RSA (material drawn from Avi Kak (kak@purdue.edu)
Lecture 12, Lecture Notes on "Computer and Network Security"

Used in asymmetric crypto. protocols

The RSA algorithm is based on the following property of positive integers.

When n satisfies a certain property to be described later, in arithmetic operations modulo n, the exponents behave modulo the totient ($\phi(n)$) of n.

(*totient(n)* is defined to be the **number** of positive integers less than or equal to n that are coprime to n (i.e. having no common positive factors other than 1))

For example, consider arithmetic modulo 15

We have $\phi(15) = 8$ for the totient

(since 1, 2, 4, 7, 8, 11, 13, 14 are coprime to 15, i.e., no common divisors)

You can easily verify the following:

$5^7 * 5^4 \bmod 15 = 5^{(7+4) \bmod 8} \bmod 15 = 5^3 \bmod 15 = 125 \bmod 15 = 5$

$(4^3)^5 \bmod 15 = 4^{(3*5) \bmod 8} \bmod 15 = 4^7 \bmod 15 = 4$

RSA

Again considering arithmetic modulo n, let's say that e is an integer that is coprime to the totient $\phi(n)$ of n.

Further, say that d is the **multiplicative inverse** of e modulo $\phi(n)$.

These definitions are summarized as follows:

n = a modulus for modular arithmetic

$\phi(n)$ = the totient of n

e = an integer that is relatively prime to $\phi(n)$ (This guarantees that e will possess a multiplicative inverse modulo $\phi(n)$)

d = an integer that is the multiplicative inverse of e modulo $\phi(n)$

Now suppose we are given an integer M , M < n, that represents our message, then we can transform M into another integer C that will represent our ciphertext by the following modulo exponentiation:

$$C = M^e \bmod n$$

We can recover M back from C by the following modulo operation

$$M = C^d \bmod n$$

RSA

How does the algorithm work?

An individual who wishes to receive messages confidentially will use the pair of integers $\{e, n\}$ as his/her public key

At the same time, this individual can use the pair of integers $\{d, n\}$ as the private key

Another party wishing to send a message to such an individual will encrypt the message using the public key $\{e, n\}$

Only the individual with access to the private key $\{d, n\}$ will be able to decrypt the message

RSA could be used as a block cipher for the encryption of the message

The block size would equal the number of bits required to represent the modulus n

RSA

If the modulus required requires 1024 bits for its representation, message encryption would be based on 1024-bit blocks

The important theoretical question here is under what conditions must be satisfied by the modulus *n* for this M ->C -> M transformation to work?

How do we choose the modulus for the RSA algorithm?

With the definitions given above for *d* and *e*, the modulus *n* must be selected in such a manner to satisfy the following:

$(M^e)^d == M^{ed} == M \pmod{n}$

We want this guarantee this because $C = M^e \bmod n$ is the encrypted form of the message integer M, and decryption is carried out by $C^d \bmod n$

It was shown by Rivest, Shamir, and Adleman (RSA) that we have this guarantee when *n* is a product of two prime numbers:

n = p*q for some prime *p* and prime *q*

RSA

If two integers *p* and *q* are coprimes (meaning, relatively prime to each other), the following equivalence holds for any two integers *a* and *b*:

$\{a == b \pmod{p} \quad \text{and} \quad a == b \pmod{q}\} \quad \text{iff} \quad \{a == b \pmod{p*q}\}$

In addition to needing *p* and *q* to be coprimes, we also want *p* and *q* to be **individually** primes.

It is only when *p* and *q* are individually prime that we can decompose the totient of *n* into the product of the totients of *p* and *q,*

$\phi(n) = \phi(p) * \phi(q) = (p - 1) * (q - 1)$

So that the cipher cannot be broken by an exhaustive search for the prime factors of the modulus *n*, it is important that both *p* and *q* be **very large primes**

Finding the prime factors of a large integer is *computationally harder* than determining its primality

Computational Steps for Key Generation in RSA

The RSA scheme is a block cipher

One typically encodes blocks of length 1024 bits

This means that the numerical value of the message integer M will be less than 2^{1024}

If this integer is expressed in decimal form, its value could be as large as 10^{309}

In other words, the message integer M could have as many as 309 decimal digits for each block of the plaintext!

The computational steps for key generation are

- Generate two different primes p and q
- Calculate the modulus $n = p * q$
- Calculate the totient $\phi(n) = (p - 1) * (q - 1)$
- Select for public exponent an integer e such that $1 < e < \phi(n)$ and $\gcd(\phi(n), e) = 1$
- Calculate for the private exponent a value for d such that $d = e^{-1} \bmod \phi(n)$

Computational Steps for Key Generation in RSA

- Public Key = [e, n]
- Private Key = [d, n]

For example, assume we want to design a 16-bit block encryption of disk files

That is our modulus *n* will span 16 bits

Since M (number of bits to encrypt) must be smaller than *n*, we need to choose a smaller block size, e.g., 8 bits

We will *pad* with 0s the remaining 8 bits -- which turns out to be important to make RSA resistant to certain vulnerabilities (see standards doc RFC 3447)

So for each 8-bit block read from disk, we pad to 16-bits with 0s to make M

So, we need to find a modulus *n* with size 16 bits

Remember, *n* must be a product of two primes *p* and *q*

Assuming we want *p* and *q* to be roughly the same size, let's allocate 8 bits each for them

Computational Steps for Key Generation in RSA

So the issue now is how to find a prime suitable for our 8-bit example?

(A random number generator can be used to do this)

A simple approach is as follows: set the first two bits and last bit to 1 for both p and q

```
1 1 - -  - - - 1 (p)
1 1 - -  - - - 1 (q)
```

Given these constraints, the minimum value is 193 for both p and q

Setting the two high order bits also ensures the product will span 2^{15} range

So the question reduces to whether there exist two primes (hopefully different) whose decimal values exceed 193 but are less than 255

If you carry out a Google search with a string like 'first 1000 primes', you will discover that there exist many candidates for such primes

http://primes.utm.edu/lists/small/1000.txt

Computational Steps for Key Generation in RSA

Let's select the following two

$p = 197$ and $q = 211$

This gives us for the modulus $n = 197 * 211 = 41567$

The bit pattern for the chosen p, q, and modulus n are:

```
1 1 0 0  0 1 1 1 (p) (0xC5)
1 1 0 1  0 0 1 1 (q) (0xD3)
1 0 1 0  0 0 1 0  0 1 0 1  1 1 1 1 (n)(0xA25F)
```

So you can see we have found a modulus for a 16-bit RSA cipher that requires 16 bits for its representation

Now let's try to select appropriate values for e and d

For e we want an integer that is relatively prime to the totient $\phi(n) = 196 * 210 = 41160$.

Computational Steps for Key Generation in RSA

Such an e will also be relatively prime to 196 and 210, the totients of p and q respectively

Since it is preferable to select a small prime for e, we could try $e = 3$

But that does not work since 3 is not relatively prime to 210

The value $e = 5$ does not work for the same reason

Let's try $e = 17$ because it is a small prime and because it has only two bits set

With e set to 17, we must now choose d as the multiplicative inverse of e modulo 41160

We can use the Bezout's identity based calculations; we write

```
gcd(17, 41160)       |
  = gcd(41160, 17)   | residue 17 = 0 x 41160 + 1 x 17
  = gcd(17, 3)       | residue 3 =  1 x 41160 - 2421 x 17
```

Computational Steps for Key Generation in RSA

```
    = gcd(3,2) | res 2= -5 x 3      + 1 x 17
               |      = -5x(1 x 41160 - 2421 x 17) + 1 x 17
               |      = 12106 x 17 - 5 x 41160
    = gcd(2,1) | res 1= 1x3 - 1 x 2
               |      = 1x(41160 - 2421x17)
               |                - 1x(12106x17 -5x41160)
               |      = 6 x 41160   - 14527 x 17
               |      = 6 x 41160   + 26633 x 17
```

(the last equality for the residue 1 uses the fact that the additive inverse of 14527 modulo 41160 is 26633)

Use a program to do this!

The Bezout's identity shown above tells us that the multiplicative inverse of 17 modulo 41160 is 26633

You can verify this fact by showing 17 * 26633 mod 41160 = 1 on your calculator

Computational Steps for Key Generation in RSA

Our 16-bit block cipher based on RSA therefore has the following numbers for *n*, *e*, and *d*:

```
n     =     41567
e     =     17
d     =     26633
```

Of course, as you would expect, this block cipher would have no security since it would take no time at all for an adversary to factorize *n* into its components *p* and *q*

As mentioned already, the message integer M is raised to the power *e* modulo *n*, which gives us the ciphertext integer C

Decryption consists of raising C to the power *d* modulo *n*

The exponentiation operation for encryption can be carried out efficiently by simply choosing an appropriate *e*

Note that the only condition on *e* is that it be coprime to $\phi(n)$)

Computational Steps for Key Generation in RSA

As mentioned previously, typical choices for *e* are 3, 17, and 65537

All these are prime and each has only two bits set

Modular exponentiation for decryption, meaning the calculation of $C^d \bmod n$, is an entirely different matter since we are not free to choose *d*

The value of *d* is determined completely by *e* and *n*

Computation of $C^d \bmod n$ can be speeded up by using the Chinese Remainder Theorem

Since the party doing the decryption knows the prime factors *p* and *q* of the modulus *n*, we can first carry out the easier exponentiations:

$V_p = C^d \bmod p$

$V_q = C^d \bmod q$

Further speedup can be obtained using Fermatt's Little Theorem

An Algorithm for RSA

After we have simplified the problem of modular exponentiation considerably by using CRT and Fermat's Little Theorem, we are still left with having to calculate:

$A^B \bmod n$

for some integers *A*, *B*, and for some modulus *n*

What is interesting is that even for small values for *A* and *B*, the value of A^B can be enormous!

For example, both *A* and *B* may consist of only a couple of digits, as in 711, but the result could still be a very large number

For example, 711 equals 1, 977, 326, 743, a number with 10 decimal digits

Now just imagine what would happen if, as would be the case in cryptography, *A* had, say, 256 binary digits (that is 77 decimal digits) and *B* was, say, 65537!

Even when *B* has only 2 digits (say, B = 17), when A has 77 decimal digits, A^B will have 1304 decimal digits!

An Algorithm for RSA

The calculation of A^B can be speeded up by realizing that if *B* can be expressed as a sum of smaller parts, then the result is a product of smaller exponentiations

We can use the following binary representation for the exponent B:

`B == b_k b_{k-1}b_{k-2} . . . b_0 (binary)`

Here, we find that it takes *k* bits to represent the exponent, each bit being represented by b_i, with b_k as the highest bit and b_0 as the lowest bit

In terms of these bits, we can write the following equality for B:

$$B = \sum_{b_i \neq 0} 2^i$$

Now the exponentiation A^B may be expressed as:

$$A^B = A^{\sum_{b_i \neq 0} 2^i} = \prod_{b_i \neq 0} A^{2^i}$$

An Algorithm for RSA

We could say that this form of A^B halves the difficulty of computing A^B

This is true b/c assuming all the bits of B are set, the largest value of 2^i will be roughly half the largest value of B

We can achieve further simplification by bringing the rules of modular arithmetic into the multiplications on the right:

$$A^B \bmod n = \left(\prod_{b_i \neq 0} \left[A^{2^i} \bmod n \right] \right) \bmod n$$

Note that as we go from one bit position to the next higher bit position, we **square** the previously computed power of A

The A^2 terms in the above product are of the following form

$$A^{2^0}, A^{2^1}, A^{2^2}, A^{2^3}, \ldots$$

So instead of calculating each term from scratch, we can calculate each by squaring the previous value

An Algorithm for RSA

We may express this idea in the following manner:

$$A, \quad A^2_{prev}, A^2_{prev}, A^2_{prev}, \ldots$$

Now we can write an algorithm for exponentiation that scans the binary representation of the exponent B from the lowest bit to the highest bit:

```
result = 1
while ( B > 0 )
  if ( B & 1 )              # check the lowest bit of B
      result = ( result * A ) mod n
  B = B >> 1                # shift B by one bit to right
  A = ( A * A ) mod n
return result
```

Try:

```
ssh-keygen -t dsa
```

or

```
ssh-keygen -t rsa
```

Complexity of RSA

Symmetric Key Algorithm	Key Size for the Symmetric Key Algorithm	Comparable RSA Key Len. giving same level of Sec.
2-Key 3DES	80	1024
3-Key 3DES	112	2048
AES-128	128	3072
AES-192	192	7680
AES-256	256	15360

For RSA

Doubling the size of the key will, in general, increase the time required for public key operations (encryption or signature verification) by a **factor of four**

And it will increase the time taken by private key operations (decryption and signing) by a **factor of 8**

Public key operations don't increase in cost as fast because *e* does not have to change size with an increase in the modulus -- while *d* does

The key generation time goes up by a **factor of 16** as the size of the key (fortunately, not a frequent operation)

Complexity of RSA

This high cost makes RSA inappropriate for encryption/decryption of actual message content for high data-rate communication links

However, RSA is ideal for the exchange of secret keys that can subsequently be used for the more traditional (and much faster) symmetric-key encryption/decryption