CSE509: (Intro to) Systems Security

Fall 2012

Radu Sion

Certificate Authorities
Signatures
Random Numbers

Cryptography Building Blocks (cont'd)

- Signatures
- Certificate Authorities (rev'd)
- Random Numbers

Signatures: overview

$$M=D_{private_A}(E_{public_A}(M))=D_{public_A}(E_{private_A}(M))$$

$private_A$ $public_A$ $public_B$ $private_B$

Alice

1

$S_A(M)=E_{private_A}(M)$ M

Bob

2

$M=D_{public_A}(S_A(M))$?

no problemo

***M**allory*

***E**ve*

Main idea

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.

What comes first: Encrypt <u>then</u> sign ?

- Mallory: replaces signature with own !
- Other problems with RSA !!!
- Not useful: only illegible ciphertext is non-repudiable

When a principal signs material that has already been encrypted, it should not be inferred that the principal knows the content of the message.

If a signature is affixed to encrypted data, then ... a third party certainly cannot assume that the signature is authentic, so non-repudiation is lost.

What comes first: Sign <u>then</u> encrypt ?

- Malicious Bob: sureptitious forwarding
  - decrypts $E_{public_B}(S_A(M))$
  - produces $E_{public_C}(S_A(M))$ and …
  - … sends it to Carol
  - Carol now believes Alice said M (to her)

How do we fix the mess ? Examples …

1. $E_{publicB}(S_A(M;B))$

2. $E_{publicB}(S_A(M;A;B))$

3. $S_A(E_{publicB}(S_A(M)))$

4. $E_{publicB}(S_A(E_{publicB}(M)))$

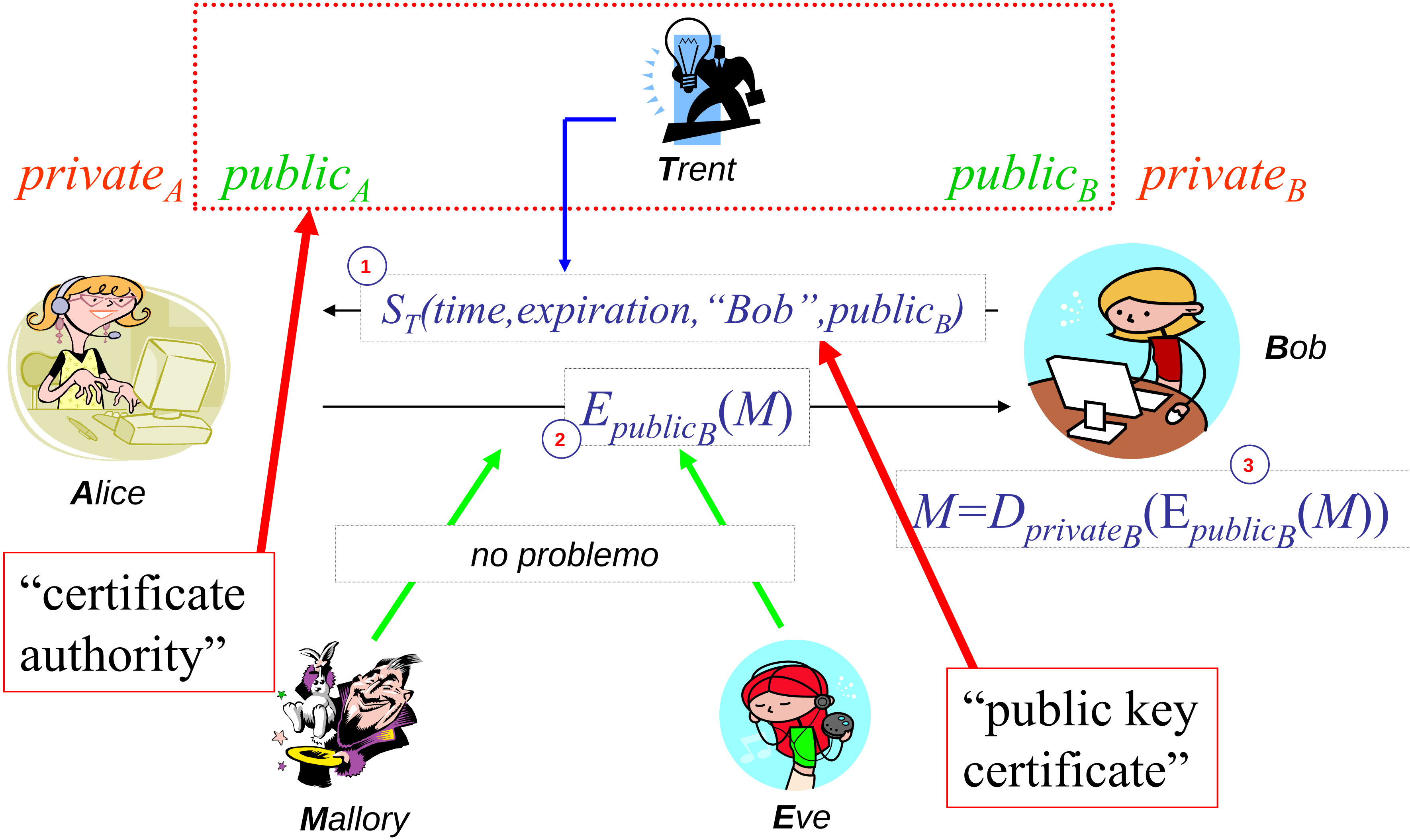
Can we sign/encrypt with same key-pair ?

In RSA S(m)=D(m). If we sign arbitrary stuff, e.g., m=E(M), then in effect we reveal M=D(E(M)) !

If you are a service, do not sign arbitrary stuff. Always sign a hash only !

Do not re-use key pair for different purposes!

Certificate Authority (Trent is one ☺)



Problem

Alice needs Trent's public key to validate received certificate:

- Needs to verify signature
- Problem pushed "up" a level
- Two approaches:
  - Merkle trees
  - Signature chains (* we discuss this *)

Cross Certification Issues

- Multiple CAs (validation issue)
  - Alice's CA is Trent; Bob's CA is Tim; how can Alice validate Bob's certificate?
  - Have Trent and Tim cross-certify
    - Each issues certificate for the other

Signature Chains

- If we have the following certificates:
  - Trent<<**Alice**>>
  - Tim<<Bob>>
  - Trent<<Tim>>
  - Tim<<Trent>>
- How does **Alice** validate Bob’s certificate ?
  - Get Trent<<Tim>>
  - Use public key of Trent to validate Trent<<Tim>>
  - Use Trent<<Tim>> to validate Tim<<Bob>>

Key Revocation

- Certificates invalidated *before* expiration
  - Usually due to compromised key
  - May be due to change in circumstance (*e.g.*, someone leaving company)
- Problems
  - Is entity revoking certificate authorized to do so ?
  - Does revocation propagate fast enough ?
    - network delays, infrastructure problems

CRLs

- *Certificate revocation list*
- Online Certificate Status Protocol (RFC 2560)
- X.509: only certificate issuer can revoke
- PGP
  - signers can revoke signatures
  - owners can revoke certificates
    - or allow others to do so

Reality Check: use <u>session</u> key for actual communication.

k k

Station to Station protocol. Use PKI for signatures, variant of Diffie Hellman for key exchange.

$E'_k(M)$

E' = symmetric encryption

Final Thoughts: Authentication vs. Key Exchange

- Which one should come first ?
- Should we maybe couple them ?
- Why ?

What is "Random" (cryptographically speaking)?

Cryptographically random numbers: a sequence of numbers X_1, X_2, … such that for any integer $k > 0$, it is **impossible** for an observer to predict X_k even if all of $X_1, \ldots, X_{k-1}$ are known.

Random Number Generators (RNGs)

True RNGs cannot be deterministically algorithmic in a closed system. “Anyone who considers arithmetic methods … is in a state of sin” (von Neuman)

There exists a certain “flow” of randomness/chaos that is preserved within the system.

True randomness can only (arguably) be achieved by a hardware device that extract randomness from real-life processes (e.g. thermal noise, RF).

What is "Pseudorandom"?

Idea: simulate a sequence of cryptographically random numbers but generate them by an algorithm.

Cryptographically pseudo-random numbers: a sequence of numbers X_1, X_2, … such that for any integer $k > 0$, it is **hard** for an observer to predict X_k even if all of X_1, …, X_{k-1} are known.

Pseudo Random Number Generators (PRNGs)

Approximating randomness (e.g., attempting to achieve a uniform distribution) – will always have period (finite output space), many other defects !

Examples:

- Linear congruential generators: $X_i = (aX_{i-1}+b) \bmod n$
- Mersenne Twister (for Monte Carlo simulations)
    - make it secure by using a hash

Best Pseudorandom Numbers

Strong mixing function: function of 2 or more inputs with each bit of output depending on some nonlinear function of all input bits:

- Examples: DES, MD5, SHA-1
- Use on UNIX-based systems:

```
(date; ps gaux) | md5
```

Cool Result

"pseudo-random number generators exist iff. one-way functions exist"

Johan Håstad, Russell Impagliazzo, Leonid A. Levin, Michael Luby: A Pseudorandom Generator from any One-way Function. SIAM J. Comput. 28(4): 1364-1396 (1999)