ENEE 459-C
Computer Security

RSA and ElGamal encryption

UNIVERSITY OF
MARYLAND

Fermat's Little Theorem

Theorem

Let p be a prime. For each nonzero x of Z_p, we have $x^{p-1} \bmod p = 1$

- Example ($p = 5$):

  $1^4 \bmod 5 = 1$ $\quad$ $2^4 \bmod 5 = 16 \bmod 5 = 1$

  $3^4 \bmod 5 = 81 \bmod 5 = 1$ $\quad$ $4^4 \bmod 5 = 256 \bmod 5 = 1$

Corollary

Let p be a prime. For each nonzero residue x of Z_p, the multiplicative inverse of x is $x^{p-2} \bmod p$

Proof

$$x(x^{p-2} \bmod p) \bmod p = xx^{p-2} \bmod p = x^{p-1} \bmod p = 1$$

Euler's Theorem

- The multiplicative group for Z_n, denoted with Z^*_n, is the subset of elements of Z_n relatively prime with n
- The totient function of n, denoted with $\phi(n)$, is the size of Z^*_n
- If N = pq (p and q are primes), φ(N) = (p-1)(q-1)
- Example

  $Z^*_{10} = \{ 1, 3, 7, 9 \}$ $\qquad \phi(10) = 4$
- If p is prime, we have

  $Z^*_p = \{1, 2, \ldots, (p-1)\}$ $\qquad \phi(p) = p - 1$

Euler's Theorem

For each element x of Z^*_n, we have $x^{\phi(n)} \bmod n = 1$

- Example ($n = 10$)

  $3^{\phi(10)} \bmod 10 = 3^4 \bmod 10 = 81 \bmod 10 = 1$

  $7^{\phi(10)} \bmod 10 = 7^4 \bmod 10 = 2401 \bmod 10 = 1$

  $9^{\phi(10)} \bmod 10 = 9^4 \bmod 10 = 6561 \bmod 10 = 1$

Computing in the exponent

- For a generator of a group g, it is: g^{order of group} = identity
- For the multiplicative group Z^*_n, we can compute in the exponent modulo $\phi(n)$
- Corollary: For Z^*_p, we can compute in the exponent modulo **p-1**
- Example

$$Z^*_{10} = \{ 1, 3, 7, 9 \} \qquad \phi(10) = 4$$

$$3 \wedge 1590 \bmod 10 = 3 \wedge (1590 \bmod 4) \bmod 10 = 3\wedge 2 \bmod 10 = 9$$

How about 2^8 mod 10?

- Example for p=19

$$Z^*_p = \{1, 2, \ldots, (p-1)\} \qquad \phi(p) = p-1$$

$$15\wedge 39 \bmod 19 = 15\wedge (39 \bmod 18) \bmod 19 = 15\wedge 3 \bmod 19 = 12$$

RSA Cryptosystem

- Setup:
  - $\boldsymbol{n} = \boldsymbol{pq}$, with $\boldsymbol{p}$ and $\boldsymbol{q}$ primes
  - $\boldsymbol{e}$ relatively prime to $\phi(n) = (\boldsymbol{p} - 1)\,(\boldsymbol{q} - 1)$
  - $\boldsymbol{d}$ inverse of $\boldsymbol{e}$ in $Z_{\phi(\boldsymbol{n})}$
- Keys:
  - Public key: $K_E = (\boldsymbol{n}, \boldsymbol{e})$
  - Private key: $K_D = \boldsymbol{d}$
- Encryption:
  - Plaintext $\boldsymbol{M}$ in Z_n
  - $\boldsymbol{C} = \boldsymbol{M}^e \bmod \boldsymbol{n}$
- Decryption:
  - $\boldsymbol{M} = \boldsymbol{C}^d \bmod \boldsymbol{n}$

- Example
  - Setup:
    - $\boldsymbol{p} = 7,\ \ \boldsymbol{q} = 17$
    - $\boldsymbol{n} = 7 \cdot 17 = 119$
    - $\phi(\boldsymbol{n}) = 6 \cdot 16 = 96$
    - $\boldsymbol{e} = 5$
    - $\boldsymbol{d} = 77$
  - Keys:
    - public key: $(119, 5)$
    - private key: 77
  - Encryption:
    - $\boldsymbol{M} = 19$
    - $\boldsymbol{C} = 19^5 \bmod 119 = 66$
  - Decryption:
    - $\boldsymbol{C} = 66^{77} \bmod 119 = 19$

Complete RSA Example

- Setup:
  - $p = 5$, $q = 11$
  - $n = 5 \cdot 11 = 55$
  - $\phi(n) = 4 \cdot 10 = 40$
  - $e = 3$
  - $d = 27$ $(3 \cdot 27 = 81 = 2 \cdot 40 + 1)$

- Encryption
  - $C = M^3 \bmod 55$
- Decryption
  - $M = C^{27} \bmod 55$

M	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
C	1	8	27	9	15	51	13	17	14	10	11	23	52	49	20	26	18	2
M	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
C	39	25	21	33	12	19	5	31	48	7	24	50	36	43	22	34	30	16
M	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
C	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

Questions

- Problem with RSA?
- Does it satisfy semantic security?

Security of RSA

- Security of RSA based on difficulty of factoring
  - Widely believed
  - Best known algorithm takes exponential time
- RSA Security factoring challenge (discontinued)
- In 1999, 512-bit challenge factored in 4 months using 35.7 CPU-years
  - 160 175-400 MHz SGI and Sun
  - 8 250 MHz SGI Origin
  - 120 300-450 MHz Pentium II
  - 4 500 MHz Digital/Compaq
- In 2005, a team of researchers factored the RSA-640 challenge number using 30 2.2GHz CPU years
- In 2004, the prize for factoring RSA-2048 was $200,000
- Current practice is 2,048-bit keys
- Estimated resources needed to factor a number within one year

Length (bits)	PCs	Memory
430	1	128MB
760	215,000	4GB
1,020	342×10^{6}	170GB
1,620	1.6×10^{15}	120TB

Correctness

- We show the correctness of the RSA cryptosystem for the case when the plaintext M does not divide n
- Namely, we show that

$$(M^e)^d \bmod n = M$$

- Since $ed \bmod \phi(n) = 1$, there is an integer k such that

$$ed = k\phi(n) + 1$$

- Since M does not divide n, by Euler's theorem we have

$$M^{\phi(n)} \bmod n = 1$$

- Thus, we obtain

$$\begin{aligned}
(M^e)^d \bmod n &= \\
&M^{ed} \bmod n = \\
&M^{k\phi(n) + 1} \bmod n = \\
&MM^{k\phi(n)} \bmod n = \\
&M\,(M^{\phi(n)})^k \bmod n = \\
&M\,(M^{\phi(n)} \bmod n)^k \bmod n = \\
&M\,(1)^k \bmod n = \\
&M \bmod n = \\
&M
\end{aligned}$$

- Proof of correctness can be extended to the case when the plaintext M divides n

Algorithmic Issues

- The implementation of the RSA cryptosystem requires various algorithms
- Overall
  - Representation of integers of arbitrarily large size and arithmetic operations on them
- Encryption
  - Modular power
- Decryption
  - Modular power
- Setup
  - Generation of random numbers with a given number of bits (to generate candidates p and q)
  - Primality testing (to check that candidates p and q are prime)
  - Computation of the GCD (to verify that e and $\phi(n)$ are relatively prime)
  - Computation of the multiplicative inverse (to compute d from e)

Modular Power

- The repeated squaring algorithm speeds up the computation of a modular power $a^p \bmod n$
- Write the exponent p in binary

  $p = p_{b-1} p_{b-2} \cdots p_1 p_0$
- Start with

  $Q_1 = a^{p_{b-1}} \bmod n$
- Repeatedly compute

  $Q_i = ((Q_{i-1})^2 \bmod n) a^{p_{b-i}} \bmod n$
- We obtain

  $Q_b = a^p \bmod n$
- The repeated squaring algorithm performs $O(\log p)$ arithmetic operations

- Example
  - $3^{18} \bmod 19$ $(18 = 10010)$
  - $Q_1 = 3^1 \bmod 19 = 3$
  - $Q_2 = (3^2 \bmod 19)3^0 \bmod 19 = 9$
  - $Q_3 = (9^2 \bmod 19)3^0 \bmod 19 =$ $81 \bmod 19 = 5$
  - $Q_4 = (5^2 \bmod 19)3^1 \bmod 19 =$ $(25 \bmod 19)3 \bmod 19 =$ $18 \bmod 19 = 18$
  - $Q_5 = (18^2 \bmod 19)3^0 \bmod 19 =$ $(324 \bmod 19) \bmod 19 =$ $17 \cdot 19 + 1 \bmod 19 = 1$

p_{5-i}	1	0	0	1	0
$2^{p_{5-i}}$	3	1	1	3	1
Q_i	3	9	5	18	1

Chinese remainder theorem light

- Let N=pq. Let
  - x mod p = a1
  - x mod q = a2
- Then
  - x **mod N** = a1* q * inverse(q in Zp)+ a2 * p * inverse(p in Zq) **mod N**
  - **Let's prove it**
  - This can be used to compute W^x mod N, for big W^x, more efficiently
  - How?
- Use of theorem
  - Say you want to compute 18^25 mod 35 (35 = 5*7)
  - Compute 18^25 mod 5 = 18^(25 mod 4) mod 5= 18^1 mod 5 = 3 = a1
  - Compute 18^25 mod 7 = 18^(25 mod 6) mod 7= 18^1 mod 7 = 4 = a2
  - Note that inverse(5 in Z_7)=3 and inverse(7 in Z_5)=3
  - Therefore the solution we are looking for is 3*7*3+4*5*3 mod 35= 18
- Used in the decryption procedure of RSA: Why cannot it be used in the encryption?
- Also we can prove correctness of RSA for general message M

Pseudoprimality Testing

- Testing whether a number is prime (primality testing) is a difficult problem, though polynomial-time algorithms exist
- An integer $n \geq 2$ is said to be a base-x pseudoprime if
  - $x^{n-1} \bmod n = 1$ (Fermat's little theorem)
- Composite base-x pseudoprimes are rare:
  - A random 100-bit integer is a composite base-2 pseudoprime with probability less than 10^{-13}
  - The smallest composite base-2 pseudoprime is 341
- Base-x pseudoprimality testing for an integer n:
  - Check whether $x^{n-1} \bmod n = 1$
  - Can be performed efficiently with the repeated squaring algorithm

RSA security and properties

- Plain RSA is deterministic.
- Why is this a problem?
- Plain RSA is also homomorphic. What does this mean?
  - Multiply ciphertexts to get ciphertext of multiplication!
  - $[(m_1)^e \bmod N][(m_2)^e \bmod N] = (m_1 m_2)^e \bmod N$
- However, not additively homomorphic
- Both additively + multiplicative homomorphic (aka fully-homomorphic) encryption open problem till 2009
- A breakthrough result from IBM (Craig Gentry) answered this open problem, constructing such an encryption scheme

Real World Usage of RSA

- Randomized RSA
  - To encrypt a message M under an RSA public key (n,e), generate a new random session AES key K, compute the cipher text as
    - [K^e mod N, $AES_K(M)$]
- This prevents an adversary distinguishing two encryptions of the same message since K is chosen at random every time the encryption takes place
- Could the following encryption work for arbitrary messages M?
  - $(M||r)^e$ mod N, for random r

ElGamal Encryption

- Encrypts messages $m \in Z_p$
- **Secret key:** random number $a \in Z_p$
- **Public key:** $A = g^a$
- **Encryption**: Pick a random $k \in Z_p$ and set
  - $K = A^k = g^{ak}$
  - $c_1 = g^k$
  - Then **Enc(m) = (c_1, c_2)** where **$c_2 = mK \bmod p$**
  - **Dec(c_1,c_2) = $c_2*(1/K) \bmod p$** where **$K = c_1^a = g^{ak}$**
- Security depends on Computational Diffie-Hellman (CDH) assumption: given (g, g^a, g^b) it is hard to compute g^{ab}
- Do not use same k twice