Fundamentals of Computer Security

Spring 2015

Radu Sion

Key Exchange
Public Key Cryptography

Public Key Cryptography

- Fundamentals
- RSA

Key Exchange

- Compute a common, shared key
  - Called a *symmetric key exchange protocol*
- Challenges:
  - I don’t know the other party
  - Alice and Bob vs. Eve (who eavesdroppes)

One Idea

- Alice: generates random **a**
- Bob: generates random **b**
- Alice sends: $\mathbf{m_a=g^a}$
- Bob sends: $\mathbf{m_b=g^b}$
- Alice does: $\mathbf{(m_b)^a} = \mathbf{g^{ba}} = \mathbf{key}$
- Bob does: $\mathbf{(m_a)^b} = \mathbf{g^{ab}} = \mathbf{key}$
- Does it work ?!!! Seems very simple !

Make it difficult for bad guy

- Discrete logarithm problem hardness:
  - Given integers *n* and *g* and prime number *p*, compute *k* such that $n = g^k \bmod p$
  - Solutions known for small *p*
  - Solutions computationally infeasible as *p* grows large

Diffie-Hellman

- Constants: prime p, integer $g \neq 0, 1, p–1$
  - Known to all participants
- Alice chooses private key k_{Alice}, computes public key $K_{Alice} = g^{k_{Alice}} \bmod p$
- To communicate with Bob, Alice computes

  $K_{shared} = {K_{Bob}}^{k_{Alice}} \bmod p$
- To communicate with Alice, Bob computes

  $K_{shared} = {K_{Alice}}^{k_{Bob}} \bmod p$
  - It can be shown these keys are equal

A couple of problems ☺

- Man in The Middle (MITM)
  - solution: authenticate first
- Are we talking to the right person ?
- Forward Secrecy (PFS)
  - future compromise does not impact past
  - station to station (STS) Protocol

Public Key Encryption

How does Alice know whom it talks to ?
What if Mallory simply replaces the public key with something else (e.g., own) !

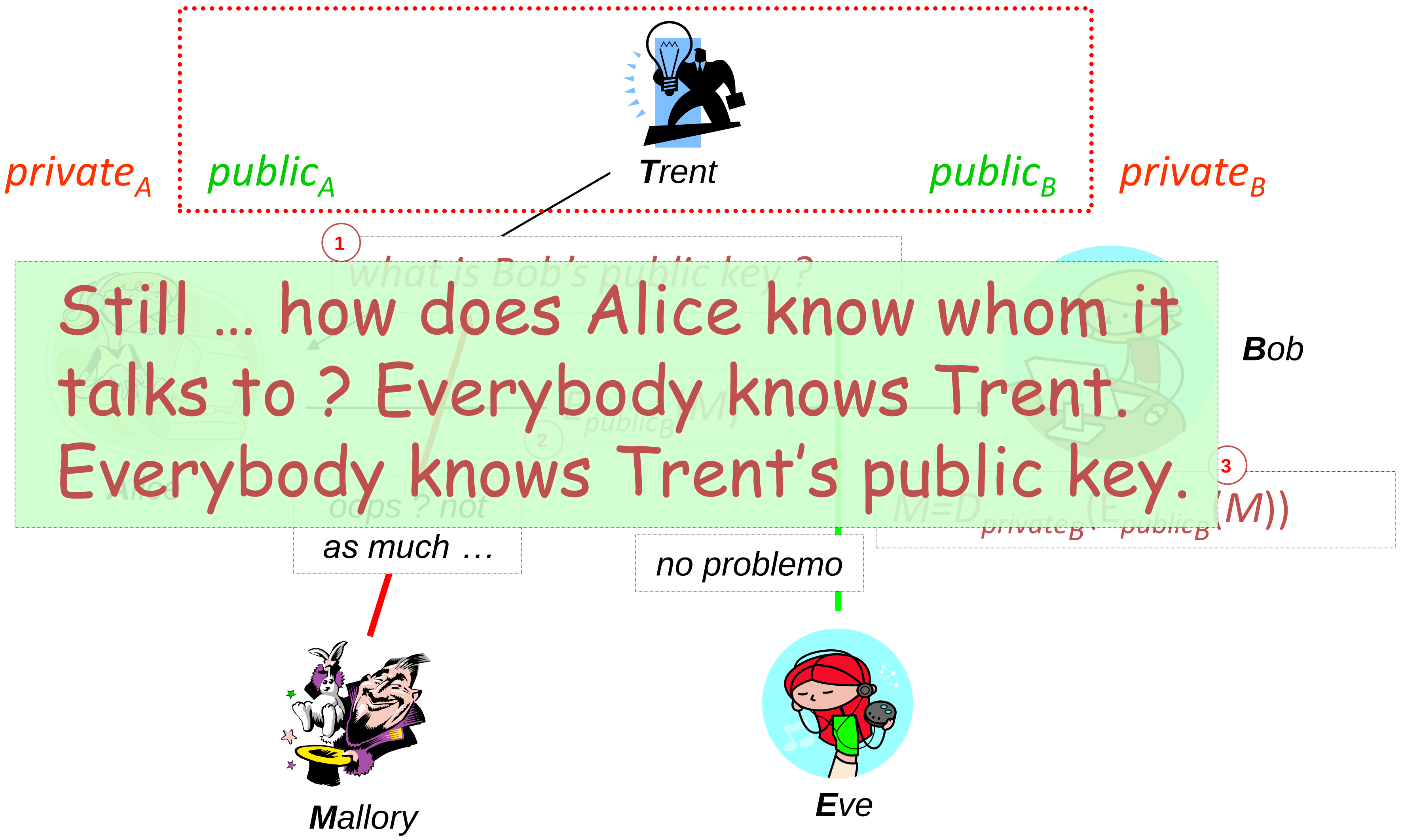
“Signatures”

Signature …

… something that **only signer can produce**
… and **everybody can verify**

verify = check for a unique association between the signer identity, text to be “signed” and the signature.

Certificate Authority



Still … how does Alice know whom it talks to ? Everybody knows Trent. Everybody knows Trent's public key.

What does this give us (1)

- Confidentiality
  - Only the owner of the private key knows it, so text enciphered with public key cannot be read by anyone except the owner of the private key
- Authentication
  - Only the owner of the private key knows it, so text enciphered with private key must have been generated by the owner (“digital signature”)
    - In real life: encrypt a hash of the text only !!!

What does this give us (2)

- Integrity
  - Enciphered letters cannot be changed undetectably without knowing private key
- Non-Repudiation
  - Message enciphered with private key came from someone who knew it

What we need to make it work

1. It must be computationally easy to encipher or decipher a message given the appropriate key
2. It must be computationally infeasible to derive the private key from the public key
3. It must be computationally infeasible to determine the private key from a chosen plaintext attack

Trapdoor

Trapdoor function (Diffie and Hellman 1976): function that is easy to compute but believed hard to invert without additional information (the "trapdoor"). We can then make the trapdoor the secret key ☺

Example: factoring primes (computing $n=p*q$ is easy, but given n, finding p and q is believed to be hard)

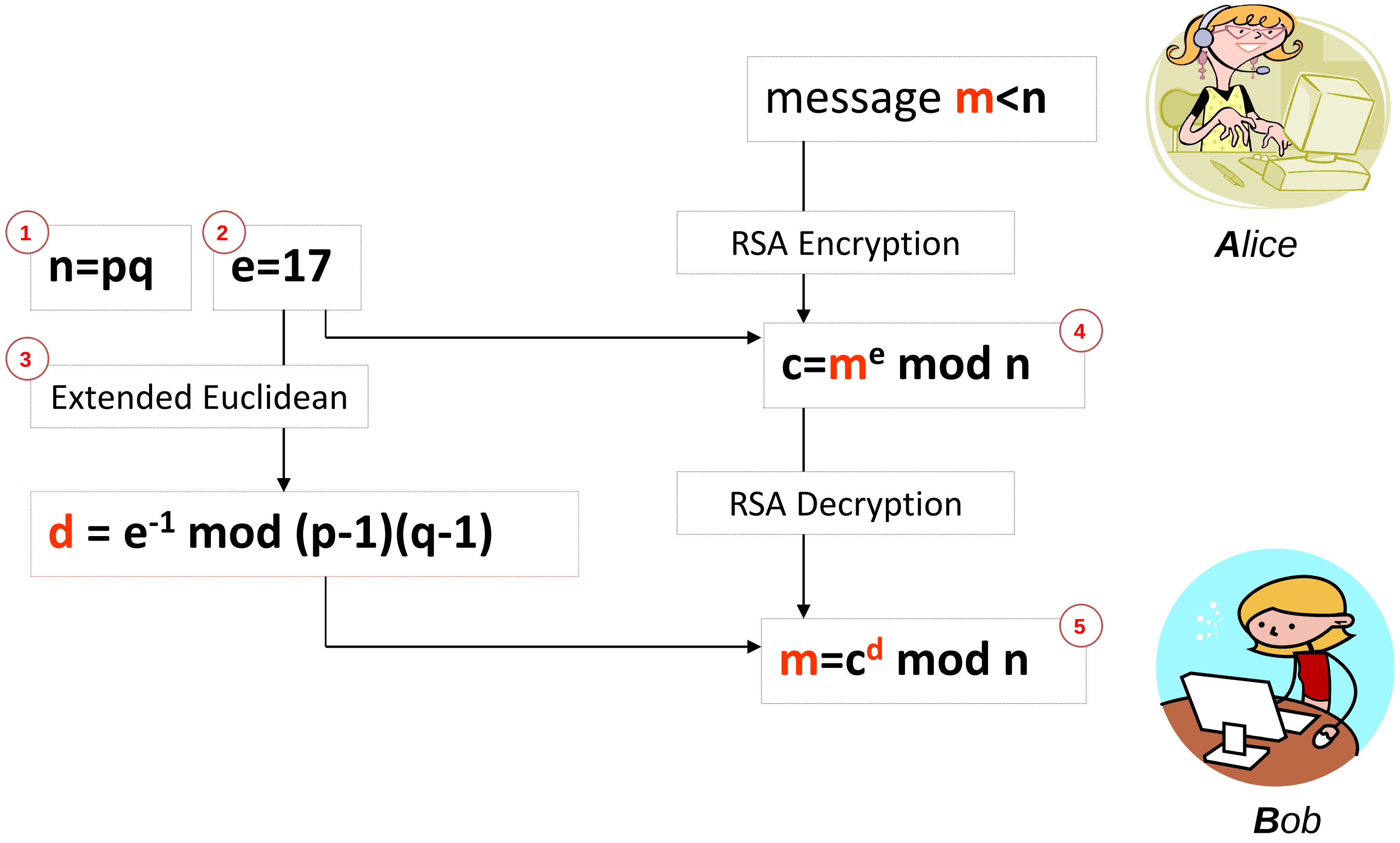
Things can be proven otherwise after a while: e.g., Merkle-Hellman Knapsack cryptosystem

Not all hard problems are trapdoors: e.g., discrete logarithm problem-related functions

RSA: Rivest, Shamir, Adelman

- Exponentiation cipher
- Relies on the difficulty of determining the number of numbers relatively prime to a large integer *n*
- Or equivalently, on the difficulty of factoring of large numbers into prime factors

Animated version



More boring version

- Key generation
  - Choose large primes p,q; let **n=pq**
  - Choose e relatively prime to (p-1)(q-1) (to have inverse !)
  - Public key **<e,n>**
  - Private key **<d,n>** where **d =** e^{-1} **mod (p-1)(q-1)**
    - Can do it fast using Extended Euclidean
- Encrypt: **c =** m^e **mod n**
- Decrypt: **m =** c^d **mod n**
- de = 1 mod (p-1)(q-1), so m = (me)d mod n
- Breakable if we can factor ☺

Larger Messages?

- Break message into pieces no greater in value than n-1 (why ?)
- Encrypt each part separately
- Use some sort of “chaining” to avoid block-related attacks
- Will likely use some padding etc. We discuss this later.

Ground Rules

- Attack: Exhaustive search for key
- Attack: Factoring n
- Timing Attacks: how long does encryption take ? – leaks information about the key
  - Solutions ?
- Attack: maintain dictionary of encrypted (public key) messages ("forward search")
- Common modulus problem
- etc. (many solved using smart padding)

RSA Common Modulus Problem

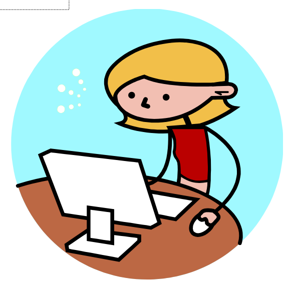
same modulus **n**

later, same message **m**

Alice

① $c_1 = m^{e_1} \bmod n$

① $c_2 = m^{e_2} \bmod n$



Bob

public network

Eve

Modified Extended Euclidean

② find r, s: $re_1 + se_2 = 1$

Extended Euclidean

③ $c_2^{-1} \bmod n$

④ $m = c_1^{r} * (c_2^{-1})^{-s} = m^{re_1 + se_2} \bmod n$

More Problems ☺

- Malleable (public key is known!)
- Probing
  - If I get e(m), I can check if m=m’
  - Solution: random pad – we discuss semantic security later
- Efficiency: can be made faster (modulo calculus tricks)
- Potential use interference: Encryption with Signatures
- Generating keys expensive
  - Select large primes
  - Find e relatively prime to (p-1)(q-1)
    - In practice, often e=3,5,17,65537
- For x<n no modular reduction takes place !!!
  - Also, given a signatures for m1, m2; can compute signature for (some) other messages

Back to Diffie Hellman

- Man in the middle solution: authentication and signatures on certain messages by first acquiring public/private key pairs
  - But why not use these keys to communicate then (instead of generating key every time) ?
    - Perfect forward secrecy ☺

Think about this

- Which one should go first:
  - –Authentication or Key Exchange ?