Chapter 2 – Meet Me in Public, Share a Secret

“We’ve never met — how do we agree on that key?”

ndhy

July 2, 2025

Public place, private handshake.

Outline

1. Why public keys exist — recap the symmetric-key pain
2. Diffie–Hellman: mathematics of a shared secret
3. One-shot asymmetric encryption (ECIES)
4. Digital signatures for authentication (ECDSA)
5. Putting it all together: ECDH-derived channel, ECIES parcels, ECDSA IDs
6. Hands-on exercise: choose ECDH or ECIES and test a round-trip

Encryption without key exchange = frustration

- Symmetric key must be shared *out of band*.
- Phone, SMS, e-mail → eavesdroppable.
- Need: agree on secrets over an open channel.

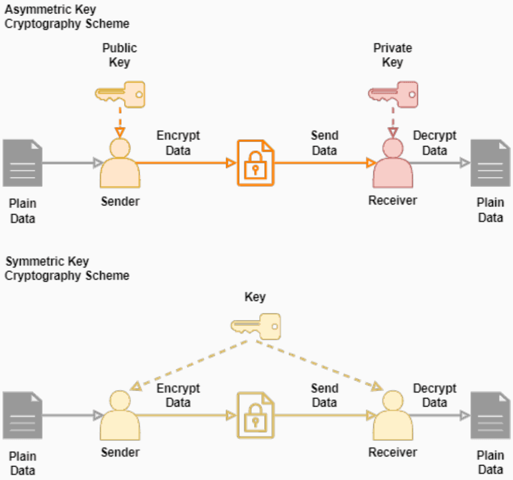


Figure 1: Symmetric vs Asymmetric

Diffie–Hellman key exchange in one slide

$$\text{Public params: } G = \langle g \rangle,\ |G| = q$$

$$\underbrace{a \leftarrow \mathbb{Z}_q}_{\text{Alice}} \quad \underbrace{b \leftarrow \mathbb{Z}_q}_{\text{Bob}}$$

$$A = g^a,\ B = g^b \quad \text{(both broadcast)}$$

$$K_{\text{shared}} = g^{ab} = B^a = A^b$$

Diffie–Hellman key exchange in one slide

$$\text{Public params: } G = \langle g \rangle,\ |G| = q$$

$$\underbrace{a \leftarrow \mathbb{Z}_q}_{\text{Alice}} \quad \underbrace{b \leftarrow \mathbb{Z}_q}_{\text{Bob}}$$

$$A = g^a,\ B = g^b \quad \text{(both broadcast)}$$

$$K_{\text{shared}} = g^{ab} = B^a = A^b$$

- Eavesdropper hears A, B but not a, b.
- Discrete-log hard ($\approx 2^{128}$ work on secp256r1).
- Output → AES-GCM for bulk traffic.

ECDH demo (secp256k1)

```
# Generate keys
openssl ecparam -name secp256k1 -genkey -out alice.pem
openssl ec -in alice.pem -pubout -out alice.pub
openssl ecparam -name secp256k1 -genkey -out bob.pem
openssl ec -in bob.pem -pubout -out bob.pub

# Derive shared secret
openssl pkeyutl -derive -inkey alice.pem -peerkey bob.pub -out alice.ss
openssl pkeyutl -derive -inkey bob.pem -peerkey alice.pub -out bob.ss
sha256sum alice.ss bob.ss # hashes match  same key
```

ECIES: one-shot asymmetric encryption

- Encrypt small blob (e.g. 128-byte receipt) to Bob's pub-key.
- Internally: fresh ECDH $+$ KDF $+$ AES/ChaCha.
- Slower than AES, so we wrap symmetric keys for bulk.

ECIES: one-shot asymmetric encryption

- Encrypt small blob (e.g. 128-byte receipt) to Bob's pub-key.
- Internally: fresh ECDH $+$ KDF $+$ AES/ChaCha.
- Slower than AES, so we wrap symmetric keys for bulk.

Speed intuition
ECDH $\approx \$\,\mu$s AES block $\approx$ ns.

ECIES demo (OpenSSL pkeyutl)

```
# Encrypt 128-byte file to Bob
openssl rand 128 > receipt.bin
openssl pkeyutl -encrypt -pubin -inkey bob.pub \
  -pkeyopt ecies -in receipt.bin -out receipt.enc

# Bob decrypts with his private key
openssl pkeyutl -decrypt -inkey bob.pem \
  -pkeyopt ecies -in receipt.enc -out receipt.dec
diff receipt.bin receipt.dec # no output  success
```

ECDSA: I am who I claim

- Private key → sign message hash.
- Anyone with public key verifies.
- Non-repudiation & tamper evidence.

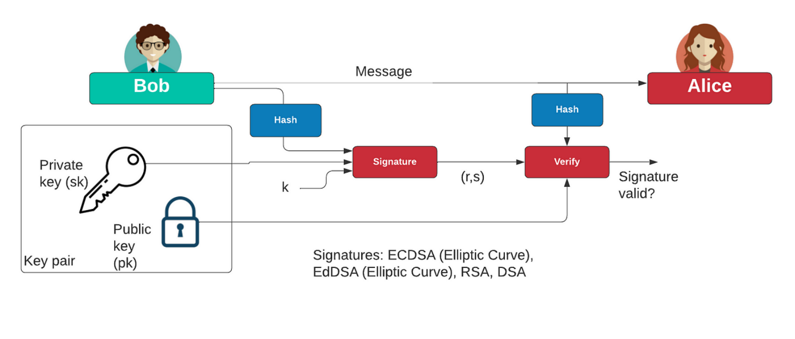
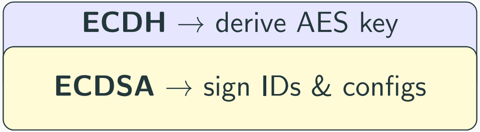


Figure 2: Digital Signatures Scheme

ECDSA in practice (SHA-256, secp256k1)

```
# Sign
openssl dgst -sha256 -sign alice.pem -out msg.sig message.txt
# Verify
openssl dgst -sha256 -verify alice.pub -signature msg.sig message.txt
```

The modern crypto stack



ECDH → derive AES key

ECDSA → sign IDs & configs

Your turn — pick a path

Option A • ECDH → AES channel

1. Derive shared secret with `pkeyutl -derive`.
2. Hash secret to 32 bytes, use as AES-256-GCM key.
3. Send 1-KB file, decrypt, verify SHA-256.

Your turn — pick a path

Option A • ECDH → AES channel

1. Derive shared secret with `pkeyutl -derive`.
2. Hash secret to 32 bytes, use as AES-256-GCM key.
3. Send 1-KB file, decrypt, verify SHA-256.

Option B • ECIES receipt

1. Generate 128-byte random note.
2. Encrypt with partner's pub-key, hand over file.
3. Partner decrypts; hashes must match.

Cheat-Sheet

- ECDH: fastest path to a shared symmetric key.
- ECIES: one-shot encrypt small blobs to a pub-key.
- ECDSA: authenticate origin integrity.
- Stack: **ECDH** → **AES**, **ECIES** for extras, **ECDSA** for trust.

Questions?