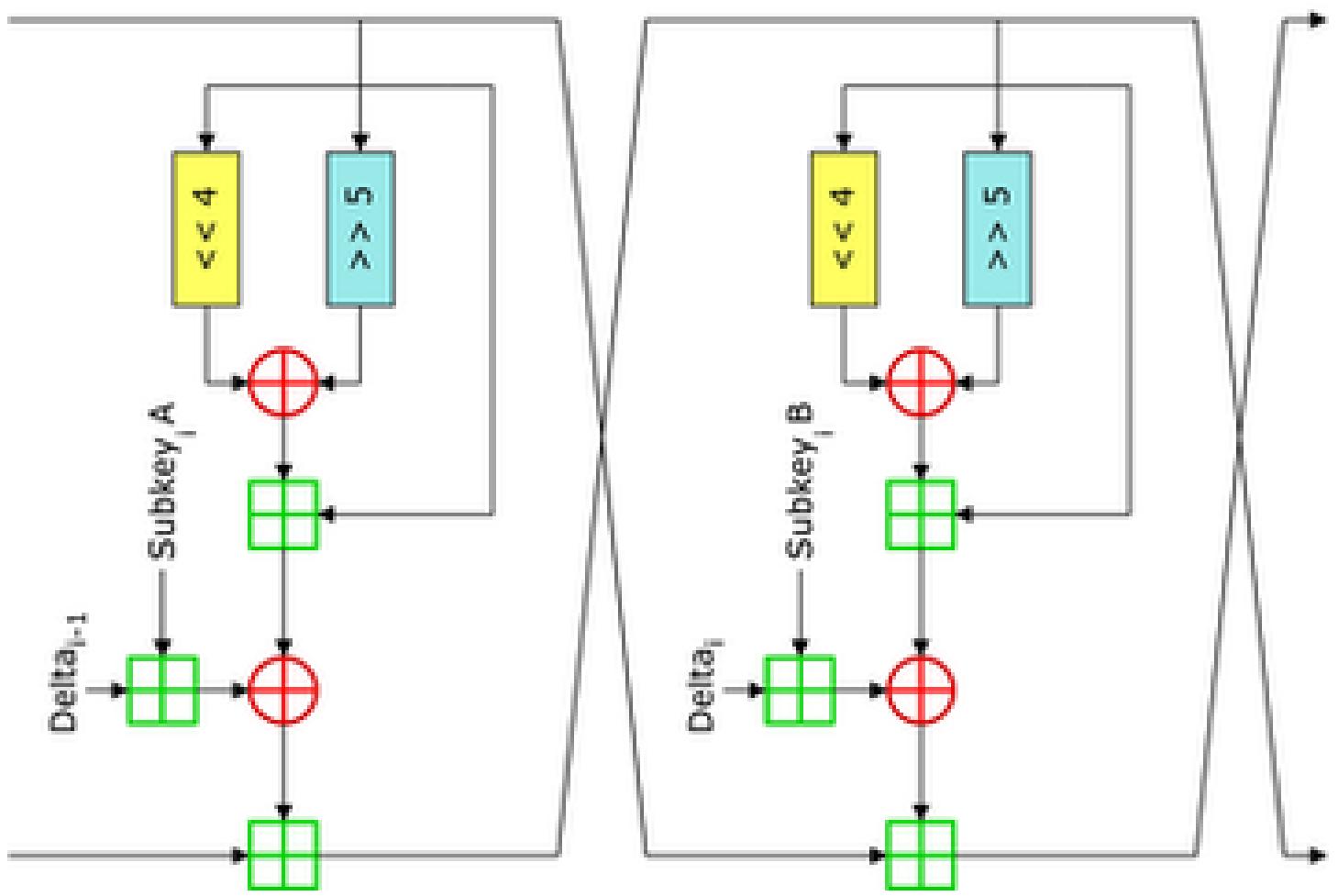
TEA Decryption

```
void decrypt(unsigned long* v, unsigned long* k) {
    unsigned long v0=v[0], v1=v[1], sum=0xC6EF3720, i; /* setup */
    unsigned long delta=0x9e3779b9; /* key schedule constant */
    unsigned long k0=k[0], k1=k[1], k2=k[2], k3=k[3]; /* cache key */
    for (i=0; i<32; i++) {                                /* basic cycle start */
        v1 -= (v0 << 4)+k2 ^ v0+sum ^ (v0 >> 5)+k3;
        v0 -= (v1 << 4)+k0 ^ v1+sum ^ (v1 >> 5)+k1;
        sum -= delta;
    }
    v[0]=v0; v[1]=v1;
}
```

- TEA has some problems, in particular with key equivalencies.
 - Each key is equivalent to three others. So the effective key space is 126 bits.
 - TEA was used in the Xbox, not for encryption purposes but for hashing. We will discuss hashing later in the course but the key equivalences in TEA made it possible to corrupt the Xbox. Basically TEA was used to check in memory had been tampered with.
- TEA is also vulnerable to related-key attacks. With 2^{23} chosen plaintexts, encrypted under two related keys, 2^{32} computations are enough to break TEA.

TEA variants

- Due to the weaknesses of TEA, mainly the related-key attack, several variants have been proposed.
 - XTEA (1997), Block TEA (1997), XXTEA (1998).
 - The best attack against XTEA is a 27-round related-key differential attack requiring $2^{20.5}$ chosen plaintexts under a related key-pair and required $2^{115.15}$ 27-round XTEA encryptions.

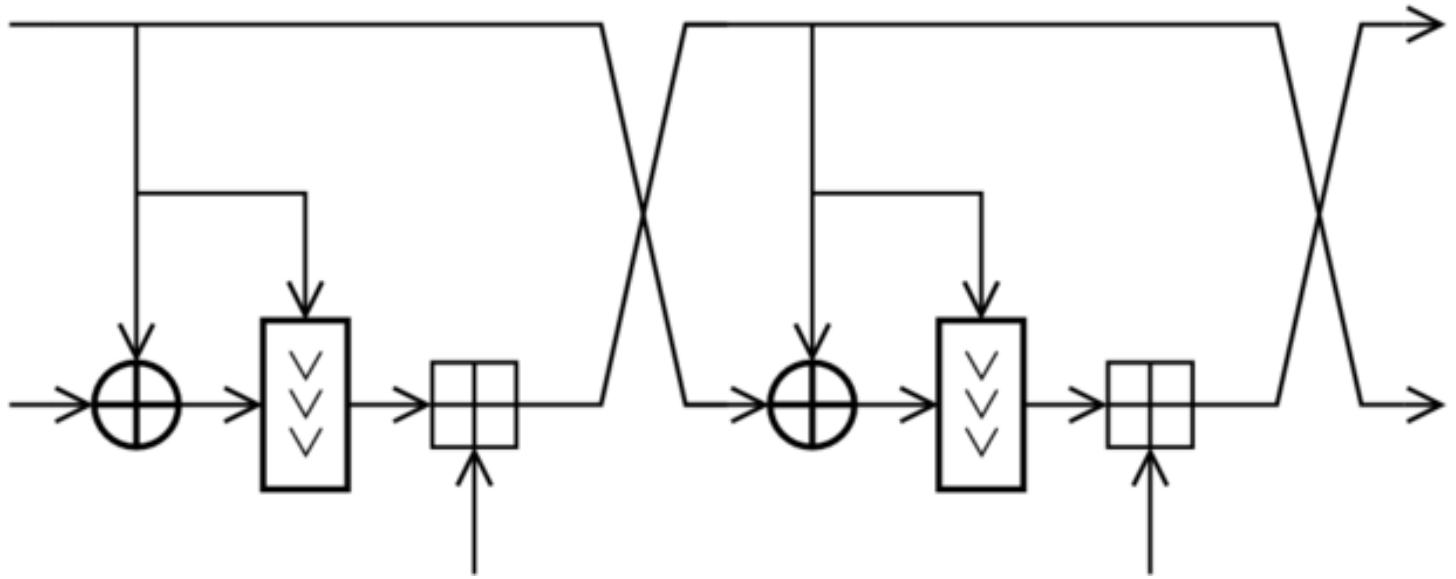


XTEA

RC5

- Designed by Ron Rivest (1994).
- Another “simple” cipher.
- Also based on XOR, additions and shifts.
- 32, **64** or 128-bit block cipher.
- 0-2040 bit key, 128-bit suggested.
- A Feistel network of 1-255 rounds, with 12 suggested.
- The shifts, or rotations, are actually data-dependent. This is unusual.

RC5



The <<< are shifts.

The square boxes are
modular additions.

The crossed circles are
XOR operations.

Breaking RC5, the hard way 😊

- There is a competition, with prize money, for breaking various levels of RC5. This is provided by RSA Security.
 - A 56-bit RC5 was cracked in 1997.
 - Tens of thousands of computers.
- “A search of approximately 34,225 trillion keys at a peak rate of over 7 billion keys per second. With over 72 quadrillion possible keys (72,057,594,037,927,936), the winning key was reported to RSA after searching a little more than 47% of the total.”

- The RC5-64 challenge was solved by distributed.net in about four years.
- 331,252 volunteers and their machines.
- Worth \$10,000.
- Finished September 2002.

CSCI361

Computer Security

Advanced Encryption Standard
(AES)

Outline

- The development of the Advanced Encryption Standard (AES).
 - Selection criteria.
 - Finalists.
- Rijndael.
 - Broad structure.
 - Detailed structure.
 - Round operations.

The development of AES

- The National Institute of Standards and Technology (NIST) worked with the international cryptographic community to develop an *Advanced Encryption Standard* (AES).
- The overall aim:
 - To develop a Federal Information Processing Standard (FIPS) for an encryption algorithm for protecting sensitive (unclassified) government information well into the twenty-first century.
 - The standard was planned for use by the U.S. Government and, on a voluntary basis, by the private sector.

- NIST selected *Rijndael* at the end of a very long and complex evaluation process:
 - January 2, 1997. NIST announced the initiation of the project.
 - September 12, 1997. NIST made a formal call for algorithms.

NIST specified minimum requirements:

- Implement symmetric key cryptography as a block cipher.
- Block size of 128 bits.
- Key sizes of 128, 192, and 256 bits.

- August 1998: NIST announced 15 AES candidate algorithms at the First AES Candidate Conference (AES1) and solicited public comments.
- March 1999: A Second AES Candidate Conference (AES2) was held to discuss the results of the analysis.
- August 1999: NIST announced its selection of five finalist algorithms from the fifteen candidates:
 - **MARS** (IBM; USA): Modified Feistel rounds in which one fourth of the data block is used to alter the other three fourths of the data block.
 - **RC6** (RSA Labs; USA): Feistel structure; 20 rounds.
 - **Rijndael** (Daemen, Rijmen; Belgium): Substitution-linear transformation network with 10, 12 or 14 rounds, depending on the key size.
 - **Serpent** (Anderson, Biham, Knudsen; UK, Israel, Norway): Substitution-linear transformation network consisting of 32 rounds.
 - **Twofish** (Counterpane; USA): Feistel network with 16 rounds, key-dependent S-boxes.

The finalists

- The five finalists are iterated block ciphers.
 - They specify a transformation that is iterated a number of times on the data block to be encrypted or decrypted.
- Each finalist also specifies a *key scheduling* algorithm for the production of sub-keys.
- NIST gave evaluation criteria to compare the candidate algorithms:
 1. Security.
 2. Cost.
 3. Algorithm and Implementation Characteristics.

■ **Security:**

- Resistance to cryptanalysis.
- Soundness of the mathematical basis.
- Randomness of the algorithm output.
- Relative security as compared to other candidates.

■ **Cost:**

- Computational efficiency (speed) on various platforms,
 - In Round 1, the speed associated with 128-bit keys (the primary key size).
 - In Round 2, hardware implementations and the speeds associated with the 192 and 256-bit key sizes were addressed.
- Memory requirements:
 - Memory requirements and software implementation constraints for software implementations.

- No attacks were reported against any of the full versions of the finalists.
- The only known attacks are against simplified variants of the algorithms: the number of rounds is reduced or simplified in other ways.
 - It is difficult to assess the significance of the attacks on reduced-round versions.
 - On the other hand, it is standard practice to try to build upon attacks on reduced-round variants.
- The *Security margin* is the degree by which the full number of rounds of an algorithm exceeds the largest number of rounds that have been attacked.
- However, not too much weighting was to be put on any single figure of merit for the security:
 - Security margin would tend to favour novel techniques but wouldn't be a good index to the resistance to new techniques.
 - There is no natural definition for the number of analyzed rounds:
 - MARS has 16 unkeyed mixing rounds and 16 keyed core rounds: is MARS a 16 round or a 32 round algorithm, or something in between?

Other areas of assessment by NIST

- Design Paradigms and Ancestry.
- Simplicity.
- Statistical Testing.
- Machine Word Size:
 - It appears that over the next 30 years, 8-bit, 32-bit, and 64-bit architectures will all play a significant role (128-bit architectures may well be added to the list at some point).
 - Performance with respect to wordsize separated into four groups: 8-bit, 32-bit C and assembler code, 64-bit C and assembler code, and other (Java, DSPs (digital signal processors), etc.).
- Software Implementation Languages
 - Assembler coding to evaluate the best performance on a given architecture.
 - Standard reference code for less costly development.

Rijndael

- Joan Daemen (Proton World International) and Vincent Rijmen (Katholieke Universiteit Leuven)
- The three criteria taken into account in the design of Rijndael were:
 - Resistance against all known attacks.
 - Speed and code compactness on a wide range of platforms.
 - Design simplicity.
- Based on the cipher **Square**, also developed by Daemen and Rijmen.

- The round transformation of Rijndael does not have the Feistel structure.
- The round transformation consists of three *invertible uniform transformations*, called *layers*.
- *Uniform*, means that every bit of the input is treated in a similar way.
- Every layer has its own function:
 - The **linear mixing layer** guarantees high *diffusion* over multiple rounds.
 - The **non-linear layer**: Parallel application of S-boxes that have *optimal worst-case nonlinearity properties*.
 - The **key addition layer**: A simple XOR of the Round Key to the intermediate State.
- The *Round Keys* (or sub-keys) are derived from the Cipher Key by means of the key schedule. This consists of two components:
 - *Key Expansion* and *Round Key Selection*.

- The basic principle for key generation is as follows:
 - The total number of Round Key bits is equal to the block length multiplied by the number of rounds plus 1. Thus for a block length of 128 bits ($N_b=4$ is the number of (32-bit) words in the block) and $N_r=10$ rounds, 1408 Round Key bits are needed.
 - The Cipher Key is expanded into an Expanded Key.
 - Round Keys are taken from this Expanded Key in the following way: the first Round Key consists of the first N_b words, the second one of the following N_b words, and so on.

The number of rounds

- The number of rounds, N_r , depends on the key size.

Key Size	Rounds
128	10
192	12
256	14

Attacks on Rijndael:

- (Ferguson et al. 2000)
- For 128-bit keys, 7 out of the 10 rounds of Rijndael have been attacked. The attack on 7 rounds requiring nearly the entire codebook.
- For 192-bit keys, 8 out of the 12 rounds have been attacked.
- For 256-bit keys, 9 out of the 14 rounds have been attacked.
- Rijndael appears to offer an adequate security margin.

The cipher

- Before we look at the specific details of the layer actions we will look at the overall cipher structure, starting with the **Round transformation**.

```
Round(State, RoundKey)
{
    ByteSub(State);
    ShiftRow(State);
    MixColumn(State);
    AddRoundKey(State, RoundKey);
}
```

- The final round of the cipher is slightly different. It is defined by:

FinalRound(State,RoundKey)

{

ByteSub(State) ;

ShiftRow(State) ;

AddRoundKey(State,RoundKey);

}

- The cipher consists of
 - An Initial Round Key addition.
 - $N_r - 1$ Rounds.
 - A final round.
- In pseudo code the cipher can be written as:

```
Rijndael(State,CipherKey)
{
    KeyExpansion(CipherKey,ExpandedKey);
    AddRoundKey(State,ExpandedKey);
    for ( i=1 ; i<Nr ; i++ )
        Round(State,ExpandedKey + Nb*i);
    FinalRound(State,ExpandedKey + Nb*Nr);
}
```

- Alternatively, the key expansion can be done beforehand, and Rijndael can be specified in terms of the *Expanded Key*.

```
Rijndael(State,ExpandedKey)
{
    AddRoundKey(State,ExpandedKey);
    for ( i=1 ; i<Nr ; i++ )
        Round(State,ExpandedKey + Nb*i) ;
    FinalRound(State,ExpandedKey + Nb*Nr);
}
```

The inverse cipher

- The inverse of a round is given by:

```
InvRound(State, RoundKey)
{
    AddRoundKey(State, RoundKey);
    InvMixColumn(State);
    InvShiftRow(State);
    InvByteSub(State);
}
```

- The inverse of the final round is given by:

```
InvFinalRound(State, RoundKey)
{
    AddRoundKey(State, RoundKey);
    InvShiftRow(State);
    InvByteSub(State);
}
```

How does it work?

- See the presentation on AES with 10 rounds.
- Then you can return to this set of slide if you wish.

Evaluation

- Rijndael designers described their motivation for design choices including substitution box, mix column transformation and the shiftRow offsets.
- The Rijndael cipher can be efficiently implemented on a wide range of processors and in dedicated hardware, 8-bit processors typical for current Smart Cards, and on 32-bit processors typical for PCs.
- There is considerable parallelism in the round transformation.

Security

- Rijndael has provable security against known attacks (in particular with respect to differential and linear cryptanalysis).
- The cipher and its inverse use different components and this practically eliminates the possibility of weak and semi-weak keys, as exist in DES.
- The non-linearity of the key expansion practically eliminates the possibility of equivalent keys.
- The security of the cipher heavily depends on the choice of the S boxes.
 - If the S boxes were linear, the whole cipher would be linear, thus easily breakable.

- There is a known attack, called the *square attack*. It is based on a dedicated attack against Square, the cipher that Rijndael is based on, and that exploits the byte-oriented structure of the Square cipher.
- The attack is also valid for Rijndael, as Rijndael inherits many properties from Square.
 - If one byte is modified in the plaintext, then exactly 4 are modified after one round, and all the 16 are modified after two rounds.
 - The same property holds in the decryption direction.
 - Thus, a one-byte difference cannot lead to a one-byte difference after three rounds.
 - This observation can be used for an attack on 5 rounds of Rijndael.
 - Using a few more tricks, up to 7 rounds can be attacked.
- Although infeasible with current technology, on a 6 round cipher this attack is faster than exhaustive key search, and therefore relevant.
- 2^{32} plaintexts, 2^{72} cipher executions, 2^{32} memory.
- The best theoretic attack breaks up to 8 rounds with over 2^{120} complexity for 128-bit keys and 2^{204} for 256-bit keys.

The details

- We shall now look at the details of the functions in the round transformation.
- Let the 128-bit blocks be divided into 32-bit words, and each word be divided into bytes.
- We view the block as a square of 4 by 4 bytes, where each column has the bytes of a word.
- 128-bit keys are views in a similar way.
- Longer keys have additional columns, e.g. 256-bit keys.

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

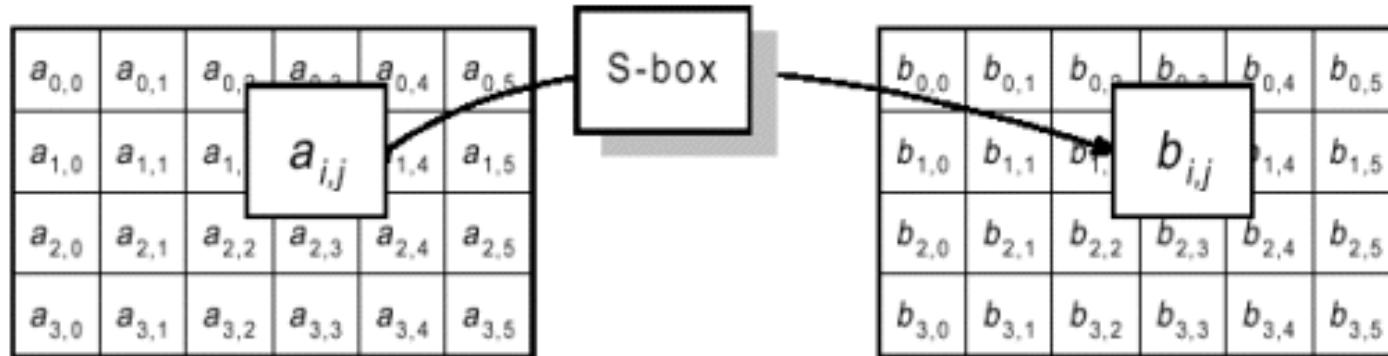
0	4	8	12	16	20	24	28
1	5	9	13	17	21	25	29
2	6	10	14	18	22	26	30
3	7	11	15	19	23	27	31

- Rjindael uses four invertible operations:
 1. Byte substitution (S-box, 8-bit to 8-bit).
 2. Shift row (rotating order of bytes in each row).
 3. Mix column (linear mixing of a word column).
 4. Key mixing (addition).

Byte substitution

- The byte substitution is a fixed S box from 8 bits (**x**) to 8 bits (**y**).
- It substitutes all the bytes of the input according to the following array:

```
y=S[x] = {  
    99, 124, 119, 123, 242, 107, 111, 197, 48, 1, 103, 43, 254, 215, 171, 118,  
    202, 130, 201, 125, 250, 89, 71, 240, 173, 212, 162, 175, 156, 164, 114, 192,  
    183, 253, 147, 38, 54, 63, 247, 204, 52, 165, 229, 241, 113, 216, 49, 21,  
    4, 199, 35, 195, 24, 150, 5, 154, 7, 18, 128, 226, 235, 39, 178, 117,  
    9, 131, 44, 26, 27, 110, 90, 160, 82, 59, 214, 179, 41, 227, 47, 132,  
    83, 209, 0, 237, 32, 252, 177, 91, 106, 203, 190, 57, 74, 76, 88, 207,  
    208, 239, 170, 251, 67, 77, 51, 133, 69, 249, 2, 127, 80, 60, 159, 168,  
    81, 163, 64, 143, 146, 157, 56, 245, 188, 182, 218, 33, 16, 255, 243, 210,  
    205, 12, 19, 236, 95, 151, 68, 23, 196, 167, 126, 61, 100, 93, 25, 115,  
    96, 129, 79, 220, 34, 42, 144, 136, 70, 238, 184, 20, 222, 94, 11, 219,  
    224, 50, 58, 10, 73, 6, 36, 92, 194, 211, 172, 98, 145, 149, 228, 121,  
    231, 200, 55, 109, 141, 213, 78, 169, 108, 86, 244, 234, 101, 122, 174, 8,  
    186, 120, 37, 46, 28, 166, 180, 198, 232, 221, 116, 31, 75, 189, 139, 138,  
    112, 62, 181, 102, 72, 3, 246, 14, 97, 53, 87, 185, 134, 193, 29, 158,  
    225, 248, 152, 17, 105, 217, 142, 148, 155, 30, 135, 233, 206, 85, 40, 223,  
    140, 161, 137, 13, 191, 230, 66, 104, 65, 153, 45, 15, 176, 84, 187, 22  
};
```



Each byte at the input of a round undergoes a non-linear byte substitution according to the following transform:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Substitution (“S”) -box

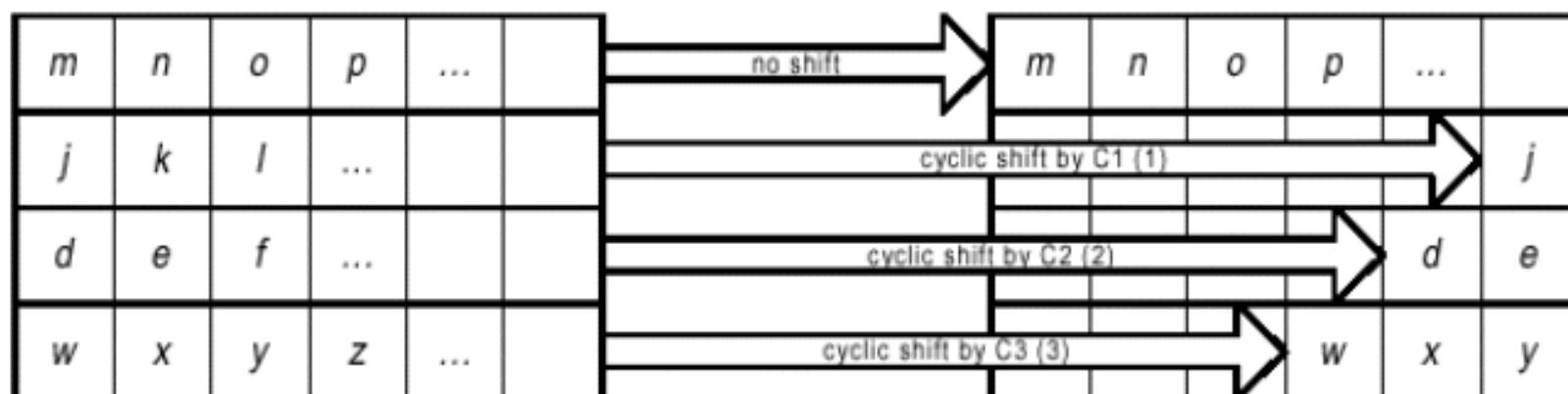
- For example: input byte $x=0 \rightarrow x_i=0, \forall i \rightarrow y_6=y_5=y_4=y_3=y_2=y_1=y_0=1 \quad y_7=y_6=y_5=y_4=y_3=y_2=y_1=y_0=0$

Shift row

- The Shift row operation rotates the order of bytes.

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

Depending on the block length, each “row” of the block is cyclically shifted according to the above table



Mix column

- Mix column is a linear mixing of a word column (four bytes) (a_0, a_1, a_2, a_3) into a word column (b_0, b_1, b_2, b_3). The mixing is defined by operations in the field $GF(2^8)$.
- It is equivalent to

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} f(a_0, a_1, a_2, a_3) \\ f(a_1, a_2, a_3, a_0) \\ f(a_2, a_3, a_0, a_1) \\ f(a_3, a_0, a_1, a_2) \end{pmatrix}$$

where

$$f(a_0, a_1, a_2, a_3) = (a_1 \oplus a_2 \oplus a_3) \oplus g((a_0 \oplus a_1) \ll 1)$$

$$g(x) = (x \& 100_x) ? (x \oplus 1b_x) : x$$

The operations \ll , $\&$, $?$: are left shift, logical AND, and conditional expressions, as used in C. The subscript x's are used to indicate hex.

- The left shift is multiplication by 2, and may turn the 9th bit on. The conditional XOR with $1b_x$ in the $g()$ function ensures that the result always fits in a byte. ($g()$ enforces the field $GF(2^8)$ modulo the polynomial denoted by $1b_x$, that polynomial being $x^4 + 1 = 2^9$ in $GF(2^8)$).

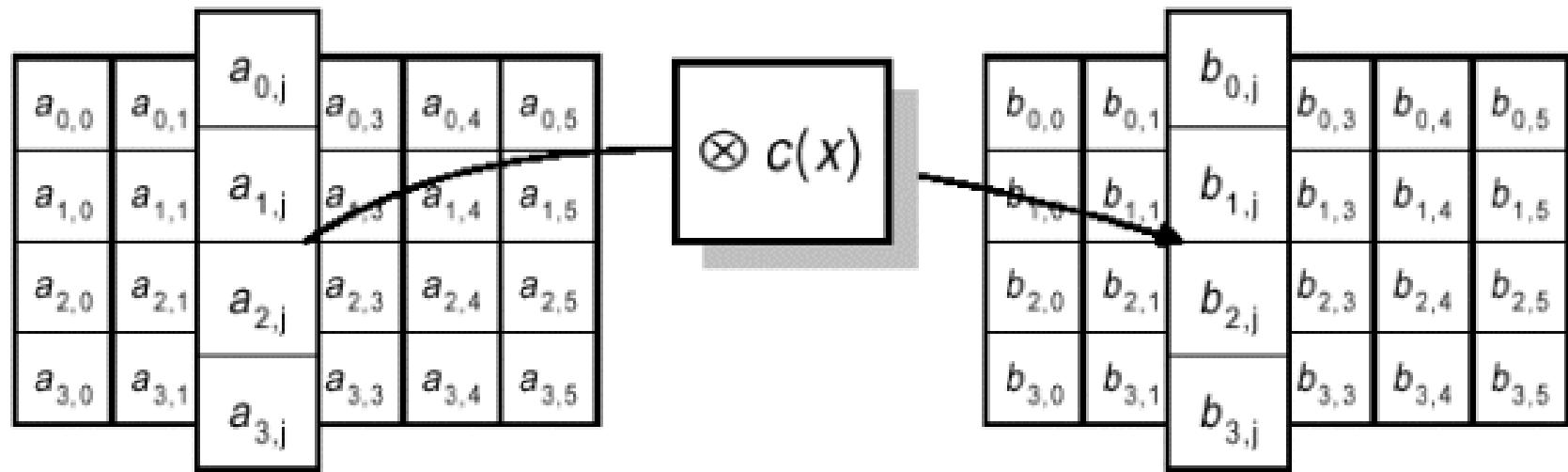
- The **inverse** of the Mix Column operation is also linear, and takes the same form, but with the f replaced by d functions where

$$d_1(a_0, a_1, a_2, a_3) = a_0 \oplus a_1 \oplus a_2 \oplus a_3$$

$$d_2(a_0, a_1, a_2, a_3) = g(d_1(a_0, a_1, a_2, a_3) \ll 1) \oplus (a_0 \oplus a_2)$$

$$d_3(a_0, a_1, a_2, a_3) = g(d_2(a_0, a_1, a_2, a_3) \ll 1) \oplus (a_0 \oplus a_1)$$

$$d_4(a_0, a_1, a_2, a_3) = g(d_3(a_0, a_1, a_2, a_3) \ll 1) \oplus (a_1 \oplus a_2 \oplus a_3)$$



Each column is multiplied by a fixed polynomial
 $C(x) = '03'*X^3 + '01'*X^2 + '01'*X + '02'$

This corresponds to matrix multiplication $b(x) = c(x) \otimes a(x)$:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Key Mixing

- The key mixing operation XORs the data with a subkey. The subkey has the same size of the blocks.
- The subkeys are computed from the key by a key scheduling algorithm, which we discuss later.

$$\begin{array}{|c|c|c|c|c|c|} \hline a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & a_{0,4} & a_{0,5} \\ \hline a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ \hline a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|c|c|} \hline k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} & k_{0,4} & k_{0,5} \\ \hline k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} & k_{1,4} & k_{1,5} \\ \hline k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} & k_{2,4} & k_{2,5} \\ \hline k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} & k_{3,4} & k_{3,5} \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|} \hline b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} & b_{0,4} & b_{0,5} \\ \hline b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} & b_{1,5} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} & b_{2,5} \\ \hline b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} & b_{3,5} \\ \hline \end{array}$$

Each word is simply EXOR'ed with the expanded round key

Encryption

- Let us recap on the order of the operations for encryption.
- Encryption is performed by
 1. First key mixing using sub-key K_0 .
 2. N_r rounds consisting of
 - a) Byte substitution
 - b) Shift row
 - c) Mix column (except for the last round)
 - d) Key mixing using sub-key K_i in round $i \in \{1, \dots, N_r\}$.

Decryption

- Decryption applies the inverses of the operations in the reverse order, using the reverse order of the sub-keys.
- Decryption requires the application of the inverse S boxes, inverse Mix column, and inverse shift rows.
- The Key mixing/adding is the same operation.

Key scheduling

- The key schedule takes an N_k -word key ($32N_k$ -bits) and computes the N_r+1 sub-keys of a total size of $4(N_r+1)$ words.
- Let W be a word array of size $4(N_r+1)$. We let the first N_k words be initialized by the key. Then we compute

```
for i :=  $N_k$  to  $4*(N_r+1)-1$  do {  
    temp =  $W[i-1]$ ;  
    if ((i% $N_k$ ) == 0)  
        temp = byteSubstitution(temp<<8) ^ RCON[i/ $N_k$ ];  
     $W[i] = W[i-N_k] \wedge temp$ ;  
}
```

where RCON is fixed as the array of powers of 2 in GF(2₈). The array is longer than needed in practice.

RCON[30] = {

01_x, 02_x, 04_x, 08_x, 10_x, 20_x, 40_x, 80_x,
1b_x, 36_x, 6c_x, d8_x, ab_x, 4d_x, 9a_x, 2f_x,
5e_x, bc_x, 63_x, c6_x, 97_x, 35_x, 6a_x, d4_x,
b3_x, 7d_x, fa_x, ef_x, c5_x, 91_x};

Then, the first four words of W become the subkey K₀, the next 4 words become K₁, etc.

Efficient implementations

- Efficient implementations of Rijndael combine the S boxes, Shift Rows and Mix Column operations together.
- The Mix Column operation can be performed by four table lookups (8-bit indices, 32-bit output), and XORing the results.
- The Shift row operation can be eliminated by carefully indexing the indices of the input bytes to the succeeding Mix Column operation.
- As the S boxes are also implemented as table lookups, the three operations together can be implemented by 16 table lookups in total (and 12 XORs), using 4 precomputed tables of

Mix Column(Shift Row(S boxes(.)))

CSCI361

Computer Security

Modern stream ciphers

Outline

- Stream ciphers.
- RC4.
- Others:
 - A5/1 & A5/2.
 - SEAL.
- eStream.

Stream ciphers

- In block ciphers plaintext characters are grouped in blocks and then each block is encrypted.
- In stream ciphers, characters are encrypted one at a time.
 - For example the Vigenère cipher.
 - Note that characters could themselves be a block of bits (in the sense the algorithm operates byte by byte say).
- We are now going to look at some modern stream ciphers.

RC4

- Developed by Rivest (1987) this is one of the best known, and most widely used, stream ciphers.
- It effectively acts as a pseudo-random number generator, generating a key-stream which is xor'd with the plaintext.
- Decryption and encryption operations are the same.
- Variable sized secret key up to 256 bytes.
- RC4 operates on bytes.
- Used in the WEP protocol, SSL/TLS
 - Wired Equivalent Privacy (Wireless)
 - Secure Sockets Layer/Transport Layer Security (Web browsers ↔ servers)

Initialisation

- Initialisation for key K of length p .

for $i = 0 \dots 255$

$S[i] = i$

for $i = 0 \dots 255$

$j = (j + S[i] = K[i \bmod p]) \bmod 256$

 swap($S[i], S[j]$)

Encryption/decryption

- Byte by byte processing.

i = 0

j = 0

loop until all message encrypted

 Get the next input byte

 i = (i + 1) mod 256

 j = (j + S[i]) mod 256

 swap(S[i],S[j])

 k = S[(S[i] + S[j]) mod 256]

 output: XOR k with input byte

Attacks against RC4

- Like all stream ciphers, RC4 is easily broken if a key is used twice.
 - Weaknesses in WEP due to key re-use.
- There is a need to discard the first outputs (1024 bytes) of the keystream.
 - The statistics of first bytes are non-random.
- Theoretical breaks may be possible if gigabytes of (plaintext, ciphertext) is available.
 - This is not usually a problem in practice.
- RC4 is safe for use in some existing applications, but isn't recommended for use to new applications. It doesn't meet various standards and will be phased out.

A5/1 and A5/2

- A5/1 was designed to provide voice privacy for GSM (Global System for Mobile Communications) cellular phones. A5/2 is a weaker version, designed for use in telecommunication environments where A5/1 couldn't be used.
- A5/1 has a 64-bit key.
- A5/2 is weak.
- A5/1 is severely broken.

SEAL

- Software Optimized Encryption Algorithm.
- Rogaway and Coppersmith (1994).
- This is a very fast stream cipher, which is optimized for machines with 32-bit architecture and lots of RAM.

Others

- FISH → Broken.
- Pike (from FISH and A5) → Okay.
- ISAAC (**I**ndirection, **S**hift, **A**ccumulate, **A**dd, **C**ount) (RC4 inspired) → Attack 4.67×10^{1240} instead of 10^{2466} . ☺
- Note that while stream ciphers are closely tied with pseudo-random number generators, the properties for a good pseudo-random number generator do not always coincide with good cryptographic properties.

eStream

- This is a project, undertaken in the European Union ECRYPT network to develop a stream cipher standard.
- It is somewhat similar to the AES development procedure, but some improvements have been made to the process.
- Submissions were called for November 2004 and it is due to be completed January 2008.
- There are basically two classes: software and hardware, although some of the ciphers are for both
- The website for the project is <http://www.ecrypt.eu.org/stream/>
- Progress is also recorded at
<http://en.wikipedia.org/wiki/ESTREAM>

eStream Result

Profile 1 (SW)

HC-128

Rabbit

Salsa20/12

SOSEMANUK

Profile 2 (HW)

Grain v1

MICKEY v2

Trivium

New types of attacks

- Fault analysis attacks

CSCI361

Computer Security

Message Integrity

Outline

- What is integrity?
- Message integrity.
- Message authentication codes (MAC).
- Unconditionally secure MAC.
- MAC based on block ciphers.

What is integrity?

- Integrity is both *assurance* that the content of the message has not been altered during transmission, and a mechanism for *detecting* if a message has been altered during transmission.

Message Integrity

- We distinguish two types of threats against the integrity of a message.
 1. **Un-intentional** due to noise:
 - Protection is provided by *error detecting/correcting codes*.
 - For example, the parity bit in the ASCII code is for error detection. (8 bits to represent 7-bits of information).
 2. **Intentional** in which an active spoofer tries to *modify* a message or *generate* a false message.
 - Protection in this case is provided by **Message Authentication Codes** (MAC).

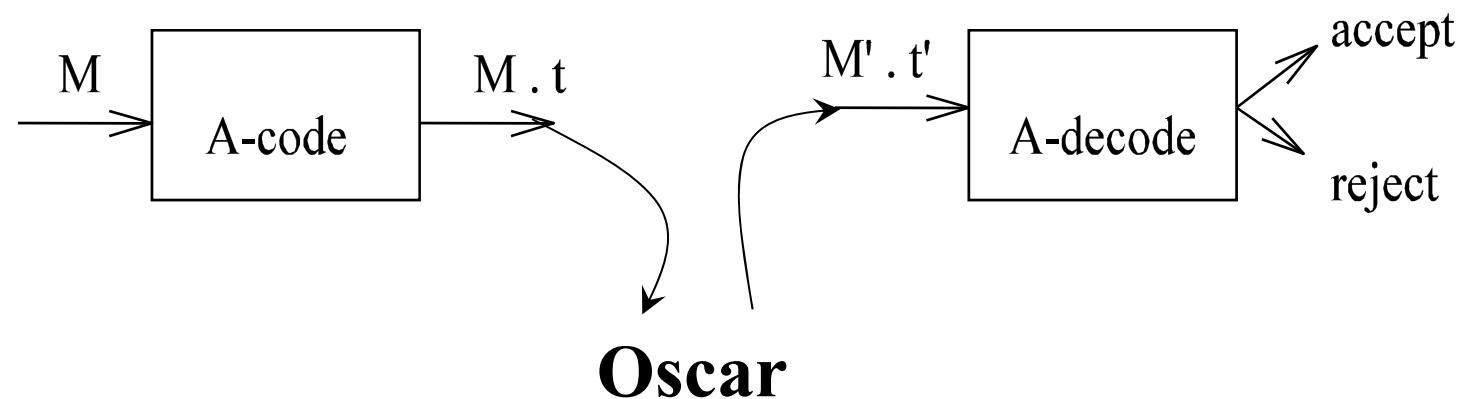
- One common mechanism of implementing protection against either threat is to append to the message a *tag* or *checksum*. Such a tag is a string of bits, in a binary representation.



Error detection: An example

- Algorithms for error detection do not need a secret key. Consider the following example:
- An algorithm for generating parity bits:
 - Consider the message is a block of 8 bits, or, equivalently, an integer in the range 0-255.
 - We attach three parity bits to each block by finding the remainder of the integer divided by 7.
- Consider the message 01111101=125.
The parity bits are calculated as $125 \bmod 7 = 6 = 110$.
So we send the block 01111101 110.
- If one bit error occurs, for example the receiver gets 01011101 110, they detect the error as follows:
 $01011101 = 93 \bmod 7 = 2 = 010 \neq 110$

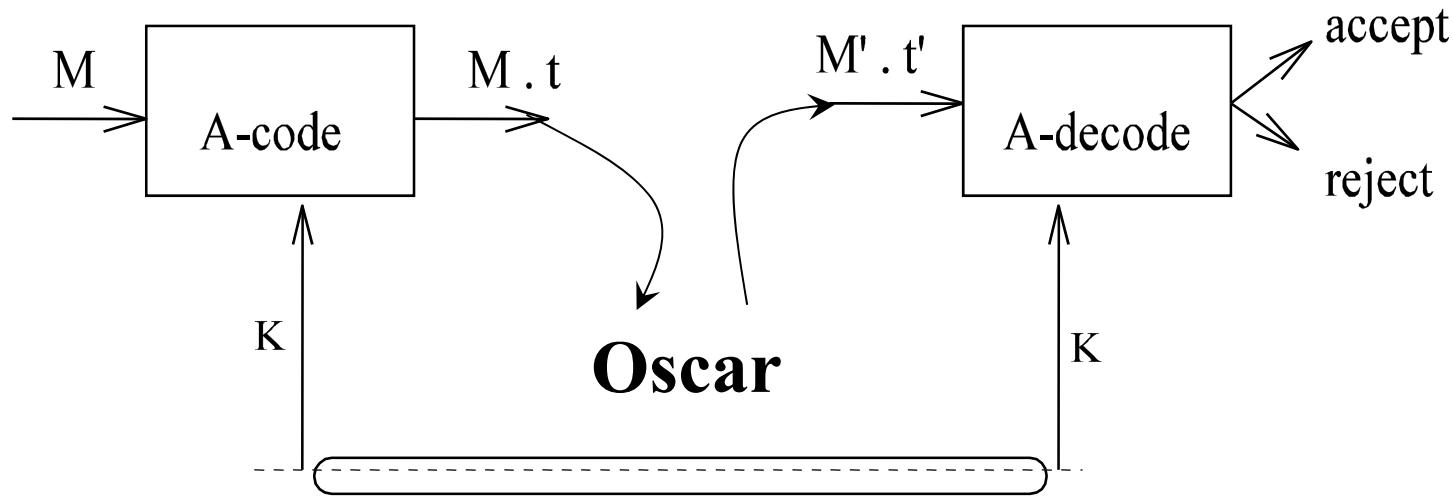
- Error-correcting/detecting codes do not provide protection against spoofing, i.e. against deliberate modification or impersonation attacks.



- Nor does encryption by itself.

Message Authentication Codes

- Transmitter and receiver share a secret key **K**. To transmit **M**, the transmitter calculates a **MAC** and appends it to **M**, thus ($M.t=MAC_K(M)$).



- The receiver receives a message (**M.X**). It uses the key **K** and **M** to calculate $MAC_K(M)$ and compare it with **X**. If the two match, the received message is accepted as authentic.
- The **MAC** is also called a **cryptographic checksum**.

- A MAC *is secure* if forging $(M, \text{MAC}_K(M))$, i.e. modifying it without being detected, is hard.
- We can construct a **MAC** with either **unconditional security** or with **practical security**.
- A MAC with unconditional security requires many key bits. Unconditional security in this case means that the chance of success of the enemy is always less than 1.

An unconditionally secure MAC

- Suppose our message consists of a number between 0 and $p-1$, where p is a prime.
- Any such message can be broken into blocks of size $\log_2 p$.
- The transmitter and the receiver choose/establish/exchange a key \mathbf{K} which is a pair of numbers; $\mathbf{K}=(a,b)$ where $0 \leq a, b \leq p-1$.
- Then

$$\text{MAC}_{\mathbf{K}}(\mathbf{M})= a \mathbf{M} + b \bmod p$$

- The enemy observes $(\mathbf{M}, \text{MAC}_{\mathbf{K}}(\mathbf{M}))$, but without knowing \mathbf{K} cannot change \mathbf{M} and recalculate its cryptographic checksum.
- This is an example of an Authentication-code (A-code).

- Example: $p=11$, $K=(a,b)=(2,3)$.
- To find the **MAC** for a message 5, we calculate

$$MAC_K(5) = 2*5 + 3 \text{ mod } 11 = 2$$
- The enemy doesn't know **K**, but does know $a*5 + b \text{ mod } 11 = 2$.
- For an arbitrary choice of 'a' (11 values) the enemy can calculate a corresponding value for 'b'. So the possible keys are $(0,2), (1,8), (2, 3), \dots$
- So the chance of correctly guessing the key is $1/11$.
- The tag is always one of the 11 elements.
- If the enemy receives another message **M'** whose MAC is calculated using the same key; say $M'=7$ with $MAC_K(7)=5$, then the enemy can solve the following equations:

$$a*5 + b \text{ mod } 11 = 2 \quad a * 7 + b \text{ mod } 11 = 6$$

to find the key **K=(2,3)**.
- Thus the key must be changed with every message.

- In general it can be proven that:

Unconditionally secure authentication systems require a large amount of key amount and so are impractical.

- **Example:** Design a MAC system with unconditional security such that the chance of success for an intruder is $P \leq 0.01$.
- Let $p=101$, and $K=(a,b) = (5,11)$.
- A message is an integer in the range 0-100; that is it requires 7 bits to represent it. The key is 14 bits long. The transmitted message is 14 bits long.

$$\text{MAC}_K(90) = 5 * 90 + 11 \bmod 101 = 0111001.$$

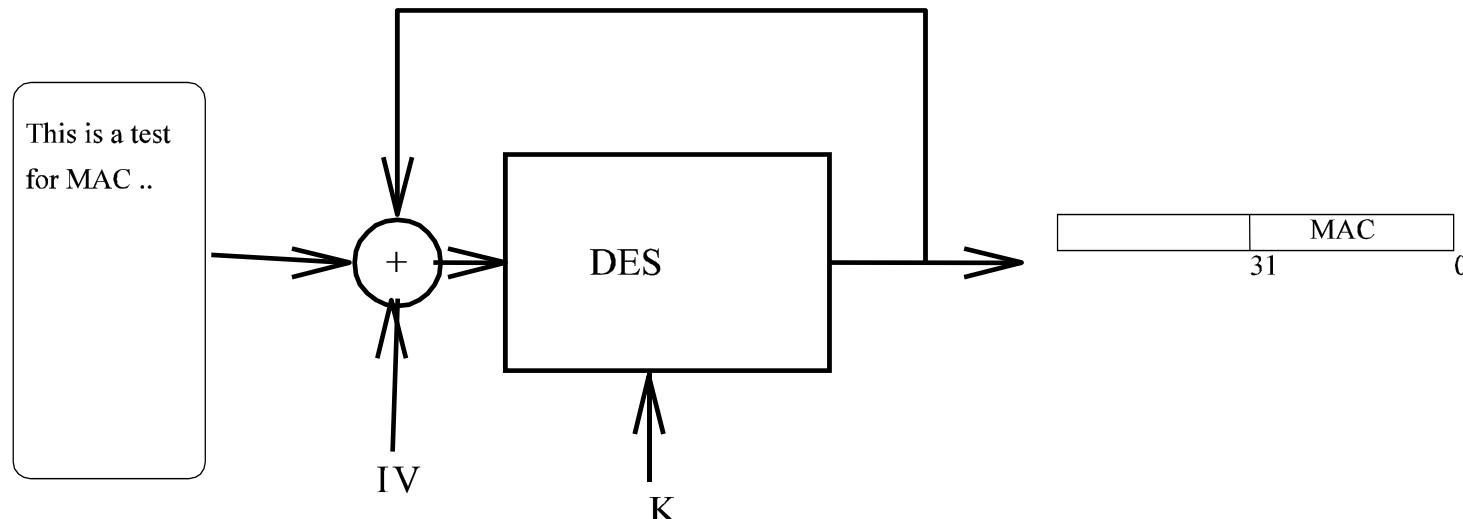
So the message sent is 1011010 **0111001**.

The chance of the enemy's success is $P=1/101 < 1/100$.

- We see that this MAC is very inefficient:
 - In terms of the number of bits appended to the message
 - In terms of the amount of key information, especially when we consider it requires a different key for every message.
- Essentially this is analogous to the one-time pad in encryption.
 - Recall that system is unconditionally secure encryption.
 - The key information required therein equals the information in the message to be transmitted.
- In a MAC with *practical security* it is always possible to forge a message, but the amount of computation required to do so is large, to the extent it is impractical for the attacker to do so.

Block cipher based MAC

- A block encryption algorithm in CBC (chain-block cipher) mode can be used to generate a checksum. Remember in CBC mode the ciphertext gets “feed” back through the encryption algorithm.
- **Example:** Consider a 32-bit MAC generated using CBC mode of DES. For an arbitrary length message M, $\text{MAC}_K(M)$ is the right 32 bits of the output of the algorithm.



Attacking block based MAC

- Question One: What is the cost/chance of the enemy successfully forging a message?
- Attack 1:
 - Enemy uses an exhaustive key search to find the key.
 - Then modifies the message as desired.
 - Then calculates the MAC for the new message using the key.
- The cost something like 2^{64} operations.

- Attack 2:

- The enemy chooses any message.
 - The enemy randomly selects a 32-bit string and attaches it to the message.

- The chance of the enemy succeeding is $\geq 1/2^{32}$.

- Question Two: Is the enemies cost/chance of success changes by using triple-DES?
- It is more difficult to find the key but the answer is **no**, because the best attack is NOT finding the key of the block cipher algorithm.

- Question 3: How can we reduce enemy's chance of success using DES in CBC mode then?
- Increase the size of the tag: use a 64 bit tag.

CSCI361

Computer Security

Public Key Cryptography I
Introduction and knapsacks

Outline

- Drawbacks of symmetric key crypto.
- Public Key Cryptography (PKC).
- Assessing security.
- One-way trapdoor functions.
- Knapsacks.
 - For encryption.
 - Super-increasing knapsacks.
 - Trapdoor knapsacks.
- Finding inverses, GCD's.
- The Euclidean and Extended Euclidean Algorithms.

Drawbacks of symmetric key crypto

- So far we have studied **symmetric key cryptosystems** where the transmitter and receiver have the same key.
- They are also called
 - Private key cryptosystems
 - Secret key cryptosystems
 - Conventional cryptosystemsbut we will generally stick to the term symmetric.
- **Key management**, that is, generating good keys, distributing and storing them in a secure way, is a bottleneck in symmetric key cryptography.
- **Example:** For a network of N computer terminals, with pairwise secret keys, the total number of secret keys is $N(N-1)/2$. For $N=100$ there are 4950 keys, with 99 at each terminal and two copies of each across the network.

- Another significant drawback of symmetric key cryptography is that the encryption is not identified with an individual, it doesn't provide **authenticity**. We will later see they are not they cannot directly provide what are referred to as *digital signatures*. Services such as **non-repudiation** cannot be achieved.
 - A **non-repudiation service** provides Alice protection against Bob later denying that some communication exchange took place. It allows the elements of the communication exchange to be linked with the transmitter.
 - In such a service Alice will have irrefutable evidence to support her claim.
- In symmetric key cryptography the secret information is shared between Alice and Bob, so whatever Alice can do Bob can do too.
 - To provide a non-repudiation service a **trusted authority** is required.

Public key cryptography

- Diffie-Hellman (1975).
- **Public Key Cryptography (PKC)** provides solutions to both of the problems mentioned.
- In a PKC:
 - The encryption and decryption keys are different.
 - The decryption key cannot be deduced from the encryption key.
- That is, keys come in pairs, (z, Z) , where z is the private and Z is the public component of the key.

- Each pair of keys satisfies the following two properties:
 - A plaintext encrypted with Z can be decrypted with z . That is z determines the inverse of the encryption with key Z .
 - Given a public key Z it is *computationally infeasible* to discover the corresponding decryption key z .
- The first property is needed in all cryptosystems: Decryption is the inverse of encryption.
$$D_z(E_z(X)) = X$$
- In secret key cryptography z can be easily obtained from Z .
- **Example:** In DES decryption sub-keys are the same as encryption sub-keys but applied in reverse order.
- Once Z is known, z can be calculated and algorithms E_z and D_z are both known. In PKC, knowledge of E_z does not imply knowledge of D_z .

- A PK cryptosystem can provide *confidentiality*, because only the receiver can decrypt and find the plaintext, and *authenticity*, because only the sender can create such a cryptogram.
- In a PKC, user Alice has a pair of keys.
 - *Public component* Z_A generates a public transformation E_{Z_A} for encryption.
 - *Private component* z_A generates a private transformation D_{z_A} for decryption.

- A public directory contains a list of public keys, including a row for Alice.

Alice	Z_A
Bob	Z_B
Fred	Z_F
Oscar	Z_O
...	
...	

- If **Bob** wants to send a message **X** to **Alice**, he checks the public directory to find Alice's public key, and forms the following cryptogram:

$$Y = E_{z_A}(X)$$

- **Alice** can recover the message using her private key, as in

$$\begin{aligned}D_{z_A}(Y) &= D_{z_A}(E_{z_A}(X)) \\&= X\end{aligned}$$

Assessing security in PK systems

- Since E_z is public the enemy can choose a plaintext and find the corresponding ciphertext. Hence a **PKC should be assessed under chosen-plaintext attack.**
- How can we construct a PKC which is resistant to such an attack?
 - PKC's can be realized by using **trapdoor one-way** functions.

One-way trapdoor functions

- A function f is called **one-way** if
 - for all X finding $f(X)$ is easy.
 - knowing $f(X)$ it is hard to find X .
- **Example:** Given n primes p_1, p_2, \dots, p_n , it is easy to find their product

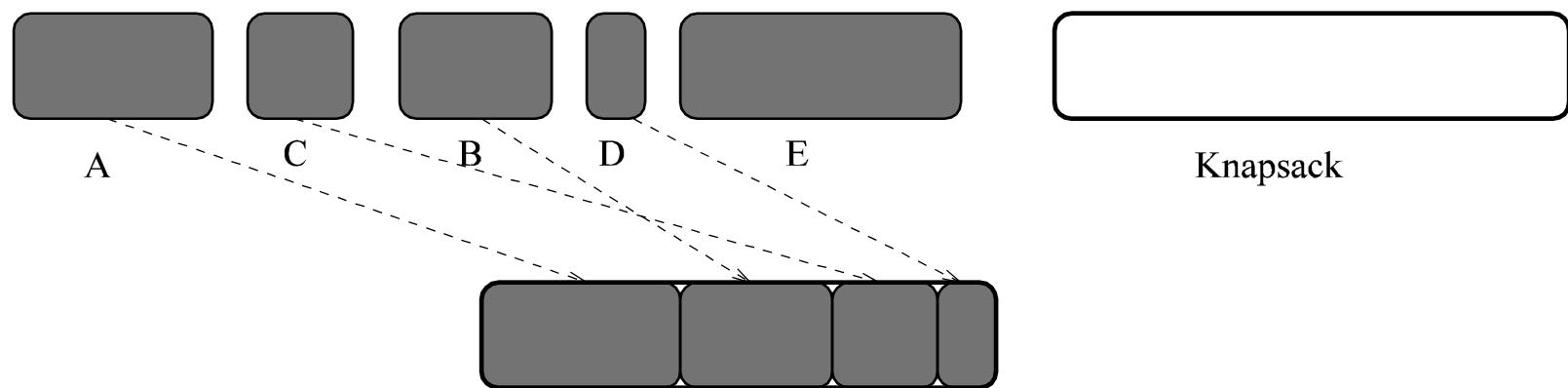
$$N = p_1 p_2 \dots p_n.$$

- Given a large number N it is difficult to find its prime factors.
- A **trapdoor** is a piece of knowledge which makes it easier to find X from $f(X)$.
- A **trapdoor one-way function** is a function which looks like a one-way function but is equipped with a secret **trapdoor**. If this secret door is known, the inverse can be easily calculated.

- A trapdoor one-way function can be used to realize a PKC:
- E_z is easy to compute (from the public component of the key Z) but is hard to invert.
- Knowledge of z , which is the private component of the key (i.e. it is the trapdoor), allows easy computation of D_z (inverting E_z).
- We are going to look at two examples of building trapdoors in one-way functions. The first one has failed (knapsacks) and the second one has provided us with the most important PK cryptosystem (RSA).

Knapsacks

- The **knapsack problem** is derived from the notion of packing an odd assortment of packages into a container:



- Find a subset of packages that fills the container.*
- In general, there is no efficient way of finding the subset: You have to try all possibilities.

- Alternatively:
 - We are given an ordered set of n positive integers a_i , $1 \leq i \leq n$, and a target number T .
 - The ordered set of positive integers is referred to as a *cargo vector*.
 - The knapsack problem is to find a subset $S \subseteq \{1, \dots, n\}$ such that
- This is also called the **subset-sum problem**. This is known to be a *difficult* problem as there is no algorithm more efficient than exhaustive search in the general case.
- **Example:** $(a_1, a_2, a_3, a_4) = (2, 3, 5, 7)$ and $T=7$.
Solutions: $(1, 0, 1, 0)$, $(0, 0, 0, 1)$.
- Note: In general we may have more than one solution.

Knapsacks for encryption

- Knapsacks are implemented as **block ciphers**. For a block size of n bits we require a **key** or **cargo vector**, of positive integers a_i , $1 \leq i \leq n$, referred to as weights.
$$\mathbf{a} = (a_1, a_2, a_3, \dots, a_n)$$

- Then to encrypt a message block of n bits:

$$\mathbf{X} = (x_1, x_2, x_3, \dots, x_n)$$

we find the number

$$T = \sum_{i=1}^n x_i a_i$$

and write it in a binary representation.

- **To decrypt:** The receiver knows T and all a_i , so goes through all possible plaintext blocks to find a block that satisfies the condition.
- In this system:
 - **Encryption is easy** and requires n additions.
 - **Decryption is difficult**, even if the key is known.
- Hence the resulting cryptosystem is not useful.
- To construct an acceptable cryptosystem, Merkle & Hellman (1977) used a special case of the knapsack problem, involving *super-increasing knapsacks*, where decryption is easy with the key.

Super-increasing knapsacks

- A knapsack is super-increasing if each element of the cargo vector is greater than the sum of the preceding elements.
- **Example:** $a=(1,2,4,8)$
Given $T=14$ say, it is easy to find $X=(x_1,x_2,x_3,x_4)$ such that $x_1+2x_2+4x_3+8x_4=14$.

i	a_i	Total	In?
4	8	14	1
3	4	6	1
2	2	2	1
1	1	0	0

$X=0111$

- At each step the current target value with the largest unchecked element a_i of the cargo vector
 - a. If the current target value is larger than that element of the cargo vector must be included element (i.e. $x_i=1$) otherwise it cannot be (i.e. $x_i=0$).
- Example: $\mathbf{a}=(171, 197, 459, 1191, 2410)$, $T=3798$.
- What is $X=(x_1, x_2, x_3, x_4, x_5)$?

i	a_i	Total	In?
5	2410	3798	1
4	1191	1388	1
3	459	197	0
2	197	197	1
1	171	0	0

$X=01011$

- As described the super-increasing knapsacks could be used as a symmetric key cryptosystem.
- But super-increasing knapsacks cannot be used directly for public key cryptography, because making the encryption key public *allows everyone to decipher X!*
- The idea of Merkle and Hellman (1977) was to start with an easy knapsack and then *disguise* it to *look difficult* for people without knowledge of the trapdoor.
- The trapdoor will only be known by the person who wants to decrypt the cryptogram. It will be their private key.

Trapdoor knapsacks

- Alice chooses a super-increasing knapsack, $a=(a_1, a_2, \dots, a_n)$, as her **private key**.
- Then she chooses an integer $m > \sum_{i=1}^n a_i$, called the **modulus**, and a random integer w , called the **multiplier**, which is relatively prime to m . Relatively prime means the largest common factor of w and m is 1. This condition implies there exists an inverse of w **modulo** m .
 - Note that if m is prime, then w can be any number.
- Alice's public key is b where
 $b=(b_1, b_2, \dots, b_n)$ and $b_i=w^*a_i \bmod m$

- Alice publishes a permuted version of \mathbf{b} as her public key. Her secret key is $(\mathbf{a}, m, \mathbf{w})$.
- **To encrypt:** Bob sends a message \mathbf{X} to Alice by computing

$$T = \sum_{i=1}^n x_i b_i$$

- **To decrypt:** Alice receives T and ...
 - Uses the inverse of \mathbf{w} , \mathbf{w}^{-1} , to calculate
 $R = \mathbf{w}^{-1} T \bmod m$.
 - Uses the easy knapsack \mathbf{a} to find \mathbf{X} , since $\mathbf{R} = \mathbf{a} \cdot \mathbf{X}$

$$R = w^{-1}T(\bmod m)$$

$$= w^{-1} \sum_{i=1}^n b_i x_i (\bmod m)$$

$$= w^{-1} \sum_{i=1}^n (w a_i) x_i (\bmod m)$$

$$= \sum_{i=1}^n (w^{-1} w a_i) x_i (\bmod m)$$

$$= \sum_{i=1}^n a_i x_i (\bmod m)$$

$$= a.X$$

Brute force attack

- For those who do not know the secret trapdoor, decryption requires an exhaustive search through all 2^n possible \mathbf{X} .
- **Example:**
- Earlier we had: $\mathbf{a}=(171, 197, 459, 1191, 2410)$
- Let $w = 2550$ and $m = 8443$.
 $\mathbf{b}=(5457, 4213, 5316, 6013, 7439)$ is the public cargo vector.

Multiple layers and the fall of knapsacks

- The disguising process can be repeated a number of times on the cargo vector to create more and more difficult knapsack problems (w_1, m_1) , (w_2, m_2) , The result is, in general, not equivalent to a single (w, m) transformation.
- Adleman (1982) broke the knapsack cryptosystem by taking a public cargo vector and finding a pair (w', m') that would convert it back to a super-increasing cargo vector sufficient for decrypting encrypted messages. It didn't have to be the same one used as the private key.
- Merkle was confident enough that the multiple layers were still secure to offer a reward of \$1000 to anyone who could break a multiple layer knapsack.
- In 1984, Brickell announced the destruction of a knapsack system, with 40 iterations and a hundred weights (elements of the cargo vector), in about one hour of Cray-1 time.
 - Merkle gave him the money ☺

Finding inverses:

The Extended Euclidean Algorithm

- Constructing trapdoors in knapsacks requires finding inverses of **w modulo m**.
- Inverses exists if **w** and **m** do not have any common factor.
- To find **w⁻¹** such that **w⁻¹w = 1 mod m** we will use the Extended Euclidean Algorithm.
 - Before doing so it is instructive to look at the Euclidean algorithm.

GCD's and the Euclidean Algorithm

- The *greatest common divisor* (GCD) of two integers n_1 and n_2 , not both zero, is the largest integer that divides n_1 and n_2 .
- It is denoted $\gcd(n_1, n_2)$.
- **Example:** $\gcd(30, 15) = 15$
 $\gcd(30, -12) = 6$
- We can calculate the gcd using an algorithm of Euclid.

Euclidean Algorithm

- 1) Divide the larger number by the smaller and retain the remainder.
 - 2) Divide the smaller original number by the remainder, again retaining the remainder.
 - 3) Continue dividing the prior remainder by the current remainder until the remainder is zero, at which point the last (non-zero) remainder is the greatest common divisor.
- **Example:** gcd(84,49).

$84/49 \rightarrow$ remainder 35.

$49/35 \rightarrow$ remainder 14.

$35/14 \rightarrow$ remainder 7.

$14/7 \rightarrow$ remainder 0.

Therefore $\text{gcd}(84,49)=7$.

Extended GCD for integers

The Extended GCD Theorem for Integers states:

Given integers n_1 and n_2 , not both zero, there exist integers a and b such that

$$\gcd(n_1, n_2) = a * n_1 + b * n_2$$

- These integers are not necessarily unique though.
- **Example:**

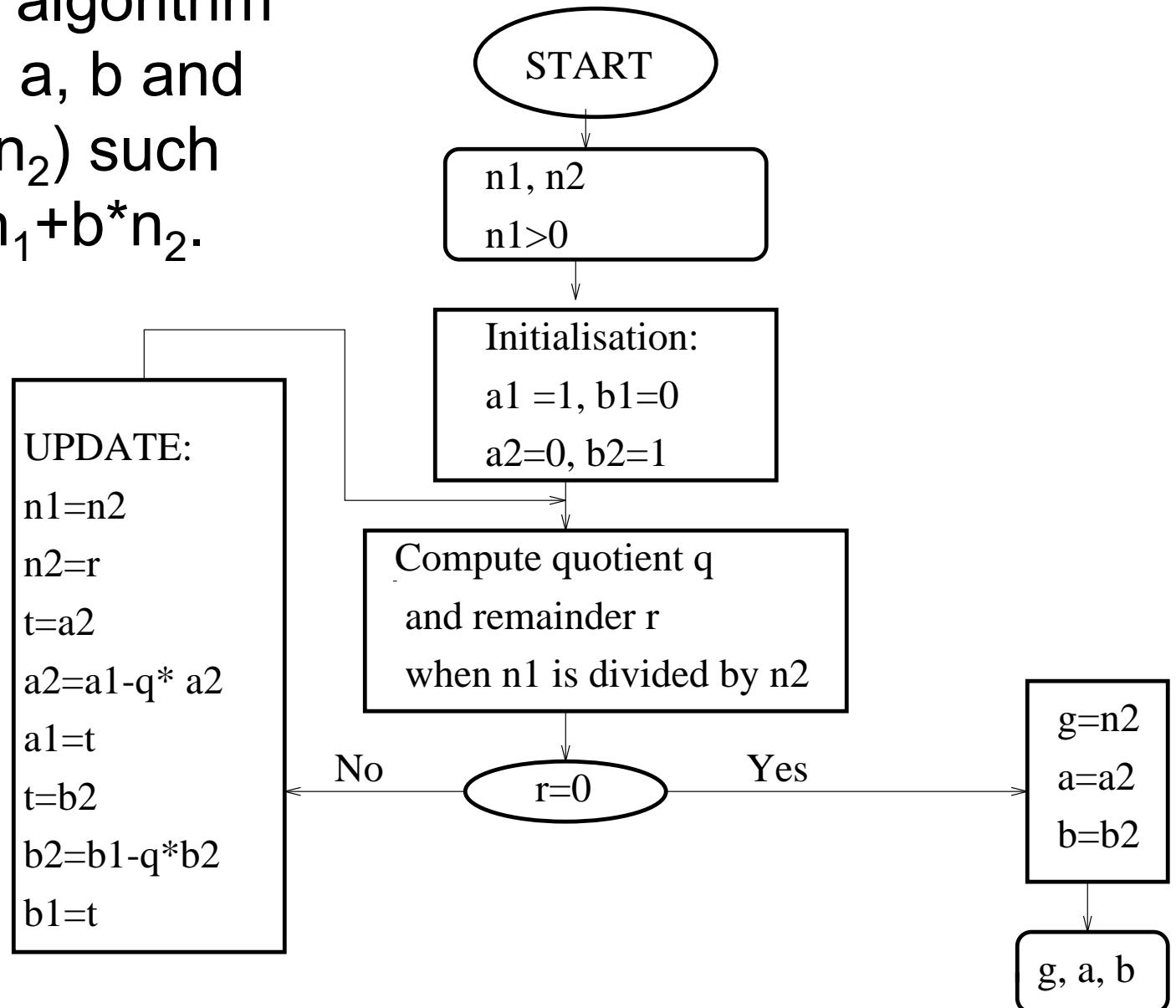
$$\begin{aligned}\gcd(15, 12) &= 3 = (+1)*15 + (-1)*12 \\ &= (+1-12)*15 + (-1+15)*12 \\ &= (-11)*15 + (+14)*12\end{aligned}$$

If $\gcd(w, m) = 1$ then it means that we can find the inverses $n_1 \bmod n_2$ and $n_2 \bmod n_1$.

$$\gcd(n_1, n_2) = a * n_1 + b * n_2 = 1$$

- **Example:**
- $$\begin{aligned}\gcd(65, 14) &= 1 = (-3)*65 + (14)*14 \\ \rightarrow 14*14 &= 1 \pmod{65}\end{aligned}$$

■ The Extended Euclidean algorithm calculates a , b and $g=\gcd(n_1, n_2)$ such that $g=a*n_1+b*n_2$.



Find $\gcd(39,11)$ and $a,b : 39a+11b=\gcd(39,11)$

Initialise

n_1	n_2	r	q	a_1	b_1	a_2	b_2
39	11	6	3	1	0	0	1
11	6	5	1	0	1	1	-3
6	5	1	1	1	-3	-1	4
5	1	0	5	-1	4	2	-7

$$\gcd(39,11)=1$$

$$1=39*2+11*(-7)$$

Example: PKC using a trapdoor knapsack

- Secret key $\mathbf{a}=(2, 5, 10, 21)$.
- Trapdoor: $m=39 > 38$ (sum of weights).
 - Random w , $w=15$.
 - gcd algorithm, $\gcd(39, 15)=3 \neq 1$
 - Another w , $w=11$.
 - $\gcd(39, 11)=1$
 - Inverse of $11 \bmod 39 = 32$
 $(32 = -7+39)$

- Using $w=11$ and $m=39$ disguise a .

$$b_1 = 2 \cdot 11 = 22 \pmod{39}$$

$$b_2 = 5 \cdot 11 = 16 \pmod{39}$$

$$b_3 = 10 \cdot 11 = 32 \pmod{39}$$

$$b_4 = 21 \cdot 11 = 36 \pmod{39}$$

- Public key: $\mathbf{b}=(22, 16, 32, 36)$.

- Secret key: $\mathbf{a}=(2, 5, 10, 21)$.

$$(w, m) = (11, 39)$$

(remember also that $w^{-1}=32$)

- To decrypt a message $T=48\dots$
 - Find $R = 48 * 32 \bmod 39 = 15$
 - Use the cargo vector \mathbf{a} to decrypt R and find $X=(0, 1, 1, 0)$.
 - We see this agrees with the public key version too ($16+32=48$).
 - In useful sized examples it would be not trivial to find the solution using just the public key, although it is easy to check the answer using it.

PKC progress...

- Since 1976, many PKC have been proposed. Many of these are broken and proven to be insecure.
- Among the ones that are considered secure, many are impractical because of either large key or large message expansion.
- PKC algorithms are, however, all slow and unsuitable when fast encryption is needed. However, they can provide high security and can be used when low processing rates are acceptable or when high security is needed.
- Remember that a PKC can be used for:
 - confidentiality (secrecy).
 - authentication (digital signatures).
- Two algorithms that can easily be used for both secrecy *and authentication* are RSA and ElGamal.
 - We are going to look at RSA next.

CSCI361

Computer Security

Public Key Cryptography II

RSA

Outline

- RSA.
 - Encryption and decryption.
- Using RSA.
- Choosing p and q.
- Implementation considerations.
- Some comments on factoring.
- Finding and testing primes.
- Fast exponentiation.
- Assessing the security of RSA.

RSA

- The RSA Public--Key Cryptosystem (**Rivest, Shamir and Adleman (1978)**) is the most popular and versatile PKC.
- It is the *de facto* standard for PKC.
- It supports *secrecy and authentication* and can be used to produce *digital signatures*.
- RSA uses the knowledge that it is *easy* to find primes and multiply them together to construct composite numbers, but it is *difficult* to factor a composite number.

The Euler phi Function

For $n \geq 1$, $\phi(n)$ denotes the number of integers in the interval $[1, n]$ which are relatively prime to n . The function ϕ is called the **Euler phi function** (or the **Euler totient function**).

Fact 1. The Euler phi function is **multiplicative**, i.e. if $\gcd(m, n) = 1$, then $\phi(mn) = \phi(m) \times \phi(n)$.

Fact 2. For a prime p and an integer $e \geq 1$, $\phi(p^e) = p^{e-1}(p-1)$.

- From these two facts, we can find ϕ for any composite n if the prime factorization of n is known.

The Euler phi Function

$$\phi(n) = \left| \{x : 1 \leq x \leq n \quad \text{and} \quad \gcd(x, n) = 1\} \right|$$

- $\phi(2) = |\{1\}| = 1$
- $\phi(3) = |\{1,2\}| = 2$
- $\phi(4) = |\{1,3\}| = 2$
- $\phi(5) = |\{1,2,3,4\}| = 4$
- $\phi(6) = |\{1,5\}| = 2$

- $\phi(37) = 36$
- $\phi(21) = (3-1) \times (7-1) = 2 \times 6 = 12$

The algorithm

1. Choose two primes p and q . Compute $n = pq$ and $m = \phi(n) = (p-1)(q-1)$.
 - $\phi(n)$ is Euler's totient function: It is the number of positive integers less than n that are relatively prime to n .
2. Choose e , $1 \leq e \leq m - 1$, such that $\gcd(e, m) = 1$.
3. Finds d such that $ed \equiv 1 \pmod{m}$.
 - This is possible because of the choice of e .
 - d is the multiplicative inverse of e modulo m and can be found using the extended Euclidean (gcd) algorithm.
4. The **Public key is (e, n) .**
The **Private key is (d, p, q) .**

Encryption and decryption

- Let X denote a plaintext block, and Y denote the corresponding ciphertext block.
- Let (z_A, Z_A) denote the private and public components of Alice's key.
- If Bob want to encrypt a message X for Alice. He uses Alice's public key and forms the cryptogram:

$$Y = E_{z_A}(X) = X^e \bmod n$$

- When Alice wants to decrypt Y , she uses the private component of her key $z_A=(d,n)$ and calculates

$$X = D_{z_A}(Y) = Y^d \bmod n$$

- X and Y are both integers in $\{0, 1, 2, \dots, n-1\}$.

- **Example:** Choose $p=11$ and $q=13$.

$$n=11 \cdot 13 = 143$$

$$m=(p-1)(q-1)=10 \cdot 12 = 120$$

$$e=37 \rightarrow \gcd(37, 120)=1$$

Using the gcd algorithm we find d such that
 $ed=1 \bmod 120$; $d=13 \rightarrow de=481$.

$$X, Y \in \{0, 1, \dots, 142\}$$

- **To encrypt:** Break the input binary string into blocks of u bits, where $2^u \leq 142$, so we choose $u=7$.
In general there is some agreed **padding scheme** from the message space into the space on which a “single run” of the cipher can act.
Optimal Asymmetric Encryption Padding (OAEP) is often used.
- For each 7-bit block X , a number between 0 and 127 inclusive, we calculate the ciphertext as $Y=X^e$.
- For $X=2=(0000010)$ we have
 $E_Z(X)=X^{37}=12 \pmod{143} \rightarrow Y=0001100$
- **To decrypt:** $X=D_Z(Y)=12^{13}=2 \pmod{143}$

An RSA public directory

User	(n,e)
Alice	(85,23)
Bob	(117,5)
Fred	(4757,11)

- An important property of the RSA algorithm is that encryption and decryption are the same function: both exponentiation modulo n.

$$E_{zA}(D_{zA}(X)) = X$$

- Example:
 - First decrypt: $2^{13} \equiv 41 \pmod{143}$
 - Then encrypt: $41^{37} \equiv 2 \pmod{143}$
- This is the basis of using RSA for authentication.

Using RSA

- The RSA system can be used for:
 1. **Confidentiality:** To hide the content of a message X , A sends $E_{z_B}(X)$ to B.
 2. **Authentication:** To ensure integrity of a message X ,
 - Alice *signs* the message by using her decryption key to form $D_{z_A}(X)$ and sends $(X, D_{z_A}(X)) = (X, S)$ to Bob.
 - When Bob wants to *verify the authenticity* of the message:
 - He computes $X' = E_{z_A}(S)$.
 - If $X'=X$ the message is accepted as authentic and from Alice.
 - Both **message integrity** and **sender authenticity** are verified.
 - This is true because even one bit change to the message can be detected, and because z_A is known only to Alice.
 - This method is inefficient. We will see later that Alice may compute a hash value of X and then apply D_{z_A} to the result.
 - We will talk about Digital Signatures later anyway.

3. Secrecy and authentication:

- If we need secrecy and authentication the following can be used:
- A sends $Y = E_{z_B}(D_{z_A}(X))$ to B.
- B recovers

$$X = E_{z_A}(D_{z_B}(Y))$$

This scheme provides *non-repudiation* if Bob holds on to $D_{z_A}(X)$. That is, Bob is protected against Alice trying to deny sending the message.

Choosing p and q

- The main conditions on p and q are:
 - They must be at least 100 decimal digits long (about 330 bits).
 - They must be of similar say, say both 100 digits.
- To choose each number the user does the following:
 - Randomly choose a random number b which is 100 digits long, or whatever length is appropriate.
 - Checks to see if b is prime. Usually using a probabilistic primality testing algorithm.
 - If b is not prime, choose another value for b.

RSA implementation considerations

- Relative to DES or AES;
 - RSA is a much slower algorithm.
 - RSA has a much larger secret key.
- Storage of p and q requires about 330 bits each, n is about 660 bits.
- Exponentiation is relatively slow for large n , especially in software.
- It can be shown that multiplication needs $m + 7$ clock pulses if n is m -bits in length.

- Approximate relative speeds (this is a few years old!)
 - Hardware: n is 507 bits: 220 kbits/sec
 - Software: n is 512 bits: 11 kbits/sec

Some comments on factoring

- The most powerful known (pretty much general purpose) factoring algorithm is the *general number field sieve*.
- It uses

$$O\left(\exp\left(\frac{64}{9} \log n\right)^{\frac{1}{3}} (\log \log n)^{\frac{2}{3}}\right)$$

steps to factor a number n .

- One of the best factoring results is for a 663-bit number (RSA-200). This was factored using a general number field sieve (announced May 9 2005). This took several months of work undertaken by a system of 80 AMD Opteron processors (~2.6GHz).
- Today for sensitive applications, a 1024 bit modulus, and in some cases a 2048 bit modulus, are considered necessary.

Finding primes

- An algorithm to generate all primes does not exist.
- However, given a number n , there exist efficient algorithms to *check whether it is prime or not*. Such algorithms are called **primality testing algorithms**.
- As mentioned earlier, prime generation for RSA is basically just a matter of guessing and testing.
- Primality Testing: Deterministic algorithms for proving primality are non-trivial and are only advisable on high performance computers. Probabilistic tests allow an *educated guess* as to whether a candidate number is prime or not.
 - This means that the probability of the guess being wrong can be made arbitrarily small.

Lehman's test

- Let n be an odd number. For any number a define

$$e(a, n) = a^{\frac{n-1}{2}} \bmod n$$

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

where $\mathbb{Z}_n^* = \{1, 2, \dots, n-1\}$.

- Example:** $n=7$

$$2^3=1, 3^3=6, 4^3=1, 5^3=6, 6^3=6 \rightarrow G=\{1, 6\}$$

- **Lehman's theorem:**

If n is odd, $G=\{1,n-1\}$ if and only if n is prime.

Example: $n=15$ isn't prime: $(n-1)/2=7$

$$2^7 \equiv 8 \pmod{15}, 3^7 \equiv 12 \pmod{15}$$

Example: $n=13$ (prime): $(n-1)/2=6$

$$2^6 \equiv -1 \pmod{13}, 3^6 \equiv 1 \pmod{13}, 4^6 \equiv 1 \pmod{13},$$

$$5^6 \equiv -1 \pmod{13}, 6^6 \equiv -1 \pmod{13}, 7^6 \equiv -1 \pmod{13},$$

$$8^6 \equiv -1 \pmod{13}, 9^6 \equiv 1 \pmod{13}, 10^6 \equiv 1 \pmod{13},$$

$$11^6 \equiv 1 \pmod{13}, 12^6 \equiv 1 \pmod{13}.$$

■ Thus, we have the following test:

if ($\text{gcd}(a,n) > 1$) *return*('composite')

else

if ($a^{(n-1)/2} = 1$) or ($a^{(n-1)/2} = -1$)

return('prime_witness')

else

return('composite')

If for a given n the test returns prime_witness for 100 randomly chosen a , then the probability of n not being not prime (i.e. being a composite disguised as a prime) is less than 2^{-100} .

Fast exponentiation

- Exponentiation can be performed by repeated multiplication. In general, we can use the *square and multiply* technique.
- To calculate X^α :
 1. Write α in base two:
$$\alpha = \alpha_0 2^0 + \alpha_1 2^1 + \alpha_2 2^2 + \dots + \alpha_{n-1} 2^{n-1};$$
 2. Calculate X^{2^i} , $1 \leq i \leq n-1$.
 3. Use $X^\alpha = (X^{2^0})^{\alpha_0} * (X^{2^1})^{\alpha_1} * \dots * (X^{2^{n-1}})^{\alpha_{n-1}}$ and
Multiply the X^{2^i} for which α_i is not zero.

- **A partial example:** $n=179$, $e=73$.

$$X=2 \rightarrow Y=2^{73} \bmod 179.$$

$$73=64+8+1=2^6+2^3+2^0$$

$$Y=2^{64+8+1}=2^{64}*2^8*2^1$$

This is only a partial example because we haven't looked at calculated the elements of the last line.

Precomputation:

$$X^2=X*X$$

$$X^4=X^{2^2}=X^2*X^2$$

...

$$X^{2^{n-1}}=X^{2^{n-2}}*X^{2^{n-2}}$$

This is a total of $n-1$ multiplications, all mod N

- **Example:** $N=1823$, $n=\log_2 1822=11$.

– Calculate $Y=5^{375} \bmod N$

- Precomputation:

X^1	5	X^2	25	X^4	625
X^8	503	X^{16}	1435	X^{32}	1058
X^{64}	42	X^{128}	1764	X^{256}	1658
X^{512}	1703	X^{1024}	1639		

- Never store a number larger than N !
- Never multiply two numbers large than N !

$$375 = 256 + 64 + 32 + 16 + 4 + 2 + 1.$$

$$5^{375} = 5 * 25 * 625 * 1435 * 1058 * 42 * 1658$$

$$= 591 \bmod 1823$$

There are various other tricks for calculating powers too, but we aren't going to look at them here!

A weakness in RSA

- In RSA not all the messages are concealed, i.e. the plaintext and ciphertext are the same.
- **Example:** $n=35=5*7$, $m=4*6$.
- $X=8$.
- $Y=8^5 \text{ mod } 35=8$

- For any key, at least 9 messages will not be concealed. But for $n \geq 200$ or so, this is not important.

- However if e is poorly chosen, less than 50% of the messages are concealed.
 - **Example:** $n=35$, $e=17$.
 $\{1,6,7,8,13,14,15,20,21,22,27,28,29,34\}$ are not concealed. ☹
- It is less likely that a system will have this problem if the primes are **safe**.
- A prime is **safe** if $p=2q+1$, where q is a prime itself.

Assessing the security of RSA

- The security of the private component of a key depends on the difficulty of factoring large integers.
- **Justification:**
 - Let $Z=(e,n)$. If n can be factorised then an attacker can find
 - $n=pq$
 - $m=(p-1)(q-1)$
 - ... and use the gcd algorithm to find the private key, $d=e^{-1} \bmod m$.
- No efficient algorithm for factoring *is known*.
- So knowing $n = pq$ does not yield p or q .
- This implies that $m=(p-1)(q-1)$ cannot be found and d cannot be computed.

- **Finding secret exponent:** If an attacker knows X and $Y = D_z(X)$ to find d they must solve

$$X = Y^d \bmod n \quad \text{or} \quad d = \log_Y X$$

- **Finding plaintext:** If the attacker knows Y and e to determine X they must take roots modulo a composite number: i.e. they need to solve $Y = X^e$.
- It is important to note that the security of RSA **is not provably** equivalent to the difficulty of factoring. That is, it might be possible to break RSA without factoring n .

Important attacks

- It is sufficient for the cryptanalyst to compute $\phi(n)$.
- Knowledge of n and $\phi(n)$ gives two equations:

$$n = pq$$

$$\phi(n) = (p-1)(q-1)$$

This system can be solved, for p say, by solving a quadratic equation:

$$p^2 - (n - \phi(n) + 1)p + n = 0$$

- **Some equivalent results:**
 - Any algorithm that can compute the decryption exponent can be used as a subroutine in a probabilistic algorithm that factors n .
 - Any algorithm that computes $\text{parity}(Y)$ can be used as a subroutine in an algorithm that computes the plaintext X .

Weak implementations

- Some implementations allow attacks:
 1. **Common modulus attack:** Consider a group of users who's public keys consists of the same modulus and different exponents.
 - If an intruder intercept two cryptograms where
 - They are encryptions of the same message with different keys.
 - The two encryption exponents do not have any common factor. Then the attacker can find the plaintext.

- Let us consider why:
- The enemy knows e_1 , e_2 , N , Y_1 and Y_2 , and furthermore that $Y_1 = X^{e_1} \text{mod } N$ and $Y_2 = X^{e_2} \text{mod } N$
- Since e_1 and e_2 are relatively prime, the Extended Euclidean algorithm can be used to find **a** and **b** such that $ae_1 + be_2 = 1$.

But then $Y_1^a Y_2^b = X^{ae_1} * X^{be_2} = X^{ae_1 + be_2} = X$

- **Using a common modulus among a number of participants is not advisable!**

- 2. Low exponent attack:** If the encryption exponent is small, encryption and signature verification will be faster.
 - However, if the enemy can find $e(e+1)/2$ linearly independent messages encrypted (signed) with the same exponent, then they can also find the message. The attack does not work if the messages are unrelated.
 - Pad messages with random strings to ensure that they are not dependent.
- 3. Low decryption exponent:** If the decryption exponent is less than $N/4$ and $e < N$ then d can be found.

RSA in real-life

1. If there is a small range of possible messages,
 - For example, messages are numbers between 0 and 1,000,000.
 - Then given a ciphertext, the plaintext can be found by exhaustively searching all plaintext.
2. The *Multiplicative property* of RSA can be used to find the plaintext.

$$C_1 = m_1^e \bmod N$$

$$C_2 = m_2^e \bmod N$$

$$C_1 C_2 = m_1^e m_2^e = (m_1 m_2)^e \bmod N$$

An attack against RSA

- Suppose m is l bits, $m \leq 2^l$.
- Suppose $m = m_1 m_2 \quad m_1, m_2 \leq 2^{l/2}$.
- The attacker knows $c = m_1^e m_2^e \pmod{N}$
- ... and builds a sorted database:
 $\{1^e, 2^e, 3^e, \dots, (2^{l/2})^e\} \pmod{N}$
- For a given c , the attacker searches the database for $c/i^e = j^e \pmod{N}$.
- This will take at most $2^{l/2}$ steps.
- The cost if $2^{l/2} \log N$ space is affordable is...

$$O\left(2^{l/2+1} * \left(\frac{l}{2} + \log^3 N\right)\right)$$

RSA

- The RSA Public--Key Cryptosystem (**Rivest, Shamir and Adleman (1978)**) is the most popular and versatile PKC.
- It is the *de facto* standard for PKC.
- It supports *secrecy and authentication* and can be used to produce *digital signatures*.
- RSA uses the knowledge that it is *easy* to find primes and multiply them together to construct composite numbers, but it is *difficult* to factor a composite number.

Euler Totient Function $\phi(n)$

- This function is to produce the number of integers between 1 and $n-1$ where

$$\gcd(\text{the number}, n) = 1$$

Fermat's Little Theorem

- For any integer a , the following equation holds:

$$a^{\phi(p)} = 1 \bmod p$$

For $0 \leq a < p$

Implication of Fermat's Little Theorem

Since $a^{\phi(p)} = 1 \pmod{p}$

Then: any computation in the exponent is done modulo $\phi(p)$

CSCI361

Computer Security

Public Key Cryptography IV
Rabin and Elgamal

Outline

- The Rabin cryptosystem.
 - Description.
 - An example.
 - Advantages and disadvantages.
- The Elgamal cryptosystem.
 - Description.
 - \mathbb{Z}_p : primitives/generators.
 - An example.

The Rabin cryptosystem

- The security of this system is equivalent to the difficulty of factoring.
- **Key generation:**
 - Bob randomly chooses two large primes,
 - p and q , and
 - calculates $N=pq$.
 - The **public key** is N and
 - The **secret key** is (p,q) .

- **To Encrypt:** the ciphertext $\textcolor{red}{Y}$ for a message (plaintext) $\textcolor{red}{X}$:
$$Y = X^2 \bmod N$$

Note this is RSA with $e=2$!

- **To decrypt** the received message:
 - Bob must find the square root of $Y \bmod N$.
 - Knowing (p,q) it is easy to find the square root, otherwise it is provably as difficult as factoring.
- **Special case:** p and q are 3 mod 4.
 - We construct four intermediate factors.

$$x_1 = Y^{(p+1)/4} \bmod p$$

$$x_2 = p - x_1$$

$$x_3 = Y^{(q+1)/4} \bmod q$$

$$x_4 = q - x_3$$

- We define

$$a = q(q^{-1} \bmod p)$$

$$b = p(p^{-1} \bmod q)$$

- This means, for example, that you calculate $q^{-1} \bmod p$ and multiply the result by q .
- Then 4 **possible plaintexts** can be calculated.

$$X_1 = (ax_1 + bx_3) \bmod N$$

$$X_2 = (ax_1 + bx_4) \bmod N$$

$$X_3 = (ax_2 + bx_3) \bmod N$$

$$X_4 = (ax_2 + bx_4) \bmod N$$

An example

- We choose $p=7$, $q=11$ so $N=77$.
 - Note that p and q are 3 mod 4.
- Bob's public key is 77, and his private key is $(7,11)$.
- To encrypt $X=3$ Alice calculates
 $Y=3^2 \text{ mod } N = 9 \text{ mod } 77$.
- To decrypt Bob calculates

$$x_1=9^2 \text{ mod } 7 = 4$$

$$x_2=7-4=3$$

$$x_3=9^3 \text{ mod } 11 = 3$$

$$x_4=11-3=8$$

You can calculate $9^3 \text{ mod } 11$ as $(-2)^3 \text{ mod } 11=-8 \text{ mod } 11=3$.

- Bob then finds **a** and **b**:

$$7(7^{-1} \bmod 11) = 7 \times 8 = 56$$

$$11(11^{-1} \bmod 7) = 11 \times 2 = 22$$

- ... and then the four possible plaintexts.

$$X_1 = 4 \times 22 + 3 \times 56 = 11 + 14 = \mathbf{25} \bmod 77$$

$$X_2 = 4 \times 22 + 8 \times 56 = 11 + (-14) = -3 = \mathbf{74} \bmod 77$$

$$X_3 = 3 \times 22 + 3 \times 56 = -11 + 14 = \mathbf{3} \bmod 77$$

$$X_4 = 3 \times 22 + 8 \times 56 = -11 - 14 = -25 = \mathbf{52} \bmod 77$$

Advantages and disadvantages.

- Provable security.
- Unless the RSA exponent e is small, Rabin's encryption is considerably faster than RSA, requiring **one modular exponentiation**.
 - Decryption requires roughly the same time as RSA.
- Decryption of a message generates 4 possible plaintexts. The receiver needs to be able to decide which one is the right message. One can append messages with known patterns, for example 20 zeros, to allow easy recognition of the correct plaintext.
- Rabin's system is mainly used for authentication (signatures).

Generator of Z_p^*

- An element α is a generator of Z_p^* if $\alpha^i, 0 < i \leq p-1$ generates all numbers $1,.., p-1$
- Finding a generator in general is a hard problem with no efficient algorithm known.
- If factorization of $p-1$ is known it is not hard.
- In particular if $p=2p_1+1$ where p_1 is also a prime we have the following.

- α in Z_p^* and $\alpha \neq \pm 1 \pmod{p}$.
- Then α is a primitive element if and only if

$$\alpha^{\frac{p-1}{2}} \neq 1 \pmod{p}$$

- Suppose α in Z_p^* and α is not a primitive element.
- Then $-\alpha$ is a primitive element.
- This gives an efficient algorithm to find a primitive elements.

- Example: \mathbb{Z}_{11}^*
 - $3^5 \equiv 1 \pmod{11}$
 - 3 is not primitive, -3 = 8 is primitive

An example : Z_{11}^*

	1	2	3	4	5	6	7	8	9	10
1	1	1								
2	2	4	8	5	10	9	7	3	6	1
3	3	9	5	4	1	3				
4	4	5	9	3	1	4				
5	5	3	4	9	1	5				
6	6	3	7	9	10	5	8	4	2	1
7	7	5	2	3	10	4	6	9	8	1
8	8	9	6	4	10	3	2	5	7	1
9	9	4	3	5	1	9				
10	10	1	10							



Discrete Logarithm Problem (DLP)

INPUT:

- Z_p^*
- g in Z_p^* , g a generator of Z_p^*
- h in Z_p^*

Find the unique number $a < p$ such that $h = g^a$

DL Assumption: There is no efficient algorithm (polynomial time) to solve DL problem.

- It is widely believed that this assumption holds.

DLP

- When p is small discrete log can be found by exhaustive search
- The size of prime for which DL can be computed is approximately the same as the size of integers that can be factored
 - In 2001: 120 digit DL, 156 digit factorization
- DL is an example of one-way function:
- A function f is **one-way** if
 - Given x , finding $f(x)$ is easy
 - Given y , finding x such that $f(x)=y$ is hard

The El Gamal Cryptosystem

■ Key generation:

- Alice chooses a prime p and two random numbers g and u , both less than p , where g is a generator of \mathbb{Z}_p^* .
- Then she finds:

$$y = g^u \bmod p$$

Alice's public key is (p, g, y) , her secret key is u .

- To encrypt a message X for Alice, Bob chooses a random number k such that $\gcd(k,p-1)=1$. Then he calculates:

$$a = g^k \bmod p$$

$$b = y^k \times X \bmod p$$

- The cryptogram is (a,b)
- The length is twice the length of the plaintext.
- To decrypt (a,b) Alice calculates

$$X = \frac{b}{a^u} \bmod p$$

ElGamal Cryptosystem

Public Key : $y = g^u$, Secret Key : u

Encryption :

$$a = g^k \pmod{p}$$

$$b = y^k \times X \pmod{p}, \quad \text{ciphertext} : (a, b)$$

Decryption :

$$X = \frac{b}{a^u}, \text{division uses Euclid extended Algorithm}$$

To avoid division :

$$X = b \times a^{-u} = b \times a^{p-1-u}$$

Example

- We choose $p=11$, $g=3$, $u=6$.
- We calculate $y=3^6 \equiv 3 \pmod{11}$
$$\begin{aligned}3^6 \pmod{11} &= (3^2)^3 \pmod{11} \\&= (-2)^3 \pmod{11} \\&= -8 \pmod{11}\end{aligned}$$
- The public key is **(11,3)**.
- To encrypt **X=6**, Bob chooses $k=7$ and calculates:
 $a=3^7 \equiv 9 \pmod{11}$, $b=3^7 \times 6 \equiv 10 \pmod{11}$
The cryptogram is **(9,10)**.
- To decrypt Alice finds:
 $X=10/9^6 \equiv 10/9 \equiv 10 \times 5 \equiv 6 \pmod{11}$

Security problems with ElGamal

- Similar to RSA:

Given $(a, b) = (g^k, y^k \times m) \pmod{p}$

Eve: – chooses a random number r in Z_p^ ,*

– computes rb

– sends (a, rb) to Alice

Alice returns the decryption: $r \times m \pmod{p}$

Eve will find: $\frac{r \times m}{r} = m \pmod{p}$

Provable security

- Adversary power
 - Eve is polynomially bounded
 - time and memory is bounded by a polynomial in the size of input
 - Cannot exhaustively search
 - Eve has access to ‘oracles’:
 - Can ask queries and receive responses
 - Attacks are modeled as a ‘game’ between an attacker Eve and an innocent user (Oracle)

CSCI361

Computer Security

Digital Signatures

Outline

- Digital signatures.
 - Required properties.
- Main differences between handwritten and digital signatures.
- Signature schemes.
 - Component algorithms and requirements on them.
- Digital signatures using PKC.
- A problem and a solution.

Digital Signatures

- Introduced by Diffie-Hellman (1976).
- A digital signature is the electronic analog of handwritten signature. It ensures integrity of the message and authenticity of the sender. That is, if Bob gets a message which supposedly comes from Alice, he is able to check that...
 - The message has not been modified.
 - The sender is truly Alice.
- Digital signatures require a few properties. They should be:
 - Easy to generate.
 - Easy to verify by a receiver.
 - Hard to forge.
- A digital signature is generally represented as a **string of bits** attached to a message.
- This is similar to the mechanism for **Message Authentication Codes** but they didn't allow for the authenticity check on the sender.

Differences between handwritten and digital signatures

- There are a few major differences between handwritten and digital signatures.
 - The digital signature needs to depend on the (*whole*) message in order for the integrity to be verifiable.
 - If you just paste your name at the end of your email message this doesn't prove you sent it!
 - To produce a true signature the signer goes through a physical process. With a digital signature the signer provides input to a process performed by a computer.
 - This input uniquely identifies the signer, it could be a password, or biometric data (fingerprint, retina print, etc) or more likely both.

- There are two major categories of digital signature:
 1. **True digital signatures**: A signed message is generated by the signer and sent directly to the receiver, who can directly verify the signature.
 - A third party is only required in the case of disputes.
 2. **Arbitrated signatures**: In these the signed message is sent via a third party.
- We are only really interested in the first type here.

Signature schemes

- A **signature scheme** consists of two algorithms (possibly three if you consider key generation too):
 1. A **signature generation algorithm** denoted S_A , takes a message X and produces a signature $S_A(X)$.
 2. A **signature verification algorithm** denoted V_A , takes a message X and the associated signature and produces a **True** or **False** value.
 - True indicates the message should be accepted while False indicates the message should be rejected.

Algorithm requirements

- The algorithms defining a signature scheme need to satisfy some requirements:
 - $V_A(X, S) = \text{True}$ if and only if $S=S_A(X)$.
 - Without knowledge of the key, it is computationally infeasible to construct a valid pair (X, S) such that $V_A(X, S) = \text{True}$.
 - To ensure **protection against forgery**, we need the property that if $(X, V_A(X, S))$ is known, it is computationally infeasible to find X' such that $V_A(X', S)=V_A(X, S)$.

- In general it is possible to have two messages with the same digital signature. That is, there exists X and X' such that,
 $S_A(X)=S_A(X')$.
- We only require that it is computationally infeasible to find such an X' if $(X, V_A(X, S))$ is known.

Digital Signatures using PKC

- Alice signs X by creating $Y = E_{z_A}(X)$. The signed message is (X, Y) .
- Bob validates Alice's signature by computing $X' = E_{z_A}(Y)$ and comparing it with X .
- Everyone has access to E_{z_A} and hence anyone can verify the signature by computing $E_{z_A}(Y)$.
- Since E_{z_A} is a trapdoor one-way function, it is not possible for an intruder to find z_A .
- Hence Alice's signature cannot be forged.

Other integrity services

- Digital signatures can provide more than just explicit authentication and integrity.
- For example, they can provide **non-repudiation**. This prevents the sender from denying he/she has sent the message.
- If Alice signs a message, since she is the only one who knows z_A , she will be held accountable for it.
→ The system therefore provides **non-repudiation**.

- Alice may be able to repudiate by claiming E_{z_A} was compromised.
 - To provide protection against this we may use an *arbitrator or public notary*.
 - A public Notary **N** signs messages on top of Alice's signature. Hence Alice cannot claim the message is forged. **N** sends a copy of the signed message back to Alice.

- **Proof of delivery** is protection against the receiver denying receipt of the message. This is normally implemented by using an **judicable protocol**. These are protocols that can be later verified by a third party.

$$A \rightarrow B : Y = E_{z_B}(E_{z_A}(X))$$

B computes X' , the decryption of the received message.

$$B \rightarrow A : Y' = E_{z_A}(E_{z_B}(X'))$$

- Bob acknowledges receipt of X .
- Alice assumes the message was not received, unless he receives an acknowledgment from Bob.

Some problems ...

- As described we would implement a PKC based digital signature by breaking a message into blocks and signing each block independently.

$$X_1, X_2, \dots, X_t$$

$$S_A(X_1), S_A(X_2), \dots, S_A(X_t)$$

- This means, however, the signed message is susceptible to *block re-ordering*, *block deletion*, and *block replication*.
- It is possible to protect against these by adding redundancy to the blocks. This makes the system less efficient however.
- Problems:
 - PKC is slow. ☹
 - The signature is the same size as the message. ☹
 - It is desirable to produce a **short** signature that is a function of the whole message.

... and a solution

- The solution is to use hash function in conjunction with PKC.
- We will look at hash functions after we have looked at some examples of digital signatures.

CSCI361

Computer Security

Digital Signatures Schemes

Outline

- The Elgamal Signature scheme.
 - Signing and verifying.
 - Example.
- The Digital Signature Standard (DSS).
 - Development history.
 - Digital Signature Algorithm.
- Other types of signature schemes.

The Elgamal signature scheme

- This is based on the difficulty of computing discrete logarithms over \mathbb{Z}_p , where p is a prime.
- **Setup and Key generation:**
 - Alice chooses a large prime p such that $p-1$ has a large prime factor.
 - Let g denote a primitive element of \mathbb{Z}_p . Alice chooses x as her private key and calculates $y=g^x \bmod p$.
- The public key is (p, g, y) .
- The private key is x .
- This is the same as for encryption.

Signing and verifying

- To sign a message X :
 - Alice chooses and integer k , $1 \leq k \leq p-1$, such that $\gcd(k,p-1)=1$.
 - Alice calculates
 - $r=g^k \bmod p$.
 - $s=k^{-1}(X-xr) \bmod (p-1)$
 - The signed message is $(X.(r,s))$.
- To verify the message; compare g^X and $y^r r^s$.

$$g^X = g^{xr+ks} = g^{xr} \cdot g^{ks} = y^r \cdot r^s$$

Using the
received
values



- **Example:** $p=11$, $g=2$. (Remember in previous notes we showed that 2 is a generator of \mathbb{Z}_{11} .)
- **Private key:** $x=3$.
- **Public key:** $y=2^3=8 \text{ mod } 11$.
- **Signing X=9.**
 - Choose $k=7$. → Check that $\gcd(7,10)=1$. ✓
 - Calculate $k^{-1}=3 \text{ mod } 10$. → Calculated with above!
 - Calculate $r=g^k=2^7=7 \text{ mod } 11$.
 - Calculate $s=3(9-7*3) \text{ mod } 10 = 4$.
- The signed message is $(9.(7,4))$.
- **Verification** is by comparison of g^x and $y^r r^s$.
 $2^9 = 6 = 8^7 * 7^4 \text{ mod } 11$

The Digital Signature Standard (DSS)

- Proposed by NIST in 1991, finally issued in 1994 (Federal Information Processing Standard FIPS 186).
- Various modifications were made, FIPS 186-2 came out in 2000.
- Work on FIPS 186-3 is currently underway (there is a draft).
- We are just going to talk about the original Digital Signature Algorithm in FIPS 186.

Digital Signature Algorithm (DSA)

- **Key generation and setup:**

- q a 160-bit prime is chosen
- p a prime. 512-1024 bits long with the length a multiple of 64. q is a factor of $p-1$.
- h a number less than $p-1$ such that
$$g = h^{p-1/q} \bmod p > 1$$
- $u < q$
- $y = g^u \bmod p$

- **Public parameters:** p, q, g . (Multiple people could use these).
- **Private key:** u .
- **Public key:** y .

Generating and verifying a signature

- To sign a message ...

 - Alice generates a random k , such that $k < q$.

 - Alice computes

$$r = (g^k \bmod p) \bmod q$$

$$s = (k^{-1}(H(X) + ur)) \bmod q$$

Using the
received
values



- To verify a message Bob calculates...

$$w = s^{-1} \bmod q$$

$$t_1 = (H(X)^*w) \bmod q$$

$$t_2 = rw \bmod q$$

$$v = ((g^{t_1} * y^{t_2}) \bmod p) \bmod q$$

H(X) is the hash
of X. We will get
to these soon.

... and accepts the signature if $v=r$.

Example

■ Setup:

- $p=29$, $q=7$ (is a factor of 28).
- $h=3$, $g=h^{(29-1)/7}=3^4=23 \text{ mod } 29$
- $u=2 < q$.
- $y=g^2 \text{ mod } 29 = 23^2 \text{ mod } 29$
 $= (-6)^2 \text{ mod } 29$
 $= 36 \text{ mod } 29 = 7$

■ Signing:

- To sign $H(X)=5$, Alice chooses $k=4$ ($< q$) and calculates $k^{-1}=2$.

$$\begin{aligned} r &= (g^4 \bmod 29) \bmod 7 \\ &= ((-6)^4) \bmod 29 \bmod 7 \\ &= (36*36 \bmod 29) \bmod 7 \\ &= (7*7 \bmod 29) \bmod 7 \\ &= (49 \bmod 29) \bmod 7 = 20 \bmod 7 = \mathbf{6} \end{aligned}$$

$$\begin{aligned} s &= k^{-1}(H(X) + ur) \bmod q \\ &= 2(5+2*6) \bmod 7 \\ &= 34 \bmod 7 \\ &= \mathbf{6} \end{aligned}$$

- **Verifying:**

$$w = s^{-1} \bmod q = 6^{-1} \bmod 7 = 6$$

$$\begin{aligned}t_1 &= (H(X)^*w) \bmod q \\&= 5 * 6 \bmod 7 = 2\end{aligned}$$

$$\begin{aligned}t_2 &= rw \bmod q \\&= 6 * 6 \bmod 7 = 1\end{aligned}$$

$$\begin{aligned}v &= ((g^{t_1} * y^{t_2}) \bmod p) \bmod q \\&= ((23^2 * 7^1) \bmod 29) \bmod 7 \\&= (((-6)^2 * 7) \bmod 29) \bmod 7 \\&= ((7 * 7) \bmod 29) \bmod 7 \\&= 6\end{aligned}$$

- Since $v=r$ the message and signature are accepted as authentic.

Notes

- The recommended hash function for DSA is **SHA-1**.
 - We will discuss this later.
- There were several criticisms against DSA.
 - DSA cannot be used for encryption or key distribution.
 - It was developed by NSA and so cannot be trusted.
 - RSA is the de facto standard.
 - The key size is too small.

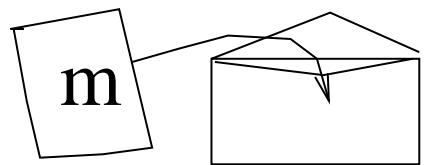
Signature scheme variants

- There are many different types of signature schemes.
- They are designed to meet the requirements of different scenarios.
- The cost of adding properties is efficiency, both in terms of computation and storage.
 - A lot of research is done into improving the efficiency of various different types of signature schemes.
- We are going to look briefly at a few types of signature schemes.

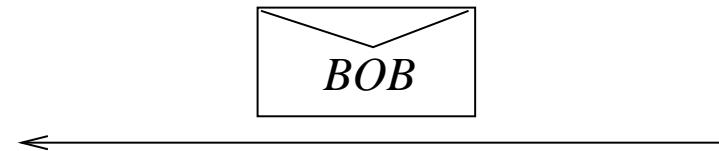
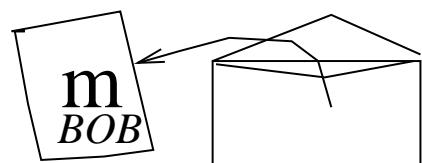
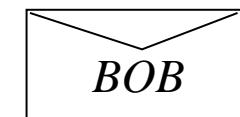
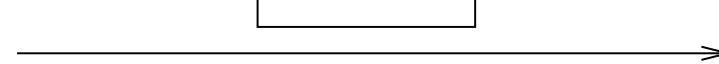
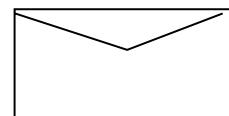
Blind signatures

- In some cases it is necessary to get the signature of a party without allowing them to see the message. For example, in electronic cash (proposed by David Chaum), a coin is the means of payment. A coin has a serial number and must be signed by the bank to be authentic.
 - To generate a coin, a customer randomly generates a number. The customer needs the bank signature on the coin before being able to spend it. To ensure privacy of the coin, that is to ensure that the bank cannot link a coin with a particular customer, the customer uses a **blind signature scheme**.
- In general:
 - The requester wants to obtain the signer's signature of message m .
 - The requester doesn't want to reveal m to anyone, including the signer.
 - The signer signs m blindly, not knowing what they are signing.
 - The requester can retrieve the signature.

Requester (Alice)



Signer (Bob)



Blind signatures using RSA

- **Setup:** Bob has (d, e) , a (private, public) key pair.
 - $N = pq$, where p, q are large primes, associated with Bob.
- For a message X , that Alice wants Bob to sign, Alice constructs $\mu = mr^e \bmod N$, where $r \in_R \mathbb{Z}_N^*$, and sends μ to Bob.
- Bob signs μ and sends his signature $\sigma = \mu^d \bmod N$ back to Alice.
- Alice retrieves the signature s of m by computing:

$$s = \frac{\sigma}{r} = \frac{\mu^d}{r} = \frac{m^d r^{ed}}{r} = \frac{m^d r}{r} = m^d \bmod N$$

Undeniable signatures

- These are non-transferable signatures.
- That is, the signature can be verified only with the collaboration of the signer.
 - This could be useful, for example, when a software distributor signs software. In this case it is important that the verifiable software cannot be duplicated. That is, the signature can be different for different people and the distributor will be able to see if there is duplication.

Group signatures

- Consider the following scenario:
 - A company has several computers connected to the local network.
 - There are a number of departments, each having a printer.
 - The printer for each department is only allowed to be used by members of that department.
 - So before printing, the printer must be convinced that the user is a member of that department.
 - To ensure privacy the user's name may not be revealed.
 - However if a printer is used too often, the director must be able to discover who misused the printer.
- Group signatures (with revocation) allow us to meet these requirements.

- A **group signature scheme** has the following properties:
 - Only members of the group can sign messages.
 - The receiver can verify the signature but *cannot discover which member of the group generated it.*
 - In the case of a dispute, the signature can be “opened” to reveal the identity of the signer.

Fail-stop signature schemes

- A Fail-Stop Signature Scheme provides enhanced security against the possibility that a very powerful adversary might be able to forge a signature.
- In the event that Oscar is able to forge Bob's signature on a message, Bob will (with high probability) subsequently be able to *prove* that Oscar's signature is a forgery.

Threshold signature schemes

- Consider the following scenario:
 - In a bank, there is a vault which must be opened everyday.
 - The bank employs three senior tellers, but they do not trust the combination to any individual teller.
 - We would like to design a system whereby any two of the three senior tellers can gain access to the vault, but no individual teller can do so.
- In the context of a signature basically the parties need to work together to construct a **proof of authority**.
- This problem can be solved by means of a *secret sharing scheme*, we will be talking about those later in this course.
- A (t, w) threshold scheme is a method of sharing a key K among a set of w participants in such a way that any t participants can compute the value of K , but no group of $t-1$ participants can do so.

CSCI361
Computer Security

Hashing

Outline

- Hash functions.
 - Cryptographic requirements.
- Signing with hash functions.
- Assessing security.
 - The birthday attack.
- Techniques for hash functions.
 - Block ciphers.
 - Modular arithmetic.
 - Dedicated or designed.

Hash functions

- A *hash function* H accepts a message X of arbitrary size and outputs a block of fixed size, called a *message digest* or *hash value*. The idea is that, in general, $H(X)$ is much smaller than X .
 - **Example:** The message is an arbitrary file and the digest is a 128 bit block.
- It is possible that two distinct messages produce the same message digest. This is called a *collision*.
 - **Example:**
$$H(X) = X \bmod 10$$
$$H(56) = H(96) = H(156)$$
- For cryptograph we require hash functions with *collision resistance*. That is, where it is computationally difficult to find two inputs x and y such that $H(x)=H(y)$.

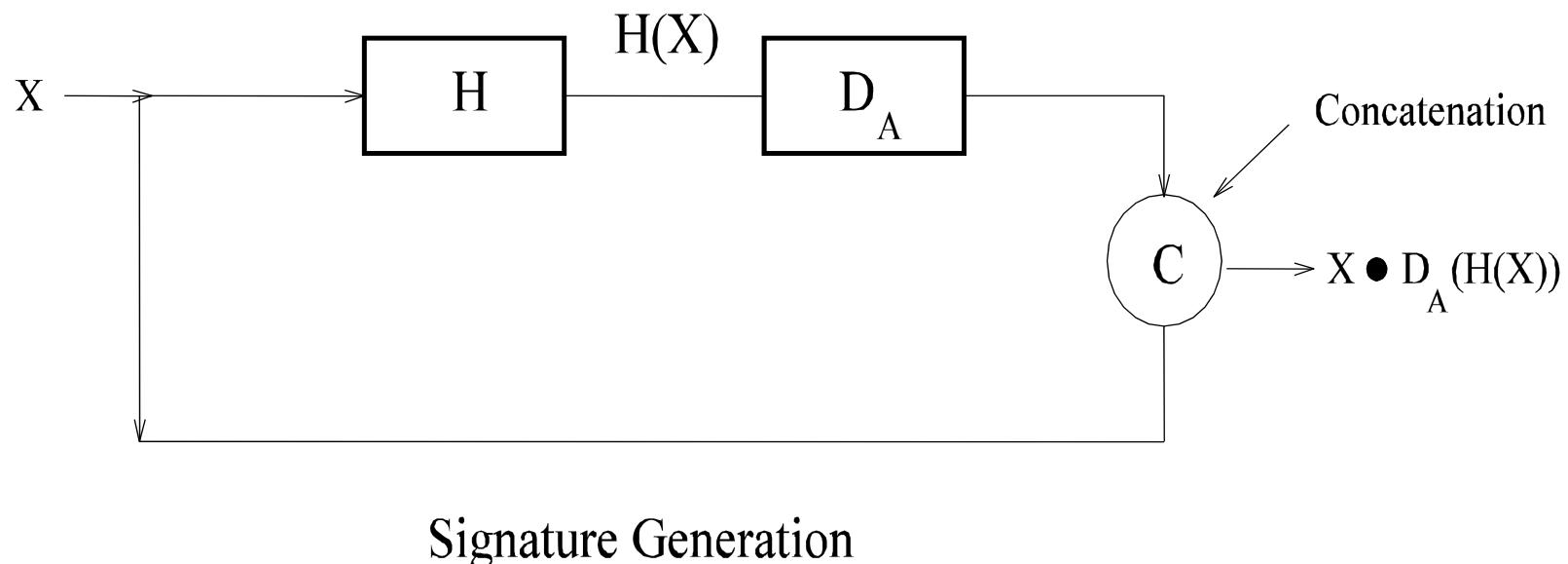
Cryptographic hash functions

- We require the following properties for cryptographic hash functions:
 1. It can be applied to any size input.
 2. The output must be of fixed size.
 3. **One-way**: Easy to calculate but hard to invert.
 4. **Pre-image resistant**: For any given Y , it is difficult to find an X such that $H(X)=Y$.
 5. **Second Pre-image resistant**: Given X_1 , it should be difficult to find another X_2 such that $H(X_1)=H(X_2)$.
 6. **Collision resistant**: It is computationally infeasible to find messages X and Y with $X \neq Y$ such that $H(X)=H(Y)$.

Properties 3 and 4 are closely related.

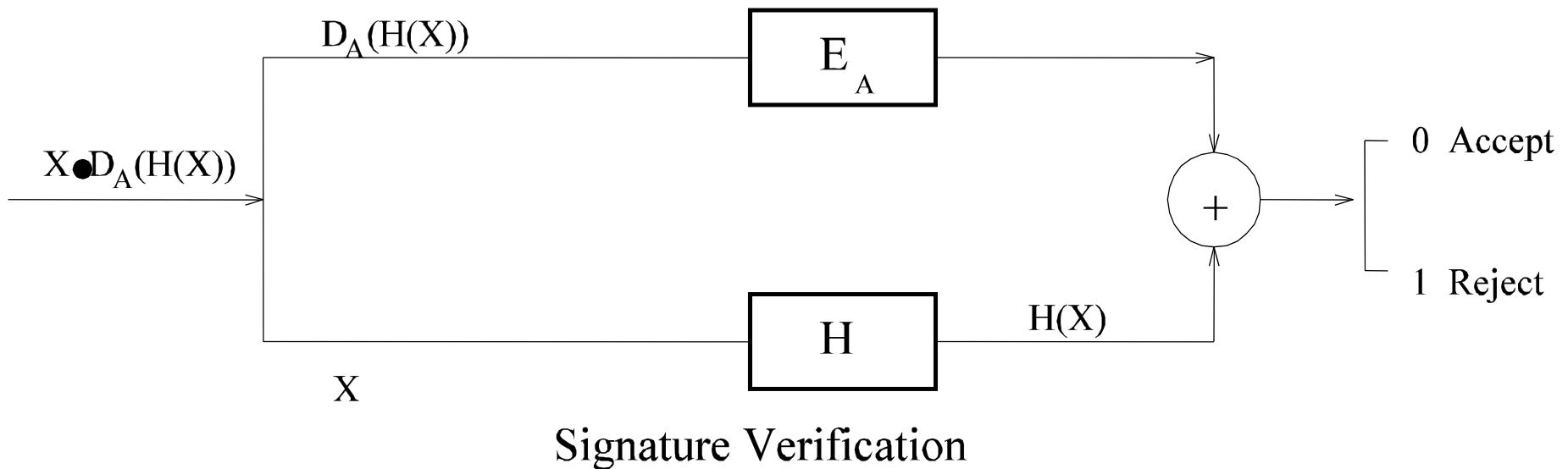
Signing with hash functions

- Hash functions can be used as a means to provide message integrity. One of the main applications of hash function is for digital signatures.
- To sign a message X , the hash value $H(X)$ is calculated and signed.



Verification

- Verification involves calculating the hash on the message again and comparing it with the transmitted hash.



- In the context of signatures;
 - Cryptographic hash properties 1, 2 and 3 are required for efficient signature generation.
 - Cryptographic hash properties 4,5 and 6 are required to stop attackers forging signatures.
- Consider that **Oscar**, a malicious person, can generate $X \neq X'$ with $H(X) = H(X')$ such that Alice is happy to sign X and produce $S_A(X)$. Now **Oscar** can use $(X', S_A(X))$ as a signed message.

- For property 4, 5 and 6 to hold, the size of the output space must be large, currently the standard is 2^{160} . That is, the hash value must be at least 160 bits.
- A strong signature scheme and a secure hash function may produce a weak signature scheme. An example of this was demonstrated This problem is shown by Coppersmith in combining RSA and the CCITT X.509 hash function.

Assessing the security of hash functions

- If the size of message space is bigger than digest space, we can always find collisions (the pigeonhole principle).
- **Example:**
 - Let messages belong to $Z_p^* = \{1, 2, \dots, p-1\}$, and digests belong to $Z_q^* = \{1, 2, \dots, q-1\}$ where q is a prime and $p > q$.
- Choose a number $g \in Z_q^*$.
- We define a hash function $h(x) = g^x \bmod q$.

- Let $q=15$, $p=11$, and choose $g=3$.
- We have,

$$3^2 = 9 \text{ mod } 11$$

$$3^3 = 5 \text{ mod } 11$$

$$3^4 = 4 \text{ mod } 11$$

$$3^5 = 1 \text{ mod } 11$$

$$3^6 = 3 \text{ mod } 11$$

$$3^7 = 9 \text{ mod } 11$$

→ Collision between $x=2$ and $x=7$.

- If 4 bits are used to represent messages for this system, then,

$$h(0010)=h(0111)$$

- Obviously this is not a good cryptographic hash function. How we do measure the likelihood of collisions, and thus the security?

Birthday Attack

- This terminology arises from the **Birthday paradox**.
- **Question:**
 - What is the smallest size class such that the chance of two students having the same birthday is *at least* $\frac{1}{2}$?

- The answer is surprisingly low: there need to be only 23 students in the class. This is only a paradox in the sense it seems counter-intuitively low.
- This can be applied to finding collisions in hash functions, or at least to calculating the probability of finding collisions.
- Suppose there are m possible hash values (message digests). Assume the associated with the same (or similar) number of messages. If we evaluate the hash value of k randomly selected messages, the probability of at least one collision is

$$P(m, k) > 1 - e^{-\frac{k(k-1)}{2m}} = \varepsilon$$

- Note that the chance of success in this attack depends only on
 - The size of the message digest, and
 - The number of messages digests are calculated for.
- In particular, it does not depend on the specific hash function used. That is the size of digest space puts a lower bound on the probability of success.

$$k = \sqrt{2m \ln\left(\frac{1}{1-\varepsilon}\right)}$$

- Calculate this for $\varepsilon=0.5$, $m=365\dots$ $k=22.49\dots$
- The formula suggests that with \sqrt{n} evaluations of the hash function there is a good chance (about 50%) of a collision being found.
- We can use this lower bound to find the required size of digest space, that is the number of bits of the digest, if we assume certain values for the level of feasible computation.

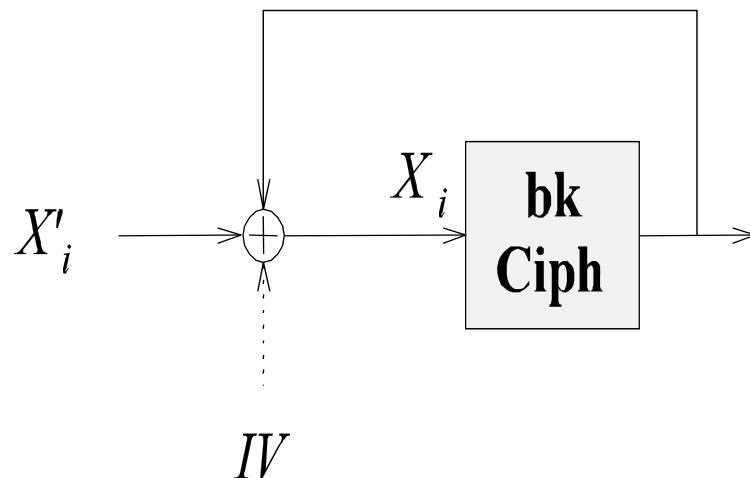
$$m = \frac{k^2}{2 \ln\left(\frac{1}{1-\varepsilon}\right)}$$

Techniques for hashing

- Hashing can be divided into three main categories:
 1. Techniques based on block ciphers (symmetric key).
 2. Techniques based on modular arithmetic.
 3. Dedicated or designed hash functions (Others).

Hash functions from block ciphers

- Using block ciphers to produce secure hash functions is not easy. Many schemes have been proposed and broken.
- The simplest way is to use a block cipher in cipher-block chaining mode.



**There is only output
at the end!**

- This is actually a message authentication code (CBC-MAC), where the key must be known to both sides.

$$X = X_1, X_2, \dots, X_n$$

$$Y_i = E_K(X_i \oplus Y_{i-1})$$

$$H(X) = Y_n$$

- This method is standardized for banking authentication (ANSI 9.9. ANSI 9.19 ISO 873-1).
- **Disadvantages:**
 - The recipient must have the key;
 - If the key is known it is easy to forge messages, that is, it doesn't actually provide authentication.

- Actually, for any n -bit block Y and any sequence of n -bit blocks X_1, X_2, \dots, X_w it is possible to choose a further n -bit block X_{w+1} such that

$$X_{w+1} = D_K(Y) \oplus Y_w$$

where Y_w is the hash value of X_1, X_2, \dots, X_w .

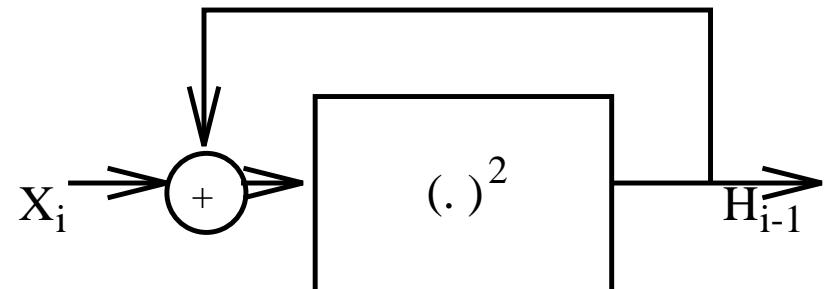
- This new message with $w+1$ blocks has exactly the same hash value since the xor's cancel out.

Hash functions based on modular arithmetic: An example.

- Quadratic Congruential Manipulation Detection Code (QCMDC) (Jueneman (1983))
- This scheme is a *keyed hash function*.
- Break the message into fixed size blocks of m -bits. H_0 is a randomly chosen *secret initial value* (the key).
 - M is a prime satisfying $M \geq 2^{m-1}$.
 - $+$ is integer addition.
- Then

$$H_i = (H_{i-1} + X_i)^2 \bmod M$$

- H_n is the hash value.
- This scheme is broken. (Coppersmith)}

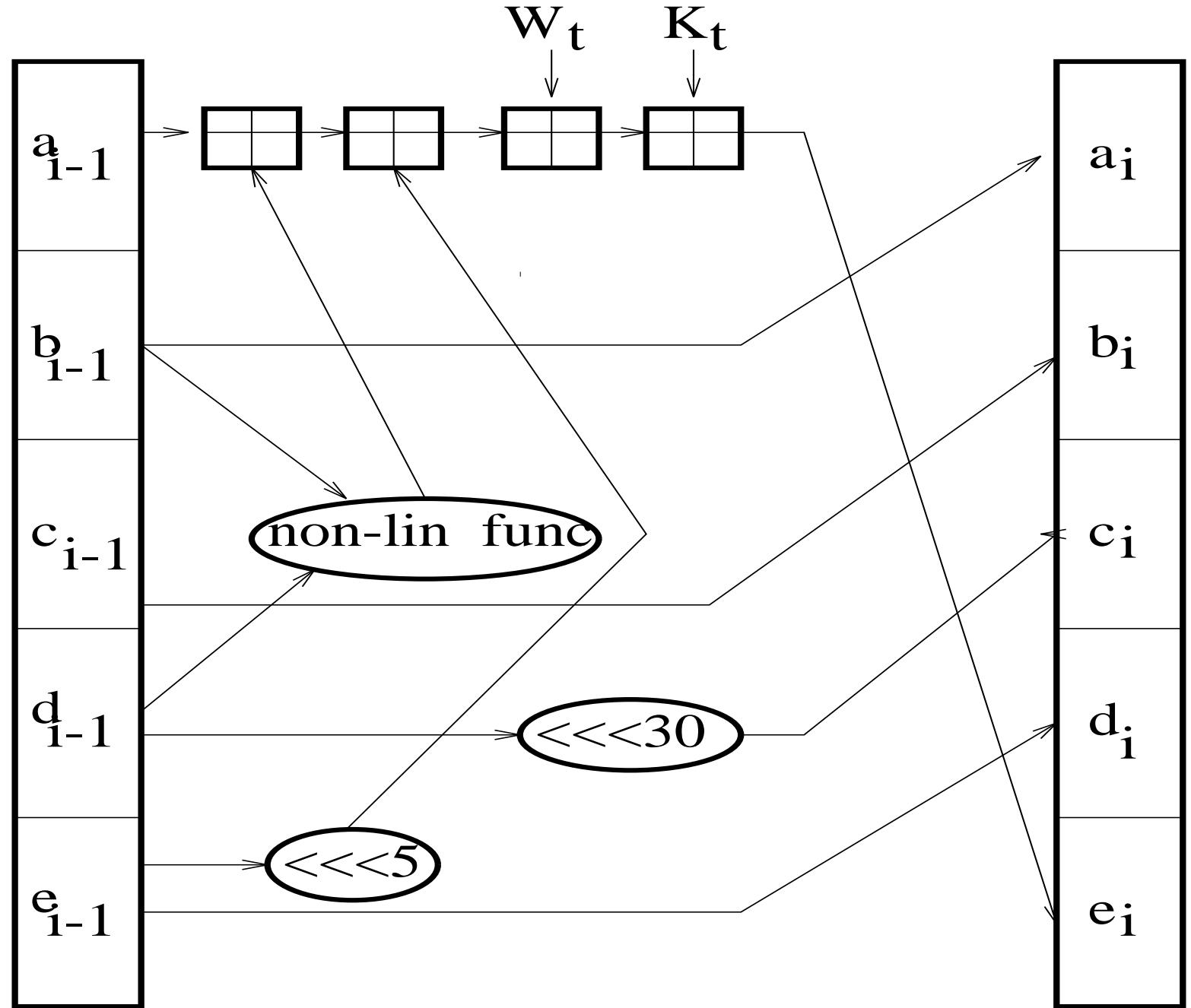


Designed hash functions

- MD5 (Rivest 1992).
- One of the best known hash algorithms.
 - It processes the input as blocks of 512-bit and produces a 128-bit message digest.
 - MD5 is the successor of MD4. It gave up a little in speed for a much greater likelihood of ultimate security.
 - The MD5 algorithm is designed to be fast on 32-bit machines.
 - It uses simple operations such as *addition modulo 32*, and can therefore be coded quite compactly and exexcuted very quickly.
- Collisions were found against MD5 in 2004, that is a method of finding collisions in a practical time.
 - As early as 1993 partial collisions were found.
- Significantly faster methods of finding collisions were demonstrated in 2005.

Secure Hash Algorithm (SHA-1)

- A hash algorithm proposed and adopted by NIST, for use with the DSA standard.
- Kind of a successor to MD5.
- Designed as an improvement to SHA-0.
- It produces a 160 bit message digest and uses the design approach used in MD5.
- On the next page we show a single SHA-1 operation.
- The plus boxes are addition mod 2^{32} .
- The non-linear function and K_t vary for different operations.



- In SHA-1 the message is padded to make 512-bit blocks (this is also done in MD5).
- The algorithm processes 512 bit blocks in each round. Each round has 4 sub-rounds and each sub-round consists of 20 operations. The nonlinear function used in each round is different.
- W_t , $i=0, 1 \dots 79$ are 32-bit blocks constructed from a 512 bit message blocks.
- K_t is a constant dependent upon the sub-round.
- SHA-1 is broken, but not yet to a practical level for attackers except for large scale distributed systems. The older SHA-0 is thoroughly broken and is insecure.
- No attacks on SHA-2 (variants) have been reported so far.

Other hash functions

- HAVAL.
- RIPEMD-(160).
- Snefru.
- Tiger.
- WHIRLPOOL.

MAC Based on a Keyed Hash Function

- Hash functions can be used to construct message authentication codes (MAC).
- In this application the sender and receiver share a key **K**.
- A common way of constructing a MAC from a hash function is:

$$\text{HMAC} = H(K|M|K)$$

- Here HMAC denotes **H**ash-based **M**AC. The key **K** is pre-fixed and post-fixed to the message. There are various variations on this.

Latest Progress

- SHA-3 Competition
- Round 1: NIST announced [14 Second Round Candidates](#) on July 24, 2009, and completed the first round of the SHA-3 Competition.
- <http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/index.html>

Final Round

- Finalist:
 - *BLAKE*
 - *Grøstl*
 - *JH*
 - *Keccak*
 - *Skein*

Winner: SHA-3 Algorithm

- NIST announced KECCAK as the winner of the SHA-3 Cryptographic Hash Algorithm Competition and the new SHA-3 hash algorithm in a press release issued on October 2, 2012.

Winner: SHA-3 Algorithm

- KECCAK was designed by a team of cryptographers from Belgium and Italy, they are:
 - Guido Bertoni (Italy) of STMicroelectronics,
 - Joan Daemen (Belgium) of STMicroelectronics,
 - Michaël Peeters (Belgium) of NXP Semiconductors, and
 - Gilles Van Assche (Belgium) of STMicroelectronics.
- http://csrc.nist.gov/groups/ST/hash/sha-3/winner_sha-3.html

CSCI361

Computer Security

Key Management

Outline

- Symmetric keys.
 - Diffie-Hellman Key exchange.
 - Man-in-the-Middle attack.
- PKC keys.
 - Certificates.
 - X.509.
 - Key generation and storage.
 - Handshakes.
 - Certification paths.

Key management

- In a cryptosystem, symmetric or public-key, it is necessary for users to obtain the required keys. This creates another security problem.
- Secure communication depends on secure key exchange (distribution/establishment/generation).

- For **secret key cryptography** the main problems are *key management* and *key distribution*.
 - Keys need to be distributed via *secure channels*.
- In **public key (PK) systems**, we have the problem of *key authentication*. Which key (really) belongs to who?
 - Keys need to be distributed via *authentic channels*.

Symmetric key establishment

- In these systems the two users must share a common key. This sharing can be achieved in a number of ways:
 1. Two users use a *supplementary secure channel*, such as a courier service.
 - **Disadvantages:** Costly, slow, questionable security.
 2. Key exchange via a *trusted authority*: Each user can communicate securely with **T**, a trusted central authority.
 - **Disadvantages:** Requires a trusted node and creates a bottleneck. For every key between two users at least two communications involving **T** are necessary. **T** can be replaced by a network of authorities, but this increases the number of entry points for the intruder.
 3. Key exchange using public channels: Something like the Diffie-Hellman key exchange protocol can be used.

Diffie-Hellman Key Exchange

- Alice and Bob agree on a common prime p and a common primitive element g of \mathbb{Z}_p .
- **Step One:** Secret keys X_A and X_B .
 - A chooses a random number X_A , $1 \leq X_A \leq p$.
 - B chooses a random number X_B , $1 \leq X_B \leq p$.
- **Step Two:** Public keys Y_A and Y_B .
 - A calculates $Y_A = g^{X_A} \text{ mod } p$.
 - B calculates $Y_B = g^{X_B} \text{ mod } p$.

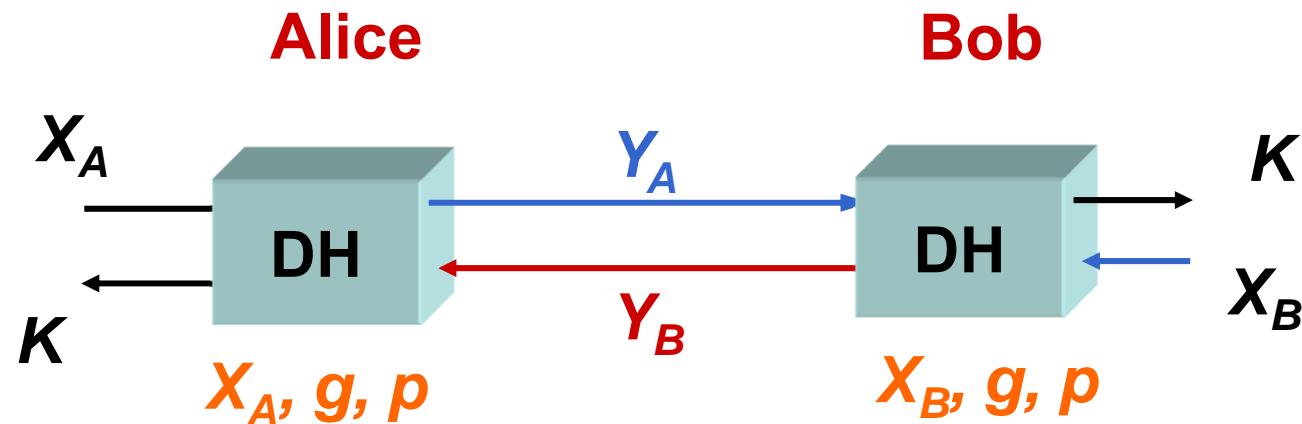
A → B : Y_A

B → A : Y_B

- Step Three:
 - A calculates $(Y_B)^{X_A} \bmod p$.
 - B calculates $(Y_A)^{X_B} \bmod p$.
 - These factors both equal $g^{X_A X_B} \bmod p$
- The security of this system depends on the difficulty of computing discrete logarithms.
 - In this context obtaining X_A from Y_A .

Diffie-Hellman Key Exchange

■ The Protocol



1: A → B: Y_A

2: B → A: Y_B

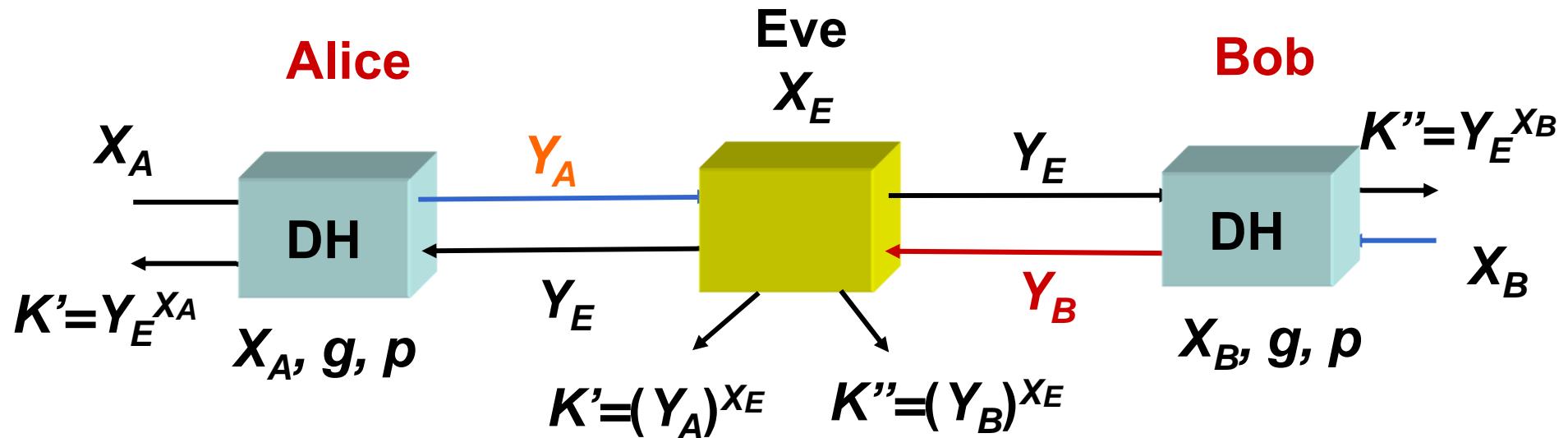
■ Example: $p = 13$, $g = 2$.

- A chooses $X_A = 4$.
- B chooses $X_B = 5$.
- $Y_A = 2^4 \bmod 13 = 3$.
- $Y_B = 2^5 \bmod 13 = 6$.
- $(Y_B)^{X_A} \bmod p = 6^4 \bmod 13 = 9$.
- $(Y_A)^{X_B} \bmod p = 3^5 \bmod 13 = 9$.

Alice and Bob have established a common key 9.

Diffie-Hellman Key Exchange

■ Man-in-the-Middle Attack



1: A \rightarrow E: Y_A

2: E \rightarrow B: Y_E

3: B \rightarrow E: Y_B

4: E \rightarrow A: Y_E

There are various ways of fixing this problem (although we aren't going to look at them here).

Key Exchange in PKC

- To use a public key algorithm, **Alice** must have **Bob's** public key.
- This can be obtained
 - Directly from **Bob**.
 - From a public directory.
- While the public component need not be kept secret, **authenticity** and **integrity** are real problems:
 - **Authentication**: If **Alice** thinks that Z_o is **Bob's** key, whereas it is actually **Oscar's** key, then **Alice** might encrypt using Z_o and (unknowingly) allow **Oscar** to decrypt the message.
 - **Integrity**: Public keys are generally over 1,000 bits. A single bit error transmission of the public component makes it useless.

- Almost all key exchange systems have some kind of trusted authority. However, the implication of compromise of the trusted authority is not as severe as in the symmetric key system, mainly because the trusted authority does not necessarily know the secret component and so cannot read the communications.
- *Validity* is an important consideration. Keys are usually valid for certain period of time, their *lifetime*. Key updates may be:
 - Sent to users in real time.
 - Obtained by the users during periodic checks with the trusted authority.

Certificates

- A useful technique that provides a partial solution to the authenticity and integrity problems, is the use of **certificates**. This technique assumes:
 - A central (trusted) authority **T**.
 - A secure communication channel between each user and **T**: for example each user knows Z_T , the public key of **T**.

- The system can be sketched as follows:
 - Alice securely sends Z_A to T.
 - Alice receives a certificate, C_A , signed by T that binds Z_A to Alice.
 - This certificate is verifiable by everyone who has the public key of T.
- A certificate has the following form:
$$M = [Z_A, \text{Alice's ID}, \text{validity period}].$$
$$C_A = D_{Z_T}(M)$$
- Look at a web browser, say Mozilla, under security preferences to see some certificates.

Encryption with a certificate

- When **Alice** wants to send an encrypted message to **Bob**:
 1. She obtains **Bob's** certificate.
 2. Verifies the signature of **T** on the certificate.
 3. Extracts Z_B and uses it to encrypt the message.
- A certificate may be invalidated before its expiry date if the key is known to be compromised.
The central authority must periodically issue a list of invalidated certificates.

Key distribution in a certificate based PKC

- C_A and C_B are certificates issued by T . Alice may obtain Bob's certificate by:
 - Asking Bob directly.
 - Looking in a public directory.
- In both cases, there is a potential integrity problem, in particular the certificate obtained may have recently been invalidated.
- One solution is to obtain Bob's certificate through T .
 - **Advantage:** The key Z_B is authentic.
 - **Disadvantage:** T is a bottleneck, concentration of trust, etc.
- Alternatively one can implement a security policy which advises persons to check for up-to-date invalidations, perhaps a list on the trusted authorities website say, before using the key.

Who should generate keys?

- If T generates keys pairs:
 - **Advantages:** Keys may be chosen more carefully.
Neither of the participants is advantaged.
 - **Disadvantages:** T can eavesdrop! Moreover it again produces a bottleneck in the system.
- If users generate their own keys the latter problem disappears.
- Users also need some provision for accessing previous keys so that the files that are encrypted with them remain accessible.

PKC for symmetric key distribution

- A major application of PKC is the distribution of symmetric keys. PKC can be used to establish the common information (secret key, initializing vectors, etc.) required by a symmetric cryptosystem.
- PKC can be used to send messages securely and authentically. The information shared for the symmetric cryptosystem is just a particular message.
- Remember that symmetric key cryptography is significantly faster than PKC for equivalent security.

CSCI361

Computer Security

Secret sharing and its applications

Outline

- Motivation
- Secret sharing: model
 - Threshold schemes.
 - General schemes.
- Verifiable secret scheme
- Application

Motivation

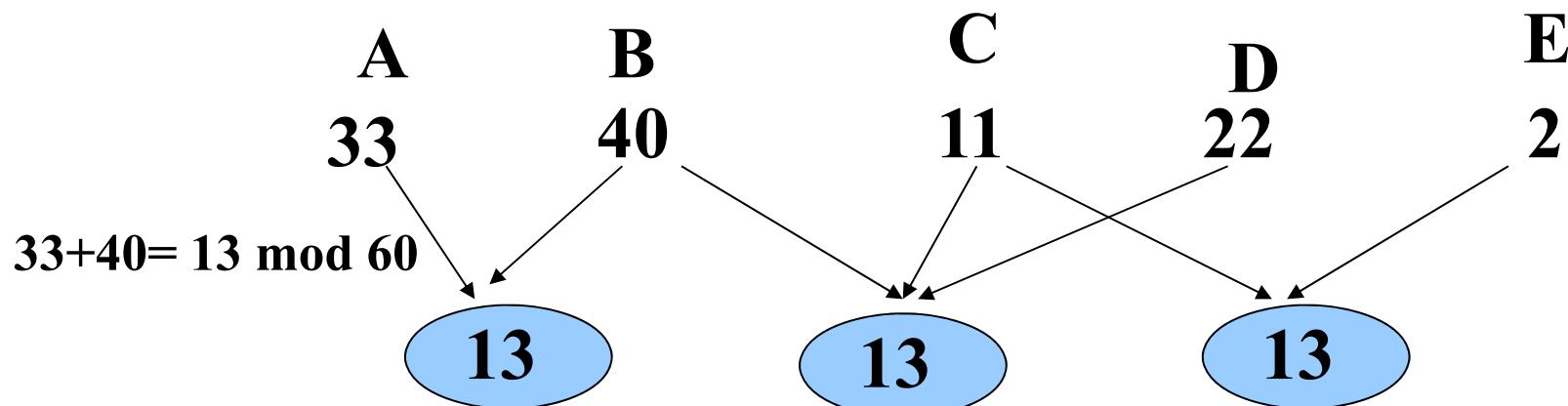
- **Principle of reduced trust:**
 - to keep a secret safe and also make the system more robust, it is best if less power is given to a single entity
 - A secret key used to encrypt a file system should not be entrusted to one person
 - What if he loses the secret?
 - He leak the secret
- **Distributing trust gives a solution to both of the above problems.**
 - Key recovery system
 - Dishonest user

Key Escrow /Key Backup

To provide **key backup**:

- Divide the secret key into pieces
- Distribute the pieces to different servers such that certain subgroups of servers can recover the key
- Consider RSA system.
- $N = 7 \times 11 = 77$, $\varphi(N) = 6 \times 10 = 60$
- $d = 13$, $e = d^{-1} = 37 \text{ mod } 60$

**$A \wedge B, B \wedge C \wedge D, C \wedge E$
can recover the
secret**



Key escrow can be (mis)used :

- In 1991 the U.S. government attempted to introduce a new standard which would enable the government to read all private communications
 - Private key is broken into two halves:
 - The government keeps one half
 - Another authority the other half
 - A court order allows an agency to access both halves
- This standard was not successful.

A numerical example

- Consider a six digit combination lock.
 - The combination can be shared among 4 people.
 - Any three can calculate the combination.
 - No two people can calculate the combination.

Person	c_1	c_2	c_3	c_4	c_5	c_6
One	1	1	1	0	0	0
Two	0	0	1	1	1	0
Three	1	0	0	0	1	1
Four	0	1	0	1	0	1

Each c_i appears twice. As long so no more than one person is missing, somebody present knows c_i .

This is a *threshold secret sharing scheme*.

Shamir's Secret Sharing (1979)

- A threshold scheme using polynomial interpolation.
- An honest dealer D distributes a secret s among n users, such that at least t users must collaborate to find the secret
 - less than t players cannot have any information about the secret

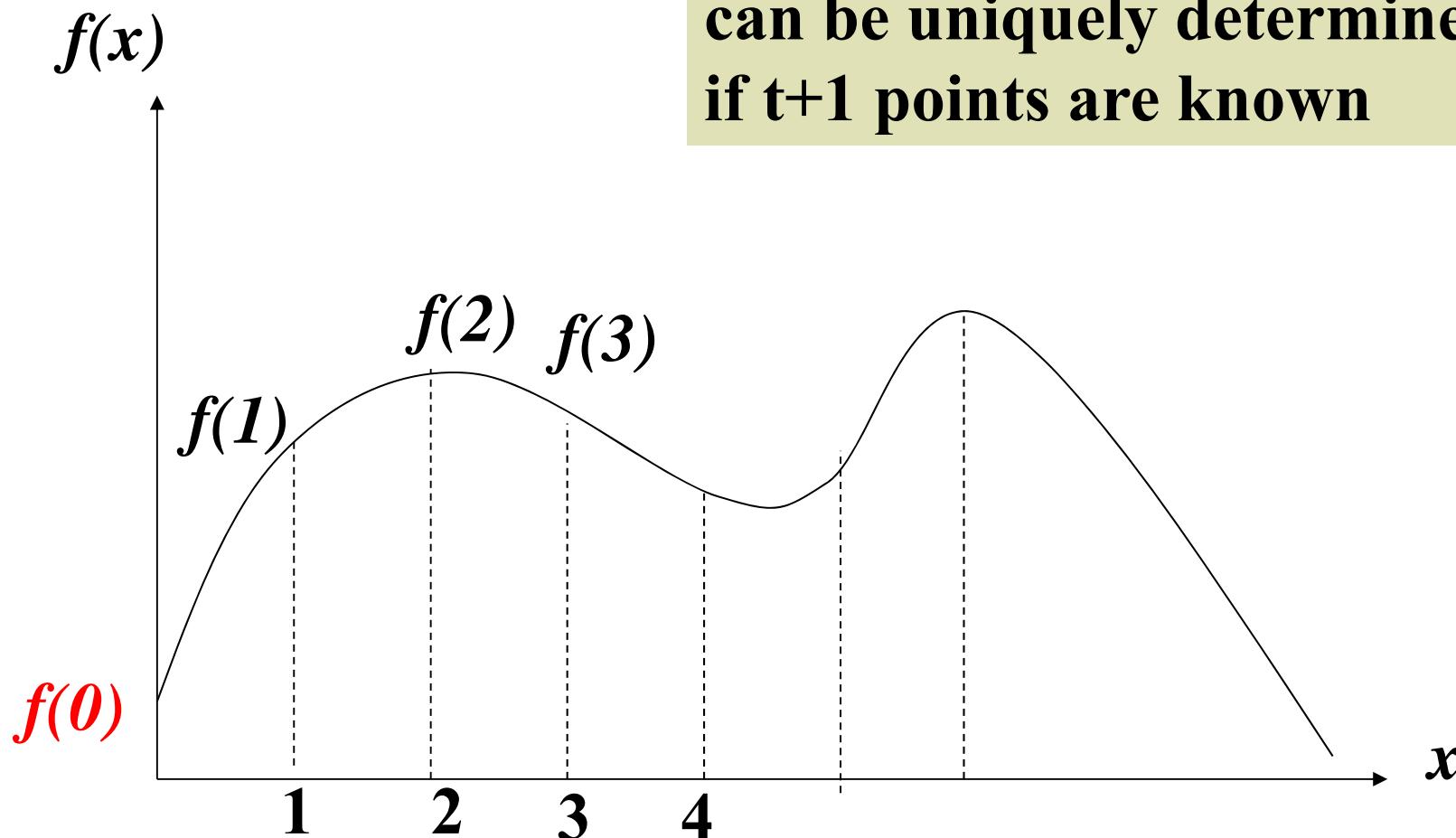
The scheme

- We want to share a secret s among users $U_1, U_2..U_n$, such that any t users can reconstruct the secret.
- Dealer D constructs a random polynomial $f(x)$ of degree $t-1$ such that $a_0 = s$.
$$f(x) = a_0 + a_1 x + \dots + a_{t-1} x^{t-1}$$
- This polynomial is constructed over numbers modulo a prime p , p is public.
- For user U_j , Dealer does the following
 - Choose x_j
 - Calculate $f(x_j)$
 - Such that all x_i $i=1,...n$, are distinct
 - User U_j gets $(x_j, f(x_j))$
- (U_j, x_j) is public
 - Without losing generality, we can assume $x_j=j$

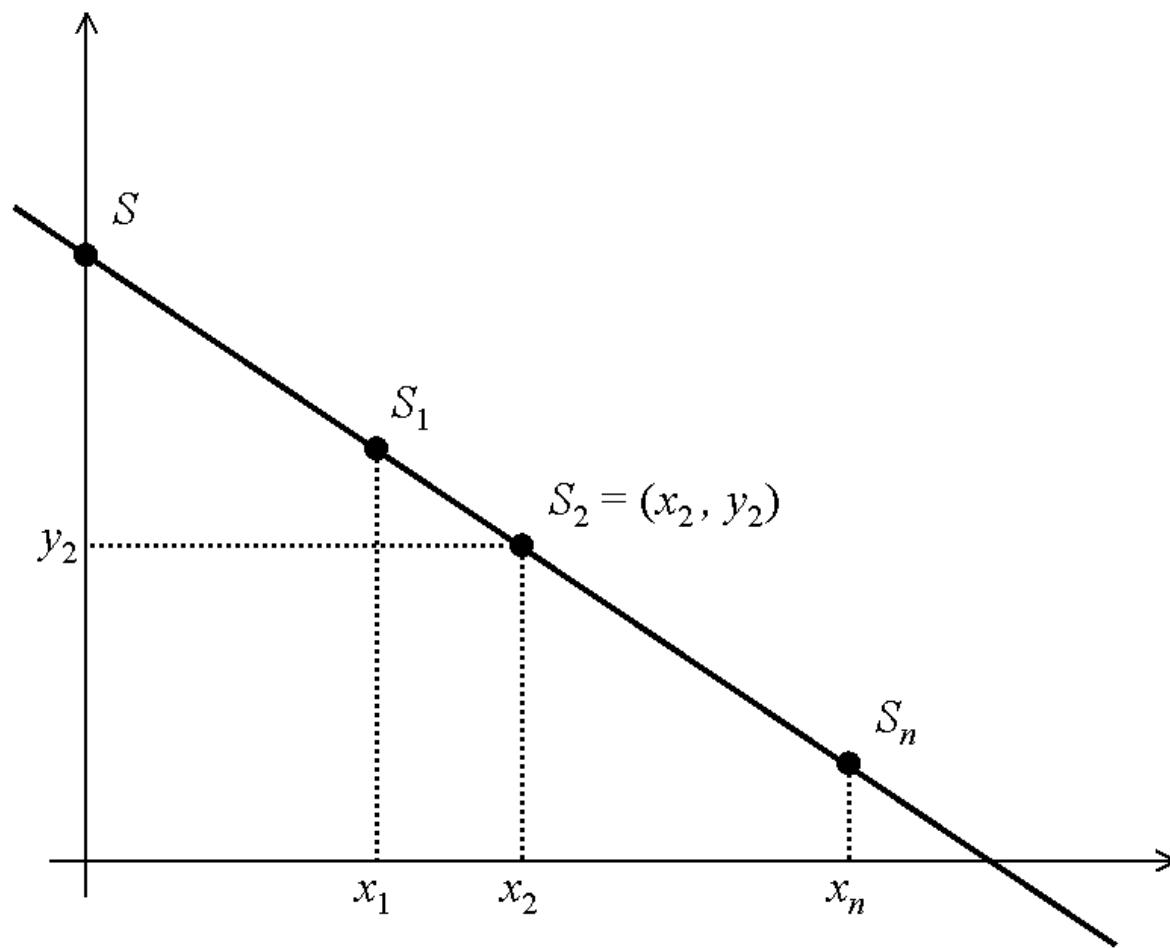
The Reconstruction Protocol

- Find the unique polynomial $f(x)$ such that $f(x)=f(j)$ and for $j=1,2,\dots,t$
- Reconstruct the secret to be $f(0)$.

A polynomial of degree t
can be uniquely determined
if $t+1$ points are known



$$t=2, \quad f(x)=a+bx$$



Lagrange interpolation

- Suppose you have n pairs $(x_i, y_i = f(x_i))$ and want to find the polynomial f .
- The polynomial of degree $n-1$ through the data is given by Lagrange interpolation.

$$f(x) = \sum_{j=1}^n f_j(x) \quad f_j(x) = y_j \prod_{k=1, k \neq j}^n \frac{(x - x_k)}{(x_j - x_k)}$$

Consider a (3,6)-SSS over \mathbb{Z}_7 .

1. Let $x_i = i$, $i=1 \dots 6$.
2. The secret is 3.
3. $f(x) = 3 + 3x + 3x^2$.
4. Share table

Share	s_1	s_2	s_3	s_4	s_5	s_6
Value	2	..	4	3

5. Assume P_1 , P_3 and P_6 cooperate, each giving an equation.

$$2 = k + a_1 + a_2$$

$$4 = k + 3a_1 + 2a_2$$

$$3 = k + 6a_1 + a_2$$

Finding k with Lagrange interpolation

The data: $Y_1=f(1)=2$, $y_3=f(3)=4$, $y_6=f(6)=3$.

$$\begin{aligned}f(0) &= \frac{(-x_3)(-x_6)}{(x_1 - x_3)(x_1 - x_6)} y_1 \\&\quad + \frac{(-x_1)(-x_6)}{(x_3 - x_1)(x_3 - x_6)} y_3 \\&\quad + \frac{(-x_1)(-x_3)}{(x_6 - x_1)(x_6 - x_3)} y_6 \\&= 2 \times \frac{(-3)(-6)}{(1-3)(1-6)} + 4 \times \frac{(-1)(-6)}{(3-1)(3-6)} + 3 \times \frac{(-1)(-3)}{(6-1)(6-3)} \\&= 3\end{aligned}$$

Properties of Shamir's SS

- **Perfect Security**
 - t users can find a unique secret ,
 - $t-1$ users cannot learn anything
- **Ideal**
 - Each share is exactly the same size as the secret.
-
- **Extendable**
 - More shares can be created
 - New users joining the system
- **Flexible**
 - can support different levels of trust
 - Given more share to more trusted people

Homomorphic property

- $f(1), f(2) \dots f(n)$ are shares of polynomial $f(x)$
- $g(1), g(2) \dots g(n)$ are shares of polynomial $g(x)$
- Then $f(1) + g(1), f(2) + g(2) \dots f(n) + g(n)$ are shares of $f(x) + g(x)$
 - That is the secret $f(0) + g(0)$

→ to multiply a secret by a constant, each share holder has to multiply by the same constant

Example

- Sharing $s=5$ among 7 people such that any three can find the secret
- $f(x) = 5 + 2x + 3x^2 \pmod{11}$
 $f(1)=10, f(2)=10, f(3)=5, f(4)=6, f(5)=2, f(6)=9, f(7)=1$
- Sharing $s=7$ among the same people
- $g(x) = 7 + x + x^2 \pmod{11}$
 $g(1)=9, g(2)=2, g(3)=8, g(4)=5, g(5)=4, g(6)=5, g(7)=8$
- Shares of $s=1$ for the same people
 - $1 (=5+7 \pmod{11})$
 - $u(x) = f(x) + g(x) = 1 + 3x + 4x^2 \pmod{11}$
 $u(1)=8, u(2)=1 \dots$

Verifiable secret sharing

- Dealer is not trusted
- Dealer needs to ‘prove’ that the shares are **consistent shares**
 - Every $t-1$ subset gives the same secret
- A verifiable secret sharing system allows users to check validity of their shares
- Two versions
 - Interactive proofs
 - Requires interaction between dealer and participants
 - costly
 - non Interactive proofs
 - dealer can send messages,
 - the shareholders cannot talk with each other or with the dealer (for share verification).
 - They can use public information to check validity of shares

Threshold signature

Threshold RSA

- Public key (e, N) , secret key (d, N)
- Share secret key among users:
 - d_1, d_2, \dots, d_n using an [extension of Shamir's scheme](#)
- For a message m that t users agree on, each user produces a partial signature
 $H(m)^{d1}, H(m)^{d2} \dots H(m)^{dt}$
- Combiner combines these partial signatures (e.g. multiply them) to obtain
$$H(m)^d = H(m)^{d1} \times H(m)^{d2} \times \dots \times H(m)^{dt}$$
- The signed message is $(m, H(m)^d)$
- Verification is as usual
- Given (m, s) , we check $H(m) = s^e \bmod N$

CSCI361

Fair Exchange and Zero Knowledge Proofs

Fair Exchange

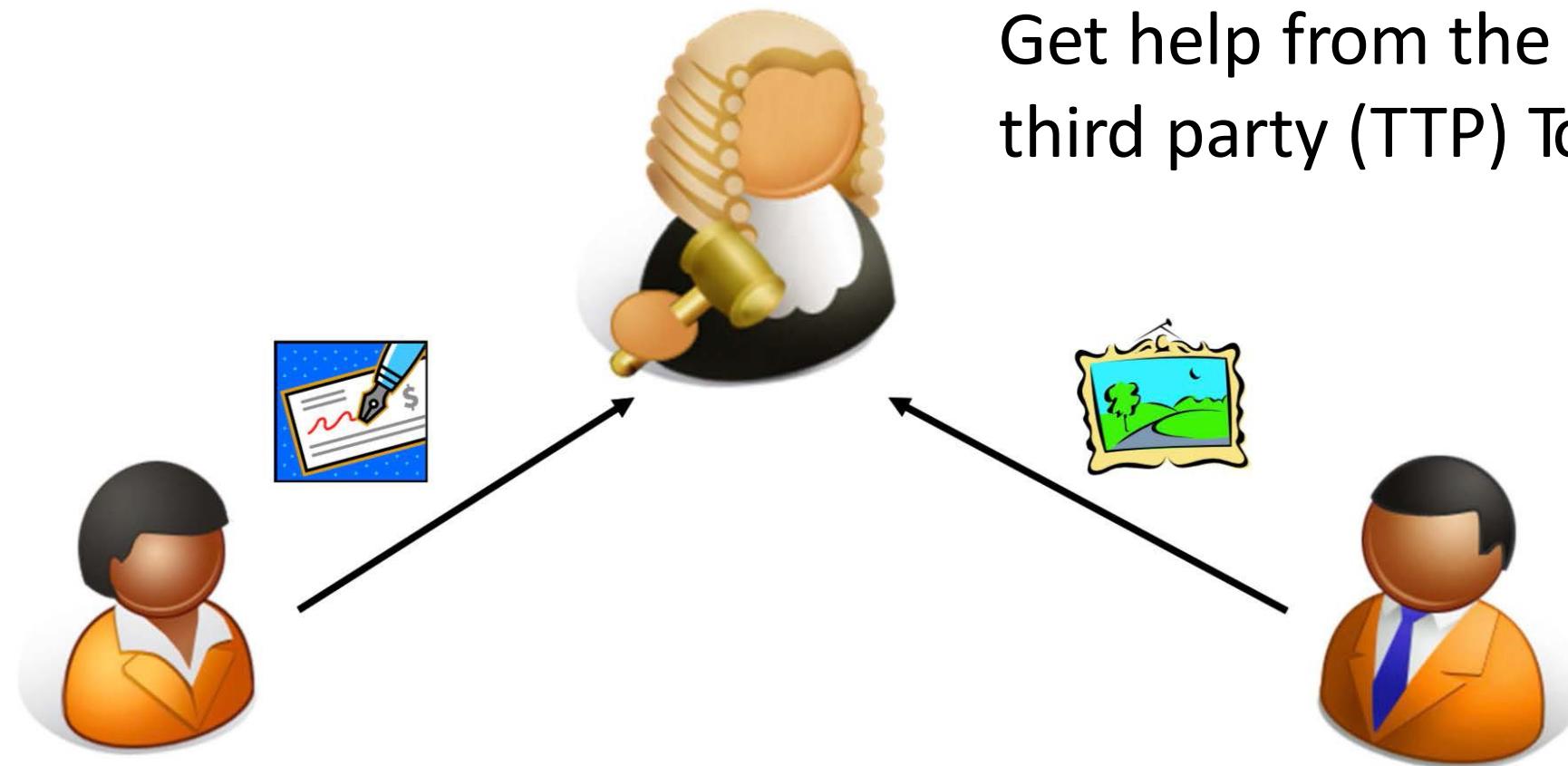
- Two parties, Alice and Bob, would like to exchange something

Let's say Alice wants to get a digital photo from Bob and Bob wants to get Alice's eCheque (which Alice signed) in exchange. We want to make sure that this transaction is **fair**.

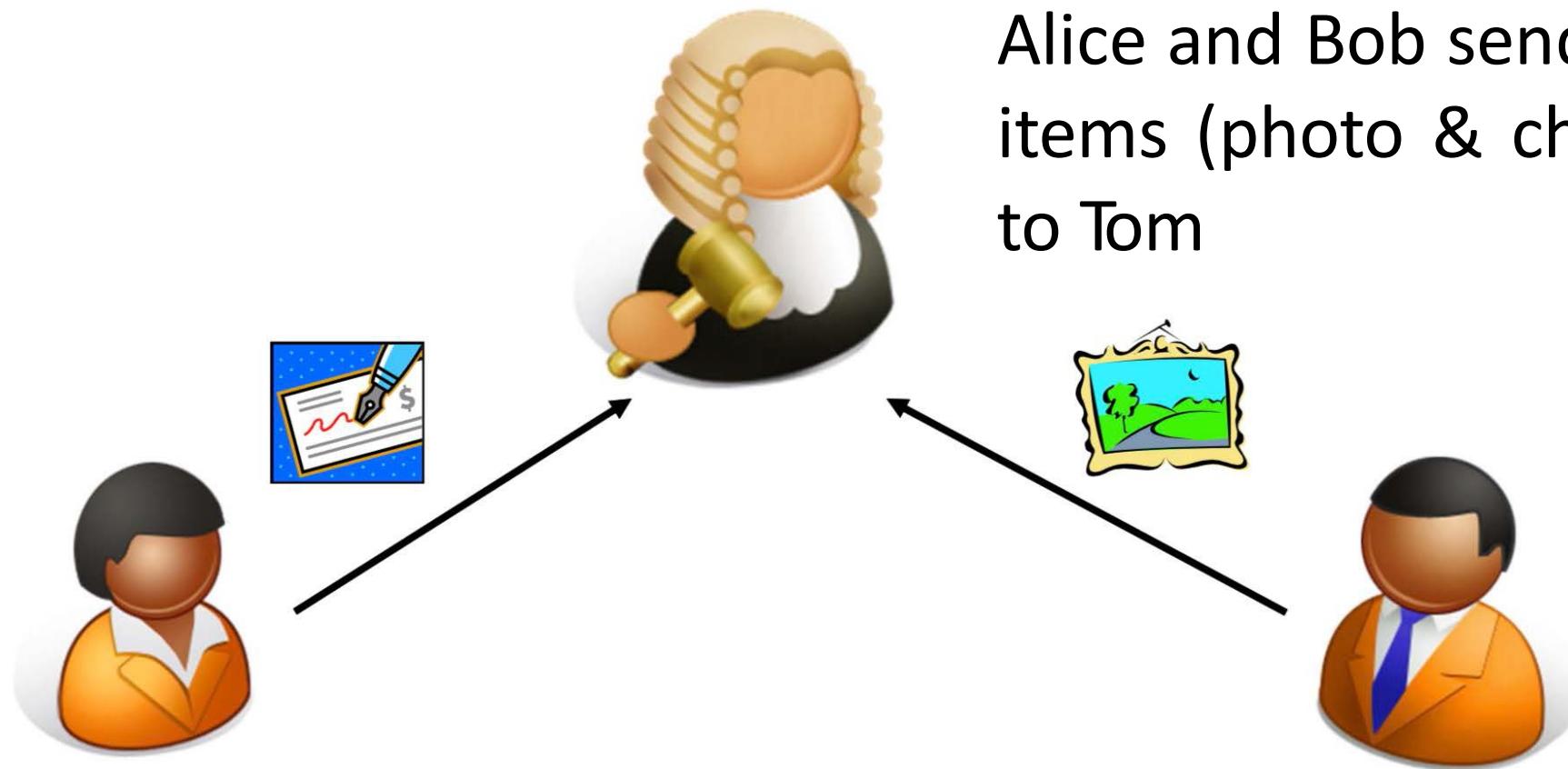
- What is **fairness** then?

- (In the context of electronic commerce), Participants shouldn't have advantages over each other.
- For example: It wouldn't be fair if one party can avoid their obligations in a contract if the other party has completed their obligations in a contract.

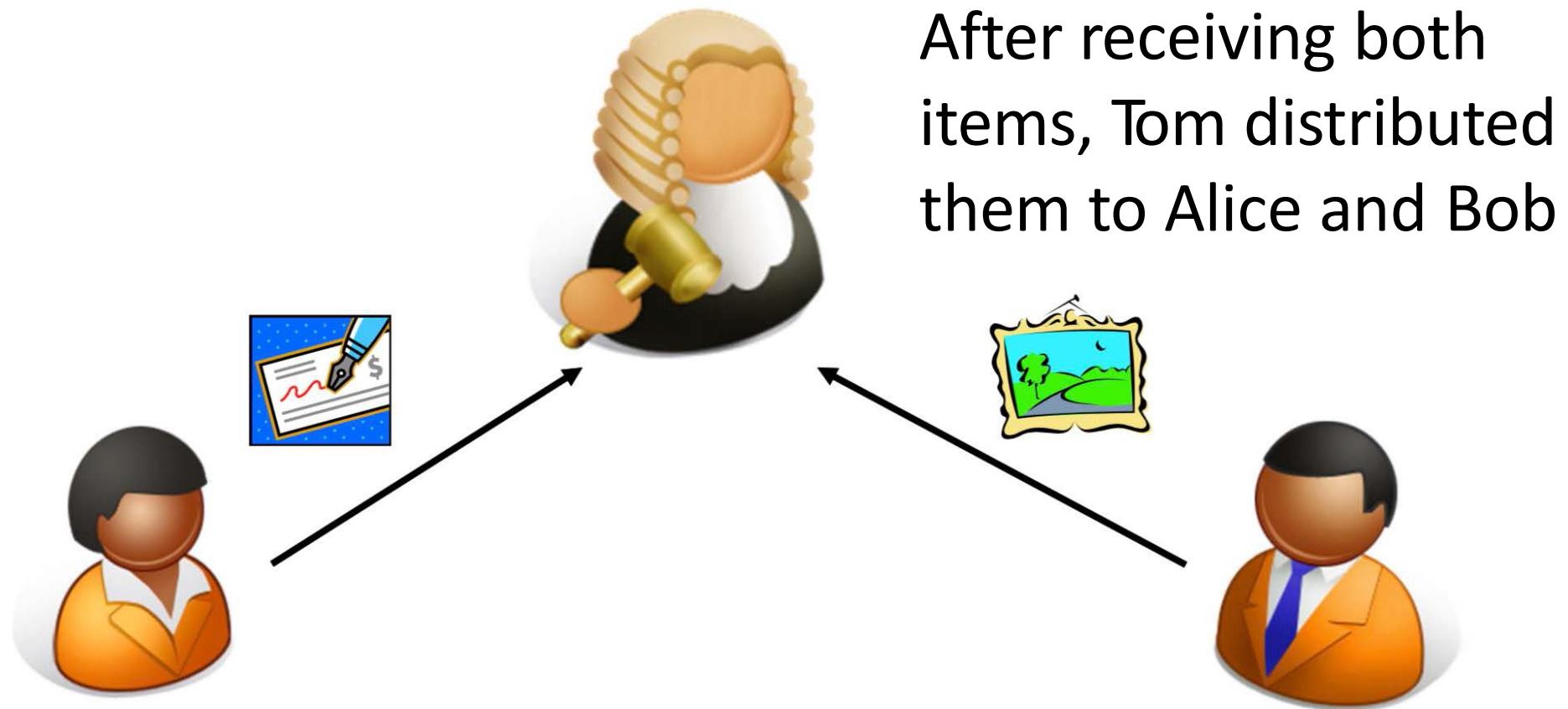
Possible Solution



Possible Solution



Possible Solution



Good...

- But Tom must be accessible all the time
- Can Alice and Bob conduct the exchange without the *active* participation from Tom? (Tom has to receive and send something from Alice and Bob.)

Optimistic Fair Exchange

Scenario: Alice wants to get a digital item from Bob and Bob wants to get Alice's signature in exchange.

- (Conceptual) Solution
 - Alice creates something called a **partial signature (P)**
 - Alice sends P to Bob.
 - Bob sends Alice his item.
 - Alice sends her full signature (S) to Bob
 - ❖ If Alice runs away after getting Bob's item, Bob can go to Tom and asks Tom to convert P into S.

Optimistic Fair Exchange - Realisation

1. Alice generates her signature, S_A .
2. Instead of sending S_A directly to Bob, Alice encrypts S_A under Tom's public key. The resulting ciphertext $P = E_{pk}(S_A)$ is the partial signature.
3. Alice sends P to Bob.
4. Bob sends his item to Alice.
5. Alice sends S_A to Bob.
 - If Alice does not send S_A to Bob, Bob can contact Tom with P and ask him to decrypt P to S_A . (Bob can finally obtain S_A .)

Optimistic Fair Exchange - Realisation

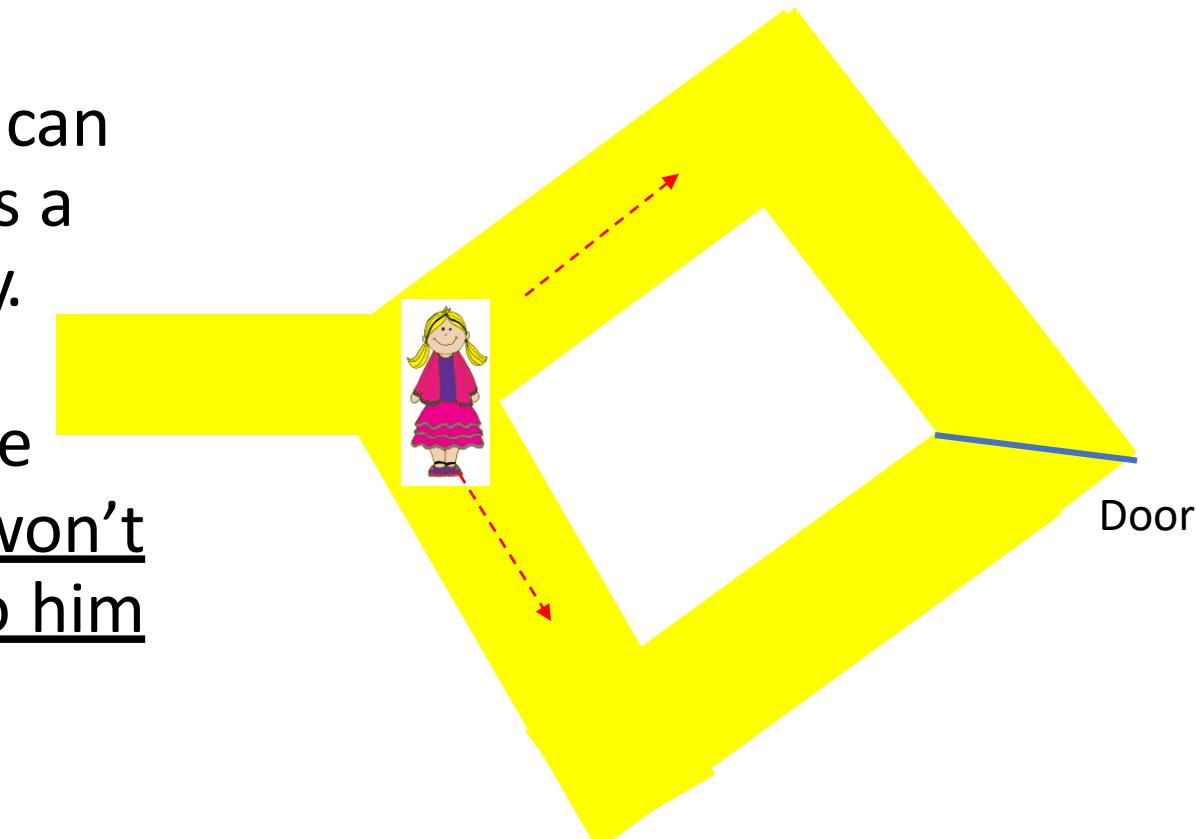
- Additional step
 - ✓ How can Bob make sure C contains a valid signature of Alice?
 - ✓ Using zero-knowledge proof → Prove that P is the encryption of a valid S_A **without revealing S_A**

Beyond Optimistic Fair Exchange

- Optimistic Fair Exchange can solve the problem of having to require active TTP
- There is a Fair Exchange scheme that does not require TTP at all!

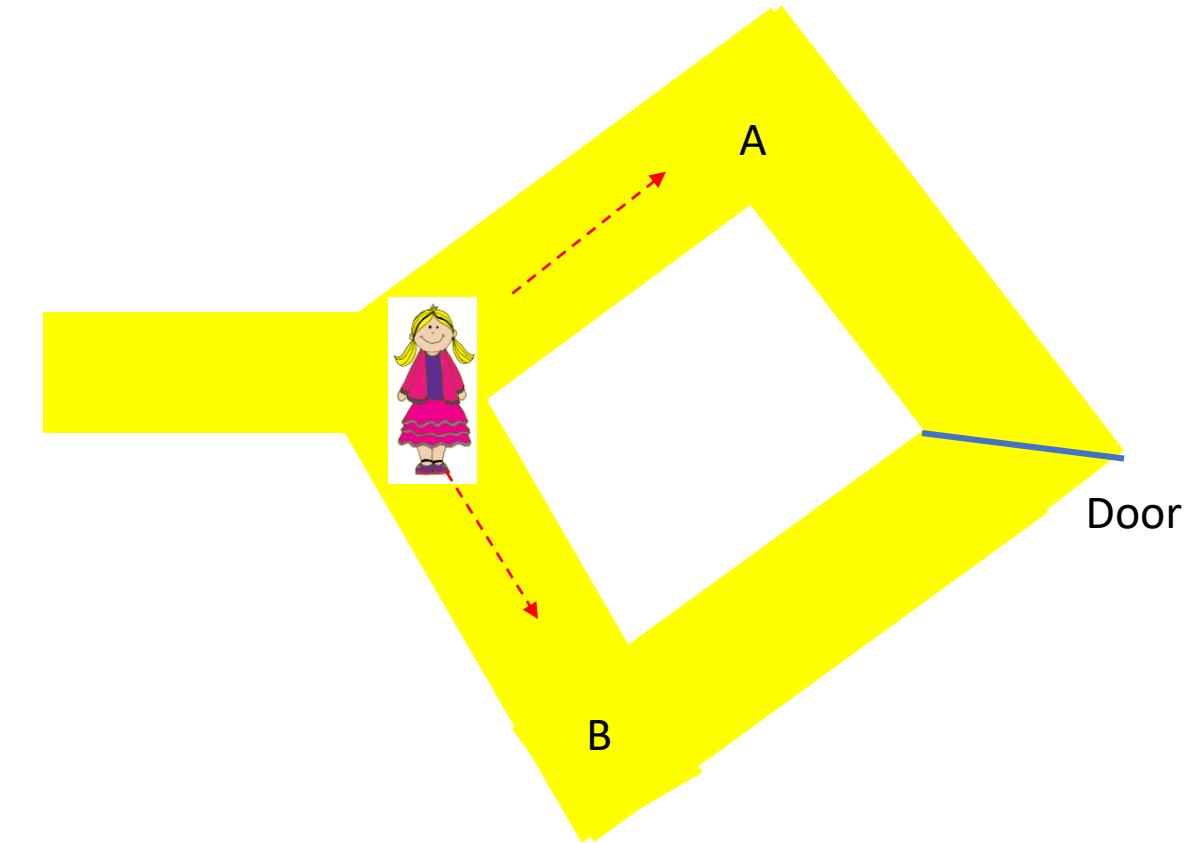
Zero Knowledge Proof: Ali Baba Cave Problem

- Setting: There is a ring-shape cave where the path is blocked by a door. Peggy can open the door if she uses a right magic word as a key. Victor wants to know whether Peggy knows the magic word. But Peggy won't reveal the magic word to him (or anyone else).



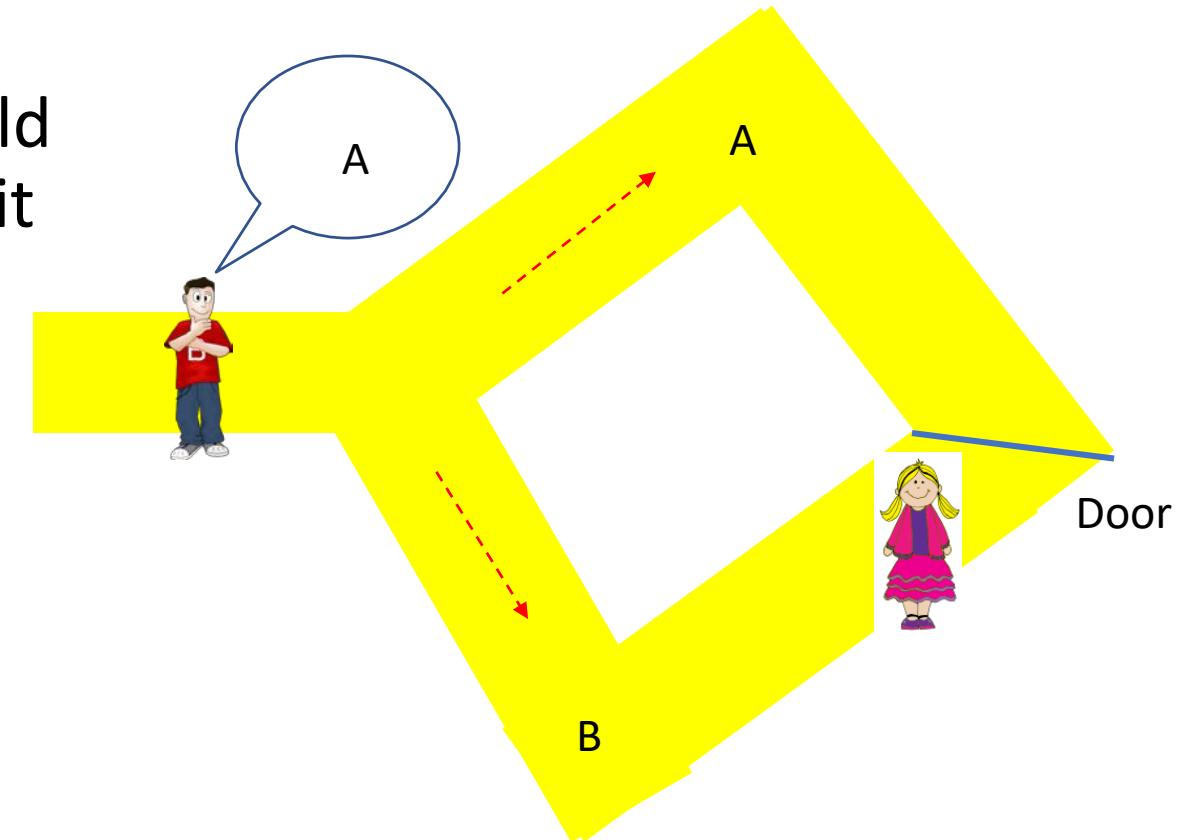
Zero Knowledge Proof: Ali Baba Cave Problem

- Step 1: Peggy chooses either path A or B and moves towards the door while Victor waits outside



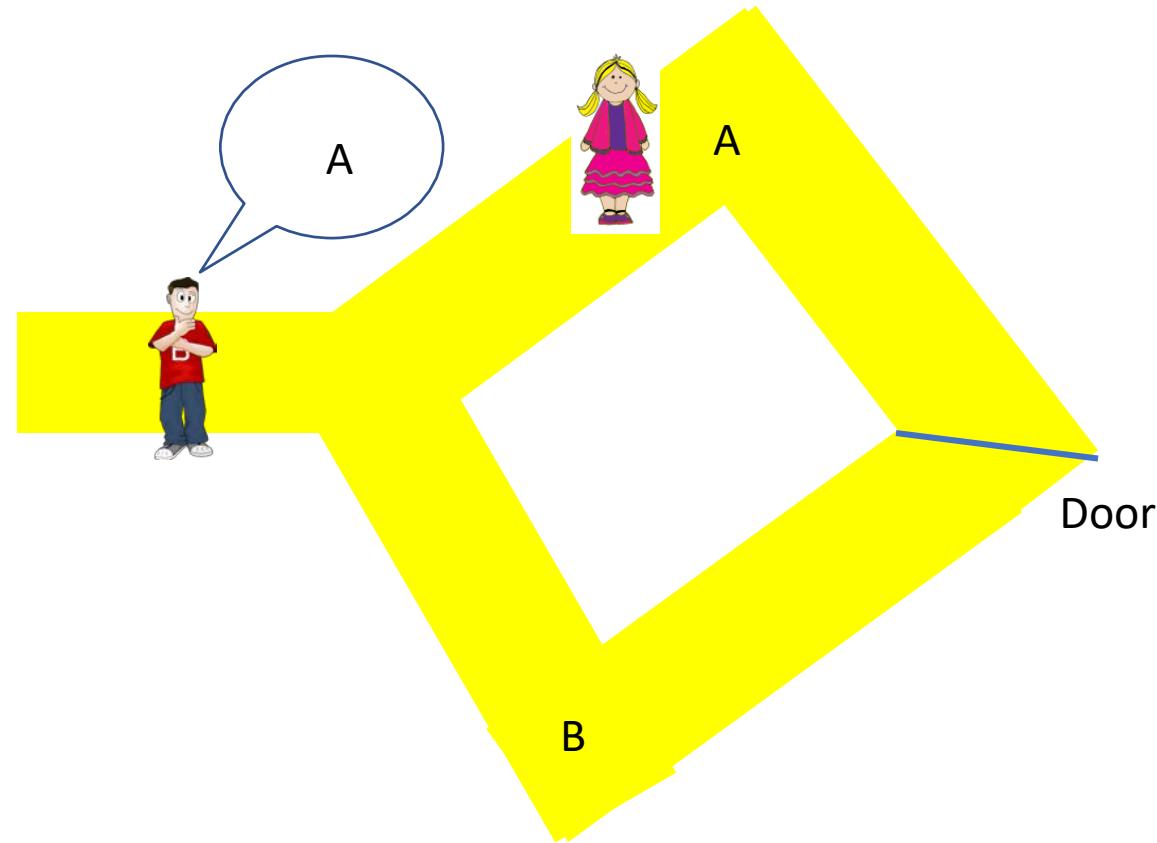
Zero Knowledge Proof: Ali Baba Cave Problem

- Step 2: Now Victor enters the cave, picks at random the name of path Peggy should use to return and shouts it out!



Zero Knowledge Proof: Ali Baba Cave Problem

- Step 3: Peggy returns to the entrance using the path Victor named



Zero Knowledge Proof: Ali Baba Cave Problem

- If Peggy knows the magic word, she can reliably return to the entrance using the path Victor named
- If Peggy does not know the magic word, she can return to the entrance only if Victor named the path she chose in Step 2 with probability $\frac{1}{2}$
- Victor repeat Step 1 to 3 many times to see whether Peggy really knows the magic word
 - In n trials, the probability that Peggy returns through the path without knowing the magic word is $(\frac{1}{2})^n$

CSCI361

Introduction to Blockchain and Cryptocurrency

Features of Cryptocurrency

- Peer-to-peer transfer of electronic money
- No centralised entity to control transactions
- Based on the cryptographic primitives
 - ✓ Digital signature
 - ✓ Hash function
- There are numerous cryptocurrencies nowadays → Bitcoin, Ethereum, Ripple, ...etc

Ledger – Where Everything Begins

- Financial transaction is all about managing a ledger
- The ledger records deposits

Alice deposits \$100

Bob deposits \$70

Charlie deposits \$130

Diana deposits \$90

Ledger

- And debits

...Deposits...

Alice pays Bob \$10

Bob pays Charlie \$40

Charlie pays Diana \$30

Diana pays Alice \$35

Ledger

- Assume that the ledger is public i.e., accessible from anyone

...Deposits...

Alice pays Bob \$10

Bob pays Charlie \$40

Charlie pays Diana \$30

Diana pays Alice \$35

❖ Anyone can add lines to the ledger.

Ledger

- Problem

...Deposits...

Alice pays Bob \$10

Bob pays Charlie \$40

Charlie pays Diana \$30

Diana pays Alice \$35

❖ Bob may want to put “Alice pays Bob \$100” to the ledger. (Why? So obvious!)

Ledger

- How to prevent this? – Through Digital Signature!

...Deposits...

Alice pays Bob \$10 S_{Alice}

Bob pays Charlie \$40 S_{Bob}

Charlie pays Diana \$30 $S_{Charlie}$

Diana pays Alice \$35 S_{Diana}

- ❖ Alice can generate a digital signature on “Alice pays Bob”. Other people also can do the same.
- ❖ As signature is unforgeable, it’s hard for an adversary to add a line without knowing the private key.

Ledger

- Hang on...What if Bob does the following?

...Deposits...

Alice pays Bob \$10 S_{Alice}

❖ All the lines are valid!

Ledger

- We need a unique sequence number for each transaction.

...Deposits...
1 Alice pays Bob \$10 S^1_{Alice}
2 Alice pays Bob \$10 S^2_{Alice}
3 Alice pays Bob \$10 S^3_{Alice}
4 Alice pays Bob \$10 S^4_{Alice}

- ❖ Each signature must be different and again it's hard to create one (or forge) without knowing the private key.
- ❖ Digital signature solved the problem of adding a valid transaction entry to the ledger!

Ledger

- Overspending : Oops! Bob has only \$70!

...Deposits...
1 Bob pays Alice \$30 S^1_{Bob}
2 Bob pays Charlie \$20 S^2_{Bob}
3 Bob pays Alice \$40 S^3_{Bob}

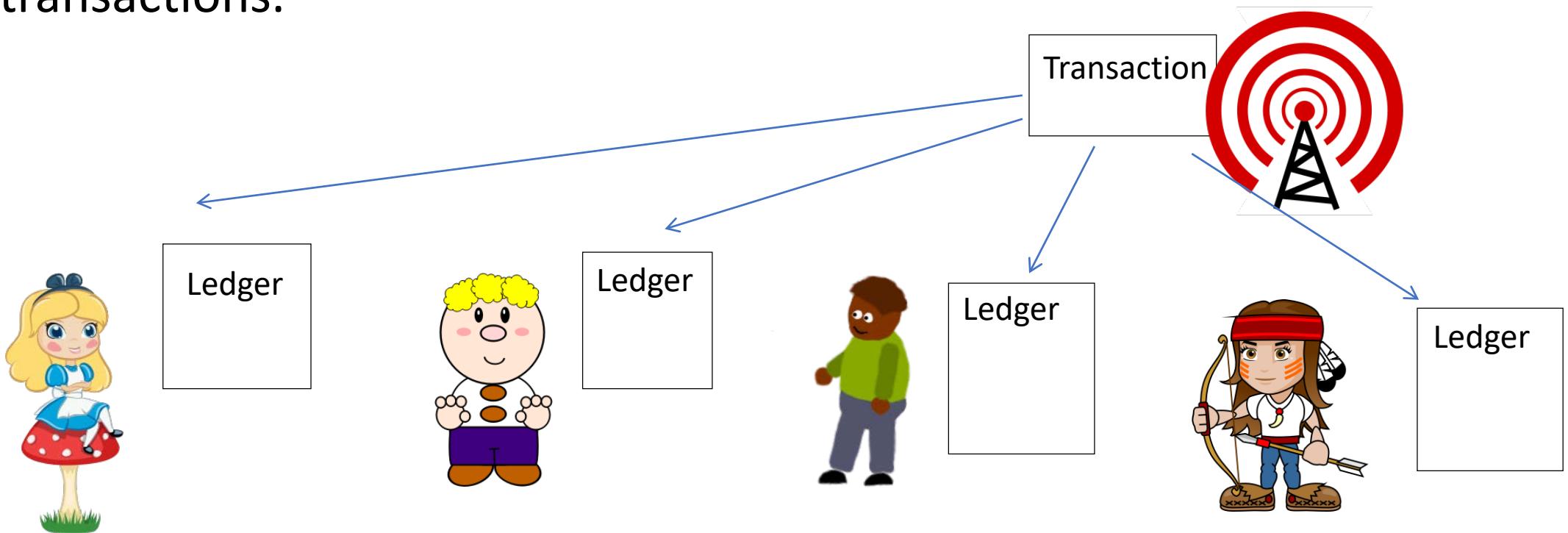
- ❖ Therefore we need to track Bob's all past transactions!
(We need to refer to Bob's deposit.)
- ❖ In general, we need to keep the history of the ledger.

Ledger

- The ledger just described is nothing new. – It can be one of the ledgers held by banks today
- We trust the ledger as we trust the bank.
- But cryptocurrencies want to remove the role of the central authority in managing transactions.
- How?

Private Ledger?

- The idea is to broadcast transactions so that every participant should record them on their private ledger to track and validate the transactions.

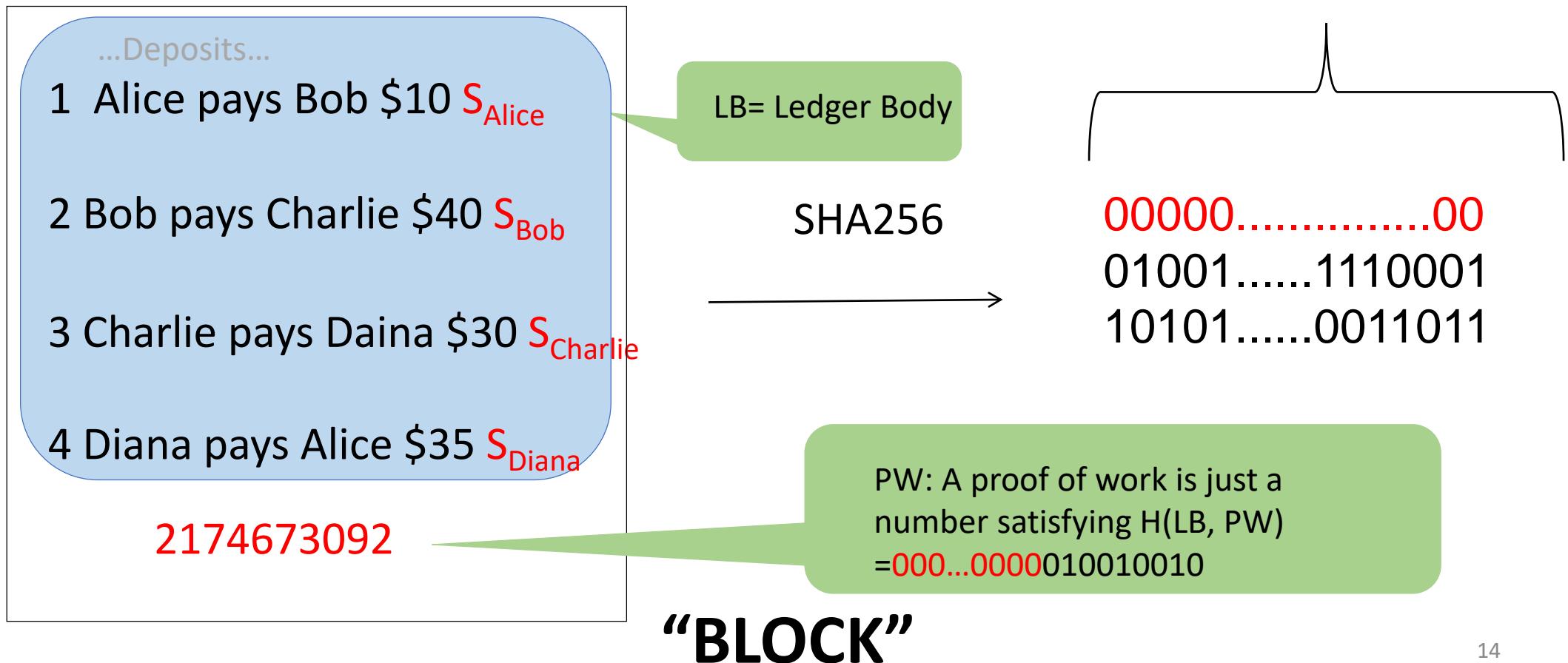


Further Problems to Solve

- But we can't just let participants record transactions on their ledger
 - ✓ The ledger should be the same for everyone.
 - ✓ Someone should consolidate the transactions in the ledger but we don't want a centralised entity to do it.
 - ✓ We ask someone who is interested to put time/efforts to consolidate the transactions.
 - ✓ That person is called a “**Miner**”.
 - ✓ He/She will perform the task by finding a “**proof of work**”.

Proof of Work

- Consolidate the ledger by finding a proof of work!

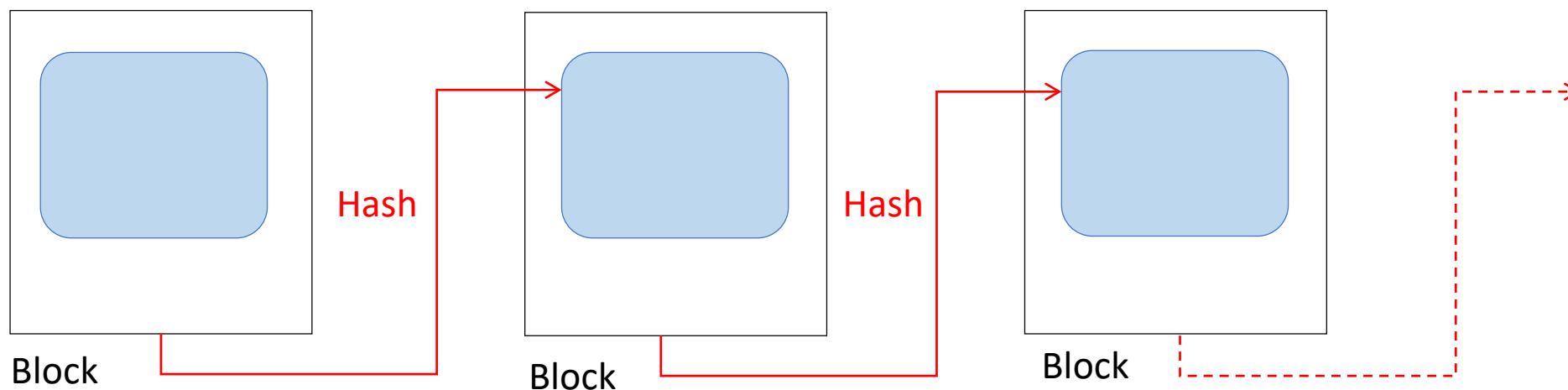


Proof of Work

- Is it hard to find such number?
 - ✓ Yes but not impossible.
 - ✓ If a Miner needs 30 leading zeros, he/she will have to try 2^{30} numbers on average assuming that hash output is random.
 - ✓ Have you seen a person running the bunch of computers for “mining”?
- Why does the Miner do it?
 - ✓ To get financial reward: Miners have to compete against each other to find a proof of work for a block of up to 2400 transactions every 10 minutes.
 - ✓ In Bitcoin, a proof of work is called a “Nonce”

Blockchain

- Chaining: We need to record the history of transactions consistently (No missing transactions/modifications..etc.): To achieve this, we have *not only* the ledger body *but also* the **previous block**



Blockchain

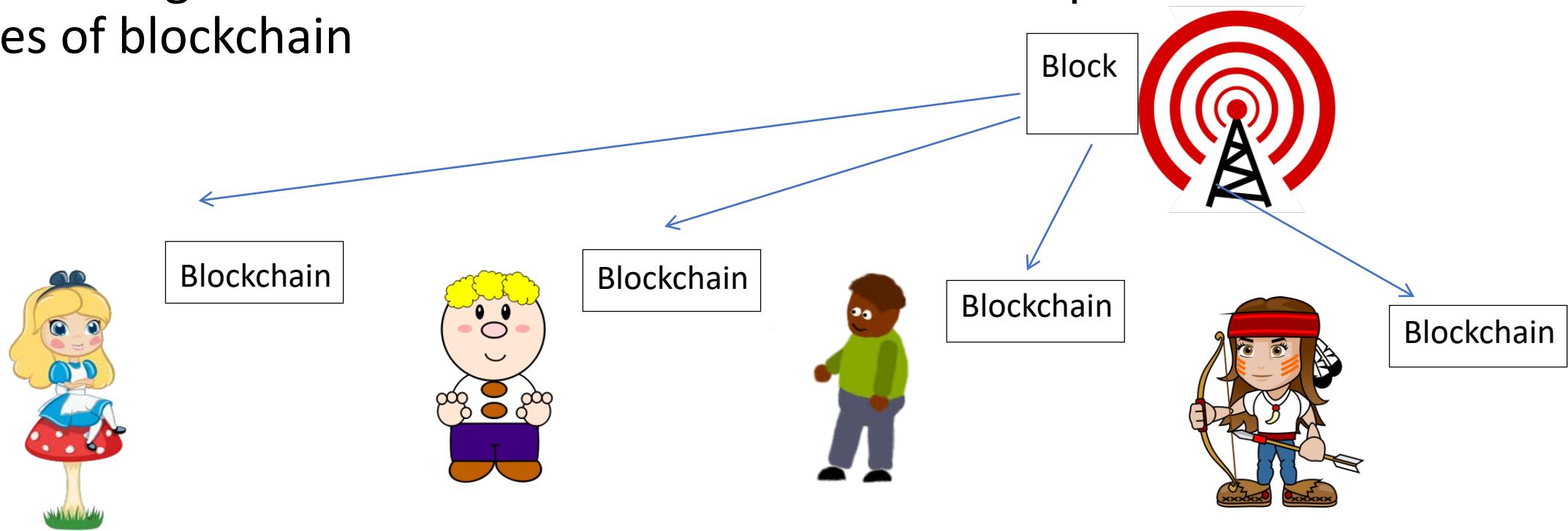
- A constantly growing ledger that keeps a permanent record of all the transactions that have taken place, in a secure chronological and immutable way.

Miners' Job (Summary)

- Listening for transactions and creating blocks
- Broadcasting those blocks
- Getting rewarded with money (cryptocurrency)

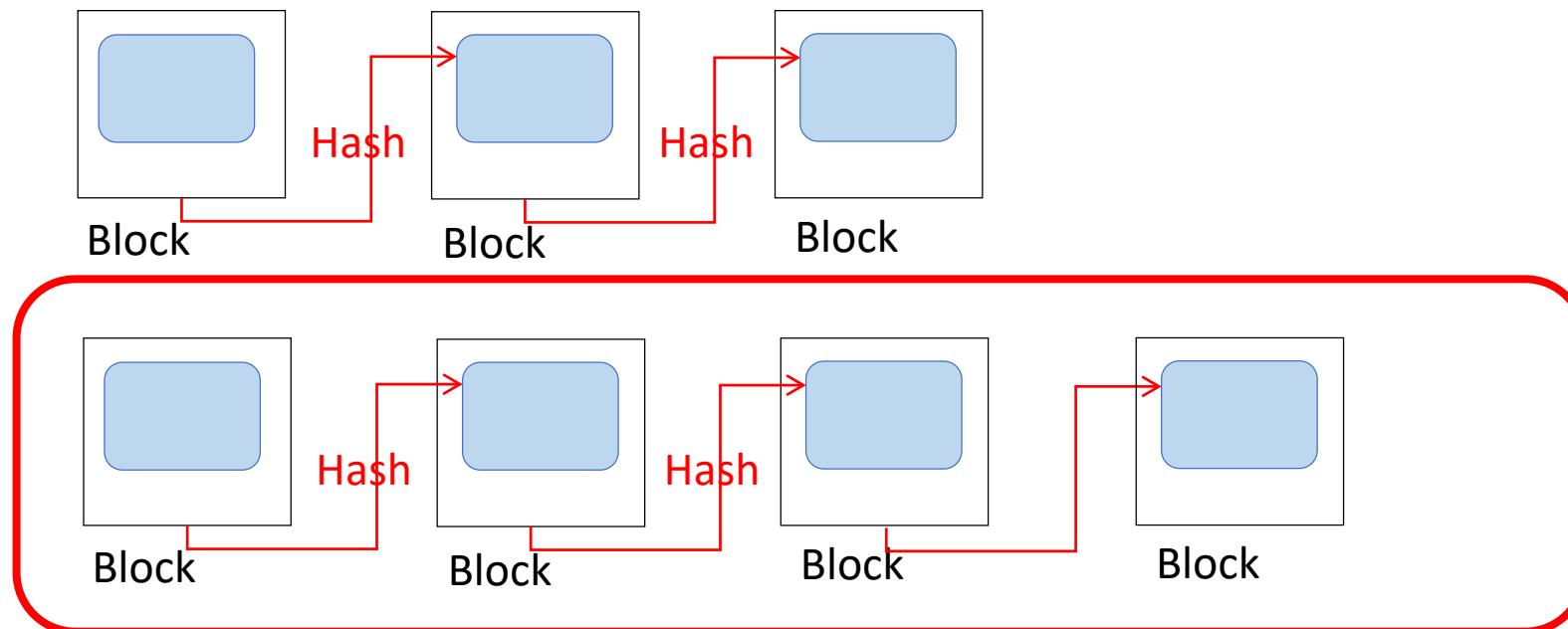
Users' Job (Summary)

- Just listening for broadcast blocks from miners and update their own copies of blockchain



An Important Blockchain Rule

- If there are two distinct blockchains with possibly conflicting transaction histories, always accept the longest one.



Fooling Blockchain (?)

- Scenario: Alice tries to fool Bob with a fraudulent block, which does not have a proof of work from the Miner.
 - Alice could get a proof of work herself.
 - But the problem is Alice needs to work out all the proofs of work after this (fraudulent) block.
 - Bob still receives another blocks from other Miners. If the resulting blockchain is longer, he should accept it by the rule.
 - Can Alice keep adding fraudulent blocks in the chain?
The answer is no. Unless she has close to 50% of the computing resources among all the Miners, the probability becomes overwhelming that the block chain that all of the other Miners are working on **grows faster** than the single fraudulent blockchain that Alice is feeding to Bob.

Cryptocurrency

- Now drop \$ sign from the ledger.
- We can replace it with any cryptocurrency.
- Cryptocurrency is being created as a reward for mining, but the reward amount will reduce half every four years. → The value of currency is maintained.
- Future of cryptocurrency? I don't know.

The Periodic Table of Cryptocurrencies

THE PERIODIC TABLE OF CRYPTOCURRENCIES

CREATED BY:
INVEST IN
BLOCKCHAIN

The Periodic Table of Cryptocurrencies																	
Payments/Currency		CREATED BY: INVEST IN BLOCKCHAIN														Privacy Coins	
Bitcoin BTC '09		Name TICKER		Year Founded		Blockchain Focus								Platforms		Monero XMR '14	
Litecoin LTC '11	Bitcoin Gold BTG '17	Computing, Data Management & Cloud Services								Ethereum ETH '14	Cardano ADA '16	EOS EOS '17	NEO NEO '14	NavCoin NAV '14	Zcash ZEC '15		
Bitcoin Cash BCH '17	Decred DCR '16	Protocols, Exchanges & Interoperability								Ethereum Classic ETC '16	Qum QTUM '16	Zilliqa ZIL '15	NEM NEM '15	Enigma ENG '17	Bytecoin BCN '12		
Nano NANO '14	Dogecoin DOGE '13	Binance Coin BNB '17	Kyber Network KNC '17	Os ZRX '16	Aion AION '17	ICON ICX '17	Golem GNT '16	Augur REP '15	Aragon ANT '17	Power Ledger POWR '16	Storm STORM '17	Steem STEEM '16	Lisk LSK '16	Rchain RHOC '17	Nxt NXT '13	PIVX PIVX '15	Verge XVG '12
Dash DASH '14	DigiByte DGB '13	Waves WAVES '16	Huobi Token HT '15	Bytom BTM '17	Hshare HSR '17	Wanchain WAN '17	SONM SNM '17	Stacoin SIA '15	DenoCoin DCN '17	Aeternity AE '17	Substratum SUB '17	Tron TRX '17	Basic Attention Token BAT '17	Elastos ELA '17	Skycoin SKY '16	Zcoin XZC '15	Bitcoin Diamond BCD '17
Gas (NEO) GAS '14	MonaCoin MONA '13	BitShares BTS '13	Loopring LRC '17	Bancor BNT '16	Ark ARK '17	Byteball Bytes GBYTE '16	Zeit ELF '17	Masternode Coin MAID '14	Iota IOTA '15	Cortex CTXC '17	Loom Network LOOM '17	Nebulus NAS '17	Status SNT '17	ReddCoin RDD '14	ZenCash ZEN '17	Bitcoin Private BTCP '17	
Electronium ETN '16	USD Tether USDT '15		KuCoin Shares KCS '13	Orbis GTO '17	Quant-Stamp QSP '17	Main XIN '17	iExec RLC '16	Starj STORJ '15	GXChain GXS '17	Holo HOT '17	WaykiChain WICC '17	Kin KIN '17	FunFair FUN '18	Wax WAX '18	Revert R '17	CloudCoin CLOAK '14	Komodo KMD '17
FinTech			Ripple XRP '12	Stellar XLM '14	OmniLedger OMG '17	Populous PPT '15	Polymath POLY '17	MakerDAO MKR '14	DigiDAO DGD '14	Request Network REQ '17	QASH QASH '17	Iconomi ICN '16	TenX PAY '15	Fusion FSN '17	Salt SALT '16	Ethos ETHOS '16	Monaco MCO '16
Business & Enterprise		VeChain Thor VET '17	Watson-chain WTC '16	Stratis STRAT '17	Ontology ONT '17	Ardor ARDR '16	iOSToken IOST '17	Dragonchain DRGN '17	Factom FCT '14	Centrality CENNZ '16	Ubiqui UBQ '14	Emercoin EMC '14	Nuls NULS '17	Neblio NEBL '14	Systain SYS ?	Future —	

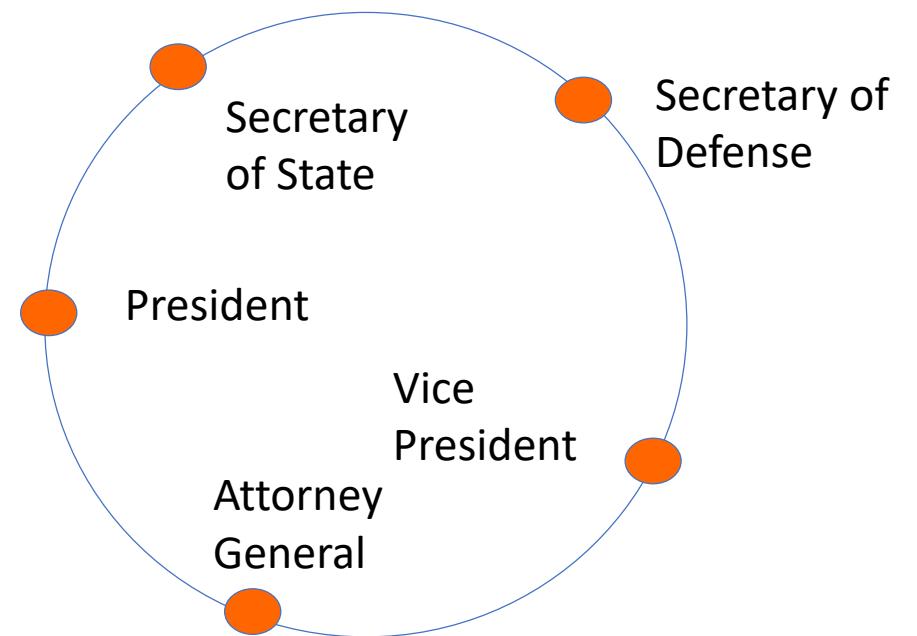
CSCI361

Ring Signatures: How to Leak a Secret

Problem

- How to generate an anonymous signature from a high-ranking White House official
 - We want to hide who signed the message
- Proposed by Rivest, Shamir and Tauman

Who signed?

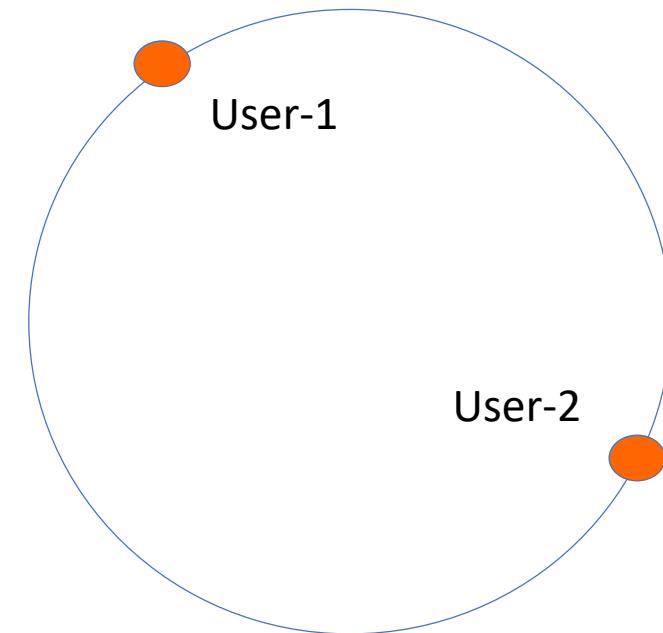


Ring Signature

- A type of digital signature that can be produced by any member of a group of users who have private keys
- A message signed with a ring signature can be verified by anyone
- Important features
 - It is impossible (computationally infeasible) to determine which of the group members' keys was used to produce the signature
 - There is no way to revoke the anonymity of a given ring signature
 - Any members in the ring can produce a ring signature without setup

Realisation of Ring Signature for Two Users

- Setup: User-1 with public key $P_1 = (e_1, N_1)$ and private key (d_1, N_1) ; User-2 with public key $P_2 = (e_2, N_2)$ and private key (d_2, N_2)
- They are using RSA for underlying signing algorithm
- H: A cryptographic hash function like SHA
- E: Symmetric encryption. E^{-1} : Symmetric decryption = D → E is called a Pseudo Random Function (PRF).



Realisation of Ring Signature for Two Users

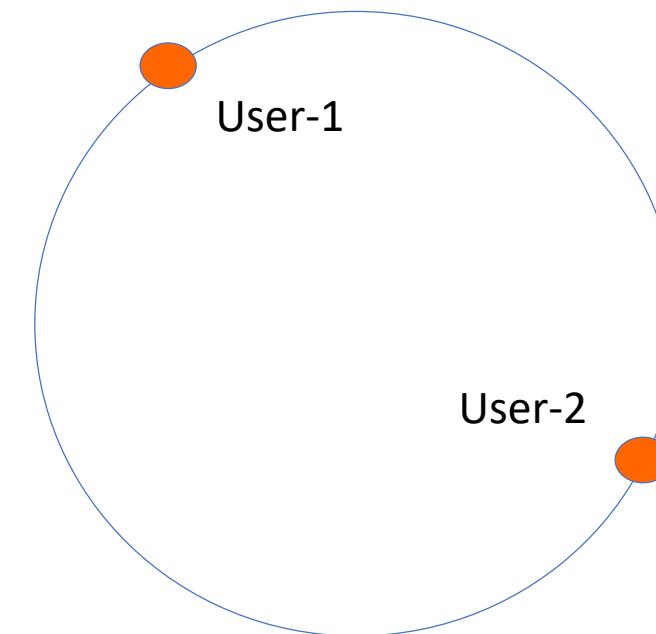
- Assume that User-2 is the signer
 - User-2 gets his message m and calculate the key $k = H(m)$.
 - Pick a random glue value v .
 - Pick a random x_1 for User-1 and calculate $y_1 = x_1^{e_1} \bmod N_1$.
 - Solve an equation for $E_k(y_2 \oplus E_k(y_1 \oplus v)) = v$, where E is symmetric encryption, to get y_2 : $y_2 = E^{-1}_k(v) \oplus E_k(y_1 \oplus v)$
 - Calculate $x_2 = y_2^{d_2} \bmod N_2$.
 - The ring signature is now (P_1, P_2, v, x_1, x_2) .

Realisation of Ring Signature for Two Users

- Signature Verification
 - Calculate $y_1 = x_1^{e_1} \bmod N_1$ and $y_2 = x_2^{e_2} \bmod N_2$.
 - Calculate $k = H(m)$.
 - Check whether $E_k(y_2 \oplus E_k(y_1 \oplus v)) = v$. If yes, the ring signature on m is valid, otherwise, it is invalid.

Example: Realisation of Ring Signature for Two Users

- Setup: User-1 with public key $P_1=(3,55)$ and private key $(27,55)$;
User-2 with public key $P_2=(5,65)$ and private key $(29,65)$
- They are using RSA for underlying signing algorithm
- H: SSHA (4 bit output)
- E: Symmetric encryption. E^{-1} :
Symmetric decryption = D



Example: Realisation of Ring Signature for Two Users

- Assume that User-2 is the signer
 - User-2 gets his message $m=5$ and calculate the key $k = 1101 = H(5)$.
 - Pick a random glue value $v=1010001$.
 - Pick a random $x_1 = 3 (=0000011)$ for User-1 and calculate $y_1 = x_1^{e_1} \bmod N_1 = 3^3 \bmod 55 = 27 = 0011011$.
 - Solve an equation for $E_k(y_2 \oplus E_k(y_1 \oplus v)) = v \rightarrow E_{1101}(y_2 \oplus E_{1101}(0011011 \oplus 1010001)) = 1010001$ to get y_2 :
 - ✓ $y_2 = E^{-1}_{1101}(1010001) \oplus E_{1101}(0011011 \oplus 1010001) = E^{-1}_{1101}(1010001) \oplus E_{1101}(1001010) = 1010111 \oplus 0100101 = 1110010 \bmod 65 = 114 \bmod 65 = 49 = 0110001$ (Assume that $E^{-1}_{1101}(1010001) = 1010111$ and $E_{1101}(1001010) = 0100101$)
 - ✓ After y_2 is calculated, we know that $E_{1101}(0110001 \oplus 0100101) = E_{1101}(0010100) = 1010001$

Example: Realisation of Ring Signature for Two Users

(continued)

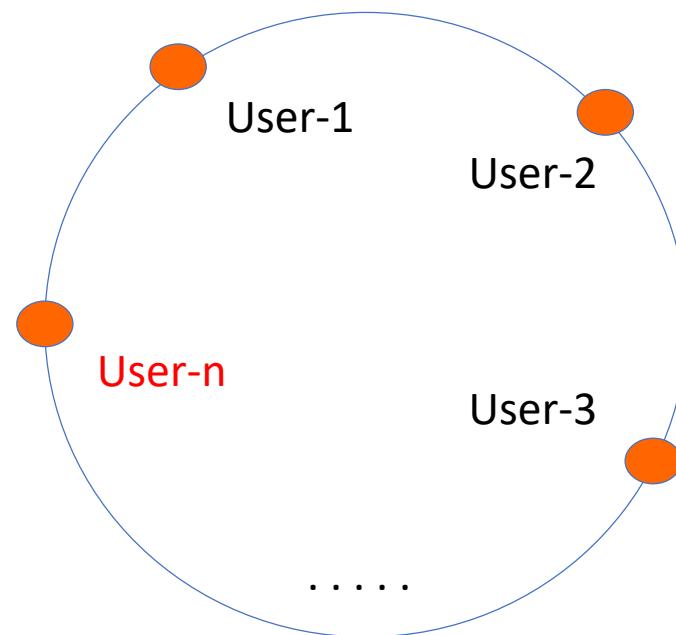
- Calculate $x_2 = y_2^{d_2} \bmod N_2 = 49^{29} \bmod 65 = 4 = 0000100$.
- The ring signature is now $(P_1, P_2, v, x_1, x_2) = ((3, 55)(5, 65), 1010001, 0000011, 0000100)$.

Realisation of Ring Signature for Two Users

- Signature Verification
 - Calculate $y_1 = x_1^{e_1} \bmod N_1 = 0000011^3 \bmod 55 = 3^3 \bmod 55 = 27$
 $(=0011011)$ and $y_2 = x_2^{e_2} \bmod N_2 = 0011101^5 \bmod 65 = 4^5 \bmod 65 = 49$ ($=0110001$).
 - Calculate $k = H(m) = H(5) = 1101$.
 - Check whether $E_k(y_2 \oplus E_k(y_1 \oplus v)) = v$.
 $\rightarrow E_{1101}(0110001 \oplus E_{1101}(0011011 \oplus 1010001)) \rightarrow E_{1101}(0110001 \oplus E_{1101}(1001010))$
 $\rightarrow E_{1101}(0110001 \oplus 0100101) \rightarrow E_{1101}(0010100) = 1010001$ (See page 8)

Realisation of Ring Signature for n Users

- Before we describe n user scheme, assume (without loss of generality) that among n users, “User-n” is always a name for the group member who generates a ring signature.
 - ✓ Of course anyone in the group can be “User-n”



Realisation of Ring Signature for n Users

- User-1 with public key $P_1 = (e_1, N_1)$ and private key (d_1, N_1) ;
User-2 with public key $P_2 = (e_2, N_2)$ and private key (d_2, N_2) ;...,
User-(n-1) with public key $P_{n-1} = (e_{n-1}, N_{n-1})$ and private key (d_{n-1}, N_{n-1})
 - User-n gets her message m and calculate the key $k = H(m)$.
 - Pick a random glue value v .
 - Pick a random x_1 for User-1, x_2 for User-2, ..., x_{n-1} for User-(n-1)
and calculate $y_1 = x_1^{e_1} \text{mod } N_1$, $y_2 = x_2^{e_2} \text{mod } N_1$, ..., $y_{n-1} = x_{n-1}^{e_{n-1}} \text{mod } N_{n-1}$.

Realisation of Ring Signature for n Users

- Solve an equation for $E_k(y_n \oplus E_k(y_{n-1} \oplus E_k(\dots \oplus E_k(y_1 \oplus v) \dots))) = v$, where E is symmetric encryption.
- Calculate $x_n = y_n^{d_n} \bmod N_n$.
- The ring signature is now $(P_1, P_2, \dots, P_n, v, x_1, x_2, \dots, x_n)$.

CSCI361

Cryptography and Secure
applications

Subject Revision

Main topics

- Security concepts
- Mathematics required
- Secret key cryptography
 - Classical cryptography
 - Block cipher
 - DES, AES
 - Modes
 - Message Authentication Code
- Public key cryptography
 - Public key encryption
 - Knapsack
 - RSA, Rabin
 - ElGamal
 - Digital Signature
 - ElGamal
 - DSS
- Hash function
- Key management
- Secret sharing
- Applications

Security concepts

- What, why, and who
- Assets & threats
- Security goals
- Security principles

Classical cryptography

- Cryptography vs Cryptoanalysis
- Kerkchoff's Law
- Possible attacks
- Caesar cipher
- Monoalphabetic ciphers
 - Additive, multiplicative, affine
 - Key phrase
- Statistical analysis
- Polyalphabetic ciphers
 - Vigenère cipher
- The Kasiski method
- The index of coincidence

Secret key cryptography

- One-time pad
 - Perfect secrecy
- Unicity distance
- Confusion and Diffusion
- DES design
 - Iterated cipher
 - Block and key size
 - The Feistel structure
 - Involution functions
 - Encryption and Decryption
 - S-box

Secret key cryptography

- DES weakness
 - Complement property
 - Weak/semi-weak keys
 - Brute-force attack
- Multiple encryption
 - Meet-in-the-middle attack
 - Triple DES

Secret key cryptography

■ AES

- Block and key size
- No. of rounds
- Round transformation
 - ByteSub
 - ShiftRow
 - MixColumn
 - AddRoundKey

■ Stream cipher

- RC4

Mode of operation

- Electronic codebook
- Cipher block chaining
- Cipher feedback
- Output feedback
- Counter
- Advantage/Disadvantage

Message authentication code

- Error detection vs MAC
- Unconditionally secure MAC
- Block cipher based MAC

Public key cryptography

- Advantages
- One-way trapdoor function
- Knapsack cryptosystem
 - Super-increasing knapsack
 - Trapdoor knapsack
- RSA
 - Key generation
 - Primality test
 - Encryption
 - Decryption
 - Correctness
 - Security assumption
 - Fast exponentiation
 - Common modulus attack

Public key cryptography

- Rabin & ElGamal
 - Key generation
 - Encryption
 - Decryption
 - Security assumption

Digital Signature

- Digital signature using PKC
 - RSA signature
- ElGamal signature scheme
- Digital Signature Standard
- Specially designed signatures:
 - Blind Signatures
 - Designated Verifier Signature, Undeniable Signatures
 - Group Signature, Ring Signature

Hash Functions

- Collision resistance
- Hash-then-sign
- Birthday attack
- Design
 - Block cipher (CBC)
 - Modular Arithmetic
 - SHA-1

Key Management

- Diffie-Hellman key exchange
- Man-in-the-middle attack
- Key distribution in PKC
- Digital certificates

Secret Sharing

- Threshold secret sharing
- Shamir's secret sharing scheme
- Homomorphic property

Application

- Zero Knowledge Proof
- Fair Exchange
- Other types of signatures: Ring signatures
- Blockchain

Test

- Approximately 2 hours
- During tutorial time
- Calculator is permitted (non programmable)

Exam

- Duration: 3hours
- Non programmable calculator is permitted
- Total Marks: 55
 - You must obtain at least 22/55, otherwise you will receive TF

Question type

- Concepts (short answer)
 - E.g. Security goals, Kerkchoff's Law, Perfect secrecy, One way functions, etc.
- Fundamentals
 - E.g. GCD, generator, fast exp, etc.
- Schemes
 - Basic functions: e.g. Additive cipher, S-box, Modes of operation, RSA Encryption & Signature, etc.
 - Cryptanalysis: variants of known schemes