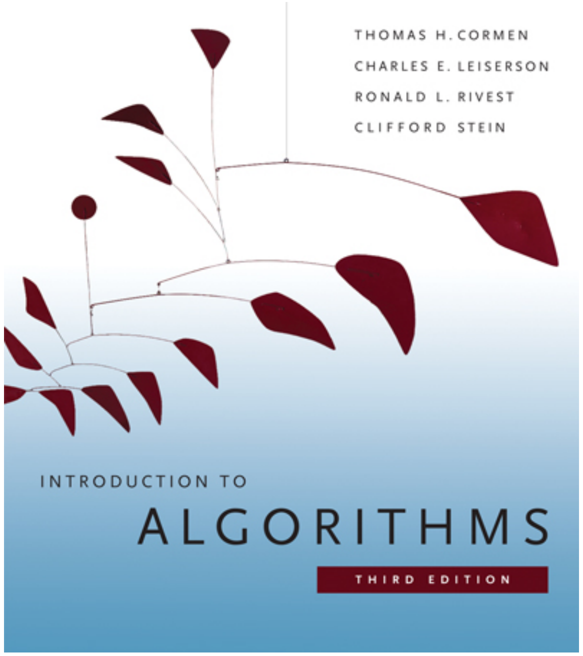
6.006- *Introduction to Algorithms*



Lecture 23

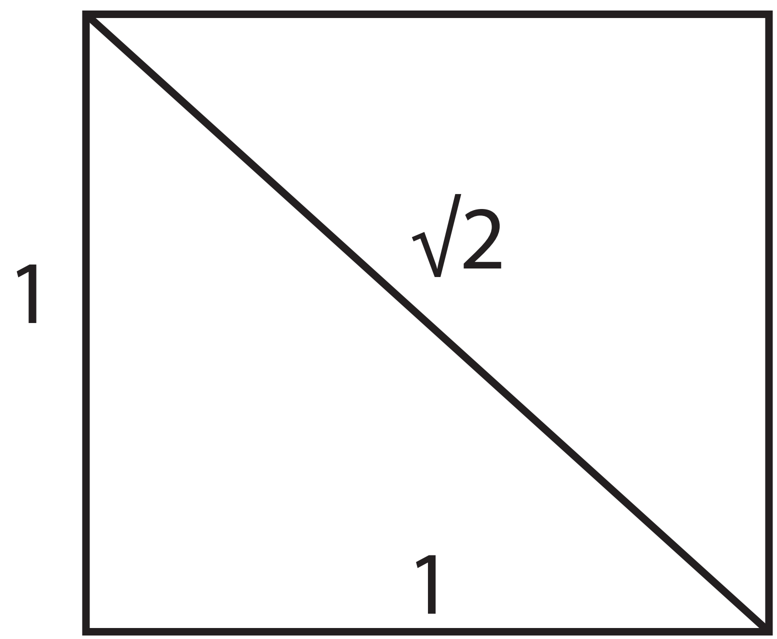
Prof. Patrick Jaillet

Outline

- "Numerics II" - algorithms for operations on large numbers
- Today:
  - – quick review: irrationals; large number operations: addition, multiplication, division
  - – cryptography (CLRS 31)
    - motivations
    - primality testing
    - modular exponentiation
    - integer factorization

2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229
233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349
353	359	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541
547	557	563	569	571	577	587	593	599	601
607	613	617	619	631	641	643	647	653	659
661	673	677	683	691	701	709	719	727	733
739	743	751	757	761	769	773	787	797	809
811	821	823	827	829	839	853	857	859	863
877	881	883	887	907	911	919	929	937	941
947	953	967	971	977	983	991	997	....	

Computing $\sqrt{h}$ to lots of digits ... why?

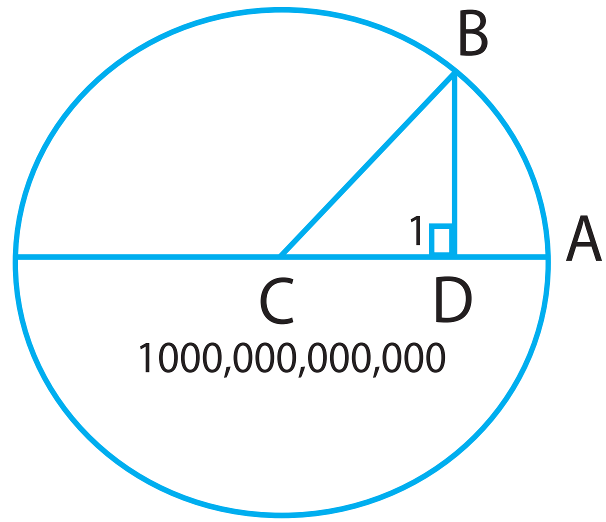


1\. 414 213 562 373 095
048 801 688 724 209
698 078 569 671 875
376 948 073 176 679 ...

question: pattern?

Computing $\sqrt{h}$ to lots of digits ... why?

- geometry problem
  - $BD = 1$
  - what is AD?



$$AD = AC - CD = 500{,}000{,}000{,}000 - \sqrt{500{,}000{,}000{,}000^2 - 1}$$

- question: first non-trivial digits?

(Taylor's expansion $\sqrt{1+x} = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3 - \frac{5}{128}x^4 + \cdots$)

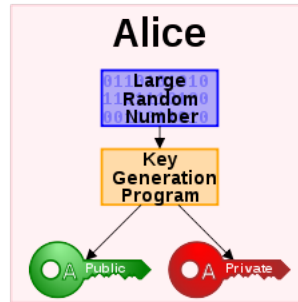
$\boxed{\Rightarrow AD = 10^{-12} + 10^{-36} + 2.10^{-60} + 5.10^{-84}}$

Cryptography

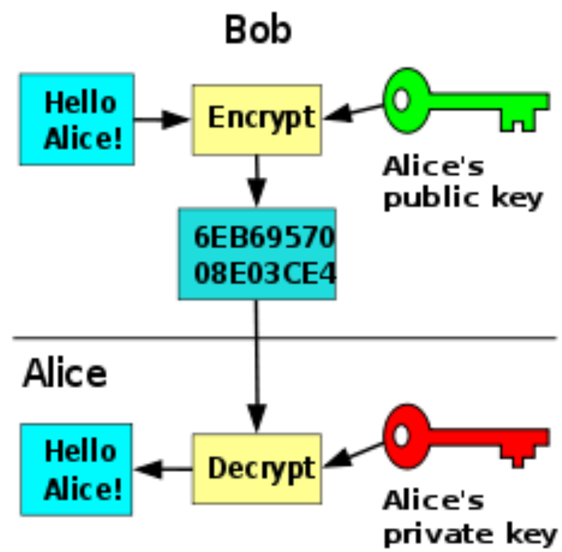
- long history
- modern development
  - public-key cryptography
    - some designed as early as 1973 – UK – but classified top-secret and revealed publicly in 1998
    - RSA (1978) for “Rivest, Shamir, and Adleman” is the first algorithm suitable for signing and encryption – widely used in electronic commerce protocols

Public-key cryptography

- key generation
  - public key
  - private key



- encryption
- decryption

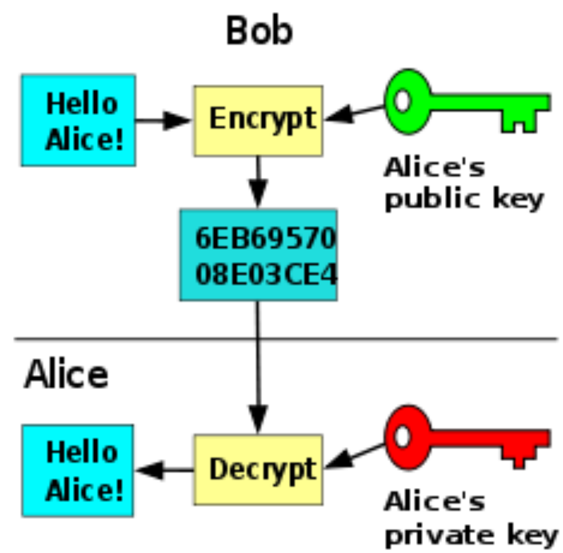


RSA: key generation

- choose two prime number p and q
- compute $n=pq$
- compute $f(n)=(p-1)(q-1)$
- choose e, $1<e<f(n)$, and $gcd(e,f(n))=1$ (e and $f(n)$ are co-prime)
  - e is released as the public key exponent
- find $d=e^{-1} \bmod f(n)$
  - d is kept as the private key exponent

RSA: encryption

- Alice transmits her public key (n,e) to Bob
- Bob wishes to send a message "Hello Alice!" to Alice
  - he turns the message into an integer m, $0<m<n$, using an agreed upon protocol (a padding scheme)
  - he computes $c = m^e \bmod n$
  - he transmits c to Alice

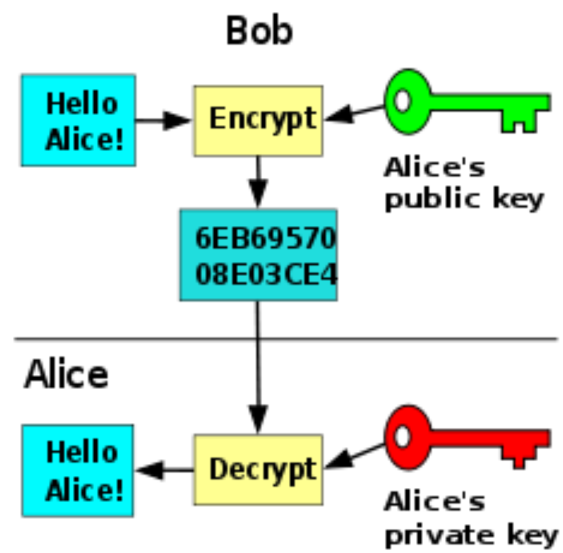


RSA: decryption

- Alice can recover m from c by using her private key exponent d as follows:

$$m = c^d \bmod n$$

- Given m, she can recover the message "Hello Alice!" by reversing the padding scheme



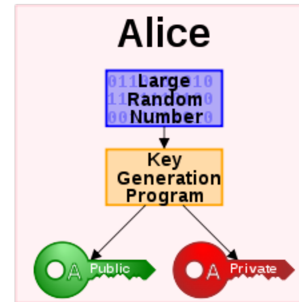
RSA: example

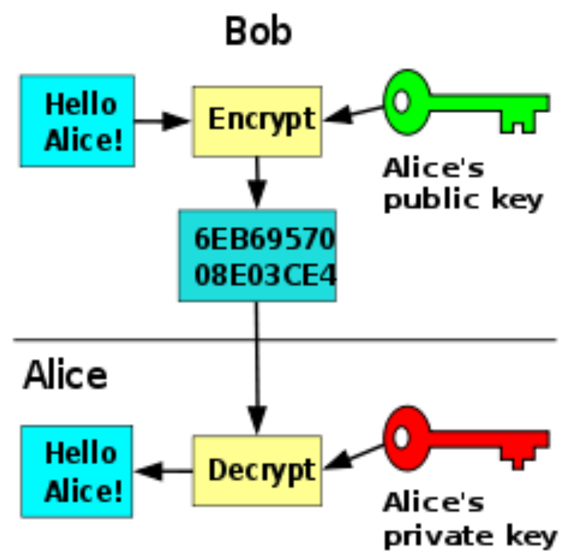
- key generation:
  - choose $p = 61$ and $q = 53$
  - compute $n=pq=3233$
  - compute $f(n)=(p-1)(q-1)=3120$
  - choose a prime number e not a divisor of 3120, say $e=17$
  - find $d = e^{-1} \bmod f(n)=2753$
  - the public key is $(n,e)=(3233,17)$
  - the private key is $(n,d)=(3233,2753)$
- encryption: $m=65$ is encrypted as

$$c = 65^{17} \bmod 3233=2790$$

- decryption: $c=2790$ is decrypted as

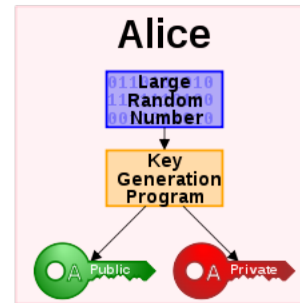
$$m = 2790^{2753} \bmod 3233=65$$

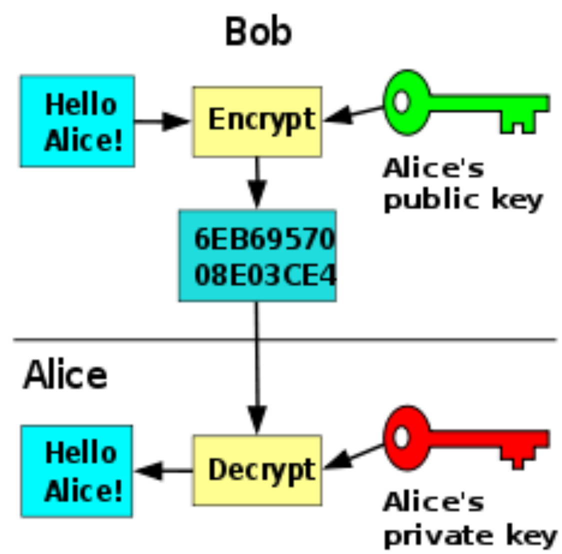




RSA: when does it work?

- keys generation
  - $n=pq$ needs to be very large (e.g. at least 200 digits) so that both the public and private key exponents are large enough.
  - p and q should come out of a “random” process (i.e., not easily guessed).
  - needs an efficient way to check if such generated p and q are indeed primes.
- encryption
  - given large n, e, and any m needs an efficient way of computing $c = m^e \bmod n$
- decryption
  - given large n, d, and any c needs an efficient way of computing $m = c^d \bmod n$
  - given large n, e, should be hard to find d
  - given large n, e, c, should be hard to find m





Modular exponentiation

- Given *n, c, d* calculate $m = c^d \bmod n$
- How?
  - divide and conquer: raising powers with repeated squaring
  - efficient when using the binary representation of *d*
  - *(e.g., d =560 =<1,0,0,0,1,1,0,0,0,0>)*

Modular exponentiation II

- Given n, c, d calculate $m = c^d \bmod n$
- procedure computes $c^i \bmod n$ as i is increased by doublings, incrementing from 0 to d:
  - $i=0; m=1$; let $d = <d_k, d_{k-1}, \ldots, d_0>$
  - for $j=k$ downto 0
    - $i = 2i$
    - $m=m*m \bmod n$
    - if $d_j = 1$
      - » i = i+1
      - » $m=m*c \bmod n$
  - return m

Modular exponentiation III

- Given n, c, d calculate $m = c^d \bmod n$
  - $i=0; m=1$; let $d = <d_k, d_{k-1}, \ldots, d_0>$
  - for $j=k$ downto 0
    - $i = 2i$
    - $m = m*m \bmod n$
    - if $d_j = 1$
      - » i = i+1
      - » $m = m*c \bmod n$
  - return m
- if n, c, d are k-bits number, total number of bit operations is $O(k^3)$

Primality testing

- Given an integer p, is p a prime number?

- Wilson's theorem:

  $\boxed{p \text{ is prime if and only if } p \text{ divides } (p\text{-}1)!+1}$

  – is nice

  – but useless for our purpose ...

  (computing $(p\text{-}1)!+1$ and testing if p divides $(p\text{-}1)!+1$ become computationally prohibitive for large p)

Primality testing I

- Given an integer p, is p a prime number?
- Basic Algorithm:

  "check whether any integer m from 2 to $[\sqrt{p}]$ divides p (skipping even integers). If none of them do, p is prime."
- complexity?
  - $\Theta(\sqrt{p})$
  - exponential in the length of p

Primality testing II

- Given an integer p, is p a prime number?
- Randomization to the rescue !!
- Pseudoprimes
  - def: p is a base-a pseudoprime if p is composite and $a^{p-1} = 1 \bmod p$
- Thm: if p is prime then $a^{p-1} = 1 \bmod p$ for all $1 \leq a \leq p-1$ (from Fermat)
- converse is "almost" true

Primality testing III

- Given an integer p, is p a prime number?
- randomization to the rescue !!
- "pseudo" prime testing:

> – input p:
> – if $2^{p-1} \neq 1 \bmod p$
>   • then return composite // definitely
> – else return prime // we hope ...

Primality testing IV

- input p:
- if $2^{p-1} \neq 1 \bmod p$
  - then return composite // definitely
- else return prime // we hope ...

will make a mistake only if p is a base-2 pseudoprime, and this is "rare" ...

- only 22 values of p less than 10,000 for which it makes a mistake (341, 561, 645 ...)
- probability of a mistake for a randomly chosen 1024-bit number is $\leq 10^{-41}$

Primality testing V

- A randomized testing

  - input p:
  - choose a random number $2 \leq a \leq p-2$
  - if $a^{p-1} \neq 1 \bmod p$
    - then return composite // definitely
  - else return prime // almost surely

Integer factorization

- Given an integer n, decompose it into a product of primes.
- Unless P=NP, this seems to be a computationally hard problem (and a good news to the cryptographers)