CSE 433S: Introduction to Computer Security

Key Exchange and Asymmetric Crypto

Washington University in St. Louis

Slides contain content from Professor Dan Boneh at Stanford University

Review

- What is authenticated encryption?
- In real systems, it is always possible to have confidentiality without integrity?
- What are the different ways to combine MAC and Symmetric Cipher? Which one is always correct?
- Given a network protocol, what is a common mistake that was shown in last class, how would you defend against them?

Key management

Problem: n users. Storing mutual secret keys is difficult

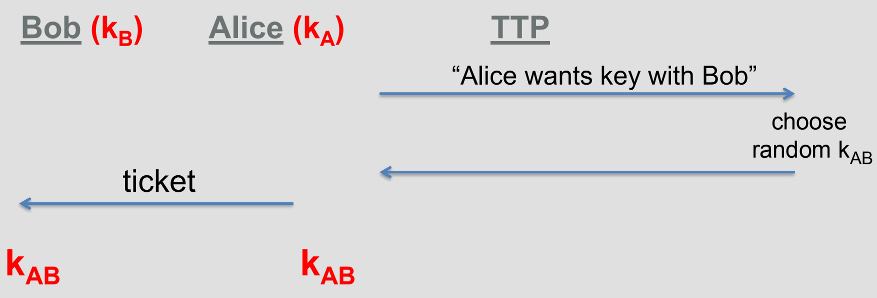
Total: O(n) keys per user

A better solution

Online Trusted 3rd Party (TTP)

Generating keys: a toy protocol

Alice wants a shared key with Bob.
Eavesdropping security only.



(E,D) a CPA-secure cipher

Generating keys: a toy protocol

Alice wants a shared key with Bob. Eavesdropping security only.

Eavesdropper sees: $E(k_A, \text{"A, B"} \parallel k_{AB})$; $E(k_B, \text{"A, B"} \parallel k_{AB})$

(E,D) is CPA-secure $\Rightarrow$
eavesdropper learns nothing about k_{AB}

But TTP needed for every key exchange, knows all session keys.

Basis of Kerberos system, good for corporate environment

Who would be this online TTP for the Internet? Gov? Which one?

Toy protocol:
Insecure against active attacks

Example: insecure against replay attacks

Attacker records session between Alice and merchant Bob
- For example a book order on the internet

Attacker replays session to Bob
- Bob thinks Alice is ordering another copy of book

Key question

Can we generate shared keys without an **online** trusted 3rd party?

Answer: yes!

Starting point of public-key cryptography:

- Merkle (1974), Diffie-Hellman (1976), RSA (1977)
- More recently: ID-based enc. (BF 2001), Functional enc. (BSW 2011)

Key exchange without an online TTP?

Goal: Alice and Bob want shared key, unknown to eavesdropper

- For now: security against eavesdropping only (no tampering)

The Diffie-Hellman protocol (informally)

Fix a large prime p (e.g. 600 digits, 2000 bits)

Fix an integer g in {1, …, p}

Alice choose random **a** in {1,…,p-1}

Bob choose random **b** in {1,…,p-1}

$$\mathbf{B^a} \pmod p = (g^b)^a = \mathbf{k_{AB} = g^{ab}} \pmod p = (g^a)^b = \mathbf{A^b} \pmod p$$

Security

Eavesdropper sees: p, g,
$A=g^a \pmod p$, and $B=g^b \pmod p$

The question, can Eve compute $g^{ab} \pmod p$??

More generally: define $DH_g(g^a, g^b) = g^{ab} \pmod p$

How hard is the DH function mod p?

How hard is the DH function mod p?

Suppose prime p is n bits long.
Best known algorithm (GNFS): run time $\exp(\tilde{O}(\sqrt[3]{n}))$

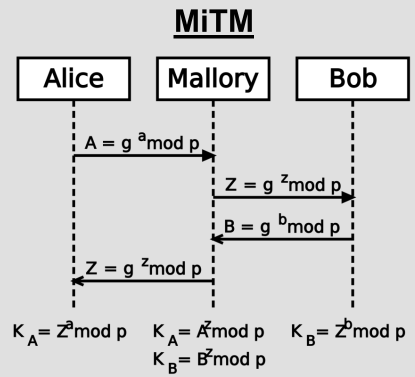
cipher key size	modulus size	Elliptic Curve size
80 bits	1024 bits	160 bits
128 bits	3072 bits	256 bits
256 bits (AES)	**15360** bits	512 bits

As a result: slow transition away from (mod p) to elliptic curves

Elliptic curve
Diffie-Hellman

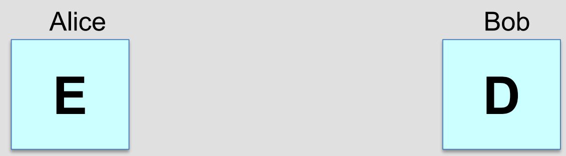
Insecure against man-in-the-middle

As described, the protocol is insecure against **active** attacks



Key Exchange based on Public Key Cryptography

Public key encryption



Public key encryption

Def: a public-key encryption system is a triple of algs. (G, E, D)

- G(): randomized alg. outputs a key pair (pk, sk)
- E(pk, m): randomized alg. that takes m∈M and outputs c ∈C
- D(sk,c): det. alg. that takes c∈C and outputs m∈M or ⊥

Consistency: ∀(pk, sk) output by G :

∀m∈M: D(sk, E(pk, m)) = m

Establishing a shared secret

Alice **Bob**

(pk, sk) ⟵ G()

"Alice", pk →

choose random $x \in \{0,1\}^{128}$

←

Security (eavesdropping)

Adversary sees **pk, E(pk, x)** and wants **x** ∈M

Semantic security ⇒

adversary cannot distinguish

{ pk, E(pk, x), x } from { pk, E(pk, rand), rand∈M }

⇒ can derive session key from x.

Note: protocol is vulnerable to man-in-the-middle

Insecure against man in the middle

As described, the protocol is insecure against **active** attacks

Alice **MiTM** **Bob**

(pk, sk) ⟵ G() (pk', sk') ⟵ G()

"Alice", pk →

choose random $x \in \{0,1\}^{128}$

← "Bob", E(pk, x) "Bob", E(pk', x)

Public key encryption: constructions

Constructions generally rely on hard problems from number theory and algebra

Further readings

- Merkle Puzzles are Optimal,
  B. Barak, M. Mahmoody-Ghidary, Crypto '09

- On formal models of key exchange (sections 7-9)
  V. Shoup, 1999

Relation to symmetric cipher security

Recall: for symmetric ciphers we had two security notions:

- One-time security and many-time security (CPA)
- We showed that one-time security $\not\Rightarrow$ many-time security

For public key encryption:

- One-time security $\Rightarrow$ many-time security (CPA)
  (follows from the fact that attacker can encrypt by himself)

*Public key encryption **must** be randomized*

Notation

From here on:

- N denotes a positive integer.
- p denote a prime.

Notation: $Z_n = \{0,1,2,3,4\ldots.N-1\}$

Can do addition and multiplication modulo N

Modular arithmetic

Examples: let $N = 12$

$9 + 8 = 5$ in $\mathbb{Z}_{12}$

$5 \times 7 = 11$ in $\mathbb{Z}_{12}$

$5 - 7 = 10$ in $\mathbb{Z}_{12}$

Arithmetic in $\mathbb{Z}_N$ works as you expect, e.g $x\cdot(y+z) = x\cdot y + x\cdot z$ in $\mathbb{Z}_N$

Modular inversion

Over the rationals, inverse of 2 is ½ . What about $\mathbb{Z}_N$?

Def: The **inverse** of x in $\mathbb{Z}_N$ is an element y in $\mathbb{Z}_N$ s.t. $x * y = 1$ in $\mathbb{Z}_N$, y is denoted x^{-1} .

Example: let N be an odd integer. The inverse of 2 in $\mathbb{Z}_N$ is

More notation

Def: $\mathbb{Z}_N^*$ = (set of invertible elements in $\mathbb{Z}_N$)
= $\{ x \in \mathbb{Z}_N : \gcd(x,N) = 1 \}$

Examples:

1. for prime p, $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\} = \{1, 2, \ldots, p-1\}$
2. $\mathbb{Z}_{12}^* = \{ 1, 5, 7, 11\}$

For x in $\mathbb{Z}_N^*$, can find x^{-1} using extended Euclid algorithm.

Fermat's theorem (1640)

Another way to compute the inverse, and more

Thm: Let p be a prime

$\forall x \in (Z_p)^*$: $x^{p-1} = 1$ in Z_p

Example: p=5. $3^4 = 81 = 1$ in Z_5

So: $x \in (Z_p)^* \Rightarrow x\cdot x^{p-2} = 1 \Rightarrow x^{-1} = x^{p-2}$ in Z_p

Another way to compute inverses, but less efficient than Euclid

Euler's generalization of Fermat (1736)

Def: For an integer N define $\phi(N) = |(Z_N)^*|$ (Euler's ϕ func.)

Examples: $\phi(12) = |\{1,5,7,11\}| = 4$; $\phi(p) = p-1$

For N=p·q: $\phi(N) = N-p-q+1 = (p-1)(q-1)$

Thm (Euler): $\forall\, x \in (Z_N)^*$: $x^{\phi(N)} = 1$ in Z_N

Example: $5^{\phi(12)} = 5^4 = 625 = 1$ in Z_{12}

Generalization of Fermat. ***Basis of the RSA cryptosystem***

The RSA trapdoor permutation

G(): choose random primes p,q ≈1024 bits. Set **N=pq.**
choose integers **e , d** s.t. $\mathbf{e \cdot d = 1 \pmod{\varphi(N)}}$
output pk = (N, e) , sk = (N, d)

F(pk, x): $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; **RSA(x) = x^e** (in Z_N)

F^{-1}(sk, y) = y^d ; $y^d = RSA(x)^d = x^{ed} = x^{k\varphi(N)+1} = \left(x^{\varphi(N)}\right)^k \cdot x = x$

The RSA assumption

RSA assumption: RSA is one-way permutation

For all efficient algs. A:

$$\Pr\left[\, A(N,e,y) = y^{1/e} \,\right] < \text{negligible}$$

where $p,q \xleftarrow{R}$ n-bit primes, $N \leftarrow pq$, $y \xleftarrow{R} Z_N^*$

Is RSA a one-way permutation?

To invert the RSA one-way func. (without d) attacker must compute:

x from $c = x^e \pmod N$.

How hard is computing e'th roots modulo N ??

Best known algorithm:
- Step 1: factor N (hard)
- Step 2: compute e'th roots modulo p and q (easy)

The factoring problem

Gauss (1805): *"The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic."*

Best known alg. (NFS): run time exp($\tilde{O}(\sqrt[3]{n})$) for n-bit integer

Current world record: **RSA-768** (232 digits)

- Work: two years on hundreds of machines
- Factoring a 1024-bit integer: about 1000 times harder
  - ⇒ likely possible this decade

Further reading

- A Computational Introduction to Number Theory and Algebra,
  V. Shoup, 2008 (V2), Chapter 1-4, 11, 12

Available at **//shoup.net/ntb/ntb-v2.pdf**

The RSA trapdoor permutation

First published: Scientific American, Aug. 1977.

Very widely used:

– SSL/TLS: certificates and key-exchange

– Secure e-mail and file systems

… many others

Trapdoor functions (TDF)

Def: a trapdoor func. $X \longrightarrow Y$ is a triple of efficient algs. (G, F, F^{-1})

- G(): randomized alg. outputs a key pair (pk, sk)
- F(pk,·): det. alg. that defines a function $X \longrightarrow Y$
- F^{-1}(sk,·): defines a function $Y \longrightarrow X$ that inverts F(pk,·)

More precisely: ∀(pk, sk) output by G

$\forall x \in X$: $F^{-1}(sk, F(pk, x)) = x$

Public-key encryption from TDFs

- (G, F, F^{-1}): secure TDF $X \longrightarrow Y$
- (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
- $H: X \longrightarrow K$ a hash function

We construct a pub-key enc. system (G, E, D):

Key generation G: same as G for TDF

Public-key encryption from TDFs

- (G, F, F^{-1}): secure TDF $X \longrightarrow Y$
- (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
- $H: X \longrightarrow K$ a hash function

E(pk, m) :

$x \xleftarrow{R} X, \quad y \leftarrow F(pk, x)$

$k \leftarrow H(x), \quad c \leftarrow E_s(k, m)$

output (y, c)

D(sk, (y,c)) :

$x \leftarrow F^{-1}(sk, y),$

$k \leftarrow H(x), \quad m \leftarrow D_s(k, c)$

output m

In pictures:

$$\underbrace{F(pk, x)}_{\text{header}} \; \underbrace{E_s(\,H(x),\ m\,)}_{\text{body}}$$

Security Theorem:

If (G, F, F^{-1}) is a secure TDF, (E_s, D_s) provides auth. enc.
and $H: X \longrightarrow K$ is a "random oracle"
then (G,E,D) is CCA^{ro} secure.

Textbook RSA is insecure

Textbook RSA encryption:

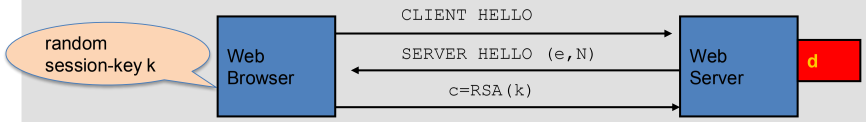
– public key: **(N,e)** Encrypt: $c \leftarrow m^e$ (in Z_N)
– secret key: **(N,d)** Decrypt: $c^d \rightarrow m$

Insecure cryptosystem !!

– Is not semantically secure and many attacks exist

⇒ The RSA trapdoor permutation is not an encryption scheme !

A simple attack on textbook RSA



Suppose k is 64 bits: $k \in \{0,\ldots,2^{64}\}$. Eve sees: $c = k^e$ in Z_N

If $\mathbf{k = k_1 \cdot k_2}$ where $k_1, k_2 < 2^{34}$ (prob. ≈20%) then $\mathbf{c/k_1^e = k_2^e}$ in Z_N

Step 1: build table: $c/1^e, c/2^e, c/3^e, \ldots, c/2^{34e}$. time: 2^{34}

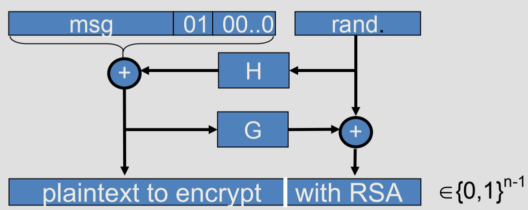
Step 2: for $k_2 = 0,\ldots, 2^{34}$ test if k_2^e is in table. time: 2^{34}

Output matching (k_1, k_2). Total attack time: $\approx 2^{40} << 2^{64}$

PKCS1 v2.0: OAEP

New preprocessing function: OAEP [BR94]

check pad
on decryption.
reject CT if invalid.



Thm [FOPS'01] : RSA is a trap-door permutation ⇒
RSA-OAEP is CCA secure when H,G are *random oracles*

in practice: use SHA-256 for H and G

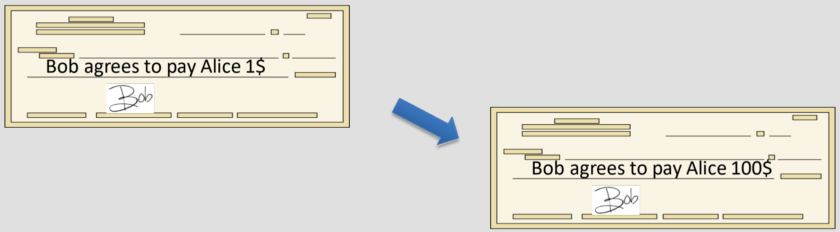
Summary

- Need for key exchange
- Key exchange using trusted third party (TTP)
- Key exchange without TTP
- Diffie-Hellman protocol
  - MiTM attack
  - Non-interactive property
  - Open problem of multiparty key establishment
- Public key encryption made possible by one-way functions with special properties.
- RSA encryption, ISO standard, PKCS, OAEP

What is digital signature?

Physical signatures

Goal: bind document to author

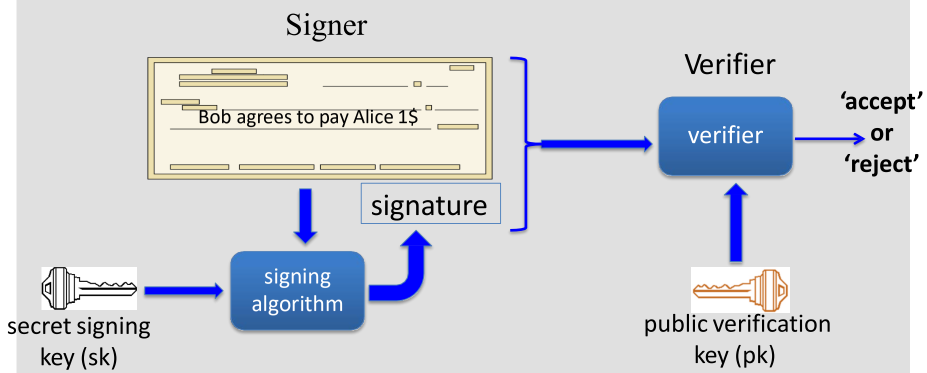


Problem in the digital world:

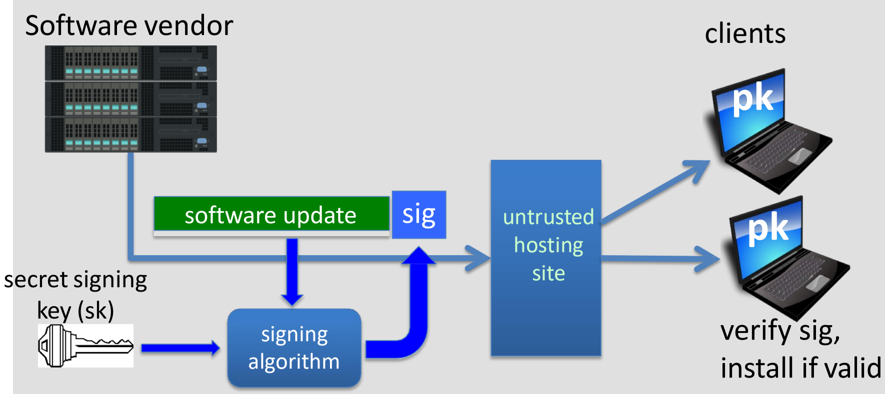
anyone can copy Bob's signature from one doc to another

Digital signatures

Solution: make signature depend on document



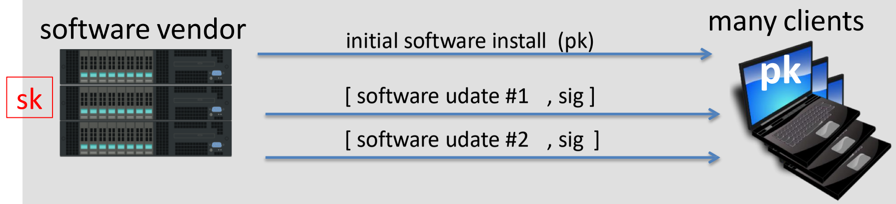
A more realistic example



Applications

Code signing:

- Software vendor signs code
- Clients have vendor's pk. Install software if signature verifies.



Review: three approaches to data integrity

1. **Collision resistant hashing**: need a read-only public space

Software Vendor

Small read-only public space

2. **MACs**: vendor must compute a new MAC of software for every client

- and must manage a long-term secret key (to generate a per-client MAC key)

3. **Digital signatures**: vendor must manage a long-term secret key

- Vendor's signature on software is shipped with software
- Software can be downloaded from an untrusted distribution site

When to use signatures

Generally speaking:

- If one party signs and **one** party verifies: **use a MAC**
  - Often requires interaction to generate a shared key
  - Recipient can modify the data and re-sign it before passing the data to a 3rd party
- If one party signs and **many** parties verify: **use a signature**
  - Recipients **cannot** modify received data before passing data to a 3rd party (non-repudiation)

Aggregate Signatures

[BGLS'03]

Certificate chain with aggregates sigs:

subj-id: Equifax	subj-id: GeoTrust	subj-id: Internal	subj-id: www.xyz.com
CA pub-key:	CA pub-key:	CA pub-key:	pub-key: ....
			aggregate-sig

Aggregate sigs: let us compress n signatures into one

$pk_1, m_1 \longrightarrow \sigma_1$

⋮

$pk_n, m_n \longrightarrow \sigma_n$

aggregate $\longrightarrow \sigma^*$

$V_{agg}(\overline{pk}, \overline{m}, \sigma^*) =$ "accept" means for i=1,…,n: user i signed msg m_i

Dan Boneh

Further Reading

- PSS. The exact security of digital signatures: how to sign with RSA and Rabin, M. Bellare, P. Rogaway, 1996.
- On the exact security of full domain hash, J-S Coron, 2000.
- Short signatures without random oracles, D. Boneh and X. Boyen, 2004.
- Secure hash-and-sign signatures without the random oracle, R. Gennaro, S. Halevi, T. Rabin, 1999.
- A survey of two signature aggregation techniques, D. Boneh, C. Gentry, B. Lynn, and H. Shacham, 2003.