Classical (Symmetric) Cryptography

Cryptography: terminology (1/2)

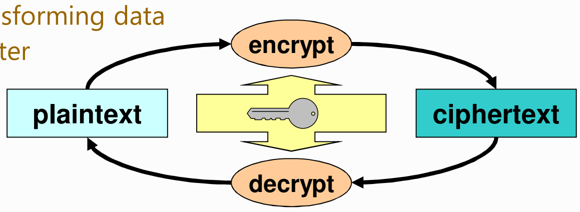
- Cryptography
  - Art or science of hidden writing
    - from Gr. *kryptós*, hidden + *graph*, r. of *graphein*, to write
  - It was initially used to maintain the confidentiality of information
  - Steganography
    - from Gr. *steganós*, hidden + *graph*, r. of *graphein*, to write
- Cryptanalysis
  - Art or science of breaking cryptographic systems or encrypted information
- Cryptology
  - Cryptography + cryptanalysis

Cryptography: terminology (2/2)

- Cipher
  - Specific cryptographic technique
- Cipher operation

  Encryption: plaintext (or cleartext) → ciphertext (or cryptogram)
  Decryption: ciphertext → plaintext

  Algorithm: way of transforming data
  Key: algorithm parameter



The players

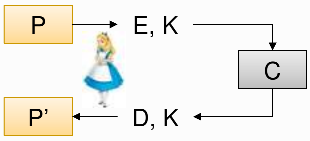
- Alice & Bob
  - The fundamental honest people
  - They represent two abstract interacting entities
- Carol, Dave, …
  - More honest entities for complex protocols
- Eve
  - Passive eavesdropper
- Mallory
  - Malicious attacker
- Trent
  - Trusted by all

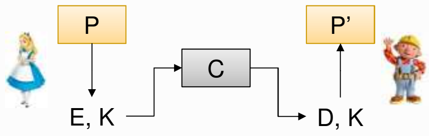
Who are Alice and Bob? (By Bruce Schneier)

Use cases

- ▷ Self-protection with key K
  - Alice encrypts plaintext P with key K
    - A: $C = \{P\}_K$
  - Alice decrypts cryptogram C with key K
    - A: $P' = \{C\}_K$
  - P' should be equal to P (requires checking)

- ▷ Secure communication with key K
  - Alice encrypts plaintext P with key K
    - A: $C = \{P\}_K$
  - Bob decrypts C with key K
    - B: $P' = \{C\}_K$
  - P' should be equal to P (requires checking)

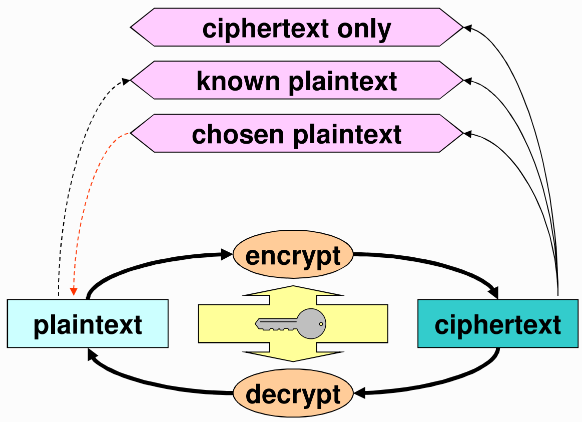




Cryptanalysis: goals

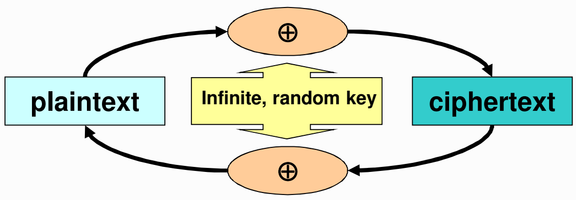
- ▷ Discover original plaintext
  - Which originated a given ciphertext
- ▷ Discover a cipher key
  - Allows the decryption of ciphertexts created with the same key
- ▷ Discover the cipher algorithm
  - Or an equivalent algorithm…
  - Usually algorithms are not secret, but there are exceptions
    - Lorenz, A5 (GSM), RC4 (WEP), Crypto-1 (Mifare)
    - Algorithms for DRM (Digital Rights Management)
  - Reverse engineering

Cryptanalysis attacks: approaches



Cryptography: Information-theoretic security

- Plaintext space
  - Set of all possible plaintext messages (M)
- Ciphertext space
  - Set of all possible ciphertext values (C)
- Key space
  - Set of all possible key values for a given algorithm (K)
- Perfect security
  - Given $c_j \in C$, $p(m_i, k_j) = p(m_i)$
  - $\#K \geq \#M$
  - Vernam cipher (one-time pad)
- The cipher cannot be broken
  - Even by adversaries with unlimited computing power



Cryptography: computational security

- ▷ The number of possible keys is finite
  - And much less than the number os possible messages
  - #K << #M
- ▷ Thus, security ultimately depends on the computing power of cryptanalysts go through all keys
  - Computations per time period
  - Storage capacity
  - Resistance time is mainly given by key length
- ▷ Provable security
  - The computational security can be demonstrated by comparing it with known hard problems

Key dimensions in perspective

- ▷ 2^{32} (4 Giga)
  - IPv4 address space
  - World population
  - Years for the Sun to become a white dwarf
- ▷ 2^{64}
  - Virtual address space of current CPU architectures
- ▷ 2^{128}
  - IPv6 address space
- ▷ 2^{166}
  - Earth atoms
- ▷ 2^{265}
  - Hydrogen atoms in the known universe
- ▷ 2^{1024} and beyond
  - Only cryptography uses them

Cryptanalysis attacks: approaches

- Brute force
  - Exhaustive search along the key space until finding a suitable key
  - Usually infeasible for a large key space
    - e.g. 2^{128} random keys (or keys with 128 bits)
    - Randomness is fundamental!
- Cleaver attacks
  - Reduce the search space to a smaller set of potential candidates

Cryptography: practical approaches (1/4)

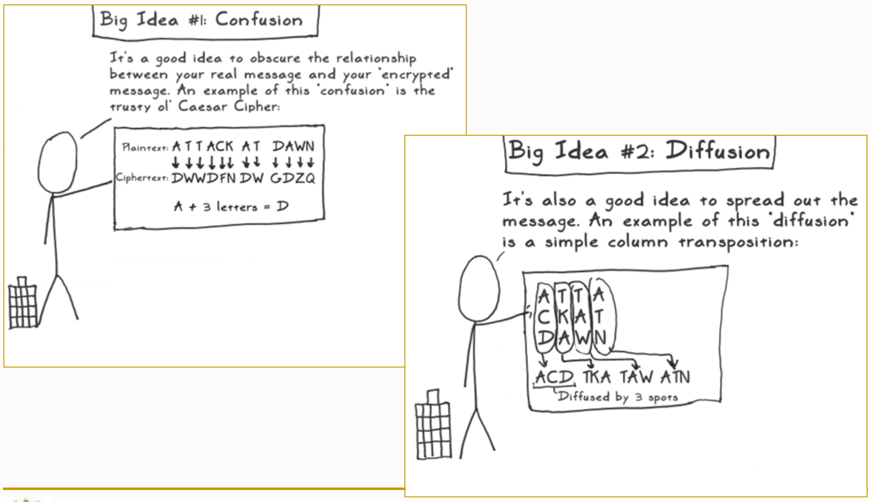
- Theoretical security vs. practical security
  - Expected use $\neq$ practical exploitation
  - Defective practices can introduce vulnerabilities
    - Example: reuse of keys
- Computational security
  - Computational complexity of break-in attacks
    - Using brute force
  - Security bounds:
    - Cost of cryptanalysis
    - Availability of cryptanalysis infra-structure
    - Lifetime of ciphertext

Cryptography: practical approaches (2/4)

- 5 Shannon criteria
  - The amount of offered secrecy
    - e.g. key length
  - Complexity of key selection
    - e.g. key generation, detection of weak keys
  - Implementation simplicity
  - Error propagation
    - Relevant in error-prone environments
    - e.g. noisy communication channels
  - Dimension of ciphertexts
    - Regarding the related plaintexts

Confusion & diffusion

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html



Cryptography: practical approaches (3/4)

- Confusion
  - Complex relationship between the key, plaintext and the ciphertext
    - Output bits (ciphertext) should depend on the input bits (plaintext + key) in a very complex way
- Diffusion
  - Plaintext statistics are dissipated in the ciphertext
    - If one plaintext bit toggles, then the ciphertext changes substantially, in an unpredictable or pseudorandom manner
  - Avalanche effect

What should be secret?

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

Cryptography: practical approaches (4/4)

- Always assume the worst case
  - Cryptanalysts know the algorithm
    - Security lies in the key
  - Cryptanalysts know/have many ciphertext samples produced with the same algorithm & key
    - Ciphertext is not secret!
  - Cryptanalysts partially know original plaintexts
    - As they have some idea of what they are looking for
    - Know-plaintext attacks
    - Chosen-plaintext attacks

Cryptographic robustness

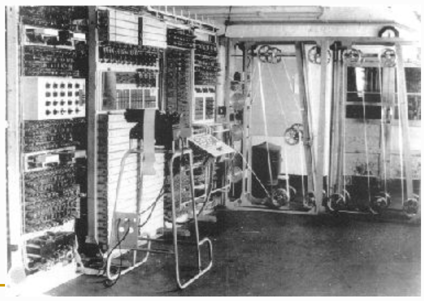
- The robustness of algorithms is their resistance to attacks
  - No one can evaluate it precisely
    - Only speculate or demonstrate using some other robustness assumptions
  - They are robust until someone breaks them
  - There are public guidelines with what should/must not be used
    - Sometimes antecipating future problems
- Algorithms with longer keys are probably stronger
  - And usually slower …
- Public algorithms w/o known attacks are probably stronger
  - More people looking for weaknesses

Cryptographic guidelines

- Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms, NIST Special Publication 800-175B Rev. 1, July 2019
- Cryptographic Storage Cheat Sheet, OWASP Cheat Sheets (last revision: 6/Jun/2020)
- Guidelines on cryptographic algorithms usage and key management, European Payments Council, EPC342-08 v9.0, 9/Mar/2020
- Algorithms, Key Size and Protocols Report, ECRYPT – Coordination & Support Action, Deliverable D5.4, H2020-ICT-2014 Project 645421, 28/Feb/2018

Ciphers: evolution of technology

- Manual
  - Simple transposition or substitution algorithms
- Mechanic
  - From XIX cent.
    - Enigma machine
    - M-209 Converter
  - More complex substitution algorithms
- Informatics
  - Appear with computers
  - Highly complex substitution algorithms
  - Mathematical algorithms



Ciphers: basic types (1/3)

- ▷ Transposition
  - Original cleartext is scrambled
    Onexcl raatre ilriad gctsm ilesb
  - Block permutations
    (13524) → **boklc pruem ttoai ns**

O	N	E	X	C	L
R	A	A	T	R	E
I	L	R	I	A	D
G	C	T	S	M	
I	L	E	S	B	

- ▷ Substitution
  - Each original symbol is replaced by another
    - Original symbols were letters, digits and punctuation
    - Actually they are blocks of bits
  - Substitution strategies
    - Mono-alphabetic (one→one)
    - Polyalphabetic (many one→one)
    - Homophonic (one→many)

Ciphers: basic types (2/3): Mono-alphabetic

- ▷ Use a single substitution alphabet
  - With #α elements
- ▷ Examples
  - Additive (translation)
    - crypto-symbol = (symbol + key) mod # α
    - symbol = (crypto-symbol – key) mod # α
    - Possible keys = #α
    - Caesar Cipher (ROT-x)
  - With sentence key
    ABCDEFGHIJKLMNOPQRSTUVWXYZ
    QRUVWXZSENTCKYABDFGHIJLMOP
    - Possible keys = # α ! → 26! ≈ 2^{88}
- ▷ Problems
  - Reproduce plaintext pattern
    - Individual characters, digrams, trigrams, etc.
  - Statistical analysis facilitates cryptanalysis
    - “The Gold Bug”, Edgar Alan Poe

```
53‡‡†305))6*;4826)4‡.)
4‡);806*;48†860))85;1‡
(;:‡*8†83(88)5*†;46(;8
8*96*?;8)*‡(;485);5*†2
:*‡(;4956*2(5*—4)88*;4
069285);)6†8)4‡‡;1(‡9;
48081;8:8‡1;48†85;4)48
5†528806*81(‡9;48;(88;
4(‡?34;48)4‡;161;:188;
‡?;
```

```
A good glass in the
bishop's hostel in the
devil's seat fifty-one
degrees and thirteen
minutes northeast and
by north main branch
seventh limb east side
shoot from the left eye
of the death's-head a
bee line from the tree
through the shot forty
feet out
```

Ciphers: basic types (3/3): Polyalphabetic

- Use N substitution alphabets
  - Periodical ciphers, with period N
- Example
  - Vigenère cipher
- Problems
  - Once known the period, are as easy to cryptanalyze as N mono-alphabetic ones
    - The period can be discovered using statistics
    - Kasiski method
      - Factoring of distances between equal ciphertext blocks
    - Coincidence index
      - Factoring of self-correlation offsets that yield higher coincidences

Vigenère cipher (or the Vigenère square)

	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

- Example of encryption of character <u>M</u> with key <u>S</u>, yielding cryptogram <u>E</u>
  - Decryption is the opposite, E and S yield M

Cryptanalysis of a Vigenère cryptogram: Example (1/2)

- Plaintext:
  Eles não sabem que o sonho é uma constante da vida
  tão concreta e definida como outra coisa qualquer,
  como esta pedra cinzenta em que me sento e descanso,
  como este ribeiro manso, em serenos sobressaltos
  como estes pinheiros altos
- Cipher with the Vigenère square and key "poema"

```
plaintext  elesnaosabemqueosonhoeumaconstantedavidataoconcretaedefinida
key        poemapoemapoemapoemapoemapoemapoemapoemapoemapoemapoemapoema
cryptogram tzienpcwmbtaugedgszhdsyyarcretpbxqdpjmpaiosoocqvqtpshqfxbmpa
```

- Kasiski test
  - With text above:

mpa	$20 = 2 \times\ 2 \times 5$
tp	$20 = 2 \times\ 2 \times 5$

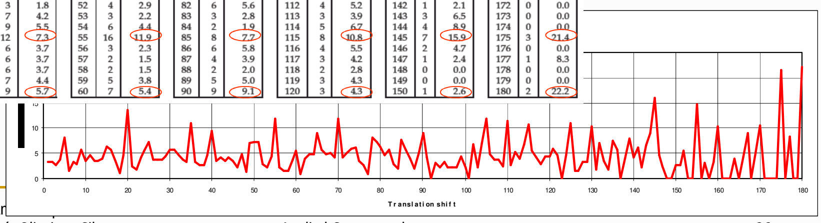
  - With the complete poem:

$175 = 5 \times\ 5 \times 7$	1
$105 = 3 \times\ 5 \times 7$	3
$35 = 5 \times 7$	1
$20 = 2 \times 2 \times 5$	4

Cryptanalysis of a Vigenère cryptogram: Example (2/2)

- Coincidence index (with full poem)

D	I	P (%)	D	I	P (%)	D	I	P (%)	D	I	P (%)	D	I	P (%)	D	I	P (%)
1	6	3.2	31	9	5.7	61	1	0.8	91	4	4.1	121	4	5.9	151	1	2.6
2	6	3.2	32	7	4.5	62	5	3.9	92	0	0.0	122	3	4.5	152	2	5.4
3	5	2.7	33	6	3.8	63	6	4.8	93	3	3.1	123	0	0.0	153	0	0.0
4	7	3.8	34	5	3.2	64	6	4.8	94	2	2.1	124	3	4.6	154	0	0.0
5	15	8.2	35	17	11.0	65	11	8.9	95	3	3.2	125	7	10.9	155	5	14.7
6	3	1.6	36	5	3.3	66	7	5.7	96	2	2.2	126	1	1.6	156	0	0.0
7	6	3.3	37	4	2.6	67	6	4.9	97	2	2.2	127	1	1.6	157	1	3.1
8	5	2.8	38	4	2.6	68	6	5.0	98	2	2.2	128	2	3.3	158	0	0.0
9	10	5.6	39	7	4.7	69	5	4.2	99	4	4.4	129	2	3.3	159	1	3.3
10	6	3.4	40	14	9.4	70	14	11.8	100	2	2.2	130	6	10.2	160	3	10.3
11	8	4.5	41	5	3.4	71	5	4.2	101	0	0.0	131	1	1.7	161	0	0.0
12	6	3.4	42	6	4.1	72	6	5.1	102	6	6.9	132	4	7.0	162	0	0.0
13	6	3.4	43	5	3.4	73	7	6.0	103	2	2.3	133	2	3.6	163	0	0.0
14	7	4.0	44	6	4.1	74	7	6.1	104	6	7.1	134	1	1.8	164	1	4.0
15	11	6.3	45	5	3.5	75	4	3.5	105	10	11.9	135	4	7.4	165	0	0.0
16	10	5.8	46	3	2.1	76	3	2.7	106	4	4.8	136	3	5.7	166	1	4.3
17	6	3.5	47	7	4.9	77	1	0.9	107	3	3.7	137	0	0.0	167	2	9.1
18	2	1.2	48	2	1.4	78	9	8.1	108	3	3.7	138	2	3.9	168	0	0.0
19	8	4.7	49	10	7.1	79	8	7.3	109	2	2.5	139	4	8.0	169	1	5.0
20	23	13.6	50	10	7.2	80	7	6.4	110	9	11.4	140	2	4.1	170	2	10.5
21	4	2.4	51	10	7.2	81	5	4.6	111	2	2.6	141	3	6.2	171	0	0.0
22	3	1.8	52	4	2.9	82	6	5.6	112	4	5.2	142	1	2.1	172	0	0.0
23	7	4.2	53	3	2.2	83	3	2.8	113	3	3.9	143	3	6.5	173	0	0.0
24	9	5.5	54	6	4.4	84	2	1.9	114	5	6.7	144	4	8.9	174	0	0.0
25	12	7.3	55	16	11.9	85	8	7.7	115	8	10.8	145	7	15.9	175	3	21.4
26	6	3.7	56	3	2.3	86	6	5.8	116	4	5.5	146	2	4.7	176	0	0.0
27	6	3.7	57	2	1.5	87	4	3.9	117	3	4.2	147	1	2.4	177	1	8.3
28	6	3.7	58	2	1.5	88	2	2.0	118	2	2.8	148	0	0.0	178	0	0.0
29	7	4.4	59	5	3.8	89	5	5.0	119	3	4.3	149	0	0.0	179	0	0.0
30	9	5.7	60	7	5.4	90	9	9.1	120	3	4.3	150	1	2.6	180	2	22.2



Rotor Machines

David J Morgan, www.flickr.com

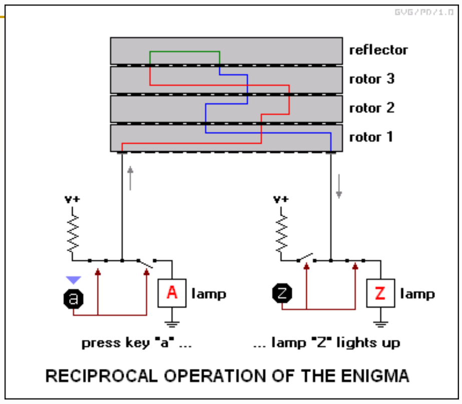
Rotor machines

- Rotor machines implement complex polyalphabetic ciphers
  - Each rotor contains a permutation
    - Same as a set of substitutions
  - The position of a rotor implements a substitution alphabet
  - Spinning of a rotor implements a polyalphabetic cipher
  - Stacking several rotors and spinning them at different times adds complexity to the cipher
- The cipher key is:
  - The set of rotors used
  - The relative order of the rotors
  - The position of the spinning ring
  - The original position of all the rotors
- Symmetrical (two-way) rotors allow decryption by "double encryption"
  - Using a reflection disk (half-rotor)

Sarah Witherby, www.flickr.com

Rotor machines


Andrew Magill, www.flickr.com

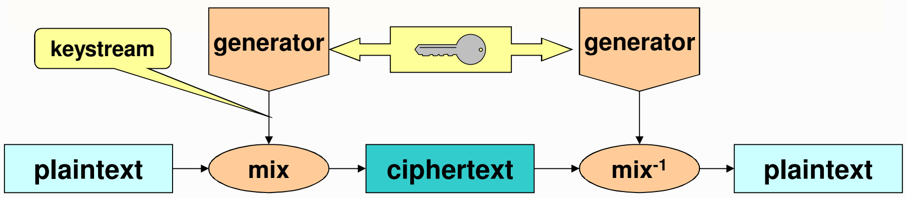


- Reciprocal operation with reflector
  - Sending operator types "A" as plaintext and gets "Z" as ciphertext, which is transmitted
  - Receiving operator types the received "Z" and gets the plaintext "A"
  - No letter could encrypt to itself !

Enigma

- WWII German rotor machine
  - Many models used
- Initially presented in 1919
  - Enigma I, with 3 rotors
- Several variants where used
  - With different number of rotors
  - With patch cord to permute alphabets
- Key settings distributed in codebooks

Stream ciphers



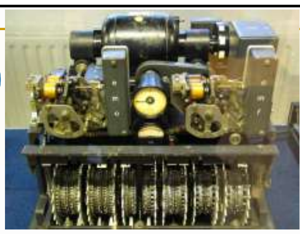
- Mixture of a keystream with the plaintext or ciphertext
  - Random keystream (Vernam's one-time pad)
  - Pseudo-random keystream (produced by generator using a finite key)
- Reversible mixture function
  - e.g. bitwise XOR
  - $C = P \oplus ks$ $\qquad$ $P = C \oplus ks$
- Polyalphabetic cipher
  - Each keystream symbol defines an alphabet

Stream ciphers

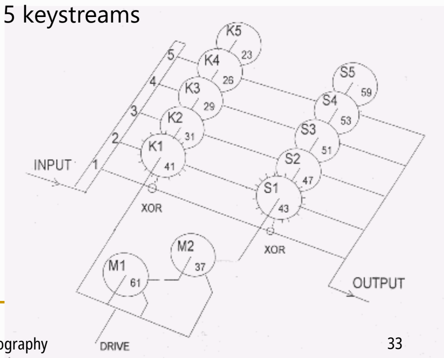
- Keystream may be infinite but with a finite period
  - The period depends on the generator
- Practical security issues
  - Each **keystream** should be used only **once**!
    - Otherwise, the sum of cryptograms yields the sum of plaintexts

      $C1 = P1 \oplus Ks$, $C2 = P2 \oplus Ks$ $\rightarrow$ $C1 \oplus C2 = P1 \oplus P2$
  - **Plaintext length** should be **smaller** than the **keystream period**
    - Total keystream exposure under know/chosen plaintext attacks
    - Keystream cycles help the cryptanalysts knowing plaintext samples
  - Integrity control is mandatory
    - No diffusion! (only confusion)
    - Ciphertexts can easily be changed deterministically

Lorenz (Tunny)

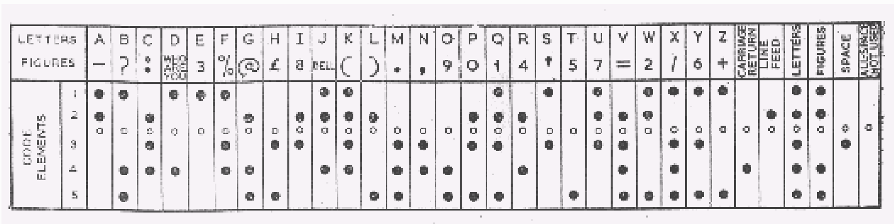


- ▷ 12-Rotor stream cipher
  - Used by the German high-command during the 2nd WW
  - Implements a stream cipher
    - Each 5-bit character is mixed with 5 keystreams
- ▷ Operation
  - 5 regularly stepped (χ) wheels
  - 5 irregularly stepped (ψ) wheels
    - All or none stepping
  - 2 motor wheels
    - For stepping the ψ wheels
  - Number of steps in all wheels is relatively prime



Cryptanalysis of Tunny in Bletchley Park

- ▷ They didn't know Lorenz internal structure
  - They observed one only at the end of the war
  - They knew about them because they could get 5-bit encrypted transmissions
    - Using the 32-symbol Baudot code instead of Morse code



Cryptanalysis of Tunny in Bletchley Park: The mistake (30 August 1941)

- A German operator had a long message (~4,000) to send
  - He set up his Lorenz and sent a 12 letter indicator (wheel setup) to the receiver
  - After ~4,000 characters had been keyed, by hand, the receiver said "send it again"
- The operator resets the machine to the same initial setup
  - Same keystream! Absolutely forbidden!
- The sender began to key in the message again (by hand)
  - But he typed a slightly different message!

$C = M \oplus Ks$

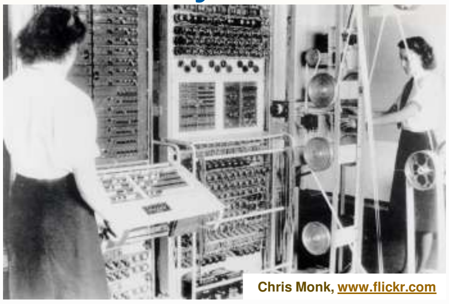
$C' = M' \oplus Ks \rightarrow M' = C \oplus C' \oplus M \rightarrow$ text variations

  - Know parts of the initial text M reveal the variations, M'

Cryptanalysis of Tunny in Bletchley Park: Breakthrough

- Messages began with SPRUCHNUMMER — "msg number"
  - The first time the operator typed S P R U C H N U M M E R
  - The second time he typed S P R U C H N R
  - Thus, immediately following the N the two texts were different!
- John Tiltman at Bletchley Park was able to fully decrypt both messages (called *Depths*) using an additive combination of them
  - The 2nd message was ~500 characters shorter than the first one
  - Tiltman managed to discover the correct message for the 1st ciphertext
- They got for the 1st time a long stretch of the Lorenz keystream
  - They did not know how the machine did it, …
  - … but they knew that this was what it was generating!

Cryptanalysis of Tunny in Bletchley Park: Colossus



Chris Monk, www.flickr.com

- The cipher structure was determined from the keystream
  - But deciphering it required knowing the initial position of rotors
- Germans started using numbers for the initial wheels' state
  - Bill Tutte invented the double-delta method for finding that state
  - The Colossus was built to apply the double-delta method
- Colossus
  - Design started in March 1943
  - The 1,500 valve Colossus Mark 1 was operational in January 1944
  - Colossus reduced the time to break Lorenz from weeks to hours

Modern Symmetric Cryptography

Modern ciphers: types

- Concerning operation
  - Block ciphers (mono-alphabetic)
  - Stream ciphers (poli-alphabetic)
- Concerning their key
  - Symmetric ciphers (secret key or shared key ciphers)
  - Asymmetric ciphers (or public key ciphers)
- Arrangements

	Block ciphers	Stream ciphers
Symmetric ciphers		
Asymmetric ciphers		

Symmetric ciphers

- Secret key
  - Shared by 2 or more peers
- Allow
  - Confidentiality among the key holders
  - Limited authentication of messages
    - When block ciphers are used
- Advantages
  - Performance (usually very efficient)
- Disadvantages
  - N interacting peers, pairwise secrecy ⇒ N x (N-1)/2 keys
- Problems
  - Key distribution

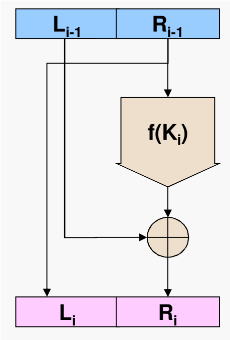
Symmetric block ciphers

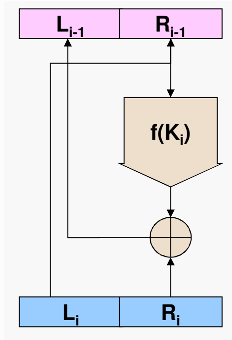
- Usual approaches
  - Large bit blocks for input, output and key
    - 64, 128, 256, etc.
  - Diffusion & confusion
    - Permutation, substitution, expansion, compression
    - Feistel networks, substitution-permutation networks
    - Iterations
    - Sub-keys (key schedules, round keys, etc.)
- Most common algorithms
  - DES (Data Enc. Stand.), D=64 K=56
  - IDEA (Int. Data Enc. Alg.), D=64 K=128
  - AES (Adv. Enc. Stand., aka Rijndael) D=128 K=128, 192, 256
  - Other (Blowfish, CAST, RC5, etc.)

Feistel networks

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

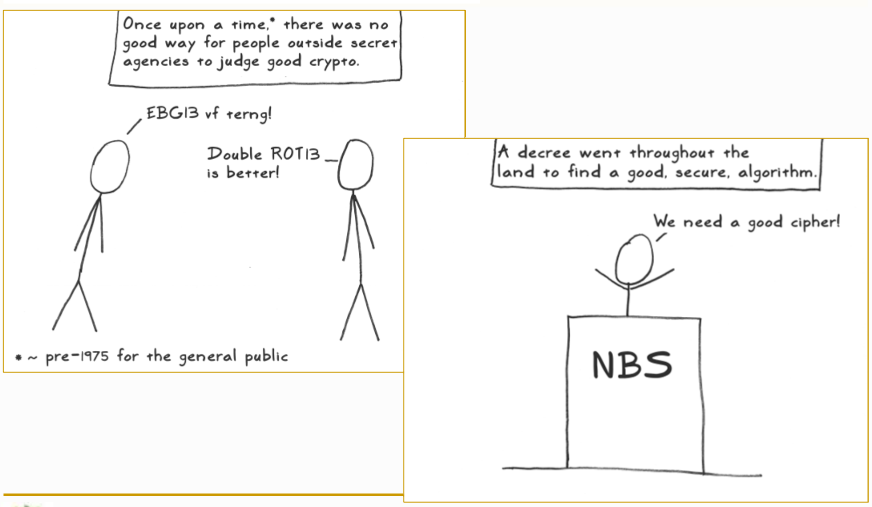
$R_{i-1} = L_i$
$L_{i-1} = R_i \oplus f(L_i, K_i)$





We need a good cipher!

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

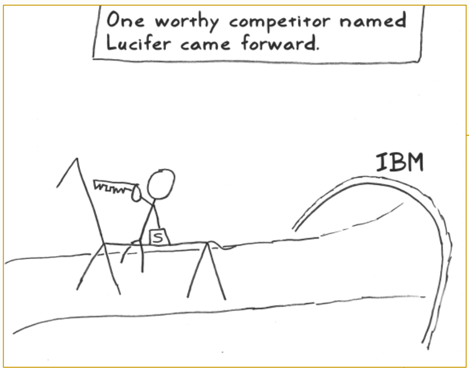


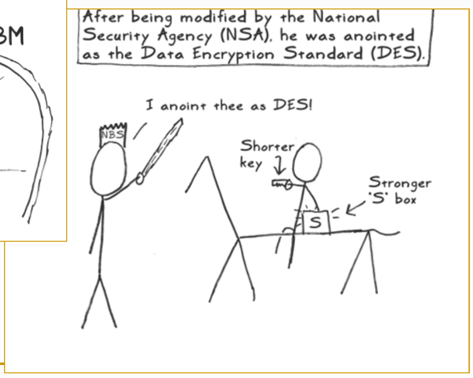
DES (Data Encryption Standard)

- 1970: the need of a standard cipher for civilians was identified
- 1972: NBS opens a contest for a new cipher, requiring:
  - The cryptographic algorithm must be secure to a high degree
  - Algorithm details described in an easy-to-understand language
  - The details of the algorithm must be publicly available
    - So that anyone could implement it in software or hardware
  - The security of the algorithm must depend on the key
    - Not on keeping the method itself (or part of it) secret
  - The method must be adaptable for use in many applications
  - Hardware implementations of the algorithm must be practical
    - i.e. not prohibitively expensive or extremely slow
  - The method must be efficient
  - Test and validation under real-life conditions
  - The algorithm should be exportable

Lucipher and DES

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

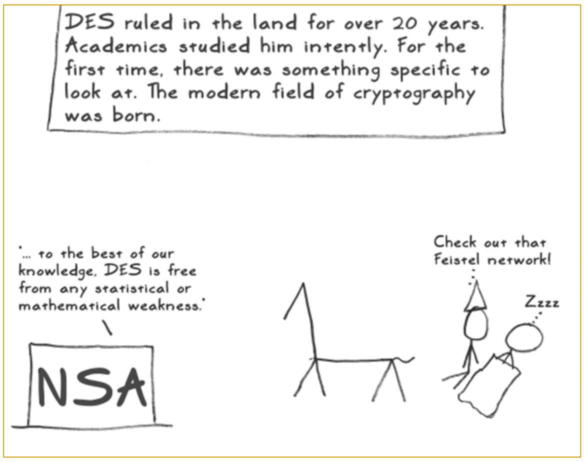




DES: proposal and adoption

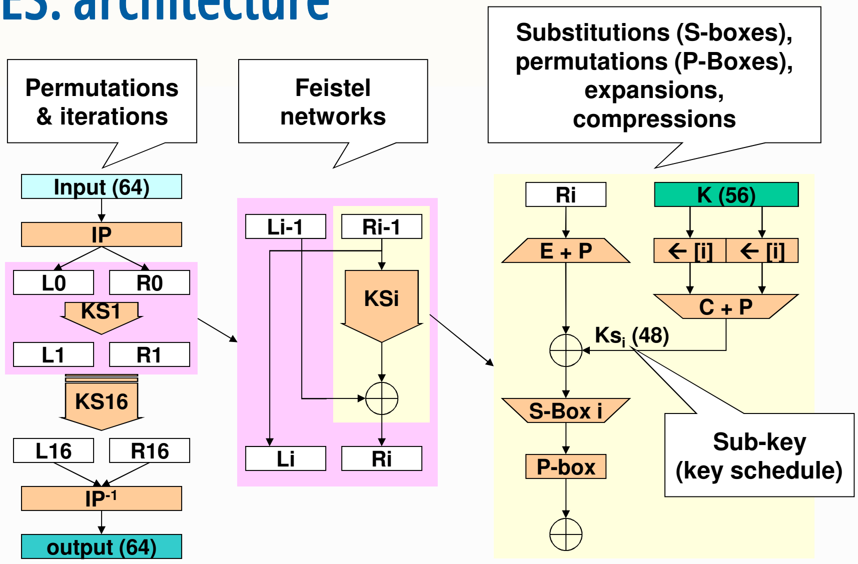
- 1974: new contest
  - Proposal based on Lucifer from IBM
  - 64-bit blocks
  - 56-bit keys
    - 48-bit subkeys (key schedules)
  - Diffusion & confusion
    - Feistel networks
    - Permutations, substitutions, expansions, compressions
    - 16 iterations
  - Several modes of operation
    - **ECB** (Electronic Code Book), **CBC** (Cypher Block Chaining)
    - **OFB** (Output Feedback), **CFB** (Cypher Feedback)
- 1976: adopted at US as a federal standard

DES as a milestone



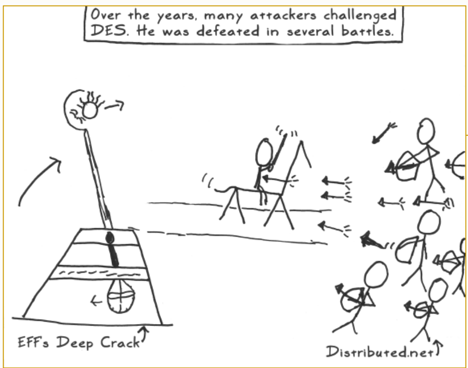
http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

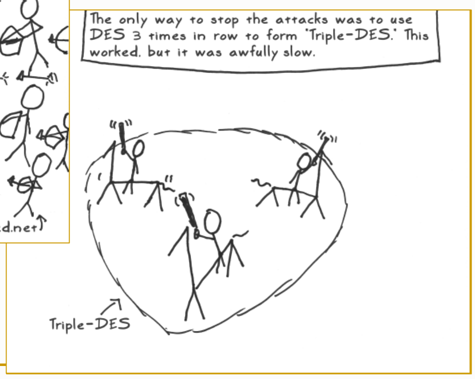
DES: architecture



DES security

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html



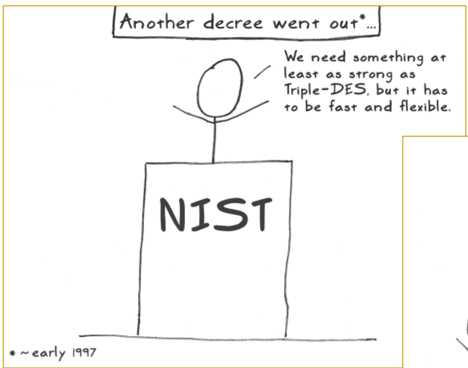


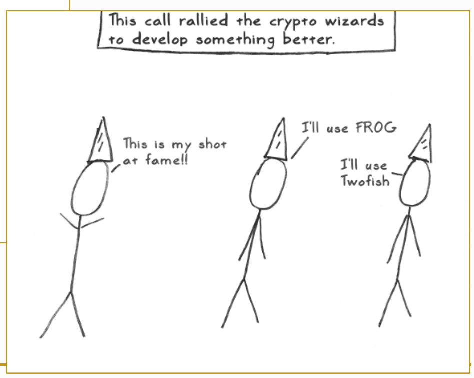
DES: offered security

- Key selection
  - Most 56-bit values are suitable
  - 4 weak, 12 semi-weak keys, 48 possibly weak keys
    - Equal key schedules (1, 2 or 4)
    - Easy to spot and avoid
- Known attacks
  - Exhaustive key space search
- Key length
  - 56 bits are actually too few
  - Exhaustive search is technically possible and economically interesting
- Multiple encryption
  - Double encryption
    - Theoretically not more secure
  - Triple DES (3DES)
    - With 2 or 3 keys
    - Equivalent key length of 112 or 168 bits
    - Secure but …slow!
  - DES-X
    - $K_1 \oplus DES(K_2) \oplus K_3$
    - Total key length = 64 + 56 + 64 = 184 bits

Replacement of DES (and DES variants)

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

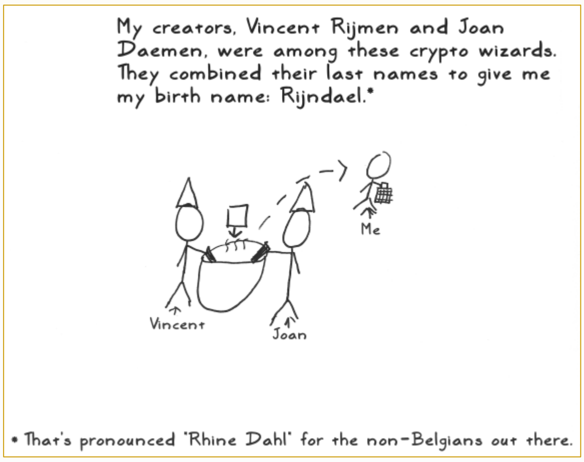




AES (Advanced Encryption Standard)

- 2/Jan/1997: Call for evaluation criteria
  - NIST publicly asked interested parties to propose a criteria to choose a DES successor
  - Many submissions received during 3 months
- 12/Sep/1997: Call for new algorithms
  - Block ciphers
  - 128-bit blocks
  - 128, 192, and 256-bit keys
  - Such ciphers were rare at the time of the call

Rijndael



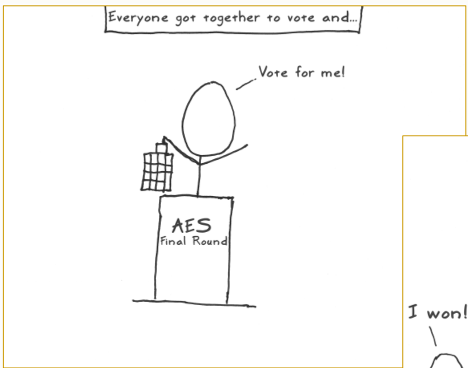
http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

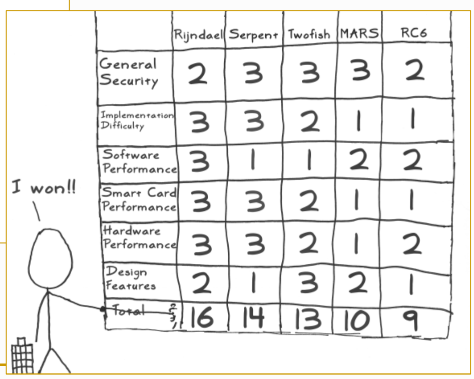
AES: evaluation rounds

- 1st round
  - 15 candidate algorithms were evaluated by the community
  - Conferences were organized for the evaluation
  - Cryptographic weakness were found
  - Performance issues were identified
    - In a variety of hardware
    - PCs, smart cards, hardware implementations
  - Constrained environment were evaluated
    - Limited memory smart cards, low gate count circuits, FPGAs
- Aug/1999: AES finalists announced
  - MARS, RC6, Rijndael, Serpent, and Twofish

Rijndael selection as AES

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html



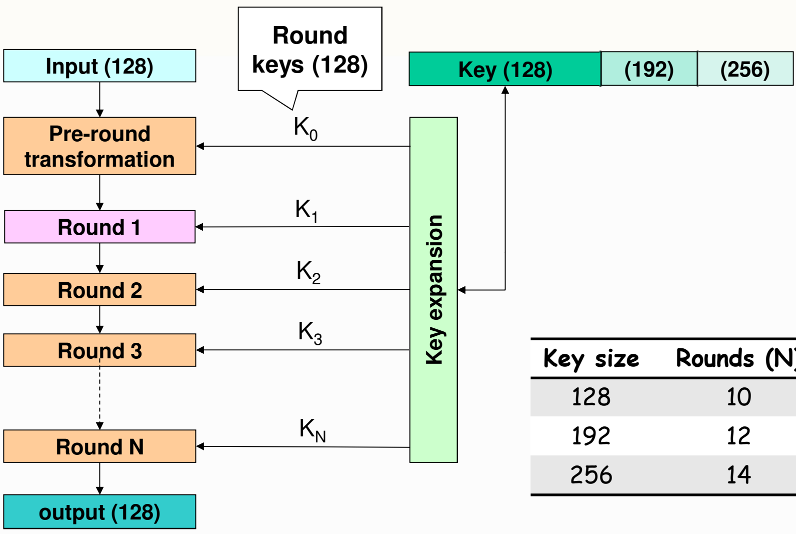


	Rijndael	Serpent	Twofish	MARS	RC6
General Security	2	3	3	3	2
Implementation Difficulty	3	3	2	1	1
Software Performance	3	1	1	2	2
Smart Card Performance	3	3	2	1	1
Hardware Performance	3	3	2	1	2
Design Features	2	1	3	2	1
Total	16	14	13	10	9

AES: evaluation rounds

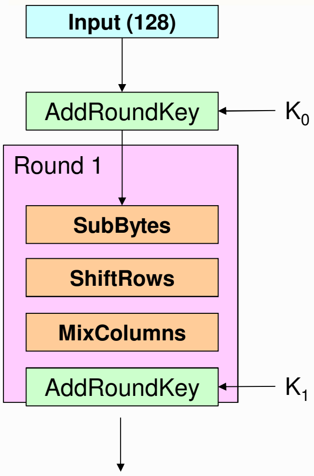
- 2nd round
  - The 5 finalists continued to be evaluated
  - In a final conference the proposal of each algorithm presented their advantage against the other
- 2/Oct/2000: AES algorithm was announced
  - Rijndael was selected
  - Proposed by Vincent Rijmen and Joan Daemen
  - Family of ciphers with different key and block sizes
- 26/Nov/2001: AES was approved by NIST
  - FIPS PUB 197
  - Subset of Rijndael (3 family members)
- Now part of the ISO/IEC 18033-3 standard

AES: architecture

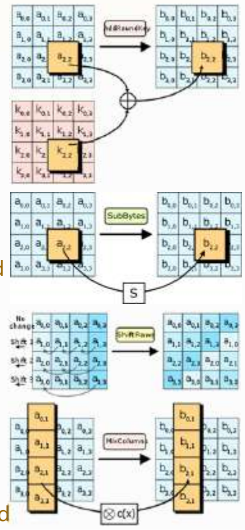


Key size	Rounds (N)
128	10
192	12
256	14

AES: architecture



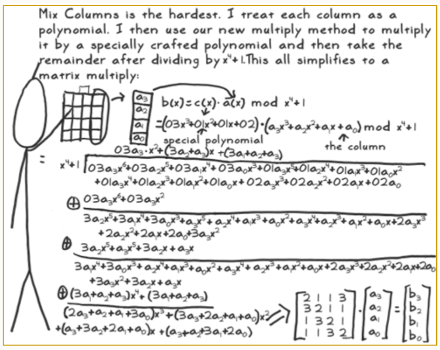
- AddRoundKey:
  - 128-bit XOR
  - Output is a 4x4 byte matrix
- SubBytes:
  - 256-element S-box
  - Each matrix bytes is substituted
- ShiftRows
  - Rows are rotated left
  - Byte shifts vary (0, 1, 2 & 3)
- MixColumns
  - Each column is transformed
  - Not performed in the last round

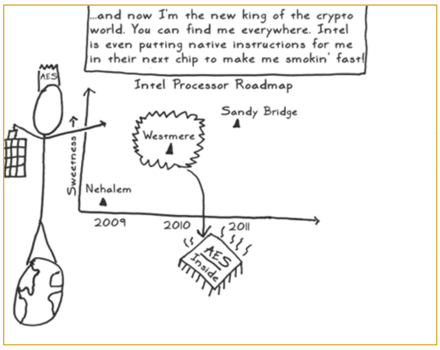


https://aescryptography.blogspot.com

AES complexity and speed-up

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html





AES in CPU instruction sets

▷ Intel AES New Instructions (AES-NI)

AESENC	Perform one round of an AES encryption flow
AESENCLAST	Perform the last round of an AES encryption flow
AESDEC	Perform one round of an AES decryption flow
AESDECLAST	Perform the last round of an AES decryption flow
AESKEYGENASSIST	Assist in AES round key generation
AESIMC	Assist in AES Inverse Mix Columns

▷ ARMv8 Cryptographic Extension

▷ … and other

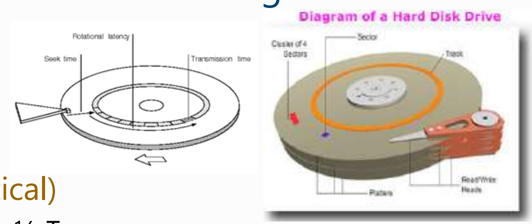
Stream ciphers

▷ Approaches
- Cryptographically secure pseudo-random generators (PRNG)
  - Using linear feedback shift registers (LFSR)
  - Using block ciphers
  - Other (families of functions, etc.)
- Usually not self-synchronized
- Usually without uniform random access
  - No immediate setup of generator's state for a given plaintext/cryptogram offset

▷ Most common algorithms
- A5/1 (US, Europe), A5/2 (GSM)
- RC4 (802.11 WEP/TKIP, etc.)
- E0 (Bluetooth BR/EDR)
- SEAL (w/ uniform random access)

Uniform random access

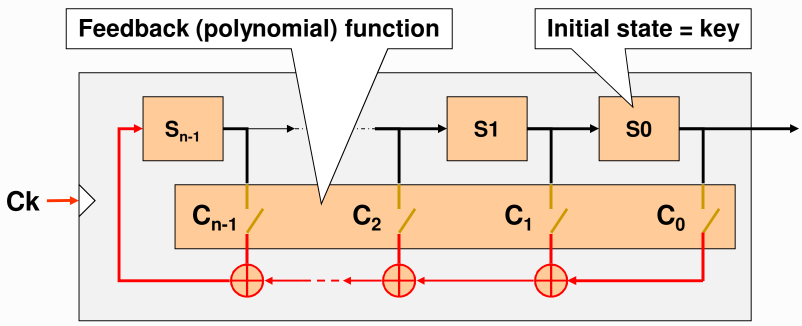
- Same time to reach and process any piece of information regardless of its storage location



- Uniform
  - Memory
  - Disks (magnetic, optical)
    - Average $T_{access} = T_{seek} + ½\ T_{revolution}$
- Non-uniform
  - Tapes (audio, video, computer)

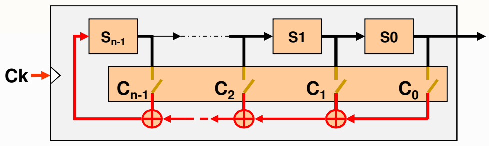
http://osr507doc.sco.com/en/PERFORM/disk_IO_mech.html
https://www.ict4u.net/components/backing-storage.php

Linear Feedback Shift Register (LFSR)



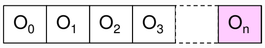
- 2^n-1 non-null sequences
  - If one of them has a 2^n-1 period length, then all have it
- Primitive feedback functions (primitive polynomials)
  - All non-null sequences have a 2^n–1 period length

Linear Feedback Shift Register (LFSR)



▷ Issue

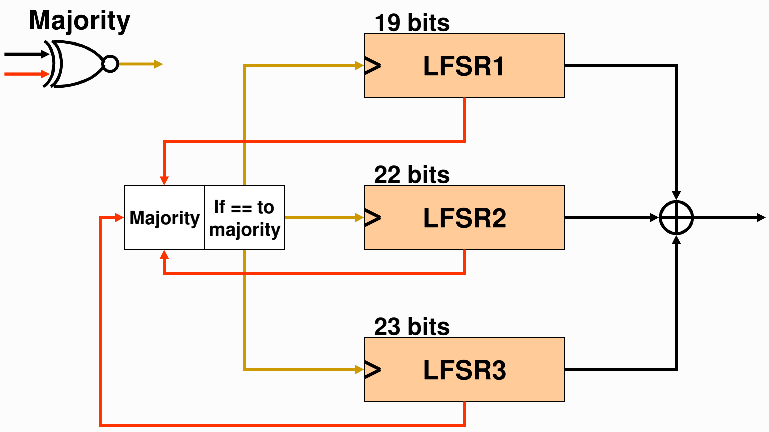
- If you know N consecutive bits of the output, you know the entire sequence ahead



$O_n = C_0O_0 + C_1O_1 + \ldots + C_{n-1}O_{n-1}$

- The output must be mixed with something else …

Generators using many LFSR: A5/1 (GSM)



Cipher modes

Deployment of (symmetric) block ciphers: Cipher modes

- Initially proposed for DES
  - ECB (Electronic Code Book)
  - CBC (Cipher Block Chaining)
  - OFB (Output Feeback)
  - CFB (Cipher Feedback)
- Can be used with other block ciphers
  - In principle ...
- Some other modes do exist
  - CTR (Counter Mode)
  - GCM (Galois/Counter Mode)

Block cipher modes: ECB and CBC

Electronic Code Book

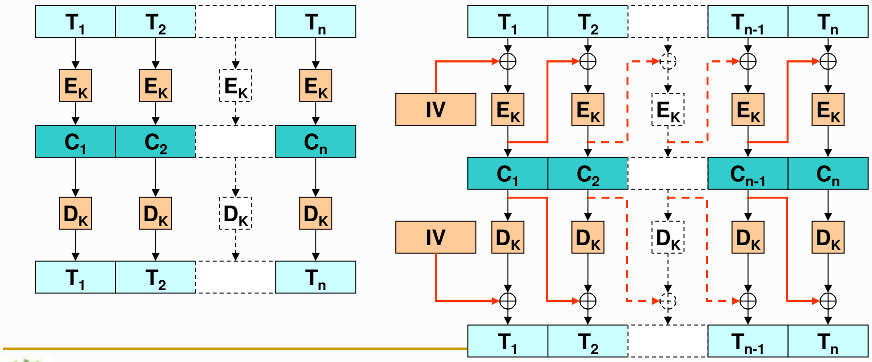
$C_i = E_K(T_i)$

$T_i = D_K(C_i)$

Cipher Block Chaining

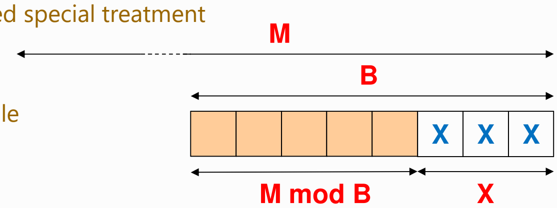
$C_i = E_K(T_i \oplus C_{i-1})$

$T_i = D_K(C_i) \oplus C_{i-1}$



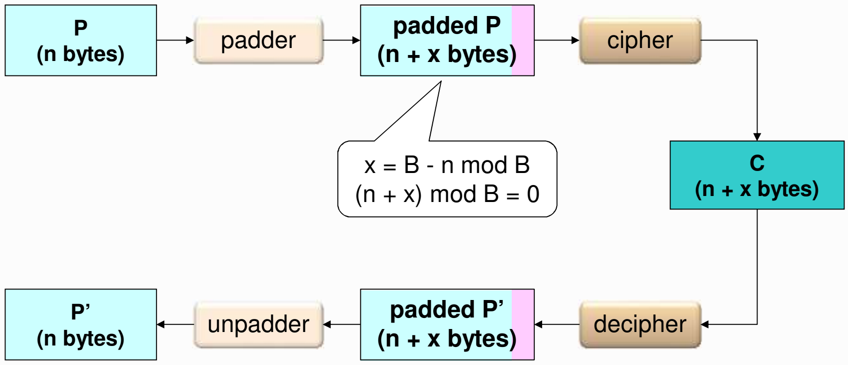
ECB/CBC cipher modes: Block alignment with padding

- Block cipher modes ECB and CBC require block-aligned inputs
  - Trailing sub-blocks need special treatment



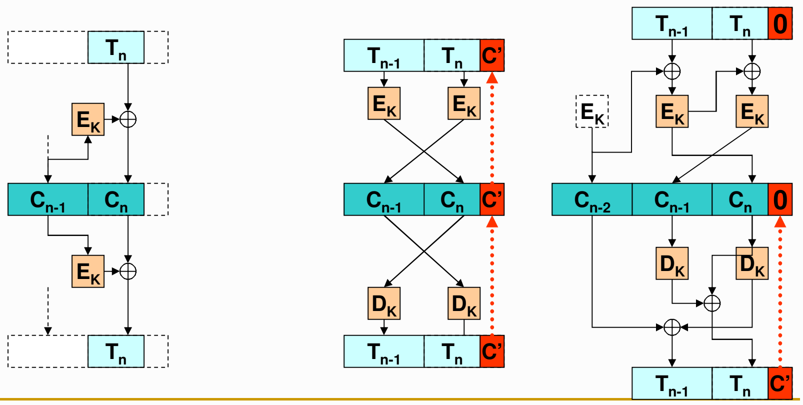
- Alternative 1: padding
  - Of last block, identifiable
  - Adds data
  - PKCS #7
    - X = B – (M mod B)
    - X extra bytes, with the value X
    - PKCS #5 (same as PKCS #7 with B = 8)
- Alternative 2: different processing for the last block
  - Adds implementation complexity

Padded block encryption / decryption



ECB/CBC cipher modes: Handling trailing sub-blocks

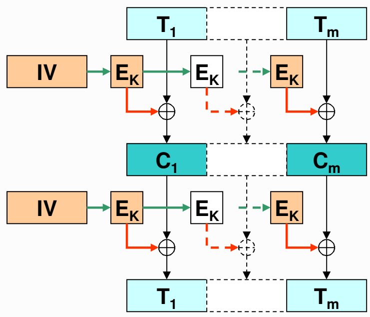
- ▷ Sort of stream cipher
- ▷ Ciphertext stealing

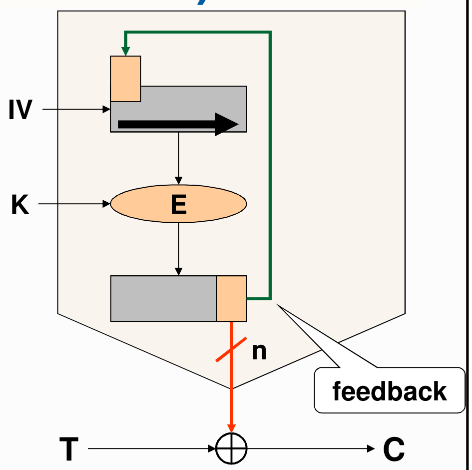


Stream cipher modes: n-bit OFB (Output Feedback)

$C_i = T_i \oplus E_K(S_i)$
$T_i = C_i \oplus E_K(S_i)$

$S_i = f(S_{i-1}, E_K(S_{i-1}))$
$S_0 = IV$

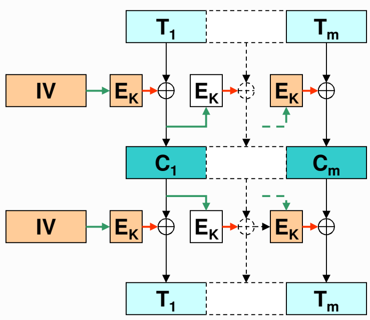


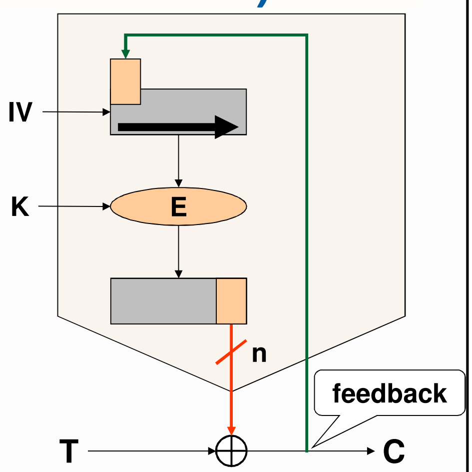


Stream cipher modes: n-bit CFB (Ciphertext Feedback)

$C_i = T_i \oplus E_K(S_i)$
$T_i = C_i \oplus E_K(S_i)$

$S_i = f(S_{i-1}, C_i)$
$S_0 = IV$

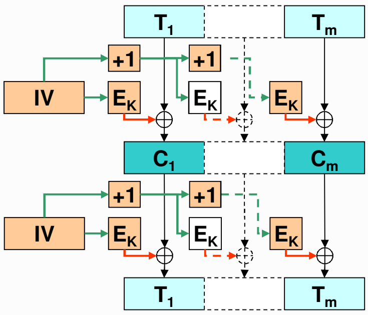


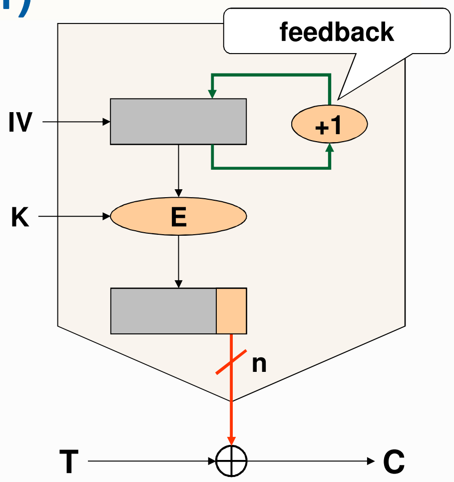


Stream cipher modes: n-bit CTR (Counter)

$C_i = T_i \oplus E_K(S_i)$
$T_i = C_i \oplus E_K(S_i)$

$S_i = S_{i-1}+1$
$S_0 = IV$

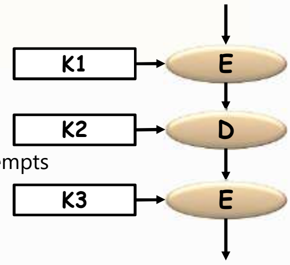




Cipher modes: Pros and cons

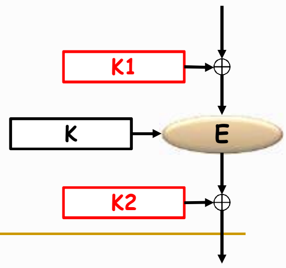
	Block		Stream		
	ECB	CBC	OFB	CFB	CTR
Input pattern hiding		✓	✓	✓	✓
Confusion on the cipher input		✓		✓	Secret counter
Same key for different messages	✓	✓	other IV	other IV	other IV
Tampering difficulty	✓	✓ (...)		✓	
Pre-processing			✓	...	✓
Parallel processing	✓	Decryption Only	w/ pre-processing	Decryption only	✓
Uniform random access					
Error propagation	Same block	Same block Next block		Some bits afterwards	
Capacity to recover from losses	Block Losses	Block Losses		✓	

Cipher modes:
Security reinforcement



▷ Multiple encryption

- Double encryption
  - Breakable with a meet-in-the-meddle attack in 2^{n+1} attempts
    - With 2 or more known plaintext blocks
    - Using 2^n blocks stored in memory …
  - Not secure enough (theoretically)
- Triple encryption (EDE)
  - $\mathbf{C_i = E_{K3}(D_{K2}(E_{K1}(T_i)))}$ $\mathbf{P_i = D_{K1}(E_{K2}(D_{K3}(C_i)))}$
  - Usually $\mathbf{K_1=K_3}$
  - If $\mathbf{K_1=K_2=K_3}$, then we get simple encryption



▷ Key whitening (DESX or DES-X)

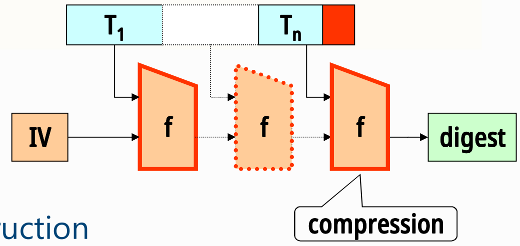
- Simple and efficient technique to add confusion
- $\mathbf{C_i = E_K(K_1 \oplus T_i) \oplus K_2}$
- $\mathbf{T_i = K_1 \oplus D_K(K_2 \oplus C_i)}$

Cryptographic Hashing

Digest functions

- Give a fixed-length value from a variable-length text
  - Sort of text “fingerprint”
- Produce very different values for similar texts
  - Cryptographic one-way hash functions
- Relevant properties:
  - Preimage resistance
    - Given a digest, it is infeasible to find an original text producing it
  - 2nd-preimage resistance
    - Given a text, it is infeasible to find another one with the same digest
  - Collision resistance
    - It is infeasible to find any two texts with the same digest
    - Birthday paradox

Digest functions: aproaches



- Merkle-Damgård construction
  - Iterative compression
  - Collision-resistent, one-way compression functions
  - Length padding (1, followed by zeros, followed by length)
- Sponge functions
  - **Absorption**: update a finite internal state (entropy pool) from a variable-length, padded input stream
  - **Squeezing**: produce an arbitrary-length output from the internal state

Digest functions: common algorithms

- MD5 (128 bits)
  - No longer secure! It's easy to find collisions!
  - Disclaimer: it can be used when collisions are not an issue
- SHA-1 (Secure Hash Algorithm, 160 bits)
  - Also no longer secure … (collisions found in 2017)
- RIPEMD (128 and 160)
- SHA-2, aka SHA-256 / SHA-348 / SHA-512
- SHA-3 (Keccak)

Rainbow tables

- ▷ We can invert a digest function with a table
  - For all possible input, we compute and store the digest
  - But the table size is given by the digest length
    - Not usually applicable
- ▷ Solution: rainbow tables
  - Trade space with time
  - Store only part of the outputs
    - For direct matching
  - Find for more matches using computation

Rainbow tables

- ▷ They are based on a reverse function R
  - Which is not the inverse of H
  - The goal of R is to produce a new input given a hashing result

$$I_0 \xrightarrow{H} O_1 \xrightarrow{R} I_1 \xrightarrow{H} O_2 \xrightarrow{R} I_2 \cdots\cdots \xrightarrow{R} I_k$$

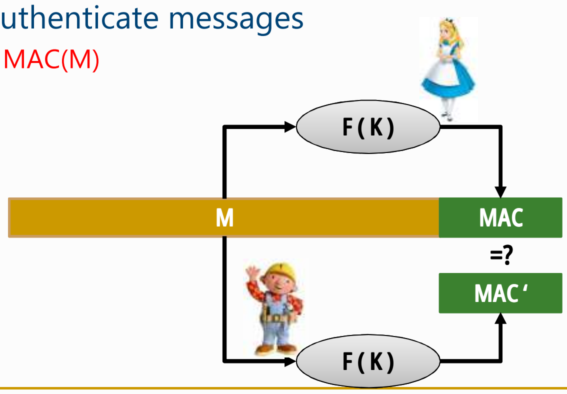
- ▷ R functions are likely to produce collisions
  - But we can use many different R functions
  - Collisions scan still occur
    - But will not create a problem unless occurring at the exact same column
    - And that case can be identified (and discarded) by identical outputs
- ▷ A table with m k-length rows can invert k×m hashes
  - At most
  - Only I_0 and I_k is stored per row

Rainbow tables: exploitation

- A set of **m** random inputs is generated
  - $I_0 = \{I_{0,1}, \dots I_{0,m}\}$
- A set of **m** k-length chain outputs is computed
  - $I_k = \{I_{k,1}, \dots I_{k,m}\}$
- Given a target o
  - Look for R(o) in I_k
  - If found in row r, compute chain from $I_{0,r}$
    - until finding i such that H(i) = o
  - If not found, compute o_r from o using H and R for each row r
    - and see if $o_r = I_{k,r}$
    - H and R are applied 1 to k times, using different R functions

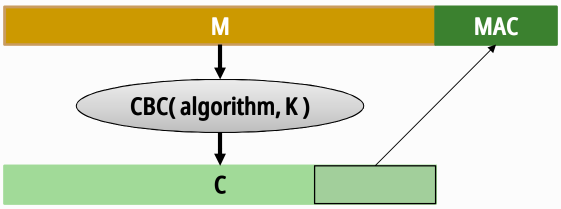
Message Authentication Codes (MAC)

- Hash, or digest, computed with a key
  - Only key holders can generate and validate the MAC
- Used to authenticate messages
  - M' = M | MAC(M)



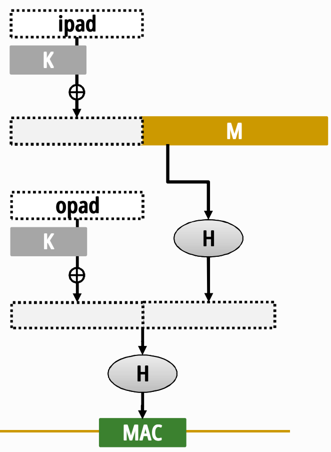
Message Authentication Codes (MAC): Approaches

- Encryption of an ordinary digest
  - Using, for instance, a symmetric block cipher
- Using encryption with feedback & error propagation
  - ANSI X9.9 (or DES-MAC) with DES CBC (64 bits)
  - CBC-MAC



Message Authentication Codes (MAC): Approaches

- Adding a key to the hashed data
  - Keyed-MD5 (128 bits)
    - MD5(K, keyfill, text, K, MD5fill)
  - HMAC (Hashed-based MAC)
    - Generic construction, uses a hash function H
    - Output length depends on H
    - HMAC-MD5, HMAC-SHA, etc.
    - H (K, opad, H (K, ipad, text))
    - ipad = 0x36 B times
    - opad = 0x5C B times

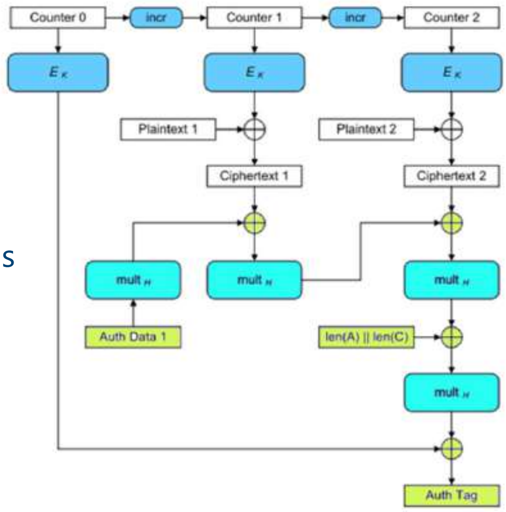


Authenticated encryption

- ▷ Encryption mixed with integrity control
  - Error propagation
  - Authentication tags
- ▷ Examples
  - GCM (Galois/Counter Mode)
  - CCM (Counter with CBC-MAC)

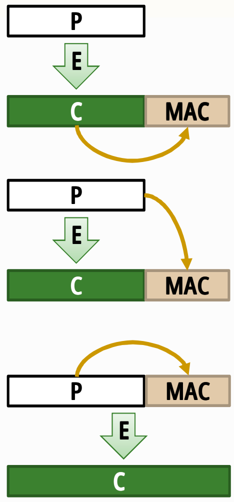
GCM

- ▷ CTR mode encryption
- ▷ Successive multiplications for integrity control
  - Multiplications in $GF(2^n)$
  - $H = E_k(0)$



Encryption + authentication

- Encrypt-then-MAC
  - MAC is computed from cryptogram
  - Should use two different keys
  - IPSec uses it
- Encrypt-and-MAC
  - MAC is computed from plaintext
  - MAC is not encrypted
  - SSH uses it
- MAC-then-Encrypt
  - MAC is computed from plaintext
  - MAC is encrypted
  - TLS uses it



Criptografia Aplicada, 2022/2023

RSA — and related subjects

The Magic Words are Squeamish Ossifrage

Guess who contributed a modest amount of computation time to this collaborative effort.

Security (spoiler)

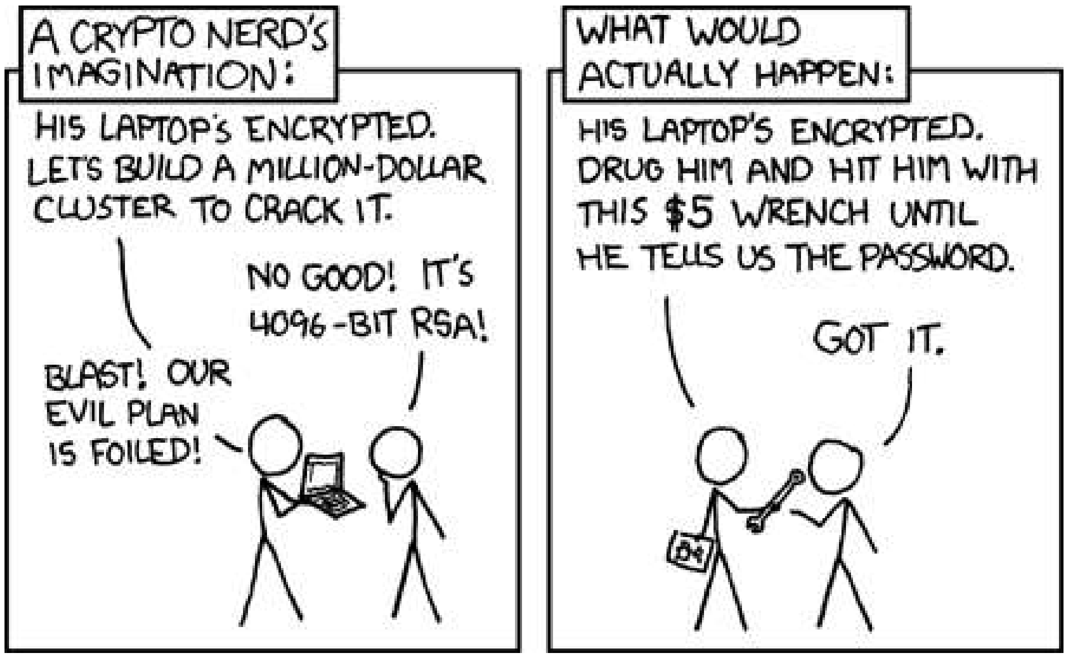
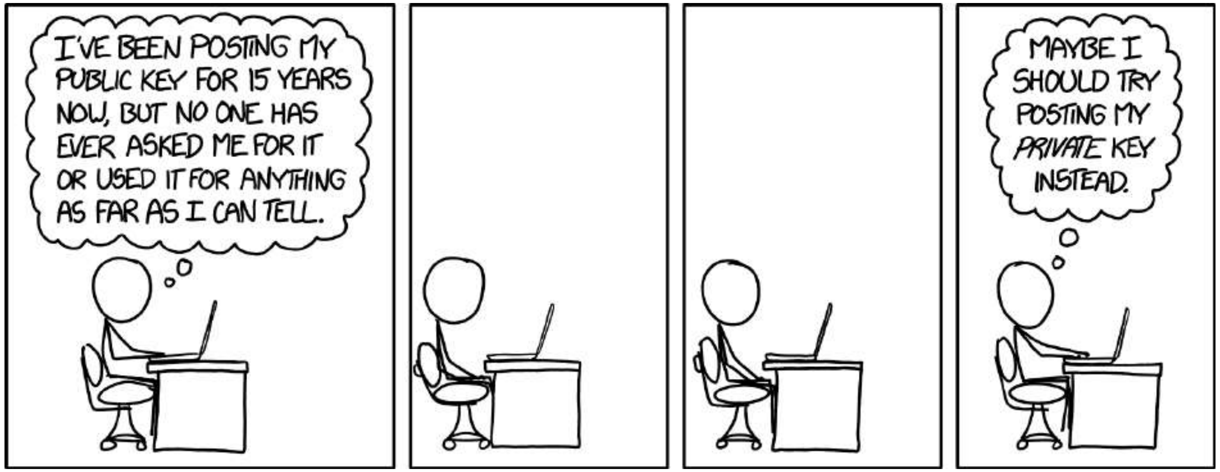


Table of Contents

Goals

- Public-key cryptography
- Sharing secrets
- Doing things without leaking information

Public Key (spoiler)



Means

- Number theory.
- In particular, modular arithmetic. Why? Because:
  - we will be performing computations with a finite set of integers (for example, there is no need to worry about round-off errors);
  - modular arithmetic can be done efficiently in almost all computing devices;
  - and last, but not least, because there exist many number theoretic theorems that have cryptographic applications.

Mathematics is the queen of the sciences and number theory is the queen of mathematics.
Carl Friedrich Gauss

The Theory of Numbers has always been regarded as one of the most obviously useless branches of Pure Mathematics. The accusation is one against which there is no valid defence; and it is never more just than when directed against the parts of the theory which are more particularly concerned with primes. A science is said to be useful if its development tends to accentuate the existing inequalities in the distribution of wealth, or more directly promotes the destruction of human life. The theory of prime numbers satisfies no such criteria. Those who pursue it will, if they are wise, make no attempt to justify their interest in a subject so trivial and so remote, and will console themselves with the thought that the greatest mathematicians of all ages have found in it a mysterious attraction impossible to resist.
Godfrey Harold Hardy

Programming languages you may use

- `C`, in particular the GNU MP library, also known as `libgmp`
- `C++`, using also the GNU MP library, but with classes and arithmetic operator overloading!
- `Python`
- `Java`, in particular the `BigInteger` class
- `pari-gp` (get it here), because it has everything we will need
- `SageMath` (get it here), because it has everything we will need and its interface uses the Python programming language (but its a big download)

Modular arithmetic

notation	meaning
$m \mid n$	m divides n.
$m \nmid n$	m does not divide n.
$n \equiv r \pmod{m}$	$m \mid (n-r)$, that is, as m divides $n-r$, n and r have the same remainder when divided by m.
$\lfloor x \rfloor$	floor function: largest integer not larger than x.
$n \bmod m$	(binary operator) remainder of n when divided by m (m is called the modulus, which we assume here to be a positive integer). Equal to $n - m\lfloor \frac{n}{m} \rfloor$. Note that $0 \leqslant r < m$. In `C`, `Python`, `Java`, and `pari-gp`, it can be computed using the % binary operator (applied to unsigned integers).
$\gcd(a, b)$	greatest common divisor of a and b.
$\operatorname{lcm}(a, b)$	least common multiple of a and b; equal to $ab/\gcd(a, b)$.
$\mathbb{Z}_m$	set of equivalence classes modulo m; slightly abusing the mathematical notation for equivalence classes, $\mathbb{Z}_m = \{\,0, 1, \ldots, m-1\,\}$.

Modular arithmetic examples

- $1 \mid 10$, $5 \mid 20$, $7 \mid 7$, $11 \mid 44$, $3 \nmid 5$
- $17 \equiv 7 \pmod{10}$, $27 \equiv 17 \pmod{10}$, $27 \equiv 7 \pmod{10}$
- $\lfloor 1.1 \rfloor = 1$, $\lfloor 7/3 \rfloor = 2$, $\lfloor -1.1 \rfloor = -2$
- $17 \bmod 6 = 5$, $7 \bmod 6 = 1$, $(17 \times 7) \bmod 6 = (5 \times 1) \bmod 6 = 5$
- $\gcd(15, 25) = 5$, $\gcd(7, 6) = 1$, when n is a positive integer, $\gcd(n, n + 1) = 1$
- $\operatorname{lcm}(15, 25) = 75$, $\operatorname{lcm}(7, 6) = 42$
- modulo m, the set of the integers — $\mathbb{Z}$ — is partitioned into m equivalence classes; we can choose as representative for each equivalence class an integer from the set $\mathbb{Z}_m$; for example, for $m = 5$, we have

equivalence class with representative 0: $\ldots, -5, \underline{0}, 5, 10, \ldots$
equivalence class with representative 1: $\ldots, -4, \underline{1}, 6, 11, \ldots$
equivalence class with representative 2: $\ldots, -3, \underline{2}, 7, 12, \ldots$
equivalence class with representative 3: $\ldots, -2, \underline{3}, 8, 13, \ldots$
equivalence class with representative 4: $\ldots, -1, \underline{4}, 9, 14, \ldots$

The binary $\bmod$ operator we defined in the previous slide computes this representative.

More modular arithmetic examples

Tables for addition (on the left) and multiplication (on the right) modulo 7.

$+$ $a\backslash b$	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

$\times$ $a\backslash b$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

- All elements of $\mathbb{Z}_7$ have a symmetric value; given any a it is also possible to find a b, which is unique, such that $a + b \equiv 0 \pmod{m}$. This is so in general.
- In this case all non-zero elements of $\mathbb{Z}_7$ have inverses. However, this is **not** general. An element a of $\mathbb{Z}_m$ has an inverse if and only if $\gcd(a, m) = 1$. The inverse of a, if it exists, is the (unique in $\mathbb{Z}_m$) b such that $ab \equiv 1 \pmod{m}$.

Modular arithmetic in C

- Addition, for small integers:

```
long add_mod(long a,long b,long m)
{ // assuming that 0 <= a,b < m, return (a+b) mod m
  long r = a + b;
  if(r >= m)
    r -= m;
  return r;
}
```

- Addition, for arbitrary precision integers (using the GNU MP library):

```
#include <gmp.h>
void add_mod(mpz_t r,mpz_t a,mpz_t b,mpz_t m)
{ // assuming that 0 <= a,b < m, compute r = (a+b) mod m
  mpz_add(r,a,b); // r = a+b
  if(mpz_cmp(r,m) >= 0)
    mpz_sub(r,r,m); // r -= m
}
```

Modular arithmetic exercises

Use a program (and perhaps brute force) to compute:

- $(1122334455 \times 6677889900) \bmod 349335433$
- $3^{-1} \bmod 7$ (this one does not require a program but do it anyway, it can be used to check if your program is working properly)
- $4^{-1} \bmod 7$ (neither does this one)
- $3^{-1} \bmod 10$ (neither does this one)
- $271828^{-1} \bmod 314159$ (just to warm up)
- $271828183^{-1} \bmod 314159265$ (now we're cooking!)
- $2718281828459^{-1} \bmod 3141592653590$ (can you handle this one?)
- $27182818284590452353602875^{-1} \bmod 31415926535897932384626434$ (is the teacher sane?)

Fast modular multiplication

A modular multiplication requires a remainder operation, which is a slow operation if the modulus is a general integer. For example, contemporary processors can multiply two 64-bit integers, producing a 128-bit result, with a latency of 3 or 4 clock cycles. But, dividing a 128-bit integer by a 64-bit integer, producing a 64-bit quotient and a 64-bit remainder, is considerably slower (tens of clock cycles). [For more information about how many clock cycles elementary arithmetic operations take on Intel/AMD processors, take a look at Agner Fog's instruction tables.]

If the modulus is a power of two, say 2^n, the remainder operation is very fast; the remainder is just the last n bits of the number being remaindered. In 1985, Peter Montgomery came up with a beautiful way to explore this to efficiently perform general remaindering operations without performing an expensive division.

Homework: Read Peter's paper Modular Multiplication Without Trial Division. You can get extra information by searching the internet for "Montgomery modular multiplication".

The greatest common divisor

- Let p_k be the k-th prime number, so that $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, and so on.
- Each positive integer can the factored into prime factors in a unique way (this is the fundamental theorem of arithmetic).
- Let $a = \prod_{k=1}^{\infty} p_k^{a_k}$, where a_k is the number of times p_k divides a. Since a is a finite number, almost all of the a_k values will be zero.
- Likewise of b, let $b = \prod_{k=1}^{\infty} p_k^{b_k}$.
- Then,

$$\gcd(a, b) = \prod_{k=1}^{\infty} p_k^{\min(a_k, b_k)}$$

  and

$$\operatorname{lcm}(a, b) = \prod_{k=1}^{\infty} p_k^{\max(a_k, b_k)}$$

- If $\gcd(a, b) = 1$ then a and b are said to be relatively prime (or coprime).
- The greatest common divison can be generalized to polynomials with integer coefficients!

The greatest common divisor (algorithm)

Assume that $a \geqslant 0$ and that $b \geqslant 0$. Then:

- $\gcd(a,b) = \gcd(b,a)$, and so $\gcd(a,b) = \gcd\big(\max(a,b), \min(a,b)\big)$. Thus, by exchanging a with b if necessary, we may assume that $a \geqslant b$.
- as any positive integer divides 0 we have $\gcd(a,0) = a$ for $a > 0$. The mathematicians say that $\gcd(0,0) = 0$, and so we can say that $\gcd(a,0) = a$ as long as $a \geqslant 0$.
- If $a \geqslant b$ then $\gcd(a,b) = \gcd(a-b,b)$. We can keep subtracting b from (the updated) a until it becomes smaller than b, and so $\gcd(a,b) = \gcd(a \bmod b, b) = \gcd(b, a \bmod b)$.

These observations give rise to the following so-called Euclid's algorithm (coded in `C`, but it can easily be translated to another programming language):

```c
long gcd(long a,long b)
{
  while(b != 0) { long c = a % b; a = b; b = c; } return a;
}
```

The GNU MP library has a function, `mpz_gcd`, for this; `pari-gp` does this with the `gcd` function.

The greatest common divisor (example)

Goal: to compute $\gcd(273, 715)$.

- Step 1: $\gcd(273, 715) = \gcd(715, 273)$.
- Step 2: $\gcd(715, 273) = \gcd(715 - 2 \times 273, 273) = \gcd(169, 273)$.
- Step 3: $\gcd(169, 273) = \gcd(273, 169) = \gcd(273-169, 169) = \gcd(104, 169)$.
- Step 4: $\gcd(104, 169) = \gcd(169, 104) = \gcd(169-104, 104) = \gcd(65, 104)$.
- Step 5: $\gcd(65, 104) = \gcd(104, 65) = \gcd(104 - 65, 65) = \gcd(39, 65)$.
- Step 6: $\gcd(39, 65) = \gcd(65, 39) = \gcd(65 - 39, 39) = \gcd(26, 39)$.
- Step 7: $\gcd(26, 39) = \gcd(39, 26) = \gcd(39 - 26, 26) = \gcd(13, 26)$.
- Step 8: $\gcd(13, 26) = \gcd(26, 13) = \gcd(26 - 2 \times 13, 13) = \gcd(0, 13)$.
- Step 9: $\gcd(0, 13) = \gcd(13, 0) = 13$.

It is known that the computational complexity of computing $\gcd(a, b)$ is $\mathcal{O}\big(\log \max(a, b)\big)$.
Compute $\gcd(15380990401719993308, 15052132919125944821)$.

The extended Euclid's algorithm

- The Euclid's algorithm starts a sequence with a and b and proceeds by doing modular reductions on consecutive terms of the sequence until zero is reached.
- But it is possible to do more!
- Let the sequence begin with $x_0 = a$ and $x_1 = b$. At any time, let $x_k = s_k a + t_k b$. So, $s_0 = t_1 = 1$, and $s_1 = t_0 = 0$.
- The next term of the sequence is given by $x_k = x_{k-2} \bmod x_{k-1}$. Let $q_k = \left\lfloor \frac{x_{k-2}}{x_{k-1}} \right\rfloor$. Then,

$$x_k = x_{k-2} - q_k x_{k-1}, \qquad s_k = s_{k-2} - q_k s_{k-1}, \quad \text{and} \quad t_k = t_{k-2} - q_k t_{k-1}.$$

- We have to stop when $x_k = 0$, at which time $\gcd(a, b) = x_{k-1}$. But here we know more:

$$x_{k-1} = s_{k-1} a + t_{k-1} b.$$

If $\gcd(a, b) = 1$ then $x_{k-1} = 1$, and this formula allows us to compute easily

$$a^{-1} \bmod b = s_{k-1} \bmod b \qquad \text{and} \qquad b^{-1} \bmod a = t_{k-1} \bmod a.$$

The extended Euclid's algorithm (example)

Goal: apply the extended Euclid's algorithm to compute $\gcd(77, 54)$.

- The following table illustrates the computations done by the extended Euclid's algorithm.

k	x_k	q_k	s_k	t_k
0	77		1	0
1	54		0	1
2	23	1	1	−1
3	8	2	−2	3
4	7	2	5	−7
5	1	1	−7	10
6	0	7	54	−77

- Because $x_6 = 0$, the information we seek corresponds to the row with $k = 5$. We have $\gcd(77, 54) = 1$, $77^{-1} \bmod 54 = -7 \bmod 54 = 47$, and $54^{-1} \bmod 77 = 10$.

The GNU MP library has a function, `mpz_gcdext`, for this; `pari-gp` also has a function, `gcdext`, for this. Let $a = 830150497265848419$ and $b = 472332647410202896$. Compute $a^{-1} \bmod b$ and $b^{-1} \bmod a$.

Linear maps

- When working modulo m it suffices to work with integers in the range $0, 1, \ldots, m-1$, i.e., it suffices to work with $\mathbb{Z}_m$.
- Let

$$f(x; m, a) = (ax) \bmod m$$

be the linear map $x \mapsto (ax) \bmod m$ from $\mathbb{Z}_m$ into itself.

- Recall that a function $f(x)$ is said to be linear if $f(\alpha x + \beta y) = \alpha f(x) + \beta f(y)$ for all α, β, x, and y.
- For example, for $m = 4$ the linear map with $a = 2$ (on the left) is **not** invertible, but the linear map with $a = 3$ (on the right) is invertible.

map for $m = 4$ and $a = 2$
$0 \mapsto 0$
$1 \mapsto 2$
$2 \mapsto 0$
$3 \mapsto 2$

map for $m = 4$ and $a = 3$
$0 \mapsto 0$
$1 \mapsto 3$
$2 \mapsto 2$
$3 \mapsto 1$

Linear maps (continuation)

- Why are we interested in inverting the map? Because the map **scrambles** the elements of $\mathbb{Z}_m$ and we may be interested in unscrambling them (think in cryptographic terms).
- So, what is the inverse map?
- It turns out that the inverse map, if it exists, is also a linear map.
- More specifically, the inverse map of $f(x, m, a \bmod m)$ is $f(x, m, a^{-1} \bmod m)$, where $a^{-1} \bmod m$ is the modular inverse of $a \bmod m$. Indeed, if $y = f(x; m, a) = ax \bmod m$ then $x = a^{-1}y \bmod m$.
- Since the modular inverse of a modulo m only exists when $\gcd(a, m) = 1$ the linear map is invertible if and only if $\gcd(a, m) = 1$.
- Keep in mind that we wish to devise a way to encrypt information by providing public data to do so (in this case it would be m and a).
- Alas, this way of scrambling information is very easy to unscramble, so useless from a cryptography point of view.
- Modular multiplication scrambles the information but it is easy to undo if we known m and a. What about modular exponentiation?

Linear maps (a failed cryptosystem)

The Merkle-Hellman knapsack cryptosystem keeps the following information secret:

- a set $W = \{ w_1, w_2, \ldots, w_n \}$ of n positive integers, such that w_k is a super-increasing sequence, i.e., $w_k > \sum_{i=1}^{k-1} w_i$ for $2 \leqslant k \leqslant n$,
- a modulus m such that $m > \sum_{i=1}^{n} w_i$,
- a scrambling integer a such that $\gcd(a, m) = 1$,

and publishes the following information:

- set $W' = \{w'_1, w'_2, \ldots, w'_n\}$, where $w'_i = (a w_i) \bmod m$, for $1 \leqslant i \leqslant n$.

Actually, it is **much** better to publish a random permutation of W'. (Homework: why?). To send a message composed by the n bits α_k, $1 \leqslant k \leqslant n$, compute and send

$$C = \sum_{k=1}^{n} \alpha_k w'_k.$$

This is a hard knapsack problem (in this case a subset sum problem). To decipher transform it into a trivial knapsack problem by computing $a^{-1} C \bmod m$, which is equal to $\sum_{k=1}^{n} \alpha_k w_k$ and so can be solved by a greedy algorithm.

Linear maps (Merkle-Hellman knapsack example)

The following example shows the Merkle-Hellman cryptosystem in action.

- Secret data: $\boldsymbol{W} = \{\, \mathbf{1, 3, 5, 12, 22, 47} \,\}$, $\boldsymbol{m} = \mathbf{100}$, and $\boldsymbol{a} = \mathbf{13}$.
- Public data: $\boldsymbol{W'} = \{\, \mathbf{13, 39, 65, 56, 86, 11} \,\}$,
- Unencrypted message to be sent: $\boldsymbol{A} = \{\, \mathbf{0, 0, 1, 1, 0, 1} \,\}$.
- Encrypted message sent: $\boldsymbol{C} = \mathbf{0{\times}13{+}0{\times}39{+}1{\times}65{+}1{\times}56{+}0{\times}86{+}1{\times}11 = 132}$.
- To decrypt compute $\mathbf{132 \times 13^{-1} \bmod 100 = 32 \times 77 \bmod 100 = 64}$ and then reason as follows [greedy algorithm for the easy subset sum problem]:

  1. $\mathbf{47}$ must be used to form the sum because $\mathbf{64 > 47}$. Hence $\boldsymbol{\alpha_6} = \mathbf{1}$. The rest of the sum is $\mathbf{64 - 47 = 17}$.
  2. $\mathbf{22}$ cannot be used to form the sum because $\mathbf{17 < 22}$. Hence $\boldsymbol{\alpha_5} = \mathbf{0}$.
  3. $\mathbf{12}$ must be used to form the sum because $\mathbf{17 > 12}$. Hence $\boldsymbol{\alpha_4} = \mathbf{1}$. The rest of the sum is $\mathbf{17 - 12 = 5}$.
  4. As so on. In this particular case, the next iteration finishes the deciphering process.

Fermat's little theorem

- The elements of $\mathbb{Z}_m$ that have an inverse are called the **units** of $\mathbb{Z}_m$. The set containing all these units is denoted by $\mathbb{Z}_m^*$. When m is a prime number, $\mathbb{Z}_m^* = \{\,1, 2, \ldots, m-1\,\}$.
- Euler's totient function $\varphi(m)$ counts how many integers in $\mathbb{Z}_m$ are relatively prime to m, i.e., it counts the number of elements of $\mathbb{Z}_m^*$. It can be computed using the formula

  $$\varphi(m) = m \prod_{p|m} \left(1 - \frac{1}{p}\right),$$

  where the product is over the distinct **prime** factors of m.
- $\varphi(m)$ can be computed in `pari-gp` with the `eulerphi` function.
- Let $P = \prod_{k \in \mathbb{Z}_m^*} k$. P has to be relatively prime to m because each of its factors is relatively prime to m. [When m is prime then $P + 1 \equiv 0 \pmod{m}$ — that's Wilson's theorem — but we will not use this fact here.]
- Now assume that $a \in \mathbb{Z}_m^*$, i.e., that $\gcd(a, m) = 1$, and let us now consider what the map $f(x; m, a)$ does to the elements of $\mathbb{Z}_m^*$.
- It scrambles them! Because everything is relatively prime to m, $\mathbb{Z}_m^*$ is mapped into itself! [$au \equiv av \pmod{m}$ implies $u \equiv v \pmod{m}$.] Furthermore, it is a bijection (a one-to-one map).

Fermat's little theorem (continuation)

- So, since the map $x \mapsto ax \bmod m$ when applied to $\mathbb{Z}_m^*$ just reorders its elements, it follows that

$$Q \equiv \left(\prod_{k \in \mathbb{Z}_m^*} ak\right) \equiv \left(a^{\varphi(m)} \prod_{k \in \mathbb{Z}_m^*} k\right) \equiv \left(a^{\varphi(m)} P\right) \pmod{m},$$

  but also that (because of the reordering!)

$$Q \equiv P \pmod{m}.$$

- Since $\gcd(P, m) = 1$, $P^{-1} \bmod m$ exists, and so we can say that, for any $a \in \mathbb{Z}_m^*$, we have (this is Fermat's little theorem)

$$a^{\varphi(m)} \equiv 1 \pmod{m}.$$

- For a prime number p we have $\varphi(p) = p - 1$, and Fermat's little theorem takes the form

$$a^{p-1} \equiv 1 \pmod{p}, \qquad \text{for all } a \text{ with } \gcd(a, p) = 1.$$

  We can take care of the case $a \equiv 0 \pmod{p}$ by multiplying both sides by a:

$$a^p \equiv a \pmod{p}, \qquad \text{for all } a.$$

Fermat's little theorem (examples)

- Let's see what happens for three distinct values of m (all exponentiations are done modulo m):

$m = 7$, $e = \varphi(m) = 6$:

k	k^e	k^{e+1}
0	0	0
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6

(The values of k for which $\gcd(k, m) = 1$ have a gray background.)

$m = 10$, $e = \varphi(m) = 4$:

k	k^e	k^{e+1}
0	0	0
1	1	1
2	6	2
3	1	3
4	6	4
5	5	5
6	6	6
7	1	7
8	6	8
9	1	9

$m = 12$, $e = \varphi(m) = 4$:

k	k^e	k^{e+1}
0	0	0
1	1	1
2	4	8
3	9	3
4	4	4
5	1	5
6	0	0
7	1	7
8	4	8
9	9	9
10	4	4
11	1	11

- What happens for $m = 2 \times 3 \times 5$?
- It looks like $a^{\varphi(m)+1} \equiv a \pmod{m}$ when m does not have repeated prime factors!

Chinese remainder theorem

- Supose that you know that $x \equiv a \pmod{m}$ and that $x \equiv b \pmod{n}$.
- From the first condition x has to be equal to $a + km$ for some integer k.
- But $a + km \equiv b \pmod{n}$, and so $k \equiv m^{-1}(b - a) \pmod{n}$. The modular inverse exists for sure if $\gcd(m, n) = 1$, which we assume is the case here.
- Therefore, we know that $k = ln + c$ for some integer l, where $c = m^{-1}(b - a) \bmod n$. Note that $c = 0$ when $b = a$.
- Finaly, we get $x = a + cm + lmn$, i.e., $x \equiv a + cm \pmod{mn}$.
- It is possible to reach the same conclusion more quickly:

$$x \equiv a(n^{-1} \bmod m)n + b(m^{-1} \bmod n)m \pmod{mn}.$$

- In general, if we know that $x \equiv a_k \pmod{m_k}$, for $1 \leqslant k \leqslant K$, with the moduli m_k pairwise coprime (i.e., $\gcd(m_i, m_j) = 1$ when $i \neq j$) then, with $M = \prod_{k=1}^{K} m_k$ and $M_k = M/m_k$, we have

$$x \equiv \sum_{k=1}^{K} a_k(M_k^{-1} \bmod m_k)M_k \pmod{M}.$$

Chinese remainder theorem (problems)

Solve the following systems of congruences:

$$\begin{cases} x \equiv 0 \pmod{8} \\ x \equiv 1 \pmod{9} \end{cases}$$

$$\begin{cases} x \equiv 0 \pmod{8} \\ x \equiv 8 \pmod{16} \\ x \equiv 3 \pmod{5} \end{cases}$$

$$\begin{cases} x \equiv 2 \pmod{3} \\ x \equiv 2 \pmod{5} \\ x \equiv 2 \pmod{7} \end{cases}$$

$$\begin{cases} x \equiv 1 \pmod{2} \\ x \equiv 2 \pmod{3} \\ x \equiv 4 \pmod{5} \\ x \equiv 6 \pmod{7} \\ x \equiv 10 \pmod{11} \\ x \equiv 12 \pmod{13} \end{cases}$$

$$\begin{cases} x \equiv 12345 \pmod{2718281828} \\ x \equiv 67890 \pmod{3141592653} \end{cases}$$

- Hint: `pari-gp` groks the chinese remainder theorem (`chinese` function). For example, the first problem cn be solved in `pari-gp` by

```
chinese(Mod(0,8),Mod(1,9))
```

Fermat's little theorem (revisited)

- Let p be any prime number. By Fermat's little theorem we know that

$$x^{\varphi(p)} \equiv x^{p-1} \equiv 1 \pmod{p}, \qquad \text{when } \gcd(x,p)=1.$$

- It follows that for any integers r and x we have

$$x^{r(p-1)+1} \equiv x \pmod{p}.$$

  For $x \equiv 0 \pmod{p}$ this is obvious. For the other cases use Fermat's little theorem to adjust the exponent.

- Now consider a second prime, q, different from p. We also have, for any integer s,

$$x^{s(q-1)+1} \equiv x \pmod{q}.$$

- Let t be the least common multiple of $p-1$ and $q-1$. If follows that

$$x^{t+1} \equiv x \pmod{p} \qquad \text{and} \qquad x^{t+1} \equiv x \pmod{q}.$$

- By the chinese remainder theorem this implies that

$$x^{t+1} \equiv x \pmod{pq}.$$

Fermat's little theorem (conclusion)

- The previous result can be generalized to K primes.
- Let $p_1, p_2, \ldots, p_K$ be K distinct primes. Here, p_1 is not necessarily the first prime (two) and so on.
- Let P be their product: $P = \prod_{k=1}^{K} p_k$.
- Let $\lambda(P)$ be the so-called Carmichael function, given by

$$\lambda(P) = \lambda(p_1 p_2 \cdots p_K) = \operatorname{lcm}(p_1 - 1, p_2 - 1, \ldots, p_K - 1).$$

- Then, for any integers k and x, we have

$$\begin{cases} x^{k\lambda(P)+1} \equiv x \pmod{P}, & \text{always,} \\ x^{\lambda(P)} \equiv 1 \pmod{P}, & \text{when } \gcd(x, P) = 1. \end{cases}$$

- This result is often presented with $\lambda(P)$ replaced by $\prod_{k=1}^{K} \varphi(p_k) = \prod_{k=1}^{K} (p_k - 1)$. The later is a multiple of the former.
- This means that in a modular exponentiation we may reduce the exponent modulo $\lambda(P)$ when $\gcd(x, P) = 1$. When $\gcd(x, P) \neq 1$ things are more complicated.

Modular exponentiation

- The modular exponentiation $a^b \bmod m$ can be done recursively using the following two observations:

$$a^{2n} \bmod m = (a^2)^n \bmod m \qquad \text{and} \qquad a^{2n+1} \bmod m = a(a^2)^n \bmod m.$$

  If follows that it can be done using $\mathcal{O}(\log n)$ modular multiplications.

- Example:

$$\begin{aligned}
&13^{21} \bmod 71 = 13 \times (13^2)^{10} \bmod 71 = 13 \times 27^{10} \bmod 71,\\
&27^{10} \bmod 71 = (27^2)^5 \bmod 71 = 19^5 \bmod 71,\\
&19^5 \bmod 71 = 19 \times (19^2)^2 \bmod 71 = 19 \times 6^2 \bmod 71,\\
&6^2 \bmod 71 = 36 \bmod 71,
\end{aligned}$$

  backsubstituting...

$$\begin{aligned}
&19^5 \bmod 71 = 19 \times 36 \bmod 71 = 45 \bmod 71,\\
&27^{10} \bmod 71 = 45 \bmod 71,\\
&13 \times 27^{10} \bmod 71 = 17 \bmod 71,\\
&13^{21} \bmod 71 = 17 \bmod 71 = 17.
\end{aligned}$$

Modular exponentiation (another way)

- Let the exponent n, with $N+1$ bits, be represented in base-2 as follows:

$$n = \sum_{k=0}^{N} n_k 2^k.$$

- Then,

$$a^n \bmod m = a^{\sum_{k=0}^{N} n_k 2^k} \bmod m = \prod_{k=0}^{N} a^{n_k 2^k} \bmod m.$$

- Using the example of the previous slide, we have $n = 21 = 10101_2$, so $N = 4$. Thus,

k	a^{2^k}	use in the final product?
0	13	yes
1	27	no; note that $27 = 13^2 \bmod 71$
2	19	yes; note that $19 = 27^2 \bmod 71$
3	6	no; in general, each number is the square of the previous number
4	26	yes

So, $13^{21} \bmod 71 = 13 \times 19 \times 26 \bmod 71 = 17$.

- Compute $12345^{67890} \bmod 123456789$.

Modular exponentiation (a slightly better way)

- It is possible to do slightly better (Brauer's algorithm). Let the exponent n, with $d+1$ base-B digits, be represented in base-B as follows:

$$n = \sum_{k=0}^{d} n_k B^k = n_0 + B\Big(n_1 + B\big(n_2 + B(\ldots + n_d)\big)\Big).$$

  The last equality is the Horner's rule to evaluate a polynomial. Note that $0 \leqslant n_k < B$. (Usually, B is a power of 2.)

- Then, $a^n \bmod m$ can be evaluated using the following sequence of steps:

$$\begin{array}{cccc} r_0 = a^{n_d} \bmod m & r_1 = r_0^B & r_2 = a^{n_{d-1}} r_1 \bmod m & r_3 = r_2^B \\ r_4 = a^{n_{d-2}} r_3 \bmod m & r_5 = r_4^B & \cdots & \cdots \\ r_{2d} = r_{2d-1} a^{n_0} \bmod m & & & \end{array}$$

- When $B = 8$, the 8 possible values of $a^{n_k} \bmod m$ can be precomputed and stored — **in an interleaved way** to avoid side-channel attacks — in memory.

first word of a^0	first word of a^1	first word of a^2	first word of a^3	first word of a^4	first word of a^5	first word of a^6	first word of a^7
second word of a^0	second word of a^1	second word of a^2	second word of a^3	second word of a^4	second word of a^5	second word of a^6	second word of a^7
...	...	...	...	...	...	...	...

- To explore further: addition chains.

Multiplicative order

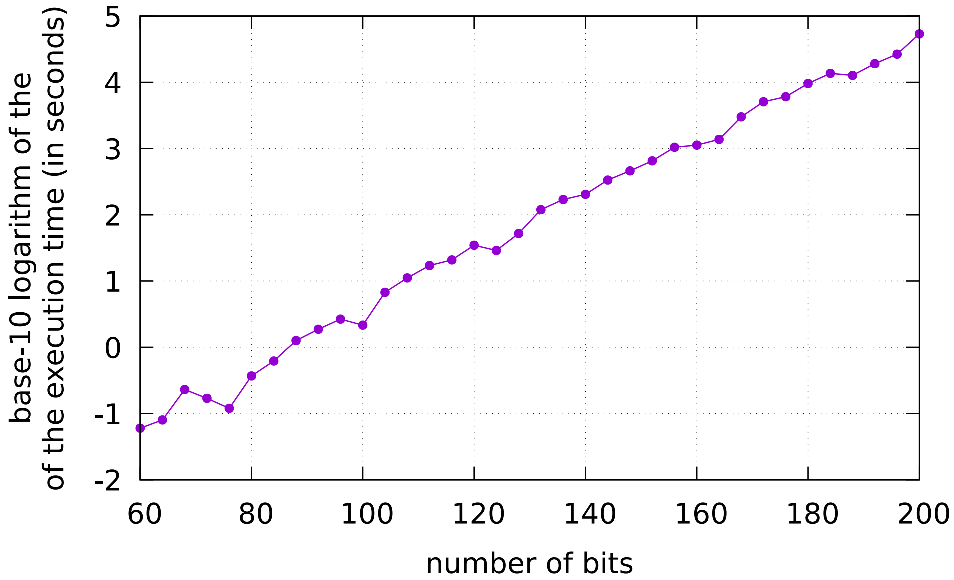
- Fermat's little theorem says that $x^{\lambda(m)} \equiv 1 \pmod{m}$ for any $x \in \mathbb{Z}_m^*$.
- For a given $x \in \mathbb{Z}_m^*$ what is the least exponent o such that $x^o \bmod m = 1$?
- This least exponent is called the order of x modulo m (the function `znorder` computes this in `pari-gp`).
- The order **has** to be a divisor of $\lambda(m)$.
- For a prime number p, $\lambda(p) = \varphi(p) = p - 1$.
- It turns out that there are $\varphi(p-1)$ elements of $\mathbb{Z}_p^*$ with maximal order $p - 1$. These elements are called **primitive roots**.
- `pari-gp` has a function, `znprimroot`, to compute one of them.
- They generate $\mathbb{Z}_p^*$ multiplicatively. In particular, let r be one primitive root. Then, for $k = 0, 1, 2, \ldots, p-2$, $r^k \bmod p$ takes all values of $\mathbb{Z}_p^*$ (without repetitions).
- We can therefore speak of logarithms (modulo p), with respect to base r. The logarithm of $a = r^x \bmod p$ in base r is obviously x. This so-called discrete logarithm problem is currently very hard to solve when p is large.

The discrete logarithm problem for $\mathbb{Z}_p^*$

- Given a prime p, a primitive root r of p, and a, find x such that

$$a \equiv r^x \bmod p.$$

- This is a **hard problem** if $p-1$ has large factors:




```
f(b=10,nt=4)={my(p,r,e,a,x,dt);
  dt=getabstime();
  p=2^b;
  for(i=1,nt,
    p=precprime(p-1);
    r=znprimroot(p);
    a=floor(p/4)+random(floor(p/4));
    x=r^a;
    e=znlog(x,r);
    if(e!=a,quit(1););
  );
  dt=(getabstime()-dt)/nt;
  printf("%4d %8.2f\n",b,0.001*dt);
};
forstep(b=60,200,4,f(b));
```

- But, if $p-1$ only has small factors, the discrete logarithm problem is easy:

```
while(1,p=1+prod(k=1,80,prime(1+random(25)));\
  if(isprime(p),break();););  r=znprimroot(p);  % p has about 400 bits
znlog(r^(10^50),r)                                % milliseconds...
```

Primality tests

- One way to prove that a given number m is prime is to find one of its primitive roots.
- Choose a random a between 2 and $m-2$.
- If $gcd(a, m) \neq 1$, then m is not prime. Better yet, the greatest common divisor allow us to partially factor m.
- By Fermat's little theorem we know that $a^{m-1} \equiv 1 \bmod m$. If this is not so, then definitely m is not prime.
- Furthermore, when m is an odd number, we must have either $a^{(m-1)/2} \equiv 1 \bmod m$ or $a^{(m-1)/2} \equiv -1 \bmod m$.
- Now, it can be shown that a is a primitive root modulo m if, for every prime divisor d of $m-1$, we have $a^{(m-1)/d} \bmod m \neq 1$. If a satisfies these conditions then the order of a modulo m must be $m-1$, and thus m must be prime.
- If not, try another a.
- These exist composite numbers, called Carmichael numbers, for which $a^{m-1} \bmod m = 1$ for all a which are relatively prime to m. For these numbers, $\lambda(m) \mid (m-1)$.

The Miller-Rabin primality test

- Goal: to test if the odd number n, with $n > 3$, is a prime number or not.
- Result of the test: either n is definitely not prime or it may be prime (a probable prime number); in the second case, the probability that the test fails to identify a composite number is **at most 0.25**.
- How it is done (to increase the confidence on the result, do steps 2 to 6 several times):
  1. Let $n-1$ be written as $n-1 = 2^r d$, with d an odd number (so r is as large as possible).
  2. Select at random an integer a uniformly distributed in the interval $2 \leqslant a \leqslant n-2$.
  3. If $\gcd(a,n) \neq 1$, then n is definitely a composite number.
  4. Compute $x_0 = a^d \bmod n$. If $x_0 = 1$ or $x_0 = n-1$ then n is a probable prime.
  5. Otherwise, for $k = 1, 2, \ldots, d-1$, compute $x_k = x_{k-1}^2 \bmod n$. If $x_k = n-1$ then n is a probable prime.
  6. Finally, if we get here, say that n is definitely a composite number (because Fermat's little theorem failed because $a^{(m-1)/2} \equiv \pm 1 \bmod m$).
- The composite numbers that pass this test (meaning that the algorithm above says that they are probable primes) for a given a are called base-a strong pseudo-primes.

The Diffie-Hellman key exchange protocol

- Alice and Bob have never met but wish to exchange a secret key (perhaps to be used in the initialization stage of a symmetric-key cipher algorithm).
- They agree, on a public channel, on a prime p and on primitive root r modulo p. (Choose a prime number with an easy to find factorization of $p - 1$, say a safe prime.)
- Alice generates a random number α between 2 and $p - 2$, or, better yet, between $p^{0.8}$ and $p - p^{0.8}$, and sends Bob the integer $A = r^{\alpha} \bmod p$. She keeps α only to herself.
- Likewise, Bob generates a random number β between 2 and $p - 2$, and sends Alice the integer $B = r^{\beta} \bmod p$. He keeps β only to himself.

  For extra protection, make sure $\gcd(\alpha, p-1) = \gcd(\beta, p-1) = 1$. This forces A and B to be primitive roots.
- Alice computes $S = B^{\alpha} \bmod p = r^{\beta\alpha} \bmod p = r^{\alpha\beta} \bmod p$.
- Bob computes $S = A^{\beta} \bmod p = r^{\alpha\beta} \bmod p$.
- They have arrived at the same number, which is their shared secret. They can now discard A, B, α, β, and p (**do not** reuse p many times).
- Anyone eavesdropping their communications (Eve?, Mallory?) has to either infer α from A or β from B. This is known as the discrete logarithm problem, which is currently a very hard problem to solve.

Diffie-Hellman key exchange protocol (exercises)

- Let $p = 101$ and $r = 2$. Alice chooses $\alpha = 52$, and so $A = 97$. Bob chooses $\beta = 46$ and so $B = 82$. Confirm that the common secret is $S = 58$.
- Let $p = 3141601$ and $r = 26$. Alice chooses $\alpha = 2437429$, and so $A = 1282989$. Bob chooses $\beta = 2988228$ and so $B = 2426580$. The common secret is $S = 1669355$. Try to find α given A and to find β given B.
- Let $p = 31415926541$ and $r = 10$. Alice chooses $\alpha = 29770170945$, and so $A = 5728872032$. Bob chooses $\beta = 23956179675$ and so $B = 22727460975$. The common secret is $S = 26991399064$. Try to find α given A and to find β given B.
- Let $p = 31415926535897932 39$ and $r = 6$. Alice chooses

  $$\alpha = 2459372999633886947, \qquad \text{and so} \qquad A = 2408130236552768716.$$

  Bob chooses

  $$\beta = 2502449096145193611, \qquad \text{and so} \qquad B = 434542471090467423.$$

  The common secret is $S = 1267222359226852228$. Can you find by yourself α given A and β given B? Hint: `pari-gp` does this with the `znlog` function.

Diffie-Hellman key exchange protocol (man-in-the-middle attack)

- Mallory, being a powerful individual, can intercept and replace all messages between Alice and Bob.
- Here's how he can compromise the Diffie-Hellman key exchange protocol.
- Mallory intercepts all messages coming from Alice in the Diffie-Hellman key exchange protocol and impersonates Bob. At the end of the key-exchange protocol he will share a secret key with Alice.
- Likewise, Mallory intercepts all messages coming from Bob in the Diffie-Hellman key exchange protocol and impersonates Alice. At the end of the key-exchange protocol he will share a secret key with Bob (different from the one he shares with Alice).
- From this point on, he decrypts all messages between them, using the appropriate shared secret key, and reencrypts them using the other shared secret key. He may even modify the messages.
- But Alice and Bob can counter this if they send their messages in two or more distinct parts in an interlocked fashion (this assumes that decoding can only be performed after all parts have been received). Also they can, and should, **authenticate** themselves to the other.

ElGamal public key cryptosystem

- Alice and Bob agree on a large prime number p and on an element g of $\mathbb{F}_p^*$ with a large prime order
- Alice chooses a private key a, with $1 < a < p - 1$, and publishes $A = g^a \bmod p$.
- Bob chooses a random ephemeral key k.
- He uses Alice's public key A to compute $c_1 = g^k \bmod p$ and $c_2 = mA^k \bmod p$, where m is the plaintext.
- He then sends (c_1, c_2) to Alice.
- To recover the plaintext m, Alice computes $m = (c_1^a)^{-1} c_2 \bmod p$. This works because $(c_1^a)^{-1} c_2 = g^{-ak} m g^{ak} = m \bmod p$.
- An eavesdropper has to find k from c_1 (discrete logarithm problem).
- A middle-man can easily manipulate c_2; for example, to replace m by $2m$ all that is necessary is to replace c_2 by $2c_2 \bmod p$.
- This public key cryptosystem, implemented exactly as above, has some security problems.

The Rivest-Shamir-Adleman cryptosystem

- The Rivest-Shamir-Adleman cryptosystem (or RSA for short), invented in 1977 on the MIT (but previously invented in 1973 by Clifford Cocks and kept classified by the GCHQ), is based on the observation (Fermat's little theorem) that when N is the product of two distinct prime numbers, i.e., $N = pq$, then for any x and any k we have

$$x^{k\lambda(N)+1} \equiv x \pmod{N}.$$

- In particular, the transformation

$$y = x^e \bmod N$$

can be undone using the transformation

$$x = y^d \bmod N$$

provided that

$$ed \equiv 1 \pmod{\lambda(N)},$$

i.e., provided that $e = d^{-1} \bmod \lambda(N)$.

Rivest-Shamir-Adleman cryptosystem (continuation)

- The key observation is that this is easy to do only when $\lambda(N)$ is known.
- In turn, $\lambda(N)$ can be computed easily only when the factorization of N is known: $\lambda(N) = \lambda(pq) = \mathrm{lcm}(p-1, q-1)$.
- Since the factorization of a large number is considered to be a hard problem — for example RSA-250 was factored in 2020 using about 2700 core years — given N and e it is hard to compute d, and thus to recover y given x.
- It is thus possible to publish N and e without revealing too much information.
- So, anyone using the RSA public key cryptosystem **publishes** hers/his own N and e.
- Sending a ciphered message to someone entails using that person's public modulus (N) and exponent (e)
- About the choice of the primes p and q:
  1. They should be random (**do not** reuse primes!)
  2. $p-1$ and $q-1$ should not have small prime factors

Rivest-Shamir-Adleman cryptosystem (continuation)

- Alice wants to send a message M to Bob.
- First, she fetches Bob's public encription data: a modulus N_{bob} and an encryption exponent e_{bob}.
- Then, she computes the ciphered message $C = M^{e_{\mathrm{bob}}} \bmod N_{\mathrm{bob}}$, and sends it to Bob.
- Bob knows that $N_{\mathrm{bob}} = p_{\mathrm{bob}} q_{\mathrm{bob}}$ (the secret information that only he knows), and so he can compute d_{bob}, the decryption exponent, such that $e_{\mathrm{bob}} d_{\mathrm{bob}} \equiv 1 \pmod{\lambda(N_{\mathrm{bob}})}$.
- Using d_{bob} he can decipher C: $M = C^{d_{\mathrm{bob}}} \bmod N_{\mathrm{bob}}$.
- This works because

$$C^{d_{\mathrm{bob}}} \bmod N_{\mathrm{bob}} = M^{e_{\mathrm{bob}} d_{\mathrm{bob}}} \bmod N_{\mathrm{bob}} = M^{k\lambda(N_{\mathrm{bob}})+1} \bmod N_{\mathrm{bob}} = M$$

- Note that the decryption can be done more efficiently using the Chinese remainder theorem (instead of doing one modular exponentiation modulo N do, perhaps in parallel, two modular exponentiations, one modulo p and another modulo q, and at the end combine them using the Chinese remainder theorem) — but, be aware of side-channel attacks. . .

Rivest-Shamir-Adleman cryptosystem (conclusion)

- The RSA cryptosystem can do even more: it is possible to ensure that the message came from a specified sender (that makes virtually impossible to forge a properly signed message)
- Main idea: Alice computes a message digest (hash) S of the message she wants to send to Bob and enciphers it using her own modulus and **private** decryption exponent:

$$S_{\text{alice}} = S^{d_{\text{alice}}} \bmod N_{\text{alice}}$$

- Bob can recover S using Alice's **public** data:

$$S_{\text{alice}}^{e_{\text{alice}}} \bmod N_{\text{alice}} = S^{e_{\text{alice}} d_{\text{alice}}} \bmod N_{\text{alice}} = S^{k\lambda(N_{\text{alice}})+1} \bmod N_{\text{alice}} = S$$

- So, Bob decodes the message Alice sent him, computes its message digest, and compares it with the S obtained from the S_{alice} data. If they match it is almost certain that it was indeed Alice that has sent the message. Otherwise, someone else was trying to impersonate Alice.
- For this to actually **work**, Bob has to trust Alice's public data. So, that data has to be signed by a party trusted by everyone. Homework: Find out how certification chains and certification authorities work.

Rivest-Shamir-Adleman cryptosystem (big example)

In August 1977, in his Scientific American Mathematical games column, Martin Gardner posed the following RSA challenge.

- Character encoding: space is 00, `A` to `Z` are 01 to 26. Other two digits combinations are illegal.
- The plain text is obtained by concatenating the two digits of each character encoding; the result is a large base-10 integer $\boldsymbol{M}$.
- The plain text was then encoded using the modulus

$$\boldsymbol{N=}\mathbf{1143816257578888676692357799761466120102182967212423625625618429}$$
$$\mathbf{35706935245733897830597123563958705058989075147599290026879543541}$$

  and the exponent $\boldsymbol{e = 9007}$. The encoded message is $\boldsymbol{C = M^e \bmod N}$, with

$$\boldsymbol{C=}\mathbf{9686961375462206147714092225435588290575999112457431987469512093}$$
$$\mathbf{0816298225145708356931476622883989628013391990551829945157815154.}$$

- What is $\boldsymbol{M}$?

Rivest-Shamir-Adleman cryptosystem (solution of the big example)

- It took more than 10 years until $N = pq$ was factored (in 1977 it was estimated that the factorization would take much more time!):

$$p{=}3490529510847650949147849619903898133417764638493387843990820577$$

  and

$$q{=}32769132993266709549961988190834461413177642967992942539798288533.$$

- That made possible the computation of $d = e^{-1} \bmod \operatorname{lcm}(p-1, q-1)$;

$$d{=}209123950501613736909419363468101957730461840930060908793048423 2\\ 20456085696971214722578758536822031722587178886785557376735780271.$$

- Once d was known, M was recovered from $M = C^d \bmod N$:

$$M{=}200805001301070903002315180419000118050019172105011309190800151919090618010705.$$

- $20 \rightarrow T$, $08 \rightarrow H$, $05 \rightarrow E$, and so on. (The complete decryption is in the first slide.)
- Any decryption exponent of the form $d + k \operatorname{lcm}(p-1, q-1)$ works. Try a few values of k to find the exponent with the smallest sideways addition (population count).

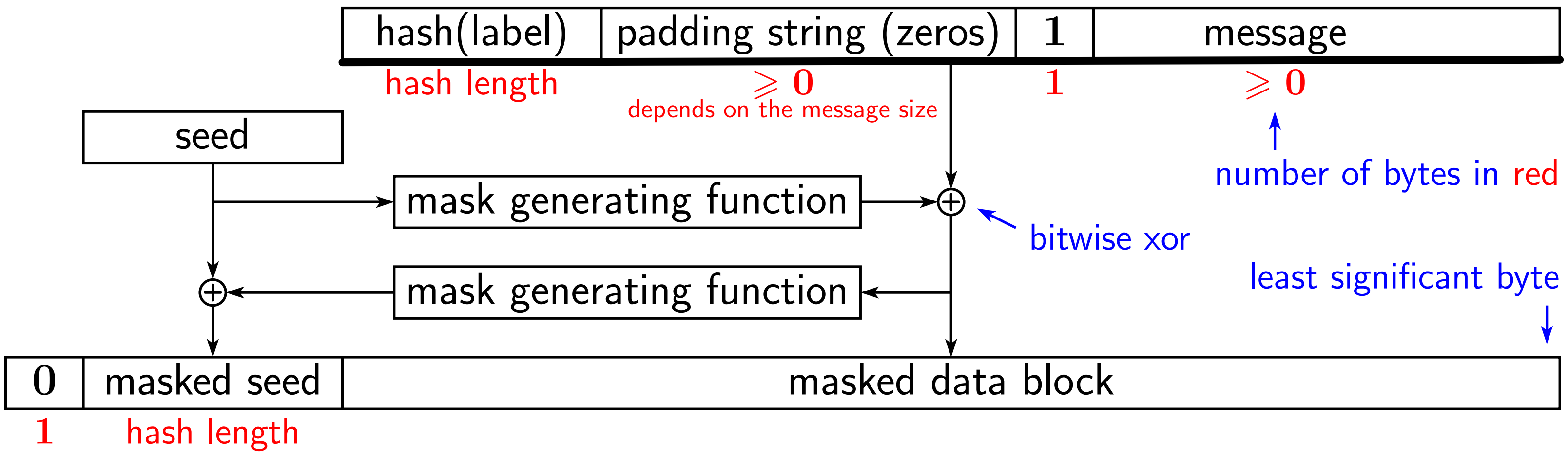
Rivest-Shamir-Adleman cryptosystem (padding)

Data source: section 7 of RFC 8017 (PKCS #1 RSA Cryptography Specifications Version v2.2).

- PKCS #1 v1.5 — avoid:

0	2	padding string (random non-zero bytes)	0	message
1	1	$\geqslant 8$, depends on the message size	1	$\geqslant 0$

- Optimal Asymmetric Encryption Padding (OAEP) — use:



Finite fields

- It is now time to generalize the modular arithmetic concept.
- In the so-called **finite fields** we do arithmetic on integers modulo a **prime** number p **and** we work with **polynomials** with coefficients in $\mathbb{Z}_p$.
- There is one extra twist: we also work modulo a polynomial!
- So our modular arithmetic will have two different aspects:
  - all integer arithmetic is done modulo a prime number p, and
  - all polynomial arithmetic is done modulo a polynomial of degree d.
- Not all polynomials of degree d can be used as the modulus: only those that are irreducible can be used. Just like a prime number, a polynomial is irreducible modulo p if it is not possible to factor it modulo p.
- The irreducibility of the polynomial modulus is fundamental. It ensures that the only polynomial of degree smaller than that of the modulus polynomial that does not have an inverse is the zero polynomial (and that is a fundamental property of a field).

Finite fields (more info)

- The modulus polynomial can (and should) be a **monic** polynomial; the leading coefficient of a monic polynomial is one.

- Indeed, let $\boldsymbol{P(x)}$ be the modulus polynomial, and let $\boldsymbol{A(x)}$ be any polynomial. Then $\boldsymbol{A(x) \bmod P(x)}$ is the remainder $\boldsymbol{R(x)}$ of the division of $\boldsymbol{A(x)}$ by $\boldsymbol{P(x)}$. We have $\boldsymbol{A(x) = Q(x)P(x) + R(x)}$, where $\boldsymbol{Q(x)}$ is the quotient:

$$\begin{array}{l|l} \boldsymbol{A(x)} & \boldsymbol{P(x)} \\ \cline{2-2} \boldsymbol{R(x)} & \boldsymbol{Q(x)} \end{array}$$

- Now, if we replace $\boldsymbol{P(x)}$ by $\boldsymbol{\alpha P(x)}$, where $\boldsymbol{\alpha}$ belongs to $\mathbb{Z}_{\boldsymbol{p}}^*$ — recall that all integer arithmetic is done modulo $\boldsymbol{p}$ and that all elements of $\mathbb{Z}_{\boldsymbol{p}}^*$ are invertible — then we have $\boldsymbol{A(x) = \big(\alpha^{-1}Q(x)\big)\big(\alpha P(x)\big) + R(x)}$, so the remainder is the case no matter how $\boldsymbol{\alpha}$ was selected.

- When the (irreducible) modulus polynomial has degree $\boldsymbol{k}$ the finite field is usually denoted by $\mathbb{F}_{\boldsymbol{p^k}}$ or by $\mathbf{GF}(\boldsymbol{p^k})$; in publications involving finite fields, $\boldsymbol{p^k}$ is often replaced by the easier to write $\boldsymbol{q}$ (if so, $\boldsymbol{q}$ has to be the power of a prime).

- For the particular case $\boldsymbol{k = 1}$ we have that $\mathbb{F}_{\boldsymbol{p}}$ is the same as $\mathbb{Z}_{\boldsymbol{p}}$.

Finite fields (example)

- Let us work with the prime $p = 5$.
- Let us work with the irreducible polynomial (modulo 5):

$$P(x) = x^3 + x^2 + 3x + 4.$$

  This irreducible polynomial was found using the following `pari-gp` code (tutorial):

```
x = ffgen([5,3]);
x.mod
```

  (`x.p` gives the integer modulus, in this case 5).

- Each element of the finite field $\mathbb{F}(5^3)$ is a polynomial of the form

$$a_2x^2 + a_1x + a_0$$

  where $a_0, a_1, a_2 \in \mathbb{F}_5$.

- Addition and subtraction of polynomials is done in the usual way (modulo 5).
- Multiplications is done in the usual way, but replacing x^3 by $-x^2 - 3x - 4$, i.e., by $4x^2 + 2x + 1$. (Why?)

Finite fields (example)

- Continuing the example of the previous slide, the quotient of the division of x^3 by $P(x)$ is $Q(x) = 1$, and so $x^3 \bmod P(x) = x^3 - P(x) = -x^2 - 3x - 4 = 4x^2 + 2x + 1$.
- In pari-gp, this can be confirmed by doing

```
x^3
```

- Here is a larger example:

$$
\begin{array}{rrrrrrr|rrrr}
 & 1x^5 & 4x^4 & 3x^3 & 0x^2 & 1x^1 & 3x^0 & 1x^3 & 1x^2 & 3x^1 & 4x^0 \\ \cline{8-11}
- & 1x^5 & 1x^4 & 3x^3 & 4x^2 & & & 1x^2 & 3x^1 & 2x^0 & \\ \cline{2-5}
 & 0x^5 & 3x^4 & 0x^3 & 1x^2 & & & & & & \\
 & - & 3x^4 & 3x^3 & 4x^2 & 2x^1 & & & & & \\ \cline{3-6}
 & & 0x^4 & 2x^3 & 2x^2 & 4x^1 & & & & & \\
 & & - & 2x^3 & 2x^2 & 1x^1 & 3x^0 & & & & \\ \cline{4-7}
 & & & 0x^3 & 0x^2 & 3x^1 & 0x^0 & & & &
\end{array}
$$

pari-gp confirmation (the modulo arithmetic is done automatically):

```
x^5+4*x^4+3*x^3+x+3
```

Finite fields (useful algorithms)

- Euclid's algorithm works!
- In particular, the extended Euclid's algorithm can be used to compute inverses.
- The modular exponentiation algorithm also works.
- Since in the finite field $\mathbb{F}_q$ — recall that $q = p^k$ — we have

  $$a^q = a \qquad \text{for all } a \in \mathbb{F}_q$$

  (this is similar to Fermat's little theorem), the inverse can also be computed using

  $$a^{-1} = a^{q-2}$$

- Note that the exponents can be reduced modulo $q - 1$.
- The factorization of $q - 1$ is something that is useful to know.

Finite fields (primitive elements)

- The invertible elements (the units) of $\mathbb{F}_q$ form a set, denoted by $\mathbb{F}_q^*$.
- For a finite field we have $\mathbb{F}_q^* = \mathbb{F}_q \backslash \{0\}$.
- The order of an element of $\mathbb{F}_q^*$ is the smallest exponent o for which $a^o = 1$.
- The order has to divide $q - 1$, which is the number of elements of $\mathbb{F}_q^*$.
- A primitive element has maximal order.
- Repeated multiplication by the same primitive element **generates** $\mathbb{F}_q^*$; when that happens the field is said to be a multiplicative cyclic group.
- There exist $\varphi(q - 1)$ primitive elements: if r is a primitive element then r^e will also be a primitive element if and only if $\gcd(e, q - 1) = 1$.
- Therefore, there exist lots of primitive elements if q is large, so finding one is easy (if the factorization of $q - 1$ is known, see next slide).
- In `pari-gp` we can compute a primitive element using the function `ffprimroot()`.

Finite fields (one way to find a primitive polynomial)

Just like in standard modular arithmetic, r is a primitive element of $\mathbb{F}_q^*$ if and only if

- $r^{q-1} = 1$, and
- for every prime divisor d of $q-1$, we have $r^{(q-1)/d} \neq 1$.

One way to find an irreducible polynomial is to find one of the primitive roots of the finite field it generates. So, to compute an irreducible polynomial of degree k when we are working modulo p — finite field $\mathbb{F}_q$ with $q = p^k$ — do the following:

1. Choose a monic polynomial of degree k.
2. Choose a desired primitive element r, say, $r = x$ (that choice is particularly useful, read the slide about cyclic redundancy checksums).
3. Check if r is a primitive element.
4. If so, the polynomial is irreducible, and we are done.
5. If not, then the polynomial may be irreducible but r is not a primitive element or it is not irreducible; go back to the beginning and try another polynomial.
6. Since there exist $\varphi(q-1)$ primitive elements when the polynomial is irreducible, and there exist $\frac{1}{k}\sum_{d|k}\mu(d)p^d$ irreducible polynomials of degree k modulo p, this procedure finds one of them in a reasonable amount of time.

Applications of finite fields

The Diffie-Hellman key exchange protocol can be trivially extended to finite fields.

- Unique difference: Alice and Bob, instead of agreeing on a prime and on one of its primitive roots, have to agree on a finite field (prime $\boldsymbol{p}$, irreducible monic polynomial of degree $\boldsymbol{k}$) and on one of its primitive elements.
- Anyone wishing to infer the shared secret has to solve the discrete logarithm problem, in this case for finite fields.

Elliptic curves (discussed later in this course) also work in finite fields.

Shamir's secret sharing scheme also works in finite fields (discussed later in this course).

- Unique difference: the coefficients of the polynomials, instead of belonging to $\mathbb{F}_{\boldsymbol{p}}$, belong to $\mathbb{F}_{\boldsymbol{p}^{\boldsymbol{k}}}$. Nice! Here we have polynomials whose coefficients are other polynomials (in another variable and subjected to modulo arithmetic in two distinct ways!)

Finite fields and Cyclic Redundancy Checksums (CRCs)

- The so-called cyclic redundancy checksum (CRC) is a way to compute a “signature” of a data set (used at the hardware level as a simple way to perform error detection).
- The data is transformed into a polynomial, and the CRC is just the remainder of that polynomial when divided by a known polynomial.
- Usually, the modulus polynomial is an irreducible polynomial having x as one of its primitive elements (a so-called primitive polynomial).
- Furthermore, this is done with $p = 2$, i.e., in the finite field $\mathbb{F}_{2^k}$. This is so because in the base field, $\mathbb{F}_2 = \mathbb{Z}_2$, addition and multiplication are particularly simple: addition is the **exclusive-or** binary logic operator and multiplication is the **and** binary logic operator.
- They are not useful (i.e., unsafe) in cryptographic applications as a way to compute message hashes (due to its linear nature, it is trivial to forge a message having a specific message hash).
- But it can be used as a hash function in an implementation of a hash table.

Elliptic curves

- Now we get to play with some weird stuff.
- We will do addition is a strange way.
- The addition operator $+$ is a binary operator; it takes two elements of a group and produces a third element of the same group.
- Addition properties (in any group):

  commutative law $x + y = y + x$

  associative law $x + (y + z) = (x + y) + z$

- Idea: suppose we have a plane curve with the following property: any straight line intersects it in exactly three points, counted with multiplicity. If so, the addition of two points on that line can be the third point!
- To make this work, it is necessary to treat the point at infinity as a legitimate point (use homogeneous coordinates, also known as projective coordinates). The point at infinity is the **neutral element**, and so it plays a fundamental role.
- Three intersection points $\Rightarrow$ **cubic equation**.

Elliptic curves (cubic equation)

- The cubic equations we will consider have in the following form (Weierstrass parameterization):

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

- Both x and y belong to a field F (or are the point at infinity).
- With a change of variables (which in some cases cannot be done due to divisions by zero), the equation above can be put in the so-called Weierstrass form

(*) $y^2 = x^3 + ax + b.$

- The so-called discriminant of the curve E, whose points satisfy equation (*), is the quantity

$$\Delta(E) = -16(4a^3 + 27b^2).$$

To avoid degenerate curves this discriminant cannot be zero.

Elliptic curves (homogeneous coordinates)

- In homogeneous coordinates we add a third coordinate: z.
- (x, y) becomes (X, Y, Z).
- (X, Y, Z), for any $Z \neq 0$, corresponds to the two-dimensional point $(\frac{X}{Z}, \frac{Y}{Z})$ [it's an equivalence class].
- $Z = 0$ represents the "points at infinity"; X and Y then specify the direction.
- For an elliptic curve in Weierstrass form, $y^2 = x^3 + ax + b$, for very large x we have $y \approx \pm x^{3/2}$.
- So, very far from the origin, y will be considerably larger than x.
- The homogeneous coordinates of the point at infinity (there are two but only one gets to be used) are $(0, 1, 0)$.

Elliptic curves (pari-gp)

- In pari-gp, the general curve

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

  can be specified using the command

```
E=ellinit([a1,a2,a3,a4,a6]);
```

- In pari-gp, the special curve

$$y^2 = x^3 + ax + b$$

  can obviously be specified using the command

```
E=ellinit([0,0,0,a,b]);
```

  The shortcut

```
E=ellinit([a,b]);
```

  can also be used.

Elliptic curves over finite fields (pari-gp)

- In pari-gp it is also possible to specify the **field** over which all computations will be performed.
- This is specified in a second (optional) argument to `ellinit`.
- If this second argument
  - ⋆ is missing or is the integer 1, the field will be $\mathbb{Q}$
  - ⋆ is the integer p, a prime number, or is a `Mod(*,p)`, the field will be $\mathbb{F}_p$
  - ⋆ is the value returned by `ffgen([p,k])`, the field will be the finite field $\mathbb{F}_{p^k}$
  - ⋆ is a real number, the field will be $\mathbb{C}$

  It may also be a more exotic object.
- The number of points on the elliptic curve can be computed using the `ellcard` function.
- In the specific case of the field $\mathbb{F}_p$ the number of points on the elliptic curve can also be computed using the more efficient `ellsea` function.

Playing with elliptic curves (pari-gp)

- To get a list of official pari-gp tutorials consult this web page.
- In particular, read the elliptic curves tutorial.
- Better yet, read the elliptic curves over finite fields tutorial.
- You can also look at the list of functions related to elliptic curves.
- Let's play!

```
/* find an elliptic curve of the form y^2=x^3+x+1 */
/* over Fp which has a prime number of points      */
forprime(p=2^100,oo,E=ellinit([1,1],p);\
         q=ellsea(E);if(isprime(q),break;);); /* 1 minute */
E.p            /* print the modulus                      */
q=ellsea(E)    /* print the number of points of the curve */
G=E.gen;G=G[1] /* get a generator (there is only one)     */
ellorder(E,G)
```

Elliptic curves (adding two points — geometric interpretation)

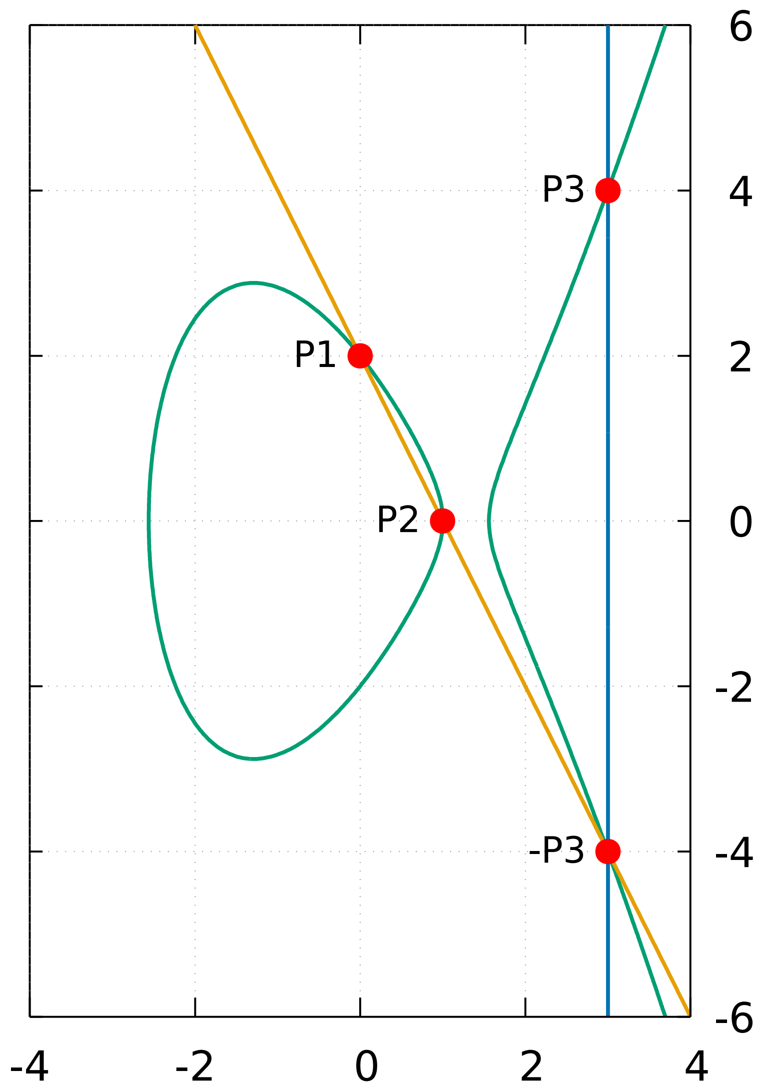
- elliptic curve over $\mathbb{Q}$:

$$y^2 = x^3 - 5x + 4.$$

- pari-gp code:

```
E=ellinit([0,0,0,-5,4]);
P1=[0,2];
P2=[1,0];
ellisoncurve(E,P1)
ellisoncurve(E,P2)
P3=elladd(E,P1,P2);
```

- Draw the line that passes through P_1 and P_2
- That line intersects the elliptic curve at a third point: $-P_3$
- Reflect it on the x axis to get the sum of P_1 and P_2



Elliptic curves (adding the point at infinity)

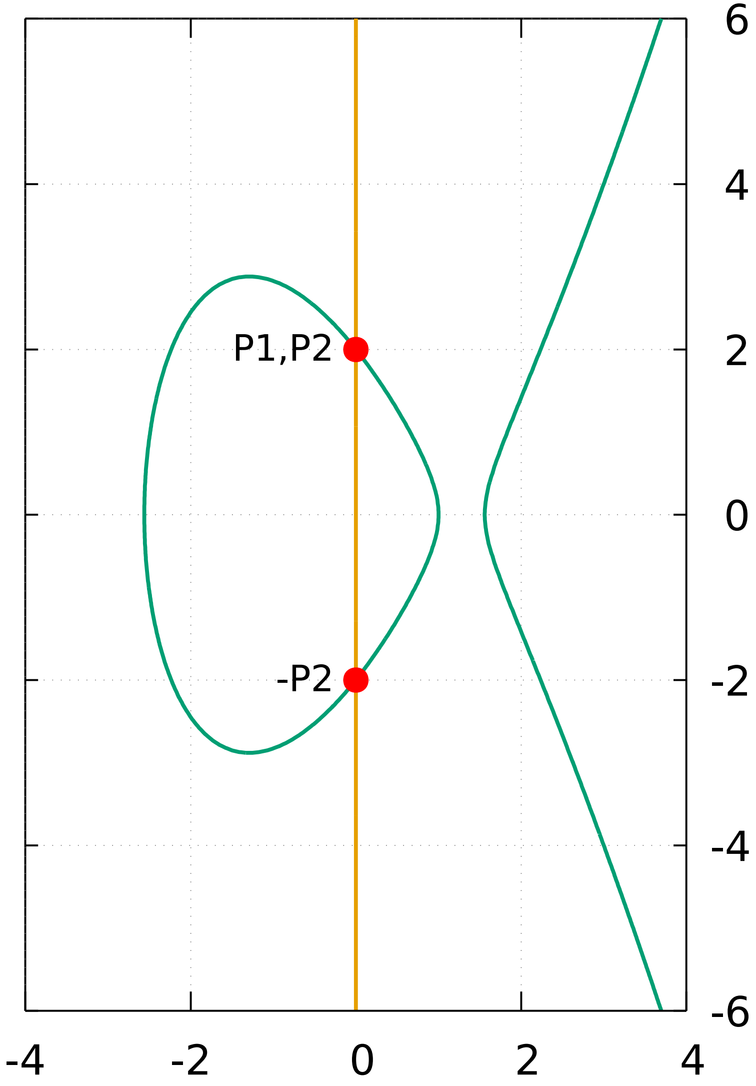
- elliptic curve over $\mathbb{Q}$:

$$y^2 = x^3 - 5x + 4.$$

- pari-gp code (the point at infinity is represented by [0]):

```
E=ellinit([0,0,0,-5,4]);
P1=[0,2];
ellisoncurve(E,P1)
ellisoncurve(E,[0])
P2=elladd(E,P1,[0]);
```

- The point of infinity is the **neutral** element (the zero).
- Adding to a point the point at infinity (intersection with a vertical line) leaves it unchanged.
- Adding a point to its symmetric (its reflection on the x axis) gives rise to the point at infinity.



Elliptic curves (adding the same point — geometric interpretation)

- elliptic curve over $\mathbb{Q}$:

$$y^2 = x^3 - 5x + 4.$$

- pari-gp code:

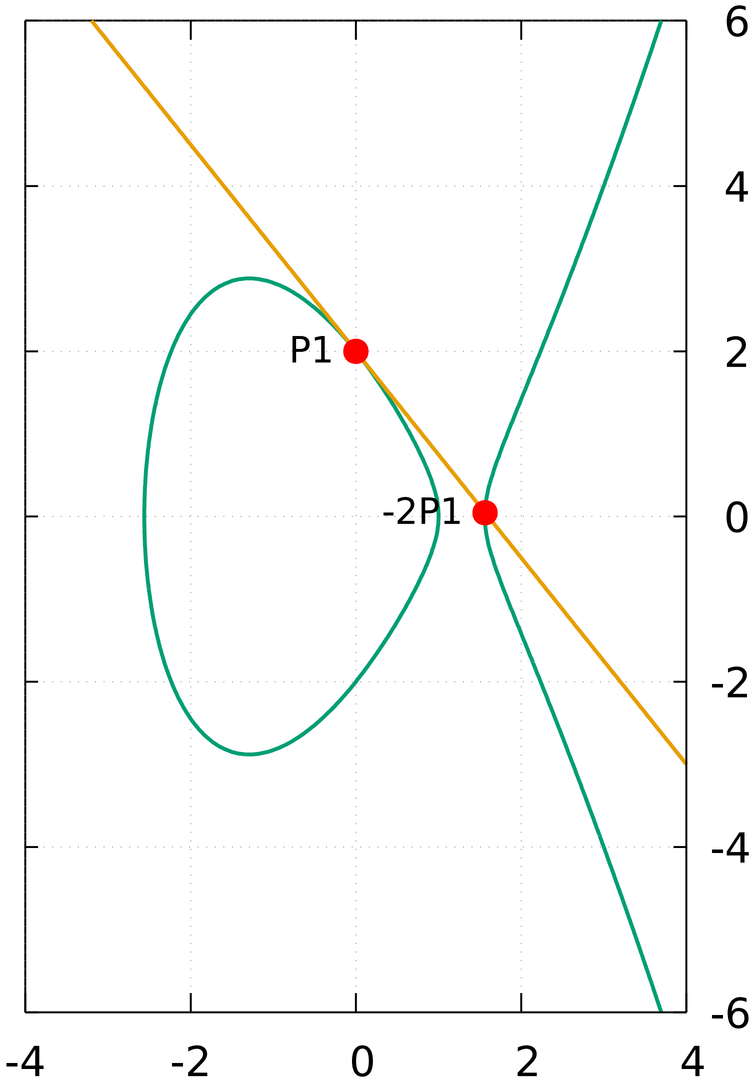
```
E=ellinit([0,0,0,-5,4]);
P1=[0,2];
ellisoncurve(E,P1)
P2=ellmul(E,P1,2);
/* same as P2=elladd(E,P1,P1); */
```

We have

$$2P1 = \left(\frac{25}{16}, \frac{-3}{64}\right)$$

$$3P1 = \left(\frac{96}{625}, \frac{-28106}{15625}\right)$$

$$4P1 = \left(\frac{352225}{576}, \frac{209039023}{13824}\right)$$



Elliptic curves (adding two points — some formulas)

- The equation of a straight line that passes through two distinct points (x_1, y_1) and (x_2, y_2) is

$$(x - x_1)(y_2 - y_1) = (x_2 - x_1)(y - y_1).$$

- It can be put in the form $Ax + By + C = 0$.
- If the inverse of B exists (i.e., the line is not a **vertical** line), then we can say that

$$y = Dx + E.$$

- Putting this in the cubic equation $y^2 = x^3 + ax + b$ gives rise to a polynomial equation of third degree in x of the general form

$$x^3 + \alpha x^2 + \beta c + \gamma = 0.$$

- It has three solutions. Two of them must be x_1 and x_2. The third one is the x coordinate of the point we are looking for.
- When we are working with rational numbers ($\mathbb{Q}$) because the sum of the roots is $-\alpha$ it follows that this third root must also be a rational number!

Multiplication by an integer (adding a point to itself several times)

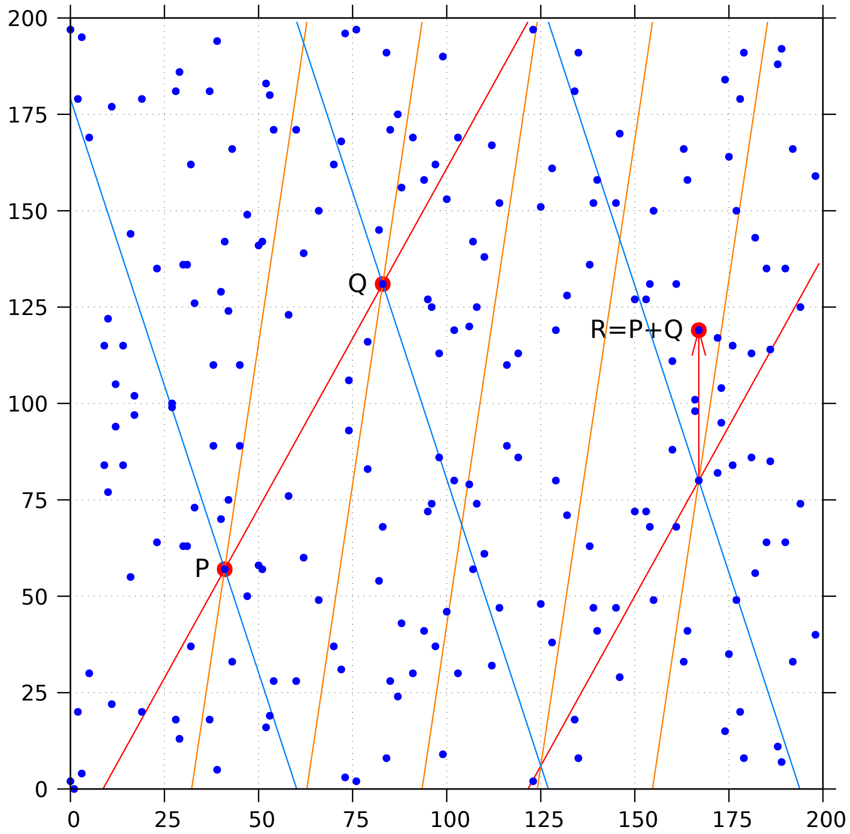
- We can now define the mathematical operation that is useful for cryptographic purposes: multiplication of a point by an integer.
- This corresponds to adding a point with itself several times.
- In terms of cryptographic applications this corresponds roughly to the modular exponentiation done in finite fields.
- Example: to compute, say, $\boldsymbol{11P}$ we can proceed as follows:
  1. $\boldsymbol{11 = 1 + 2 + 8}$
  2. compute $\boldsymbol{2P = P + P}$
  3. compute $\boldsymbol{4P = (2P) + (2P)}$
  4. compute $\boldsymbol{8P = (4P) + (4P)}$
  5. finally, compute $\boldsymbol{11P = (1P) + (2P) + (8P)}$
- This multiplication algorithm is similar in spirit to the algorithm presented in the modular exponentiation slides.
- Hard problem (on some elliptic curves): given $\boldsymbol{P}$ and $\boldsymbol{kP}$ find $\boldsymbol{k}$.

Elliptic curves (aspect of the “curve” on a finite field)

- elliptic curve over $\mathbb{F}_{199}$:

$$y^2 = x^3 - 5x + 4.$$

- it has $\mathbf{218}$ points (including the point “at infinity”):




```
p=199;
E=ellinit([0,0,0,-5,4],p);
N=ellsea(E)             /* 218 */
P=Mod([41,57],p);
ellisoncurve(E,P)       /* 1   */
Q=Mod([83,131],p);
ellisoncurve(E,Q)       /* 1   */
R=elladd(E,P,Q);
lift(Q[1])              /* 167 */
lift(Q[2])              /* 119 */
```

Diffie-Hellman using elliptic curves

- We can now explain how the Diffie-Hellman secret sharing scheme can be done using elliptic curves.
- Alice and Bob agree on an elliptic curve and on a point $\boldsymbol{P}$ — of large order — of that elliptic curve.
- Alice chooses a private random integer $\boldsymbol{k_A}$ and sends $\boldsymbol{k_A P}$ to Bob.
- Bob chooses a private random integer $\boldsymbol{k_B}$ and sends $\boldsymbol{k_B P}$ to Alice.
- The shared secret $\boldsymbol{S}$ is the point $\boldsymbol{k_A k_B P}$; Alice and Bob can compute it easily using the private information they have and the information each received from the other one.
- A third party will have to attempt to compute $\boldsymbol{k_A}$ from the information Alice sent to Bob (over a possibly compromised channel) or to compute $\boldsymbol{k_B}$ from the information Bob sent to Alice. This can be a very hard problem (discrete logarithm for elliptic curves).

Let's play some more with elliptic curves in pari-gp

- Let's see how long it takes to compute k given P and kP:

```
#
bits=50;
p=nextprime(random([2^(bits-1)+1,2^bits-1]));
E=ellinit([0,0,0,1,1],p);
P=random(E);
o=ellorder(E,P)
k=random([2,o-2])
Q=ellmul(E,P,k);
elllog(E,Q,P)
```

Can we do RSA-like things with elliptic curves?

- No. . .

```
bits=100;
p=nextprime(random([2^(bits-1)+1,2^bits-1]));
E=ellinit([0,0,0,1,1],p);
P=random(E);
o=ellorder(E,P);
k=0;while(gcd(k,o)!=1,k=random([2,o-2]);); /* public multiplier */
Q=ellmul(E,P,k);
kInv=lift(1/Mod(k,o)); /* private multiplier used for decoding  */
R=ellmul(E,Q,kInv)     /* we recover P                          */
```

- But here we do not have any hidden secret.
- We would need a point in an elliptic curve for which it would be extremely difficult to compute its order without knowing the "secret".

If you want to know more

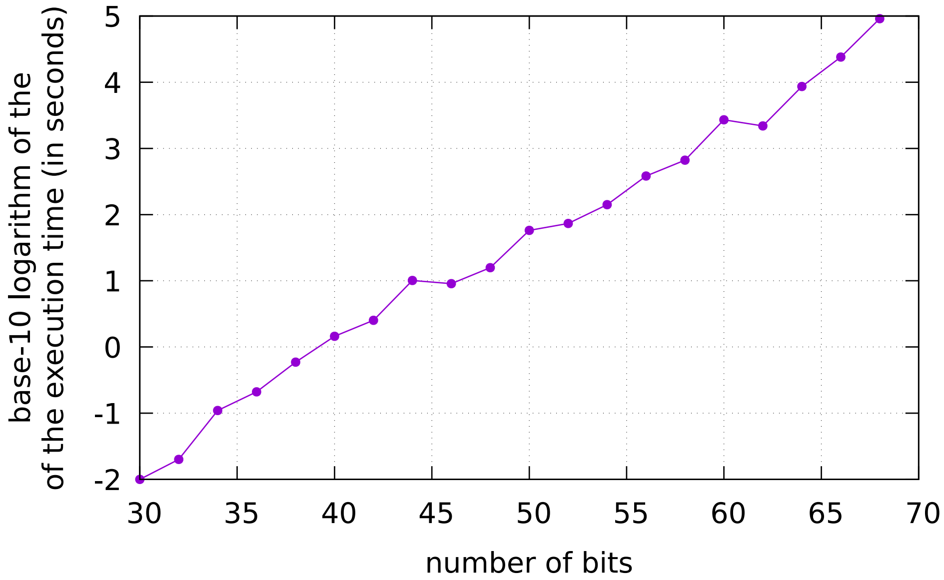
- Edwards curves (alternative parameterization of elliptic curves) — paper about them
- "Safe" elliptic curves
- Curve 25519, wikipedia

The discrete logarithm problem for elliptic curves

- Given points P and Q on an elliptic curve, find k such that

  $$Q = kP.$$

- This is a **hard problem** if the order of P (number of times we have to add P to itself until reaching the point at infinity) is large and has large factors:




```
f(b=10,nt=4)={my(p,q,bad,dt,E,G,P,k1,k2);
  p=2^b; bad=1;
  while(bad>0,
    p=precprime(p-1);
    E=ellinit([1,1],p); q=ellsea(E);
    if(isprime(q)==1,bad=0;);
  );
  printf("%d %d %d\n",b,p,q);
  G=E.gen; G=G[1]; dt=getabstime();
  for(i=1,nt,
    k1=random([1000,q-1000]);
    P=ellmul(E,G,k1); k2=elllog(E,P,G);
    if(k2!=k1,quit(1););
  );
  dt=(getabstime()-dt)/nt;
  printf("%4d %8.2f\n",b,0.001*dt);
};
forstep(b=30,68,2,f(b));
```

Secret sharing

Problem:

- n persons what to share a secret.
- **Any** group of t persons can recover the secret.
- Obviously, $n \geqslant 1$ and $1 \leqslant t \leqslant n$.
- On a computer program, the secret will ultimately be an integer.

How to do it:

- A trusted central entity prepares and distributes part of the secret (a secret share) to each person.

Hurdle to overcome:

- Knowing $t - 1$ shares of the secret **must** not give **any** information about the secret.

Resilience to tampering:

- To completely destroy the secret $n - t + 1$ secret shares have to be corrupted.

Secret sharing (how to do it, idea 1, when $n = t$)

- Let the secret be the integer S, and let it have k bits.
- Let the first $n-1$ shares of the secret, s_1 to s_{n-1}, be random integers with k bits.
- Let the last share of the secret be the exclusive-or of the secret with all the other shares of the secret ($\oplus$ denotes here the bit-wise exclusive-or binary operator):

$$s_n = S \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_{n-1}.$$

- To recover the secret it is only necessary to perform an exclusive-or of all secret shares:

$$S = s_1 \oplus s_2 \oplus \cdots \oplus s_n.$$

- Knowledge of $n-1$ secret shares does not give any information about the secret.
- It is possible to replace the bit-wise exclusive-or operations by addition and subtractions modulo m. In this case, the first $n-1$ secret shares are random integers from 0 to $m-1$, and the last secret share is $(S - s_1 - s_2 - \cdots s_{n-1}) \bmod m$. To recover the secret it is only necessary to add all secret shares (modulo m, of course). If m is a prime number, we can replace addition by multiplication.

Secret sharing (how to do it, idea 2)

Blakley's secret sharing scheme:

- The secret is a point P in a t-dimensional space.
- Each share of the secret is a linear equation (with t unknowns) that has P has one of its solutions.
- Putting together t equations allows us to find P.
- It is necessary to ensure that the system of equations has a unique solution for all possible $C^n_t = \frac{n!}{t!(n-t)!}$ possible combinations of t equations chosen from the n equations, and that is cumbersome.
- Each share of the secret is composed by $t+1$ numbers.
- Improved security: the secret is kept **only** in one of the coordinates of the point P.
- Modular arithmetic should be used (why?).

Secret sharing (how to do it, idea 3)

Shamir's secret sharing scheme:

- The secret in the independent coefficient a_0 of a polynomial of degree $t-1$,

$$A(x) = \sum_{k=0}^{t-1} a_k x^k.$$

- Each secret share in the pair $\bigl(x_k, A(x_k)\bigr)$.
- It is necessary to ensure that distinct values of x_k are used.
- Again, modular arithmetic should be used (why?).
- Each share of the secret is composed by only 2 numbers.

Things to think about:

- Can we do it using square matrices for the a_k coefficients?
- And how about for the a_k coefficients and for the x_k values?

Secret sharing (polynomial interpolation)

Given points (x_k, y_k), for $k = 0, 1, \ldots, n$, with $x_i \neq x_j$ for $i \neq j$, compute the unique polynomial of degree n that passes through these points.

- Newton's interpolation formula:

$$P_0(x) = y_0,$$

and, for $k = 1, 2, \ldots, n$,

$$P_k(x) = P_{k-1}(x) + \big(y_k - P_{k-1}(x_k)\big)\frac{(x - x_0)\cdots(x - x_{k-1})}{(x_k - x_0)\cdots(x_k - x_{k-1})}.$$

- Lagrange's interpolation formula:

$$P_n(x) = \sum_{k=0}^{n} y_k \prod_{\substack{i=0\\ i\neq k}}^{n} \frac{x - x_i}{x_k - x_i}.$$

If arithmetic modulo p is used we must have $x_i \not\equiv x_j \pmod{p}$ for $i \neq j$. If so, all modular inverses needed by Newton's or Lagrange's intepolation formulas exist.

Quadratic residues

- Let n be a positive integer and let a be an integer such that $\gcd(a, n) = 1$.
- a is said to be a quadratic residue modulo n if and only if there exists a x such that

$$x^2 \equiv a \pmod{n}.$$

- When n is a prime number $(n = p)$ there exist three cases:
  1. either a is a multiple of p, or
  2. a is a quadratic residue, or
  3. a is a not a quadratic residue (a quadratic nonresidue).
- The **Legendre** symbol $\left(\frac{a}{p}\right)$ captures this as follows

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \text{if } p \text{ divides } a, \\ +1, & \text{if } p \text{ does not divide } a \text{ and } a \text{ is a quadratic residue modulo } p, \\ -1, & \text{if } p \text{ does not divide } a \text{ and } a \text{ is a quadratic nonresidue modulo } p. \end{cases}$$

Quadratic residues (Legendre symbol)

- For $p > 2$ the Legendre symbol satisfies the equation

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

(Recall that from Fermat's little theorem we know that $a^{p-1} \equiv 1 \pmod{p}$ when p does not divide a.)

- So, $\left(\frac{a}{p}\right) = \left(\frac{a \bmod p}{p}\right)$ and, if a is not divisible by p, $\left(\frac{a^2}{p}\right) = 1$.
- In particular, it is possible to prove that

$$\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}, \quad \text{that} \quad \left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}, \quad \text{and that} \quad \left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right).$$

- If q is an odd prime, we also have (this is the famous law of quadratic reciprocity)

$$\left(\frac{q}{p}\right) = (-1)^{\frac{(p-1)(q-1)}{4}}\left(\frac{p}{q}\right).$$

- These properties allow us to easily compute the Legendre symbol for any a and p (if the factorization of a is known).

Quadratic residues (Legendre symbol computation)

- Example: Compute $\left(\frac{-14}{73}\right)$.
- $\left(\frac{-14}{73}\right) = \left(\frac{-1}{73}\right)\left(\frac{2}{73}\right)\left(\frac{7}{73}\right)$
- $\left(\frac{-1}{73}\right) = (-1)^{36} = +1$.
- $\left(\frac{2}{73}\right) = (-1)^{666} = +1$.
- $\left(\frac{7}{73}\right) = (-1)^{108}\left(\frac{73}{7}\right) = \left(\frac{3}{7}\right)$.
- $\left(\frac{3}{7}\right) = (-1)^{3}\left(\frac{7}{3}\right) = -\left(\frac{1}{3}\right) = -1$. (Obviously, $\left(\frac{1}{p}\right) = +1$.)
- So, putting it all together, we have $\left(\frac{-14}{73}\right) = -1$
- pari-gp agrees (in pari-gp the Legendre symbol can be computed with the kronecker function):

```
kronecker(-14,73)  /* returns -1 */
```

Quadratic residues (Jacobi symbol)

- The **Jacobi** symbol is an extension of the Legendre symbol to the case where the modulus is not a prime number.
- Let $n = p_1^{m_1} p_2^{m_2} \cdots p_k^{m_k}$.
- The Jacobi symbol $\left(\frac{a}{n}\right)$ — yes, it is denoted in exactly the same way as the Legendre symbol — is given by

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{m_1} \left(\frac{a}{p_2}\right)^{m_2} \cdots \left(\frac{a}{p_k}\right)^{m_k}.$$

(The right-hand side of this formula uses Legendre symbols!)

- Its properties are similar to those of the Legendre symbol, but we also have

$$\left(\frac{a}{mn}\right) = \left(\frac{a}{m}\right)\left(\frac{a}{n}\right).$$

- If $\left(\frac{a}{n}\right) = -1$ then a is **not** a quadratic residue modulo nm. **But**, if $\left(\frac{a}{n}\right) = +1$ then a may, or may not, be a quadratic residue modulo nm.

Quadratic residues (counts)

- For a prime p, the number of integers belonging to the set $\{1, 2, \ldots, p-1\}$ that are quadratic residues is exactly $(p-1)/2$.

```
a(n)=local(v,s);v=vector(eulerphi(n));s=0;\
     for(k=1,n,if(gcd(k,n)==1,s=s+1;v[s]=k;););return(v);
qr(n)=local(v);v=a(n); /* number (#) of true quadratic residues */\
     return(length(Set(vector(length(v),k,(v[k]^2)%n))));
j(n)=local(v,c);v=a(n); /* # of true or fake quadratic residues */\
     return(sum(k=1,length(v),kronecker(v[k],n)==1));
f(n)=return([eulerphi(n),qr(n),j(n)]);
f(101) /* returns [100,50,50] */
f(103) /* returns [102,51,51] */
f(107) /* returns [106,53,53] */
f(109) /* returns [108,54,54] */
```

- How about composite numbers that are the product of two distinct prime numbers?

```
f(13*17) /* returns [192,48,96] --- half are fakes! */
f(11*13) /* returns [120,30,60] --- half are fakes! */
f(11*19) /* returns [180,45,90] --- half are fakes! */
```

Quadratic residues (square roots)

- Let p be a prime number of the form $4k+3$ and let a be a quadratic residue modulo p, i.e., $\left(\frac{a}{p}\right) = +1$.
- Then a has two **square roots**.
- They are given by the formula $r = \pm a^{\frac{p+1}{4}} \bmod p$. This is so because if a is a quadratic residue we must have, by Fermat's little theorem, that $a^{\frac{p-1}{2}} = 1 \bmod p$, and so $a^{\frac{p+1}{2}} = a \bmod p$. But $(p+1)/2$ is an even number so the square roots can be computed easily as stated above.
- If n is the product of two primes p and q of the form $4k+3$ and if a is a quadratic residue modulo n then a will have **four** square roots. They can be easily computed using the Chinese remainder theorem. Two of them will have a Jacobi symbol of $+1$ and two will have a Jacobi symbol of -1.
- Example:

```
p=11; q=19; n=p*q; r=20; a=lift(Mod(r^2,n));
rp=lift(Mod(a,p)^((p+1)/4)); rq=lift(Mod(a,q)^((q+1)/4));
r1=lift(chinese(Mod( rp,p),Mod( rq,q))); /*  20, (r1/p)=+1, (r1/q)=+1 */
r2=lift(chinese(Mod( rp,p),Mod(-rq,q))); /*  75, (r2/p)=+1, (r2/q)=-1 */
r3=lift(chinese(Mod(-rp,p),Mod( rq,q))); /* 134, (r3/p)=-1, (r3/q)=+1 */
r4=lift(chinese(Mod(-rp,p),Mod(-rq,q))); /* 189, (r4/p)=-1, (r4/q)=-1 */
```

Quadratic residues (square roots and factorization)

- Let n be the product of two primes.
- Let a be a quadratic residue modulo n.
- Then, it will have four square roots.
- Let x and y be two of then.
- Then $x^2 = y^2 \bmod n$, i.e., $x^2 - y^2 = (x - y)(x + y) = 0 \bmod n$.
- If $y = x$ or $y = -x$, then the above equation gives us nothing.
- Otherwise, we can factor n. Just compute $\gcd(x - y, n)$ and $\gcd(x + y, n)$.
- Example (continuation of the code of the previous slide):

```
p=11; q=19; n=p*q;
r1=20; r2=75; r3=134; r4=189; /* square roots of 191 */
gcd(r1-r2,n);                 /* 11                  */
gcd(r1+r2,n);                 /* 19                  */
```

- pari-gp can only compute square roots when the modulus is prime:

```
sqrt(Mod(5,11))  /* ok (because 5 is a quadratic residue)    */
sqrt(Mod(9,14))  /* error (because the modulus is not prime) */
```

Zero-Knowledge proofs

- In a zero-knowledge proof one party (the prover) proves to another party (the verifier) that she/he knows a secret without revealing any information about it.
- Usually, the proof is probabilistic, i.e., the zero-knowledge proof has several rounds. The larger the number or rounds, the smaller the probability of an impostor faking the proof.

One of two oblivious transfer

- Alice holds two items of information, say m_0 and m_1.
- Bob wants to know one of these two items of information, but does not want Alice to know which one he wants.
- This problem is known as the oblivious transfer problem (in the case, one of two).
- It can be solved in several ways. We will do it here using RSA techniques.
- N is Alice's public RSA modulus and e is the corresponding public exponent; d is the corresponding private decryption exponent.
- At Bob's request, Alice generates two random messages x_0 and x_1 (random numbers smaller than N) and sends then to Bob.
- Bob wants m_b, where $b \in \{0, 1\}$. So, Bob generates a random k and computes and sends to Alice $v = (x_b + k^e) \bmod N$.
- Alice computes $m'_0 = m_0 + (v - x_0)^d \bmod N$ and $m'_1 = m_1 + (v - x_1)^d \bmod N$ and sends both to Bob. Either $(v - x_0)^d \bmod N$ or $(v - x_1)^d \bmod N$ will be equal to k, but Alice has no way of knowing which one is the case.
- Bob computes $m_b = m'_b - k \bmod N$. He can not infer m_{1-b} from m'_{1-b}.

Coin flipping

- Alice and Bob want to flip a coin (by telephone in Manuel Blum's 1981 paper) to decide who wins (in Blum's paper, who gets the car after a divorce).
- Actually, each one flips a coin and if they came out equal (two heads or two tails) Bob wins.
- How can this be done **fairly** and **without cheating** when the two are far apart?
- Using computers: square roots of quadratic residues!

  1. One of the two, say Bob, selects two large random primes p and q of the form $4k+3$ (Blum primes!) and then computes $n = pq$. He then sends n to Alice.
  2. Alice chooses a random b and sends $a = b^2 \bmod n$ to Bob.
  3. Bob computes the 4 square roots $\pm x$ and $\pm y$ of a, chooses one of then, let us call it r, and sends it to Alice.
  4. Alice checks if $\pm r = b$. If so, then Bob wins. If not, he looses.
  5. Alice proves her claims by disclosing b. (Observe that if Alice does not like the outcome she may simply do not finish the execution of this protocol, but that would be cheating.)

- To flip m coins, do step 2 m times, then step 3 m times and so on. It has to be done in this way because as soon as Alice receives the square roots from Bob she will likely be able to factor n (and so be able to change her choice of the b).

Zero-knowledge proofs of identity (main idea)

- Let us introduce two new protagonists:
  1. **Peggy**, who wishes to prove to Victor that she knows a secret
  2. **Victor**, who wishes to verify that Peggy knows the secret
- The proof will be based on challenge-response pairs and it will be probabilistic in nature.
- The probability that an impersonator is accepted (false proof) decreases as more challenge-response pairs are used.
- One of the first published ways to do it uses (again) the hardness of factoring large integers.
- Again, the underlying problem is computing **square roots** modulo $n = pq$.

Zero-knowledge proofs of identity (Feige-Fiat-Shamir scheme)

Preparatory steps (disclosure of public information):

- Peggy chooses a large number n that is the product of two primes of the form $4k+3$ (such a number is called a Blum number). The interesting thing here is that -1 is not a quadratic residue modulo n but its Jacobi symbol has value $+1$; $x^2 \equiv -1 \pmod{pq}$ implies $x^2 \equiv -1 \pmod{p}$ and $x^2 \equiv -1 \pmod{q}$, so -1 can only be a quadratic residue modulo pq if it is a quadratic residue modulo both p and q, which is not the case here because -1 is not a quadratic residue for primes of the form $4k+3$.
- She also chooses k large random numbers $S_1, S_2, \ldots, S_k$ coprime to n.
- Finally, she also chooses each I_j (randomly and independently) as $\pm S_j^{-2} \bmod n$. The interesting thing here is that no matter which choice was made we always have $\left(\frac{I_j}{n}\right) = +1$, so without computing square roots an external observer cannot determine which choice was made. The S_j are witnesses of the quadratic character of the I_j.
- She publishes n and the $I = I_1, I_2, \ldots, I_k$ (but keeps $S = S_1, S_2, \ldots, S_k$ secret).

Instead of publishing n herself, Peggy could have used any Blum integer computed by a trusted entity (the factors of n are not used anywhere in this scheme.)

Zero-knowledge proofs of identity (Feige-Fiat-Shamir scheme)

To generate and verify a proof of identity, Peggy and Victor execute the following T times (the higher T is the harder it will be to fake the proof of identity):

- Peggy chooses a random R and sends to Victor $X = \pm R^2 \bmod n$. Here she also chooses the sign, either $+$ or $-$ randomly, so X is, or isn't a quadratic residue. (Remember, zero knowledge leaked!)
- **[The challenge]** Victor send to Peggy the random vector of bits $E = E_1, E_2, \ldots, E_k$; each E_j is either 0 or 1.
- **[The reply]** Peggy computes and sends to Victor $Y = \pm R \prod_{E_j=1} S_j \mod n$; here, again, she chooses the sign in a random way.
- **[The verification]** Victor checks if $X = \pm Y^2 \prod_{E_j=1} I_j \mod n$, and rejects immediately the proof if this is not so.
- Anyone trying to impersonate Peggy (Eve?) could try to guess the E_j — let the guesses be E'_j — them precompute the next round of the protocol by selecting a random Y and by presenting $X = \pm Y^2 \prod_{E'_j=1} I_j$ when so requested. The probability of success of this cheating attempt is 2^{-k} per round (so, 2^{-kT} overall).

Schnorr Non-interactive Zero-Knowledge Proof

(To be explained in the next school year, but you can look at it now!)

Homomorphic encryption

- Idea: do some useful operation, or operations, using only encrypted data
- Example: in the RSA cryptosystem with unpadded messages, multiplication of the ciphertexts corresponds to multiplication of the plaintexts.
- Using a lot of processing power (and using somewhat cumbersome methods), it is possible to apply an arbitrary function (a logic function described by a boolean circuit) to the encrypted data (to know more, search for fully homomorphic encryption schemes and lattice-based cryptography).

Homomorphic encryption (Paillier cryptosystem)

- Choose two large primes p and q. Ensure that p is not a factor of $q-1$, and vice-versa.
- Compute $n = pq$ and $\lambda = \mathrm{lcm}(p-1, q-1)$.
- Select a random integer g in the interval $]0, n^2[$ that is coprime to n.
- Compute $u = g^\lambda \bmod n^2$. It must be of the form $u = vn + 1$.
- Compute $\mu = \big((u-1)/n\big)^{-1} \bmod n$.
- The public key is (n, g).
- The private key is (λ, μ).
- To encrypt the plaintext m, with $0 \leqslant m < n$, select a random r such that $0 < r < n$, and compute the ciphertext $c = g^m r^n \bmod n^2$.
- To decrypt, compute $x = c^\lambda \bmod n^2$. Them $m = \big((x-1)/n\big)\mu \bmod n$.

Bibliography (work in progress)

1. Eric Bach and Jeffrey Shallit, **Algorithmic Number Theory, Volume 1, Efficient Algorithms**, MIT Press, 1996.

2. Richard Crandall and Carl Pomerance, **Prime Numbers. A Computational Perspective**, second edition, Springer-Verlag, 2005.

3. Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno, **Cryptography Engineering. Design Principles and Practical Applications**, Wiley, 2010.

4. Steven Galbraith, **Mathematics of Public Key Cryptography. Version 2.0**, 2018.

5. Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, **Handbook of Applied Cryptography**, fifth printing, CRC Press, 2001.

6. Richard A. Mollin, **Advanced Number Theory with Applications**, CRC Press, 2009.

7. Richard A. Mollin, **Codes. The Guide to Secrecy from Ancient to Modern Times**, Chapman & Hall/CRC, 2005.

8. Bruce Schneier, **Applied Cryptography. Protocols, Algorithms, and Source Code in C**, second edition, Wiley, 1996.

Asymmetric key management

Asymmetric key management : Goals

- Key pair generation
  - When and how should they be generated
- Exploitation of private keys
  - How can they be kept private
- Distribution of public keys
  - How can them be distributed correctly worldwide
- Lifetime of key pairs
  - Until when should they be used
  - How can one check the obsoleteness of a key pair

Generation of key pairs: Design principles

- Good random generators for producing secrets
  - Bernoulli ½ generator
    - Memoryless generator, unpredictability is crucial!!
    - P(b=1) = P(b=0) = 1/2
- Facilitate without compromising security
  - Efficient RSA public keys
    - Few bits, typically 2^k+1 values (3, 17, $65537 = 2^{16} + 1$)
    - Accelerates operations with public keys
    - No security issues
- Self-generation of private keys
  - To maximize privacy
  - This principle can be relaxed when not involving signatures

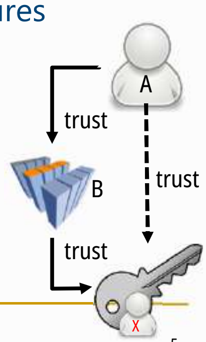
Exploitation of private keys

- Correctness
  - The private key represents a subject
    - Its compromise must be minimized
    - Physically secure backup copies can exist in some cases
  - The access path to the private key must be controlled
    - Access protection with password or PIN
    - Correctness of applications
- Confinement
  - Protection of the private key inside a (reduced) security domain (ex. cryptographic token)
    - The token generates key pairs
    - The token exports the public key but never the private key
    - The token internally encrypts/decrypts with the private key

Distribution of public keys

- ▷ Distribution to all **senders** of confidential data
  - Manual
  - Using a shared secret
  - Ad-hoc using digital certificates
- ▷ Distribution to all **receivers** of digital signatures
  - Ad-hoc using digital certificates
- ▷ Trustworthy dissemination of public keys
  - Transitive trust paths / graphs

    If entity A trusts entity B and B trust in K_X^+,
    then A trusts in K_X^+
  - Certification hierarchies / graphs



Public key (digital) certificates

- ▷ Documents issued by a Certification Authority (CA)
  - Bind a public key to an entity
    - Person, server or service
  - Are public documents
    - Do not contain private information, only public one
  - Are cryptographically secure
    - Digitally signed by the issuer, cannot be changed
- ▷ Can be used to distribute public keys in a trustworthy way
  - A certificate receiver can validate it
    - With the CA's public key
  - If the signer (CA) public key is trusted, and the signature is correct, then the receiver can trust the (certified) public key
    - As the CA trust the public key, if the receiver trusts on the CA public key, the receiver can trust on the public key

Public key (digital) certificates

- X.509v3 standard
  - Mandatory fields
    - Version
    - Subject
    - Public key
    - Dates (issuing, deadline)
    - Issuer
    - Signature
    - etc.
  - Extensions
    - Critical or non-critical
- PKCS #6
  - Extended-Certificate Syntax Standard
- Binary formats
  - ASN.1 (Abstract Syntax Notation)
    - DER, CER, BER, etc.
  - PKCS #7
    - Cryptographic Message Syntax Standard
  - PKCS #12
    - Personal Information Exchange Syntax Standard
- Other formats
  - PEM (Privacy Enhanced Mail)
  - base64 encodings of X.509

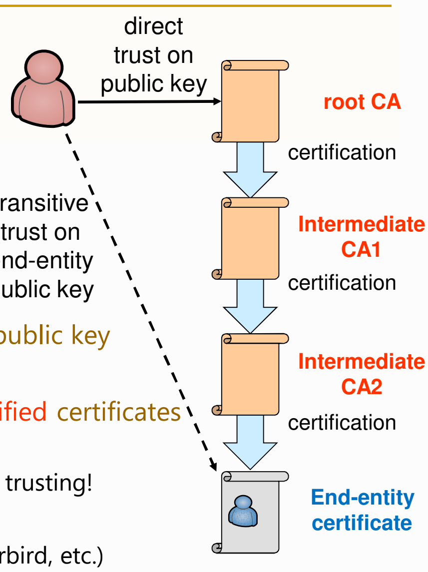
Key pair usage

- A key pair is bound to a usage profile by its public key certificate
  - Public keys are seldom multi-purpose
- Typical usages
  - Authentication / key distribution
    - Digital signature, Key encipherment, Data encipherment, Key agreement
  - Document signing
    - Digital signature, Non-repudiation
  - Certificate issuing
    - Certificate signing, CRL signing
- Public key certificates have an extension for this
  - Key usage (critical)

Certification Authorities (CA)

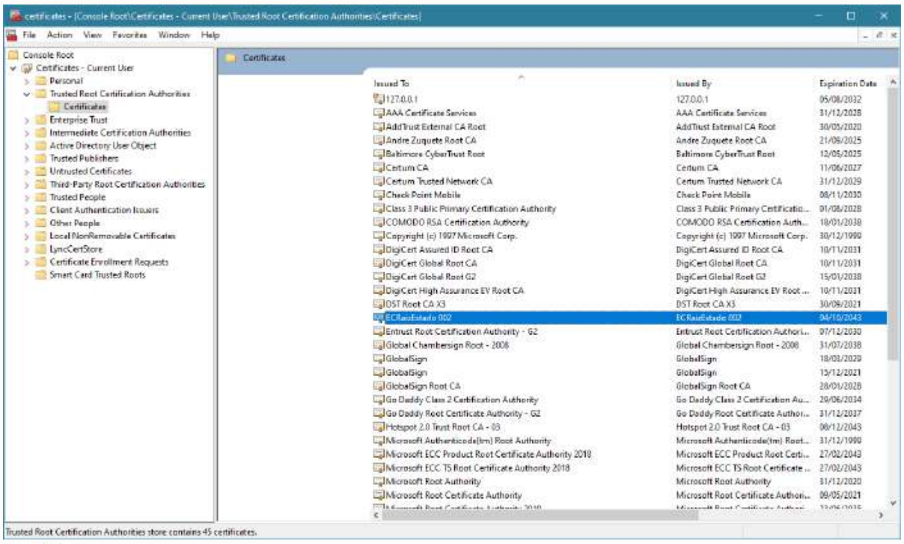
- Organizations that manage public key certificates
- Define policies and mechanisms for
  - Issuing certificates
  - Revoking certificates
  - Distributing certificates
  - Issuing and distributing the corresponding private keys
- Manage certificate revocation lists
  - Lists of revoked certificates

CA types

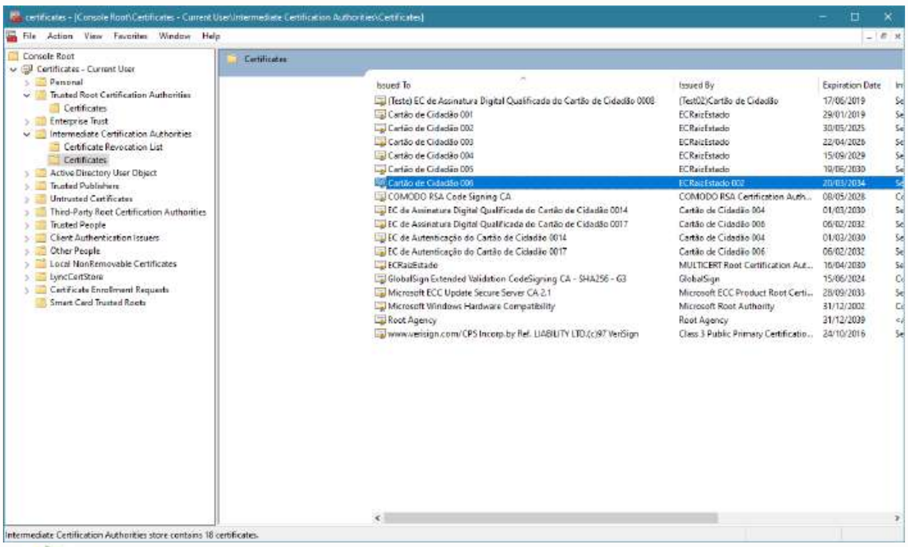


- Intermediate CAs
  - CAs certified by other Cas
- Root CAs
  - CAs for which one has a trusted public key
  - Trust anchor
  - Usually implemented by self-certified certificates
    - Issuer = Subject
    - Self-certification is not a reason for trusting!
  - Manual distribution
    - Tools' repositories (Firefox, Thunderbird, etc.)
    - Operating systems' repositories

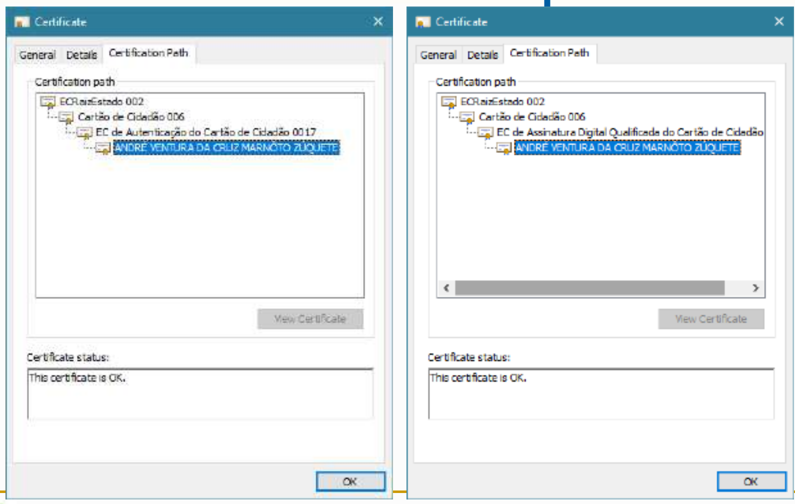
Certificates of Root CAs: Windows 10



Certs. of Intermediate CAs: Windows 10



Certification hierarchies (or chains, paths): Cartão de Cidadão example



Certification hierarchies: PEM (Privacy Enhanced Mail) model

- Distribution of certificates for PEM (secure e-mail)
  - Worldwide hierarchy (monopoly)
  - Single root (IPRA)
  - Several PCA (Policy Creation Authorities) bellow the root
  - Several CA below each PCA
    - Possibly belonging to organizations or companies
- Never implemented
  - Forest of hierarchies
    - Each with its independent root CA
    - Oligarchy
  - Each root CA negotiates the distribution of its public key along with some applications or operating systems
    - ex. Browsers, Windows

Certification hierarchies: PGP (Pretty Good Privacy) model

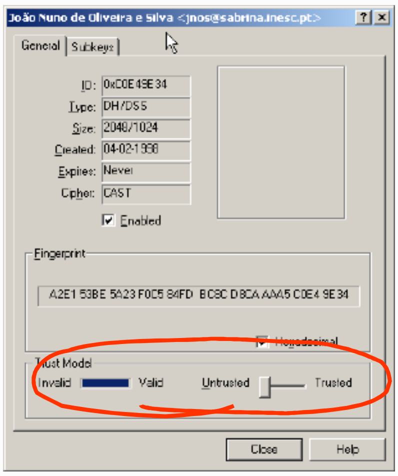
- Web of trust
  - No central trustworthy authorities
    - Each person is a potential certifier
    - Can certify a public key (issue a certificate) and publish it
  - People uses 2 kinds of trust
    - Trust in the keys they know
      - Validated using any means (FAX, telephone, etc.)
    - Trust in the behavior of certifiers
      - Assumption that they know what they are doing when issuing a certificate
- Transitive trust
  - If

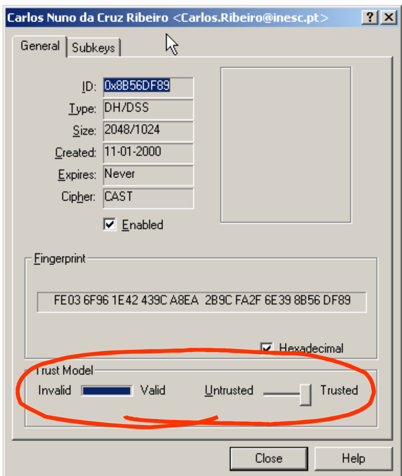
Alice trusts Bob is a correct certifier; and Bob certified the public key of Carl,

  - then

Alice trusts the public key belongs to Carl

PGP public key certificates: Validity vs. trust





Refreshing of asymmetric key pairs

- Key pairs should have a limited lifetime
  - Because private keys can be lost or discovered
  - To implement a regular update policy
- Problem
  - Certificates can be freely copied and distributed
  - The universe of certificate holders is unknown!
    - Thus, cannot be told to eliminate specific certificates
- Solutions
  - Certificates with a validity period
  - Certificate revocation lists
    - To revoke certificates before expiring their validity

Certificate revocation lists (CRL)

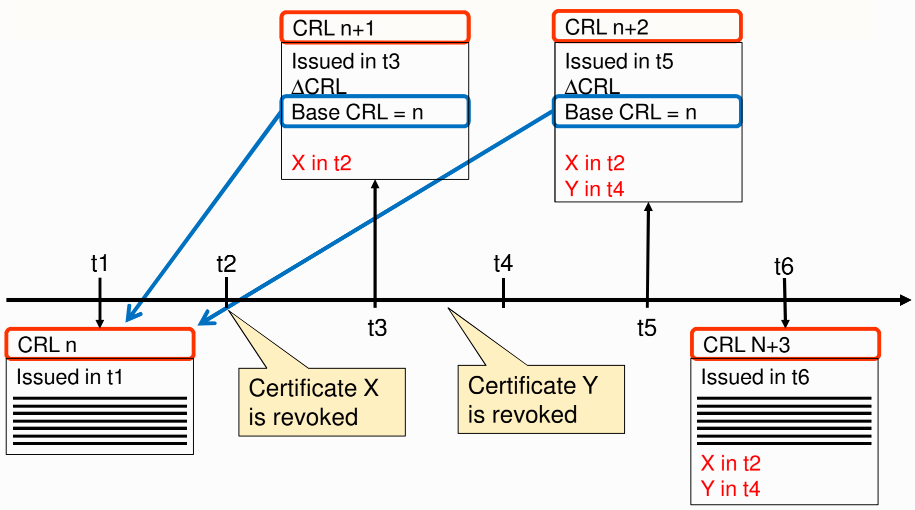
- Base or delta
  - Complete / differences
- Signed list of identifyers of prematurely invalidated certificates
  - Can tell the revocation reason
  - Must be regurlarly fetched by verifiers
    - e.g. once a day
- Single certificate validations
  - OCSP (RFC 6960) query/response
  - OCSP stappling (RFCs 6066, 6961, 8446)
- Publication and distribution of CRLs
  - Each CA keeps its CRL and allows public access to it
  - CAs exchange CRLs to facilitate their widespreading

RFC 3280

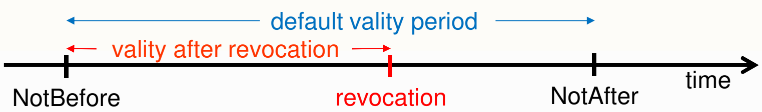
unspecified (0)
keyCompromise (1)
CACompromise (2)
affiliationChanged (3)
superseded (4)
cessationOfOperation (5)
certificateHold (6)

removeFromCRL (8)
privilegeWithdrawn (9)
AACompromise (10)

CRL and Delta CRL



Validity of signatures



- A signature is **valid** if it was generated during the **validity period** of the corresponding pub key certificate
  - The validity period starts on the certificate's **NotBefore** date field
  - By default, the validity ends on the **NotAfter** date field
    - Unless revoked
- A private key can be used out of that period
  - But the signature it produces is invalid
- A public key certificate can be used anytime
  - Namely, after the validity period to check past signatures

Distribution of public key certificates

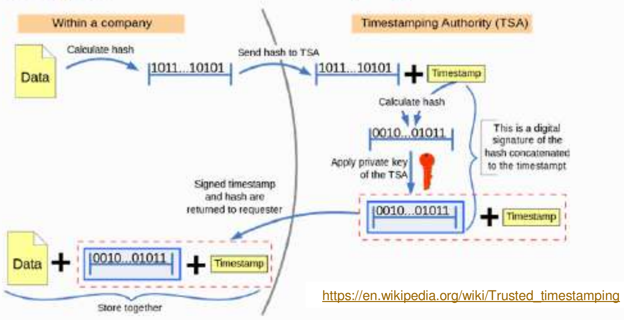
- Integrated with systems or applications
- Directory systems
  - Large scale
    - ex. X.500 through LDAP
  - Organizational
    - ex. Windows 2000 Active Directory (AD)
- Together with signatures
  - Within protocols using certificates for peer authentication
    - e.g. secure communication protocols (SSL, IPSec, etc.)
  - As part of document signatures
    - PDF/Word/XML, etc. documents, MIME mail messages

Distribution of public key certificates

- Explicit (voluntarily triggered by users)
- User request to a service for getting a required certificate
  - e.g. request sent by e-mail
  - e.g. access to a personal HTTP page
- Useful for creating certification chains for frequently used terminal certificates
  - e.g. certificate chains for authenticating with the Cartão de Cidadão

Time Stamping Authority (TSA)

- A service that provides signatures over a timestamp
  - Linked with a data digest

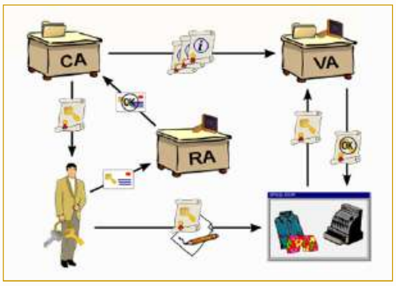


- This is useful for adding trust to a data signature date
  - The signature date becomes linked to the signed data

PKI (Public Key Infrastructure)

- Infrastructure for enabling the use of keys pairs and certificates
  - Creation of asymmetric key pairs for each enrolled entity
    - Enrolment policies
    - Key pair generation policies
  - Creation and distribution of public key certificates
    - Enrolment policies
    - Definition of certificate attributes
  - Definition and use of certification chains (or paths)
    - Insertion in a certification hierarchy
    - Certification of other CAs
  - Update, publication and consultation of CRLs
    - Policies for revoking certificates
    - Online CRL distribution services
    - Online OCSP services
  - Use of data structures and protocols enabling inter-operation among components / services / people

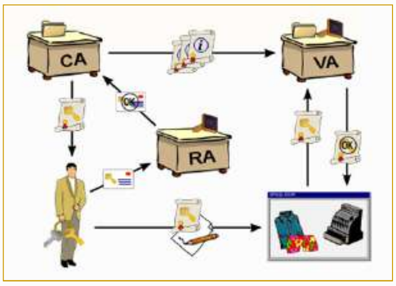
PKI entities: Registration Authority (RA)



- The actual interface with certificate owners
  - Identification and authentication of certificate applicants
  - Approval or rejection of certificate applications
  - Initiating certificate revocations or suspensions under certain circumstances
  - Processing subscriber requests to revoke or suspend their certificates
  - Approving or rejecting requests by subscribers to renew or re-key their certificates

Image src: https://en.wikipedia.org/wiki/Public_key_infrastructure

PKI entities: Validation Authority (VA)



- A service that helps to validate certificates
  - OCSP service

Image src: https://en.wikipedia.org/wiki/Public_key_infrastructure

PKI:
Example: Cartão de Cidadão policies

▷ Enrollment
- In loco, personal enrolment

▷ Multiple key pairs per person
- One for authentication
- One for signing data
- Generated in smartcard, not exportable
- Require a PIN in each operation

▷ Certificate usage (authorized)
- Authentication
  - SSL Client Certificate, Email (Netscape cert. type)
  - Signing, Key Agreement (key usage)
- Signature
  - Email (Netscape cert. type)
  - Non-repudiation (key usage)

▷ Certification path
- PT root CA below global root (before 2020)
- PT root CA (after 2020)
- CC root CA below PT root CA
- CC Authentication CA and CC signature CA below CC root CA

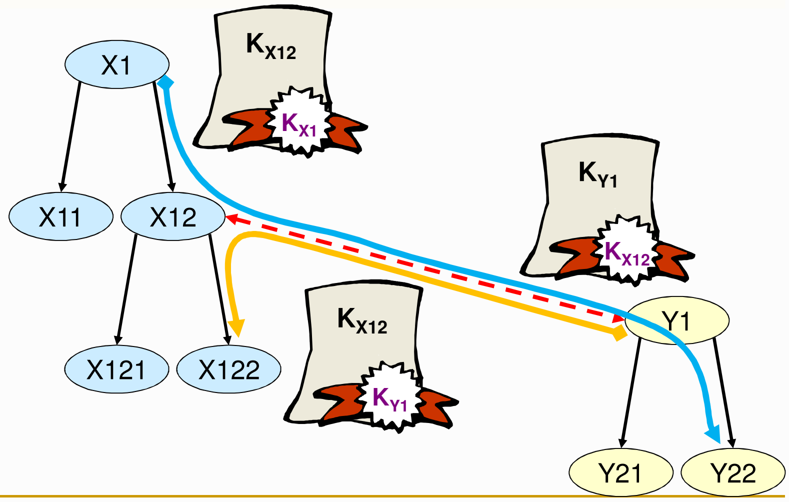
▷ CRLs
- Signature certificate revoked by default
  - Removed if owner explicitly requires the usage of signatures
- Certificates revoked upon a owner request
  - Requires a revocation PIN
- CRL distribution points explicitly mentioned in each certificate

PKI:
Trust relationships

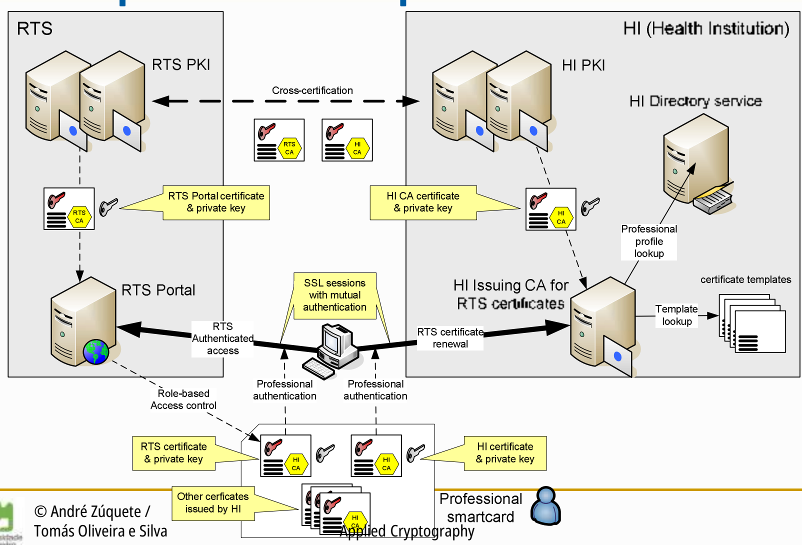
▷ A PKI defines trust relationships in two different ways
- By issuing certificates for the public key of other CAs
  - Hierarchically below; or
  - Not hierarchically related
- By requiring the certification of its public key by another CA
  - Above in the hierarchy; or
  - Not hierarchically related

▷ Usual trust relationships
- Hierarchical
- Crossed (A certifies B and vice-versa)
- Ad-hoc (mesh)
  - More or less complex certification graphs

PKI: Hierarchical and crossed certifications



Cross-certification of PKIs: A practical example



Additional documentation

- [RFC 5280] Internet X.509 Public Key Infrastructure: Certificate and CRL Profile
  - Updated by RFCs 6818, 8398 and 8399
- Other RFCs

  [RFC 4210] Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP) (**+ RFC 6712**)
  [RFC 4211] Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF) (**+ RFC 9045**)
  [RFC 3494] Lightweight Directory Access Protocol version 2 (LDAPv2) to Historic Status
  [RFC 6960] X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP (**+ RFC 8954**)
  [RFC 2585] Internet X.509 PKI Operational Protocols: FTP and HTTP
  [RFC 4523] Internet X.509 PKI LDAPv2 Schema
  [RFC 5519] Internet X.509 PKI Data Validation and Certification Server Protocols
  [RFC 3161] Internet X.509 PKI Time-Stamp Protocol (TSP) (**+ RFC 5816**)
  [RFC 3279] Algorithms and Identifiers for the Internet X.509 PKI Certificate and Certificate Revocation List (CRL) Profile (**+ RFCs 4055**, **5756**, **4491**, **5480**, **8813**, **5758** and **8692**)
  [RFC 5755] An Internet Attribute Certificate Profile for Authorization
  [RFC 3647] Internet X.509 PKI Certificate Policy and Certification Practices Framework
  [RFC 3709] Internet X.509 PKI: Logotypes in X.509 Certificates (**+ RFC 3709**)
  [RFC 3739] Internet X.509 PKI: Qualified Certificates Profile
  [RFC 3779] X.509 Extensions for IP Addresses and AS Identifiers
  [RFC 3820] Internet X.509 PKI Proxy Certificate Profile

Digital signatures

Digital signatures: goals

- ▷ Authenticate the contents of a document
  - Ensure its integrity
- ▷ Authenticate its author
  - Ensure the identity of the creator/originator
- ▷ Non-repudiation
  - Prevent signing repudiation

Digital signatures: fundamental approach

- Signature generation
  - Production of a value using a private key
  - Signer (or signatory) is the private key owner
- Signature verification
  - Validation of an expression using the signature and a public key
  - Anyone can verify
    - Since public keys can be universally known
  - Signature can be linked to the public key owner

Signature schemes

- With message (or document) recovery
  - The message is fully recovered upon a signature validation
  - Signature validation is mandatory prior to message observation
- With appendix
  - The signature is detached from the message
  - The message can be observed anytime

Key elements of a digital signature

- The message (or document)
  - It only makes sense with the signed object
- The signature date
  - Because is usually required
  - Because key pairs have validity periods
- The identity of the signatory
  - Otherwise it would not mean anything

The document to sign

- It may accommodate digital signatures as appendixes
  - PDF, XML
  - DOCX (archive of XML components)
- Other formats may group document and signature
  - S/MIME (mail)
  - JOSE (JSON Object Signing and Encryption)

The signature date

- It may be given by the signatory machine
  - Does not protect against time forgery attacks by the signatory
- It may be given by a Time Stamping Authority (TSA)
  - Does not protect against the future discovery of the private keys used

The identity of the signatory

- Usually provided by a X.509 public key certificate
  - It provides several attributes of the identity
  - It provides the public key for signature validation
  - It provides the acceptable signing time frame
    - Together with the respective CRL

Optional elements of a digital signature

- ▷ Attributes that can help to interpret it
  - Location
    - Where it was signed
  - Reason
    - Why it was signed
  - Appearance
    - Handwritten signature (usually without legal value)
    - Name of the signatory
    - Date of signature
    - Some kind of logo

Digital signatures' algorithms

- ▷ Message recovery scheme
  - Asymmetric encryption and decryption
  - Only for RSA
- ▷ Signing
  $A_x(doc) = info + E(K_x^{-1}, doc)$
- ▷ Verification
  $info \rightarrow K_x$
  $D(K_x, A_x(doc))$
  Check integrity of doc

- ▷ Message appendix scheme
  - Digest functions
  - Asymmetric signature and validation
  - RSA, ElGamal (DSA), EC
- ▷ Signing
  $A_x(doc) = info + E(K_x^{-1}, h(doc+info))$
  $A_x(doc) = info + S(K_x^{-1}, h(doc+info))$
- ▷ Verification
  $info \rightarrow K_x$
  $D(K_x, A_x(doc)) \equiv h(doc + info)$
  $V(K_x, A_x(doc), h(doc + info)) = True$

RSA signatures

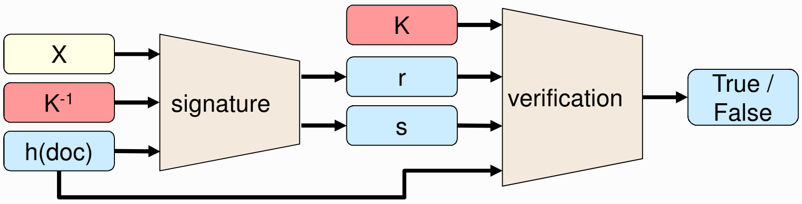
- Creation with private key
  - Validation with the corresponding public key
- Special padding for Signature Scheme w/ Appendix
  - RSASSA-PKCS#1 (v1.5)
    - Deterministic
  - RSASSA-PSS (Probabilistic Signature Scheme)
    - Randomized (EMSA-PSS)
- Hash function prefixing
  - ASN.1 algorithm OID

ASN.1 digest algorithm prefixes

Digest	ASN.1 OID	Perfix (bytes)
MD5	1.2.840.113549.2.5	30 20 30 0C 06 08 **2A 86 48 86 F7 0D 02 05** 05 00 04 10
RIPEMD-160	1.3.36.3.2.1	30 21 30 09 06 05 **2B 24 03 02 01** 05 00 04 14
SHA-1	1.3.14.3.2.26	30 21 30 09 06 05 **2B 0E 03 02 1A** 05 00 04 14
SHA-224	2.16.840.1.101.3.4.2.4	30 2D 30 0D 06 09 **60 86 48 01 65 03 04 02 04** 05 00 04 1C
SHA-256	2.16.840.1.101.3.4.2.1	30 31 30 0D 06 09 **60 86 48 01 65 03 04 02 01** 05 00 04 20
SHA-384	2.16.840.1.101.3.4.2.2	30 41 30 0D 06 09 **60 86 48 01 65 03 04 02 02** 05 00 04 30
SHA-512	2.16.840.1.101.3.4.2.3	30 51 30 0D 06 09 **60 86 48 01 65 03 04 02 03** 05 00 04 40

Digital Signature Standard (DSS)

- With a variant of ElGamal
  - Digital Signature Algorithm (DSA)
  - Uses a random value **X**, and its multiplicative inverse, $\mathbf{X^{-1}}$
  - **r** depends on **X**, **s** depends on $\mathbf{X^{-1}}$
- With elliptic curves (ECDSA)
  - Similar to DSA with EC



Blind signatures

- Signatures made by a "blinded" signer
  - Signer cannot observe the contents it signs
  - Similar to a handwritten signature on an envelope containing a document and a carbon-copy sheet
- Useful for ensuring anonymity of the signed information holder, while the signed information provides some extra functionality
  - Signer X knows who requires a signature (Y)
  - X signs T_1, but Y afterwards transforms it into a signature over T_2
    - Not any T_2, a specific one linked to T_1
  - Requester Y can present T_2 signed by X
    - But it cannot change T_2
    - X cannot link T_2 to the T_1 that it observed when signing

Chaum Blind Signatures

- Implementation using RSA
  - Blinding
    - Random blinding factor K
    - $k \times k^{-1} \equiv 1 \pmod{N}$
    - $m' = k^e \times m \bmod N$
  - Ordinary signature (encryption w/ private key)
    - $A_x(m') = (m')^d \bmod N$
  - Unblinding
    - $A_x(m) = k^{-1} \times A_x(m') \bmod$

Qualified electronic signature

- An electronic signature compliant with the EU eIDAS Regulation
  - Regulation No 910/2014
- Enables to verify the authorship of a declaration in electronic data exchange
  - Over long periods of time
- Can be considered as a digital equivalent to handwritten signatures

Qualified electronic signature

- Three main requirements:
  - The signatory must be linked and uniquely identified to the signature
  - The data used to create the signature must be under the sole control of the signatory
  - Must have the ability to identify if the data that accompanies the signature has been tampered with since the signing of the message

Qualified electronic signature

- Must be created using a qualified signature creation device
  - This device uses specific hardware and software that ensures that the signatory only has control of their private key
- A qualified trust service provider manages the signature creation data that is produced
  - But the signature creation data must remain unique, confidential and protected from forgery

Signature devices

- Crypto tokens
  - Smartcards
  - Cartão de Cidadão
- Cloud HSM (Hardware Secure Modules)
  - Mainly for mobile devices
  - Chave Móvel Digital

PKCS #11

- Crypto tokens' standard interface
  - Cryptoki
- Enables applications to use arbitrary PKCS #11 libraries
  - Developed for a specific set of crypto tokens
- Specification in C
  - There are interfaces for other languages

Microsoft Cryptographic API (CAPI)

- Unique OS security middleware hub
  - Applications use the abstractions it provides
- Cryptographic Services Providers (CSP)
  - Target-specific software module under the CAPI
    - It enables a particular functionality
  - Signature capabilities can be added with CSPs
    - For local crypto tokens
    - For remote, cloud-based HSMs

Long-Term Validation (LTV)

- A document signature may become invalid upon an initial verification
  - Due to a late certification revocation
- Signature algorithms may become vulnerable
  - Allowing signatures with old credentials to be forged
- LTV attempts to handle both issues
  - With successive signature layers
  - Performed by signed documents' holders

LTV Advanced Electronic Signatures (AdES)

- PAdES
  - PDF Advanced Electronic Signature
- CAdES
  - Cryptographic Message Syntax Advanced Electronic Signatures
- XAdES
  - XML Advanced Electronic Signatures