CPSC156: The Internet Co-Evolution of Technology and Society

Lectures 19,20, and 21:

April 5, 10, and 12, 2007

Cryptographic Primitives

Outline

- Motivation
- Symmetric-key encryption
- Public-key encryption
- Digital signature
- Certificates
- Cryptographic hash functions

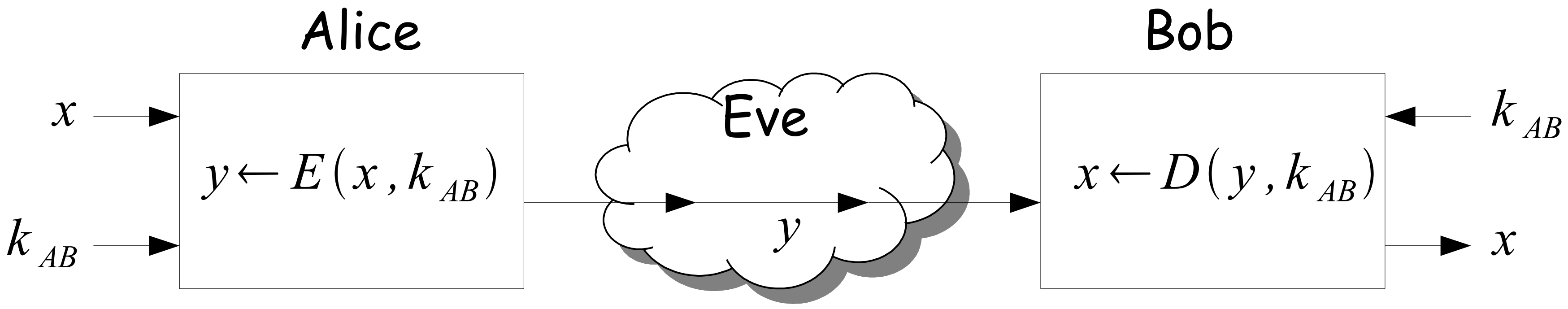
Motivation

Cryptographic primitives are basic building blocks of Internet security, including some popular browser-based security and privacy tools.

http://zoo.cs.yale.edu/classes/cs156/lectures/lecture22.ppt

http://crypto.stanford.edu/antiphishing

Symmetric-key Encryption (1)



x: plaintext	y: ciphertext
E: encryption function	D: decryption function

k_{AB}: Alice's and Bob's shared secret key

Symmetric-key Encryption (2)

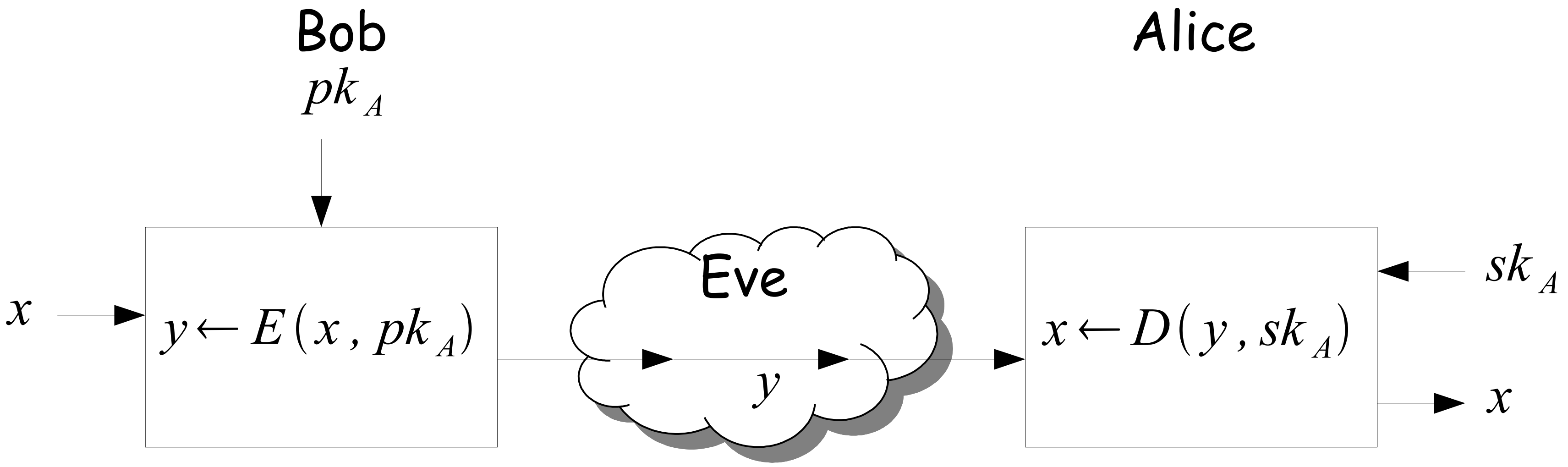
- For all messages x and all keys k, D(E(x, k), k) = x. That is, decryption “undoes” encryption perfectly, as long as the right key is used.
- If she does not know the secret key k_{AB}, the eavesdropper Eve cannot deduce the plaintext x from the ciphertext y, *even if she knows the encryption and decryption functions E and D.*
- Alice and Bob must keep k_{AB} secret and must run E and D in a private environment.
- Examples of widely used symmetric-key encryption schemes: AES (Advanced Encryption Standard) and DES (Data Encryption Standard)

Symmetric-key Encryption (3)

- Problem: Alice and Bob must meet to create a shared key k_{AB} before they can communicate privately over a public network (or they must belong to a group with a common "key manager"). Won't work for global-scale Internet commerce.
- Solution: Public-key encryption

Public-key Encryption (1)

- Alice runs the *key generator* to obtain a *public-key, secret-key pair* (pk_A, sk_A).
- She publishes "Alice: pk_A" and keeps sk_A secret.
- To communicate securely, Bob first looks up pk_A.



Public-key Encryption (2)

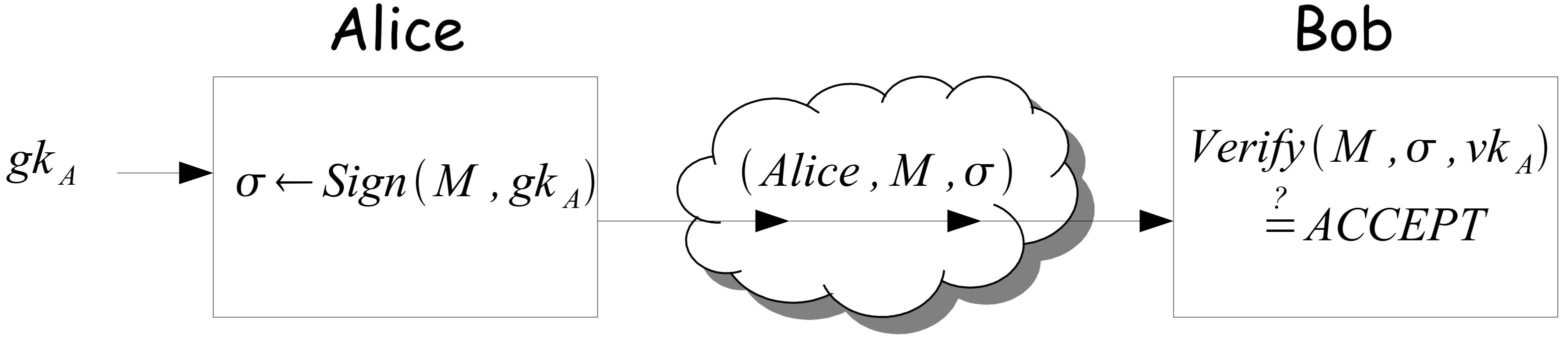
- With overwhelming probability, the key generator produces a different key pair each time it is run.
- For each pk, there is a unique sk, and vice versa. Given pk, it is *computationally infeasible* to find the corresponding sk.
- For all messages x and all key pairs (pk, sk), D(E(x, pk), sk) = x. Decryption “undoes” encryption perfectly if the corresponding key is used.
- If she does not know sk, the eavesdropper Eve cannot deduce the plaintext x from the ciphertext y, *even if she knows pk, E, and D.*

Public-key Encryption (3)

- This solves the scalability problem by enabling anyone who wants to receive secure communication to “sign up” unilaterally.
- The possibility of public-key cryptography was a radical conceptual breakthrough put forth by W. Diffie and M. Hellman in 1976.
- A well known and widely used public-key encryption system is the RSA system, named for its inventors R. Rivest, A. Shamir, and L. Adleman. Roughly speaking, the difficulty of computing sk from the corresponding pk stems from the fact that integer factoring is far harder than multiplication.

Digital Signatures (1)

- Alice runs the *key generator* to obtain a *verification-key, signing-key pair* (vk_A, gk_A).
- She publishes "Alice: vk_A" and keeps gk_A secret.
- To verify her signature, Bob first looks up vk_A.



Alice
Bob
gk_A
$\sigma \leftarrow Sign(M, gk_A)$
$(Alice, M, \sigma)$
$Verify(M, \sigma, vk_A) \overset{?}{=} ACCEPT$

Digital Signatures (2)

- With overwhelming probability, the key generator produces a different key pair each time it is run.
- For each vk, there is a unique gk, and vice versa. Given vk, it is *computationally infeasible* to find the corresponding gk.
- For all documents M and all key pairs (vk, gk), Verify(M, Sign(M, gk), vk) = *ACCEPT*.
- If he does not know gk, a forger cannot produce a valid signature (i.e., one that would cause the Verify procedure to accept), *even if he knows vk, Sign, and Verify*.

Discussion Point

Note that Alice's digital signature on digital document M_1 will be different from her digital signature on digital document M_2. Thus, digital signatures are not analogous to handwritten signatures.

Why is this necessary?

On “publishing“ name-key pairs

- <u>Problem:</u> An impersonator could generate a key pair (pk_I, sk_I) [resp., (vk_I, gk_I)] and then publish “*Alice*: pk_I” [resp., “*Alice*: vk_I”]. This would enable him to read *Alice*’s private communication [resp., forge *Alice*’s signature].
- <u>Solution:</u> Certified keys

Certificates

- We can leverage our confidence in the integrity of a single verification key.
- Let *CA* (for "certifying authority") be an entity with a valid, highly available verification key vk_{CA}. Verisign is an example of a *CA*.
- The *CA* can verify that Alice, Bob, Eve, *etc.*, have appropriate IDs and have not presented keys that are already owned by someone else. It can then publish "certified" name-key pairs:

$$(Alice, pk_A, Sign((Alice, pk_A), gk_{CA})),$$

$$(Bob, pk_B, Sign((Bob, pk_B), gk_{CA})),$$

$$(Eve, pk_E, Sign((Eve, pk_E), gk_{CA})), \ldots$$

Cryptographic Hash Functions

- A "hash function" h maps an arbitrary-length input to a fixed-length (*e.g.*, 256-bit) output. Note that, of necessity, there are many inputs x_1, x_2, x_3, … that are mapped to the same hash value y.
- h is a (keyed) "cryptographic hash function" if

  - each user has a secret key k that he uses to compute $y = h_k(x)$;

  - it is computationally infeasible for someone who does not know k to find *any* x* such that $h_k(x^*) = y$, even if he knows h and y.
- Note that this is different from encryption, in which each ciphertext produced with a given key corresponds to a *unique* plaintext that must be recoverable.