Discrete Mathematics
Lecture 4

Liangfeng Zhang
School of Information Science and Technology
ShanghaiTech University

Summary of Lecture 3

Floor: $\lfloor \alpha \rfloor$ is the largest integer $\leq \alpha$ $//a = bq + r;\ q = \lfloor a/b \rfloor$

Residue Class: $[a]_n = a + n\mathbb{Z} = \{a + nx: x \in \mathbb{Z}\}$

- $[a]_n \cap [b]_n = \emptyset$ or $[a]_n = [b]_n$
- $[a]_n = [b]_n$ iff $a \equiv b \pmod{n}$

The Set $\mathbb{Z}_n = \{[0]_n, [1]_n, \ldots, [n-1]_n\}$

- $[a]_n + [b]_n = [a+b]_n$; $[a]_n - [b]_n = [a-b]_n$; $[a]_n \cdot [b]_n = [a \cdot b]_n$
- $[s]_n \in \mathbb{Z}_n$ is called an **inverse** of $[a]_n$ if $[a]_n[s]_n = [1]_n$
  - $[a]_n \in \mathbb{Z}_n$ has an inverse iff $\gcd(a, n) = 1$

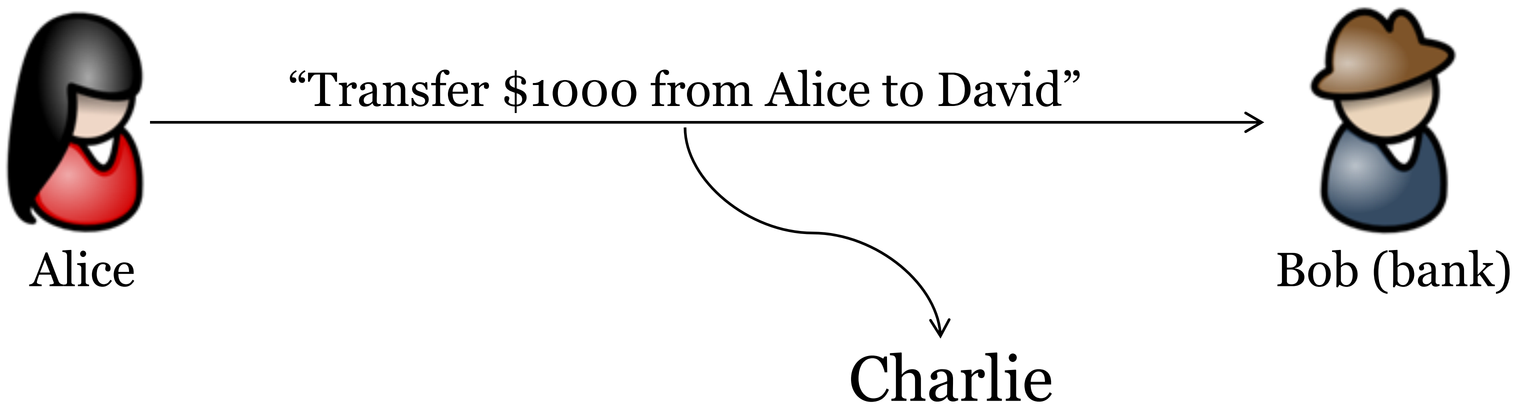
The Set $\mathbb{Z}_n^* = \{[a]_n \in \mathbb{Z}_n: \gcd(a, n) = 1\}$

- **Euler's Phi Function:** $\phi(n) = |\mathbb{Z}_n^*|, \forall n \in \mathbb{Z}^+$
- $n = p_1^{e_1} \cdots p_r^{e_r} \Rightarrow \phi(n) = n(1 - p_1^{-1}) \cdots (1 - p_r^{-1})$

Euler's Theorem Let $n \geq 1$ and $\alpha \in \mathbb{Z}_n^*$. Then $\alpha^{\phi(n)} = 1$.

- **Fermat's Little Theorem**: If p is a prime, then $\alpha^p = \alpha, \forall \alpha \in \mathbb{Z}_p$.

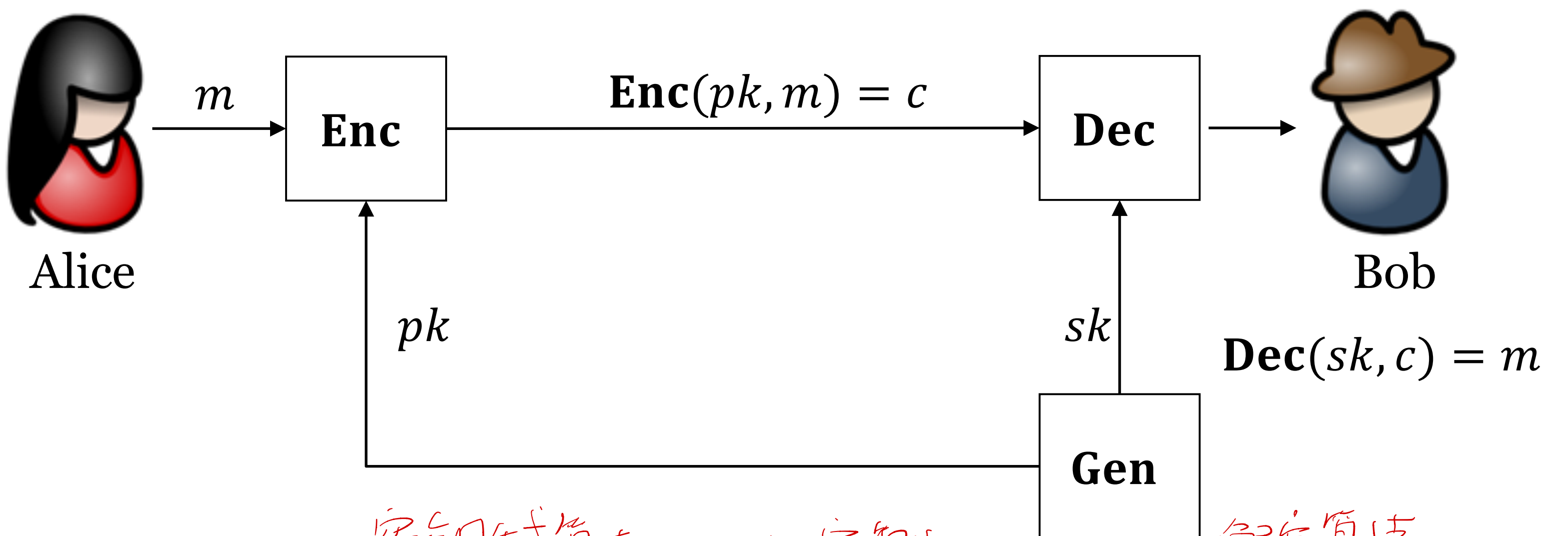
Cryptography



- **Confidentiality**: The property that sensitive information is not disclosed to unauthorized individuals, entities, or processes. --FIPS 140-2

加密

Public-Key Encryption



密钥生成算法 加密算法 解密算法

- **Gen, Enc, Dec**: key generation, encryption, decryption

明文 密文 公钥 私钥

- m, c, pk, sk: plaintext (message), ciphertext, public key, private key

明文空间 密文空间

- $\mathcal{M}, \mathcal{C}$: plaintext space, ciphertext space
- $\Pi = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec}) + \mathcal{M}, |\mathcal{M}| > 1$

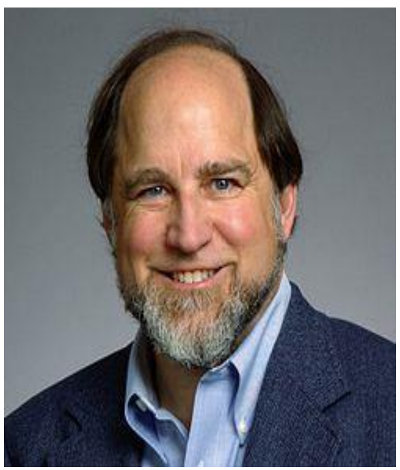
正确性 c 都指定的

  - **Correctness**: $\mathbf{Dec}\big(sk, \underbrace{\mathbf{Enc}(pk, m)}_{c}\big) = m$ for any $\boxed{pk, sk}, m$
  - **Security**: if sk is not known, it's difficult to learn m from pk, c

RSA

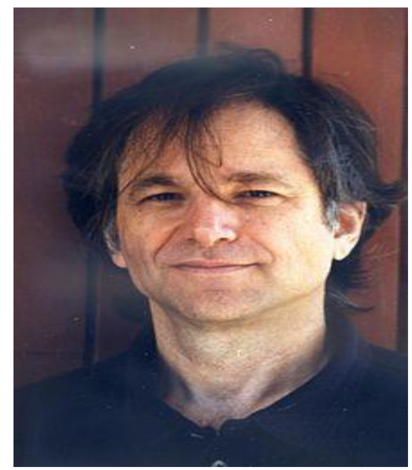
A method for obtaining digital signatures and public-key cryptosystem

- Ronald Rivest, Adi Shamir and Leonard Adleman (1977)-MIT
- Scientific Contributions: Turing Award (2002)
  - Public-Key Encryption: the first construction
  - Digital Signature: the first construction



Rivest

Shamir



Adleman

Plain RSA

CONSTRUCTION: $\Pi = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec}) + \mathcal{M}$, the message space is $\mathcal{M} = \{m : m \in \{1,2,\ldots,N\}, \gcd(m,N) = 1\}$ 即 $\mathbb{Z}_N^*$

- $(pk, sk) \leftarrow \mathbf{Gen}(1^n)$ 安全参数 $1^n: \overbrace{111\cdots11}^{n个1}$

  生成的公钥与私钥

  输入

  - choose two n-bit primes $p \neq q$; 最大 2^n-1
  - $N = pq$; $\phi(N) = (p-1)(q-1)$
  - $[e]_{\phi(N)} \leftarrow \mathbb{Z}^*_{\phi(N)}$
  - ∃d s.t. $[d]_{\phi(N)} = \left([e]_{\phi(N)}\right)^{-1}$ 即 $[d]_{\varphi(N)} \cdot [e]_{\varphi(N)} = [1]_{\varphi(N)}$
    - $0 \leq e, d < \phi(N)$
  - output $pk = (N, e)$ and $sk = (N, d)$
- $c \leftarrow \mathbf{Enc}(pk, m)$:
  - output $c = m^e \bmod N$
    - $0 \leq c < N$
- $m \leftarrow \mathbf{Dec}(sk, c)$:
  - output $m = c^d \bmod N$
    - $0 \leq m < N$

$\mathbf{Dec}(sk, \mathbf{Enc}(pk, m)) = m$

- $[d]_{\phi(N)} = \left([e]_{\phi(N)}\right)^{-1}$
- $\exists t \in \mathbb{Z}$ s.t. $ed = 1 + t \cdot \phi(N)$
- $$\begin{aligned}[c^d]_N &= ([c]_N)^d \\ &= ([m^e]_N)^d \\ &= \left(([m]_N)^e\right)^d \\ &= ([m]_N)^{ed} \\ &= ([m]_N)^{1+t\phi(N)} \\ &= [m]_N \cdot ([m]_N)^{\phi(N)t} \\ &= [m]_N \cdot [1]_N \\ &= [m]_N\end{aligned}$$
- $m = c^d \bmod N$

Plain RSA

EXAMPLE: this is a toy example; all numbers are very small

- $(pk, sk) \leftarrow \mathbf{Gen}(1^n)$
  - $p = 7, q = 13,$
  - $N = 91, \phi(N) = 72$
  - $[e]_{72} = [5]_{72}$ $e \in N^*_{\varphi(N)}$
  - $[d]_{72} = [29]_{72}$ $[e]_{\varphi(N)}[d]_{\varphi(N)} = [1]_{\varphi(N)}$
  - $pk = (91, 5);\ sk = (91, 29)$
- $c \leftarrow \mathbf{Enc}(pk, m)$:
  - $m = 2$
  - $c = (2^5 \bmod 91) = 32$
- $m \leftarrow \mathbf{Dec}(sk, c)$:
  - $c = 32$
  - $m = (32^{29} \bmod 91) = 2$

- $32^{29} = (2^5)^{29} = 2^{145}$
- $2^{145} \equiv ?\ (\text{mod } 91)$
- $[2^{145}]_{91} = [?]_{91}$
- $([2]_{91})^{145} = [?]_{91}$
- $[2]_{91} \in \mathbb{Z}^*_{91}$
- $([2]_{91})^{\phi(91)} = [1]_{91}$
- $([2]_{91})^{145} = ([2]_{91})^{72}([2]_{91})^{72}[2]_{91}$

$$= [1]_{91}[1]_{91}[2]_{91}$$

$$= [2]_{91}$$

Security

Security: If sk is not known, it's difficult to learn m from pk, c

- At least, it should be difficult to learn d from pk

Plain RSA and Integer Factoring (given N, find p, q**):**

- "Factoring is easy" ⇒ "Plain RSA is not secure"
  - $N \rightarrow (p, q) \rightarrow \phi(N) \rightarrow d$: computable with EEA
- "Plain RSA is secure" ⇒ "Factoring is hard"
- "Factoring is hard"⇏ "Plain RSA is secure"
- It is likely that "Factoring is hard"⇒ "Plain RSA is secure"
  - The best known method of computing d is via factoring N

How Large is the N in practice?

- $|N| = 2048$ is recommended from present to 2030
- $|N| = 3072$ is recommended after 2030

RSA

EXAMPLE: A sample execution of the RSA public-key encryption.

- p =179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624225795083
- q =179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624227077847
- N =32317006071311007300714876688669951960444102669715484032130345427524655138867890893197201411522913463688717960921898019494119559150490921095088152386448283120630877367300996091750197750389652106796057638384067568276792218642619756161838094338476170470581645852036305042887575891541065808607552399123931212190742861198666048560131098081430518774846347259215332611759149330725252437276424147817808729273755165527379964561074264587032664709511346018327798373715290148129504141795132314929388992688247440232727539575514688633282447719228530664706520939357878528540284184156513405575872085703420500969966917951381310826301
- $\phi(N)$ =32317006071311007300714876688669951960444102669715484032130345427524655138867890893197201411522913463688717960921898019494119559150490921095088152386448283120630877367300996091750197750389652106796057638384067568276792218642619756161838094338476170470581645852036305042887575891541065808607552399123931212190383322571693585378585237043272713828122751863426871297212289168409787085665422221552391774628940093485139736801331477871715085171882512773342103512436341899373968354945401344376784553485755251993821373671344677095606146354543604901758694718276224054213583162787340809095977593826461068360296205292132857953372

RSA

EXAMPLE: A sample execution of the RSA public-key encryption.

- e =15
- d =4308934142841467640095316891822660261392547022628731204284046057003287351849052119092960
188203055128491829061456253069265882607886732122812678420318193104416084116982306799478900026366718620280906141018451209009103572295819015967488245079245130156062744219446938174005718343452205475441147673653216524161625384443009559144717144698272436361843749700248456916172961638555787971611422056296206985569950525345798018631573510863716228678022917668369778947134991512253249862447326053512583571273798100700265842849822845956946080819513939147320234492629103496540561811088371645441212797012510194809114706160705617714393783
- m =106049217547587214457616546941448530089527776082804376150454723656215287406799155692700515031915225000364485571724879590119261120383983594027565731495416443309686417676306220707206300611302597838253559482233713309491580368127421870570456049345468117909489758782001441890483442498732003202992772344656890394099896223192326839842418437111832120019914577935287528129781340727874047902070314820994449682521086902963637735785947031026173867382976750802957740914472401975212215460354590300865381144285160786447331806555401091337782416072602736553356617778941736651379287879603652207120251207852579072445617216927647552103758
- c =105263899581389629195955940934111588930997435084659023471284781399087746143117780973547953457917267683842527516376939955924037578561854370837388298360724722433895833679102687994533780394197213455665495167301873084368644600883966117266700507232420801391760803347202941953040489150038056563418165483072498860490279104882493186600627143357030575765760169885134841483085125749502525354631858248656654997490335982013703421429019446325492535640376393124428750397358269093293568406659937836951014476104859227269159699679685846612404304259821941895044004698897625742758242694754953949201079210667232777692261994755580686270497

Implementation Issues

CONSTRUCTION: $\Pi = (\mathbf{Gen}, \mathbf{Enc}, \mathbf{Dec}) + \mathcal{M}$, the message space is $\mathcal{M} = \{m : m \in [N], \gcd(m, N) = 1\}$

- $(pk, sk) \leftarrow \mathbf{Gen}(1^n)$
  - choose two n-bit primes $p \neq q$
  - $N = pq$; $\phi(N) = (p-1)(q-1)$
  - $[e]_{\phi(N)} \leftarrow \mathbb{Z}^*_{\phi(N)}$
  - $[d]_{\phi(N)} = \left([e]_{\phi(N)}\right)^{-1}$
    - $0 \leq e, d < \phi(N)$
  - output $pk = (N, e)$ and $sk = (N, d)$
- $c \leftarrow \mathbf{Enc}(pk, m)$:
  - output $c = m^e \bmod N$
    - $0 \leq c < N$
- $m \leftarrow \mathbf{Dec}(sk, c)$:
  - output $m = c^d \bmod N$
    - $0 \leq m < N$

Questions

- Choose p, q efficiently?
  - Prime number generation
- Compute d efficiently?
  - Square-and-multiply
- Compute c/m efficiently?
  - Square-and-multiply

Addition

Bit Length of Integer: $\ell(a) = \begin{cases} \lfloor \log_2(|a|) \rfloor + 1 & a \neq 0 \\ 1 & a = 0 \end{cases}$

Binary Representation: a 0-1 sequence

- $a = (a_{k-1} \dots a_1 a_0)_2 \Leftrightarrow a = a_{k-1} 2^{k-1} + \cdots + a_1 \cdot 2^1 + a_0 \cdot 2^0$

Algorithm for Addition:

- **Input**: $a = (a_{k-1} \cdots a_1 a_0)_2, b = (b_{k-1} \cdots b_1 b_0)_2$
- **Output**: $c = a + b = (c_k c_{k-1} \cdots c_1 c_0)_2$
  - $carry \leftarrow 0$
  - for $i \leftarrow 0$ to $k - 1$ do
    - $t \leftarrow a_i + b_i + carry$;
    - set c_i and $carry$ such that $t = 2 \cdot carry + c_i$
  - $c_k \leftarrow carry$
- **Complexity**: $O(k)$ bit operations

Subtraction

Algorithm for Subtraction:

- **Input**: $a = (a_{k-1} \cdots a_1 a_0)_2, b = (b_{k-1} \cdots b_1 b_0)_2, a \geq b$
- **Output**: $c = a - b = (c_{k-1} \cdots c_1 c_0)_2$
  - $carry \leftarrow 0$
  - for $i \leftarrow 0$ to $k - 1$ do
    - $t \leftarrow a_i - b_i + carry$;
    - set c_i and $carry$ such that $t = 2 \cdot carry + c_i$
- **Complexity**: $O(k)$ bit operations

Multiplication

Algorithm for Multiplication:

- **Input**: $a = (a_{k-1} \cdots a_0)_2, b = (b_{k-1} \cdots b_0)_2$
- **Output**: $c = ab = (c_{2k-1} \cdots c_0)_2$
  - $c \leftarrow 0; x \leftarrow a$
  - for $i \leftarrow 0$ to $k - 1$ do
    - if $b_i = 1$, then $c \leftarrow c + x$;
    - $x \leftarrow x + x$;
- **Complexity**: $O(k^2)$ bit operations

Division

Algorithm for Division:

- **Input**: $a = (a_{k-1} \cdots a_0)_2, b = (b_{l-1} \cdots b_0)_2, a \geq b, a_{k-1} = b_{l-1} = 1$
- **Output**: $q = (q_{k-l} \cdots q_0)_2$ and $r = (r_{l-1} \cdots r_0)_2$ s.t. $a = bq + r, 0 \leq r < b$
  - $(r_k r_{k-1} \cdots r_0)_2 \leftarrow (0 a_{k-1} \cdots a_0)_2$
  - for $i \leftarrow k - l$ down to 0 do
    - $q_i \leftarrow 2r_{i+l} + r_{i+l-1}$;
    - If $q_i \geq 2$, then $q_i \leftarrow 1$;
    - $(r_{i+l} \cdots r_i)_2 \leftarrow (r_{i+l} \cdots r_i)_2 - q_i \cdot b$;
    - while $r_{i+l} < 0$ do
      - $(r_{i+l} \cdots r_i)_2 \leftarrow (r_{i+l} \cdots r_i)_2 + b$;
      - $q_i \leftarrow q_i - 1$;
  - output $q = (q_{k-l} \cdots q_0)_2$ and $r = (r_{l-1} \cdots r_0)$;
- **Complexity**: $O((k - l + 1) \cdot l)$ bit operations

Arithmetic Modulo n

THEOREM: Let $a, b \in \{0,1,\dots,n-1\}$. Then

- $(a \pm b) \bmod n$ can be computed in $O(\ell(n))$ bit operations
  - $\ell(a), \ell(b) \leq \ell(n)$
    - $a \pm b$ are computable in $O(\ell(n))$ bit operations
  - $0 \leq |a+b|, |a-b| < 2n$
    - $(a \pm b) \bmod n$ are computable in $O((\ell(2n) - \ell(n) + 1)\ell(n)) = O(\ell(n))$ bit operations
- $(ab) \bmod n$ can be computed in $O(\ell(n)^2)$ bit operations
  - $\ell(a), \ell(b) \leq \ell(n)$
    - ab is computable in $O(\ell(n)^2)$ bit operations
  - $0 \leq |ab| < n^2$
    - $(ab) \bmod n$ is computable in $O\big((\ell(n^2) - \ell(n) + 1)\ell(n)\big) = O(\ell(n)^2)$ bit operations.

Arithmetic Modulo n

Modulo exponentiation: For $0 \le a < n, e \in \mathbb{N}$, $a^e \bmod n = ?$

- Complexity? How to compute efficiently?

EXAMPLE: modulo exponentiation

- m =1437339113920498981637906207424471163445460406448981415203760376263650078098 9961566579389511210479437355107978772736352915127780140263030574243344234098335878739419385503392646991360376271216372316046211564902 5
- e =4631001162549482394344687394431824369029736722768833120796257387139181880015661440418125399478543429257625536255388418199849246329730346646442802201832756472381022836757671552531962337198345690506439249417678 5
- N =2452466449002782119765176635730880184670267876783327597434144517150616008300385872169522083993320715491036268271916798640797767232430056005920356312465612184658179041001318592996199338170121493350348758705510 67
- $\phi(N)$ =245246644900278211976517663573088018467026787678332759743414451715061600830038587216952208399332071549102628322861627039184220494270313938703906283392288487724394251766892786817697178343799758481228648091667216
- $m^e \bmod N = ?$

Square-and-Multiply

ALGORITHM: compute $a^e \bmod n$ in polynomial time

- **Input:** $a \in \{0,1,\ldots,n-1\}$; $e = (e_{k-1} \cdots e_0)_2$ $\quad //k = \ell(e)$
  - $e = e_{k-1} \cdot 2^{k-1} + \cdots + e_1 \cdot 2^1 + e_0 \cdot 2^0$
- **Output:** $a^e \bmod n$
  - **Square**: this step requires $O(k)$ multiplications modulo n
    - $x_0 = a$
    - $x_1 = (x_0^2 \bmod n) = (a^2 \bmod n)$
    - $x_2 = (x_1^2 \bmod n) = (a^{2^2} \bmod n)$
    - …
    - $x_{k-1} = (x_{k-2}^2 \bmod n) = (a^{2^{k-1}} \bmod n)$
  - **Multiply**: this step requires $O(k)$ multiplications modulo n
    - $(a^e \bmod n) = (x_0^{e_0} \cdot x_1^{e_1} \cdots x_{k-1}^{e_{k-1}} \bmod n)$
- **Complexity**: $O(k)$ multiplications modulo n

Square-and-Multiply

EXAMPLE: Compute 2^{123} mod 35 using square-and-multiply.

- **Input**: $a = 2; n = 35;\ e = 123 = (1\ 1\ 1\ 1\ 0\ 1\ 1)_2;\ k = 7$
- **Square:** $k - 1$ multiplications modulo n will be done
  - $x_0 = a = 2;$
  - $x_1 = x_0^2 \bmod n = 4$
  - $x_2 = x_1^2 \bmod n = 16$
  - $x_3 = x_2^2 \bmod n = 11$
  - $x_4 = x_3^2 \bmod n = 16$
  - $x_5 = x_4^2 \bmod n = 11$
  - $x_6 = x_5^2 \bmod n = 16$
- **Multiply:** at most $k - 1$ multiplications modulo n will be done
  - $a^e = x_0 x_1 x_3 x_4 x_5 x_6 = 2 \times 4 \times 11 \times 16 \times 11 \times 16 \equiv 8 \pmod{35}$
  - $(2^{123} \bmod 35) = 8$

Euclidean Algorithm (EA)

ALGORITHM: compute $\gcd(a, b)$

- **Input**: a, b $(a \geq b > 0)$
- **Output**: $d = \gcd(a, b)$
  - $r_0 = a; r_1 = b;$
  - $r_0 = r_1 q_1 + r_2 \;\; (0 < r_2 < r_1)$
  - $\vdots$
  - $r_{i-1} = r_i q_i + r_{i+1} \;\; (0 < r_{i+1} < r_i)$
  - $\vdots$
  - $r_{k-2} = r_{k-1} q_{k-1} + r_k \; (0 < r_k < r_{k-1})$
  - $r_{k-1} = r_k q_k$
  - output r_k

$a = 12345, b = 123$		
i	r_i	q_i
0	12345	
1	123	100
2	45	2
3	33	1
4	12	2
5	9	1
6	3	3
7	0	

Correctness: $d = \gcd(r_0, r_1) = \cdots = \gcd(r_{k-1}, r_k) = r_k$

Extended Euclidean Algorithm (EEA)

ALGORITHM: compute $d = \gcd(a, b)$, s, t such that $as + bt = d$

- **Input**: a, b $(a \geq b > 0)$
- **Output**: $d = \gcd(a, b)$, integers s, t such that $d = as + bt$
    - $r_0 = a; r_1 = b; \begin{pmatrix} s_0 \\ t_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \begin{pmatrix} s_1 \\ t_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix};$
    - $r_0 = r_1 q_1 + r_2 \ (0 < r_2 < r_1); \begin{pmatrix} s_2 \\ t_2 \end{pmatrix} = \begin{pmatrix} s_0 \\ t_0 \end{pmatrix} - q_1 \begin{pmatrix} s_1 \\ t_1 \end{pmatrix}$
    - $\vdots$
    - $r_{i-1} = r_i q_i + r_{i+1} \ (0 < r_{i+1} < r_i); \begin{pmatrix} s_{i+1} \\ t_{i+1} \end{pmatrix} = \begin{pmatrix} s_{i-1} \\ t_{i-1} \end{pmatrix} - q_i \begin{pmatrix} s_i \\ t_i \end{pmatrix}$
    - $\vdots$
    - $r_{k-2} = r_{k-1} q_{k-1} + r_k \ (0 < r_k < r_{k-1}); \begin{pmatrix} s_k \\ t_k \end{pmatrix} = \begin{pmatrix} s_{k-2} \\ t_{k-2} \end{pmatrix} - q_{k-1} \begin{pmatrix} s_{k-1} \\ t_{k-1} \end{pmatrix}$
    - $r_{k-1} = r_k q_k$
    - output r_k, s_k, t_k

EEA

Correctness: We have that $r_i = as_i + bt_i$ for $i = 0,1,2,\dots,k$

- $r_0 = a = (a,b)\binom{s_0}{t_0}$; $r_1 = b = (a,b)\binom{s_1}{t_1}$;
- $r_2 = r_0 - q_1 r_1 = (a,b)\binom{s_0}{t_0} - q_1 \cdot (a,b)\binom{s_1}{t_1} = (a,b)\binom{s_2}{t_2}$;
- $\vdots$
- $r_k = r_{k-2} - q_{k-1} r_{k-1} = (a,b)\binom{s_{k-2}}{t_{k-2}} - q_{k-1} \cdot (a,b)\binom{s_{k-1}}{t_{k-1}} = (a,b)\binom{s_k}{t_k}$

EXAMPLE: Execution of the EEA on input $a = 12345, b = 123$

i	r_i	q_i	s_i	t_i
0	12345		1	0
1	123	100	0	1
2	45	2	1	−100
3	33	1	−2	201
4	12	2	3	−301
5	9	1	−8	803
6	3	3	11	−1104
7	0			

Complexity

THEOREM: Let $\alpha = \frac{1}{2}(1+\sqrt{5})$. Then $k \leq \ln b / \ln \alpha + 1$ in EA.

- $k = 1$: $k \leq \ln b / \ln \alpha + 1$
- $k > 1$: we show that $r_{k-i} \geq \alpha^i$ for $i = 0,1,\ldots,k-1$
  - $i = 0$: $r_k \geq 1 = \alpha^0$
  - $i = 1$: $r_{k-1} > r_k \Rightarrow r_{k-1} \geq r_k + 1 \geq 2 \geq \alpha^1$
  - Suppose that $r_{k-i} \geq \alpha^i$ for $i \leq j$
    - $$\begin{aligned} r_{k-(j+1)} &= r_{k-j} q_{k-j} + r_{k-(j-1)} \\ &\geq \alpha^j + \alpha^{j-1} \\ &= \alpha^{j-1}(\alpha+1) \\ &= \alpha^{j+1} \end{aligned}$$
- $b = r_1 \geq \alpha^{k-1} \Rightarrow k \leq \ln b / \ln \alpha + 1$

Complexity of EA and EEA: $O(\ell(a)\ell(b))$ bit operations

Prime Number Theorem

DEFINITION: For $x \in \mathbb{R}^+$, $\pi(x) = \sum_{p \le x} 1$: # of primes $\le x$

THEOREM: $\lim_{x \to \infty} \pi(x)/(x/\ln x) = 1$

- Conjectured by Legendre and Gauss
- Chebyshev: if the limit exists, then it is equal to 1
- Rosser and Schoenfeld:
  - $\pi(x) > \frac{x}{\ln x}(1 + \frac{1}{2\ln x})$ when $x \ge 59$
  - $\pi(x) < \frac{x}{\ln x}(1 + \frac{3}{2\ln x})$ when $x > 1$

NOTATION: $\mathbb{P}$- the set of all primes; $\mathbb{P}_n = \{p \in \mathbb{P}: 2^{n-1} \le p < 2^n\}$.

THEOREM: $|\mathbb{P}_n| \ge \frac{2^n}{n \ln 2}\left(\frac{1}{2} + O\left(\frac{1}{n}\right)\right)$ when $n \to \infty$.

Number of n-bit Primes

EXAMPLE: The number of n-bit primes for $n \in \{10, \ldots, 25\}$.

n	$\lvert\mathbb{P}_n\rvert$	$2^{n-1}/n \ln 2$	n	$\lvert\mathbb{P}_n\rvert$	$2^{n-1}/n \ln 2$
10	75	73.8	18	10749	10505.4
11	137	134.3	19	20390	19904.9
12	255	246.2	20	38635	37819.4
13	464	454.6	21	73586	72036.9
14	872	844.2	22	140336	137525.0
15	1612	1575.8	23	268216	263091.4
16	3030	2954.6	24	513708	504258.5
17	5709	5561.7	25	985818	968176.3

Prime Number Generation

Basic Idea: randomly choose n-bit integers until a prime found.

- The number of n-bit integers is 2^{n-1}
- $|\mathbb{P}_n| \geq \frac{2^n}{n \ln 2}\left(\frac{1}{2} + O\left(\frac{1}{n}\right)\right)$ when $n \to \infty$
- The probability that a prime is chosen in every trial is equal to
$$\alpha_n = \frac{1}{n \ln 2}\left(1 + O\left(\frac{1}{n}\right)\right), n \to \infty$$
- In $\alpha_n^{-1} = \frac{n \ln 2}{1+O\left(\frac{1}{n}\right)} \leq 2n \ln 2$ trials, we get a prime.

Efficient Algorithms: An algorithm is considered as efficient if its (expected) running time is a polynomial in the bit length of its input. //a.k.a. (expected) polynomial-time algorithm

EXAMPLE: Choosing an n-bit prime can be done efficiently.

- The expected # of trials is $\leq 2n \ln 2$, a polynomial in n (input length)
- Determine if an n-bit integer is prime can be done efficiently