EECS 388

Introduction to Computer Security

Lecture 6:

Key Exchange and Public-Key Cryptography

September 14, 2023
Prof. Halderman

Authenticated Encryption with Associated Data

Preferred approach to integrity+confidentiality:

Authenticated encryption with associated data (AEAD)

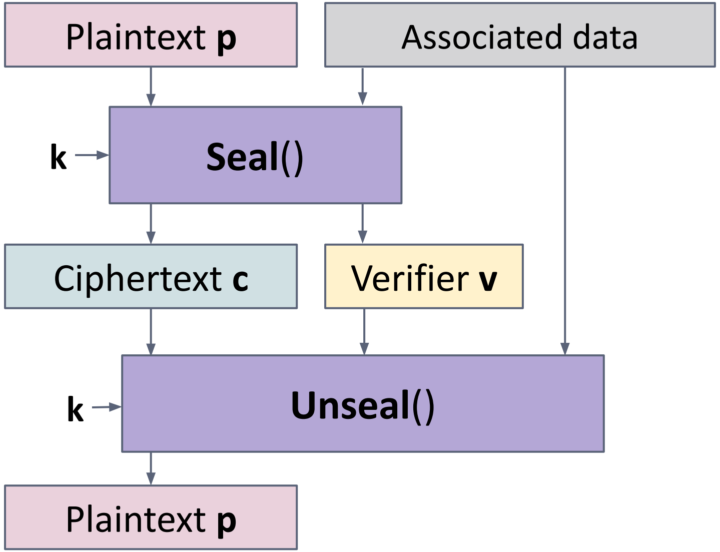
Integrity and encryption in a single primitive:

c, **v** := **Seal**(**k**, **p**, [**associated_data**])
encrypts plaintext **p** and returns
ciphertext **c** and a verifier **v** (called a "**tag**")

p, **err** := **Unseal**(**k**, **c**, **v**, [**associated_data**])
returns **p** or an error if **v** does not match the
supplied **c** and **associated_data**

Optional **associated_data** is covered by verifier but *not encrypted*.

Useful for binding data to its context:
e.g., counter, sender ID, etc.



Examples:

AES-GCM ("Galois Counter Mode")
hardware accelerated in recent CPUs

ChaCha20-Poly1305, common on mobile

Diffie-Hellman Key Exchange

Issue: How do we get a shared key?

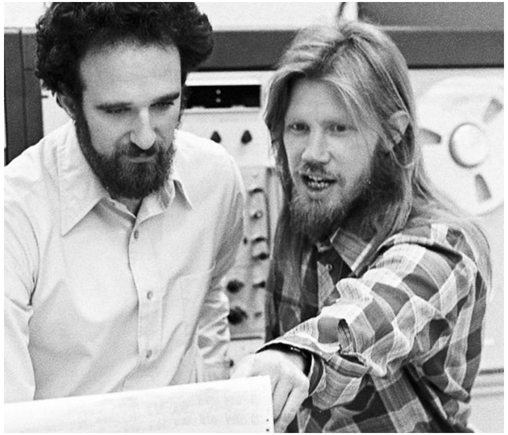
Amazing fact: Alice and Bob can have a public conversation to derive a secret shared key!

Diffie-Hellman (D-H) key exchange

The birth of modern cryptography

1976:
Whit Diffie and
Martin Hellman
"New Directions
in Cryptography"

(Earlier, in secret, by Malcolm Williamson at GCHQ)



Turing Award in 2015

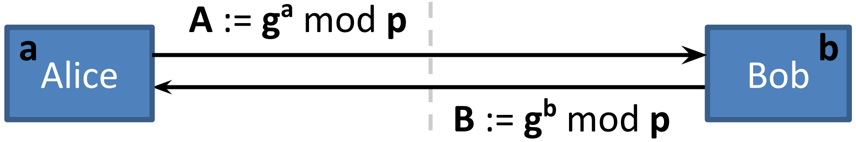
1. Use public (often from standards) parameters:
   - **p**: a large prime (say, 2048-bits)
   - **g**: a primitive root modulo **p** (usually small: 2, 3…)

2. **Alice**

   Generates random secret value **a** ($0<a<p$)

   Bob

   Generates random secret value **b** ($0<b<p$)



$\mathbf{A} := \mathbf{g}^a \bmod \mathbf{p}$

$\mathbf{B} := \mathbf{g}^b \bmod \mathbf{p}$

3. Computes

   $\mathbf{s} := \mathbf{B}^a \bmod \mathbf{p}$

   Computes

   $\mathbf{s} := \mathbf{A}^b \bmod \mathbf{p}$

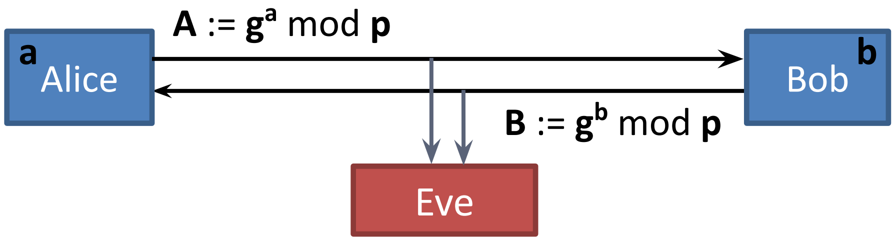
Alice and Bob each obtain the same value:

$$\mathbf{B}^a \bmod \mathbf{p} = \mathbf{g}^{ba} \bmod \mathbf{p} = \mathbf{g}^{ab} \bmod \mathbf{p} = \mathbf{A}^b \bmod \mathbf{p}$$

Today: Elliptic curves ***much*** more common!

Security of Diffie-Hellman

Passive eavesdropping: Fails



a Alice

$\mathbf{A} := \mathbf{g}^{a} \text{ mod } \mathbf{p}$

b Bob

$\mathbf{B} := \mathbf{g}^{b} \text{ mod } \mathbf{p}$

Eve

Eve knows: $\mathbf{p}$, $\mathbf{g}$, $\mathbf{g}^{a} \text{ mod } \mathbf{p}$, $\mathbf{g}^{b} \text{ mod } \mathbf{p}$

Eve wants to compute $\mathbf{s} := \mathbf{g}^{ab} \text{ mod } \mathbf{p}$

(Computational Diffie-Hellman problem)

Best known approach: Find **a** or **b**, then compute **s**

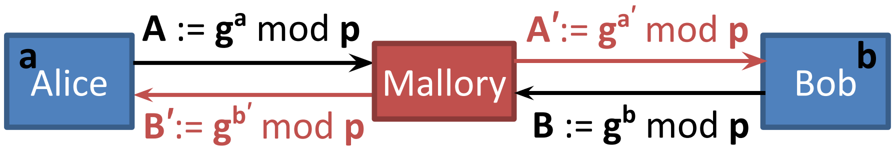
Finding **x** given $\mathbf{g}^{x} \text{ mod } \mathbf{p}$ is an instance of the **discrete logarithm problem**

Best known algorithm: **number field sieve**

superpolynomial but subexponential in |**p**|

Believed (hoped) intractable for 2048-bit **p**

Active MITM attack: Succeeds!



a Alice

$\mathbf{A} := \mathbf{g}^{a} \text{ mod } \mathbf{p}$

Mallory

$\mathbf{A}' := \mathbf{g}^{a'} \text{ mod } \mathbf{p}$

b Bob

$\mathbf{B}' := \mathbf{g}^{b'} \text{ mod } \mathbf{p}$

$\mathbf{B} := \mathbf{g}^{b} \text{ mod } \mathbf{p}$

Alice does D-H, *really with Mallory*, ends up with $\mathbf{u} := \mathbf{g}^{ab'} \text{ mod } \mathbf{p}$

Bob does D-H, *really with Mallory*, ends up with $\mathbf{v} := \mathbf{g}^{a'b} \text{ mod } \mathbf{p}$

Alice and Bob think they’re talking with the other, but really they’ve each computed a separate secret key **shared with mallory**

D-H gives you a shared secret,
but you don’t know who it’s shared with!

(We’ll fix that using digital signatures…)

Issues About Keys

Defending D-H from MITMs

Approaches:

1) **Trust-on-first-use**

Hope there's no attacker the first time, but detect if keys change later (e.g.: **SSH**)

2) **Have users communicate out-of-band**

Ask people to meet or call each other to verify the have the same **k** (e.g.: **Signal**)

3) **Rely on proximity to limit MITMs**

Attacker must be nearby, or in contact (e.g.: Many **IoT devices** attempt this)

4) **Use digital signatures**

What HTTPS websites do. Much more later…

Why use D-H at all if we have other methods (as we'll see) of sending a key confidentially?

Important goal: Forward Secrecy

We usually want to communicate **repeatedly**

Pitfall: Adversary could record all the ciphertexts, then later steal our key

Solution: For each session, use D-H to generate a temporary ***session key***. Use separate ***long-term key*** to authenticate it

Benefits: Compromising long-term key would allow future impersonation but not retrospective decryption (assuming attacker can't break D-H)

RSA: Public-key Cryptography

Since antiquity, encryption key = decryption key
"**symmetric key crypto**"

What if Alice publishes data to lots of people and wants them to verify integrity… [Publish a MAC key?]

What if Bob wants to receive data from lots of people, confidentially… [Why is this impractical?]

Breakthrough: Keys can be distinct.

Public-key crypto

1977: RSA Cryptosystem
Ron Rivest, Adi Shamir, Leonard Adleman
(Earlier, in secret, by Clifford Cocks at GCHQ)

Turing Award in 2002

"Textbook" RSA

Key generation (in secret):

1. Generate large random primes, **p** and **q** (say, 2048 bits each)
2. Compute "modulus" **N** := **pq** (say, 4096 bits)
3. Pick small "encryption exponent" **e** (say, 65537) Must be relatively prime to (**p** – 1)(**q** – 1)
4. Compute "decryption exponent" **d** such that **ed** ≡ 1 mod (**p** – 1)(**q** – 1)

Yields **RSA key pair**: **Public key**: (**e**, **N**)
Private key: (**d**, **N**)

Public-key encryption !!!

Encrypt: $\mathbf{c} := \mathbf{m}^{\mathbf{e}} \bmod \mathbf{N}$

Decrypt: $\mathbf{m} := \mathbf{c}^{\mathbf{d}} \bmod \mathbf{N}$

Digital signatures !!!

Sign: $\mathbf{s} := \mathbf{m}^{\mathbf{d}} \bmod \mathbf{N}$

Verify: $\mathbf{m} \overset{?}{=} \mathbf{s}^{\mathbf{e}} \bmod \mathbf{N}$

Facts about RSA

Subtle fact: RSA can be used for confidentiality, integrity, and/or authenticity

Confidentiality: Public-key encryption

Alice encrypts **m** using Bob's public key to get **c**

Bob decrypts **c** using Bob's private key to get **m**

Integrity/sender authenticity: Digital signatures

Alice signs **m** using Alice's private key to get **s**

Bob verifies **m**,**s** using Alice's public key

Both properties: Use both, with two key pairs

Alice encrypts **m** using Bob's public key to get **c**, then Alice signs **c** using Alice's private key to get **s**

Bob verifies **c**,**s** using Alice's public key, then Bob decrypts **c** using Bob's private key to get **m**

Is RSA secure?

Best known way to compute **d** from **e** is factoring **N** into **p** and **q**.

Best known* algorithm: number field sieve (again)

superpolynomial but subexponential in |**N**|

Believed (hoped) intractable for 4096-bit **N**

But: Range of subtle mathematical attacks

RSA drawbacks: >1000X slower than AES

Messages must be shorter than **N**

Keys must be relatively large [Why?]

Elliptic-curve cryptography (ECC)

Elliptic curves are a mathematical alternative for constructing key exchange, signing, encryption

Pros: Attacks (hoped) harder, so key can be shorter

Examples: **ECDH, ECDSA, Ed25519**

Subtle Attacks on Textbook RSA

Textbook RSA Encryption

Encrypt: $\mathbf{c} := \mathbf{m}^{\mathbf{e}} \bmod \mathbf{N}$ Decrypt: $\mathbf{m} := \mathbf{c}^{\mathbf{d}} \bmod \mathbf{N}$

Small *e* attack:

If $\mathbf{e} = 3$ and $\mathbf{m} < \mathbf{N}^{1/3}$: $\mathbf{m} = \mathbf{c}^{1/3}$ (No mod! Cube root's fast)

Stereotyped message attacks:

We can efficiently compute up to a 1/**e**-fraction of the bits of an RSA-encrypted message with public exponent **e** if we know the rest of the plaintext.

Ciphertext malleability:

RSA is homomorphic under multiplication.

From $\mathbf{c} = \mathbf{m}^{\mathbf{e}} \bmod \mathbf{N}$, attacker can forge encryption of $\mathbf{xm}$: $\mathbf{c'} := \mathbf{x}^{\mathbf{e}}\mathbf{c} \bmod \mathbf{N} = \mathbf{x}^{\mathbf{e}}\mathbf{m}^{\mathbf{e}} \bmod \mathbf{N} = (\mathbf{xm})^{\mathbf{e}} \bmod \mathbf{N}$

Key generation failures:

Suppose Alice and Bob use a bad RNG, generate moduli *with the same* ***p***: $\mathbf{N}_1 = \mathbf{pq}_1$, $\mathbf{N}_2 = \mathbf{pq}_2$.

Eve easily learns both private keys: $\mathbf{p} := \mathrm{GCD}(\mathbf{N}_1,\mathbf{N}_2)$

Textbook RSA Signatures

Sign: $\mathbf{s} := \mathbf{m}^{\mathbf{d}} \bmod \mathbf{N}$ Verify: $\mathbf{m} \overset{?}{=} \mathbf{s}^{\mathbf{e}} \bmod \mathbf{N}$

Forging signatures on random messages:

When there are no constraints on messages, can trivially forge *some* messages with valid signatures.

Attacker picks random **s**, does $\mathbf{m} := \mathbf{s}^{\mathbf{e}} \bmod \mathbf{N}$.

Producing an arbitrary message with a valid signature doesn't prove sender knows private key!

Forging signatures on specific messages:

Suppose Bob will sign any message Mallory provides that he deems *non-sensitive*.

Mallory can trick Bob into signing *sensitive* **m'**:

Mallory computes: $\mathbf{m} := \mathbf{r}^{\mathbf{e}}\mathbf{m'}$ for random **r**

Bob signs **m**: $\mathbf{s} := \mathbf{m}^{\mathbf{d}} \bmod \mathbf{N} = (\mathbf{r}^{\mathbf{e}}\mathbf{m'})^{\mathbf{d}} \bmod \mathbf{N}$

Then, Mallory finds signature $\mathbf{s'} = (\mathbf{m'})^{\mathbf{d}} \bmod \mathbf{N}$ by computing $\mathbf{s}(\mathbf{r}^{-1}) \bmod \mathbf{N} = (\mathbf{r}^{\mathbf{e}}\mathbf{m'})^{\mathbf{d}}(\mathbf{r}^{-1}) \bmod \mathbf{N}$.

RSA in Practice

Textbook RSA is dangerously insecure, but we can use it as a basis for better constructions:

Safely encrypting with RSA

Encrypt a message $\mathbf{m}$ to RSA public key ($\mathbf{e}$,$\mathbf{N}$):

Use RSA to encrypt a random $\mathbf{x} < \mathbf{N}$, use a KDF to derive a key from $\mathbf{x}$, then encrypt message using a symmetric cipher and key $\mathbf{k}$.

1. Generate random $\mathbf{x} < \mathbf{N}$
2. $\mathbf{c}_1 := \mathbf{x}^{\mathbf{e}} \bmod \mathbf{N}$
3. $\mathbf{k} := \mathrm{KDF}(\mathbf{x})$
4. $\mathbf{c}_2 := \text{AES-GCM}_{\mathbf{k}}(\mathbf{m})$

This is called "hybrid" encryption. Advantages:

- Identical messages yield different ciphertexts
- Don't have to worry about RSA padding
- Don't have to worry about message length

Safely signing with RSA

Sign a message $\mathbf{m}$ with RSA private key ($\mathbf{d}$,$\mathbf{N}$):

Use RSA to sign a *carefully padded* version of the digest of $\mathbf{v}$. Many gotchas! A good padding scheme is "Probabilistic Signature Scheme" (PSS).

1. $\mathbf{v} := \text{SHA-256}(\mathbf{m})$
2. $\mathbf{x} := \mathrm{PSS}(\mathbf{v})$
3. $\mathbf{s} := \mathbf{x}^{\mathbf{d}} \bmod \mathbf{N}$

Verifier checks padding:

1. $\mathbf{v'} := \text{SHA-256}(\mathbf{m'})$
2. $\mathbf{x'} := \mathbf{s'}^{\mathbf{e}} \bmod \mathbf{N}$
3. Verify that $\mathbf{x'}$ is correct PSS-padded $\mathbf{v'}$

Caution If you don't correctly verify the padding, attacker can forge signatures w/ **Bleichenbacher attack**

You'll exploit in Project 1!

In practice, almost always should use **crypto libraries** to get such details right…

A Secure Channel Protocol

Putting it all together…

Everybody knows
Alice's public key (e_a,N_a)
Bob's public key (e_b,N_b)

Only Alice knows
Alice's private key (d_a,N_a)

Only Bob knows
Bob's private key (d_b,N_b)

1. Use **Diffie-Hellman** to generate a shared secret

Alice: Generates random a
$A := g^a \bmod p$ →
Bob: Generates random b
← $B := g^b \bmod p$
Alice: $k := B^a \bmod p$
Bob: $k := A^b \bmod p$

2. Use **RSA signatures** to confirm we're really talking to each other (*sender authenticity*)

Alice: x := PSS(SHA-256(A,B))
$s_a := x^{d_a} \bmod N_a$ →
Bob: x := PSS(SHA-256(A,B))
← $s_b := x^{d_b} \bmod N_b$
Alice: Verifies that $(s_b)^{e_b} \bmod N_b$ is correct PSS-padded x
Bob: Verifies that $(s_a)^{e_a} \bmod N_a$ is correct PSS-padded x

3. Derive symmetric keys for each direction using a **PRF**

Alice:
k_a := HMAC-SHA256$_k$("→")
k_b := HMAC-SHA256$_k$("←")

Bob:
k_a := HMAC-SHA256$_k$("→")
k_b := HMAC-SHA256$_k$("←")

4. Use **AEAD** (e.g., **AES-GCM**) or **encrypt-then-MAC** messages (*integrity* and *confidentiality*)

Alice: c_a,v_a := Seal(k_a,"Hi Bob!")
c_a,v_a →
Bob: m_a := Unseal(k_a,c_a,v_a)
Bob: c_b,v_b := Seal(k_b,"Hi Alice!")
← c_b,v_b
Alice: m_b := Unseal(k_b,c_b,v_b)

* Please don't actually code anything like this. Just use TLS!

Coming Up

Reminders:

Project 1, Part 1 due TODAY at 6 p.m.

Project 1, Part 2 due next Thursday at 6 p.m.

Quiz on Canvas after every lecture

Tuesday

The Web Platform

Intro to the Web platform

HTTP, cookies, Javascript, etc.

Thursday

Web Attacks and Defenses

XSS, CSRF, and SQLi

attacks and defenses