Cryptography

Elliptic Curve Cryptography and TLS

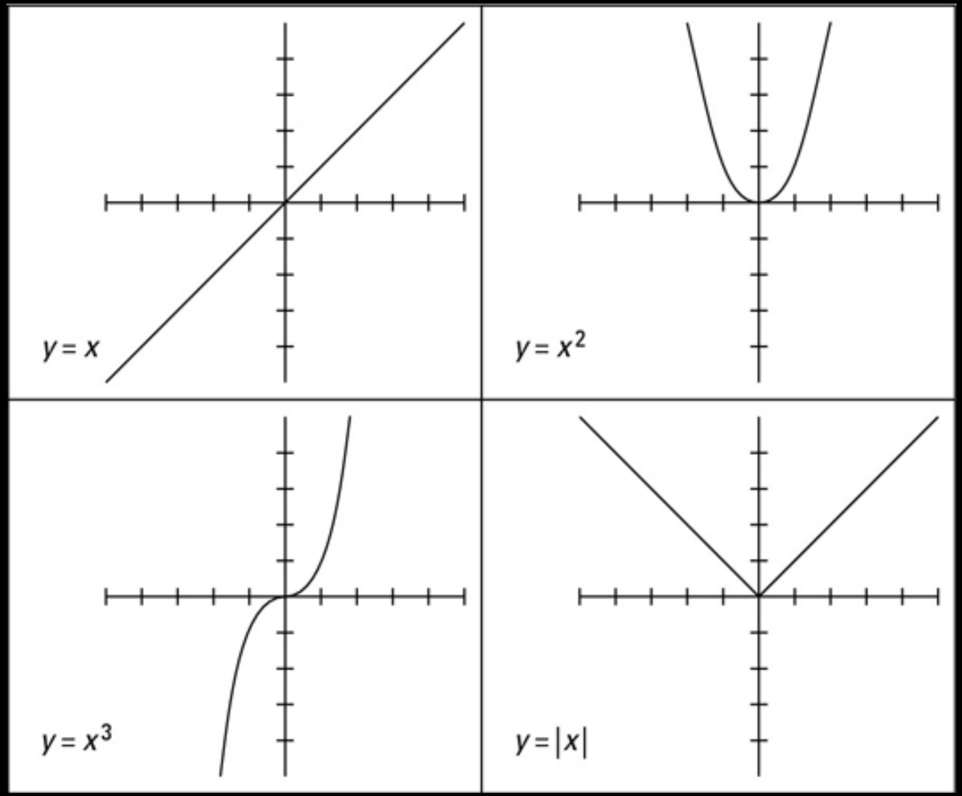
Content

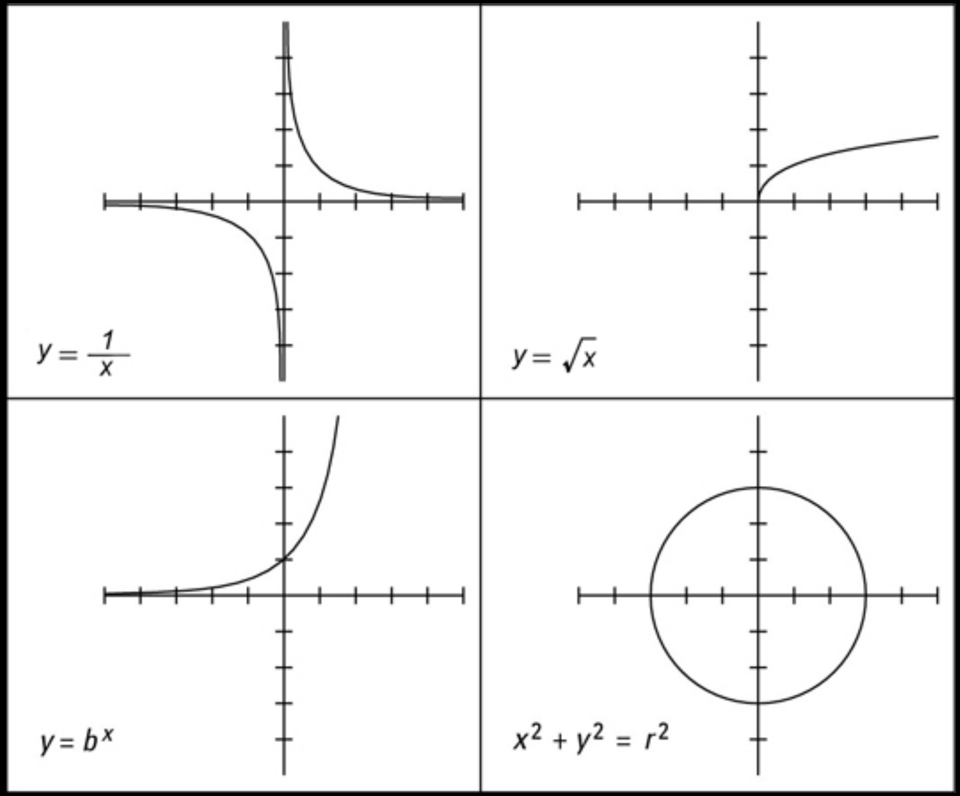
What is an Elliptic Curve?

- Elliptic curve cryptography (ECC) is more efficient than RSA and ElGamal.
- ECC with a 256-bit key is stronger than RSA with a 4096-bit key.
- ECC is more complex.
- There are many elliptic curves.
  - simple and sophisticated, efficient and inefficient, and secure and insecure.
- ECC is used in:
  - Apple’s iMessage application (https://security.apple.com/blog/imessage-pq3/)
  - Bitcoin cryptocurrency
- ECC do **encryption**, **signature**, and **key agreement** faster than their classical counterparts.

What is an Elliptic Curve?

- Curve: a group of points with x and y coordinates.
- A curve's equation defines all the points that belong to that curve.





What is an Elliptic Curve?

- A cryptographic elliptic curve - Weierstrass form:

$$y^3 = x^3 + ax + b$$

This is a cubic equation, a and b are constants define the shape of the curve

What is an Elliptic Curve?

- Example: the elliptic curve $y^2 = x^3 - 4x$

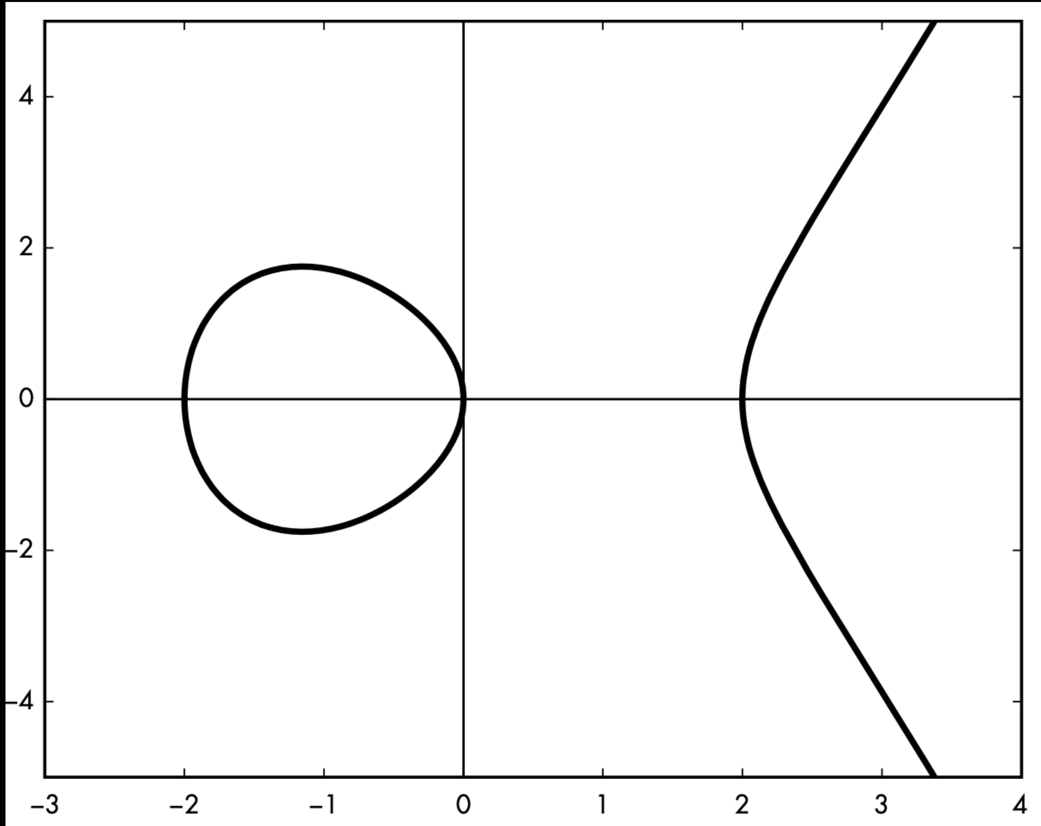


Figure 12-1: An elliptic curve with the equation $y^2 = x^3 - 4x$, shown over the real numbers

What is an Elliptic Curve?

- Note that points in the interval (0, 2) are not defined.
- Substitute for $x = 1$ in $y^2 = x^3 - 4x$,you get $y^2 = -3$; no natural number solution.
- Important to distinguish between points **on the curve** from **points off the curve**:
  - ECC works with points on the curve
  - Points off the curve often present a security risk.
- Note that for $x = -1$, there are two solutions:
  - $y = \sqrt{3}$ and $y = -\sqrt{3}$

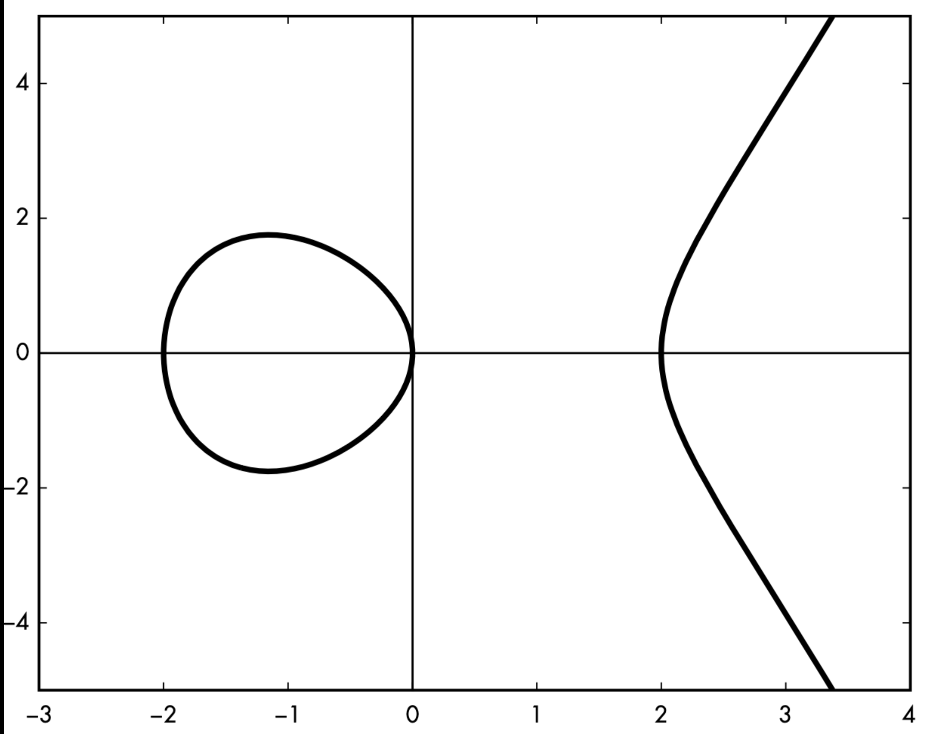


Figure 12-1: An elliptic curve with the equation $y^2 = x^3 - 4x$, shown over the real numbers

What is an Elliptic Curve?

- Cryptography deals with elliptic curves over integers.
- This is the same previous curve, $y^2 = x^3 - 4x \;\; mod \; 191$

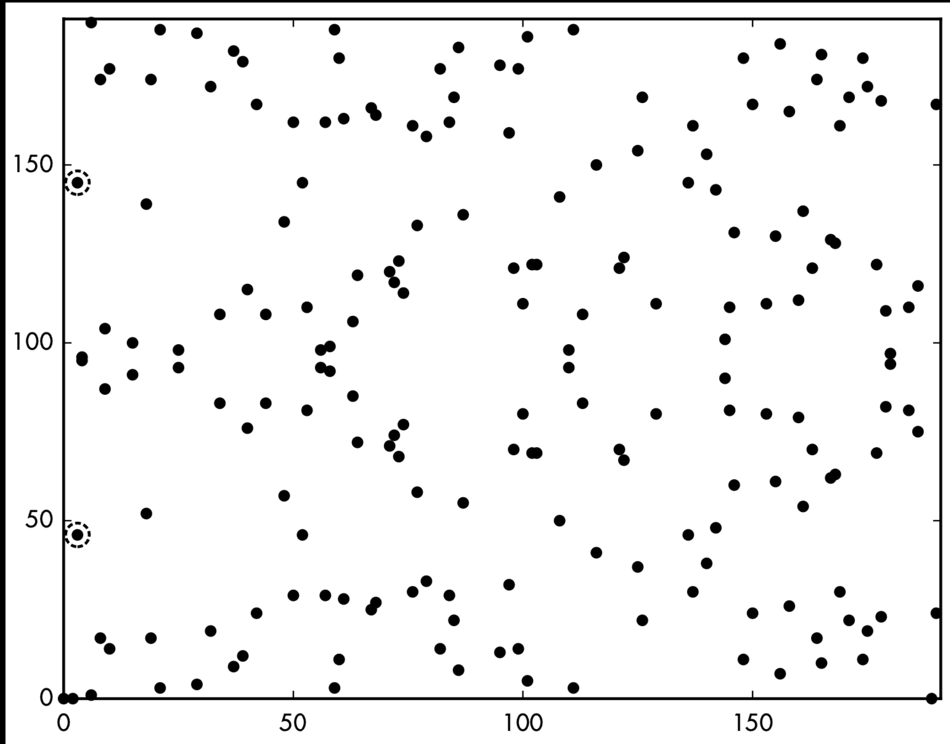


Figure 12-2: The elliptic curve with the equation $y^2 = x^3 - 4x$ over $\mathbf{Z}_{191}$, the set of integers modulo 191

What is an Elliptic Curve?

- Example: if $x = 3$ on $y^2 = x^3 - 4x$, then $y^2 = 15$, which has two solutions:
    - $y = 46 \quad | \; 46^2 \; mod \; 191 = 15$
    - $y = 145 \quad | \; 145^2 \; mod \; 191 = 15$

- Thus, the points $(3, 46)$ and $(3, 145)$ belong to the curve

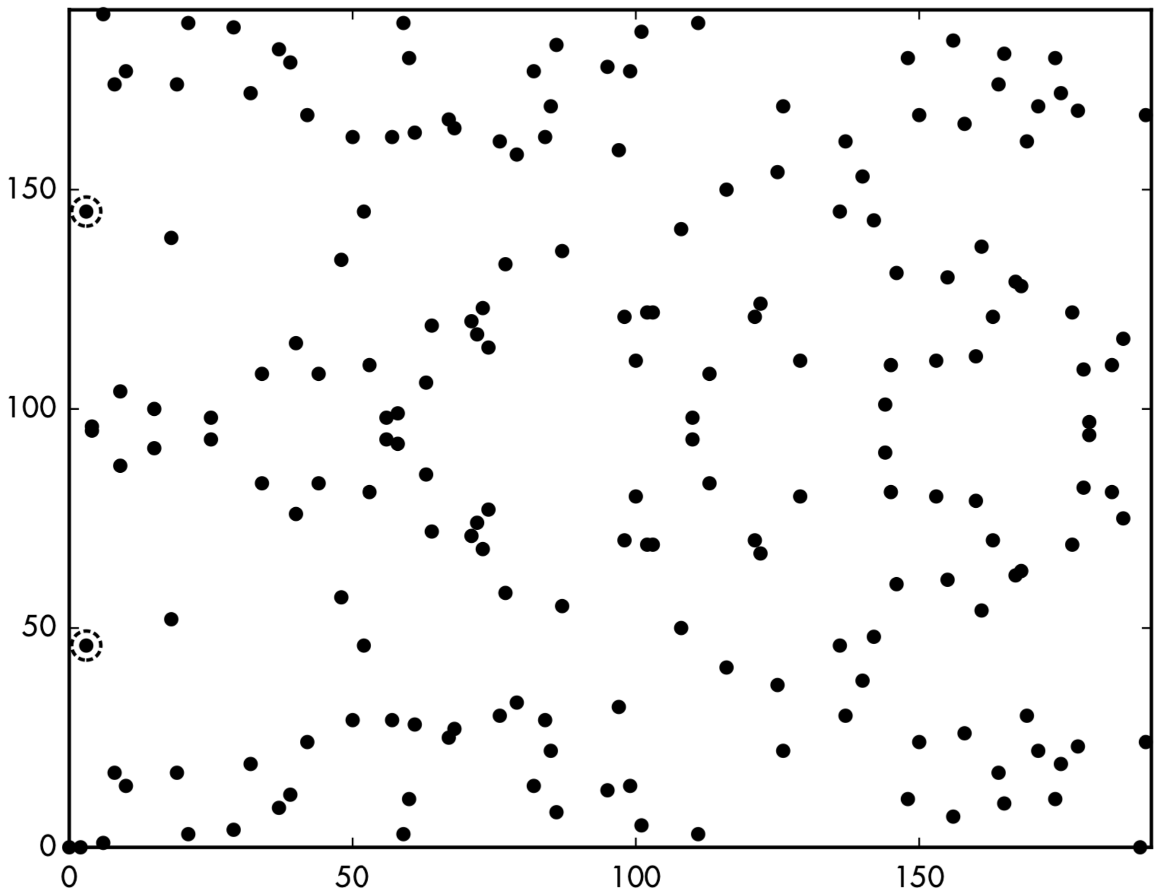


Figure 12-2: The elliptic curve with the equation $y^2 = x^3 - 4x$ over $\mathbf{Z}_{191}$, the set of integers modulo 191

Content

Adding and Multiplying Points

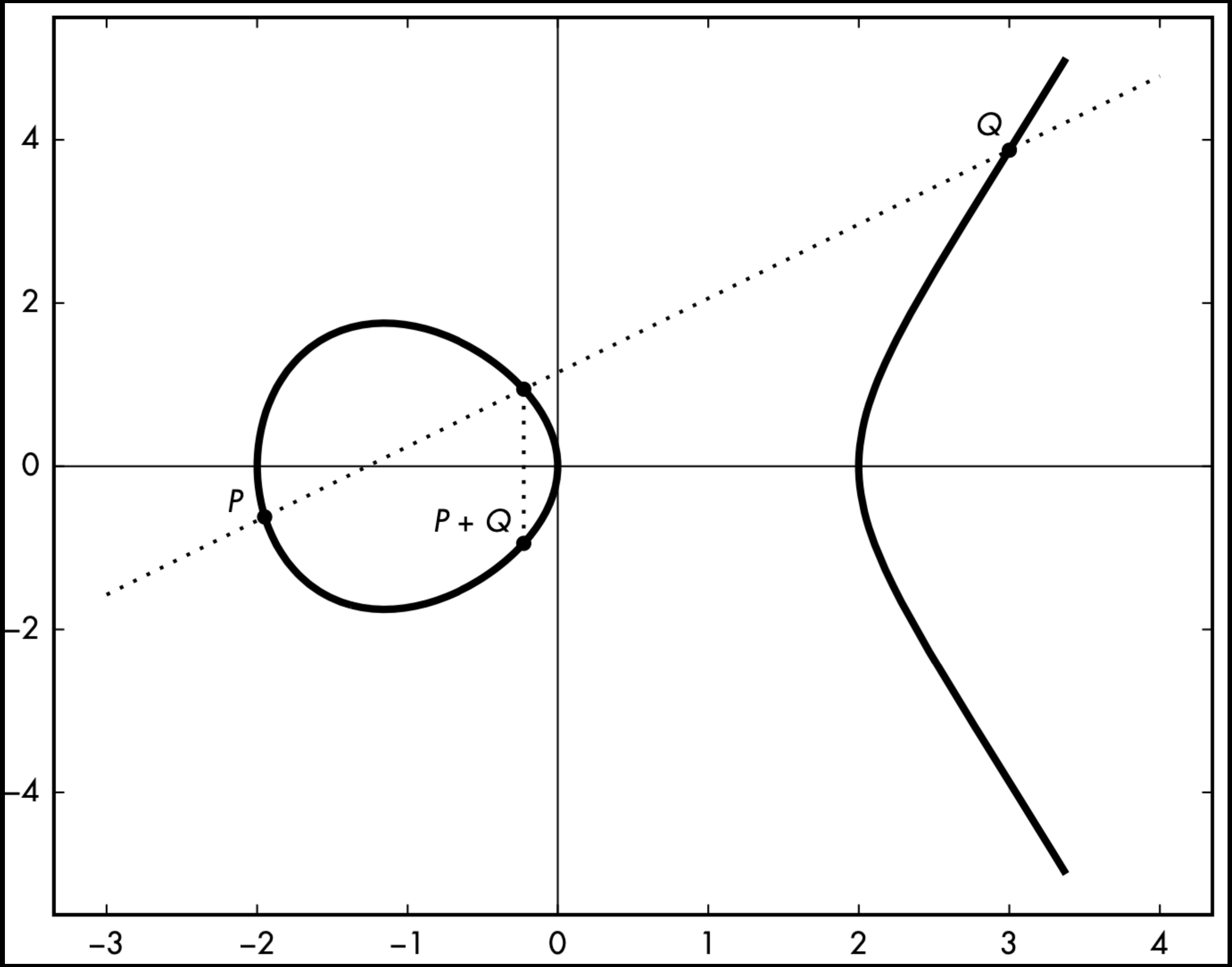
- Operations on elliptic curve points

Adding two points
Adding a point to its negative
Doubling a point
Multiplication

Adding and Multiplying Points

Adding two points

- Given $P(x_P, y_P)$ and $Q(x_Q, y_Q)$
- Compute $R(x_R, y_R) = P + Q$
- Compute the slope $m = \frac{y_Q - y_P}{x_Q - y_P}$
- $x_R = m^2 - x_P - x_Q$
- $y_R = m(x_P \ - x_R) - y_P$
- These formulas don't work when $P = Q$ or $P = -Q$



Adding and Multiplying Points

Adding a Point and Its Negative

- Given $P = (x_P, y_P)$ and $-P = (x_P, -y_P)$
- $P + (-P) = O$, where O is called the point at infinity
- The line between P and $-P$ runs to infinity

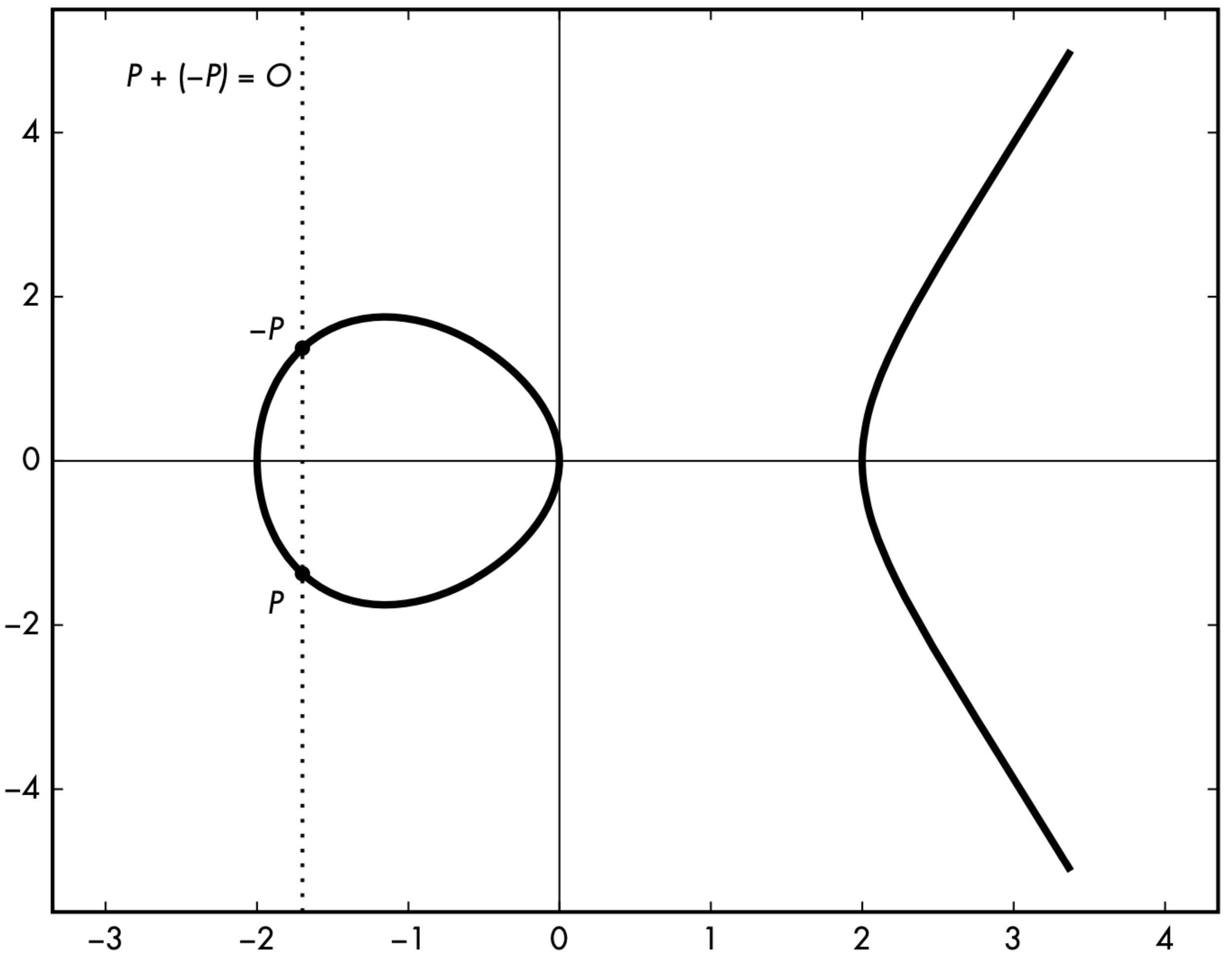


Figure 12-4: The geometric rule for adding points on an elliptic curve with the operation P + (–P) = O when the line between the points never intersects the curve

Adding and Multiplying Points

Doubling a point

- Given $P = Q \ \rightarrow P = P \ \rightarrow 2P$
  - We can't use the previous rules; we can't draw a line between P and itself.
- Compute the slope $m = \frac{3x_P^2 + a}{2y_P}$, where a is the curve's parameter in $y^2 = x^3 + ax + b$
- $x_R = m^2 - x_P - x_Q$
- $y_R = m(x_P - x_R) - y_P$

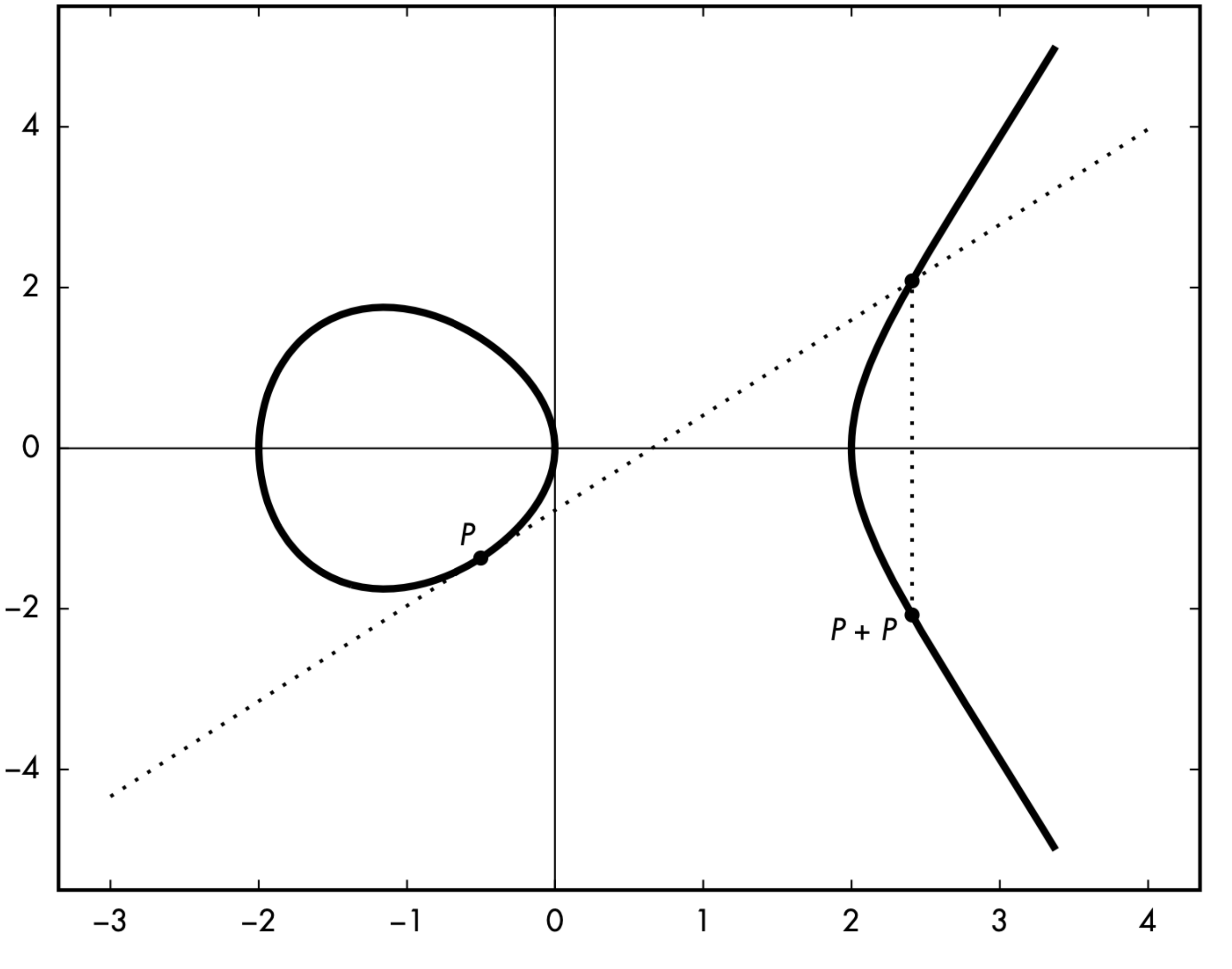


Figure 12-5: The geometric rule for adding points over an elliptic curve using the doubling operation P + P

Adding and Multiplying Points

Multiplication

- Given P and an integer k.

- To compute $R = kP$, we add P to itself $k - 1$ times
  - Use the previous addition law – very inefficient if k is large, 2^{256}

- To speedup the process, use the technique of fast exponentiation algorithm:
  - For example: instead of computing $8P = P + P + P + P + P + P + P + P$,
  - Compute $2P = P + P \ \rightarrow 4P = 2P + 2P \ \rightarrow 8P = 4P + 4P$

Content

The ECDLP Problem

- Elliptic Curve Discrete Logarithm Problem: finding the number k given a base point P where the point $Q = kP$.
- EASY to compute Q given k and P
- HARD to compute k given Q and P

DLP	ECDLP
• Operate on integers • Based on exponentiation • Big numbers → high security	• Operate on points • Based on multiplication • Small numbers → high security

The ECDLP Problem

- ECDLP: $Q = kP$ – the x, y coordinates numbers **modulo** a prime p

- Security level: $n/2$ bits, where n is the bit length of p
    - E.g., if p is 256 bits, then the security level is $\frac{256}{2} = 128$-bit
    - DLP requires numbers ≥ 1024 to achieve a similar security level

The ECDLP Problem

- To solve ECDLP, we need to **find collisions**.
- Finding collisions requires solving this equation (find c_1, d_1, c_2, d_2):

$$c_1P + d_1Q = c_2P + d_2Q$$

Where P and Q are the points in $Q = kP$

- To find the points, replace Q with the value kP:

$$c_1P + d_1kP = P(c_1 + d_1k) = c_2P + d_2kP = P(c_2 + d_2k)$$

- we know P and the modulus p, then compute

$$k = (c_1 - c_2)/(d_2 - d_1)$$

Content

Elliptic Curve Cryptography

- What is an Elliptic Curve
- Adding and Multiplying Points
- The ECDLP Problem
- DH Key Agreement over Elliptic Curves
- Elliptic Curves Signature
- Elliptic Curves Encryption
- Choosing a Curve
- How Things Can Go Wrong

TLS

- TLS
- The TLS Protocol Suite
- TLS1.3 Security
- How Things Can Go Wrong

DH Key Agreement over Elliptic Curves

- Classical DH: two parties establish a shared secret by exchanging public values

1. Select a fixed number g
2. Alice picks a secret number a and computes $A = g^a$, sends it to Bob
3. Bob picks a secret number b and computes $B = g^b$, Bob sends it to Alice
4. Compute the shared secret $A^b = B^a = g^{ab}$

DH Key Agreement over Elliptic Curves

- ECDH: similar to classical DH
  - Authenticated DH protocols and MQV can adapt the elliptic curves as hardness problem

Select a fixed point G

Alice picks a secret number d_A and computes $P_A = d_A G$, sends it to Bob

Bob picks a secret number d_B and computes $P_B = d_B G$, send it to Alice

Compute the shared secret $d_A P_B = d_B P_A = d_A d_B G$

Content

Elliptic Curves Signature

- ECDSA: Elliptic Curve Digital Signature Algorithm.
  - Used in Bitcoin and is supported by many TLS and SSH implementations.
- ECDSA consists of two algorithms:
  - **Generation** – the signer uses their **private key** to sign a message
  - **Verification** – the verifier uses the **public key** to check the signature's correctness
  - Private key: d, public key: $\mathrm{P} = dG$
- The signer and the verifier know:
  - The elliptic curve to use,
  - the number of points on the curve, n,
  - the base point G

Elliptic Curves Signature
ECDSA Signature Generation

Compute $h = hash(msg)$

Pick a random number k between 1 and $n - 1$

Compute kG, a point with coordinates (x, y)

Compute $r = x \bmod n$ and $s = \frac{h+rd}{k} \bmod n$

The signature is (r, s)

Elliptic Curves Signature
ECDSA Signature Verification

The verifier has the signature (r, s) and the message hash h

Compute $w = \frac{1}{s} = \frac{k}{h+rd}\ mod\ n$

Compute: $\mathrm{u} = \ wh = \frac{hk}{h+rd}, v = wr = \frac{rk}{h+rd}$

Given u, v, compute $Q = uG + vP$, where P is the public key and G is the fixed point

The signature is valid if the x coordinate of Q is equal to the r value of the signature

Elliptic Curves Signature
ECDSA Signature Verification

- The last step $Q = uG + vP$ works as follows:

$$Q = uG + vP \rightarrow uG + v(dG) \rightarrow G(u + vd)$$

- Substitute for $u = hk/(h + rd)$ and $v = rk/(h + rd)$

$$\therefore u + vd \rightarrow \frac{hk}{h + rd} + \frac{drk}{h + rd} = \frac{hk + drk}{h + rd} = \frac{k(h + rd)}{h + rd} = k$$

- So, $G(u + vd) = kG$, where k is chosen during signature generation.
- Valide signature: kG's x coordinate is equal to the r received

Elliptic Curves Signature

- RSA vs ECDLP signatures
  - The output: time to sign, time to verify, no. signs/second, no. verifications/second

```
$ openssl speed ecdsap256 rsa4096
                              sign      verify     sign/s     verify/s
rsa 4096 bits             0.007267s  0.000116s     137.6      8648.0
                              sign      verify     sign/s     verify/s
256 bit ecdsa (nistp256)   0.0000s    0.0001s    21074.6      9675.7
```

Listing 12-2: Comparing the speed of 4096-bit RSA signatures with 256-bit ECDSA signatures

Content

Elliptic Curves Encryption

- ECs are preferable for signing more than encryption.

- ECs have restrictions on the ptxt size: it can fit about 100 bits of plaintext.
  - RSA can fit almost 4000 bits with the same security level.

- Elliptic curves use the integrated encryption scheme (IES)
  - IES is a **hybrid key encryption algorithm** based on the Diffie–Hellman key exchange.
  - **Hybrid** means it combines the convenience of a public-key algorithms with the efficiency of a symmetric-key algorithms.
  - This is called Elliptic-Curve Integrated Encryption Scheme (**ECIES**).

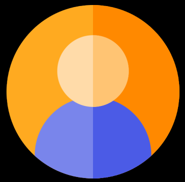
Elliptic Curves Encryption

- ECIES works as follows: G is a public fixed point

Alice

Pick random d_A

Bob



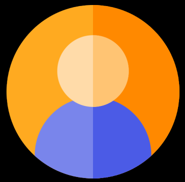
Pick random d_B

Elliptic Curves Encryption

- ECIES works as follows: G is a public fixed point

Alice

Bob



Pick random d_A

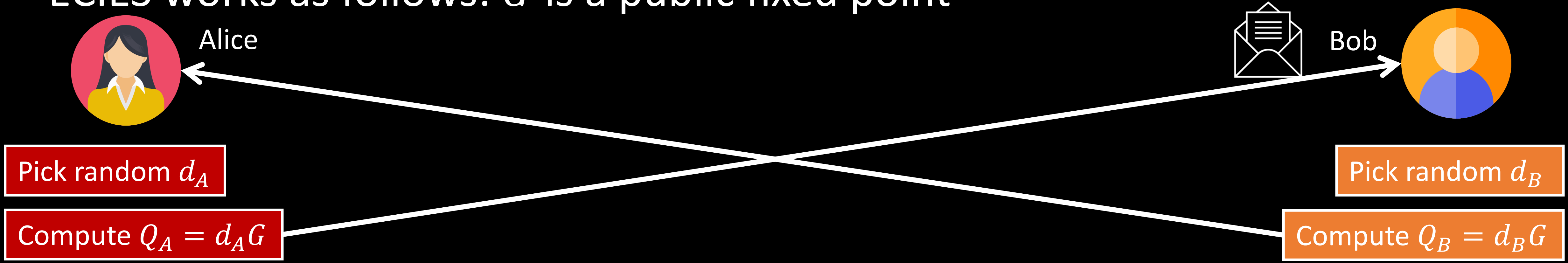
Compute $Q_A = d_A G$

Pick random d_B

Compute $Q_B = d_B G$

Elliptic Curves Encryption

- ECIES works as follows: G is a public fixed point



Elliptic Curves Encryption

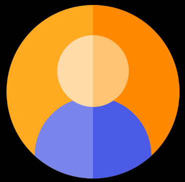
- ECIES works as follows: G is a public fixed point

Alice

Bob's PuK: Q_B

Bob

Alice's PuK: Q_A



Pick random d_A

Compute $Q_A = d_A G$

Compute shared secret: $S = d_A Q_B = d_A d_B G$

Pick random d_B

Compute $Q_B = d_B G$

Compute shared secret: $S = d_B Q_A = d_B d_A G$

Elliptic Curves Encryption

- ECIES works as follows: G is a public fixed point

Alice

Bob's PuK: Q_B

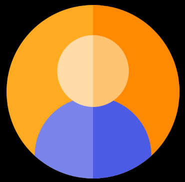
Pick random d_A

Compute $Q_A = d_A G$

Compute shared secret: $S = d_A Q_B = d_A d_B G$

Derive a symmetric key $S_k = KDF(S)$

Bob



Alice's PuK: Q_A

Pick random d_B

Compute $Q_B = d_B G$

Compute shared secret: $S = d_B Q_A = d_B d_A G$

Derive a symmetric key $S_k = KDF(S)$

Elliptic Curves Encryption

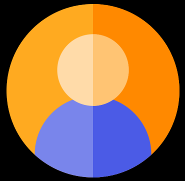
- ECIES works as follows: G is a public fixed point

Alice

Bob's PuK: Q_B

Bob

Alice's PuK: Q_A



Alice:

- Pick random d_A
- Compute $Q_A = d_A G$
- Compute shared secret: $S = d_A Q_B = d_A d_B G$
- Derive a symmetric key $S_k = KDF(S)$

Bob:

- Pick random d_B
- Compute $Q_B = d_B G$
- Compute shared secret: $S = d_B Q_A = d_B d_A G$
- Derive a symmetric key $S_k = KDF(S)$
- Use an authenticated cipher to encrypt M using S_k

ctxt tag

Elliptic Curves Encryption

- ECIES works as follows: G is a public fixed point

Alice

Bob's PuK: Q_B

Pick random d_A

Compute $Q_A = d_A G$

Compute shared secret: $S = d_A Q_B = d_A d_B G$

Derive a symmetric key $S_k = KDF(S)$

Use an authenticated cipher to verify the tag and decrypt M using S_k

Bob

Alice's PuK: Q_A

Pick random d_B

Compute $Q_B = d_B G$

Compute shared secret: $S = d_B Q_A = d_B d_A G$

Derive a symmetric key $S_k = KDF(S)$

Use an authenticated cipher to encrypt M using S_k

ctxt tag

Content

Choosing a Curve

- You MUST carefully choose the curve that IS SECURE.
  - In practice, you may use **standard curves**.
- Some criteria for safe curves:
  - The order of the curve (number of points) **should not** be a product of small numbers
  - More can be found at https://safecurves.cr.yp.to/
- *Sony PS3 was hacked by $fail0verflow$ because the developers misused the ECDSA (https://fahrplan.events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf)*
  - *The developers reused the number k in the signing process (https://deeprnd.medium.com/decoding-the-playstation-3-hack-unraveling-the-ecdsa-random-generator-flaw-e9074a51b831)*

Choosing a Curve

Standard curves: NIST Curves

- 5 curves that work modulo a prime number – called **Prime Curves.**

- The most common curve is the P-256:
  - The modulus is $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
  - The equation of the curve: $y^2 = x^3 - 3x + b$, where b is 256-bit number
  - b can be computed by computing $SHA1$ of the following constant: c49d360886e704936a6678e1139d26b7819f7e90
  - No one knows why the NSA picked this particular constant!

Choosing a Curve

Non-standard curves: Curve25519

- Faster and have shorter keys than NIST curves
- Doesn’t have suspicious constants
- Use a unified formula for adding distinct points or doubling a point
- The prime number is $2^{255} - 19$
- The equation is $y^2 = x^3 + 486662x^2 + x$
- The b coefficient is 486662
- Used in Google Chrome, Apple systems, OpenSSH, and many other systems.

Content

How Things Can Go Wrong

ECDSA with bad randomness

- The signing process requires a secret **random** number k: $s = \frac{h+rd}{k} mod\ n$
- If the same k is used to sign another message, an attacker can reveal k and then recover the private key d
- Used to hack PlayStation 3

How Things Can Go Wrong

ECDSA with bad randomness

$$s_1 = \frac{(h_1 + rd)}{k} mod\ n, \qquad s_2 = \frac{h_2 + rd}{k}\ mod\ n$$

$$s_1 - s_2 = \left(\frac{(h_1 + rd)}{k} - \frac{h_2 + rd}{k}\right) mod\ n = \frac{(h_1 - h_2)}{k}\ mod\ n$$

$$\therefore k = \frac{h_1 - h_2}{s_1 - s_2}\ mod\ n$$

Now, d can be computed as follows:

$$\frac{ks_1 - h_1}{r} = \frac{(h_1 + rd) - h_1}{r} = \frac{rd}{r} = d$$

How Things Can Go Wrong

Breaking ECDH using another curve

- This attack is called **invalid curve attack**
- Occur when you compute the sum of two curves P and Q: $P + Q$
- Computing $P + Q$ does not use the b coefficient of the curves
- Thus, you can never be sure that you're working on the right curve
  - Because you may be adding points on a different curve with a different b coefficient
- Refer to the textbook for more explanation
- https://github.com/ashutosh1206/Crypton/blob/master/Diffie-Hellman-Key-Exchange/Attack-Invalid-Curve-Point/README.md

TASK

- Implement the ECDSA algorithm using pycryptodome in Python
- Implement the EC Encryption algorithm using AES algorithm in pycryptodome
    - Pycryptodome has no direct support for ECC encryption/decryption

Content

TLS

- TLS protects the connection between servers and clients.
    - TLS is an updated version of the SSL

- TLS is application agnostic:
    - Websites and their visitors (the s in HTTPS)
    - Email servers
    - Mobile apps and their servers
    - Video game servers and the players
    - Connection between IoT devices

TLS

- TLS establishes secure channel between two machines to protect:
  - Integrity – data is unmodified
  - Authenticity – data is authenticated
  - Confidentiality – data is secure

- TLS protects against Man-in-the-Middle attacks.
  - MitM: an attacker intercepts encrypted traffic, decrypts it, and re-encrypts it to send to the receiving party.
  - Use certificates to authenticate servers

Content

Elliptic Curve Cryptography
What is an Elliptic Curve
Adding and Multiplying Points
The ECDLP Problem
DH Key Agreement over Elliptic Curves
Elliptic Curves Signature
Elliptic Curves Encryption
Choosing a Curve
How Things Can Go Wrong

TLS
TLS
The TLS Protocol Suite
TLS1.3 Security
How Things Can Go Wrong

The TLS Protocol Suite

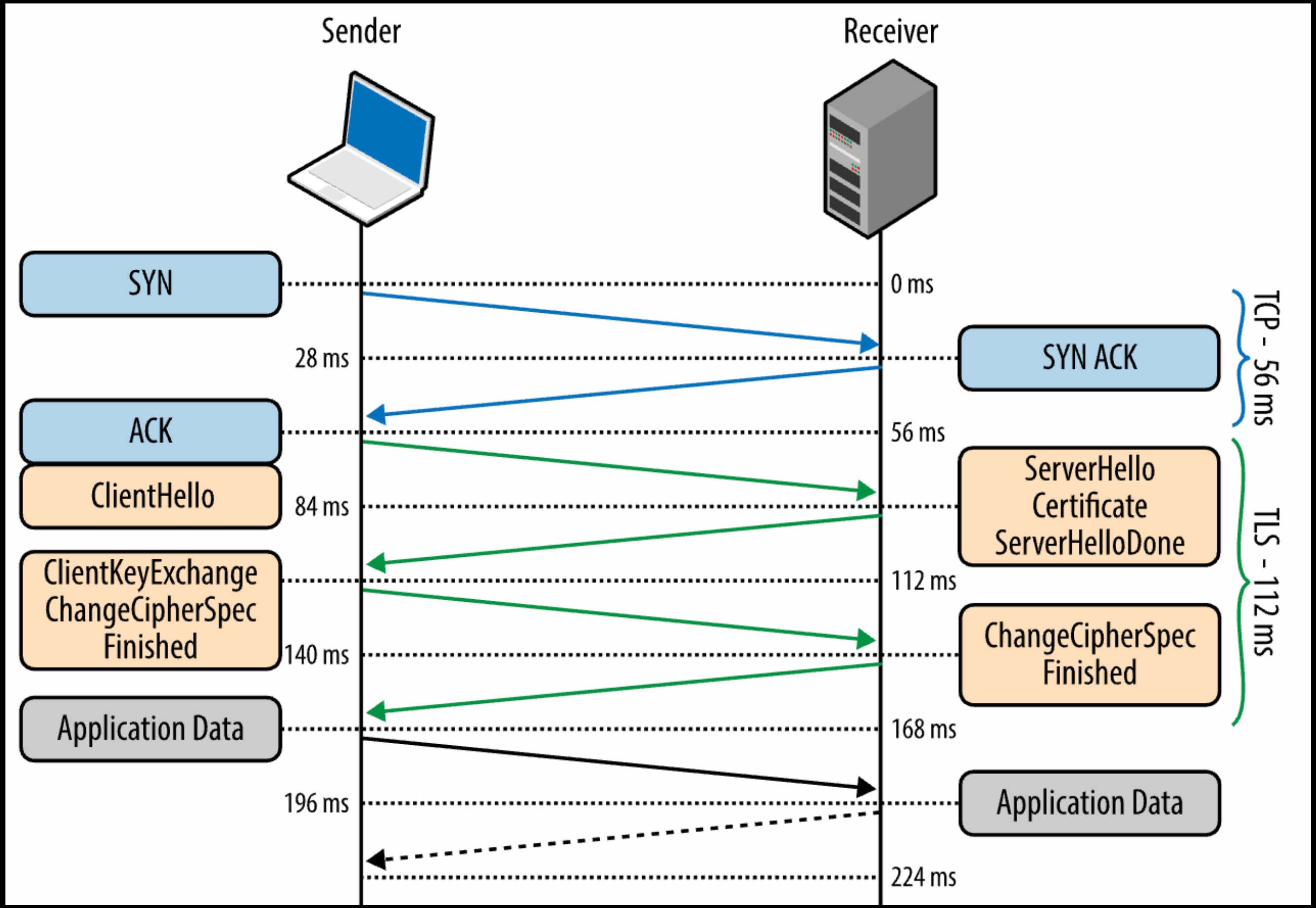
- TLS isn't a single protocol, but a **suite** of different versions of other protocols
- It operates between TCP and application layer protocols (e.g., HTTPS, SMTP)
  - There is a different version for the UDP, called DTLS
- Two main protocols in TLS

Main protocols in TLS

- Record protocol — Defines the packet format to encapsulate the data
- Handshake protocol — A key agreement protocol. It started by the client to initiate a secure connection

The TLS Protocol Suite

- The Handshake protocol



The TLS Protocol Suite

- **Certificate**: public key and a signature of the key and other information
    - The server uses the certificate to authenticate itself to the client
- The browser receives the certificate from a *certificate authority* (CA)
    - CA: a trusted entity that issues digital certificates to authenticate web servers
    - If the signature is verified, the browser establishes a secure connection
- A list of CAs are **hard coded** into the browser and the OS
    - Your system stores the public key of the CA, where the private key (signing key) belongs to the trusted organization

The TLS Protocol Suite

- Example:

```
$ openssl s_client -connect www.google.com:443
CONNECTED(00000003)
--snip--
---
Certificate chain
❶ 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
❷ 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
❸ 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEgDCCA2igAwIBAgIISCr6QCbz5rowDQYJKoZIhvcNAQELBQAwSTELMAkGA1UE
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvb2dsZSBJbnRl
--snip--
cb9reU8in8yCaH8dtzrFyUracpMureWnBeajOYXRPTdCFccejAh/xyH5SKDOOZ4v
3TP9GBtClAH1mSXoPhX73dp7jipZqgbY4kiEDNx+hformTUFBDHDOeO/s2nqwuWL
pBH6XQ==

-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
issuer=/C=US/O=Google Inc/CN=Google Internet Authority G2
--snip--
```

The TLS Protocol Suite

The certificate chain is an ordered list of certificates.

It contains the TLS Certificate and CA’ Certificates.

It enables the receiver to verify that the sender and all CA's are trustworthy.

```
$ openssl s_client -connect www.google.com:443
CONNECTED(00000003)
--snip--
---
Certificate chain
❶ 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
    i:/C=US/O=Google Inc/CN=Google Internet Authority G2
❷ 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
    i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
❸ 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
    i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEgDCCA2igAwIBAgIISCr6QCbz5rowDQYJKoZIhvcNAQELBQAwSTELMAkGA1UE
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvb2dsZSBJbnRl
--snip--
cb9reU8in8yCaH8dtzrFyUracpMureWnBeajOYXRPTdCFccejAh/xyH5SKDOOZ4v
3TP9GBtClAH1mSXoPhX73dp7jipZqgbY4kiEDNx+hformTUFBDHDOeO/s2nqwuWL
pBH6XQ==

-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
issuer=/C=US/O=Google Inc/CN=Google Internet Authority G2
--snip--
```

The TLS Protocol Suite

s: describes the subject name

i: describes the issuer name

```
Certificate chain
❶ 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
    i:/C=US/O=Google Inc/CN=Google Internet Authority G2
❷ 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
    i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
❸ 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
    i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
```

The TLS Protocol Suite

- Certificate 0 is the one received by $www.google.com$
  - Issued by Google Internet Authority
- Certificate 1 belongs to the entity that signed certificate 0
  - GeoTrust is the issuer of certificate 1
  - Grants permission to issue certificate 0
- Certificate 2 belongs to the entity that signed certificate 1
  - Equifax is the issuer of certificate 2
  - Grants permission to issue certificate 1

```
Certificate chain
❶ 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
    i:/C=US/O=Google Inc/CN=Google Internet Authority G2
❷ 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
    i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
❸ 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
    i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
```

The TLS Protocol Suite

The data of the certificate is decoded in base64 format.

You can read the data from the browser or using OpenSSL to decode it

```
$ openssl s_client -connect www.google.com:443
CONNECTED(00000003)
--snip--
---
Certificate chain
❶ 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
    i:/C=US/O=Google Inc/CN=Google Internet Authority G2
❷ 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
    i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
❸ 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
    i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIEgDCCA2igAwIBAgIISCr6QCbz5rowDQYJKoZIhvcNAQELBQAwSTELMAkGA1UE
BhMCVVMxEzARBgNVBAoTCkdvb2dsZSBJbmMxJTAjBgNVBAMTHEdvb2dsZSBJbnRl
--snip--
cb9reU8in8yCaH8dtzrFyUracpMureWnBeajOYXRPTdCFccejAh/xyH5SKDOOZ4v
3TP9GBtClAH1mSXoPhX73dp7jipZqgbY4kiEDNx+hformTUFBDHD0eO/s2nqwuWL
pBH6XQ==

-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
issuer=/C=US/O=Google Inc/CN=Google Internet Authority G2
--snip--
```

The TLS Protocol Suite

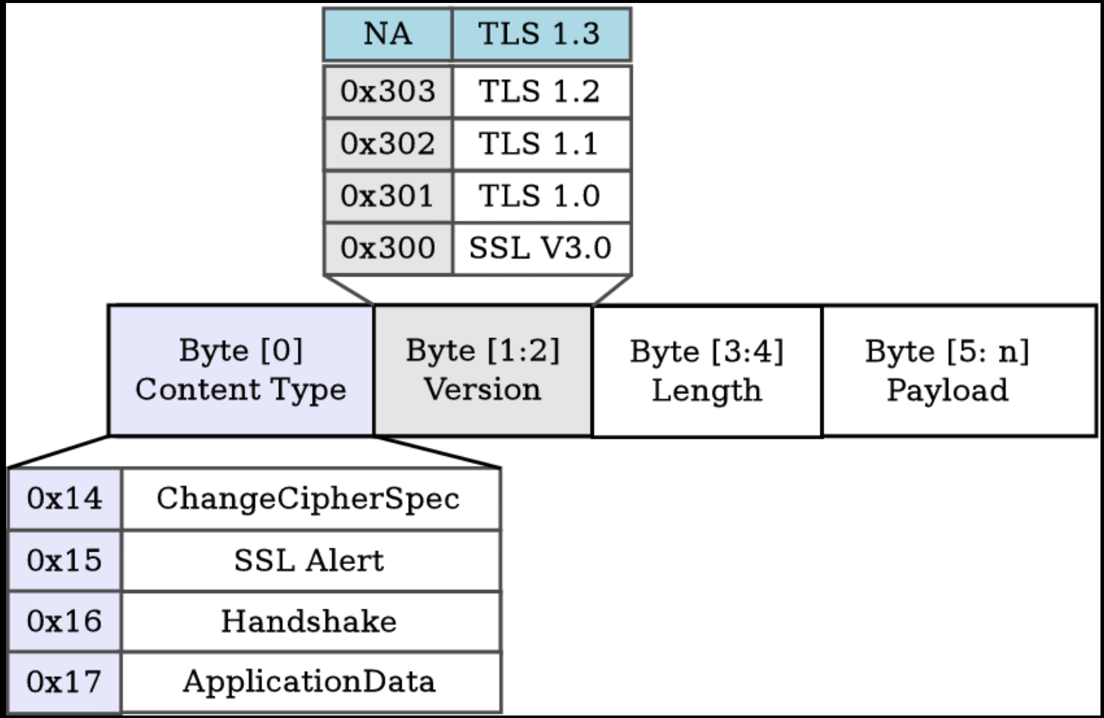
To read the certificate using OpenSSL, copy the certificate, run the shown command, and paste the certificate.

- The data shown are:
  - The version and the serial number
  - The signature algorithm
    - SHA256 with RSA encryption
  - The issuer of the certificate
    - Google Internet Authority
  - Validity of the certificate:
    - Expiry date = Not After
  - The encryption algorithm and the public key
    - A 2048-bit modulus
    - The exponent is 65537

```
$ openssl x509 -text -noout
-----BEGIN CERTIFICATE-----
--snip--
-----END CERTIFICATE-----
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 5200243873191028410 (0x482afa4026f3e6ba)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=Google Inc, CN=Google Internet Authority G2
        Validity
            Not Before: Dec 15 14:07:56 2016 GMT
            Not After : Mar  9 13:35:00 2017 GMT
        Subject: C=US, ST=California, L=Mountain View, O=Google Inc,
CN=www.google.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:bc:bc:b2:f3:1a:16:3b:c6:f6:9d:28:e1:ef:8e:
                    92:9b:13:b2:ae:7b:50:8f:f0:b4:e0:36:8d:09:00:
--snip--
                    8f:e6:96:fe:41:41:85:9d:a9:10:9a:09:6e:fc:bd:
                    43:fa:4d:c6:a3:55:9a:9e:07:8b:f9:b1:1e:ce:d1:
                    22:49
                Exponent: 65537 (0x10001)
--snip--
    Signature Algorithm: sha256WithRSAEncryption
```

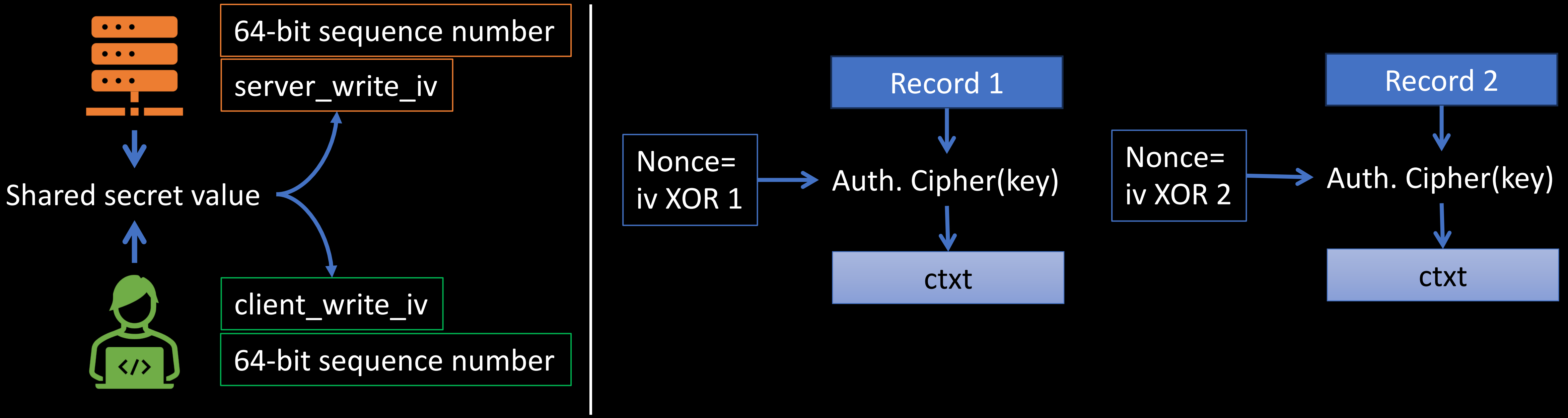
The TLS Protocol Suite

- TLS 1.3 data packets are called *records.*
- The TLS record protocol (the *record layer*) is a transport protocol.
- Structure of a TLS record.



The TLS Protocol Suite

- TLS protocol uses authenticated ciphers.
  - Authenticated ciphers output a ciphertext and a tag.
- Ciphers may use Nonces, but TLS doesn’t specify the nonce to be used!
- The nonces used are derived from 64-bit sequence numbers:



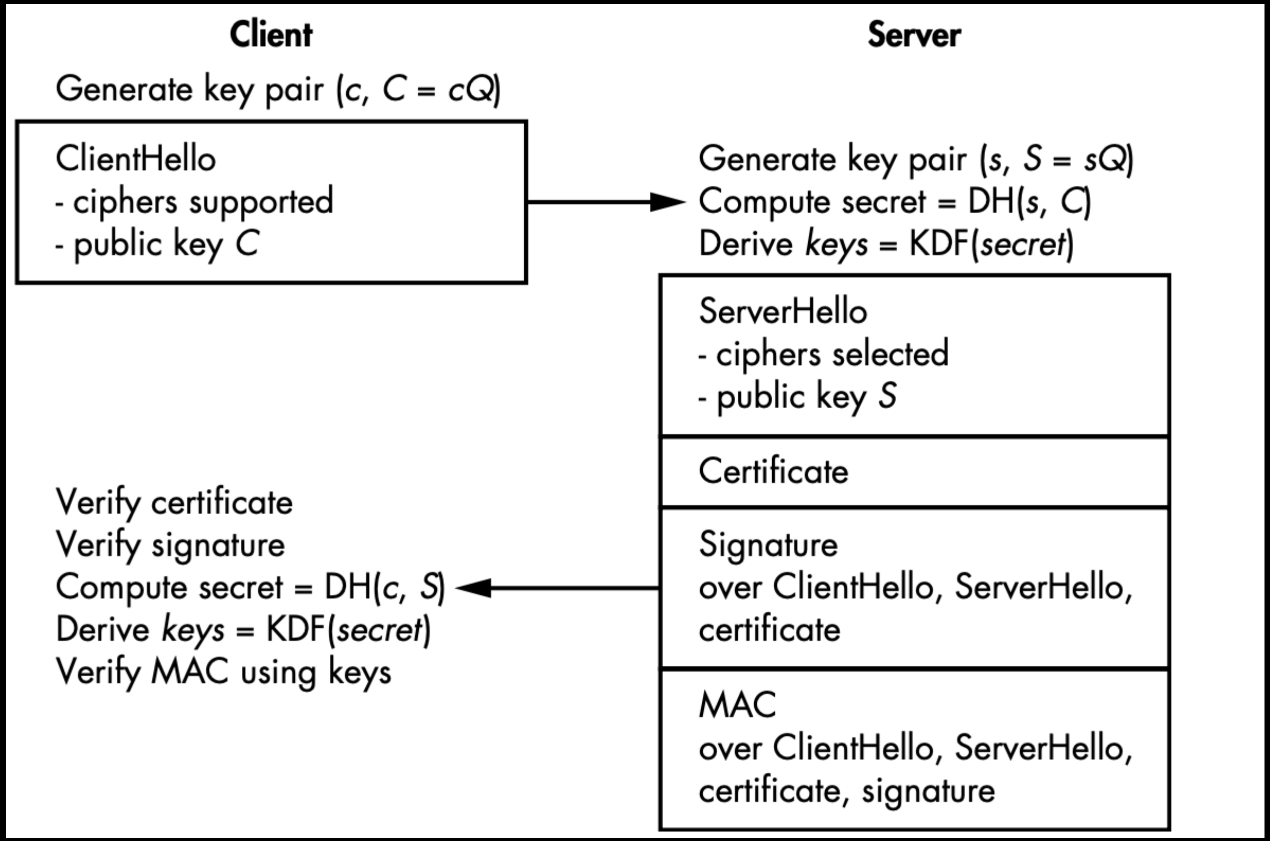
The TLS Protocol Suite

- For more information about the key calculation of the TLS, review:
  - https://blog.cloudflare.com/tls-nonce-nse/
  - https://www.rfc-editor.org/rfc/rfc5246.html#section-6.3

- TLS 1.3 has a feature called *zero-padding*:
  - Add extra zeros as a padding to the plaintext to hide its actual size

The TLS Protocol Suite

- The *handshake* is the key TLS agreement protocol.
  - Establish client/server shared secret keys to initiate secure communications.



The TLS Protocol Suite

- TLS 1.3 uses three three algorithms

Authenticated encryption algorithm (key size is 128- or 256- bit)

- AES-GCM – AES cipher with Galois/Counter mode
- AES-CCM – AES cipher with Counter and Cipher Block Chaining mode
- ChaCha20 stream cipher with Poly1305 MAC algorithm

Key derivation function

- A hash function that derives secret keys from a shared secret
- Based on a construction called HMAC
- HMAC uses SHA-256 or SHA-384 hash functions

Diffie Hellman for key exchange

- ECDHE – Diffie-Hellman based on elliptic curves
- Traditional Diffie-Hellman

Content

Elliptic Curve Cryptography
What is an Elliptic Curve
Adding and Multiplying Points
The ECDLP Problem
DH Key Agreement over Elliptic Curves
Elliptic Curves Signature
Elliptic Curves Encryption
Choosing a Curve
How Things Can Go Wrong

TLS
TLS
The TLS Protocol Suite
TLS1.3 Security
How Things Can Go Wrong

TLS1.3 Security

Authentication in TLS 1.3

- During *handshake*, the server authenticates itself to the client using *certificates.*

- Clients authenticate themselves to a server after establishing a secure connection.
    - Such as Gmail using a username and a password or a *secure cookie*

- It’s rare to find clients authenticate themselves to servers
    - Complex operation

TLS1.3 Security

Forward secrecy in TLS 1.3

- Forward secrecy: previous sessions aren’t compromised when the present session is compromised.

- TLS 1.3 forward secrecy holds for a **data leak** and a **data breach** models:
    - **Data leak model:** only temporary secrets are compromised
        - The attacker recovers DH private keys of a specific session
    - **Data breach model:** long-term secrets are compromised
        - The attacker recovers the private key of a certificate
        - The attacker will only be able to **falsely authenticate itself** to the client, but cannot decrypt previous sessions

TLS1.3 Security

Forward secrecy in TLS 1.3

- TLS forward secrecy can be compromised if attackers read the session keys (previous and present) found in stored in memory/file.

- To ensure forward secrecy, the TLS implementation must erase the keys from memory once they are no longer in use.

Content

How Things Can Go Wrong

- Common failovers

Failovers

- Compromised Certificate Authority
  - CAs are trusted third parties.
  - Compromised CA → compromised certificates' private keys
  - Dutch certificate authority DigiNotar was hacked, the attacker could authenticated fake websites
- Compromised Server
  - Attacker target the web servers to get into the system
  - Independent of the TLS protocol
  - Attackers can see all the data in clear
- Compromised Client
  - Attackers target applications running at the client side (e.g., browser)
  - They can read session keys and read decrypted data
  - They can install rogue certificates to authenticate fake servers
- Bugs in Implementations

How Things Can Go Wrong

- The most classical vulnerability on TLS was the **heartbleed**.
    - a buffer overflow in the OpenSSL implementation of a TLS feature called heartbeat.
    - The server doesn't confirm that the length is correct; read as many characters as the client tells it to.