RSA - Efficiency

- We can do much better if the exponent is a power of 2 that is: X^y and $y = 2^z$
  - r = x
  - Repeat z times
    - r := r · r mod n
  - Output r
- Example x^{32}= (x)^2^5
  - x^2, x^4, x^8, x^{16}, x^{32}
- Example 123^{32} mod 678
  - $123^2 = 123 \cdot 123 = 15129 \equiv 213 \bmod 678$
  - $123^4 = 213 \cdot 213 = 45369 \equiv 621 \bmod 678$
  - $123^8 = 621 \cdot 621 = 385641 \equiv 537 \bmod 678$
  - $123^{16} = 537 \cdot 537 = 288369 \equiv 219 \bmod 678$
  - $123^{32} = 219 \cdot 219 = 47961 \equiv \boxed{501} \bmod 678$

RSA - Efficiency

- What if y is not a power of 2? From previous computations, if you know how to compute x^y, then you can easily compute: $x^{2y} = (x^y)^2$; $x^{y+1} = x \cdot x^y$; $x^{2y+1} = x \cdot (x^y)^2$
- Consider the binary representation of y
  - E.g.: y = 54 = 110110
- Compute x raised to the sequence of powers where each successive power concatenates one more bit of y (from left to right).
  - E.g.: 1 ; 11 ; 110 ; 1101 ; 11011; 110110
- Each successive power is either twice the preceding power or one more than twice the preceding power. Computing x^y is done by repeated squaring, and multiplying by x according to such sequence.
- Example 123^{54} mod 678
  - $123^1 \equiv 123 \bmod 678$
  - $123^2 = 123 \cdot 123 = 15{,}129 \equiv 213 \bmod 678$
  - $123^3 = 123^2 \cdot 123 = 213 \cdot 123 = 26{,}199 \equiv 435 \bmod 678$
  - $123^6 = (123^3)^2 = 435^2 = 189{,}225 \equiv 63 \bmod 678$
  - $123^{12} = (123^6)^2 = 63^2 = 3{,}969 \equiv 579 \bmod 678$
  - $123^{13} = 123^{12} \cdot 123 = 579 \cdot 123 = 71{,}217 \equiv 27 \bmod 678$
  - $123^{26} = (123^{13})^2 = 27^2 = 729 \equiv 51 \bmod 678$
  - $123^{27} = 123^{26} \cdot 123 = 51 \cdot 123 = 6{,}273 \equiv 171 \bmod 678$
  - $123^{54} = (123^{27})^2 = 171^2 = 29{,}241 \equiv 87 \bmod 678$

- The number of multiplies and divides rises linearly with the length of the exponent in bits rather than with the value of the exponent itself

RSA - Efficiency

Exercise: $12^{43} \bmod 50$

- $43 = 101011$
- Consider the sequence: 1 ; 10; 101; 1010; 10101; 101011
- $12^1 \equiv 12 \bmod 50$
- $12^2 = 12 \cdot 12 = 144 \equiv 44 \bmod 50$
- $12^4 = (12^2)^2 = 44^2 = 1936 \equiv 36 \bmod 50$
- $12^5 = (12^4) \cdot 12 = 36 \cdot 12 = 432 \equiv 32 \bmod 50$
- $12^{10} = (12^5)^2 = 32^2 = 1024 \equiv 24 \bmod 50$
- $12^{21} = (12^{10})^2 \cdot 12 = (24)^2 \cdot 12 = 6912 \equiv 12 \bmod 50$
- $12^{43} = (12^{21})^2 \cdot 12 = (12)^2 \cdot 12 = 1728 \equiv 28 \bmod 50$

RSA – Generating keys

- Most uses of public key cryptography do not require frequent generation of RSA keys.
- It does not need to be as efficient as the operations that use the keys.
- However, it still has to be reasonably efficient.

RSA – Generating keys

Recall: to generate public and private keys

- Choose two large prime numbers p and q
- $n = p \cdot q$
- 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.
  - We suppose it is practically impossible to compute p and q from n.

RSA – Generating keys

Choose two large prime numbers p and q

- There is an infinite number of prime numbers. However, they thin out as numbers get bigger and bigger.
- Prime Number Theorem: says that for large enough n, the probability that a random integer not greater than n is prime is very close to 1 /ln(n).
- Therefore the probability of a randomly chosen number n being prime is approximately 1/ln(n), where ln(n) rises linearly with the size of the number represented in bits.
  - For a ten-digit (digit is a number from 0 to 9) number, there is about one chance in 23 of it being prime (because $\ln 10^{10} = 10 \ln 10 \approx 23$).
  - For a hundred-digit number (like in RSA), there is about one chance in 230.
- We choose a random number, and test if it is prime.
  - On the average, we'll only have to try 230 of them before we find one that is a prime.
- How do we test if a number n is prime?

RSA – Generating keys

- Very very naïve method:
  - Divide n by all numbers <n and check if the remainder is 0.
- Very naïve method:
  - Divide n by all numbers <n/2 and check if the remainder is 0.
- Naïve method:
  - Divide n by all numbers smaller or equal than the square root of n and check if the remainder is 0.
  - we can also skip all the even numbers greater than 2.
- …. You can also think about smarter ideas.
- However for each of these methods, if n is big, the computational time needed is too long even for generating keys.
- Unfortunately there is no known practical way for absolutely determining that a number of big size is prime (however there exists a polynomial time algorithm with complexity $\tilde{O}((\log n)^6)$).
- Fortunately, there are tests for determining that a number is **probably** prime, and the more time we spend testing a number the more assured we can be that the number is prime.
- The cost of a mistake (i.e. if n is actually not prime) would be that:
  - RSA might fail, i.e. the message cannot be decrypted.
  - Someone might be able to compute a private exponent (d) with less effort.

RSA – Generating keys

We use the Euler and Fermat Theorems

- **Euler Theorem.**
  - If a is co-prime to n, then $a^{\phi(n)} \equiv 1 \bmod n$. (This is still a short cut for $a^{\phi(n)} \bmod n = 1$)

Recall that if n is a prime number, then $\phi(n) = n - 1$. We have the following particular case.

- **Fermat (little) Theorem**.
  - If n is prime and $0 < a < n$ (if n is prime then any $a<n$ is always co-prime to n), then $a^{n-1} \equiv 1 \bmod n$.

Therefore, we can test if a number n is prime by checking whether $a^{n-1} \equiv 1 \bmod n$.

However, the Fermat theorem only gives a necessary condition!

Does $a^{n-1} \equiv 1 \bmod n$ holds also when n is **not** prime?

RSA – Generating keys

- In other words, consider the following statements:
  - A = {n is prime}
  - B = {for any $0 < a < n$ we have $a^{n-1} \equiv 1 \bmod n$}
- By the Fermat theorem:
  - A => B
  - NOT(B) => NOT(A)
- Does it implies that
  - NOT(A) => NOT(B) ?
  - NO!
    - Indeed, there exist numbers n that are not prime and such that $a^{n-1} \equiv 1 \bmod n$
    - However, they are pretty rare.

RSA – Generating keys

- A primality test for a number n is:
  - Pick a number a < n,
  - Compute a^{n-1}, and check whether the answer is 1 mod n.
  - If it is not 1 mod n, n is certainly not prime.
  - If it is 1 mod n, n may or may not be prime.
- Example n = 221.
  - Randomly pick a = 38.
  - $a^{n-1} = 38^{220} \equiv 1 \bmod 221$.
  - Is 221 a prime?
  - If we take another a, say 24:
    - $a^{n-1} = 24^{220}$ and $24^{220} \bmod 221 \neq 1$.
    - 221 is not prime, infact 221= 13*17!
  - 38 is called a **Fermat liar**.
- If n is a randomly generated number of about 100 digits, the probability that n is not prime but $a^{n-1} \equiv 1 \bmod n$ is about $1/10^{13}$.
- In many cases we can accept this risk.

RSA – Generating keys

- Anyway, if the risk of 1 in 10^{13} is unacceptable, the former primality test can be made more reliable.
- A straightforward way is to try multiple values of a.
  - If they are independent, the probability is $1/10^{13t}$, where t is the number of values of a used.
- Unfortunately, there exist numbers n which
  - Are not prime,
  - satisfy $a^{n-1} \equiv 1 \bmod n$ for all values of a which are co-prime with n.
  - They are called **Carmichael numbers**.
  - Carmichael numbers are extremely rare.
- Nevertheless, there exists primality test that detects non-primes (even Carmichael numbers) with high probability and negligible additional computation. However we do note see them in this course.

RSA – Generating keys

Finding d and e given p and q

- We choose any number e that is co-prime to (p-1)(q-1), and then we find the number d such that $ed \equiv 1 \bmod \phi(n)$. This last step can be done by using the Euclid's algorithm.
- Two strategies to choose e,p and q such that e is co-prime to (p-1)(q-1).
  - After p and q are selected, choose e at random. Test whether e is co-prime to (p-1)(q-1) by using the Euclid's Algorithm. If not, select another e.
  - First choose e, then select p and q so that (p-1) and (q-1) are guaranteed to be co-prime to e.
    - If e is chosen to be small, or easy to compute, then the operations of encryption and signature verification become much more efficient.

RSA – Generating keys

- As far as we know, RSA is no less secure if e is always chosen to be the same number.
- If e is chosen to be small, or easy to compute, then the operations of encryption and signature verification become much more efficient.
- Usually e is chosen as a small constant and then d is derived accordingly.
- This makes public key operations faster while leaving private key operations unchanged.
- A value popular for e is 3.

RSA – Generating keys

Choosing 3

- It is the smallest possible e
  - 2 is not relatively prime to (p-1)(q-1) (which must be even because p and q are both odd).
- With 3, public key operations (encryption and verification of signature) require only two multiplies.
  - Using 3 as the public exponent maximizes performance.
- As far we know, using e = 3 does not weaken the security of RSA if some practical constraints on its use are followed.

RSA – Generating keys

Problems with using e = 3

(possible threat) If a message m to be encrypted is smaller than the cube root of n then m^3 mod n = m^3

- The encrypted message could be decrypted simply by taking a cube root.
- Workaround: padding each message with a random number before encryption.

RSA – Generating keys

Choosing e=3 works only if 3 is co-prime to $\phi(n)$ $=(p-1)(q-1)$ (in order for it to have an inverse d).

- Clearly, (p-1) and (q-1) must each be co-prime to 3.
- To ensure that p-1 is co-prime to 3, we choose $p \equiv 2 \bmod 3$.
  - This implies $p-1 \equiv 1 \bmod 3$.
  - Similarly, we want q such that $q \equiv 2 \bmod 3$.
- We can do it by choosing a random number, multiplying by 3 and adding 2, and using that as the number we will test for primality.
- Since the prime must be odd, we should start with an odd number, multiply by 3 and add 2.

Diffie-Hellman

- The Diffie-Hellman cipher is the oldest public key system still in use.
- It is less general than RSA
  - it does neither encryptions nor signatures
- It offers better performance than RSA for what it does.
- What does it do?
- Diffie-Hellman allows two individuals to agree on a shared key, by using a public communication channel.
  - Neither Alice nor Bob start out with any secrets, yet after the exchange of two messages that Trudy can overhear, Alice and Bob will know a secret number.
  - Once they know a secret number, they can use cryptography with secret key (i.e., the secret number) for encryption.
- Weakness: although two individuals can agree on a shared secret key, there is no authentication, which means that Alice might be establishing a secret key with Trudy.

Diffie-Hellman

- Let us consider two numbers p and g
  - p is a large prime number
  - g is a number smaller than p (not necessarily prime)
    - There are some restrictions that are not important for a basic understanding of the algorithm.
  - p and g are known beforehand and can be publicly known.
- Once Alice and Bob agree on p and g, each chooses a 512-bit number at random and keeps it secret.
  - Alice chooses S_A and Bob S_B.
  - Alice computes $T_A = g^{(S_A)} \bmod p$
  - Bob computes $T_B = g^{(S_b)} \bmod p$.
  - Alice sends T_A to Bob and Bob sends T_B to Alice.
  - Alice computes $T_B^{(S_A)} \bmod p$ and Bob computes $T_A^{(S_B)} \bmod p$
    - $T_B^{(S_A)} \bmod p = T_A^{(S_B)} \bmod p = g^{(S_A * S_B)} \bmod p$.

Diffie-Hellman

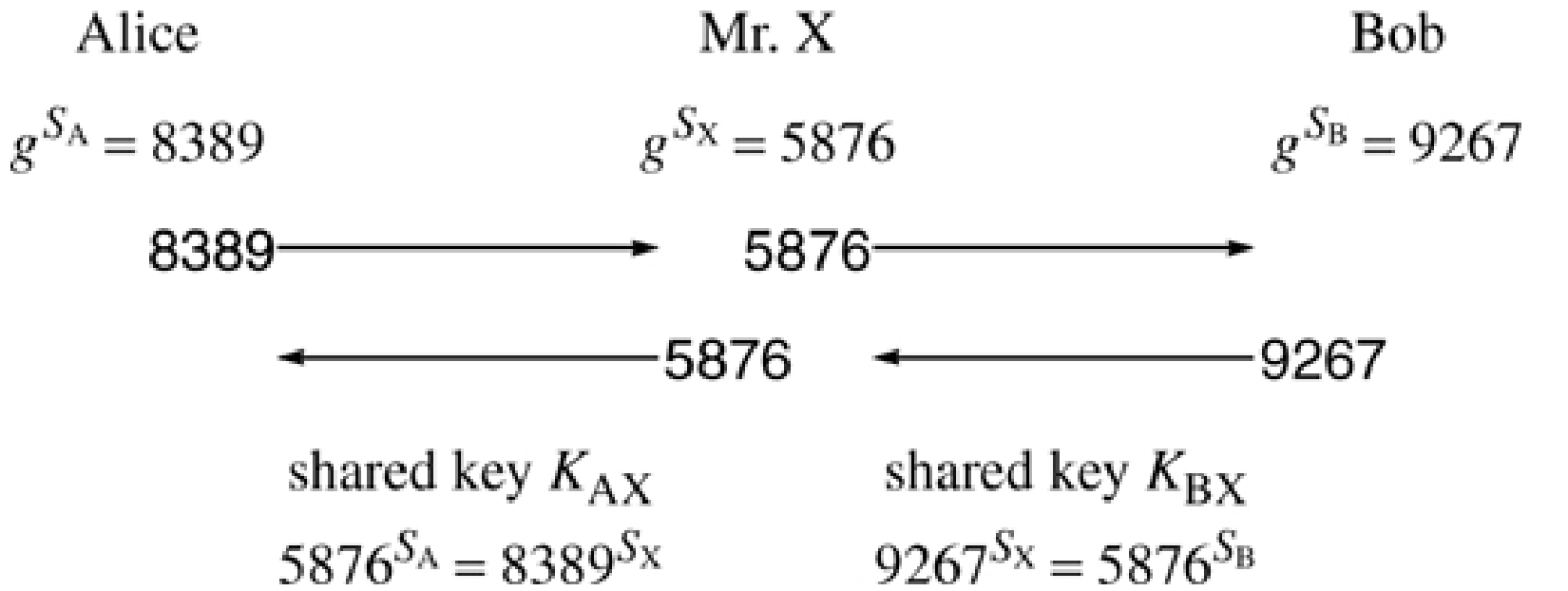
- Nobody else can compute g^(S_A S_B) mod p in a reasonable amount of time even though they know g^(S_A) mod p and g^(S_B) mod p.
- If they could compute **discrete logarithms**, i.e. figure out S_A based on seeing g^(S_A) mod p, then they could figure out the Alice/Bob shared key.
- We assume they can't compute discrete logarithms in an efficient way.
  - Moreover polynomial time algorithms are not known.

Diffie-Hellman

Man-in-the-Middle Attack:

- If Alice receives T_B indirectly, there is no way for her to know for sure whether the number came from Bob.
- She will establish a secret key with whoever transmitted T_B, but it certainly might not be Bob.
- Let's assume Alice is talking to X, who may or may not be Bob. Once Alice and X establish a secret key, they can encrypt all their messages so that only Alice and X can read them.
- Let us suppose that the first thing Alice and X exchange in their encrypted communication is a password that Alice and Bob have previously agreed upon, one password that Bob is to say to Alice, “The fish is green”, and one that Alice is to say to Bob, for instance “The moon sets at midnight”.
- If Alice receives the expected password from X, can she assume she is talking to Bob?

Diffie-Hellman: Man-in-the-Middle Attack



Alice and Bob agree on a p and g (the following operations are mod p),

- Alice chooses S_A and Bob S_B.
- Mr. X chooses S_x
- Alice computes $T_A = g\text{^}(S_A) = 8389$ and Bob computes $T_B = g\text{^}(S_b) = 9267$
- Alice sends T_A to Bob and Bob sends T_B to Alice
- Mr. X substitute $T_A = 8389$ and $T_B = 9267$ with $T_X = g\text{^}(S_X) = 5876$
- Alice computes $T_X\text{^}(S_A) = 5876\text{^}(S_A) = 8389\text{^}(S_X) = g\text{^}(S_X S_A) = K_{AX}$
- Bob computes $T_X\text{^}(S_B) = 5876\ \text{^}(S_B) = 9267\text{^}(S_X) = g\text{^}(S_X S_B) = K_{BX}$
- Mr. X shared a key K_{AX} with Alice and a key K_{BX} with Bob

Diffie-Hellman

- Now suppose Alice sends an encrypted message (to Mr. X) which includes the password “The fish is green”.
- Mr. X can decrypt the message because it is encrypted using the key he shares with Alice.
- Mr. X re-encrypts and transmits Alice's password to Bob, which reassures Bob that he is indeed talking to Alice, and Bob then transmits (encrypted, to Mr. X) “The moon sets at midnight”.
- Mr. X decrypts Bob's message, extracts the password, and re-encrypts and transmits the password to Alice.
- Both Alice and Bob think they are talking to each other.
- Using Diffie-Hellman alone, there's really nothing Alice and Bob can do to detect the intruder through whom they are communicating.
- This form of Diffie-Hellman is only secure against **passive attack** where the intruder just watches the messages but cannot change and/or block them.

Diffie-Hellman - Defenses Against Man-in-the-Middle Attack

Authenticated Diffie-Hellman exchange

- This vulnerability can be overcome with the use of digital signatures and public-key certificates (however we do not see it).