CS 88: Security and Privacy

17: Asymmetric Key Cryptography and PKI

03-28-2024

slides adapted from Dave Levine
BLACKBOX #3:
HASH FUNCTIONS
Authenticated Encryption: Secrecy + Integrity

We have seen how we can achieve two independent goals: encryption and authentication. How about putting them together?

\[
\begin{align*}
Enc(k_1, m) &= c \\
Mac(k_2, m) &= t \\
Dec(k_1, c) &= m \\
Verify(k_2, m, t) &= 1
\end{align*}
\]

Encrypt and Authenticate: Is it secure?

A. Yes, encryption is randomized with proper K, IV  
B. No the tag might leak information  
C. No the MAC is deterministic
Encrypt then authenticate

We have seen how we can achieve two independent goals: encryption and authentication. How about putting them together?

\[ \text{Encrypt then Authenticate: Is it secure?} \]

A. Yes, encryption is randomized with proper K, IV
B. No the tag might leak information
C. No the MAC is deterministic
Secure Sessions: Consider parties who wish to communicate securely over the course of a session using authenticated encryption. Are they immune to the following attacks?

- Securely = secrecy and integrity
- Session = period of time over which parties are willing to maintain state.

A. Yes  
B. No
Secure Sessions: Consider parties who wish to communicate securely over the course of a session using authenticated encryption. Are they immune to the following attacks?

- Securely = secrecy and integrity
- Session = period of time over which parties are willing to maintain state.
Symmetric Key Cryptography

**CONFIDENTIALITY**
Block ciphers
- Deterministic ⇒ use IVs
- Fixed block size ⇒ use encryption “modes”

**INTEGRITY**
Message Authentication Codes (MACs)
- Send (message, tag) pairs
- Verify that they match
Symmetric Key Cryptography

**CONFIDENTIALITY**

Block ciphers

- Deterministic $\implies$ use IVs
- Fixed block size $\implies$ use encryption “modes”

**INTEGRITY**

Message Authentication Codes (MACs)

- Send (message, tag) pairs
- Verify that they match

---

Next

How do we establish $K$?

How do we know with whom we are communicating?
Shortcomings of symmetric key

Establishing a pairwise key requires a **key exchange**, which requires both parties to be **online**

**Issue #1: Requires pairwise key exchanges**

*File downloads*

*One-to-many: $O(N)$ key exchanges*

*Email / chat*

*All-to-all: $O(N^2)$ key exchanges*
Shortcomings of symmetric key

Establishing a pairwise key requires a **key exchange**, which requires both parties to be **online**

**Issue #1: Requires pairwise key exchanges**

- **File downloads**
  - Blue user uploads a document, then goes offline (e.g., forever)

- One-to-many: **O(N) key exchanges**
  - Later, a yellow user wants to get a copy; how can it know the copy is really from the blue user?
BLACKBOX #4: DIFFIE HELLMAN KEY ESTABLISHMENT
Asymmetric/Public-key Cryptography

• Main insight: separate keys for different functions

• Keys come in pairs, and are related to each other by a specific algorithm.
  • Public key (PK): used to encrypt or verify signatures
  • Private key (SK): used to decrypt and sign

• Encryption and decryption are inverse operations

• Secrecy: ciphertext reveals nothing about the plaintext
  • computationally hard to decrypt in polynomial time without key
  • to an attacker, the shared key k established, is indistinguishable from a uniform key
Diffie-Helman Key Exchange

\[ x \mod N \]

g is a generator of mod N if
\[ \{1, 2, \ldots, N-1\} = \{g^0 \mod N, g^1 \mod N, \ldots, g^{N-2} \mod N\} \]

\[ \begin{align*}
N=5, \ g=3 \\
3^0 \mod 5 = 1 & \quad 3^1 \mod 5 = 3 & \quad 3^2 \mod 5 = 4 & \quad 3^3 \mod 5 = 2
\end{align*} \]

Given \( x \) and \( g \), it is efficient to compute
\[ g^x \mod N \]

Given \( g \) and \( g^x \), it is efficient to compute \( x \)
(simply take \( \log_g g^x \))

Given \( g \) and \( g^x \mod N \) it is infeasible to compute \( x \)

Discrete log problem
Public knowledge: $g$ and $N$
g^N \mod N
\quad g^a \mod N
\quad g^b \mod N
\quad g^{ab} \mod N
\[
\begin{align*}
& g \mod N \\
& g^a \mod N \\
& g^b \mod N \\
& g^{ab} \mod N
\end{align*}
\]

Given \( g \) and \( g^x \mod N \) it is infeasible to compute \( x \)

Discrete log problem

Note that just multiplying \( g^a \) and \( g^b \) won't suffice:
\[
g^a \mod N \times g^b \mod N = g^{a+b} \mod N
\]

Key property:
An eavesdropper cannot infer the shared secret \( (g^{ab}) \).

But what about active intermediaries?
The attacker can interpose between the two communicating parties and insert, delete, and modify messages.

The attacker can now eavesdrop on the conversation.

Key property: Diffie-Hellman is not resilient to a MITM attack
The attacker can interpose between the two communicating parties and insert, delete, and modify messages.

The attacker can now eavesdrop on the conversation. Key property: Diffie-Hellman is not resilient to a MITM attack

Fix: Need to authenticate messages
Computational complexity for integer problems

- Integer multiplication is efficient to compute
- There is no known polynomial-time algorithm for general purpose factoring.
- Efficient factoring algorithms for many types of integers. *Easy to find small factors of random integers.*
- Modular exponentiation is efficient to compute
- Modular inverses are efficient to compute
Textbook RSA Encryption

**Public Key pk**
- N = pq modulus
- e encryption exponent

**Secret key sk**
- p, q primes
- d decryption exponent
- d = $e^{-1} \mod (p-1)(q-1) = e^{-1} \mod \Phi(N)$

$c = \text{Enc}_{PK}(m) = m^e \mod N$

$d = \text{Dec}_{SK}(c) = c^d \mod N$
RSA Security

• Best algorithm to break RSA: Factor N and compute d
• Factoring is not efficient in general
• Current key size recommendations: N >= 2048 bits
• Do not implement this yourself. Factoring is hard only for some integers, and textbook RSA is insecure.
TO FIX THIS PROBLEM WE NEED . . .

BLACKBOX #5: PUBLIC KEY CRYPTOGRAPHY
Shortcomings of symmetric key

Issue #3: How do you know to whom you’re talking?

Diffie-Hellman is resilient to eavesdropping, but not tampering.
Trusted Third Party

A protocol that solves this with trust

Trent: A trusted third party

Alice ← Trent ← Bob
Trusted Third Party

A protocol that solves this with trust

Trent: A trusted third party

1. Everybody establishes a pairwise key with Trent

**Good:** $O(N)$ key exchanges
A protocol that solves this with *trust*

**Trent**: A *trusted* third party

\[ E(K_{AT}, \text{msg} \ || \ \text{to:Bob}) \]

1. Everybody establishes a pairwise key with Trent
   
   *Good: O(N) key exchanges*

2. Trent validates each user’s identity; includes in message
   
   *Good: Authenticated communication*
Trusted Third Party

A protocol that solves this with trust

Trent: A trusted third party

E(K_{AT}, \text{msg} \ || \ \text{to:Bob}) \\
E(K_{BT}, \text{msg} \ || \ \text{from:Alice}) \\

1. Everybody establishes a pairwise key with Trent
   Good: \textit{O(N)} key exchanges

2. Trent validates each user’s identity; includes in message
   Good: Authenticated communication

Bad: All messages get sent through Trent
What are we trusting Trent not to do?

Just as "secure" meant nothing without an attack model, "trusted" means nothing without a trust model.

(Oh wow, "msg"!)

E(K_{AT}, \text{msg} || \text{to:Bob})

\[ \rightarrow \]

E(K_{BT}, \text{msg} || \text{from:Alice})

1. Do not read messages
What are we trusting Trent not to do?

Just as “secure” meant nothing without an attack model, “trusted” means nothing without a trust model.

\[ E(K_{AT}, \text{msg} || \text{to:Bob}) \quad \rightarrow \quad K_{AT} \quad \rightarrow \quad E(K_{BT}, \text{msg'} || \text{from:Alice}) \]

1. Do not read messages
2. Do not alter messages
What are we trusting Trent not to do?

Just as “secure” meant nothing without an attack model, “trusted” means nothing without a trust model.

...nothing...

Bob

Alice

$K_{AT}$

$K_{BT}$

$E(K_{BT}, \text{msg'} \ || \ \text{from:Alice})$

1. Do not read messages
2. Do not alter messages
3. Do not forge messages
What are we trusting Trent not to do?

Just as “secure” meant nothing without an attack model, “trusted” means nothing without a trust model.

E($K_{AT}$, msg || to:Bob)

1. Do not read messages
2. Do not alter messages
3. Do not forge messages
4. Do not go offline
Public key encryption

A public key encryption scheme comprises three algorithms.

**Key generation** $G$

- $PK =$ public key
- $SK =$ secret key

**Encryption** $E(PK, m)$

- $c =$ cipher text

**Decryption** $D(SK, c)$

- $m =$ original msg

**Correctness**

$$D(SK, E(PK, m)) = m$$

**Security**

- $E(PK, m)$ should appear random
- (small change to $(PK,m)$ leads to large changes to $c$)

- $E()$ should approximate a one-way trapdoor function: cannot invert without access to $SK$
Protocols with public key encryption

Goal: deliver a confidential message

Symmetric key

Generate public/private key pair (PK, SK)

Announce PK publicly (on website, in newspaper, …)

Obtain PK

Send $c = E(PK, \text{msg})$

Decrypt $D(SK, c) = \text{msg}$

$O(N^2)$ key exchanges

$O(N)$ keys in total
Overcoming fixed message sizes

Encryption \( E(PK, \text{msg}) \)
- Inputs
  - **Public** key \( PK \)
  - Message \( \text{msg} \) of **fixed size**
- Outputs: a cipher text \( c \) same size as \( \text{msg} \)

Like block ciphers, but there are not “modes” of public key encryption

Public key operations are **sloooow**!

Symmetric key operations are fast
Issues with public-key encryption

• No perfectly secret public-key encryption
• No deterministic public-key encryption scheme can be CPA-secure!
• CPA-security implies security for encrypting multiple messages as in the private-key case
Hybrid encryption

1. Generate public/private key pair (PK, SK); publicize PK
2. Obtain PK
3. Generate symmetric key K
4. Compute $c_{msg} = e(K, \text{msg})$
5. Compute $c_K = E(PK, K)$
6. Send $c_K || c_{msg}$
7. Decrypt $D(SK, c_K) = K$
8. Decrypt $d(K, c_{msg}) = \text{msg}$

Decryption done in the obvious way
Hybrid encryption

Obtain PK
Generate *symmetric* key $K$
Compute $c_{msg} = e(K, msg)$
Compute $c_K = E(PK, K)$
Send $c_K || c_{msg}$

The easy key distribution of public key

The speed and arbitrary message length of symmetric key
Protocols with public key cryptography
Goal: determine from whom a message came

Symmetric key

File downloads

One-to-many:
O(N) key exchanges
Digital signatures

A digital signature scheme comprises two algorithms

**Signing function** $\text{Sgn}(SK, m)$
- Inputs
  - **Secret** key SK
  - Fixed-length message
- Outputs: a *signature* $s$

This is a *randomized* algorithm
(nondeterministic output)

SK a.k.a. “Signing key”

Only one person can sign with a given (PK,SK) pair

**Verification function** $\text{Vfy}(PK, m, s)$
- Inputs
  - **Public** key PK
  - Message and signature
- Outputs: Yes/No if valid $(m,s)$

Deterministic algorithm

Anyone with the PK can verify
Digital signatures
A digital signature scheme comprises two algorithms

\[
\text{Signing } \text{Sgn}(SK, m) \rightarrow \text{a signature } s
\]

\[
\text{Verification } \text{Vfy}(PK, m, s) \rightarrow \text{Yes/No if valid } (m, s)
\]

Correctness
\[\text{Vfy}(PK, m, \text{Sgn}(SK, m)) = \text{Yes}\]

Security
Same as with MACs: even after a chosen plaintext attack, the attacker cannot demonstrate an existential forgery
Protocols with digital signatures
Goal: determine from whom a message came

**Symmetric key**

*File downloads*

- Generate public/private key pair (PK,SK)
- Announce PK publicly (on website, in newspaper, …)
- Compute $\text{sig} = \text{Sgn}(\text{SK}, \text{msg})$
- Publish $\text{msg} \parallel \text{sig}$

**One-to-many:** $O(N)$ key exchanges

*can now go offline!*
Digital signature properties

**Authenticity**
Bob can prove that a message signed by Alice is truly from Alice (even without a *pairwise* key)

**Integrity**
Bob can prove that no one has tampered with a signed message

**Non-repudiation**
Once Alice signs a message, she cannot subsequently claim she did *not* sign that message
RECALL OUR PROBLEM WITH DIFFIE-HELLMAN

The two communicating parties thought, *but did not confirm*, that they were talking to one another.

Therefore, they were vulnerable to MITM attacks.

Certificates allow us to verify with whom we are communicating.

We will solve this by incorporating public key cryptography.
Back to authentication

Generate public/private key pair (PK, SK); publicize PK

How can we know it was really who posted PK?

E(K_{AT}, msg || to: Bob)

E(K_{BT}, msg || from: Alice)

Can we achieve authentication without Trent in the middle of every message?
Authentication with public keys

1. Trent’s public key is widely disseminated (pre-installed in browsers/operating systems).

2. Alice generates a public/private key pair and asks Trent to bind her \( PK_A \) to her identity.

3. Trent signs a message (with \( SK_T \)):
   
   "The owner of the secret key corresponding to \( PK_A \) is Alice"

   This message + sig = Certificate
Authentication with public keys

Trent (PKₜ, SKₜ)

1. Trent's public key is widely disseminated (pre-installed in browsers/operating systems)

Trent vets Alice

Alice (PKₐ, SKₐ)

2. Alice generates a public/private key pair and asks Trent to bind her PKₐ to her identity

Alice = PKₐ

Bob

3. Trent signs a message (with SKₜ):

“The owner of the secret key corresponding to PKₐ is Alice”

This message + sig = Certificate
Authentication with public keys

4. Alice makes her certificate publicly available (or Bob simply asks for it).

5. Bob verifies the certificate using $PK_T$.
   - If Bob trusts Trent, then Bob trusts that he properly vetted Alice, and thus that her public key is $PK_A$.

6. Bob (via hybrid encryption) sends a message to Alice using her public key $PK_A$. 
Authentication with public keys

Properties

Trent need be online only when giving out **certificates**, not any time users want to communicate with one another.

Alice and Bob can communicate in an authenticated manner without having to go through Trent.

Trent: $(PK_T, SK_T)$

Alice: $(PK_A, SK_A)$

Bob: $(PK_T)$

Alice = $PK_A$

Bob = $PK_A$
Authentication with public keys

Trent

(PK_T, SK_T)

Trent vets Alice

Alice

PK_T

(PK_A, SK_A)

Alice = PK_A

Bob

PK_T

Alice = PK_A

Trust assumptions from our symmetric key protocol:

1. Do not read messages
2. Do not alter messages
3. Do not forge messages
4. Do not go offline

Trust assumptions in this public key protocol:

1. Correctly vet users
   (Some more in practice….)
Certificate revocation

3. Trent signs a message (with $SK_T$):

“The owner of the secret key corresponding to $PK_A$ is Alice”

This message + sig = Certificate

Put another way:

“The only person who knows $SK_A$ is Alice”

What happens if Alice’s key gets compromised?
(Stolen, accidentally revealed, …)
Certificate revocation

Please revoke my certificate (ID #3912...)

Trent signs a message (with SK_T):

“Certificate ID #3912... is no longer valid, as of April 5, ...”
Certificate revocation

Please revoke my certificate (ID #3912...)

Trent signs a message (with $SK_T$):

"Certificate ID #3912... is no longer valid, as of April 5, ..."

This message + sig = revocation

Bob obtains revocation information
Obtaining revocation data

Certificate Revocation Lists (CRLs)

A (often large) signed list of revocations

“Certificate ID #3912… is no longer valid, as of April 5, …”

Browsers and OSes occasionally download CRLs

Disincentive: CRLs can be large, so it takes time & bandwidth

Result: delayed days/weeks/forever
Obtaining revocation data

Online Certificate Status Protocol (OCSP)

Browsers and OSes perform OCSP checks on-demand (when verifying the certificate)

Bob ➔ Is certificate ID #3912… still valid? ➔ Trent

“Certificate ID #3912… is still longer valid, as of April 5, …”

Disincentive: Still delays the initial validation of the certificate (can increase webpage load time)
Obtaining revocation data

OCSP Stapling

Websites issue OCSP requests, include responses in initial handshake

Is certificate ID #3912… still valid?

“Certificate ID #3912… is still longer valid, as of April 5, …”

Alice forwards this to Bob along with the certificate when they first start to communicate
Certificate revocation responsibilities

Alice’s responsibility:
Request revocations

Trent’s responsibility:
Make revocations publicly available

Bob’s responsibility:
Check for revocations
Certificates in the wild

The lock icon indicates that the browser was able to authenticate the other end, i.e., validate its certificate.
Certificate chain

**Subject** (who owns the public key)

**Common name:** the URL of the subject

**Issuer** (who verified the identity and signed this certificate)