The obvious mathematical breakthrough would be development of an easy way to factor large prime numbers.

--- Bill Gates, *The Road Ahead*, page 265

**Public-key encryption**

- general principles
- RSA cryptosystem
  - operation
  - properties of the textbook RSA
  - PKCS#1
- ElGamal cryptosystem

---

**Reminder**

- asymmetric-key encryption
  - it is hard (computationally infeasible) to compute $k'$ from $k$
  - $k$ can be made public (public-key cryptography)

- public-keys are not confidential but they must be authentic!
- most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric key schemes

© Levente Buttyán
Digital enveloping

plaintext message

symmetric-key cipher (e.g., in CBC mode)

generate random symmetric key

public key of the receiver

asymmetric-key cipher

bulk encryption key

digital envelop

Two important complexity classes

- **class P**: problems solvable with an algorithm that is deterministic and p-time bounded
  - asymptotic worst case complexity is a polynomial function of the input length n

- **class NP**: problems solvable with an algorithm that is non-deterministic and run in p-time on a non-deterministic machine
  - problems in NP have no known deterministic p-time algorithms
  - asymptotic worst case complexity of the most efficient algorithms known is often an exponential function of the input length n
  - however, a solution to an NP problem can be verified in p-time on a deterministic machine

- it is conjectured that P ≠ NP, but it has not been proven yet
Examples

- **factoring problem**
  - given a positive integer \( n \), find its prime factors
    - true complexity is unknown
    - it is believed that it does not belong to \( P \)

- **discrete logarithm problem**
  - given a prime \( p \), a generator \( g \) of \( \mathbb{Z}_p^* \), and an element \( y \) in \( \mathbb{Z}_p^* \), find
    the integer \( x \), \( 0 \leq x \leq p-2 \), such that \( g^x \mod p = y \)
    - true complexity is unknown
    - it is believed that it does not belong to \( P \)

- **Diffie-Hellman problem**
  - given a prime \( p \), a generator \( g \) of \( \mathbb{Z}_p^* \), and elements \( g^x \mod p \) and
    \( g^y \mod p \), find \( g^{xy} \mod p \)
    - true complexity is unknown
    - it is believed that it does not belong to \( P \)

### RSA (Rivest–Shamir–Adleman) cryptosystem

- **key generation**
  - select \( p, q \) large primes (about 500 bits each)
  - \( n = pq \), \( \phi(n) = (p-1)(q-1) \)
  - select \( e \) such that \( 1 < e < \phi(n) \) and \( \gcd(e, \phi(n)) = 1 \)
  - compute \( d \) such that \( ed \mod \phi(n) = 1 \) (this is easy if \( \phi(n) \) is known)
    - the public key is \((e, n)\)
    - the private key is \( d \)

- **encryption**
  - represent the message as an integer \( m \) in \([0, n-1]\)
  - compute \( c = m^e \mod n \)

- **decryption**
  - compute \( m = c^d \mod n \)
Proof of RSA decryption

- \( c^d \mod n = m^e \mod n = m^{k \phi(n) \cdot 1} \mod n = m^{k(p-1)(q-1)} \mod n \)
- since \( m < n \), it is enough to prove that \( m^{k(p-1)(q-1)} = m \mod n \)
- Fermat theorem
  - if \( r \) is a prime and \( \gcd(a, r) = 1 \), then \( a^{r-1} = 1 \mod r \)
- if \( \gcd(m, p) = 1 \)
  - \( m^{p-1} = 1 \mod p \)
  - \( m^{k(p-1)(q-1)} = m \mod p \)
- if \( \gcd(m, p) = p \)
  - \( p \mid m \)
  - \( m^{k(p-1)(q-1)} = m = 0 \mod p \)
- for all \( m \), \( m^{k(p-1)(q-1)} = m \mod p \)
- similarly, for all \( m \), \( m^{k(p-1)(q-1)} = m \mod q \)
- \( p, q \mid m \)
  - \( m^{k(p-1)(q-1)} = m \mod pq \)

Implementing RSA – Computing \( d \)

- \( d \) can be computed using the extended Euclidean algorithm
- complexity:
  - let \( k \) be the length of \( n \) in bits (\( k = \lceil \log_2 n \rceil + 1 \))
  - adding two \( k \)-bit integers: \( O(k) \)
  - multiplication of two \( k \)-bit integers: \( O(k^2) \)
  - reduction modulo \( n \) of a \( 2k \)-bit integer: \( O(k^2) \)
  - modular multiplication of two \( k \)-bit integers: \( O(k^2) \)
  - complexity of each step of the Euclidean algorithm: \( O(k^2) \)
  - number of iterations in the Euclidean algorithm: \( O(k) \)
  - complexity of computing \( d \): \( O(k^2) \)
Implementing RSA – Modular exponentiation

- naive approach:
  - \( m^n \mod n = \underbrace{m \cdot m \cdot m \ldots m}_{\text{mod } n} \)
  - complexity of \( x \)-1 modular multiplication is \( O(xk^2) \)
  - unfortunately \( x \) can be as big as \( \phi(n) \)-1, hence \( x \sim O(n) = O(2^k) \)
  - complexity of the naive approach is \( O(2^k) \)

there’s a better method for modular exponentiation

- \( x = b_k 2^{k-1} + b_{k-2} 2^{k-2} + \ldots + b_2 + b_0 \)
- \( m^x = \underbrace{m^0(m^x)^2}_x \) where \( x_0 = (x-b_0)/2 = b_k 2^{k-1} + b_{k-2} 2^{k-2} + \ldots + b_1 \)
- \( m^x = \underbrace{m^1(m^x)^2}_x \) where \( x_1 = (x_0-b_1)/2 = b_{k-1} 2^{k-3} + b_{k-2} 2^{k-4} + \ldots + b_2 \)
- \( \ldots \)
- \( m^{x_k} = \underbrace{m^{k-3}(m^{x_k})^2}_{x_k} \) where \( x_k = (x_{k-2}-b_{k-2})/2 = b_{k-1} \)
- \( m^{x_{k-2}} = \underbrace{m^{k-2}(m^{x_{k-2}})^2}_{x_{k-2}} \) where \( x_{k-2} = (x_{k-3}-b_{k-3})/2 = b_{k-2} \)
- \( m^{x_{k-1}} = \underbrace{m^{k-1}}_{x_{k-1}} \)

- “square and multiply” algorithm
  
  \[
  \begin{cases}
    c = 1 \\
    \text{for } i = k-1 \text{ to } 0 \text{ do} \\
    \quad c = c^2 \mod n \\
    \quad \text{if } b_i = 1 \text{ then } c = c \cdot m \mod n \\
    \text{end for} \\
    \text{output } c = m^x \mod n
  \end{cases}
  \]

- complexity:
  - \( k \) modular squaring (multiplication)
  - at most \( k \) modular multiplication
  - complexity of the clever approach is \( O(kk^2) = O(k^3) \)
RSA toy example

- key generation
  - let \( p = 73, q = 151 \)
  - \( n = 73 * 151 = 11023 \)
  - \( \phi(n) = 72*150 = 10800 \)
  - let \( e = 11 \)
  - compute \( d \) with the extended Euclidean algorithm as follows:
    
    \[
    \begin{align*}
    10800 &= 981 * 11 + 9 \\
    t_2 &= 0 - 981x1 \mod 10800 = 9819 \\
    t_1 &= 1 - 1x9819 \mod 10800 = 982 \\
    t_0 &= 9819 - 4x982 = 5891 \\
    \end{align*}
    \]
  - public key is \((11, 11023)\), private key is \(5891\)

- encryption
  - let \( m = 17 \)
  - we compute \( c \) with the "square and multiply" algorithm as follows:
    
    \[
    \begin{align*}
    e &= 11 = 1011 \text{ (in binary)} \\
    c &= 1 \\
    b_3 &= 1 \rightarrow c &= c \cdot m \mod n = 17 \\
    b_2 &= 0 \rightarrow c &= c^2 \mod n = 289 \\
    b_1 &= 1 \rightarrow c &= c \cdot m \mod n = 1419957 \mod 11023 = 8913 \\
    b_0 &= 1 \rightarrow c &= c^2 \mod n = 1419957^2 \mod 11023 = 1782 \\
    \text{output } c &= 1711 \mod 11023 = 1782 \\
    \end{align*}
    \]

- decryption
  - \( d = 5891 = 1011100000011 \) (in binary)
  - we compute \( m = c^d \mod n \) with the "square and multiply" algorithm as above

Implementing RSA – Primality testing

- what is the probability of the event that a randomly selected large integer is prime?
  - prime number theorem:
    number of primes smaller than \( n \) is approximately \( \Pi(n) \sim n/\ln(n) \)
  - corollary:
    probability that a randomly selected \( k \)-bit long integer is prime is
    \[
    \frac{\Pi(2^k) - \Pi(2^{k-1})}{2^k - 2^{k-1}} \sim \frac{1}{(k-1)\ln(2)}
    \]
  - example:
    \( k = 512 \), probability is \( 1/354 = 0.0028 \)
    if we consider only randomly selected odd integers, then the probability is \( 1/177 \)

- how can we know if a given integer is prime or not?
  - PRIME is in \( P \) (there is a polynomial time deterministic decision algorithm)
  - in practice, people use probabilistic primality testing algorithms
### Implementing RSA – Fermat-test

- **Fermat theorem:**
  - if p prime and gcd(b, p) = 1, then $b^{p-1} \equiv 1 \pmod{p}$

- a composite number n is pseudo-prime for a base b if
  - $b^{n-1} \equiv 1 \pmod{n}$
  - where $1 < b < n$ and gcd(b, n) = 1

- **testing approach**
  - choose a random base b, and check if $b^{n-1} \equiv 1 \pmod{n}$ holds
  - if not, then n is composite
  - if yes, then n may be prime and we need to test it further with other bases
  - if n passes the test for many bases, then we accept it as a prime
  - this is a Monte Carlo algorithm
    - the algorithm always gives an answer
    - the answer may be wrong with some probability ε

- **what is the probability of a false answer?**

---

### Implementing RSA – Fermat-test

- **bad news:**
  - there exist composite numbers that always pass the Fermat-test (for every possible base)
  - these are called Carmichael-numbers, and they are quite rare
  - example: 561

- **good news:**
  - if n is composite and not a Carmichael number, then n passes the test for at most half of the possible bases
  - if we run T tests, and n passes all of them, then the probability of error is upper bounded by $2^{-T}$
  - error probability can be made arbitrarily low
Relation to factoring

- the problem of computing $d$ from $(e, n)$ is computationally equivalent to the problem of factoring $n$
  - if one can factor $n$, then he can easily compute $d$
  - if one can compute $d$, then he can efficiently factor $n$
- the problem of computing $m$ from $c$ and $(e, n)$ (RSA problem) is believed to be computationally equivalent to factoring
  - if one can factor $n$, then he can easily compute $m$ from $c$ and $(e, n)$
  - there's no formal proof for the other direction
- given the latest progress in developing algorithms for factoring, the size of the modulus should at least be 1024 bits

Problems - Unconcealed messages

- a message is unconcealed if it encrypts to itself (i.e., if $m^e \mod n = m$)
- trivial examples for unconcealed messages are $m = 0$, $m = 1$, and $m = n-1$
- the exact number of unconcealed messages is $(1 + \gcd(e-1, p-1))(1 + \gcd(e-1, q-1))$
  - if $p$, $q$, and $e$ are selected at random (or $e$ is small such as $e = 3$), then the number of unconcealed messages is negligibly small
Problems - Small encryption exponent $e$

- To improve efficiency of encryption, it is desirable to select a small exponent $e$ (e.g., $e = 3$ is typical).
- A group of entities may use the same exponent, but different moduli (e.g., $e = 3$, and $n_1, n_2, ...$).
- In this case, an attacker may find a plaintext $m$ efficiently, if $m$ is sent to several (at least 3) recipients:
  - Assume that the attacker observes $c_i = m^e \mod n_i$ ($i = 1, 2, 3$).
  - Let $x = m^e$.
  - The attacker must solve for $x$ the following system of congruences:
    - $x \equiv c_1 \mod n_1$.
    - $x \equiv c_2 \mod n_2$.
    - $x \equiv c_3 \mod n_3$.
  - Chinese remainder theorem: If $n_1, n_2, ..., n_k$ are pairwise relatively primes, then such a system has a unique solution $(\mod n_1 \cdot n_2 \cdot \ldots \cdot n_k)$.
  - Since $m^e < n_1 \cdot n_2 \cdot n_3$, the solution found must be $m^e$.
  - The attacker then computes the cube root of $m^e$ to get $m$.

Salting

- Appending a (pseudo) random bit string to the plaintext prior to encryption.
- Salting is a solution to the small exponent problem:
  - Even if the same message $m$ has to be sent to many recipients, the actual plaintext that is encrypted will be different for everyone due to salting.
- Another problem of small exponents where salting helps:
  - If $m < n^{1/e}$, then $m^e < n$, and hence $c = m^e$.
  - $m$ can be computed from $c$ by taking the $e$th root of $c$.
  - Salting helps, because it increases the plaintext so that it becomes larger than $n^{1/e}$.
- It is also good for preventing forward search attacks:
  - If the message space is small and predictable, then an attacker can pre-compute a dictionary by encrypting all possible plaintexts.
  - Salting increases the number of possible plaintexts and makes pre-computing a dictionary harder.
Problems - Homomorphic property

- if \( m_1 \) and \( m_2 \) are two plaintext messages and \( c_1 \) and \( c_2 \) are the corresponding ciphertexts, then the encryption of \( m_1 m_2 \mod n \) is \( c_1 c_2 \mod n \)
  \[
  (m_1 m_2)^e = m_1^e m_2^e = c_1 c_2 \mod n
  \]
- this leads to an adaptive chosen-ciphertext attack on RSA
  - assume that the attacker wants to decrypt \( c = m \mod n \) intended for Alice
  - assume that Alice will decrypt arbitrary ciphertext for the attacker, except \( c \)
  - the attacker can select a random number \( r \) and submit \( c \cdot r^e \mod n \) to Alice for decryption
  - since \( (c \cdot r^e)^d = c^d \cdot r^{ed} = m \cdot r \mod n \), the attacker will obtain \( m \cdot r \mod n \)
  - he then computes \( m \) by multiplication with \( r^{-1} \mod n \)
- this attack can be circumvented by imposing some structural constraints on plaintext messages
  - e.g., a plaintext must start with a well-known constant bit string
  - since \( r \) is random, \( m \cdot r \mod n \) will not have the right structure with very high probability, and Alice can refuse to respond

RSA encryption in practice: PKCS #1

- PKCS v1.5 encoding
  \[
  \begin{array}{c|c|c|c}
  0x00 & 0x02 & \text{at least 8 non-zero random bytes} & 0x00 \text{ message to be encrypted} \\
  \end{array}
  \]
- PKCS v2.0 encoding
  \[
  \begin{array}{c|c|c|c}
  \text{hashed label} & \text{some 0x00 bytes} & 0x01 & \text{message to be encrypted} \\
  \end{array}
  \]

\[0x00 \text{ masked seed} \quad \text{masked message} \]

© Levente Buttyán
Bleichenbacher’s attack on PKCS1 v1.5

- adaptive chosen ciphertext attack
- the goal is to decrypt a message with the help of an oracle that
  - inputs an arbitrary message
  - decrypts it
  - verifies PKCS formatting
  - responds with 1 if the obtained plaintext is PKCS conform, and 0 otherwise
- the attack needs $\sim 2^{20}$ oracle call only

ElGamal cryptosystem

- key generation
  - generate a large random prime $p$ and choose generator $g$ of the
    multiplicative group $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$
  - select a random integer $a$, $1 \leq a \leq p-2$, and compute $A = g^a \mod p$
  - the public key is $(p, g, A)$
  - the private key is $a$
- encryption
  - represent the message as an integer $m$ in $[0, p-1]$
  - select a random integer $r$, $1 \leq r \leq p-2$, and compute $R = g^r \mod p$
  - compute $C = m \cdot A^r \mod p$
  - the ciphertext is the pair $(R, C)$
- decryption
  - compute $m = C \cdot R^{p-1-a} \mod p$
- proof of decryption
  \[ C \cdot R^{p-1-a} = m \cdot A^r \cdot R^{p-1-a} = m \cdot g^a \cdot g^{r(p-1-a)} = m \cdot (g^{p-1})^r = m \mod p \]
**Relation to hard problems**

- security of the ElGamal scheme is said to be based on the discrete logarithm problem in $\mathbb{Z}_p^*$, although equivalence has not been proven yet
- recovering $m$ given $p$, $g$, $A$, $R$, and $C$ is equivalent to solving the Diffie-Hellman problem
- given the latest progress on the discrete logarithm problem, the size of the modulus $p$ should at least be 1024 bits

**Notes on the ElGamal scheme**

- encryption requires two modular exponentiations, whereas decryption requires only one
- encrypted message is twice as long as the plaintext (message expansion)
- all entities in a system may choose to use the same prime $p$ and generator $g$
  - size of the public key is reduced
  - encryption can be speed up by pre-computation