Chapters 8 and 9

More Number Theory and RSA Algorithm

Dr. Shin-Ming Cheng

NTUST

CONNECTIVITY LAB

CS4003701

Implementation aspects

- The RSA cryptosystem uses only one arithmetic operation (modular exponentiation) which makes it conceptually a simple asymmetric scheme
- Even though conceptually simple, due to the use of very long numbers, RSA is orders of magnitude slower than symmetric schemes, e.g., DES, AES
- When implementing RSA (esp. on a constrained device such as smartcards or cell phones) close attention has to be paid to the correct choice of arithmetic algorithms
- The square-and-multiply algorithm allows fast exponentiation, even with very long numbers…

Square-and-Multiply

- Basic principle: Scan exponent bits from left to right and square/multiply operand accordingly

<u>Algorithm: Square-and-Multiply for $x^H \bmod n$</u>
Input: Exponent H, base element x, Modulus n
Output: $y = x^H \bmod n$
1. Determine binary representation $H = (h_t, h_{t-1}, \ldots, h_0)_2$
2. FOR $i = t-1$ TO 0
3. $\quad y = y^2 \bmod n$
4. $\quad$ IF $h_i = 1$ THEN
5. $\quad\quad y = y \times x \bmod n$
6. RETURN y

  - Rule: Square in every iteration (Step 3) and multiply current result by x if the exponent bit $h_i = 1$ (Step 5)
  - Modulo reduction after each step keeps the operand y small

Example: Square-and-Multiply

- Computes x^{26} without modulo reduction
- Binary representation of exponent:

$$26 = (1,1,0,1,0)_2 = (h_4, h_3, h_2, h_1, h_0)_2$$

Step		Binary exponent	Op	Comment
1	$x = x^1$	$(1)_2$		Initial setting, h_4 processed
1a	$(x^1)^2 = x^2$	$(10)_2$	SQ	Processing h_3
1b	$x^2 \times x = x^3$	$(11)_2$	MUL	$h_3 = 1$
2a	$(x^3)^2 = x^6$	$(110)_2$	SQ	Processing h_2
2b	-	$(110)_2$	-	$h_0 = 0$
3a	$(x^6)^2 = x^{12}$	$(1100)_2$	SQ	Processing h_1
3b	$x^{12} \times x = x^{13}$	$(1101)_2$	MUL	$h_1 = 1$
4a	$(x^{13})^2 = x^{26}$	$(11010)_2$	SQ	Processing h_0
4b	-	$(11010)_2$	-	$h_0 = 0$

- Observe how the exponent evolves into $x^{26} = x^{11010}$

Complexity of Square-and-Multiply Alg.

› The square-and-multiply algorithm has a logarithmic complexity, i.e., its run time is proportional to the bit length (rather than the absolute value) of the exponent

› Given an exponent with $t\ +1$ bits $H = (h_t, h_{t-1}, \ldots, h_0)_2$ with $h_t = 1$,we need the following operations
  - # Squarings $= t$
  - Average # multiplications $= 0.5t$
  - Total complexity: #SQ+#MUL $= 1.5t$

› Exponents are often randomly chosen, so 1.5 t is a good estimate for the average number of operations

› Note that each squaring and each multiplication is an operation with very long numbers, e.g., 2048 bit integers

Speed-Up Techniques

- Modular exponentiation is computationally intensive
- Even with the square-and-multiply algorithm, RSA can be quite slow on constrained devices such as smart cards
- Some important tricks:
  - Short public exponent e
  - Chinese Remainder Theorem (CRT)
  - Exponentiation with pre-computation (*not covered here*)

Fast encryption with small public exponent

› Choosing a small public exponent e does not weaken the security of RSA

› A small public exponent improves the speed of the RSA encryption significantly

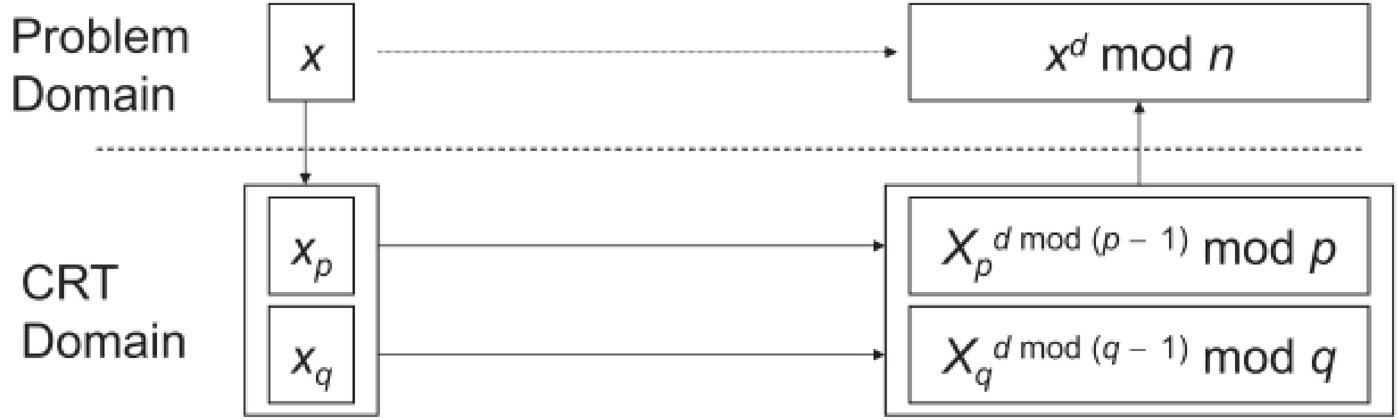
Public Key	e as binary string	#MUL + #SQ
$2^1 + 1 = 3$	$(11)_2$	$1 + 1 = 2$
$2^4 + 1 = 17$	$(1\ 0001)_2$	$4 + 1 = 5$
$2^{16} + 1$	$(1\ 0000\ 0000\ 0000\ 0001)_2$	$16 + 1 = 17$

– This is a commonly used trick (e.g., SSL/TLS, etc.) and makes RSA the fastest asymmetric scheme with regard to encryption!

Fast decryption with CRT

- Choosing a small private key d results in security weaknesses!
  - In fact, d must have at least $0.3t$ bits, where t is the bit length of the modulus n
- CRT can be used to accelerate exponentiation with the private key d
  - Based on the CRT we can replace the computation of $x^{d \bmod \phi(n)} \bmod n$ by two computations $x_p^{d \bmod (p-1)} \bmod p$ and $x_q^{d \bmod (q-1)} \bmod q$
    - where q and p are "small" compared to n

Basic principle of CRT-based exponentiation



- CRT involves three distinct steps
  1. Transformation of operand into the CRT domain
  2. Modular exponentiation in the CRT domain
  3. Inverse transformation into the problem domain
- These steps are equivalent to one modular exponentiation in the problem domain

CRT: Step 1 – Transformation

- Transformation into the CRT domain requires the knowledge of p and q
- p and q are only known to the owner of the private key, hence CRT cannot be applied to speed up encryption
- The transformation computes (x_p, x_q) which is the representation of x in the CRT domain. They can be found easily by computing $x_p \equiv x \mod p$ and $x_q \equiv x \mod q$

CRT: Step 2 – Exponentiation

- Given d_p and d_q such that $d_p \equiv d \bmod (p-1)$ and $d_q \equiv d \bmod (q-1)$
- One exponentiation in the problem domain requires two exponentiations in the CRT $y_p \equiv x_p^{d_p} \bmod p$ and $y_q \equiv x_q^{d_q} \bmod q$
- In practice, p and q are chosen to have half the bit length of n,
  - $|p| \approx |q| \approx |n|/2$

CRT: Step 3 – Inverse Transformation

- Inverse transformation requires modular inversion twice, which is computationally expensive $c_p \equiv q^{-1} \bmod p$ and $c_q \equiv p^{-1} \bmod q$
- Inverse transformation assembles y_p, y_q to the final result $y \bmod n$ in the problem domain $y \equiv [q \times c_p] \times y_p + [p \times c_q] \times y_q \mod n$
  - The primes p and q typically change infrequently, therefore the cost of inversion can be neglected because the two expressions $[q \times c_p]$ and $[p \times$

CRT: Step 3 – Inverse Transformation

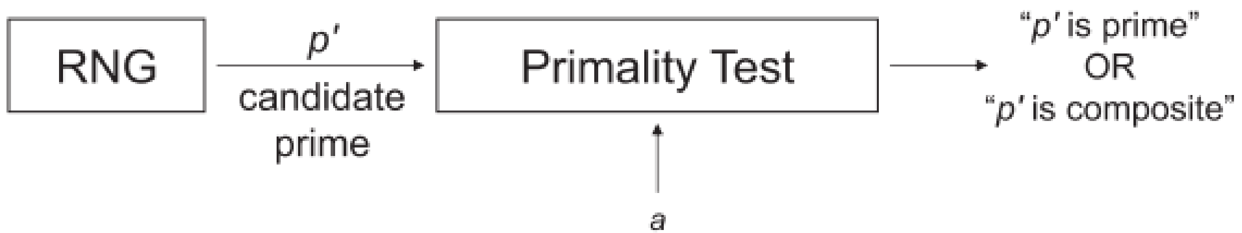
- To decrease the amount of storage and calculation
- To recover x from x_p and x_q, use the CRT
  - Compute $t = p^{-1} \bmod q$ and store it with the private key
  - Computer $u = (x_q - x_p)t \bmod q$, then $x = x_p + pu$

Complexity of CRT

- We ignore the transformation and inverse transformation steps since their costs can be neglected
- n has $t+1$ bits, both p and q are about $t\,/\,2$ bits long
- The complexity is determined by the two exponentiations in the CRT domain.
  - The operands are only $t/2$ bits long.
- For the exponentiations we use the square-and-multiply algorithm:
  - # squarings (one exp.): #SQ = $0.5\,t$
  - # aver. multiplications (one exp.): #MUL = $0.25\,t$
  - Total complexity: $2 \times$ (#MUL+#SQ) $= 1.5\,t$
- Since the operands have half the bit length compared to regular exponent., each operation (i.e., multipl. and squaring) is 4 timers faster !

Finding Large Primes

- Generating keys for RSA requires finding two large primes p and q such that $n = p \times q$ is sufficiently large
- The size of p and q is typically half the size of the desired size of n
- To find primes, random integers are generated and tested for primality:



- The random number generator (RNG) should be non-predictable otherwise an attacker could guess the factorization of n

Prime Numbers

- prime numbers only have divisors of 1 and self
  - they cannot be written as a product of other numbers
  - note: 1 is prime, but is generally not of interest
  - eg. 2,3,5,7 are prime, 4,6,8,9,10 are not
- prime numbers are central to number theory
- list of prime number less than 200 is:

```
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
```

Prime Factorisation

- to **factor** a number n is to write it as a product of other numbers: $n = a \times b \times c$
- note that factoring a number is relatively hard compared to multiplying the factors together to generate the number
- the **prime factorisation** of a number n is when its written as a product of primes
  - $91 = 7 \times 13;\ \ 3600 = 2^4 \times 3^2 \times 5^2$
  - $a = \prod_{p \in \mathrm{P}} p^{a_p}$

Relatively Prime Numbers & GCD

- two numbers a, b are **relatively prime** if have **no common divisors** apart from 1
  - 8 and 15 are relatively prime since factors of 8 are $1,2,4,8$ and of 15 are $1,3,5,15$ and 1 is the only common factor
- conversely can determine the greatest common divisor by comparing their prime factorizations and using least powers
  - $300 = 2^1 \times 3^1 \times 5^2$ $\quad 18 = 2^1 \times 3^2$ hence $\mathrm{GCD}(18,300) = 2^1 \times 3^1 = 6$

Prime Number Theorem

- **Prime Number (Distribution) Theorem:** The number of primes less than N is about $N/\log N$
  - This means primes are quite common
  - The number of primes $< 2^{512}$ is about 2^{503}
  - The first and last bits are set as 1, other 510 bits are random
- If N is a number chosen at random, then the probability of being a prime is about $1/\log N$ [base e, natural logarithm]
  - A random number of 512 bits is a prime with probability 355^{-1}
  - So on average we need to select 177 odd numbers of size 2^{512} before finding a prime number
  - Hence, it is practical to generate large primes, as long as we can test primality efficiently

Primality Tests

- Factoring p and q to test for primality is typically not feasible
- However, we are not interested in the factorization, we only want to know whether p and q are composite
- Typical primality tests are probabilistic, i.e., they are not 100% accurate but their output is correct with very high probability
- A probabilistic test has two outputs:
  - "p' is composite" – always true
  - "p' is a prime" – only true with a certain probability
- Among the well-known primality tests are the following
  - Fermat Primality-Test
  - Miller-Rabin Primality-Test

Fermat's Test

- Basic idea: Fermat's Little Theorem holds for all primes, i.e., if a number p' is found for which $a^{p'-1} \neq 1 \bmod p'$, it is not a prime

Algorithm: Fermat Primality-Test
Input: Prime candidate p', security parameter s
Output: "p' is composite" or "p' is likely a prime"
1. FOR $i = 1$ TO s
1.1 choose random $a \in \{2, 3, \ldots, p' - 2\}$
1.2 IF $a^{p'-1} \neq 1 \bmod p'$ THEN
1.3 RETURN "p' is composite"
2 RETURN "p' is likely a prime"

Fermat's Test

- To test N for primality:

  For $i = 1$ to k do

  Pick a randomly from Z_N^*

  Compute $b \equiv a^{N-1} \pmod{N}$

  If $\neq 1$ output (Composite, a)

  Output "Probably Prime"

- If the above outputs (Composite, a), then
  - N is definitely composite
  - a is a witness for this compositeness

Fermat's Test

- For certain numbers ("Carmichael numbers") this test returns "p' is likely a prime" often – although these numbers are composite
- Example: $561 = 3 \times 11 \times 17$
  - $a^{561} \equiv a \pmod{561}$
  - If $3 \perp a$, $a^{561} \equiv a(a^2)^{280} \equiv a \pmod{3}$
- Therefore, the Miller-Rabin Test is preferred

Miller-Rabin Test

- Miller-Rabin Test
  - Its original version, due to Gary L. Miller, is deterministic, but the determinism relies on the unproven generalized Riemann hypothesis; Michael O. Rabin modified it to obtain an unconditional probabilistic algorithm
  - A modification of the Fermat Test
  - Avoids the problem of composites without witness
  - Has probability $1/4$ of accepting a composite as prime for each random base a
    - Prob (a composite not finding a witness) ≤ ¼
    - Repeating the test k times ⇒Prob (error) ≤ 4^{-k}

Theorem for Miller- Rabin's test

- The more powerful Miller-Rabin Test is based on the following theorem

Theorem
Given the decomposition of an odd prime candidate p'

$$p' - 1 = 2^u \times r$$

where r is odd. If we can find an integer a such that
$a^r \neq 1 \mod p'$ and $a^{r^{2j}} \neq p' - 1 \mod p'$
For all $j = \{0,1,\dots,u-1\}$, then p' is composite.
Otherwise it is probably a prime.

- This theorem can be turned into an algorithm

Miller-Rabin Test

- Concept:
  - If $x^2 \equiv 1 \pmod{p}$ for a prime p, then $x = \pm 1 \pmod{p}$
  - If $x \neq \pm 1 \pmod{N}$ but $x^2 \equiv 1 \pmod{N}$, then N is a composite
- To test N for primality:

  Write $N - 1 = 2^k m$ with m is odd

  Choose $a \in \{2, \dots, N-2\}$

  Compute $b = a^m \pmod{N}$

  If $(b \neq 1$ and $b \neq (N-1)$

  $i = 1$

  While $(i < k$ and $b \neq (N-1))$

  $b = b^2 \pmod{N}$

  If $(b = 1)$ Output (Composite, a)

  $i = i + 1$

  If $(b \neq (N-1))$ Output (Composite, a)

  Output “Probable Prime”

PRIMES is in P

- AKS primality test determines whether a number is prime or composite within polynomial time
  - The first primality-proving algorithm to be simultaneously general, polynomial, deterministic, and unconditional
    - Previous algorithms have achieved any three of these properties, but not all four
  - Major result in Algorithms (AKS, 2002)
    - Manindra Agrawal, Neeraj Kayal, Nitin Saxena, "PRIMES is in P", *Annals of Mathematics* 160 (2004), no. 2, pp. 781–793.
  - Unclear as to its practical importance
  - Based on the fact that $(x-a)^N \equiv x^N - a \pmod{N}$ for $\gcd(a, N) = 1$ is true if and only if N is prime
    - Generalization of Fermat's little theorem

Attacks and Countermeasures

- There are two distinct types of attacks on cryptosystems
  - **Analytical attacks** try to break the mathematical structure of the underlying problem of RSA
    - Calcuate p and q of n
  - **Implementation attacks** try to attack a real-world implementation by exploiting inherent weaknesses in the way RSA is realized in software or hardware

Attacks and Countermeasures
Analytical attacks

- Mathematical attacks
  - The best known attack is factoring of n in order to obtain $\phi(n)$
  - Can be prevented using a sufficiently large modulus n
  - The current factoring record is 768 bits. Thus, it is recommended that n should have a bit length between 1024 and 3072 bits

Factorization of RSA-768

- http://eprint.iacr.org/2010/006
- RSA-768: (232 digits)
  - 1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413
- Factorization: (Both factors have 384 bits and 116 digits)
  - 33478071698956898786044169848212690817704794983713768568912431388982883793878002287614711652531743087737814467999489
  - 36746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917

RSA Challenge

- Active from 1990 and inactive since 2007

#decimals	Data or year	Algorithm	Effort (MIPS years)
39	Sep 13, 1970	CF	
50	1983	CF	
55-71	1983-1984	QS	
45-81	1986	QS	
78-90	1987-1988	QS	
87-92	1988	QS	
93-102	1989	QS	
107-116	1990	QS	275 for C116
RSA-100	Apr 1991	QS	7
RSA-110	Apr 1992	QS	75
RSA-120	Jun 1993	QS	835
RSA-129	Apr 1994	QS	5000
RSA-130	Apr 1996	NFS	1000
RSA-140	Feb 1999	NFS	2000
RSA-155	Aug 1999	NFS	8400

Attacks and Countermeasures
Analytical attacks

- Protocol attacks
  - Exploit the malleability of RSA, i.e., the property that a cipher-text can be transformed into another ciphertext which decrypts to a related plaintext – without knowing the private key
  - $s^e y \bmod N$
  - $(s^e y)^d \equiv s^{ed} \cdot y^d \bmod N \equiv s \cdot x$

- Can be prevented by proper padding
  - Last 20 bits are 1010 or 0000

Attacks and Countermeasures
Implementation attacks

- Side-channel analysis
  - Exploit physical leakage of RSA implementation (e.g., power consumption, EM emanation, etc.)
- Fault-injection attacks
  - Inducing faults in the device while CRT is executed can lead to a complete leakage of the private key