University of Colorado **Boulder**

Department of Computer Science

CSCI 2824: Discrete Structures

Chris Ketelsen

Cryptography

Public Key Encryption and RSA

Public Key Encryption and RSA

Public Key Encryption allows senders and receivers to determine secret keys by transferring public information completely in the open

RSA is the most common Public Key Encryption system, developed by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977

It was actually discovered in 1973 by British Intelligence and immediately classified

Public Key Encryption and RSA

To Develop RSA we'll need these things that we've seen before:

- Fast Modular Exponentiation
- The Euclidean Algorithm
- Bezout's Theorem
- Modular Inverses
- Fermat's Little Theorem
- The Chinese Remainder Theorem

Public Key Encryption and RSA

Basic Idea:

Alice wants to send a message M to Bob that she wishes no one else to read.

In a public key system Bob will send Alice his public key and Alice will use it to encrypt message M as a cipher C

Upon receiving the message from Alice, Bob decrypts it using his private key.

Anyone in the world could intercept the public key and use it to **encrypt** messages

But **no one** else can decrypt the message without Bob's private key

Public Key Encryption and RSA

Implementation in Practice:

The basic idea is to create a *one-way* function F that given a message M and `publicKey`, encrypts the message as $C = F(M, \texttt{publicKey})$ to yield the cipher C

The function F is called *one-way* because

- Given M and `publicKey`, it is easy to compute $C = F(M, \texttt{publicKey})$ to encrypt M
- However if someone intercepts C and `publicKey` it is **extremely** computationally hard to invert F and compute $M = F^{-1}(C, \texttt{publicKey})$

Public Key Encryption and RSA

If it is hard for someone to invert F, how is it that Bob can do so?

We will assume that using the private key, Bob has a function G that will allow him to compute $M = G(C, \texttt{privateKey})$ quickly

So our scheme should give us functions F and G where

- F is easy to compute but very hard to invert.
- Given `privateKey`, G is easy to compute and inverts F

Public Key Encryption and RSA

The basic scheme for RSA uses a really large number n which is a product of two primes p and q (think 200 digits each)

The public key is a pair of numbers (e, n)

The private key is a pair of numbers (d, n)

Each message M is assumed to be a number between 0 and $n - 1$

The basic idea is to encrypt a message M by computing

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

We know that we can do this very fast with modular exponentiation

Public Key Encryption and RSA

Example: Let's take the message $M = 1098$ and encrypt it using the key $e = 13$ and $n = 17947 = 131 \cdot 137$

We simply compute

$$C = 1098^{13} \textbf{ mod } 17947 = 13535$$

If a snooper sees the encoded message C and the public key (e, n) it will be extremely difficult for them to find M

Public Key Encryption and RSA

So far we have identified the one-way function F:

$$F(M, (e, n)) = M^e \bmod n$$

Now the idea is to find a private key (d, n) that decrypts the message via

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

This is our inversion function G:

$$G(C, (d, n)) = C^d \bmod n$$

Public Key Encryption and RSA

To do this we need to find e, d that satisfy

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

Or said another way, we need e, d such that

$$M^{ed} \bmod n = M$$

The question then becomes, how do we find a pair (e, d) that does this where it's hard to discover d if you know (e, n)?

Next we will state some assumptions about the various keys, and then show why this leads to a practically unbreakable cryptosystem

Public Key Encryption and RSA

Assumptions:

- $n = pq$ where p and q are very large primes
- e is a number that is relatively prime to $(p-1)(q-1)$
- d is the **inverse** of e modulo $(p-1)(q-1)$

We know that such an inverse exists because

$$\gcd(e, (p-1)(q-1)) = 1$$

Public Key Encryption and RSA

Assumptions:

- $n = pq$ where p and q are very large primes
- e is a number that is relatively prime to $(p-1)(q-1)$
- d is the **inverse** of e modulo $(p-1)(q-1)$

To see why these assumptions are helpful, notice that if

$$de \equiv 1 \ (\mathbf{mod}\ (p-1)(q-1))$$

then there exists an integer k such that

$$de = 1 + k(p-1)(q-1)$$

Public Key Encryption and RSA

It then follows that for some integer k

$$C^d = (M^e)^d = M^{de} = M^{1+k(p-1)(q-1)}$$

We now need to show that this implies that

$$C^d \equiv M \pmod{n}$$

Assuming that $\gcd(M, p) = 1$ and $\gcd(M, q) = 1$, **Fermat's Little Theorem** tells us that

$$M^{p-1} \equiv 1 \pmod{p} \quad \text{and} \quad M^{q-1} \equiv 1 \pmod{q}$$

Now we'll write C^d in terms of these things and then tie it all together with the **Chinese Remainder Theorem**

Public Key Encryption and RSA

With respect to p we have

$$\begin{aligned} C^d &\equiv M^{1+k(p-1)(q-1)} \pmod{p} \\ &\equiv M \cdot (M^{p-1})^{k(q-1)} \pmod{p} \\ &\equiv M \cdot 1 \pmod{p} \\ &\equiv M \pmod{p} \end{aligned}$$

Similarly, with respect to q we have

$$\begin{aligned} C^d &\equiv M^{1+k(p-1)(q-1)} \pmod{q} \\ &\equiv M \cdot (M^{q-1})^{k(p-1)} \pmod{q} \\ &\equiv M \cdot 1 \pmod{q} \\ &\equiv M \pmod{q} \end{aligned}$$

Public Key Encryption and RSA

So now we have

$$C^d \equiv M \pmod{p}$$
$$C^d \equiv M \pmod{q}$$

Notice that this looks like a system of congruences

$$x \equiv M \pmod{p}$$
$$x \equiv M \pmod{q}$$

where $x = C^d$ is a solution. Since p and q are relatively prime the **Chinese Remainder Theorem** tells us that x is a unique solution modulo $pq = n$. Thus

$$C^d \equiv M \pmod{n}$$

Public Key Encryption and RSA

What Bob Needs to Do:

1. Choose two very large primes p and q and form $n = pq$

   - **Note**: You can find large primes p and q very quickly with probabilistic guess and checking

2. Find an e relatively prime to $(p-1)(q-1)$

3. Find d, the inverse of e modulo $(p-1)(q-1)$

   - **Note**: The **Euclidean Algorithm** can do this very quickly as $(p-1)$ and $(q-1)$ are not prime (not even odd!)

4. Save `privateKey` (d, n) and send Alice `publicKey` (e, n)

Public Key Encryption and RSA

Example: Encrypt and Decrypt the message $M = 1819$ using a Public Key Encryption based on $p = 43$, $q = 59$ and $e = 13$

We have $n = 43 \cdot 59 = 2537$ and $(p-1)(q-1) = 2436$

Note that

$$2436 = 187 \cdot 13 + 5$$
$$13 = 2 \cdot 5 + 3$$
$$5 = 1 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$

Thus $\gcd(2346, 13) = 1$ and so $e = 13$ works as a public key

Public Key Encryption and RSA

Example: Encrypt and Decrypt the message $M = 1819$ using a Public Key Encryption based on $p = 43$, $q = 59$ and $e = 13$

We have $n = 43 \cdot 59 = 2537$ and $(p-1)(q-1) = 2436$

Now we need to find d, the inverse of $e = 13$ modulo 2436

$$2436 = 187 \cdot 13 + 5$$
$$13 = 2 \cdot 5 + 3$$
$$5 = 1 \cdot 3 + 2$$
$$3 = 1 \cdot 2 + 1$$

Public Key Encryption and RSA

Example: Encrypt and Decrypt the message $M = 1819$ using a Public Key Encryption based on $p = 43$, $q = 59$ and $e = 13$

We have $n = 43 \cdot 59 = 2537$ and $(p-1)(q-1) = 2436$

Now we need to find d, the inverse of $e = 13$ modulo 2436

$$1 = 937 \cdot 13 - 5 \cdot 2436$$

So, by Bezout's Thm, the inverse of $e = 13 \mod 2436$ is $d = 937$

Public Key Encryption and RSA

Example: Encrypt and Decrypt the message $M = 1819$ using a Public Key Encryption based on $p = 43$, $q = 59$ and $e = 13$

We now have our public and private keys

$$\texttt{publicKey} = (e, n) = (13, 2537)$$

$$\texttt{privateKey} = (d, n) = (937, 2537)$$

Public Key Encryption and RSA

Example: Encrypt and Decrypt the message $M = 1819$ using a Public Key Encryption based on $p = 43$, $q = 59$ and $e = 13$

We now have our public and private keys

$$\texttt{publicKey} = (e, n) = (13, 2537)$$

$$\texttt{privateKey} = (d, n) = (937, 2537)$$

Encryption: $C = M^e \textbf{ mod } n = 1819^{13} \textbf{ mod } 2537 = 2081$

Decryption: $M = C^d \textbf{ mod } n = 2081^{937} \textbf{ mod } 2537 = 1819$

Public Key Encryption and RSA

Breaking RSA

OK, so why can we not do a brute-force attack on (e, n) and determine d?

Remember, d is the inverse of e modulo $(p - 1)(q - 1)$

Without knowing that $n = pq$ we have no idea that we should be finding an inverse modulo $(p - 1)(q - 1)$

Thus to find d we first need to factor n

Factoring large products of primes is a **HARD** problem. There is no known polynomial-time solution.

Public Key Encryption and RSA

Breaking RSA

The best-known factoring method is estimated to have complexity

$$2^{\text{number of bits in } n}$$

In 2009 recearchers successfully factored a 232 decimial digit product of two primes.

It took **two years** running in parallel on **hundreds of machines**. It took an equivalent of **2000 years** running on a single-core machine

Factoring is hard!

Public Key Encryption and RSA

More Details / Block Ciphers

Note that when we translate M to a number, we have to be careful that the numerical representation of M is less than n

Public Key Encryption and RSA

More Details / Block Ciphers

Note that when we translate M to a number, we have to be careful that the numerical representation of M is less than n

If M exceeds or equals n then we would be unable to distinguish between M and $M \bmod n$

Typically we break the message into blocks, encode the blocks as digits, and then encrypt the blocks

Example: *HELP* broken up into two-letter blocks becomes

$$HE = 0704 \quad \text{and} \quad LP = 1115$$

Public Key Encryption and RSA

Blocks of two letters works for $n > 2525$

If we have $n > 252525$ we can block by 3 letters, etc

Example: Encrypt *HELP* using the keys from the previous example

We had $(e, n) = (13, 2537)$, so

$$HE = 0704^{13} \textbf{ mod } 2537 = 1980$$

$$LP = 1115^{13} \textbf{ mod } 2537 = 461$$

So the encoded message numerically would be $1980 \;\; 461$

Note that there is no need to pad the encrypted message by 0's unless you want to turn them back into characters

Public Key Encryption and RSA

Example: Decrypt $C = 1188 \;\; 1346$ using the keys from the previous example

We had $(d, n) = (937, 2537)$, so

$$1188^{937} \textbf{ mod } 2537 = 1814 \rightarrow SO$$

$$1346^{937} \textbf{ mod } 2537 = 1823 \rightarrow SX$$

So the decoded message in characters would be *SOSX*.

There is a convention that if your message doesn't fit perfectly into your block size, that you pad the end of the last block with *X*'s.

So this message is likely *SOS*