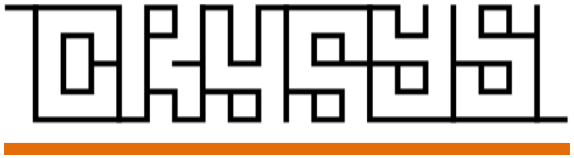


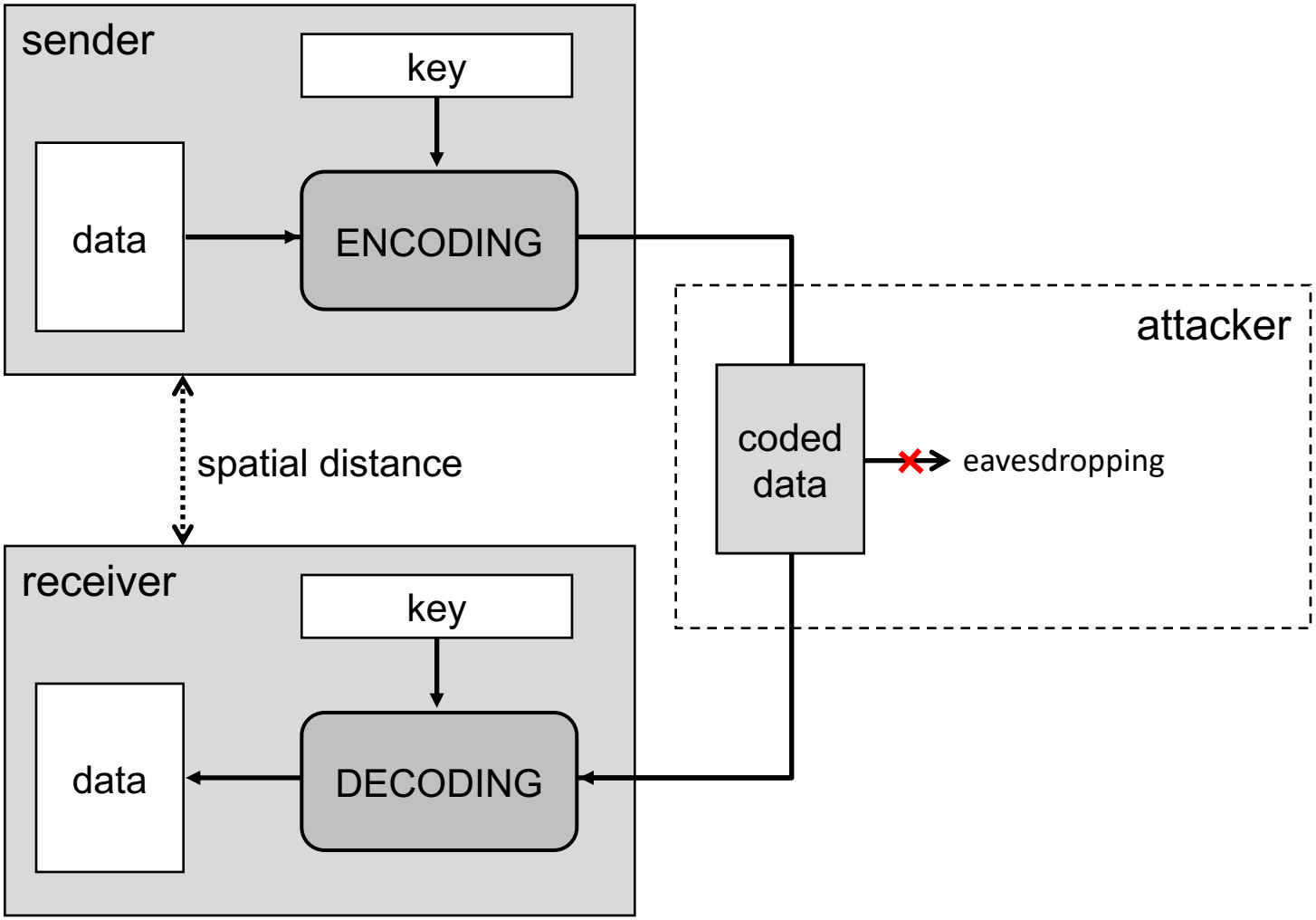
Symmetric Key Encryption
Stream Ciphers

Levente Buttyán

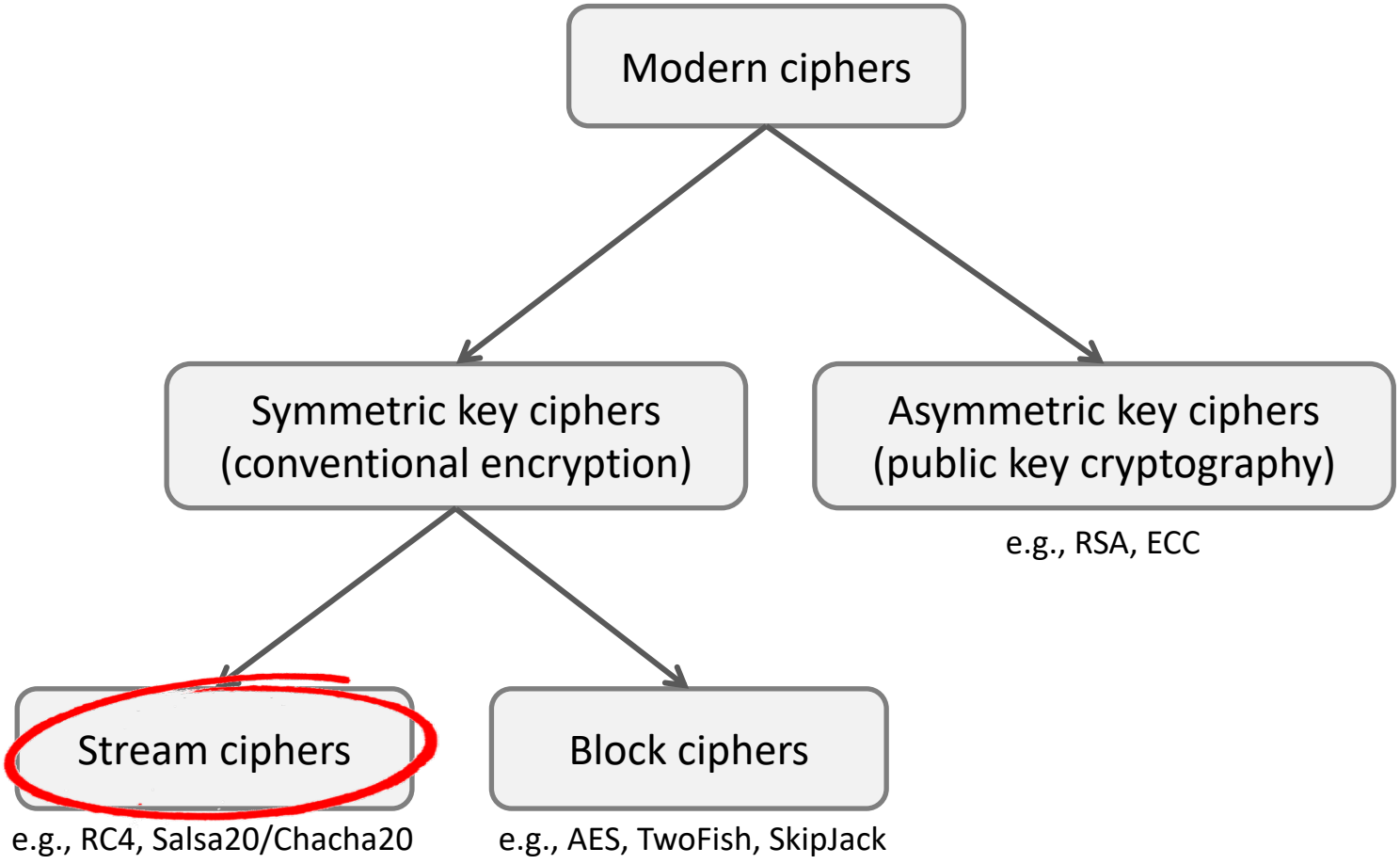
CrySyS Lab, BME

buttyan@crysys.hu

Basic model of symmetric key encryption



Classification of ciphers



XOR operation

- XOR (+ or $\oplus$)
  - 0+0 = 0; 0+1 = 1; 1+0 = 1; 1+1 = 0
- XOR of bit vectors (words)
  - we XOR each corresponding bit pairs
  - e.g., 0011 + 1010 = 1001
- exercise:
  - 1101 + 1001 = ____ ?
  - 1010 + 0001 = ____ ?
  - 1101 + 1101 = ____ ?
  - 1010 + 1111 = ____ ?

Main properties to remember

1. $X + 0 = 0 + X = X$
2. $X + X = 0$
3. if $A + B = C$, then $A = B + C$ (and $B = A + C$)

Hexadecimal numbers

- hexadecimal digits
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- from binary to hexadecimal (hex)
  - any 4-bit value (nibble) can be represented as a hex digit
    - » e.g., b1011 = 1*8 + 0*4 + 1*2 + 1*1 = 11 = xB
    - » e.g., b1001 = 1*8 + 0*4 + 0*2 + 1*1 = 9 = x9
    - » e.g., b1111 = xF
    - » e.g., b111 = b0111 = 7 = x7
  - any byte value can be represented as a two digit hex number
    - » e.g., b10111001 = xB9 (because b1011 = xB and b1001 = x9)
  - longer bit vectors can be represented as multi-digit hex numbers
    - » e.g., b1011100111110111 = b1011 1001 1111 0111 = xB9F7
- from hex to binary and to decimal
    - » e.g., x9F = b10011111 (because x9 = 9 = b1001 and xF = 15 = b1111)
    - » e.g., x9F = 9*16 + 15*1 = 159

XOR-ing hex numbers

1. convert them to binary
2. XOR the binary values
3. convert back the binary result to hex

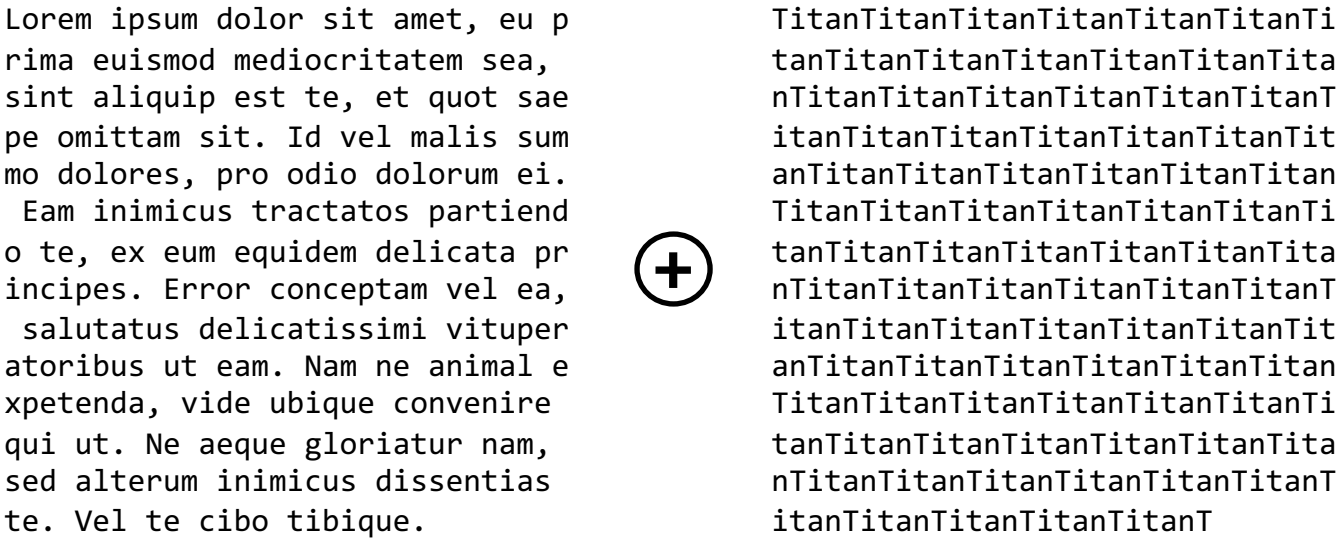
   - examples:
     - xB + x9 = b1011 + b1001 = b0010 = x2
     - xC + xC = ____ ?
     - xDEAD + xBEEF = ____ ?

ASCII code of characters

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Simple XOR cipher

- encryption
  - represent the plaintext as a sequence of bytes
  - take a password and repeat it many times to get a byte string as long as the plaintext
  - obtain the ciphertext by XOR-ing together the plaintext and the password string



- decryption
  - XOR the same password string to the ciphertext to recover the plaintext

Breaking the simple XOR cipher

sometimes, it can be trivial...

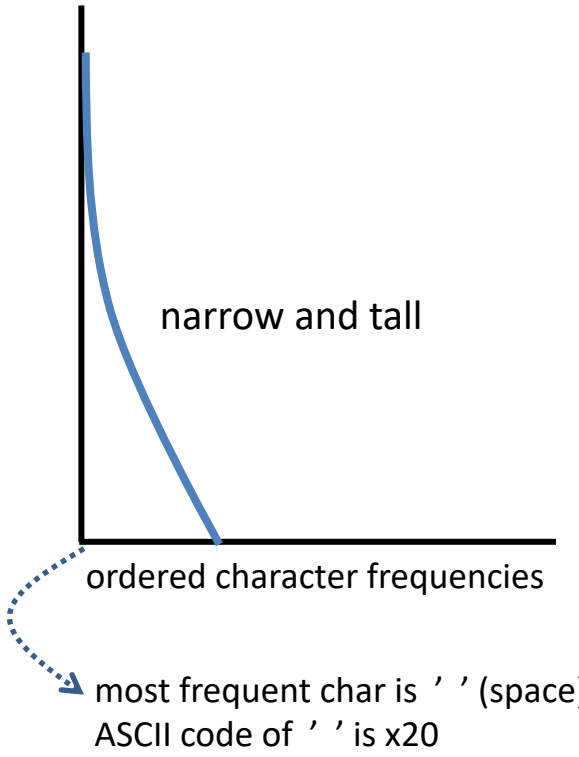
```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

000163D0   6F 6E 67 20 73 33 63 72 33 74 20 6B 33 79 2E 2E  ong s3cr3t k3y..
000163E0   2E 20 50 41 44 44 49 4E 47 72 65 61 6C 6C 79 20  . PADDINGreally
000163F0   6C 6F 6E 67 20 73 33 63 72 33 74 20 6B 33 79 2E  long s3cr3t k3y.
00016400   2E 2E 20 50 41 44 44 49 4E 47 72 65 61 6C 6C 79  .. PADDINGreally
00016410   20 6C 6F 6E 67 20 73 33 63 72 33 74 20 6B 33 79   long s3cr3t k3y
00016420   2E 2E 2E 20 50 41 44 44 49 4E 47 72 65 61 6C 6C  ... PADDINGreall
00016430   79 20 6C 6F 6E 67 20 73 33 63 72 33 74 20 6B 33  y long s3cr3t k3
00016440   79 2E 2E 2E 20 50 41 44 44 49 4E 47 72 65 61 6C  y... PADDINGreal
00016450   6C 79 20 6C 6F 6E 67 20 73 33 63 72 33 74 20 6B  ly long s3cr3t k
00016460   33 79 2E 2E 2E 20 50 41 44 44 49 4E 47 72 65 61  3y... PADDINGrea
00016470   6C 6C 79 20 6C 6F 6E 67 20 73 33 63 72 33 74 20  lly long s3cr3t
00016480   6B 33 79 2E 2E 2E 20 50 41 44 44 49 4E 47 72 65  k3y... PADDINGre
00016490   61 6C 6C 79 20 6C 6F 6E 67 20 73 33 63 72 33 74  ally long s3cr3t
000164A0   20 6B 33 79 2E 2E 2E 20 50 41 44 44 49 4E 47 72   k3y... PADDINGr
000164B0   65 61 6C 6C 79 20 6C 6F 6E 67 20 73 33 63 72 33  eally long s3cr3
000164C0   74 20 6B 33 79 2E 2E 2E 20 50 41 44 44 49 4E 47  t k3y... PADDING
000164D0   72 65 61 6C 6C 79 20 6C 6F 6E 67 20 73 33 63 72  really long s3cr
000164E0   33 74 20 6B 33 79 2E 2E 2E 20 50 41 44 44 49 4E  3t k3y... PADDIN
000164F0   47 72 65 61 6C 6C 79 20 6C 6F 6E 67 20 73 33 63  Greally long s3c
00016500   72 33 74 20 6B 33 79 2E 2E 2E 20 50 41 44 44 49  r3t k3y... PADDI
00016510   4E 47 72 65 61 6C 6C 79 20 6C 6F 6E 67 20 73 33  NGreally long s3
00016520   63 72 33 74 20 6B 33 79 2E 2E 2E 20 50 41 44 44  cr3t k3y... PADD
00016530   49 4E 47 72 65 61 6C 6C 79 20 6C 6F 6E 67 20 73  INGreally long s
00016540   33 63 72 33 74 20 6B 33 79 2E 2E 2E 20 50 41 44  3cr3t k3y... PAD
00016550   44 49 4E 47 72 65 61 6C 6C 79 20 6C 6F 6E 67 20  DINGreally long
00016560   73 33 63 72 33 74 20 6B 33 79 2E 2E 2E 20 50 41  s3cr3t k3y... PA
00016570   44 44 49 4E 47 72 65 61 6C 6C 79 20 6C 6F 6E 67  DDINGreally long
00016580   20 73 33 63 72 33 74 20 6B 33 79 2E 2E 2E 20 50   s3cr3t k3y... P
00016590   41 44 44 49 4E 47 72 65 61 6C 6C 79 20 6C 6F 6E  ADDINGreally lon
000165A0   67 20 73 33 63 72 33 74 20 6B 33 79 2E 2E 2E 20  g s3cr3t k3y...
000165B0   50 41 44 44 49 4E 47 72 65 61 6C 6C 79 20 6C 6F  PADDINGreally lo
```

Breaking the simple XOR cipher

T	i	t	a	n	…

⊕

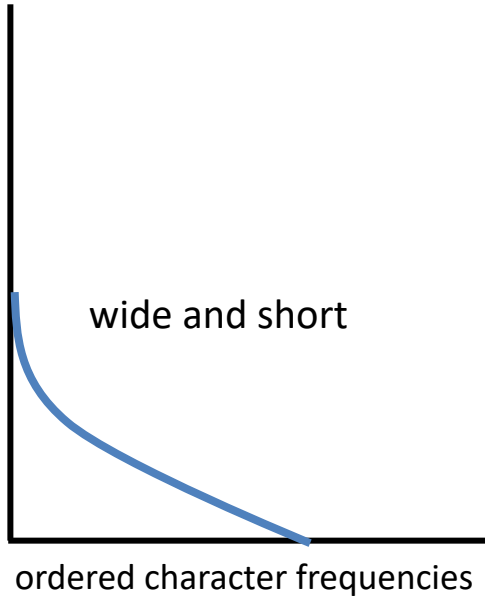
L	o	r	e	m		i	p	s	u	m		d	o	l	o	r		s	i	t		a	m	e	t	,		e	u		p
r	i	m	a		e	u	i	s	m	o	d		m	e	d	i	o	c	r	i	t	a	t	e	m		s	e	a	,	
s	i	n	t		a	l	i	q	u	i	p		e	s	t		t	e	,		e	t		q	u	o	t		s	a	e
p	e		o	m	i	t	t	a	m		s	i	t	.		I	d		v	e	l		m	a	l	i	s		s	u	m
m	o		d	o	l	o	r	e	s	,		p	r	o		o	d	i	o		d	o	l	o	r	u	m		e	i	.
	E	a	m		i	n	i	m	i	c	u	s		t	r	a	c	t	a	t	o	s		p	a	r	t	i	e	n	d
o		t	e	,		e	x		e	u	m		e	q	u	i	d	e	m		d	e	l	i	c	a	t	a		p	r
i	n	c	i	p	e	s	.		E	r	r	o	r		c	o	n	c	e	p	t	a	m		v	e	l		e	a	,
	s	a	l	u	t	a	t	u	s		d	e	l	i	c	a	t	i	s	s	i	m	i		v	i	t	u	p	e	r
a	t	o	r	i	b	u	s		u	t		e	a	m	.		N	a	m		n	e		a	n	i	m	a	l		e
x	p	e	t	e	n	d	a	,		v	i	d	e		u	b	i	q	u	e		c	o	n	v	e	n	i	r	e	
q	u	i		u	t	.		N	e		a	e	q	u	e		g	l	o	r	i	a	t	u	r		n	a	m	,	
s	e	d		a	l	t	e	r	u	m		i	n	i	m	i	c	u	s		d	i	s	s	e	n	t	i	a	s	
t	e	.		V	e	l		t	e		c	i	b	o		t	i	b	i	q	u	e	.								



Breaking the simple XOR cipher

Let's determine the length of the key ...

```
Lorem ipsum dolor sit amet, eu p
rima euismod mediocritatem sea,
sint aliquip est te, et quot sae
pe omittam sit. Id vel malis sum
mo dolores, pro odio dolorum ei.
 Eam inimicus tractatos partiend
o te, ex eum equidem delicata pr
incipes. Error conceptam vel ea,
 salutatus delicatissimi vituper
atoribus ut eam. Nam ne animal e
xpetenda, vide ubique convenire
qui ut. Ne aeque gloriatur nam,
sed alterum inimicus dissentias
te. Vel te cibo tibique.
```

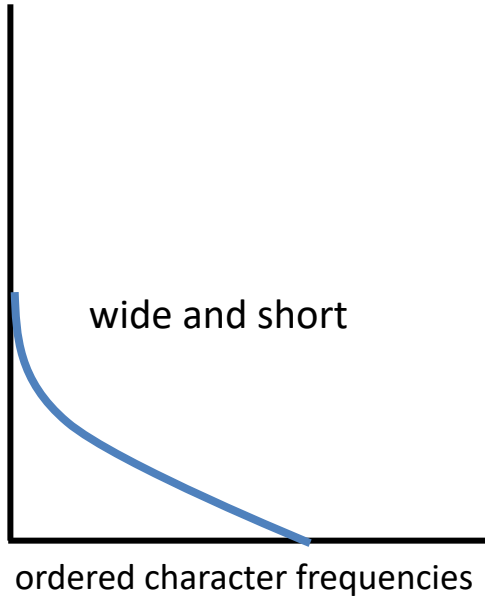


Breaking the simple XOR cipher

Let's determine the length of the key ...

e.g., keeping every 3rd letter...

L	e	i	u	d	o	s		e						
i		i	o	m	i	r	a	m	e					
n	a	q	p	s	t			o	s					
p	o	t	m	i			l	a	s	u				
o	o	r	,	r	o	o	o	r		.				
a	i	m	u	t	c	t		r	e					
o	e	e	e		u	e	d	i	t	p				
n	p	.	r	r	o	e	a	v		,				
a	t	u	d	i	t	s	i	i	p					
a	r	u	u	e	.	a	n	a	m					
p	e	a	v	e	b	u	c	v	i					
i	t	N	a	u	g	r	t		m					
s		t	u	i	m	u	d	s	t	s				
e	V			b	t	i	e							

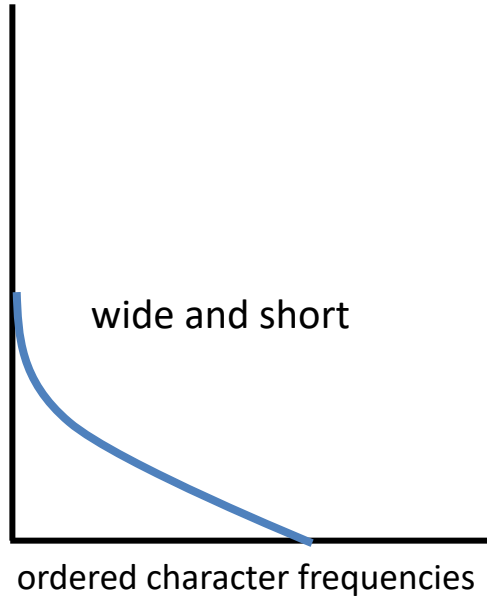


Breaking the simple XOR cipher

Let's determine the length of the key ...

e.g., keeping every 4th letter...

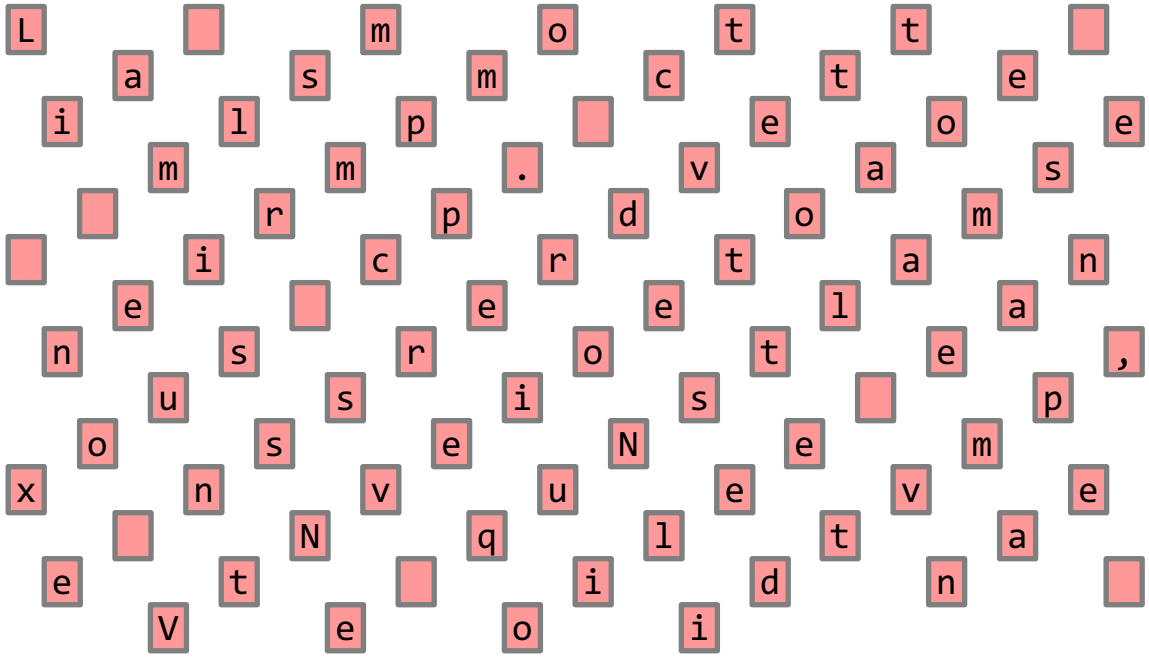
L	m	s	d	r	t	e	e
r	m	s	i	I	i	e	i
s	o	q	p	o	e	q	a
p	,	a	s	a	t	a	u
m	p	e	o	i	p	o	a
o	u	m	e	o	s	p	i
i	i	u	e	a	e	i	a
a	e	,	d	b	r	a	u
x	u	N	e	i	q	n	a
q	a	r	i	t		u	i
s	V	t	i			s	a
t							i

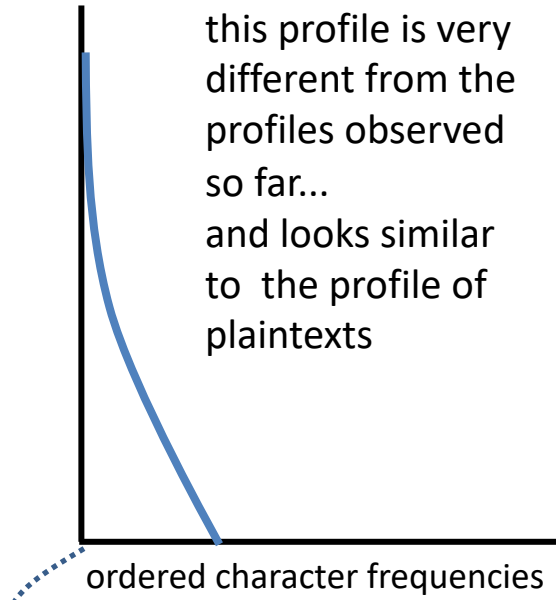


Breaking the simple XOR cipher

Let's determine the length of the key ...

e.g., keeping every 5th letter...





most frequent char is x74

' ' + ▢ = x20 + ▢ = x74

▢ = x74 + x20 = x54 = 'T'

Let's determine the characters of the key ...

One-time pad

- encryption
  - represent the message as a sequence of bytes: $m_1\ m_2\ ...\ m_L$
  - take a sequence of true random bytes: $k_1\ k_2\ ...\ k_L$
  - obtain the encrypted message by XOR-ing them together:

    $m_1\ m_2\ ...\ m_L + k_1\ k_2\ ...\ k_L = c_1\ c_2\ ...\ c_L$ where $c_i = m_i + k_i$ for all i = 1, ..., L

- decryption
  - XOR the same stream of key bytes to the encrypted message:

    $c_1\ c_2\ ...\ c_L + k_1\ k_2\ ...\ k_L = m_1\ m_2\ ...\ m_L$ (because $c_i + k_i = m_i + k_i + k_i = m_i$)

Properties of the one-time pad

- **perfect secrecy**
  - informally: observing the encrypted message provides no information (in an information theoretic sense) about the original message
  - formally: H(m|c) = H(m), where H(x) is the entropy of x
  - illustration
    - » let the clear message be: x41 (ASCII code for letter 'A')
    - » let the key be: xAD
    - » encrypted message observed by the attacker: xEC (x41 + xAD = xEC)
    - » from the attacker's point of view, the original message may be:
      - x00 if the key was xEC
      - x01 if the key was xED
      - ...
      - x41 ('A') if the key was xAD
      - x42 ('B') if the key was xAE
      - x43 ('C') if the key was xAF
      - ...
      - xFF if the key was x13

all these cases have equal probability, because the key is chosen randomly

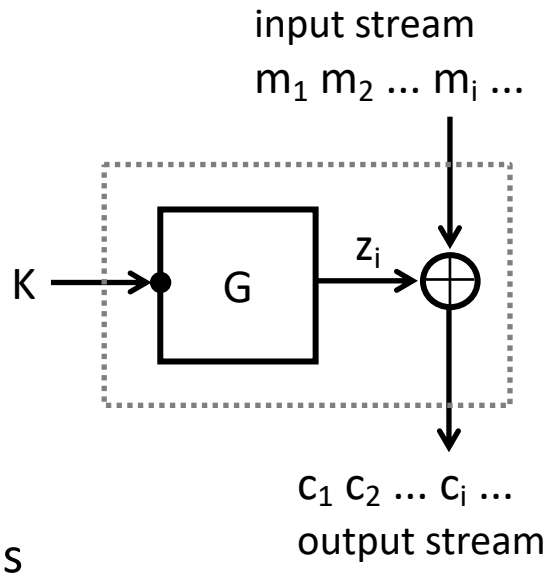
Properties of the one-time pad

- large key size
  - needs a truly random key that has the same length as the (compressed) message
- impractical in many applications
  - how to send the key in a secure way to the recipient?
  - in practice, the only possibility is to exchange a large amount of truly random key material in an *out-of-band* manner (e.g., by physical meeting, via a quantum channel) before the communication takes place
  - we have to do this with all potential communication partners
  - key management becomes cumbersome

General model of stream ciphers

- idea: simulate the truly random key stream of the one-time pad with a pseudo-random sequence generated from a random seed

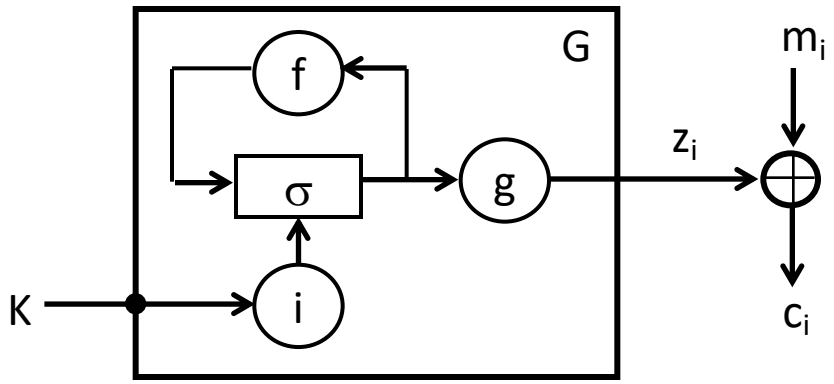
- terminology:
  - m_i – plaintext character
  - c_i – ciphertext character
  - z_i – key-stream character
  - K – key (seed)
  - G – key-stream generator



- application:
  - encryption of data → confidentiality services
  - PRNG (Pseudo-Random Number Generator)

- examples:
  - LFSR based (hardware), RC4, A5 (GSM), E0 (Bluetooth), Salsa20/Chacha

Inside the key stream generator

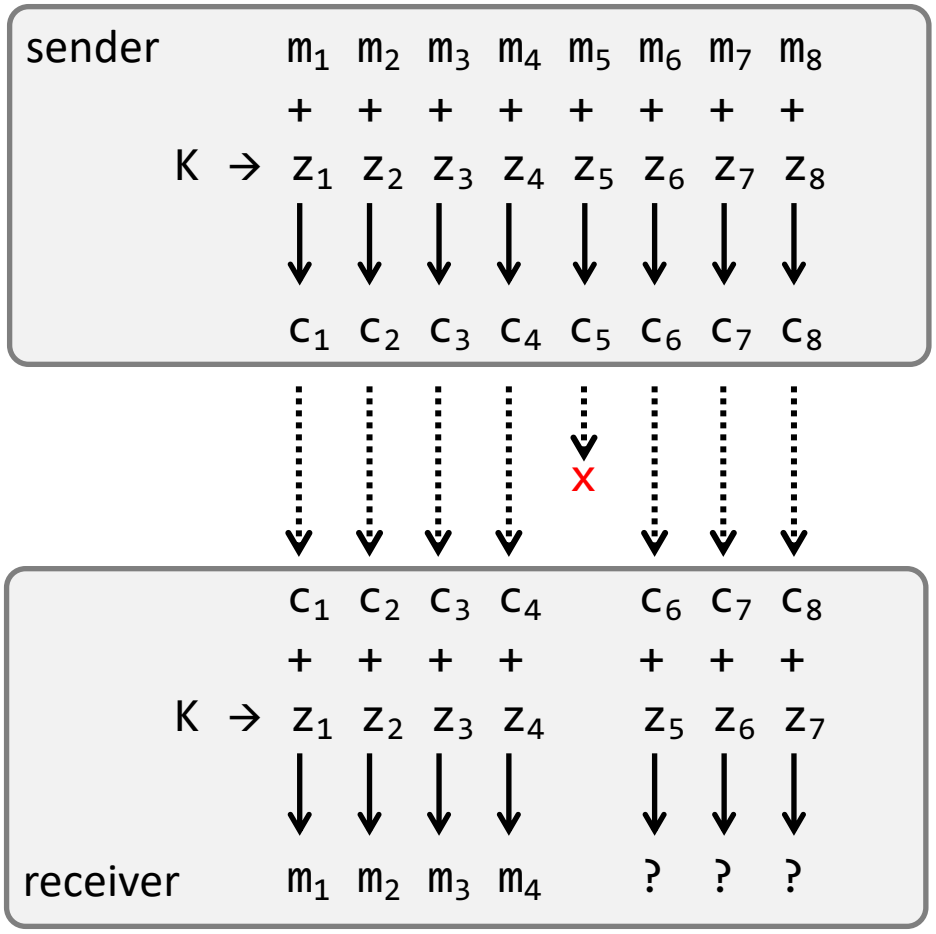


- the key stream is generated independently of the plaintext and of the ciphertext
  - key K is used to initialize an internal state σ (seed the generator)
  - once initialized, the state is updated using some function f after each step
  - the output is generated from the state with another function g
- effective size of the state space must be large
  - otherwise the key stream starts repeating
  - if two plaintext characters m_i and m_j are encrypted with the same key character z, then the XOR sum of the ciphertext characters $(m_i + z) + (m_j + z) = m_i + m_j$

Other properties of stream ciphers

- stream ciphers are usually very efficient
  - fast (especially in hardware)
  - require small memory to store the internal state and the code of the generation and state update functions
- the ciphertext always has the same length as the plaintext (in some block encryption modes, the ciphertext is longer)
- synchronization is needed between the sender and the receiver
  - loss of synchrony needs to be detected and addressed
- stream ciphers do not provide any integrity protection !!!
  - an attacker can make changes to selected ciphertext characters and know exactly what effect these changes have on the plaintext
  - the receiver may not notice these changes

Loss of synchrony illustrated



$$c_6 + z_5 = m_6 + z_6 + z_5 \neq m_6$$
$$c_7 + z_6 = m_7 + z_7 + z_6 \neq m_7$$
$$\ldots$$

Controlled modification of the ciphertext

- an attacker can add any value Δ to a ciphertext character c_i
- the receiver will decode: $(c_i + \Delta) + z_i = (m_i + z_i + \Delta) + z_i = m_i + \Delta$
- hence, the plaintext character will be modified by Δ
  - bits of m_i will be flipped in every position where Δ has a bit 1
  - bits of m_i will be unchanged in every position where Δ has a bit 0

$$
\begin{array}{ll}
m_i & = 00101100 \\
 & \quad ++++++++ \\
\Delta & = 10000100 \\
\hline
m_i' & = 10101000
\end{array}
$$

Summary

- XOR operation on bits and bit vectors, 3 main properties of XOR
- hexadecimal numbers and ASCII codes of characters
- the simple XOR cipher and how to break it
- the one-time pad and its properties
  - perfect secrecy
  - the key must be as long as the plaintext (impractical)
- stream ciphers
  - try to simulate the one-time pad, use a pseudo-random key stream
  - general structure and operation
  - properties
    - » must have a large state space
    - » need for detecting loss of synchrony
    - » no integrity protection at all
  - examples: LFSR-based, RC4, Salsa20

Control questions

- What are the three main properties of the XOR operation?
- How does the one-time pad work?
- What does perfect secrecy mean intuitively?
- What are the disadvantages of the one-time pad?
- How stream ciphers work? (general internal structure)
- Why should the state space of a stream cipher be large?
- What are the advantages of stream ciphers?
- What are the disadvantages of stream ciphers?