Special Topics on Algorithms

Public Key Cryptosystems

Vangelis Markakis

Public-key cryptosystems

- Main disadvantage of symmetric cryptosystems: Alice and Bob need to agree in advance about the key K through some secure channel
- What if this is infeasible? Can we have encryption without Alice and Bob communicating with each other beforehand?
- Idea: Every entity has a **P**ublic and a **S**ecret key.
- RSA: the public key is a pair of integers
- Suppose Alice (**A)** and Bob (**B)** have public and secret keys as follows:
  - $\mathbf{P_A}$, $\mathbf{S_A}$ for Alice
  - $\mathbf{P_B}$, $\mathbf{S_B}$ for Bob.

Public-key cryptosystems

- ✓ Let E_A() be the encryption function of Alice, and D_A() be the decryption function

- ✓ Challenge for developing a computationally feasible public-key cryptosystem:
  - Need a system where we can reveal the encryption function **E_A()** without running the danger of making the decryption function **D_A()** known
  - On the contrary, in symmetric cryptosystems knowing **E_A()** leads to identifying **D_A()** as well

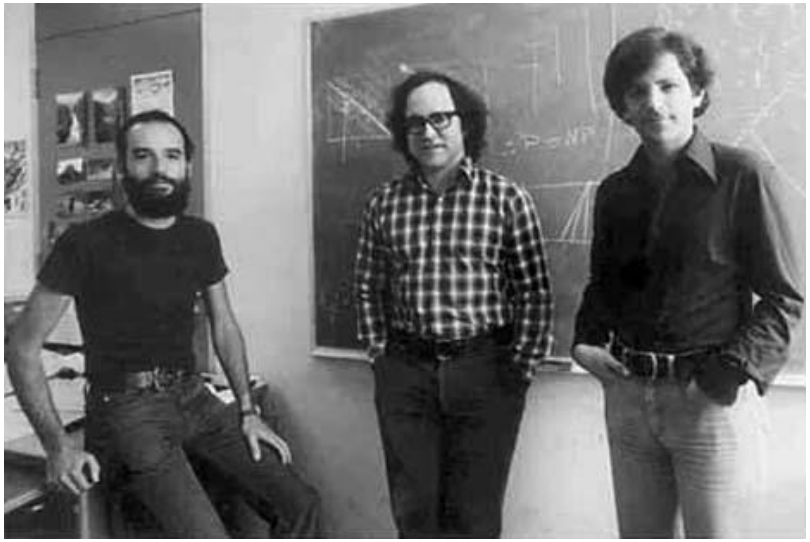
- Public-key cryptosystems
- Hence, overall requirements:
  - ✓ Computationally feasible for a user **B** to produce a pair of keys (Public key P_B, Secret key S_B)
  - ✓ Computationally feasible for a sender **A**, who knows the public key of **B** and wants to send the plaintext **M,** to create the ciphertext: $C = E_B(M)$
  - ✓ Computationally feasible for the receiver **B**, who knows his private key and receives the ciphertext **C** to retrieve the original plaintext **M**: $M=D_B(C)=D_B(E_B(M))$
  - ✓ **Computationally infeasible** to find the private key S_B, knowing only the public key P_B
  - ✓ **Computationally infeasible** to find the message **M**, knowing only the public key P_B and the ciphertext **C**

- Public-key cryptosystems

  Trapdoor one way functions

  - ✓ One-way functions: functions that are easy to compute but hard to invert
  - ✓ Trapdoor: some extra information that allows us to invert a one-way function
  - ✓ Trapdoor one-way functions: one-way functions that are easy to invert when we have the trapdoor
  - ✓ Essentially, in public-key cryptography we are looking for trapdoor one-way functions
  - ✓ [Diffie-Hellman, 1976]: New Directions in Cryptography

- RSA - Rivest, Shamir, Adleman (1978, MIT)
  - ✓ Turing award, 2003



RSA

- RSA - Rivest, Shamir, Adleman (1978, MIT)
  - ✓ Block cipher
  - ✓ All calculations take place in $\mathbf{Z}_n$, for some large n (message space = integers mod n)

Key generation

Choose 2 big and distinct prime numbers	p, q
Compute n:	$n = p \cdot q$
Compute φ(n):	$\varphi(n) = (p-1)(q-1)$
Choose integer e	
$(1<e<\varphi(n))$, such that:	$\gcd(\varphi(n), e) = 1$
Compute d, such that:	$de = 1 \bmod (\varphi(n))$
Public key	$P = \{e, n\}$
Secret key	$S = \{d, p, q\}$

Euler function

- RSA - Rivest, Shamir, Adleman (1978, MIT)

  - ✓ In principle, we could have a phone directory with the public keys of all users

Encryption

Initial message: integer M such that $0 \leq M \leq n-1$

Ciphertext: $C = E(M) = M^e \bmod n$

Decryption

Ciphertext: $0 \leq C \leq n-1$

Message recovery: $M = D(C) = C^d \bmod n$

  - ✓ For the exponentiation: use the repeated squaring algorithm

- In more detail:
- How do we choose e?
  - ✓ Suffices to choose some prime number > max{p, q} (smaller prime numbers can also be suitable) - use primality testing
  - ✓ Recommended value in some systems: $e = 2^{16} + 1 = 65537$
- How do we compute d?
  - ✓ Use extended Euclidean algorithm

Key generation

Choose 2 big and distinct prime numbers	p, q
Compute n:	$n = p \cdot q$
Compute φ(n):	$\varphi(n) = (p-1)(q-1)$
Choose integer e	
$(1<e<\varphi(n))$, such that:	$\gcd(\varphi(n), e) = 1$
Compute d, such that:	$de = 1 \bmod (\varphi(n))$
Public key	$P = \{e, n\}$
Secret key	$S = \{d, p, q\}$

- Example

Key generation		
Choose 2 big and distinct prime numbers	p, q	→ p = 7, q = 17
Compute n:	n = p·q	→ n = 119
Compute φ(n):	φ(n) = (p-1) (q-1)	→ φ(n) = 96
Choose integer e (1<e<φ(n)), such that:	gcd(φ(n), e) = 1	→ e = 5
Compute d, such that:	de = 1 mod(φ(n))	→ d = 77 since 5*77=1 mod96
Public key	P = {e, n}	
Secret key	S = {d, p, q}	

Let M = **19**

Encryption: $C = M^5 \bmod n = 19^5 \bmod 119 = 66$ → Repeated Squaring Algorithm:

Decryption: $M = C^{77} \bmod n = 66^{77} \bmod 119 =$ **19** ↗ (Repeated Squaring Algorithm)

Proof of correctness

- Theorem: For every message M
  - **E(D(M)) = M** and
  - **D(E(M)) = M**
- Proof:

  Let $M \in Z_n$

  Since d is the multiplicative inverse of e modulo φ(n) = (p - 1)(q – 1):

  ed = 1 + k φ(n) for some integer k.

  i) If M ≠ 0 (mod p), we have:

  $$M^{ed} \pmod p \equiv M^{1 + k\varphi(n)} \pmod p$$
  $$\equiv M\,(M^{\varphi(n)})^k \pmod p$$
  $$\equiv M\,(M^{p-1})^{k(q-1)} \pmod p$$
  $$\equiv M \pmod p \text{ (from Fermat's theorem)}$$

  ii) If M = 0 (mod p), then again $M^{ed} \pmod p \equiv M \pmod p$

Proof of Correctness

- ✓ Hence, for every M, M^{ed} (mod p) ≡ M (mod p)
- ✓ Similarly M^{ed} (mod q) ≡ M (mod q)
- ✓ From the corollary of the Chinese Remainder Theorem: when n=pq, x = y mod n iff x=y mod p and x=y mod q
- ✓ ⇒ **D(E(M)) =** M^{ed} (mod n) = M (mod n)

Simpler proof when gcd(M, n)=1:

- ✓ ed = 1 + k φ(n) for some k.

D(E(M)) = $M^{ed} \equiv M^{1 + k\varphi(n)} \pmod{n}$

$\equiv M (M^{\varphi(n)})^k \pmod{n}$

$\equiv M \pmod{n}$ (from Euler's theorem)

Asymmetry of RSA

- ✓ Usually e is a relatively small number ⇒ **fast encryption**
- ✓ E.g. when $e = 2^{16} + 1$, we can encrypt with 17 multiplications
- ✓ The private key d is usually a larger number ⇒ **slower decryption**
- ✓ Around 2000 multiplications or more
- ✓ RSA-Chinese Remaindering (RSA-CRT): Another version of RSA for making decryption faster
  - Almost all operations in the decryption phase are done mod p and mod q and then combined to return the message mod n
  - Intermediate numbers are half in size than before
  - ≈ 4 times faster

- RSA Cryptanalysis
  - ✓ Conjecture: the function $f(x) = x^b \bmod n$, where n is a product of 2 primes is a one-way function
  - ✓ At the moment, there is no function that is **provably** one-way
  - ✓ Theorem: If there are one-way functions, then **P ≠ NP**
  - ✓ Trapdoor in RSA: φ(n) or the factoring of n

- RSA Cryptanalysis

Reduction to the integer factorization problem:

✓Suppose Oscar can easily factor the number n

- If he finds p and q, he can compute φ(n)
- Then, he can easily find d such that de = 1 mod(φ(n)) using the extended Euclidean algorithm

✓For the opposite, we also know that:

✓Theorem: Any algorithm that can compute the exponent d in RSA, can be converted into a randomized algorithm for factoring n

- Hence, if d is revealed, it is not enough to change just d, e, we should also change n

- RSA Cryptanalysis
  - ✓ Note: For factoring n, it suffices to know φ(n)
  - ✓ Suppose φ(n) becomes known
  - ✓ We can solve the system:

$$n = pq$$

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

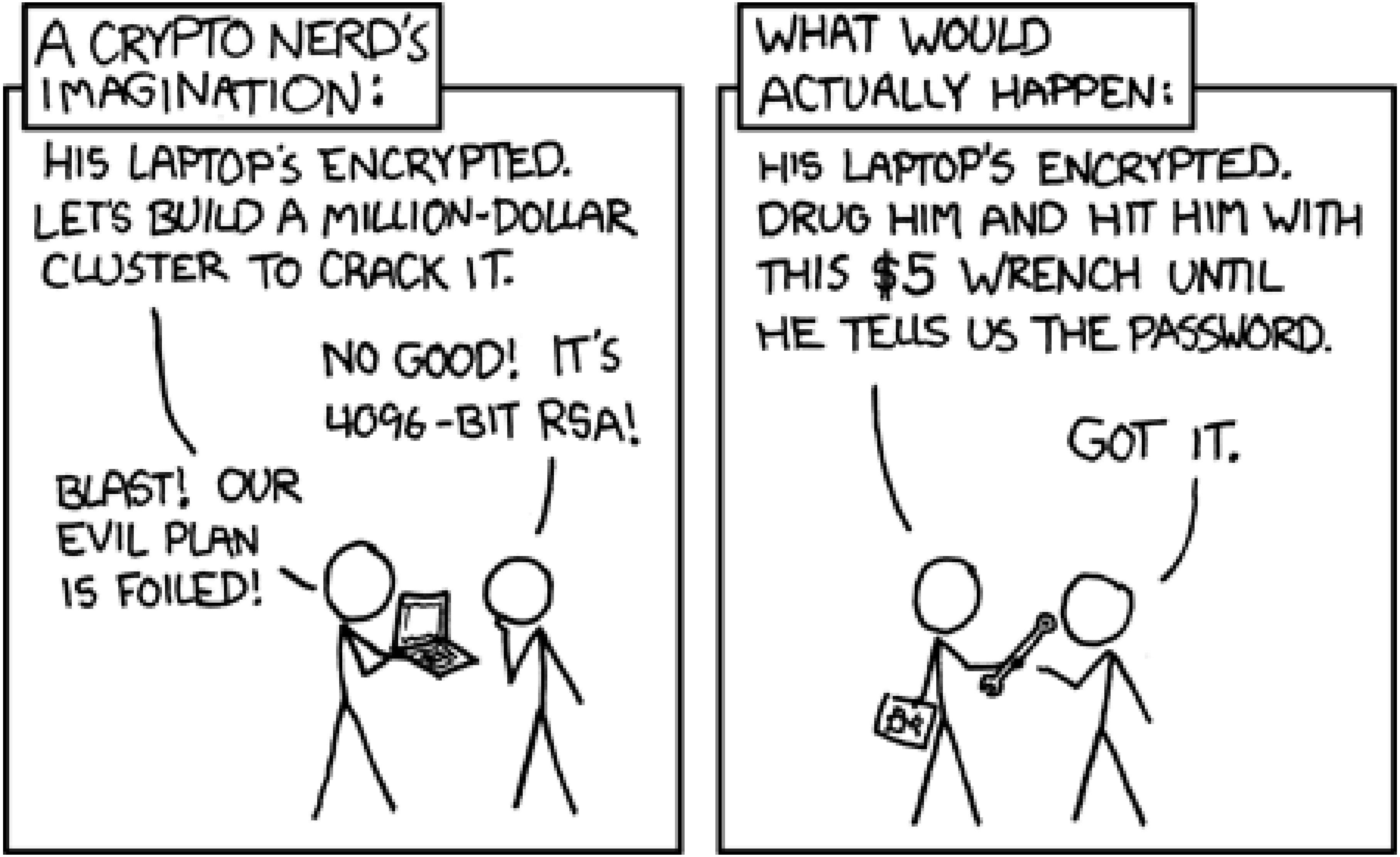
  - ✓ If q = n/p, the factors are derived by solving $p^2 – (N – \varphi(n)+1)p + N = 0$
  - ✓ Corollary: Computing φ(n) is not easier than factoring n

- RSA Cryptanalysis
- In practice:
  - ✓ If we work with 2048 bits, then the key is not breakable within a “reasonable” amount of time, using current knowledge and technology (n > 200 decimal digits)
  - ✓ Factoring algorithms do well for numbers up to around 130 decimal digits
  - ✓ Great open problem to come up with improved factoring algorithms!
  - ✓ NIST guidelines:
    - Since 1/1/2011: 1024-bit keys were declared “deprecated” (acceptable but possibly with some small risk)
    - Since 1/1/2014: 1024 bits no longer acceptable, only 2048 bits

- RSA Cryptanalysis
- Other known attacks (implementation attacks):
  - ✓ Timing attacks [Kocher '97]: The time it takes to do the decryption may yield information about d
  - ✓ Power attacks [Kocher '99]: Measuring power consumption in a smartcard during the run of the repeated squaring algorithm, may also reveal the bits of d
    - Chips should not be vulnerable to power analysis
  - ✓ Fault attacks [Lenstra '96, Boneh, de Millo, Lipton '97]: If some mistake takes place during decryption Oscar may guess d! (applicable mostly for RSA-CRT)
    - These methods work if the computations mod p have been done correctly, and there is a mistake on the computations mod q
    - Rule of thumb: After decryption, we could check that the calculations are all correct, i.e., check that $(C^d)^e \equiv C \bmod n$

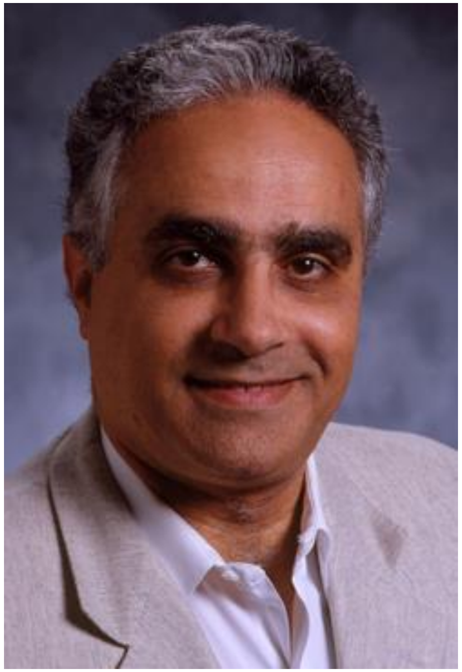
RSA

- RSA Cryptanalysis



http://xkcd.com/538/

- Κρυπτοσύστημα ElGamal
  - ✓ T. Elgamal (1985)



- Discrete logarithm problems
  - ✓ Let $Z^*_p = Z_p - \{0\} = \{1, 2, ..., p-1\}$
  - ✓ The set Z^*_p for a prime p, always has at least one generator: a number g such that for every $a \in Z^*_p$ there exists z with $g^z \equiv a \pmod{p}$
  - ✓ g generates the whole Z^*_p
    - In abstract algebra terms: Z^*_p with multiplication is a cyclic group
  - ✓ For $a \in Z^*_p$, the number z is called the discrete logarithm of a, mod p with basis g
  - ✓ There are known algorithms for finding generators of Z^*_p

- Discrete logarithm problems
  - ✓ When we want to compute the k-th power of a number:
    - Easy by repeated squaring. In Z^*_{17} with k=4, $3^4 \equiv 13$ mod17
  - ✓ **Discrete logarithm in Z_p (DLP)**: the reverse of raising to a power
    - Given that $3^k \equiv 13 \pmod{17}$, find k
    - More generally: Given a generator $g \in Z^*_p$, and an element $\beta \in Z^*_p$, find the unique integer $k \in Z_p$ for which $g^k \equiv \beta \pmod{p}$
  - ✓ Considered a hard problem, when p is chosen carefully
    - For example, for $p \approx 1024$ bits and when p-1 has a «large» prime factor

- ElGamal cryptosystem (T. ElGamal, 1985)

- Based on the difficulty of DLP
- Defined over Z^*_p for some large prime p
  - ✓ Key generation
    - First, select a large prime p such that DLP is difficult
    - An indicative method: Find a prime p such that $p-1 = mq$ for some small integer m and large prime q
    - E.g., with m=2, we can first choose a large prime q and then test whether p=2q+1 is a prime number
      - Use primality testing
    - Choose a generator $g \in Z^*_p$, (hence $g^{p-1} \equiv 1 \bmod p$)
    - Choose an element $\alpha \in \{2, ..., p-2\}$

ElGamal cryptosystem

- Key generation
  - Public + private keys = {(p,g,α,β): $\beta \equiv g^{\alpha} \bmod p$)}
  - Public Key: The numbers p, g, β
  - Private Key: the exponent α
- Encryption algorithm for a message x:
  - Alice chooses a secret random number $k \in Z^*_{p-1}$ and sends to Bob $E(x,k) = (y_1, y_2)$, where
    - $y_1 = g^k \bmod p$
    - $y_2 = x\beta^k \bmod p$ //mask on x
- Decryption algorithm:
  - Upon receiving y_1, y_2, do:
    - $D(y_1, y_2) = y_2(y_1^{\alpha})^{-1} \bmod p$
      - Which results at x

- ElGamal cryptosystem
- Proof of correctness

Claim: $D(y_1, y_2)=y_2(y_1^{\alpha})^{-1} \bmod p = x$

- $y_2(y_1^{\alpha})^{-1} = x\beta^k ((g^k)^{\alpha})^{-1}$
  $= x\beta^k ((g^{\alpha})^k)^{-1}$
  $= x\beta^k ((\beta)^k)^{-1}$ (because $\beta \equiv g^{\alpha} \bmod p$)
  $= x$

- Features
  - ✓ The plaintext **x** is “masked” through the multiplication by $\boldsymbol{\beta^k}$ (yielding $\mathbf{y_2}$)
  - ✓ The ciphertext contains also the value $\mathbf{g^k}$
  - ✓ Bob knows his private key **α,** hence he can derive $(y_1)^{\alpha}$
  - ✓ He then removes the mask by multiplying $\mathbf{y_2}$ with the inverse of $\boldsymbol{\beta^k}$

Example

- ✓ Let $p = 2579$, $g = 2$, $\alpha = 765$
- ✓ $\beta = 2^{765} \bmod 2579 = 949$
- ✓ Suppose Alice wants to send the message $x = 1299$
- ✓ Suppose also that she chooses at random $k = 853$

- ✓ Then:
  - $y_1 = 2^{853} \bmod 2579 = 435$
  - $y_2 = 1299\ (949)^{853} \bmod 2579 = 2396$

- ✓ Bob then calculates
  - $2396\ (435^{765})^{-1} \bmod 2579 = 1299$

Cryptanalysis for ElGamal

- The cryptanalysis can be reduced to the discrete logarithm problem
- Given the public parameters (p, g, β) and the ciphertext (y_1, y_2), Oscar should
  - ✓ either compute the exponent α, from the relation $\beta \equiv g^{\alpha} \bmod p$ (DLP)
  - ✓ or find k from the relation $y_1 \equiv g^k \bmod p$ (again DLP), and then compute x via: $x = y_2(\beta^k)^{-1} \bmod p$

- Other public key cryptosystems
  - ✓ Merkle-Hellman Knapsack systems, all broken except:
    - Chor-Rivest
  - ✓ McEliece
  - ✓ Elliptic Curve systems

Elliptic Curve Systems

- ✓ Studied initially in [Miller ’86, Koblitz ’87]
- ✓ Wider use from 2004 onwards
- ✓ NIST approval: 2006
- ✓ Important advantage: smaller key size for the same security level as other public-key systems
- ✓ Applications: Bitcoin, SSH (about 10% of ssh implementations), Austrian citizen card, etc
- ✓ Main idea:
  - DLP can be defined not just over Z^*_p but over other abelian groups
  - Find suitable such groups where DLP is difficult

- Elliptic Curve Systems

Symmetric Scheme (key size in bits)	ECC-Based Scheme (size of *n* in bits)	RSA/DSA (modulus size in bits)
56	112	512
80	160	1024
112	224	2048
128	256	3072
92	384	7680
256	512	15360

Source: Certicom

Using elliptic curves we decrease significantly the key size!

- Other applications of public-key cryptosystems
  - ✓ Digital signatures
  - ✓ Bit pattern that depends on the message to be signed
  - ✓ Idea 1: use the decryption algorithm as a signing algorithm (treat the message as a ciphertext)
  - ✓ Size of signature could be big
  - ✓ Idea 2: Apply the signing algorithm to a hash of the message
  - ✓ Digital Signature Standard (DSA): Based on ElGamal and the Secure Hash Algorithm (produces signature size around 320 bits)

Bibliography on Number Theory and Cryptography

- [DPV] S. Dasgupta, C. H. Papadimitriou, U. V. Vazirani : "Algorithms"
  - ✓ Chapter 1, Sections 1.1 – 1.4
  - ✓ Representative exercises: 1.11 – 1.13, 1.19 – 1.22, 1.25, 1.27-1.28
- [CLRS] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: "Introduction to Algorithms"
  - ✓ Chapter 31 on number-theoretic algorithms
  - ✓ Representative exercises: most exercises up until the RSA section