TK2100: Informasjonssikkerhet
Lesson 02: Cryptography

Dr. Andrea Arcuri
Westerdals Oslo ACT
University of Luxembourg

Goals

- Understand the basics of cryptography and crypto analysis
- Knowledge of the most common crypto algorithms, and when/where they should be used
- Crypto is heavily based on **MATH**… but we will not dig into too many details…
- Revision of HTML/CSS/JS
  - Needed for exercises, and rest of the course

Cryptography

- Practice and study of technique for *secure communication*, even in the presence of adversary listeners
- Why? Communication from you computer to a server (eg, on Internet) goes through **many** routers in different countries
  - Anyone having access to these routers can read your messages
- You would like your Internet Banking and access to email to be protected…

13 machines from Fjerdingen to *google.com*

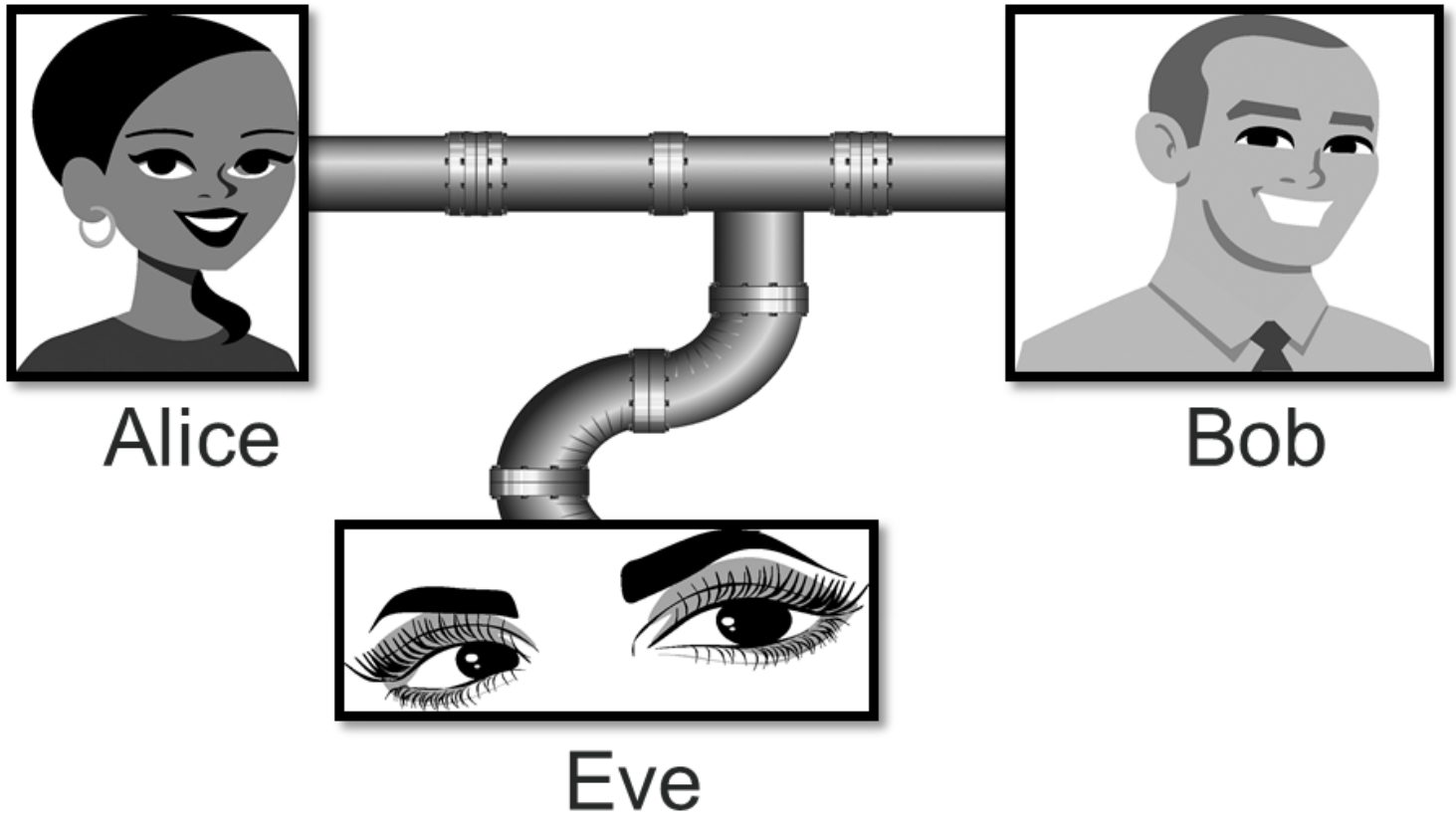
```
MINGW64:/e/WORK/code/teaching
arcur@DESKTOP-IR7IFID MINGW64 /e/WORK/code/teaching
$ tracert google.com

Tracing route to google.com [216.58.209.142]
over a maximum of 30 hops:

  1     1 ms     1 ms     1 ms  10.96.16.2
  2    <1 ms    <1 ms    <1 ms  10.240.33.1
  3    <1 ms    <1 ms    <1 ms  97.sopp10.party.fredrikstadlan.no [81.175.52.97]
  4    <1 ms    <1 ms    <1 ms  77.88.111.60
  5     1 ms    <1 ms    <1 ms  po5-vl3008.ooe121-060.as41572.net [81.175.32.117]
  6     5 ms     3 ms     4 ms  ten-2-1-vl1504.ooe121-070.as41572.net [81.175.32.226]
  7     1 ms     1 ms     1 ms  oso-b3-link.telia.net [62.115.12.37]
  8     8 ms     8 ms     8 ms  s-bb4-link.telia.net [62.115.137.252]
  9     8 ms     8 ms     8 ms  s-b5-link.telia.net [62.115.133.27]
 10     9 ms     9 ms     9 ms  google-ic-314684-s-b5.c.telia.net [62.115.61.30]
 11     8 ms     8 ms     8 ms  216.239.49.13
 12     9 ms     8 ms     8 ms  216.239.49.217
 13     8 ms     8 ms     8 ms  arn09s05-in-f142.1e100.net [216.58.209.142]

Trace complete.
```

- Even if Eve can intercept messages between Alice and Bob, she should not be able to understand them
- Note: Alice, Bob and Eve are typical names in security examples



Encrypt/Decrypt

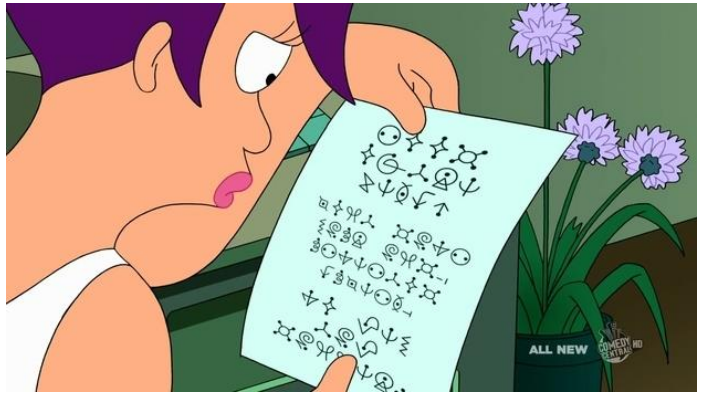
- Scenario: Alice wants to send message T to Bob
- Alice encrypt T using an encryption algorithm *e()* with a secret S, ie *e(T,S)=C*, where the resulting C is called *ciphertext*
- Bob receives encrypted C, and decrypt it using *d()* and secret S', ie *d(C,S') = T*
  - Note: we will see some cases when S != S'

Why is it secure???

- *Even if* Eve can intercept the ciphertext C, and knows all the details of the algorithms *e()* and *d()*, she should **NOT** be able to *decrypt* the original message
- Ie, only by knowing S' one should derive *d(C,S')=T*
  - S' can be just a small text, eg like a password
- But how can we write algorithms *e()* and *d()* to ensure it?
  - *e()* and *d()* are usually called *ciphers*
- During history, many different algorithms proposed
- Problem: what seems *secure* today can get *broken* tomorrow…

Substitution Cipher

Futurama Alien Language

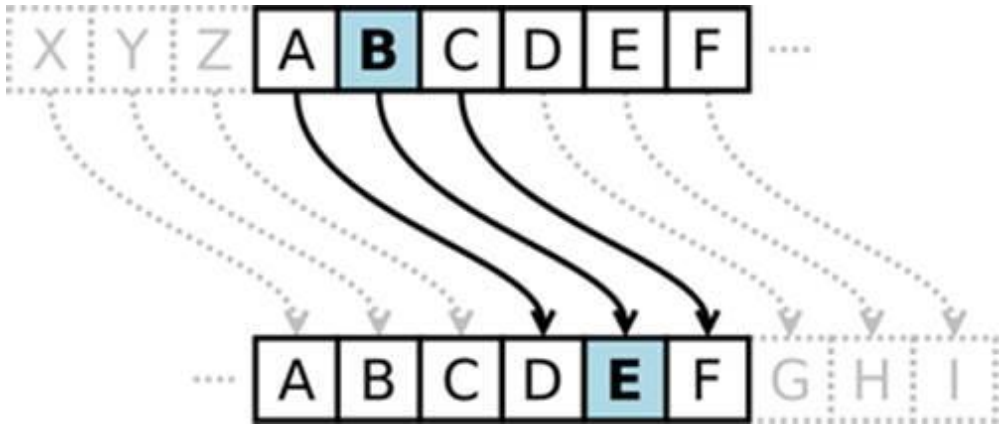


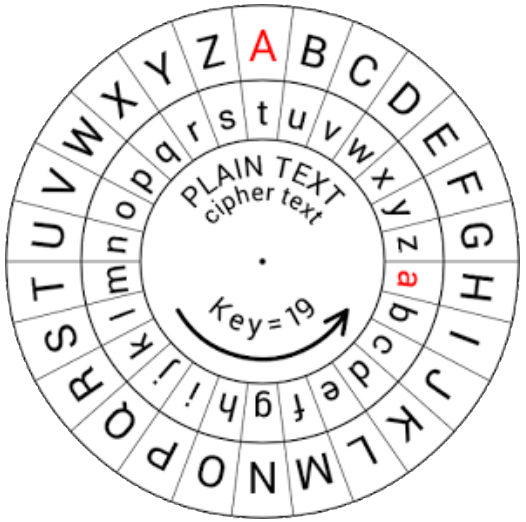
A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12
.	!	-	'	;	:	?						

- Each symbol is actually representing a letter
- 1-to-1 mapping from letter to Alien symbol
- Text on the right says: “WATCH FUTURAMA THURSDAYS AT 10”

Ciphers during Roman Empire

- Communications to generals in the army should be kept secret
- Even if messages get caught by the enemy
- Do not need to invent special symbols, but just mapping from a letter to another





Mapping in Code

- Before digging into it, need to recall how characters are stored in a computer
- As for any data, a character is a 0/1 binary sequence
- A character can be seen as a *number*
- Different mappings from numbers to characters to display, called *charsets*
- Different charsets, but we ll just focus here on ASCII
  - You will see more on this next semester, eg in Algorithms course, like UTF-8

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

ASCII 1967

- 7 bits per char, so 2^7=128 chars
- Latin letters and symbols (no Norwegian øæåØÆÅ)
  - for Norwegian you need other encodings like UTF-8…
- 0-31 and 127 are *control characters*, non-printable
  - many, but not all, are just there for historical reasons, and not really used any more
- Upper and lower case letters use different codes
  - a=97 whereas A=65

Substitution in Code

- c’ = c + s
- Where “s” is just a constant, and will be your “secret” in this cipher (also called *key*)
- Example, if s==7, then letter “Z” (90) turns into an “a” (97)
- Slightly tricky: you need to use module “%” to avoid the sum going over 127, and also need to avoid control 0-32 and 127 values
- Note: implementing this will be one of the exercises…

Secret

- The added constant “s” (eg 7) is going to be the “secret” you would share before communication
- But how “secure” is such cipher?
- You could brute force substitutions from s=1 to s=128, and check against a dictionary for word matches
- **Trivial** to break with a PC

Generic Mapping

- Each letter directly mapped to another letter
- Can be implemented with an array
- But is it better than c’=c+s???

A	B	C	D	E	F	G	H	I	J	K	…
W	C	P	T	E	J	Z	A	U	M	B	…

Combinations

- To keep it simple, let's just consider the 26 English letters
- 26 different ways to map "A" to another letter
- Once "A" mapping is chosen, cannot be reused, and so left 25 options for "B"
- Once "B" (and "A") are chosen, 24 options for "C"
- and so on, until only 1 option for "Z" when all other letters are already mapped
- So, 26*25*24*23…*3*2*1 = 26! =~ 4*10^26
- "Secret" to share: 26 letter mapping, whereas before was just a number

Polygraphic Substitution Ciphers

- 10^26 is a lot, but could do more
- Instead of mapping single letter, could do mapping of blocks of letters, eg by 2
- There are 26*26=676 pairs of letters
- So, factorial 676! combinations, which is HUGE
- Negative side: now “secret” to share is 676 pair mapping…
- Could even consider more than 2 letters at a time, ie blocks of *m* letters
  - eg, Vigenere Cipher, 1586, we will see in the next slides
- Again, **is it secure???**

AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	…
BW	CA	PR	TT	JE	JD	ZY	AB	UY	RM	KB	…

Nope… not secure at all…

- Substitution ciphers do NOT hide the frequency of the characters
- Some characters are used more often, and the mapped ones will appear as often in the ciphertexts
- Those frequency values can be computer from existing corpora, eg books, dictionaries and magazines
- Eg, the most common used letter in English is “e”
- Can analyze frequency of letters in ciphertext to infer the used secret, ie mapping
- Note: this assumes knowledge of type of message (eg text and language vs other kind of data, eg images)

a:	8.05%	b:	1.67%	c:	2.23%	d:	5.10%
e:	12.22%	f:	2.14%	g:	2.30%	h:	6.62%
i:	6.28%	j:	0.19%	k:	0.95%	l:	4.08%
m:	2.33%	n:	6.95%	o:	7.63%	p:	1.66%
q:	0.06%	r:	5.29%	s:	6.02%	t:	9.67%
u:	2.92%	v:	0.82%	w:	2.60%	x:	0.11%
y:	2.04%	z:	0.06%				

Mark Twain: Tom Sawyer(Engelsk)

One-Time Pads

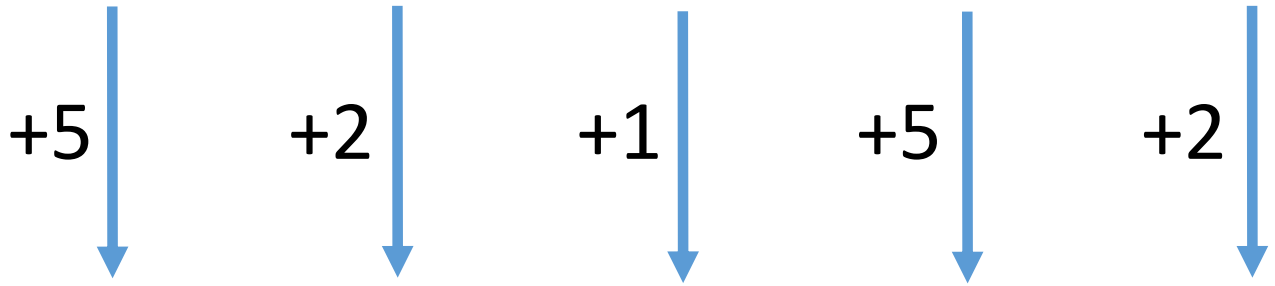
Vigenere Cipher

- Made in 1586
- Substitution cipher (discussed in previous slides)
- Encrypt in blocks of *m* characters
- Each character is shifted by s_i value, where *i* is the position in the block
- The “secret” is hence these *m* shift values $s_1, s_2, \ldots s_m$

Vigenere Example

- S = {5, 2, 1}, *m*=3, text = “Hello”

H	e	l	l	o
72	101	108	108	111



M	g	m	q	q
77	103	109	113	113

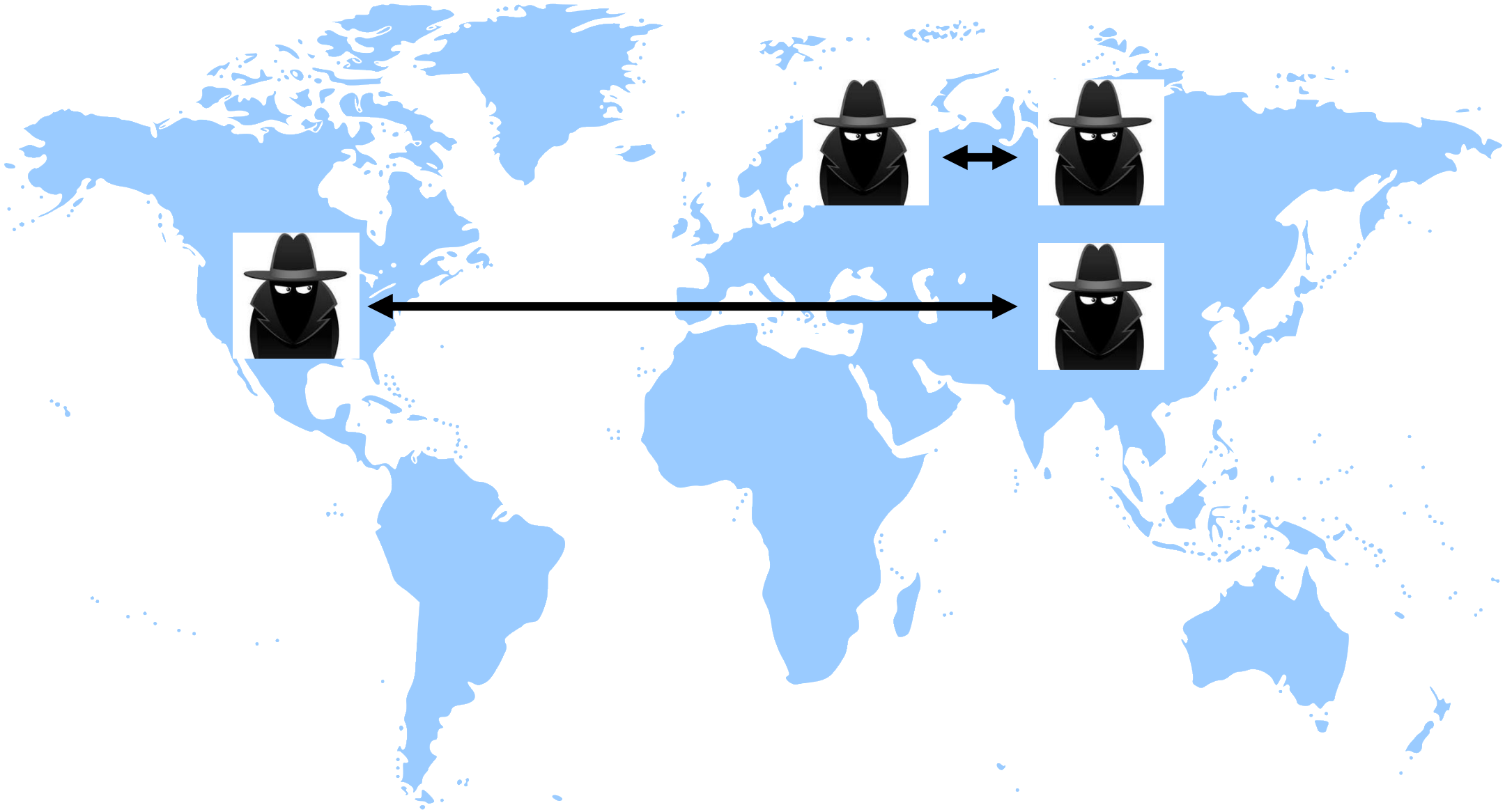
One-Time Pads

- As said in previous slides, substitution ciphers are not secure
- The same black of *m* letter will be mapped same way
- Given a *long enough* ciphertext, can do statistical analysis of most common patterns to infer the secret S
- But… what if *m* is longer than the cipher text…
- … and the values s_i are chosen at random???
- Then there would be *impossible* to decrypt the ciphertext, as there would be no correlation for statistical analysis

Sharing of Secrets

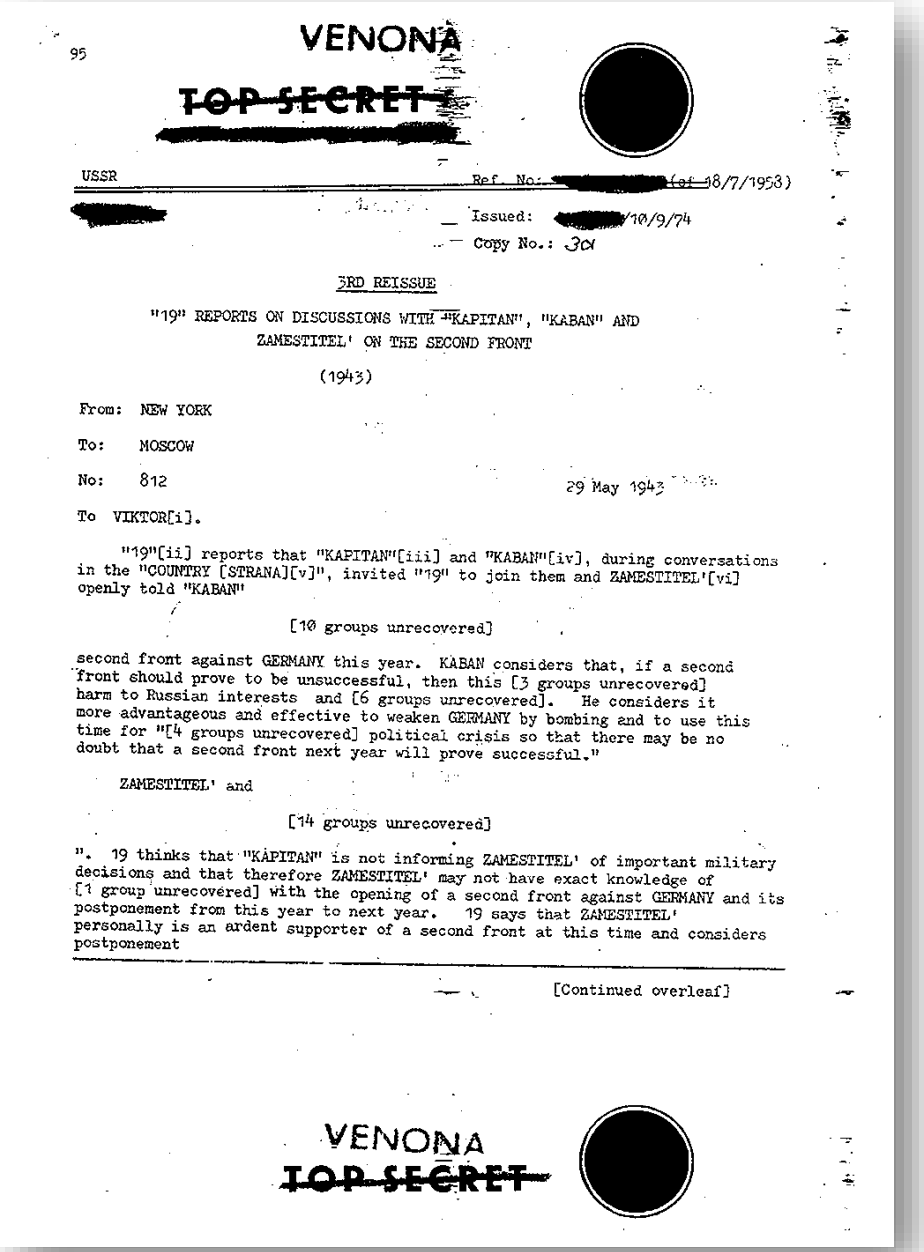
- With base Vigenere Cipher, need first to share a secret of length *m* (eg 10 values), and then can encrypt/decrypt as long text as I want, for how many times I want
  - Eg, a whole book
- With One-Time pad (ie Vigenere with long random S), the length of the secret is equal or more than the ciphertext
- Encrypting a book would require a S that is as long as the book
- Is securely sharing a long secret S *practical*???

Can first share (long) S when in secure context, and then *afterwards* use it as One-Time Pad when requiring secure communications



Cold War 1962-1979

- One-Time Pads were used by spies during the Cold War between USSR and USA
- Problem: once a spy used the whole S for his communications, could not get a new one (eg, when infiltrated in an enemy country)… so started to re-use it…
- … but when re-using it, it is exactly like Vigenere, so can be statistically analysed and broken
- Ie, One-Time Pad is secure only as long as secret S is not re-used



95

VENONA

~~TOP SECRET~~ [redacted]

USSR Ref. No: [redacted] (of 18/7/1953)

[redacted] Issued: [redacted] 10/9/74

Copy No.: 301

3RD REISSUE

"19" REPORTS ON DISCUSSIONS WITH "KAPITAN", "KABAN" AND ZAMESTITEL' ON THE SECOND FRONT

(1943)

From: NEW YORK

To: MOSCOW

No: 812 29 May 1943

To VIKTOR[i].

"19"[ii] reports that "KAPITAN"[iii] and "KABAN"[iv], during conversations in the "COUNTRY [STRANA][v]", invited "19" to join them and ZAMESTITEL'[vi] openly told "KABAN"

[10 groups unrecovered]

second front against GERMANY this year. KABAN considers that, if a second front should prove to be unsuccessful, then this [3 groups unrecovered] harm to Russian interests and [6 groups unrecovered]. He considers it more advantageous and effective to weaken GERMANY by bombing and to use this time for "[4 groups unrecovered] political crisis so that there may be no doubt that a second front next year will prove successful."

ZAMESTITEL' and

[14 groups unrecovered]

". 19 thinks that "KAPITAN" is not informing ZAMESTITEL' of important military decisions and that therefore ZAMESTITEL' may not have exact knowledge of [1 group unrecovered] with the opening of a second front against GERMANY and its postponement from this year to next year. 19 says that ZAMESTITEL' personally is an ardent supporter of a second front at this time and considers postponement

[Continued overleaf]

VENONA

~~TOP SECRET~~

Binary One-Time Pads

- So far we have been talking of written text
- But might need to encrypt any kind of data (eg images)
- So, we need to generalize to 0/1 bit strings
- But, let’s talk of XOR first…

Exclusive-or (XOR)

- Bit operation: takes 2 bits, gives 1 as output
- Result 0: if the bits are equal
  - 0 xor 0 = 0
  - 1 xor 1 = 0
- Result 1: if the bits are different
  - 0 xor 1 = 1
  - 1 xor 0 = 1
- For bit strings, xor one bit at a time
  - eg, 10100 xor 01101 = 11001
- Highly efficient operation

XOR	0	1
0	0	1
1	1	0

A	1	0	1	0	0
B	0	1	1	0	1
XOR	1	1	0	0	1

Binary One-Time Pads: Cont.

- Message *M* as binary 0/1 string
- Secret *P* (pad) being a *random* binary string
- Ciphertext simply: *C = M xor P*
- Recover message: *M = C xor P*
- Highly efficient to compute, can be applied to any kind of data

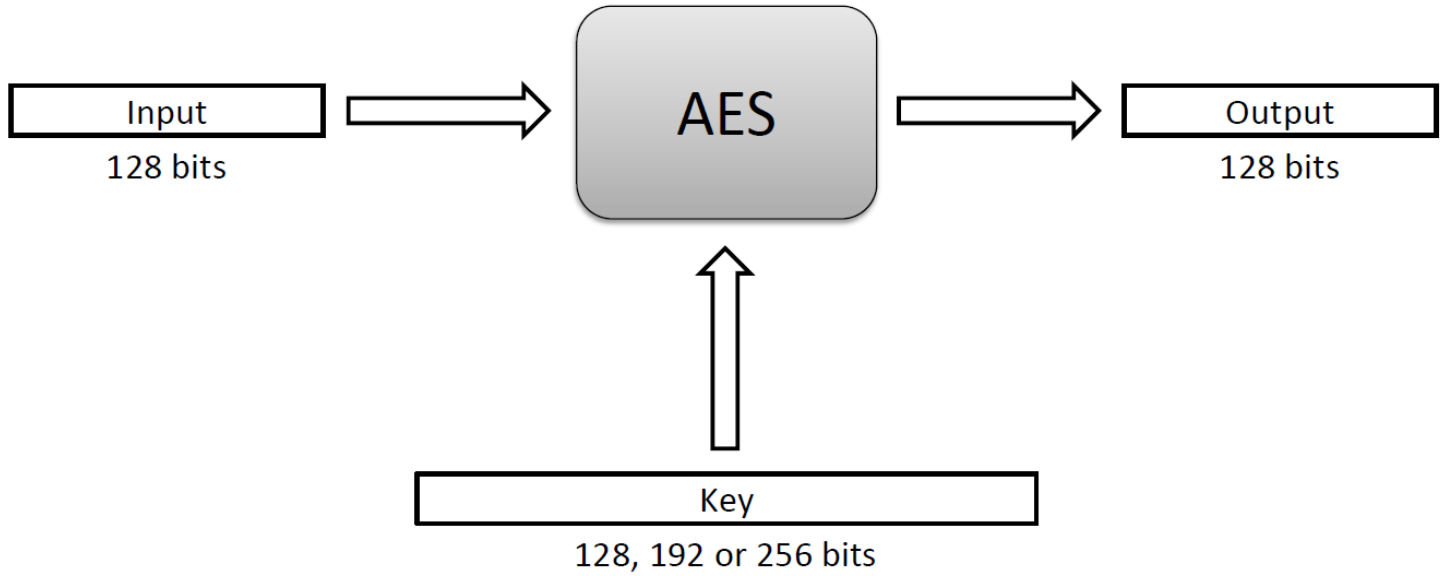
Advanced Encryption Standard (AES)

Advanced Encryption Standard (AES)

- One-Time Pads are secure as long as the pad is not re-used
  - So, not too practical for current security needs
- AES (1997) is current best approach for *symmetric* encryption
- *Symmetric*: both sender and receiver use the same key/secret
  - ie, S=S’
- 3 different versions: 128, 192 and 256 bits
  - tradeoff between security and efficiency of the algorithms
  - ie, more secure version (256) takes longer to compute

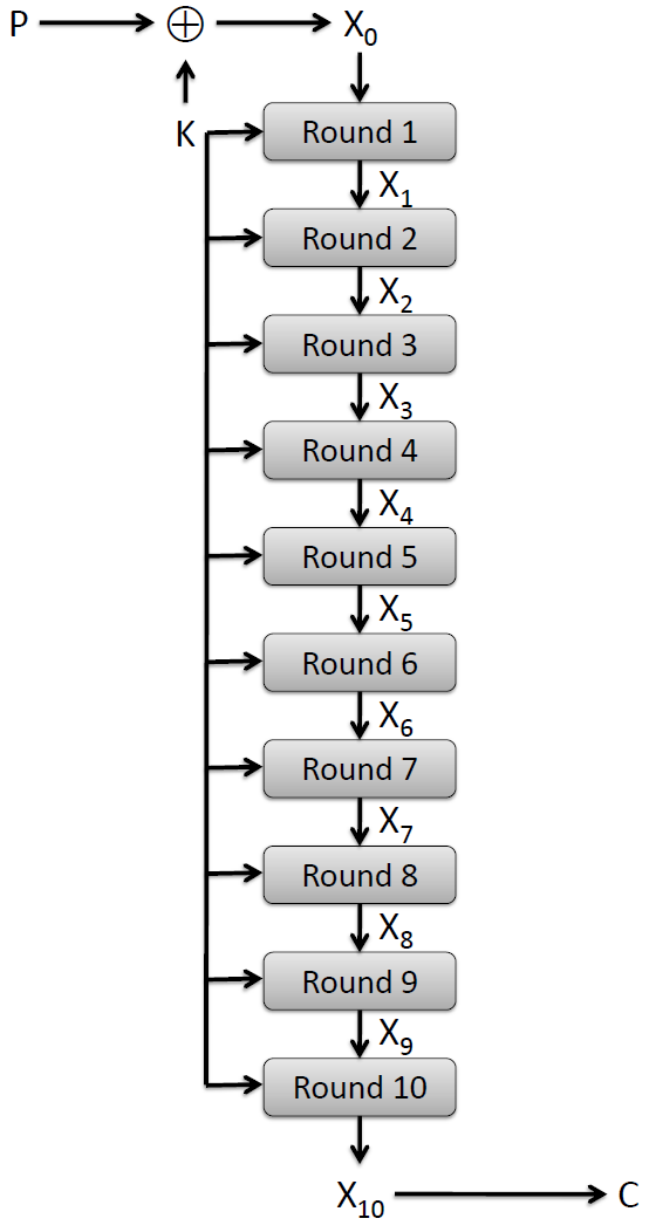
Blocks

- AES encrypts/decrypts a message in blocks of 128 bits
- But how does it works internally?



128 Bit Version

- Quite complex algorithm, we ll NOT see full details
  - see book if interested
- Starting from $X_0 = P\ xor\ K$
  - *P* is the plaintext, whereas *K* is the 128 bit key
  - So, first step is same as One-Time Pads
- $X_i = r(X_{i-1}, K)$
  - complex steps: S-box substitution, permutation, etc
- Repeated 10 times until ciphertext $C = X_{10}$



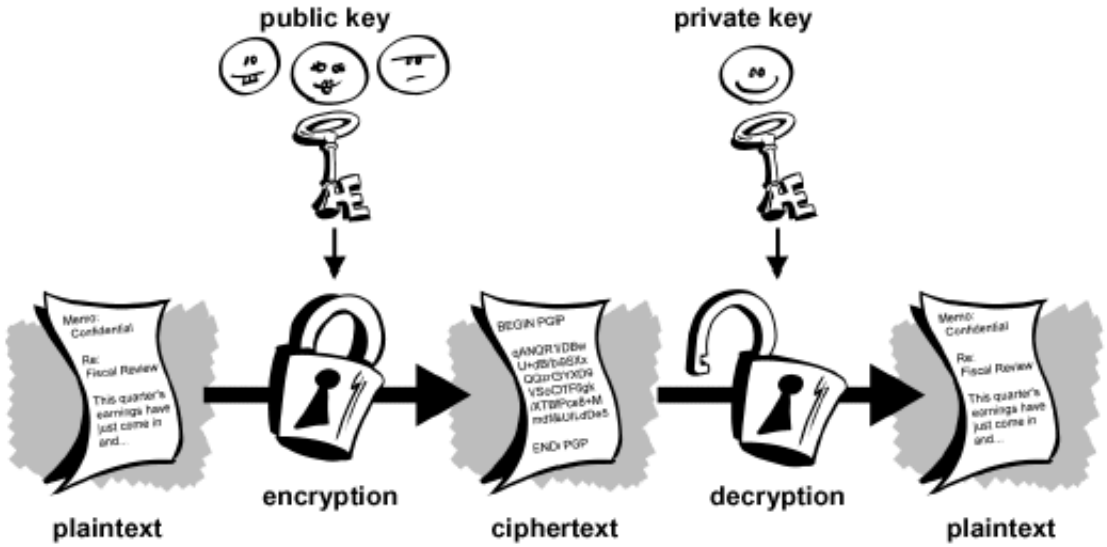
AES Security

- AES is highly secure, especially the 256 bit version
  - Not possible to brute force it with current hardware/systems
- In contrast to One-Time Pads, the secret/key can be reused
- *But* how can Alice and Bob share a 256 bit key *secretly* if they do not know each other?
- Eg, if you want to communicate *securely* with different web sites (your bank, Facebook, etc.), how to share the secret key?
- With symmetric keys, you can't do it directly…

Public-Key Cryptography

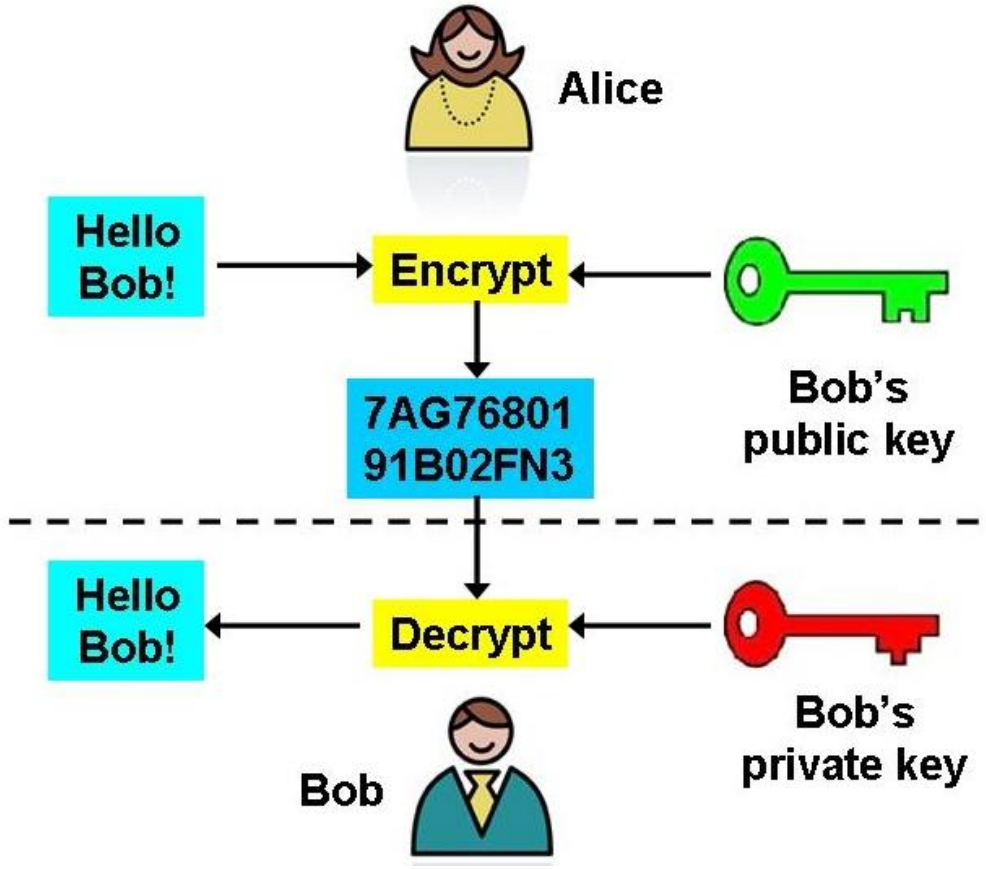
Asymmetric Keys

- The key K used for encryption is different from the K’ used for decryption
- So, need at least 2 keys
- Why? Can *publicly* share the encryption key K, but then keep *private* the decryption K’ one



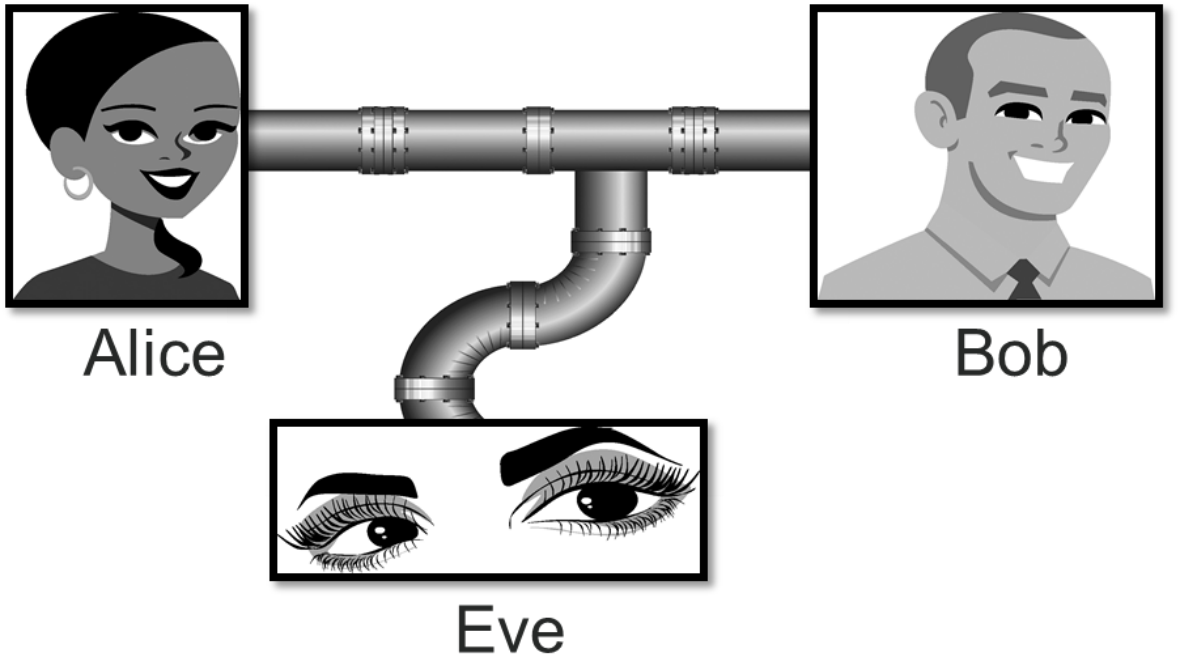
Public/Private Keys

- Each user will have 2 keys: one *private*, and one *public*
- When Alice wants to speak with Bob, it will be *encrypt* with the *public* key of Bob
- Only Bob can *decrypt* the ciphertext using his *private* key
- When replying, Bob will use Alice's *public* key, and so on



Why does it work?

- Alice and Bob do publicize and share their public keys in the open
- But how can we ensure that, even if Eve intercepts the encrypted messages, she cannot brute force and decrypt them even without Bob's private key?



RSA Cryptosystem

- Named after its authors
  - Adi Shamir, Ron Rivest and Len Adleman
- Security based on mathematical problems (*factorization*) that we do not know how solve efficiently
- $n = pq$, where p and q are prime numbers
  - prime number, only dividable by 1 and itself, eg 1, 2, 3, 5, 7, 11, …
- Given n, we have no efficient algorithm to derive p and q

RSA Cont.

- Bob generates 2 *large* prime numbers *p* and *q,* where $n = pq$
- Choose *e* such that it is relative prime to $\phi(n)$
- $\mathrm{d} = e^{-1}\ mod\ \phi(n)$
- Note: mathematically details of these formulas are **NOT** important for this course
- Public key: pair *(e, n)*
- Private key: *d*
- Note: if one would derive *p* and *q* from *n,* then deriving private key *d* would be trivial

RSA Encryption/Decryption

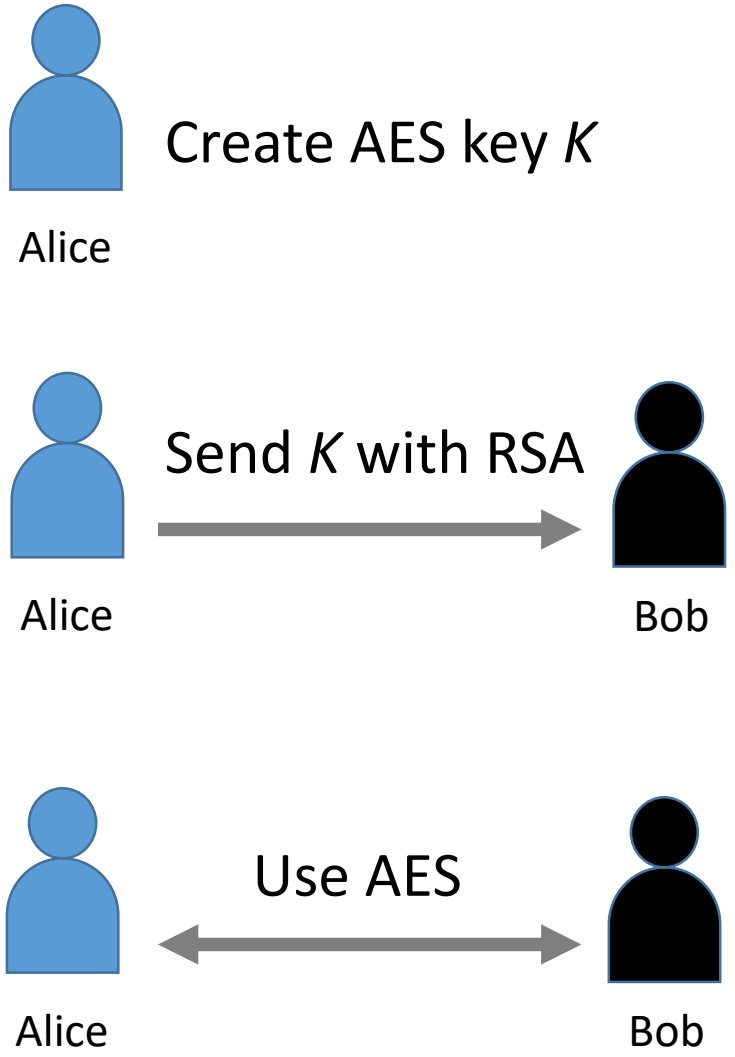
- Alice sending message *M* to Bob using Bob's public key *(e,n)*
- Note: *M*, being a 0/1 bit string, can be represented with a (large) number
- Encryption using public *(e,n)*: $C = M^e \ mod \ n$
- Decryption using private *d*: $M = C^d \ mod \ n$

RSA Problem

- Factorization (ie deriving *p* and *q* from *n*) is a hard problem, but can be brute forced if *n* is not too large
- Currently, to be secure, recommended to use *n* with at least 2048 bits, which is a HUGE number
- Problem: because RSA works on very large numbers, it is *slow,* much, much slower than AES
- So, not efficient when needing to send many encrypted messages

Key Exchange: RSA + AES

- In AES we have the problem of securely share the symmetric key
- In RSA, we don't have such issue, as public keys can be shared in the open
- But RSA is very slow
- RSA + AES: using RSA in the first step to share an AES key, and then keep communicating by using AES



HTTPS: SSL/TLS

- When you access websites using HTTPS, those exchanges of messages/files are encrypted
- Usually using RSA under the hoods
- Certificate: the public key for the site you want to access (eg Facebook)
- Certificate Authority: handling of certificates for the different web sites
- Will go back on this when doing web security later in the course

HTML + CSS + JavaScript

Overlap With Other Courses

- This course is about Information Security, not programming
- But to understand security, you need to know how programming works
- Studying just theory is not the same as doing some actual *hacking*
- Later in the course there is going to be a focus on web, so need to understand the bases of web development
- Although this is not a course on web development, we will go through it, even though the details will / should had been covered in other courses…
- Think about it as a repetition…
- However, the (3-h written) exam will be “theoretical”, and I will **NOT** ask you to write code (although I *might* show you code, and ask you questions about it)

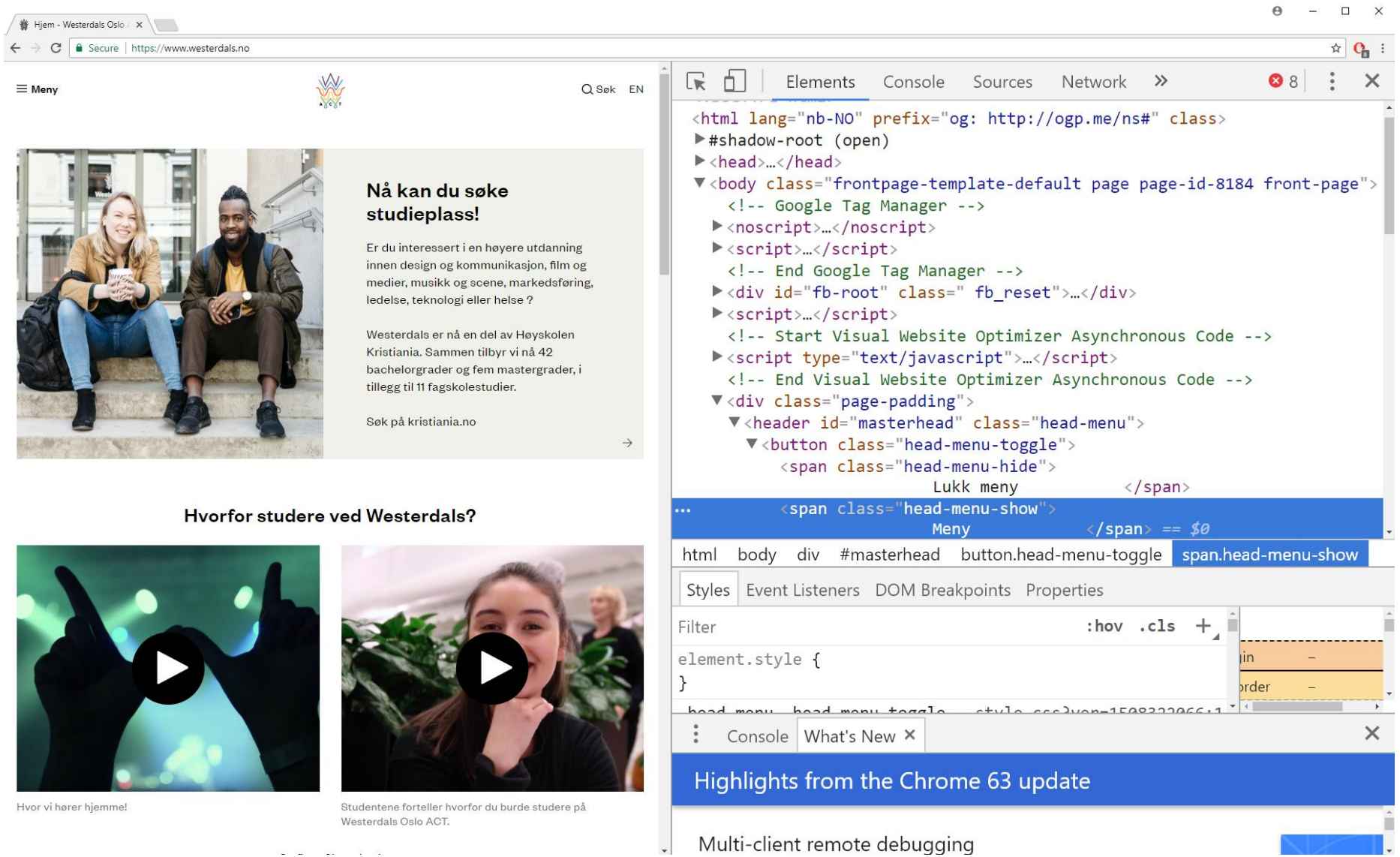
HTML (**H**yper**T**ext **M**arkup **L**anguage)

- A **markup** language, not a programming one
- Composed by tags, eg <body>, <p>, <a>, etc
- When a browser receives a HTML file, it will parse its content, and create a GUI based on those tags
- Most HTML pages do have references to other files, like for example images
- Before a page is fully displayed in browser, those other files need to be downloaded

A Cat and a Dog

```
<HTML>
  <HEAD>
    <TITLE>Example</TITLE>
  </HEAD>
  <BODY BGCOLOR="gray">
   <h2> A Cat and a Dog </h2>
   <DIV>
    <IMG SRC="figs/cat.jpg" style="width:256px">
    <IMG SRC="figs/dog.jpg" style="width:256px">
   </DIV>
 </BODY>
</HTML>
```

Chrome -> More tools -> Developer tools



CSS (Cascading Style Sheets)

- Language used to specify how the different HTML components should be displayed
- Needed to specify for example: layout, colors and fonts
- Can be embedded directly in HTML pages (eg, in the header)
- However, often in separated files
  - for separation of document content from document presentation

JavaScript

- JavaScript (JS) has nothing to do with Java
- Programming language executed in the browser
- JS code referenced by webpages like any other resource (eg images and CSS files), or can be embedded directly in HTML
- JS can manipulate the DOM (Document Object Model) to alter the webpages structure/content based on user's interactions (eg mouse clicks)

JavaScript is King on the Browser

- If web page needs to execute code on browser, you use JS
- But historically there were other options in the (not so long ago) past:
  - Java with Java Applets (practically dead)
  - Flash (still found in some old web pages)
  - Silverligth
  - Etc.
- Those were not natively supported by browser, and you had to install plugins to run them

But JavaScript is a badly designed language…

- When the most famous book is called “The Good Parts”, that tells you something…
- However, there are other languages that do transpile to JS, like TypeScript and Kotlin
  - But configure them would be too complex at this point, and out of scope of this course…
- … and WebAssembly might (hopefully) replace JS one day…
- But we will need to write not so many lines of JS in this course, so you will survive…

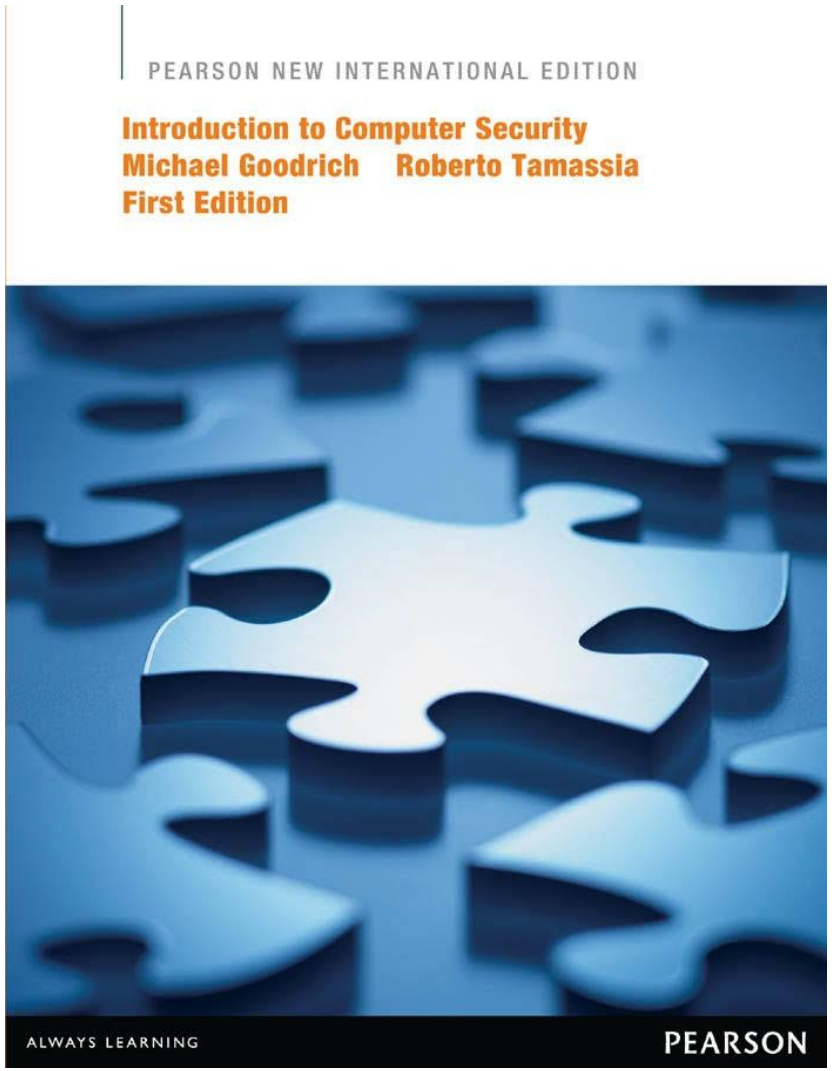
Videos

- https://www.destroyallsoftware.com/talks/wat
- https://www.youtube.com/watch?v=EtoMN_xi-AM

DEMO

- See *lessons/02/html/lesson02_example.html* in the Git repository
- Note: this demo is not related to security, but it will be the starting point for this week exercise

For Next Week



- Book pages: 25-30, 388-394, 399-400, 410-412
- Note: when I tell you to **study** some specific pages in the book, it would be good if you also *read* the other pages in the same chapter at least once
- Exercises for Lesson 02 on GitHub repository