STREAM & BLOCK CIPHERS: AN INTRODUCTION

Applied Cryptography

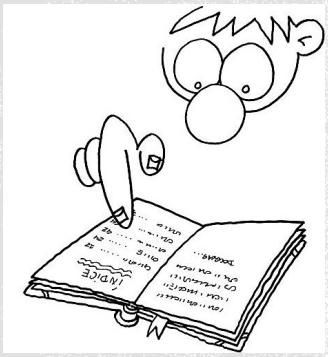
Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

CONTENTS

- Digression on using bits and bytes
- Vernam in a nutshell
- From Vernam Cipher to Stream & Block ciphers
- Introduction to stream & block ciphers
  - Stream ciphers
    - Digression on attacking Vernam cipher or the importance of being able to generate "long" key streams
  - Block ciphers
  - Stream vs block ciphers



2

DIGRESSION ON USING BITS & BYTES

WHY BITS & BYTES?

- Work with an alphabet of *bytes* rather than (English, lowercase) *letters*
  - Works natively for arbitrary data!
- It is possible to represent data as sequences / blocks of bits by using appropriate encodings
- For instance, it is possible to use hexadecimal representation to compactly represent groups of 4 bits and thus 2 hexadecimals yields a byte

Example: **0x**AF
0xAF = 16*A + F = 16*10 + 15 = 175
0xAF = 1010 1111

Hex	Bits	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Hex	Bits	Decimal
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

ASCII CHARACTERS

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Encoding characters as bytes

Examples
- '1' = 0x31 = 0011 0001
- 'F' = 0x46 = 0100 0110

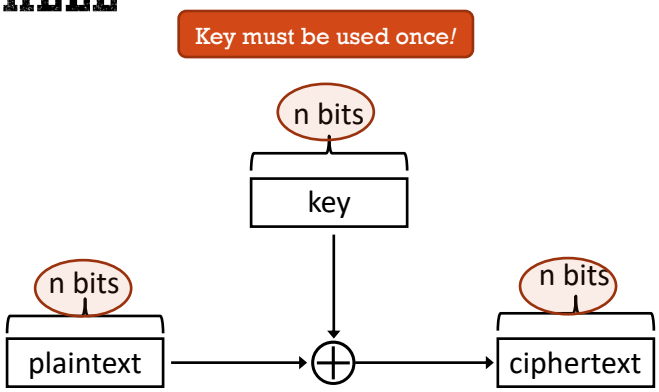
5

END OF DIGRESSION ON USING BITS & BYTES

VERNAM IN A NUTSHELL

- Let the set of plaintexts $\mathcal{P} = \{0,1\}^n$
- Choose uniformly a key $k \in \mathcal{K} = \{0,1\}^n$
- $Enc_k(m) = k \oplus m$
- $Dec_k(c) = k \oplus c$

- Correctness:
  $Dec_k(\ Enc_k(m)\ ) = k \oplus (k \oplus m)$
  $= (k \oplus k) \oplus m = m$



7

FROM VERNAM TO STREAM & BLOCK CIPHERS

Symmetric cryptography

DRAWBACKS OF VERNAM

- Must generate a truly random key sequence as long as the message
- Find a secure channel for transportation of the key to the message recipient
- Do this for every single message to be exchanged

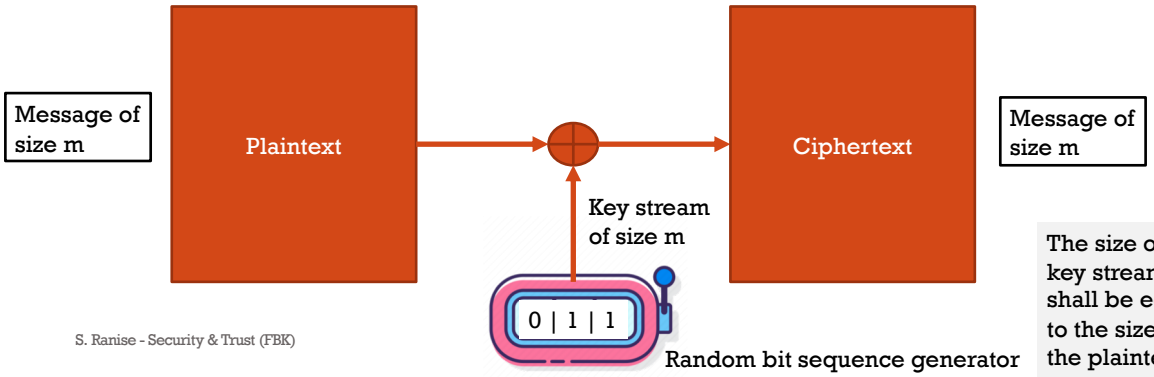
- In summary, the problem is to securely transfer large quantities of secure keys…

IMPROVING THE VERNAM CIPHER

- Two different approaches
  - **Stream** ciphers
    - create very long keys (ideally of infinite length) to be xor-ed with plaintexts
  - **Block** ciphers
    - use the same key more than once in a way that does not compromise the cipher

IMPROVING THE VERNAM CIPHER (1)

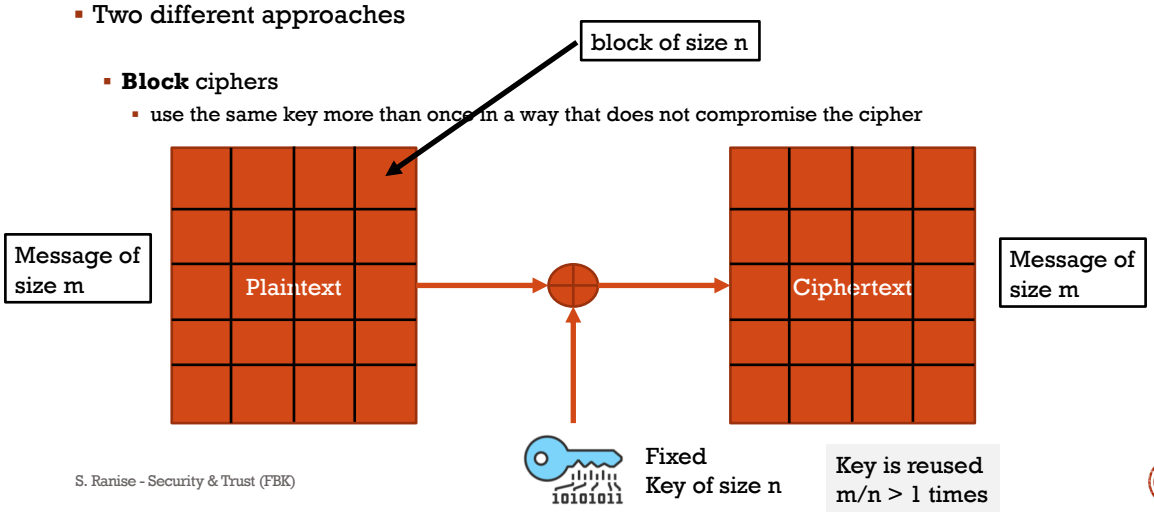
- Two different approaches
  - **Stream** ciphers
    - create very long keys (ideally of infinite length) to be xor-ed with plaintexts



The size of the key stream shall be equal to the size of the plaintext

IMPROVING THE VERNAM CIPHER (2)

- Two different approaches
  - **Block** ciphers
    - use the same key more than once in a way that does not compromise the cipher



Key is reused m/n > 1 times

IMPROVING THE VERNAM CIPHER (3)

- Two different approaches
  - **Stream** ciphers
    - create very long keys (ideally of infinite length) to be xor-ed with plaintexts
  - **Block** ciphers
    - use the same key more than once in a way that does not compromise the cipher

The size of the key stream shall be equal to the size of the plaintext

Key is reused m/n > 1 times

Recall that

φ is a perfect cipher if and only if both the following conditions hold

1. the **keys** are **perfectly random**
2. for any pair *(m,c)* of plaintext-ciphertexts, there is **one and only one key *k*** such that $c = \varphi(m,k)$

IMPROVING THE VERNAM CIPHER (4)

- Big questions to improve Vernam cipher

1. For stream ciphers:
   - **How can we generate perfectly random keys of arbitrary size?**
   - **Additionally, how can we replicate the random streams of bytes for decryption?**
2. For block ciphers:
   - **How can we reuse multiple times the same key without enabling an attacker to perform cipher-text only attacks?**
   - **Additionally, how can we avoid attackers to exploit the block structure?**

14

STREAM & BLOCK CIPHERS: AN INTRODUCTION

15

STREAM CIPHERS

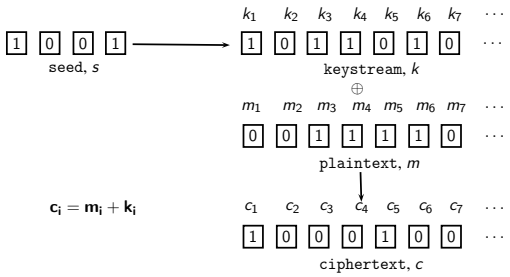
OVERVIEW

An abstract view of symmetric ciphers

Perform a series of operations on the bits

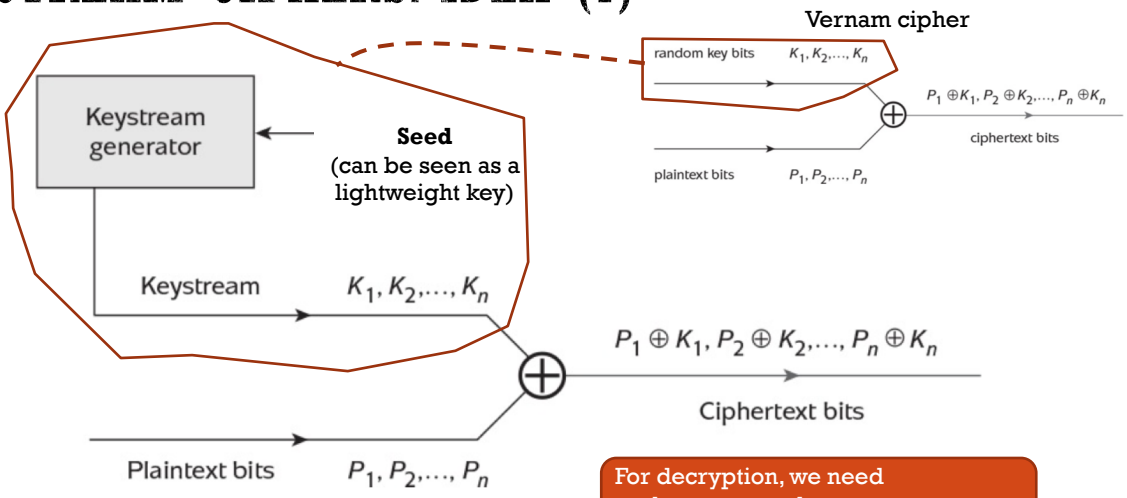
- Classification of symmetric algorithms
  - *Stream ciphers*, if the plaintext is **processed one bit at a time**
    - The algorithm selects one bit of plaintext, performs a series of operations on it, and then outputs one bit of ciphertext
  - *Block ciphers*, if the plaintext is **processed in blocks** (groups) **of bits at a time**
    - The algorithm selects a block of plaintext bits (typically 64 bits), performs a series of operations on them, and then outputs a block of ciphertext bits
- Stream vs block ciphers
  - Stream ciphers could be regarded as block ciphers with a block size of one
  - Some stream ciphers process data in bytes, and hence could be regarded as block ciphers with a block size of 8
  - **Rule of thumb** → when block size is << 64 (e.g., 1 or 8) we talk about stream ciphers otherwise we talk about block ciphers

STREAM CIPHERS: IDEA (0)



- Take a **seed**, i.e. a small vector of a few **random** bits, to be kept secret
- Construct a **keystream**, i.e. a **very long sequence of pseudorandom bits**, …
- … to be xor-ed (bitwise) with the plaintext bitwise in order to produce the ciphertext
- The idea is to generalize the approach of the Vernam cipher

STREAM CIPHERS: IDEA (1)



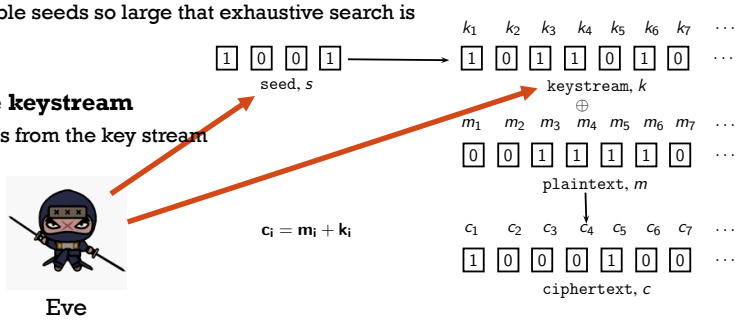
For decryption, we need
- the same seed
- the same keystream generator

STREAM CIPHERS: IDEA (2)

- The **real work** in **designing a good stream cipher** goes into **designing the keystream generator**
- Keystream generators produce output which appears to be randomly generated, but is actually not randomly generated
  - We talk about **pseudorandom generators**
- The Vernam Cipher can be regarded as a stream cipher whose keystream is truly randomly generated
- In many cases, stream ciphers combine the keystream with the plaintext in more complex ways than a simple bitwise xor operation

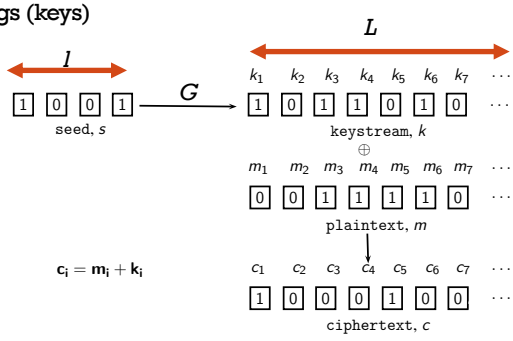
STREAM CIPHERS: IDEA (3)

- For a stream cipher to be good, Eve **should not be able** to
  - **recover** the **seed**
    - Make the set of possible seeds so large that exhaustive search is very hard in practice
  - **predict** the rest of the **keystream**
    - Eliminate any patterns from the key stream



STREAM CIPHERS: MORE PRECISELY (3)

- Set up
  - Use a short *l*-bit "seed" *s* as the encryption key where *l* is much smaller than the size *L* of the plaintext to be encrypted and then…
  - … "stretch" the seed *s* into a longer *L*-bit string (the key) that is used to mask the message and decrypt the ciphertext
  - The seed *s* is stretched using some efficient, deterministic algorithm *G* that maps *l*-bit strings (seeds) to *L*-bit strings (keys)
- **Encryption**: *G(s)* + *m*
  for any seed s (of size *l*) and plaintext m (of size *L*)
- **Decryption**: *G(s)* + *c*
  for any seed s (of size *l*) and ciphertext c (of size *L*)
  where *G* is called a *pseudo-random generator*



STREAM CIPHERS: MORE PRECISELY (4)

- If $l < L$, then by Shannon's Theorem, stream ciphers **cannot** be perfect
- However, if G has certain properties, then stream ciphers are secure in practice:
  - Suppose s is a random l-bit string and r is a random L-bit string
  - If **Eve cannot effectively tell the difference between G(s) and r**, then it should **not be able to tell the difference between stream ciphers and one-time pad**
  - **Since the one-time pad latter cipher is secure, so should be the stream cipher**

ON TELLING THE DIFFERENCE…

- **Statistical test**
  - Algorithm used to distinguish a pseudo-random string $G(s)$ from a truly random string r
  - It takes a string as input, and outputs 0 or 1
  - Test is effective if the probability that it outputs 1 on a pseudo-random input is significantly different than the probability that it outputs 1 on a truly random input
    - Even a relatively small difference in probabilities (e.g., 1%) is considered significant (why?)
    - A non-zero but negligible difference in probabilities (e.g., 2^−100) is not helpful
- Example
  - count the number k of 1's appearing in the string
  - For a truly random string, we would expect $k \approx L/2$
  - If G had some bias towards either 0-bits or 1-bits, we could effectively detect this by checking that it outputs 1 if $|k - 0.5L| < 0.01L$ and 0 otherwise

STREAM CIPHERS: MAIN ISSUES (6)

- A stream cipher is well equipped to encrypt a **single message** from Alice to Bob
- Already with two messages, there may be problems
- Assume Alice and Bob want to exchange messages *m1* and *m2*
- Let *c1* = *m1* + *G(s)* and *c2* = *m2* + *G(s)*
- Eve intercepts both *c1* and *c2* and calculates the xor of the ciphertexts

  $c1 + c2 = (m1 + G(s)) + (m2 + G(s)) = (m1 + m2) + (G(s) + G(s)) = m1 + m2$
- English text contains enough redundancy that given *m1* + *m2,* Eve can recover both *m1* and *m2* in the clear by using frequency analysis
- **A stream cipher key should never be used to encrypt more than one message**

STREAM CIPHERS: MAIN ISSUES (7)

- What about integrity*?*
- Let *m* be a plaintext and *s* the seed
- Eve changes a ciphertext *c* = *m* + *G(s)* by xor-ing a certain message *d*(elta), i.e. it intercepts *c* and then forwards *c'* = *c* + *d*
- The receiver will get the following plaintext

  $m' = c' + G(s) = (c + d) + G(s) = (c + G(s)) + d = m + d$
- Knowing neither *m* nor *s*, Eve was able to cause the decrypted message to become *m* + *d* for *d* of its choice
- Stream-ciphers are said to be **malleable** since an attacker can cause predictable changes to the plaintext

KEY MANAGEMENT

- **Length of the key**
  - In a one-time pad, the key has to be as long as the plaintext
  - Although the keystream for a stream cipher needs to be as long as the plaintext, the seed used to generate it (and which must be distributed and stored) is much shorter
- **Random generation of the key**
  - In a one-time pad, the key has to be truly randomly generated, which involves costly generation techniques
  - The keystream in a stream cipher is pseudorandom and thus is much cheaper to generate
- **One-time use**
  - A keystream generator is a deterministic process since every time the same seed is input into the keystream generator, it will result in the same keystream being output
  - If we reuse a seed to produce the same keystream and then encrypt two plaintexts using the same portion of the keystream, then, just as in a one-time pad, the xor between the two ciphertexts will tell us the difference between the two corresponding plaintexts
  - We can avoid this problem by, e.g., making the **keystream dependent** on **time varying data** or **generating the ciphertext with more complex operations than a xor**…

DIGRESSION ON ERROR PROPAGATION

- Types of errors in communication systems
  - *Transmission errors* are errors occurring in the communication channel
  - *Transmission losses* occur when bits get lost in the communication channel
  - *Computational errors* are errors occurring somewhere during a (cryptographic) computation
  - A 1-bit transmission error/loss or computational error occurs if 0 becomes a 1 or a 1 becomes a 0 or it is lost somewhere on the communication channel or during computation
- Transmission errors or losses are very frequent over noisy or unreliable channels
- *Error propagation* occurs when a number of errors in the ciphertext (regardless of error type) result in a greater number of errors in the resulting plaintext
  - In the simplest case of a 1-bit error in the ciphertext, error propagation occurs if this has an impact of more than one erroneous bit in the resulting plaintext

PROS & CONS OF STREAM CIPHERS

- They do not give rise to error propagation as each bit in the ciphertext depends on just one bit in the plaintext
  - 1 bit transmission errors will result in 1 bit error in the plaintext → this makes stream ciphers very attractive for communication services (e.g., **mobile communication**) and applications with poor communication channels
- They are **very fast** making them ideal for real time applications (e.g., **mobile communication** services)
- They are easy to implement (in hardware) and they do not require large memory capabilities
- Since stream ciphers process data bitwise, it is crucial that **sender and receiver keep their keystreams in perfect synchronisation**
  - 1 bit data loss may have catastrophic consequences as decryptions are performed on the wrong bits after the receiver is out of synch of the sender
  - resynchronization mechanisms must be put in place to avoid these problems

29

DIGRESSION ON ATTACKING THE VERNAM CIPHER

VIGENÈRE CIPHER (1)

- It can be seen as a variant of Vernam cipher whereby the key is a sequence of bits of fixed length
- For concreteness, consider the problem of encrypting a plaintext in English whose letters are encoded as ASCII codes
- Then, the Vigenère cipher can be described as follows
  - The key is a string of bytes
  - The plaintext is a string of bytes
  - To encrypt, XOR each character in the plaintext with the next character of the key
    - Wrap around in the key as needed

VIGENÈRE CIPHER (2)

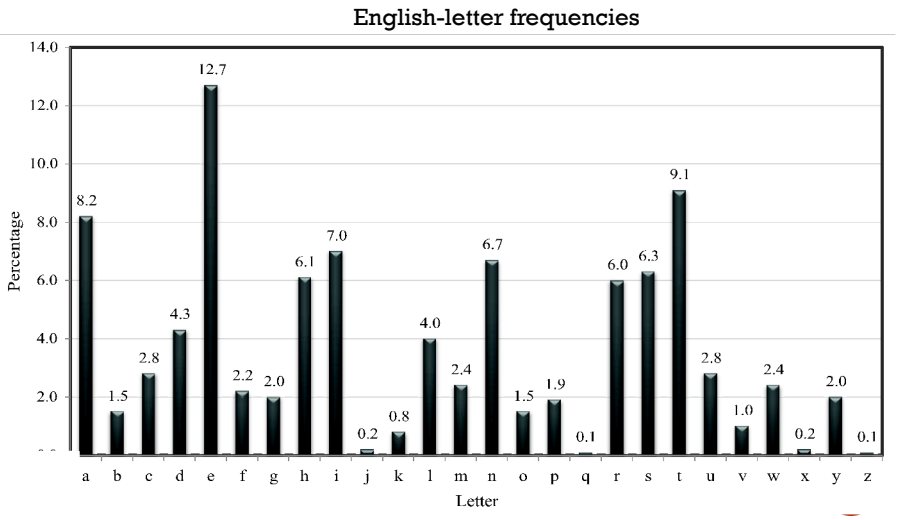
- Let plaintext be "Hello!" and key be "¡/"
- Thus
  - "Hello!" = 0x48 65 6C 6C 6F 21
  - "¡/" = 0xA1 2F
- Compute:

  0x48 65 6C 6C 6F 21 XOR 0xA1 2F A1 2F A1 2F
- So: 0x48 ⊕ 0xA1
  - 0100 1000 ⊕ 1010 0001 = 1110 1001 = 0xE9
  - …
- **Ciphertext**: "éJÍCÎ" = 0xE9 4A CD 43 CE 0E

ATTACKING VIGENÈRE CIPHER (1)

- Two steps:
  - Determine the key length
  - Determine each byte of the key

by using **frequency analysis**



ATTACKING VIGENÈRE CIPHER (2)

- **Determining key length**
- Let p_i (for $0 \leq i \leq 255$) be the frequency of **byte** i in general English text
  - I.e., $p_i = 0$ for $i < 32$ or $i > 127$
  - I.e., p_{97} = frequency of 'a'
  - The distribution is far from uniform
- **If the key length is N**, then every N^{th} character of the plaintext is encrypted using the same byte
  - If we take every N^{th} character and calculate frequencies, we should get the p_i's in permuted order
  - If we take every M^{th} character (M not a multiple of N) and calculate frequencies, we should get something close to uniform

ATTACKING VIGENÈRE CIPHER (3)

Indeed, the larger the key size, the longer it takes to perform both steps of the attack!

- **Determine each byte of the key**…
- Consider the i-th byte
- Look at every N^{th} character of the ciphertext, starting with the i^{th} character
  - Call this the i^{th} ciphertext "stream"
  - Note that all bytes in this stream were generated by XORing plaintext with the same byte of the key
- Try decrypting the stream using every possible **byte value B**
  - Get a candidate plaintext stream for each value
- When the guess B is correct:
  - All bytes in the plaintext stream will be between 32 and 127
  - Frequencies of lowercase letters should be close to known English-letter frequencies

35

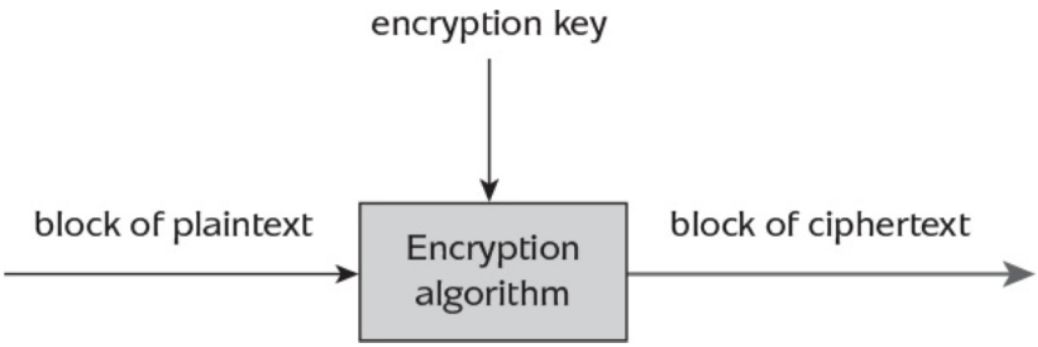
END OF DIGRESSION ON ATTACKING THE VERNAM CIPHER

36

BLOCK CIPHERS

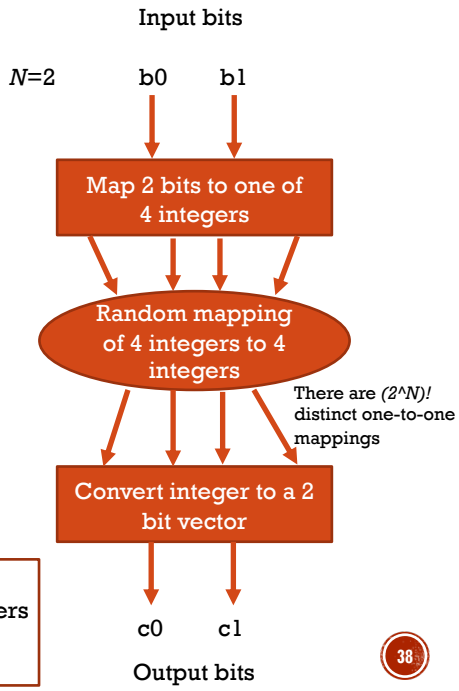
OVERVIEW

- typical block size = 64 or 128 bits
- while block size is fixed, size of encryption key may vary



BLOCK CIPHERS (1)

- Main idea
  - Replace a block of *N* bits from the plaintext with a block of *N* bits from the ciphertext
  - The relationship between the input blocks and the output block is completely random
  - It must be invertible for decryption to work
  - Thus, it has to be one-to-one, i.e. each input block is mapped to a unique output block
  - Usually, *N=64, 128, 256*



- The first number among the *2^N* can map to any of the *2^N* numbers
- The second number can map to any of the remaining *(2^N) - 1* numbers
- … and so on
- The total number of one-to-one functions is *(2^N)!*

38

DIGRESSION ON THE BLOCK SIZE

- If the block size is too small, then the number of different plaintext blocks that can ever be encrypted may be too small for an attacker to launch a type of dictionary attack by building up a dictionary of plaintext/ciphertext pairs
  - A larger block size makes this attack harder because the dictionary needs to be larger
  - Past block ciphers has a block size of 64 bits nowadays the block size is 128 bits
- If the block size is too large, then the block cipher becomes inefficient to operate, particularly for **plaintexts smaller than the block size** as they need **padding**
- The block size should be a multiple of 8, 16, 32, or 64 as these are the sizes of words supported on currently available computers

39

BLOCK CIPHERS (2)

- There are *2^N* different possible *N*-bit patterns
- Each pattern can be represented by an integer between *0* and *(2^N)-1*
  - The bit pattern with *N 0*s could be represented by the integer *0*
  - The bit pattern with *N 1*s would be represented by the integer *(2^N)-1*
- The mapping from the input bit blocks to the output bit blocks can also be defined as a mapping from the integers corresponding to the input bit blocks to the integers corresponding to the output bit blocks

BLOCK CIPHERS (3)

- The **encryption key** for the ideal block cipher is the table (also called the **codebook**) that shows the relationship between the input and the output blocks
- Think of each possible input block as one of *2^N* integers and for each such integer we can specify an output *N*-bit block
- It is possible to define the codebook by displaying just the output blocks in the order of the integers corresponding to the input blocks
- The **codebook** will be of **size** *N*(2^N)*
  - If *N=64* then *N*(2^N)* is around *10^(21)*
- This means that the **encryption key** for the ideal block cipher using *N*-bit blocks will be of size *N*(2^N)*
- The size of the encryption key would make the ideal block cipher an impractical idea
  - Think of the issues related to the transmission, distribution, and storage of such large keys

Encryption key (codebook)

Input	Output
0	?
1	?
…	…
(2^N)-1	?

2^N lines

N bits

Can be suppressed by assuming standard ordering

BLOCK CIPHERS (4)

- A *perfect* block cipher is a **random permutation**

- It can be described as follows
  - **Encryption**
    - Daemon checks in the left hand column to see if it has a record of plaintext
    - If not, it asks the random number generator to generate a ciphertext that does not yet appear in the right hand column, and then writes down the plaintext/ciphertext pair in the codebook and returns the ciphertext
    - If it does find a record, it returns the corresponding ciphertext from the right hand column
  - **Decryption**
    - Same procedure as above with columns swapped

Encryption key (codebook)

Plain text	Cipher text
0	?
1	?
…	…
(2^N)-1	?

Daemon 

Random Number Generator 

BLOCK CIPHERS (5)

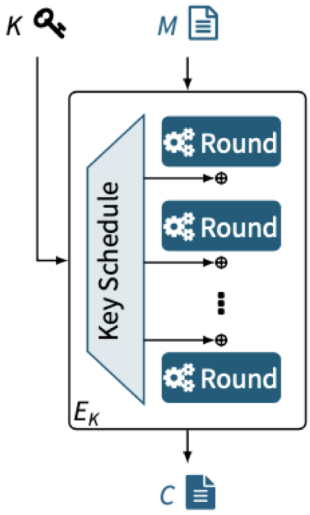
The size of the key is less than the number of *N* bits:
- DES
  N = 64 and *K = 56*
- AES
  N = 128 and *K = 128, 192, 256*

- A block cipher is a **keyed family of pseudorandom permutations**
- For each key, we have a single permutation that is independent of all the others
- Think of each key as corresponding to a different codebook
  - Strategy: choose *2^K* permutations uniformly at random from the set of all *(2^N)!* permutations
- The intuitive idea is that an implementation of the cipher should output
  - the ciphertext given the plaintext and the key
  - the plaintext given the ciphertext and the key

but given only the plaintext and the ciphertext it should output nothing

BLOCK CIPHERS (6)

- For a block cipher to be good, Eve should not be able to
  - recover the key **even using multiple plaintext-ciphertext pairs**
- With the Vernam cipher this is impossible to achieve…
- … the idea here is to build a cipher which is **much more complex** by re-using the idea underlying Vernam cipher
- Complexity implies that block ciphers are slower than stream ciphers but are generally considered more secure than the latter



PROS & CONS OF BLOCK CIPHERS

- They are used as basic building blocks of other cryptographic primitives including stream ciphers
- They are widely adopted across several different applications
  - Today, AES is the reference choice for block ciphers
- They can be used in various modes of operation to satisfy additional properties
- They give rise to error propagation
  - Despite a 1 bit transmission error will change just 1 bit in a block, after decryption this 1 bit error will give rise to errors in around half of the bits in the plaintexts. This is because of a crucial property of block ciphers, namely
    - **diffusion** → 2 blocks differing in a single bit shall generate 2 very different ciphertexts
- They need padding to align the length of the plaintext with that of the block size
  - **Example** a 400 bits plaintext can be accommodated in 4 blocks of size 128 bits, while the first 3 blocks can be filled, the remaining 16 bits should be padded with 112 bits, redundant information, to fill the 4^{th} block
  - Padding introduces an overhead in communication in case of small messages and may introduce security issues

46

STREAM VS BLOCK CIPHERS

SOME EXAMPLES OF SYMMETRIC CIPHERS

Stream	Block
Fast	Slow
Hardware	Software
Less secure	More secure

Type	Cipher	Applications
Stream	A5/1, A5/2, A5/3	Phone (GSM)
	RC4	Internet
	E0	Bluetooth
Block	DES	(old)
	3DES	Smart cards
	AES	Everywhere
	PRESENT	sensor networks

47

EXAMPLES OF STREAM CIPHERS

- **RC4**
  - Simple and fast stream cipher with a relatively low level of security
  - Probably the most widely implemented stream cipher in software and widely used in SSL/TLS, WEP, and Microsoft Office
- **A5/1**
  - One of the stream cipher algorithms used in GSM to secure the communication channel over the air from a mobile phone to the nearest base station
- **E0**
  - The stream cipher algorithm used to encrypt Bluetooth communications
- Notice that both RC4 and A5/1 were both originally proprietary designs which have become publicly known over time

EXAMPLES OF BLOCK CIPHERS

- **DES**
  - The default block cipher of the 1980s and 1990s, but now considered broken due primarily to its small key size
  - The two variants based on repeated DES applications commonly known as *3DES* are still respected block ciphers, although there are now faster block ciphers available
- **AES**
  - A default block cipher based on the encryption algorithm *Rijndael* which won the AES design competition by NIST to identity the successor of DES
- **Serpent**
  - A respected block cipher with a block size of 128 bits and key lengths of 128, 192, or 256, which was a finalist with AES in the NIST competition
  - Considered slower but somehow more secure than AES but not as widely adopted

COMMON ASSUMPTION ON STREAM CIPHERS

- Alice and Bob agreed on a secret key before any ciphertext is transmitted

Alice and Bob access the same shared key:
Symmetric Cryptography