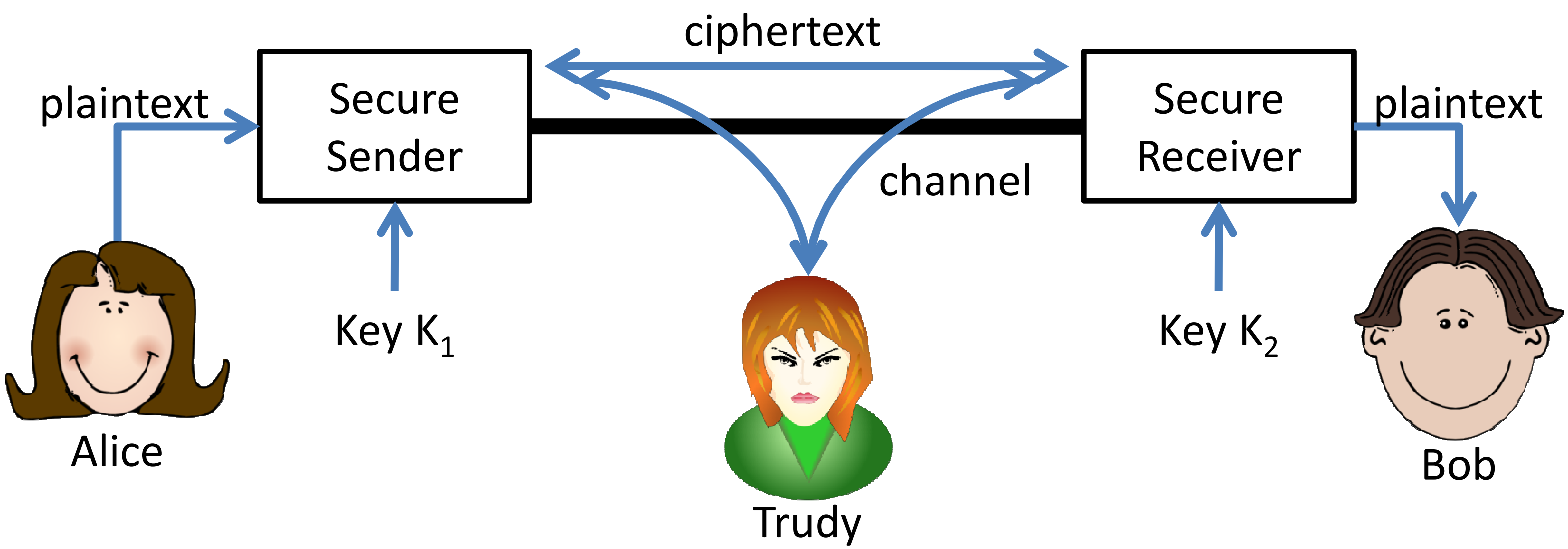
Basics of Cryptology
Public key cryptography

Symmetric and Asymmetric cryptography

- Symmetric cryptography
  - Both Sender/Receiver use the same algorithms/keys for encryption/decryption
  - $K_1 = K_2$ (=K)
  - Requires sender and receiver to share a key (How? What if never met?)
- Asymmetric cryptography (Public key cryptography)
  - Sender/receiver can employ different keys
  - $K_1 \neq K_2$
  - Does not require to share a key

Alice, Bob (and Trudy)



- Suppose Bob's key pair is < K_1, K_2 >.
- Everybody knows key K_1 (also denoted by e)
- Only Bob knows (Alice does not know) key K_2 (also denoted by d)
- Note that Trudy might be allowed to know K_1
- K_1 is known as **public key**
- K_2 is known as **private key**

Public key cryptography

- We assume that Bob has two keys:
  - A public key e
  - A private key d
- Only Bob knows d
- Everyone knows e (it is public!)
- If Alice wants to send a message m to Bob
  - Alice encrypts m by using e
    - $c = E_e(m)$
  - Bob decrypts the message by using d
    - $m = D_d(c)$
  - Everyone can encrypt messages but only Bob can decrypt them
- E, D, e, and d must be such that
  - $D_d(E_e(m)) = m$
  - $E_e(D_d(m)) = m$

Public key cryptography: signature

- We assume that Alice has two keys:
  - A public key e
  - A private key d
- Only Alice knows d
- Everyone knows e (it is public!)
- If Alice wants to authenticate a message m (to put a signature)
  - Alice computes a signature by using d
    - $s = D_d(m)$
  - Everyone can verify that the message is authentic by verifying the signature with e
    - $m = E_e(s)$
- E, D, e, and d must be such that
  - $D_d(E_e(m)) = m$
  - $E_e(D_d(m)) = m$

First: Some arithmetic

- From now on we will consider positive integers.

Modular addition

- x+y mod n, practical method:
  - Take the regular sum of x+y
  - Divide the result by n
  - Take the remainder
- Examples:
  - 3+5 mod 10 = ?
  - 8+7 mod 10 = ?
  - 5+5 mod 10 = ?
- Cryptography with modular addition:
  - Caesar cipher (recall?)
  - Addition of a constant mod n is used for encryption (it maps each digit to a different digit in a way that is reversible)
  - The constant is our secret key.
  - Decryption is done by subtracting the secret key modulo n.

Modular multiplication

- $x \cdot y \bmod n$, practical method:
  - Take the regular multiplication of x and y
  - Divide the result by n
  - Take the remainder
- Examples:
  - $2 \cdot 2 \bmod 6 = ?$
  - $1 \cdot 5 \bmod 6 = ?$
  - $3 \cdot 4 \bmod 6 = ?$
- The multiplicative inverse of x (written x^{-1}) is the number by which you multiply x to get 1 (mod n).
- Let us use modular multiplication and its inverse as a cipher.
- Encryption is done by multiplying by the key (mod n) and decryption is done by multiplying by the inverse of the key (mod n).
- Example:
- $n=7; x=2\ x^{-1} = ?$
- $x^{-1} = 4$

m=5 c=5*2 mod 7=3

3*4 mod 7 =5=m

c x^{-1}

Modular multiplication

- Finding a multiplicative inverse in mod n arithmetic (if it exists!) is not straightforward, especially if n is very large.
- Exhaustive search, that is trying all the values smaller than n, works. In fact $x*y \bmod n = x*(y+an) \bmod n$, for any positive integer a.
- However exhaustive search requires too much time for large n.
- An efficient algorithm is called Euclid's Algorithm, we don't give details:
  - Given x and n, it finds the number y (if it exists!) such that
    $x \cdot y \bmod n = 1$ (x,y and n are integers).
  - It also finds the greatest common divisor (gcd) between two numbers.

Modular multiplication

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

- Consider the mod 6 multiplication table
- Multiply by 1 and 5 could work as a cipher because there is a one-to-one correspondence between each digit of the plaintext and each digit of the ciphertext. What are the inverse of 1 and 5 w.r.t. multiplication mod 6 ?
- Multiply by 2, 3, and 4 cannot work as a cipher because different digits of the plaintext would be encoded with the same digit in the ciphertext
  - Key = 2: m = 2 -> c = 4; m = 5 -> c = 4. From c = 4 we cannot know if the plaintext was 2 or 5.
  - Key = 3: half of the digits are encoded as 0, and half as 3

Modular multiplication

- Why 1 and 5?
  - Because they do not share any common factor with 6, other than 1. Or equivalently, they are such that the greatest common divisor with 6 is 1, i.e., gcd(*1*, *6*) = 1, gcd(*5*, *6*) = 1.
  - We say they are (**relatively prime**) or **co-prime** with 6.
  - 2 and 3 are factors of 6 (gcd(*2*, *6*) = 2 , gcd(*3*, *6*) = 3).
  - 4 has the factor 2 in common with 6 (gcd(*4*, *6*) = 2).
- The multiplicative inverse of x modulo n exists if and only if x and n are co-prime. (Homework: are you able to show this?)
- Given n, we have to find a key that has a multiplicative inverse mod n
  - We have to find the numbers <u>smaller</u> than n that are co-prime with n
- How many (integer) numbers <u>smaller</u> than n are relatively prime to n?

Modular multiplication

How many (integer) numbers <u>smaller</u> than n are co-prime to n?

- $\phi(n)$ is called the (Euler) totient function of n (from total and quotient) and denotes the number of integer numbers smaller than n that are co-prime to n.
- If n is prime $\phi(n) = n-1$. All the numbers smaller than n are co-prime with n.
- If n is the product of two primes p and q, then $\phi(n) = (p-1)(q-1)$.
  - All the numbers smaller than n but the multiples of p and q, are co-prime with n.
  - There are p multiples of q, and q multiples of p. Therefore we exclude $p + q - 1$ number (0 is a multiple of every number, and it is counted twice).
  - $\phi(n) = pq - (p + q - 1) = (p-1)(q-1)$.
  - Example: 15=3*5
    - Multiples of 3: 3,6,9,12,0.
    - Multiples of 5: 5,10,0.
    - Co-prime of 15 are 1,2,4,7,8,11,13,14 the number is 8=(2*4)

Modular exponentiation

- X^y mod n, practical method:
  - Take the regular exponentiation of X^y
  - Divide the result by n
  - Take the remainder
- Examples:
  - 3^2 mod 10 = ?
  - 2^2 mod 10 = ?
  - 2^6 mod 10 = ?
  - 2^{12} mod 10 = ?
- Note that 2^2 mod 10 and 2^{12} mod 10 are different.
- In general 2^x mod n and 2^{x+n} mod n are different.
- If we use the exponentiation for encrypting. How would you decrypt? Is there an exponentiative inverse like there is a multiplicative inverse?
- Just like with multiplication, the answer is sometimes.

Modular exponentiation

	0	1	2	3	4	5	6	7	8	9	10	11	12
0		0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	2	4	8	6	2	4	8	6	2	4	8	6
3	1	3	9	7	1	3	9	7	1	3	9	7	1
4	1	4	6	4	6	4	6	4	6	4	6	4	6
5	1	5	5	5	5	5	5	5	5	5	5	5	5
6	1	6	6	6	6	6	6	6	6	6	6	6	6
7	1	7	9	3	1	7	9	3	1	7	9	3	1
8	1	8	4	2	6	8	4	2	6	8	4	2	6
9	1	9	1	9	1	9	1	9	1	9	1	9	1

- Consider the mod 10 exponentiation table
- Exponentiation by 1, 3, 5, 7, 9, and 11 could work as a cipher. The other exponentiations would not.
- Note that:
  - Columns 1, 5, and 9 are identical
  - Columns 2, 6, and 10 are identical
  - Columns x and x + 4 are identical
  - $\phi(10) = 4$
- An amazing property of the (Euler) totient function is the following: if n is a prime number or the product of two prime numbers:
  - $X^y \bmod n = X^{y \bmod \phi(n)} \bmod n$
- In particular, if $y \bmod \phi(n)=1$, then for any number x, $x^y \bmod n = x \bmod n$

RSA
Overview

- RSA stands for, Rivest, Shamir, and Adleman.
- It is a public key cryptographic algorithm that does encryption, decryption, signing, verification of signature.
- Public key = asymmetric.
- The key length is variable. Anyone using RSA can choose a long key for enhanced security, or a short key for efficiency.
  - The most commonly used key length for RSA is 512 bits.
- The block size in RSA (the chunk of data to be encrypted) is also variable.
  - The plaintext block must be smaller than the key length.
  - The ciphertext block will be the length of the key.
- RSA is much slower to compute than secret key algorithms like DES. As a result, RSA does not tend to get used for encrypting long messages. Mostly it is used to encrypt a secret key, and then secret key cryptography is used to actually encrypt the message.

RSA
Encoding, decoding and signing

- To generate public and private keys
  - Choose two large prime numbers p and q (probably around 256 bits each)
  - $n = p \cdot q$
  - Choose a number e that is co-prime with $\phi(n) = (p-1)(q-1)$
    - Public key: (e,n)
  - Find the number d that is the multiplicative inverse of e mod $\phi(n)$.
    - Private key: (d,n)
  - Notes:
    - p and q remain secret.
    - It is practically impossible to compute p and q from n.
- To encode a message m (smaller than n)
  - $c = m^e \bmod n$
- To decode c
  - $m = c^d \bmod n$
- To sign a message m (smaller than n)
  - $s = m^d \bmod n$
- To verify the signature
  - $m = s^e \bmod n$

RSA
Correctness

- Recall:
  - $n = p \cdot q$
  - $\phi(n) = (p-1)(q-1)$
- We have chosen e and d such that
  - $1 = (d \cdot e) \mod \phi(n)$,
  - Recall that:
    - if $1 = y \mod \phi(n)$, then for any number x, $x^y \mod n = x \mod n$
  - Therefore:
    - For any number x, $x^{de} \mod n = x \mod n$
- To decode we compute:
  - $c^d \mod n = (m^e)^d \mod n = m^{ed} \mod n = m \mod n = m$

The plaintext block must be smaller than the key length n.

- To verify a signature we compute
  - $s^e \mod n = (m^d)^e \mod n = m^{de} \mod n = m \mod n = m$

RSA
Security

- We cannot formally show that RSA is secure
- Lots of smart people have been trying to figure out how to break RSA, and they haven't come up with anything yet.
  - Many algorithms and heuristics are known but they are not efficient enough.
  - The largest such number factored was RSA-768 (768 bits) on December 12, 2009. This factorization was a collaboration of several research institutions, spanning two years and taking the equivalent of almost 2000 years of computing on a single-core 2.2 GHz AMD Opteron.

RSA
Security

- The only assumption is that factoring a big number is hard (it seems that the most difficult case is when n is a semi-prime number, i.e., the product of two prime numbers).
- No algorithm is known that can factor all integers in polynomial time, i.e., that can factor b-bit numbers in time $O(b^k)$ for some constant k, neither the existence nor non-existence of such algorithms has been proved.
- In particular, it is also not known whether the problem is NP-complete.

RSA
Security

- If we are able to factorize a large number we can break RSA.
- Suppose you are given Alice's public key (e,n).
- If Trudy could find the inverse of e w.r.t. exponentiation mod n, then she has the Alice's private key (d,n).
- How can she find it? Alice did it by knowing the factors of n, allowing her to compute $\phi(n)$.
  - Alice found (for instance by using the Euclid's Algorithm) the number that was e's multiplicative inverse mod $\phi(n) = (p-1)(q-1)$.
  - Trudy can do what Alice did if she can factor n to get p and q.

RSA
Security

- A possible attack is based on the fact that anyone knows the public key and hence anyone can encode.
- Suppose that Bob is sending a message m to Alice by using Alice's public key and m is in a small finite set of messages.
- Trudy can sniff the encoded message, encode all the possible messages that can be sent, and compare the results with the sniffed message.
- Workaround: concatenate a large random number to the message. In this way we extend the space of possible messages.

RSA
Security

- When encrypting with low encryption exponents (e.g., $e = 3$) and small values of the m, (i.e., $m < n^{1/e}$) the result of m^e is strictly less than n. In this case, ciphertexts can be easily decrypted by taking the e-th root of the ciphertext over the integers.

- Workaround: concatenate a large random number to the message. In this way we avoid the above issue.

RSA - Efficiency

- The operations that need to be routinely performed with RSA are:
  - Encryption
  - Decryption
  - Generating a signature
  - Verifying a signature.

- They must be efficient!
- Finding an RSA key (choosing n, d, and e) is done less frequently
  - It needs to be reasonably efficient but it isn't as critical as the other operations

RSA - Efficiency

- Encryption, decryption, signing, and verifying signatures requires to compute the exponentiation of a number mod n.
  - x^y mod n
- Straightforward way:
  - r = x;
  - Repeat (y – 1) times:
    - r := r · x;
  - Divide r by n and output the remainder
- This is far too slow!
  - Even representing such numbers (for instance 150-digit number to a 150-digit power) in main memory is not trivial.

RSA - Efficiency

- We compute the modular reduction after each multiplication step
  - r = x;
  - Repeat (y – 1) times:
    - r := r · x;
    - Divide r by n and assign the reminder to r;
  - Output r
- This reduces the problem to y-1 small multiplies and y-1 small divides instead of y-1 big multiplies and 1 big divide.
- Example 123^{54} mod 678
  - 123^2 : $123 \cdot 123 = 15129 \equiv 213 \mod 678$ (short cut for 15129 mod 678 = 213)
  - 123^3 : $123 \cdot 213 = 26199 \equiv 435 \mod 678$
  - 123^4 : $123 \cdot 435 = 53505 \equiv 621 \mod 678$
  - ....

- It is still unacceptable for exponents of the size used with RSA.