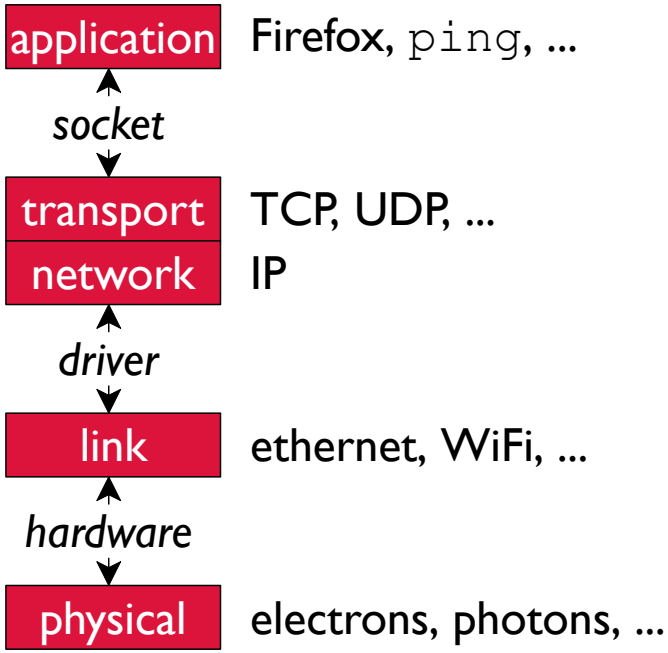
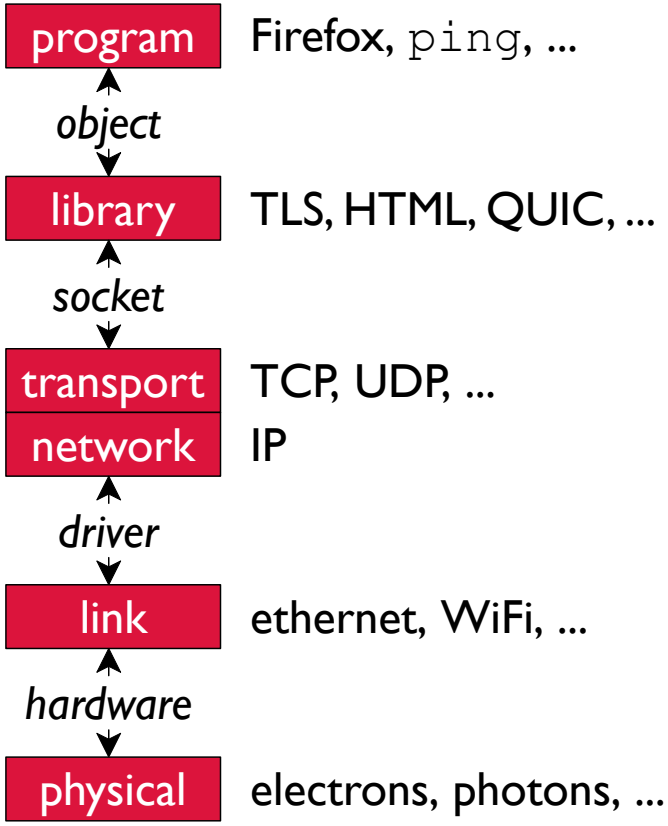
Network Layers

application	Firefox, `ping`, ...
transport	TCP, UDP, ...
network	IP
link	ethernet, WiFi, ...
physical	electrons, photons, ...

Network Layers



Network Layers



Transport Layer Security

Transport Layer Security (TLS):
a layer between applications and TCP

- Encrypts communication over TCP
- Typically presented as an alternative socket interface

Originally **Secure Socket Layer (SSL)**

and many APIs still say "SSL"

TLS Server

```
import javax.net.ssl.SSLServerSocketFactory;
/* .... same as TCP client .... */

/* Needs a keystore file created with
     openssl pkcs12 -export -inkey serverPrivate.key
             -in CASignedServerCertificate.pem
             -out ServerKeyStore.p12
             -passout 'pass:hello!'
*/

public class Main {
    public static void main(String[] args) throws IOException {
        int server_port = 5678;
        System.setProperty("javax.net.ssl.keyStore", "ServerKeyStore.p12");
        System.setProperty("javax.net.ssl.keyStorePassword", "hello!");
        ServerSocket listener = SSLServerSocketFactory.getDefault().createServerSocket(server_port);
        System.out.println("Listening at " + server_port);
        /* .... same as TCP server .... */
    }
}
```

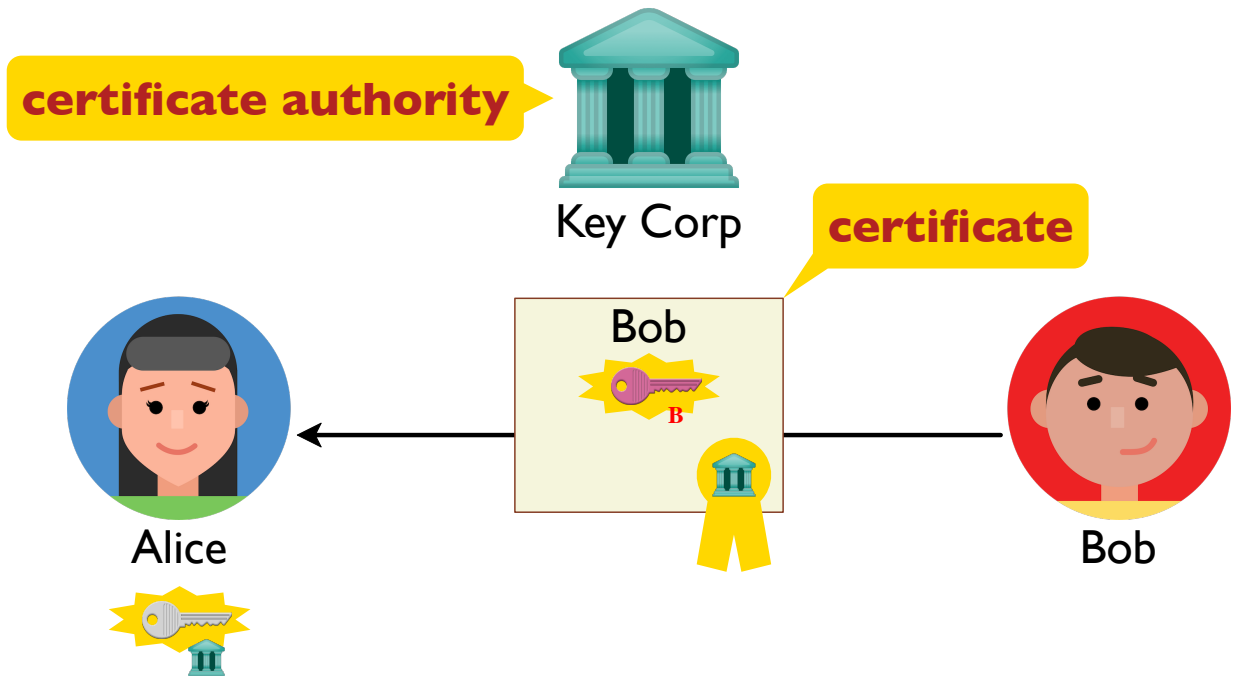
TLS Client

```
import javax.net.ssl.SSLSocketFactory;
import java.io.IOException;
/* .... same as TCP client .... */

/* Needs a keystore file created with
     keytool -importcert --file CASignedServerCertificate.pem
             -keystore ClientKeyStore.jks
             -storepass 'hello!' -noprompt */

public class Main {
    public static void main(String[] args) throws IOException {
        int server_port = 5678;
        System.setProperty("javax.net.ssl.trustStore", "ClientKeyStore.jks");
        System.setProperty("javax.net.ssl.trustStorePassword", "hello!");
        Socket socket = SSLSocketFactory.getDefault().createSocket("localhost", server_port);
        InputStream input = socket.getInputStream();
        /* .... same as TCP client .... */
    }
}
```

Distributing Public Keys



Using OpenSSL for Certificates

Create a certificate authority (like Key Corp):

```
openssl req -x509 -newkey rsa:4096 -nodes -days 30
            -keyout CAprivateKey.pem
            -out CAcertificate.pem
```

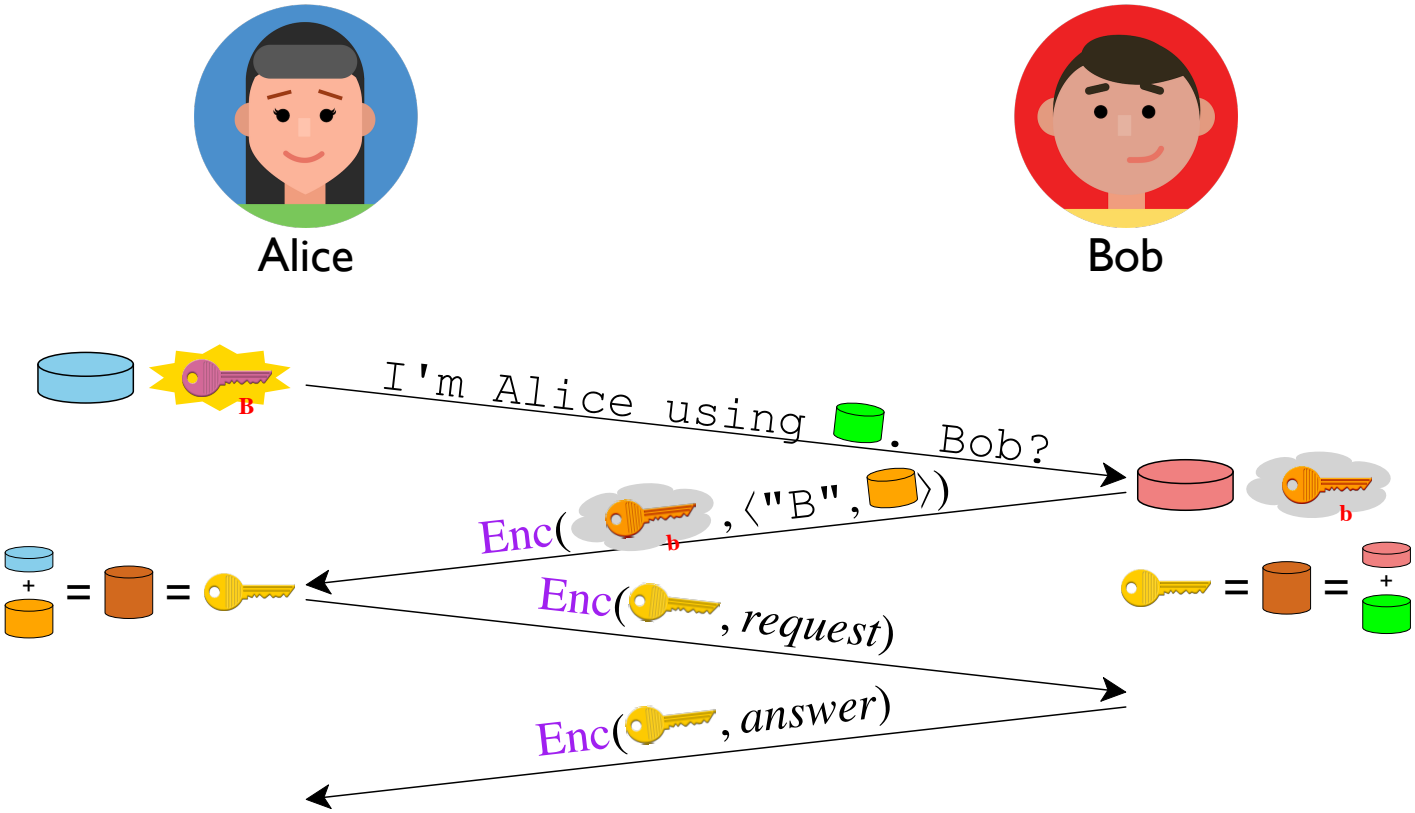
Create a certificate signing request (from Bob or Alice):

```
openssl req -new -newkey rsa:4096 -nodes
            -keyout bobPrivate.key
            -out bob.csr
```

Sign a certificate (by Key Corp):

```
openssl ca -config config.cnf
           -cert CAcertificate.pem
           -keyfile CAprivateKey.pem
           -in bob.csr
           -out CASignedBobCertificate.pem
```

Authentication and Encryption



TLS Handshake

client

server

R_{client}

TLS Handshake

client

server

R_{client} $\xrightarrow{\langle \texttt{ClientHello}, R_{client}, \textit{cipher choices} \rangle}$ R_{client}

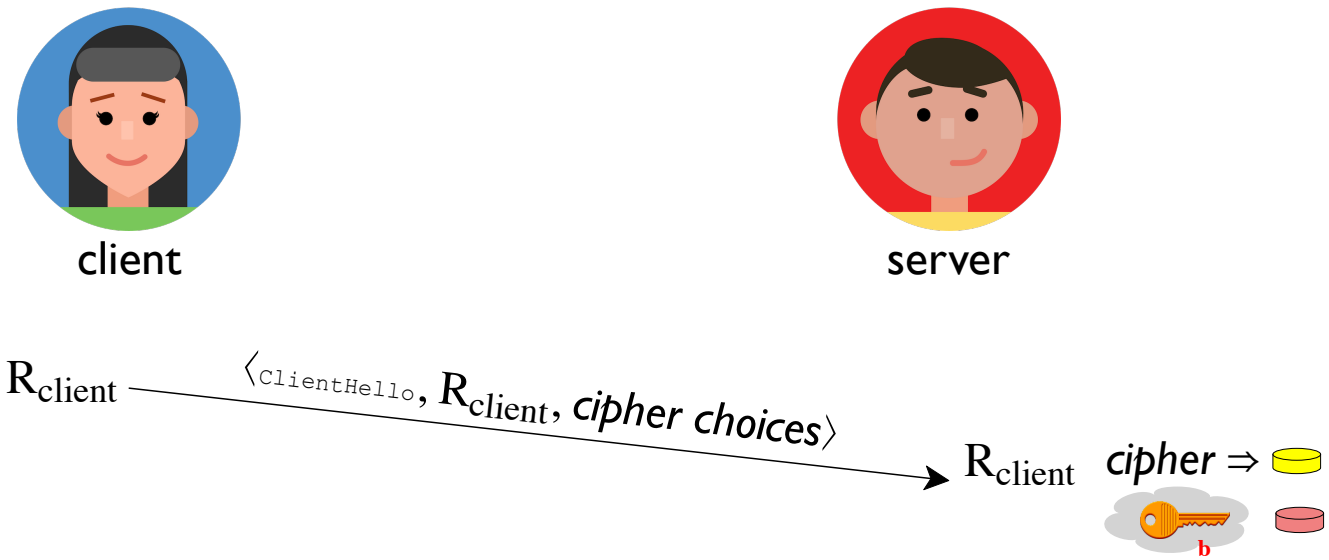
TLS Handshake

client

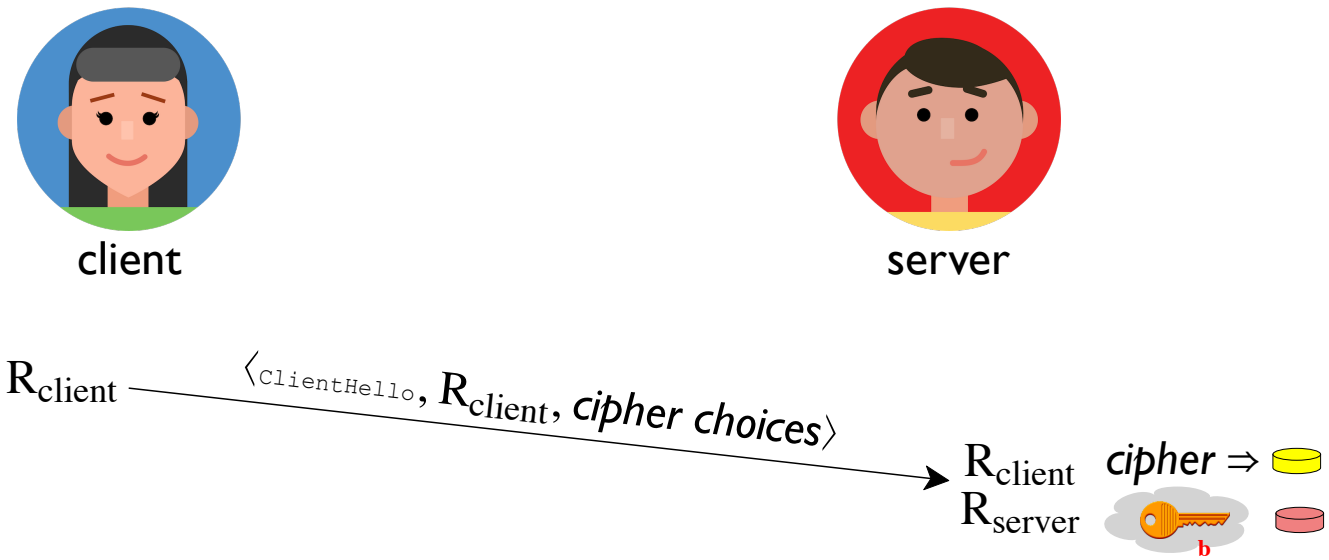
server

R_{client} $\langle$`ClientHello`, R_{client}, *cipher choices*$\rangle$ R_{client} *cipher* $\Rightarrow$

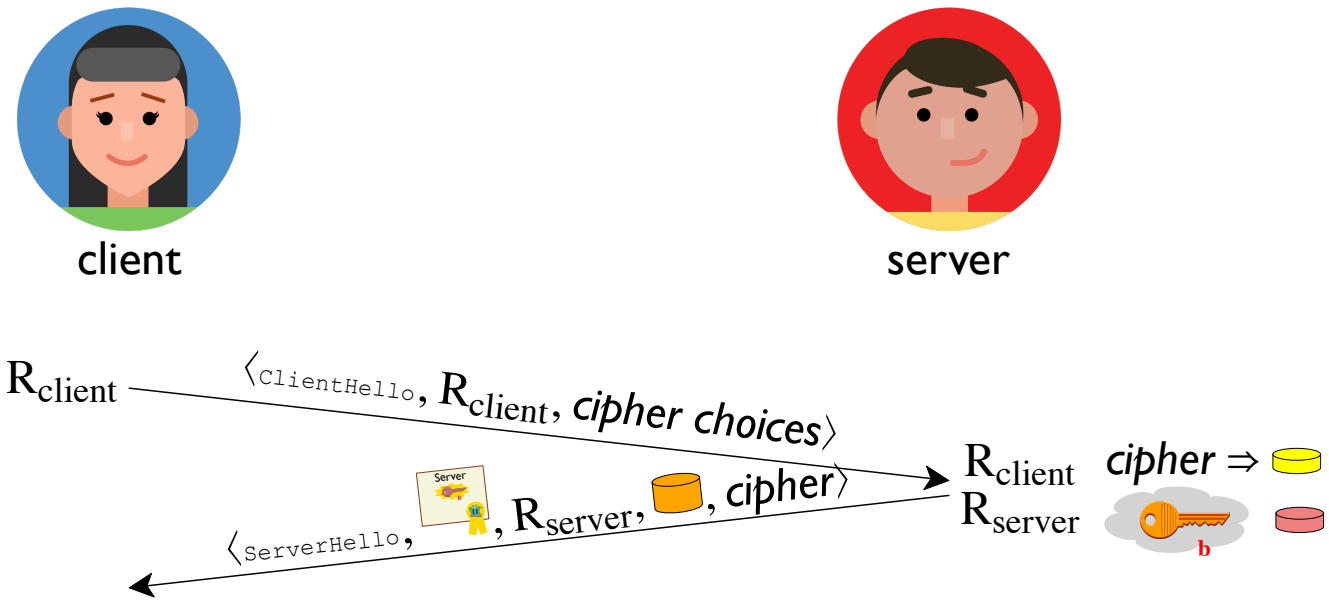
TLS Handshake



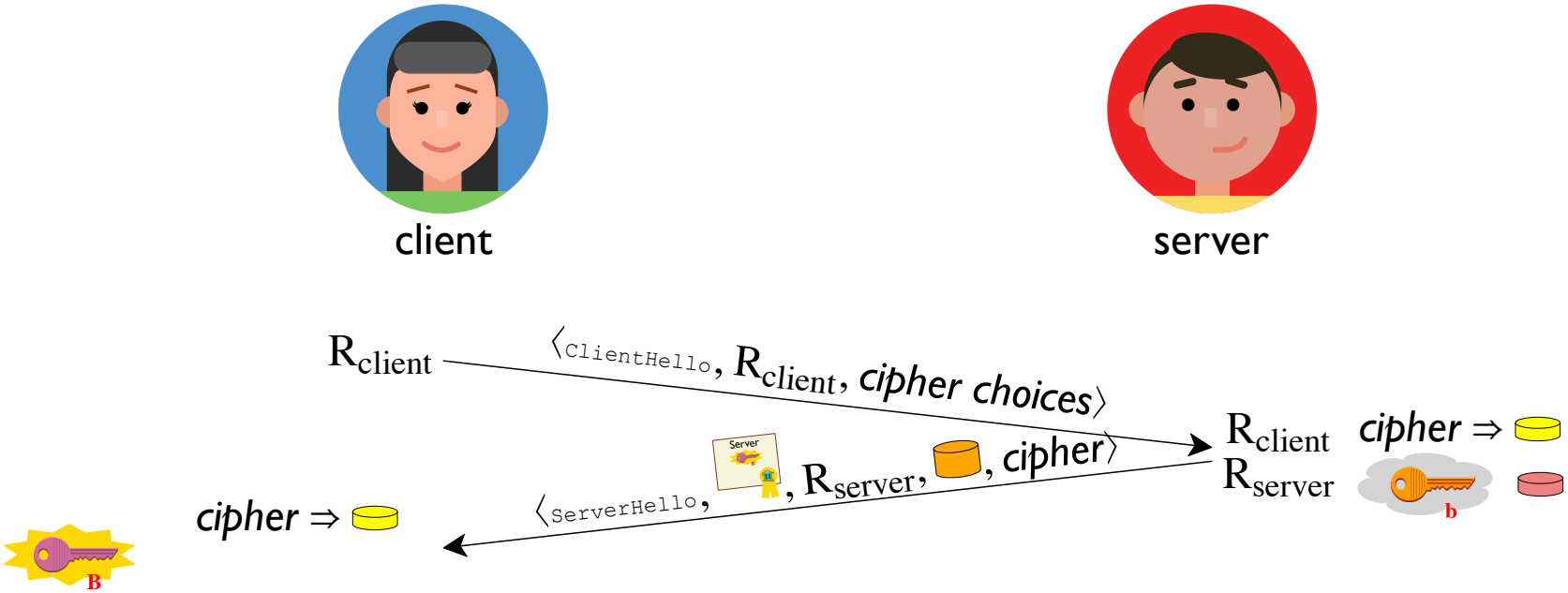
TLS Handshake



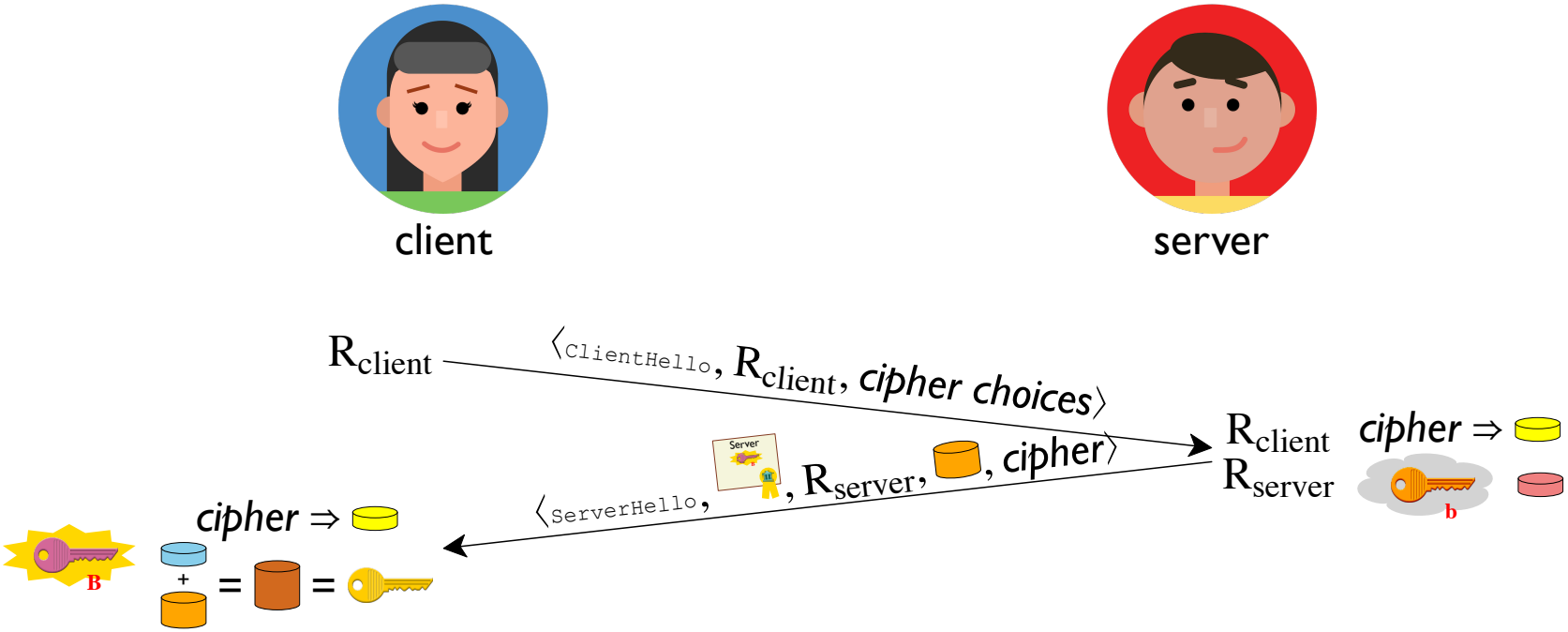
TLS Handshake



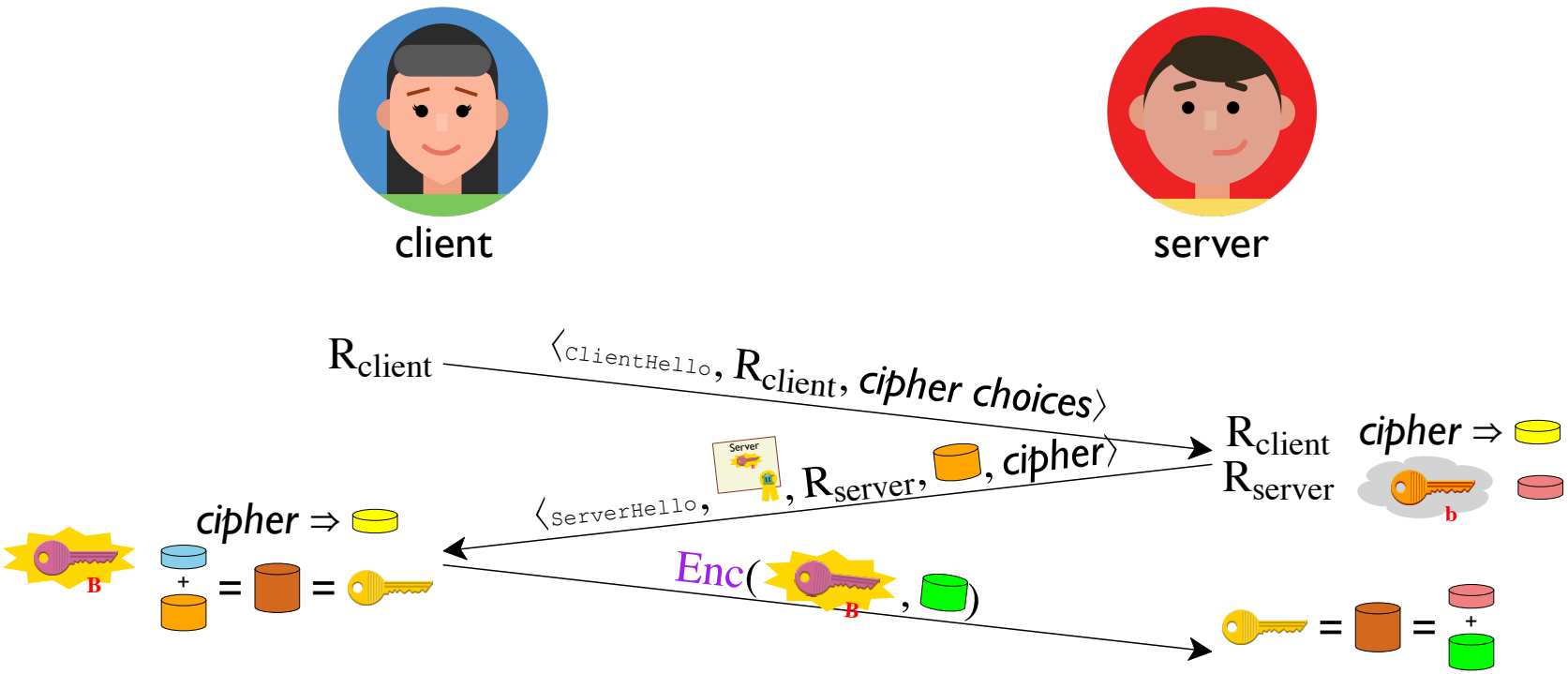
TLS Handshake



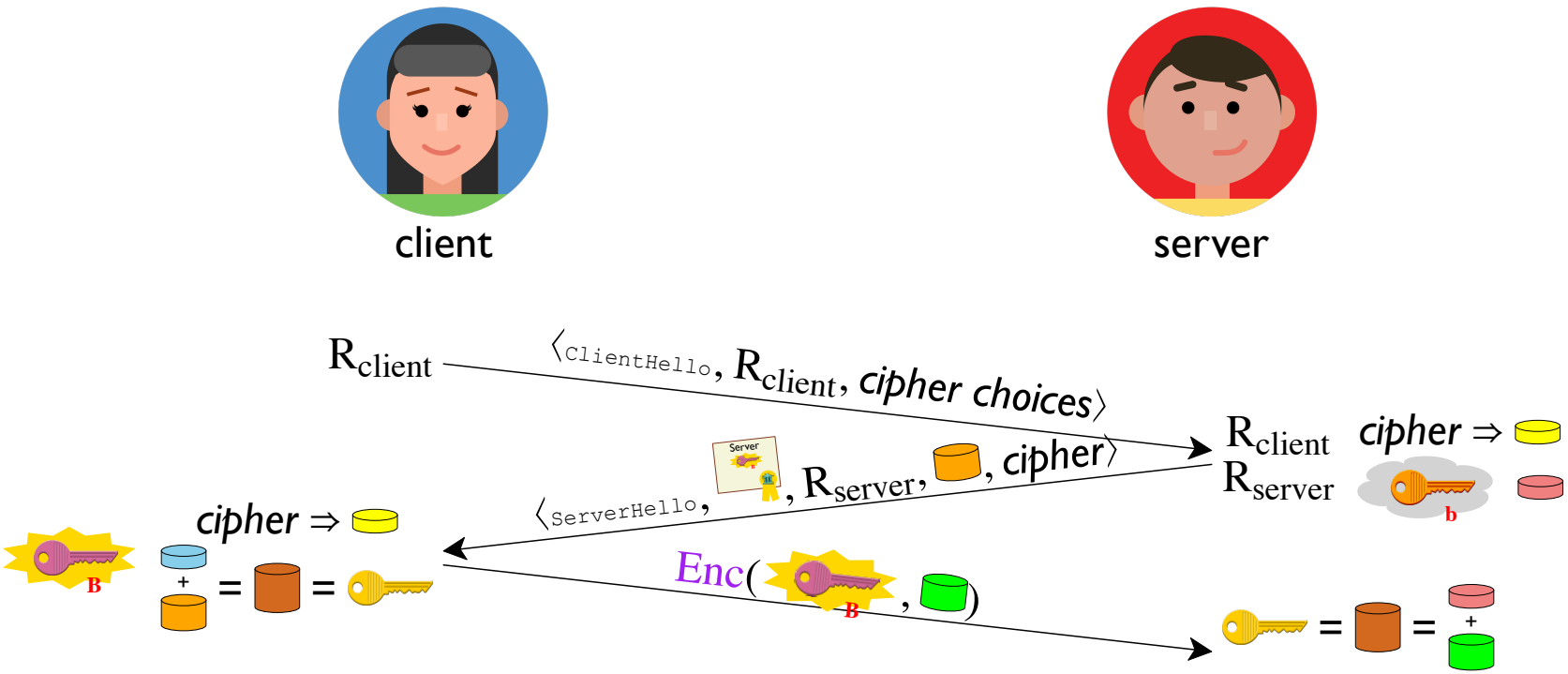
TLS Handshake



TLS Handshake

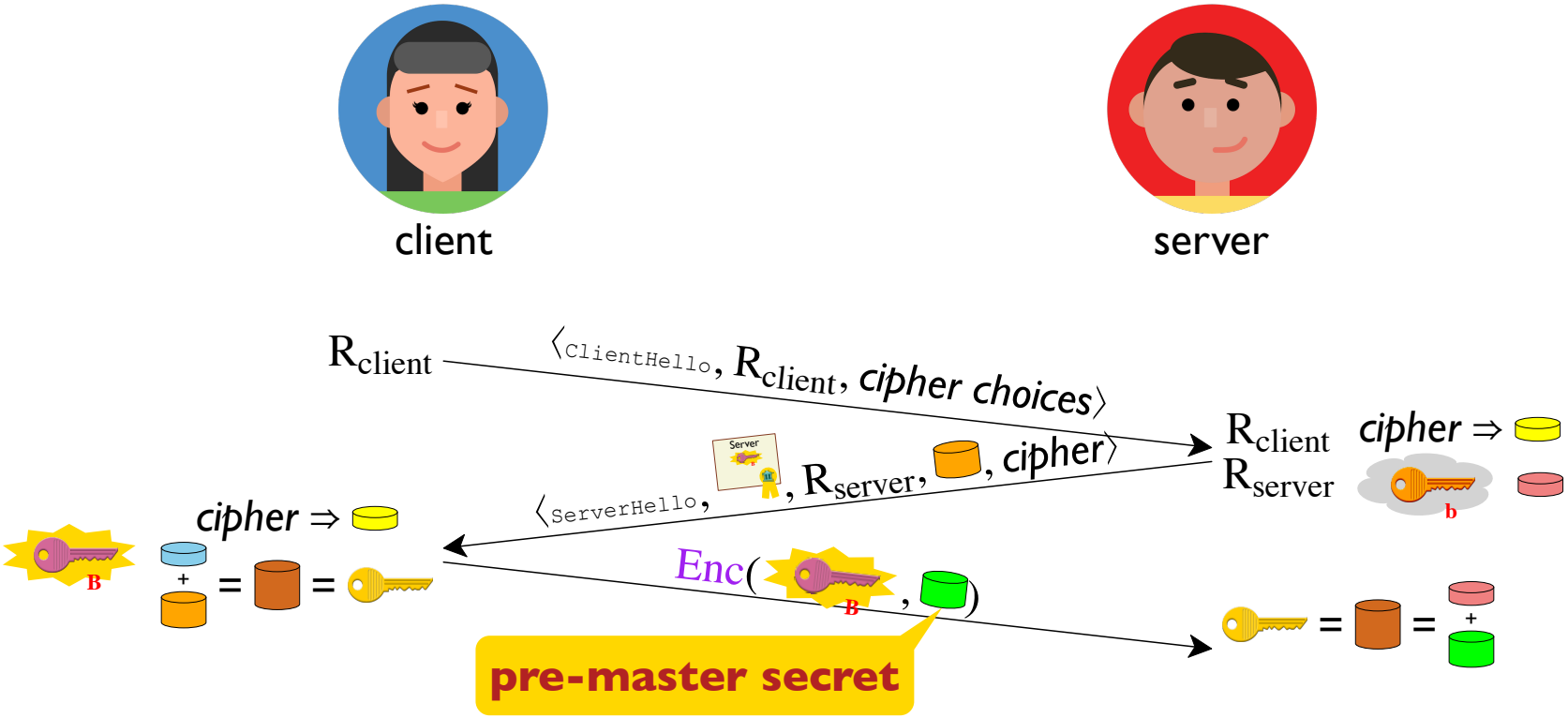


TLS Handshake



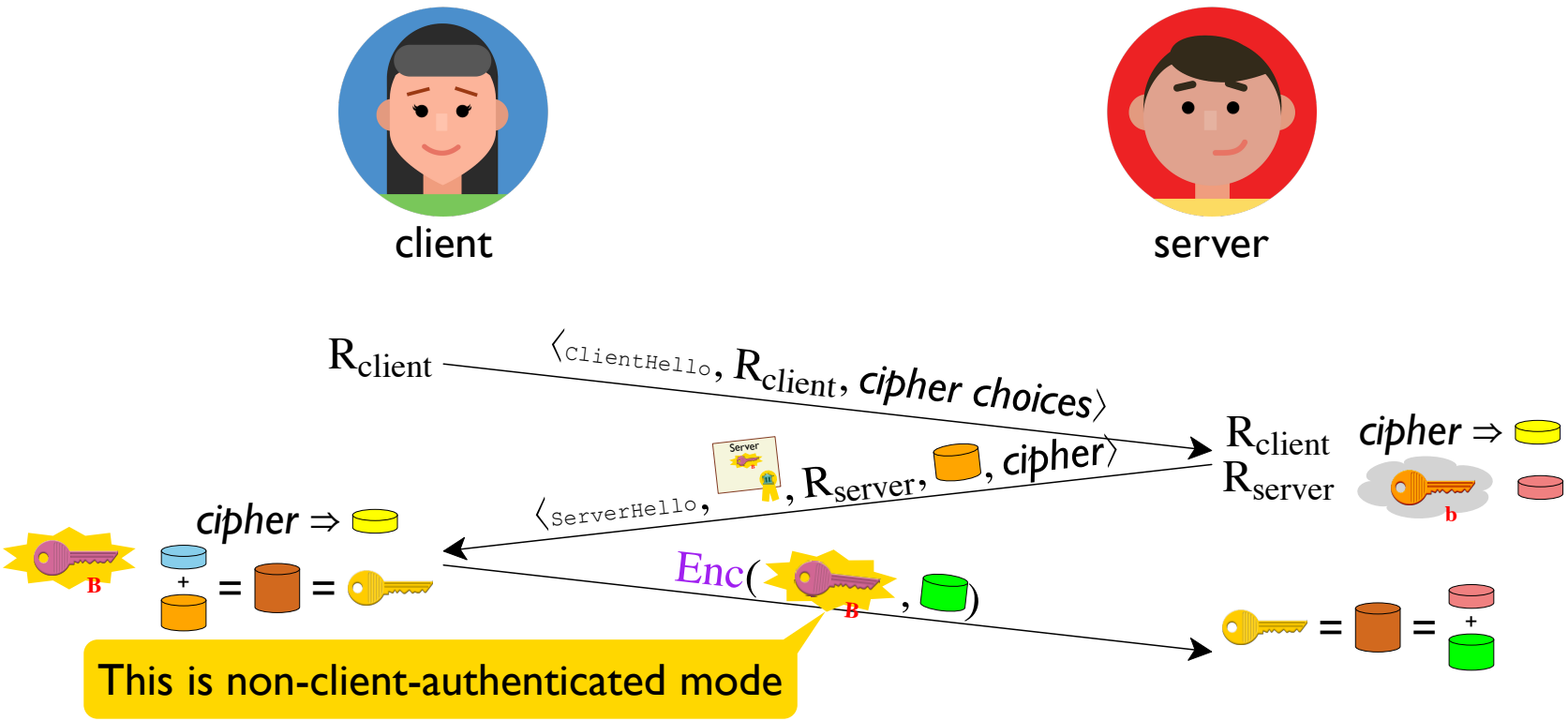
$$\text{key} + R_{client} + R_{server} = \textbf{master secret}$$

TLS Handshake



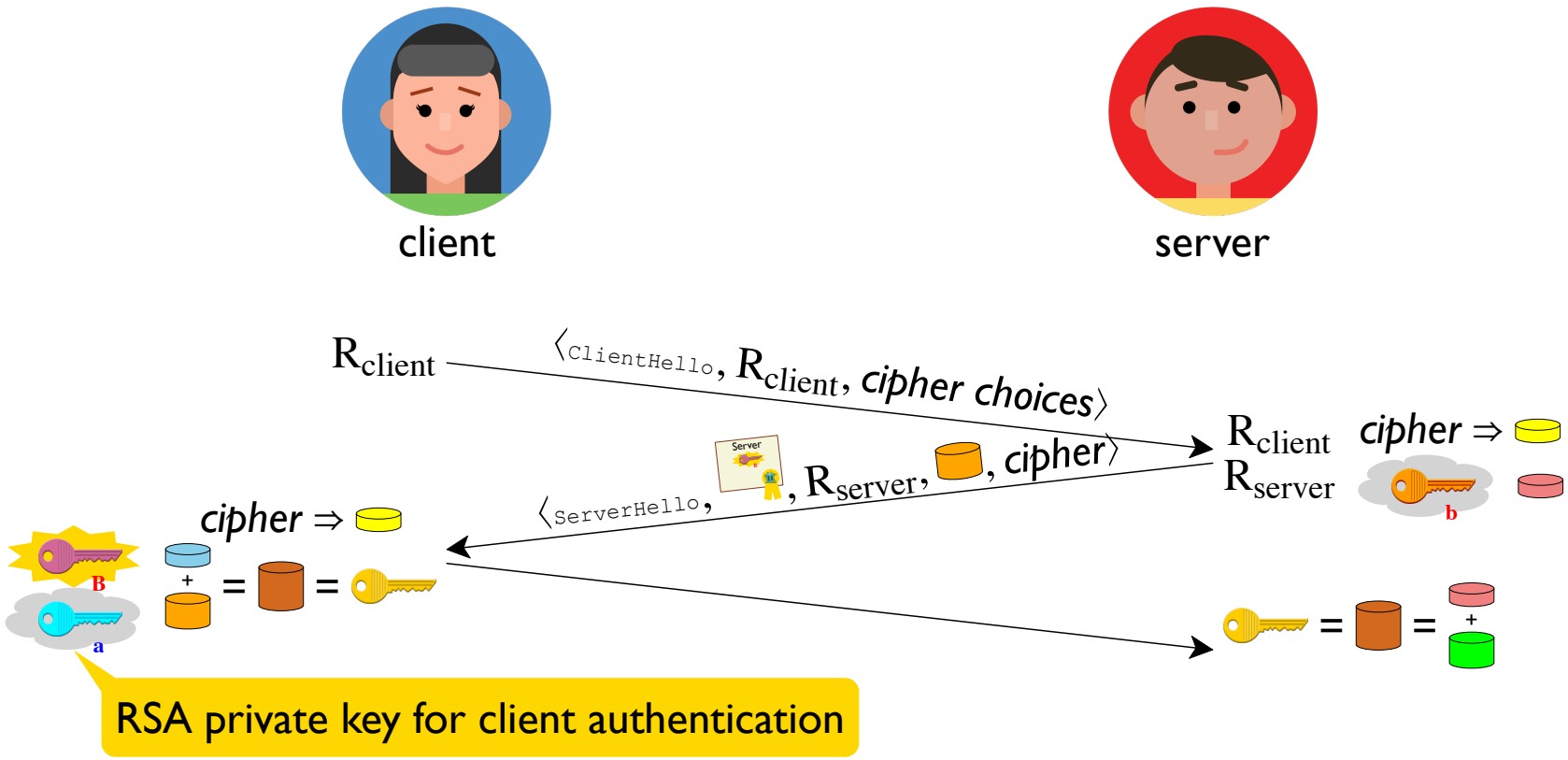
$+ R_{client} + R_{server} =$ **master secret**

TLS Handshake

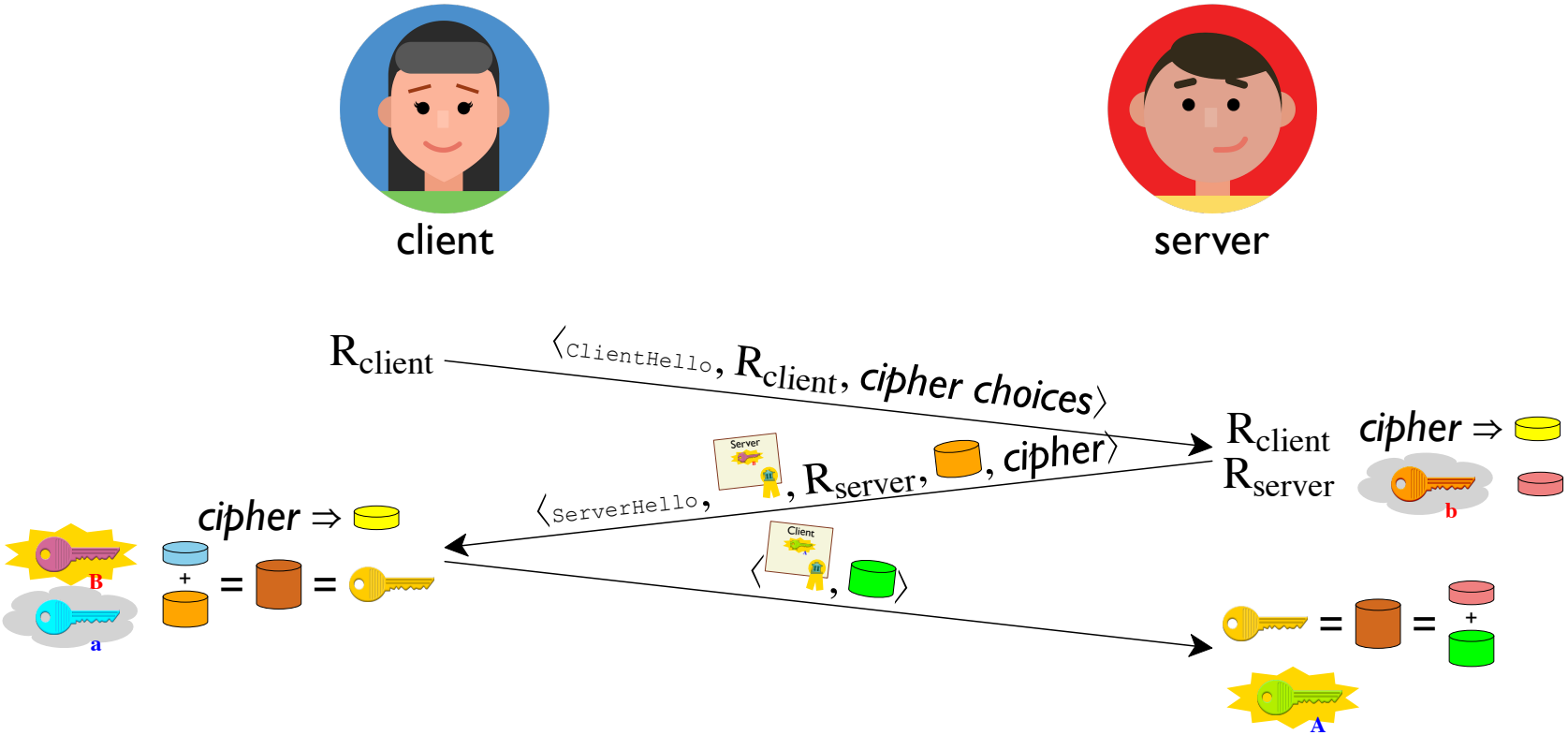


$+ R_{client} + R_{server} =$ **master secret**

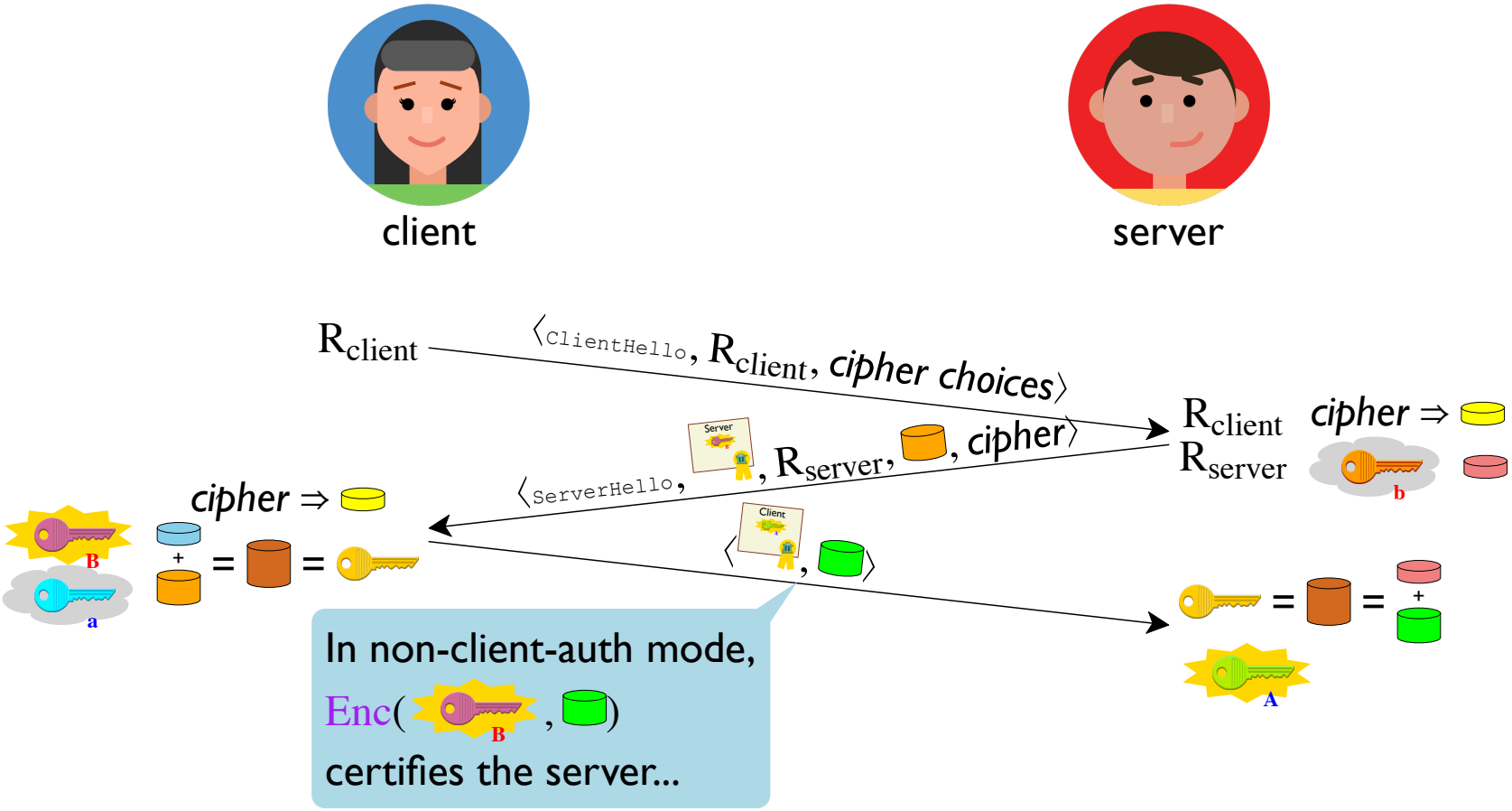
TLS Handshake



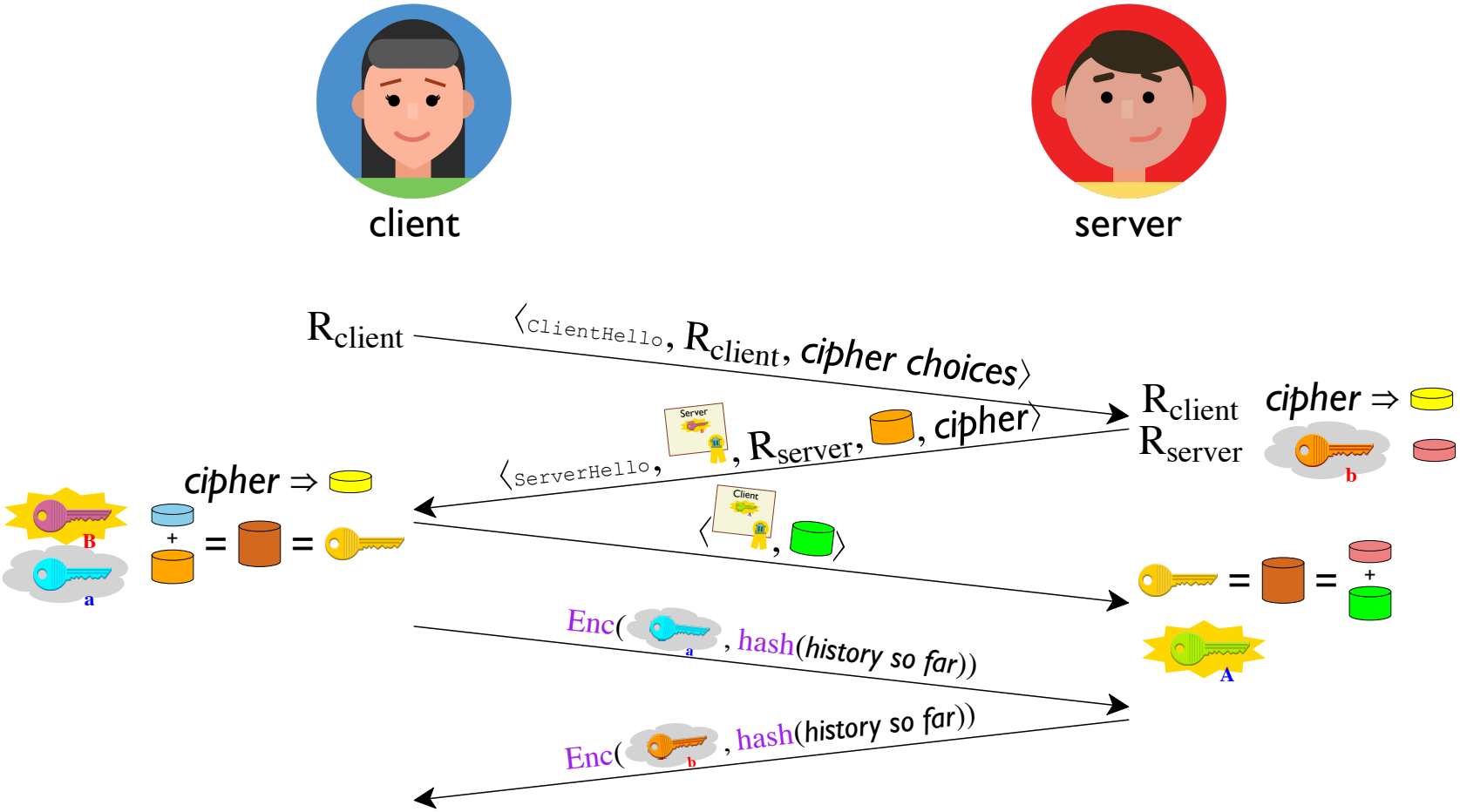
TLS Handshake



TLS Handshake



TLS Handshake



Session Keys

Use the master secret with a **key deivation function (KDF)** for all **session keys**:

- client-to-server encryption
- server-to-client encryption
- initialization vectors for CBC init vector, one for each direction
- MAC keys and , one for each direction
- ...

Exact set of session keys depends on the selected cipher suite

Typical KDF is HMAC(, *description string*), which is known as **HKDF**

e.g, HMAC(, `"client encrypt"`)

TLS Handshake Finish

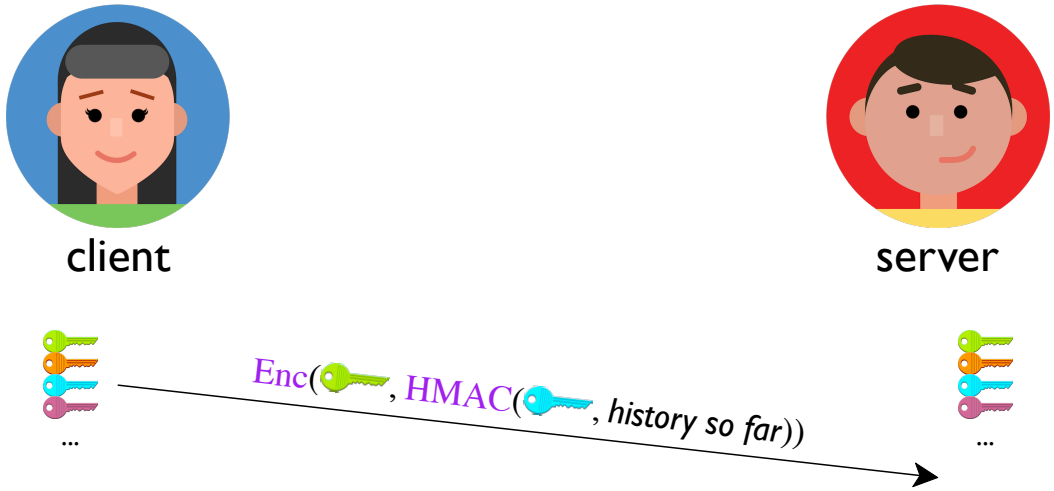
client

...

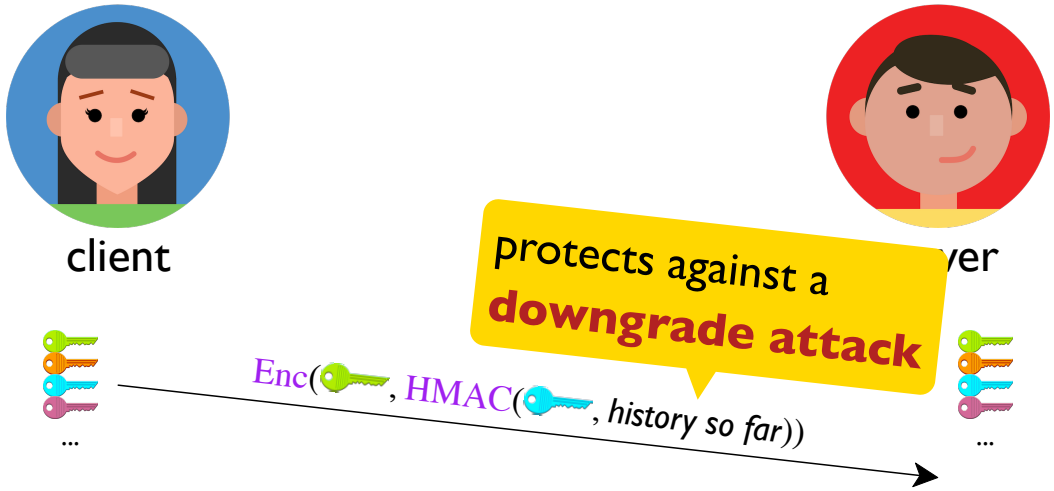
server

...

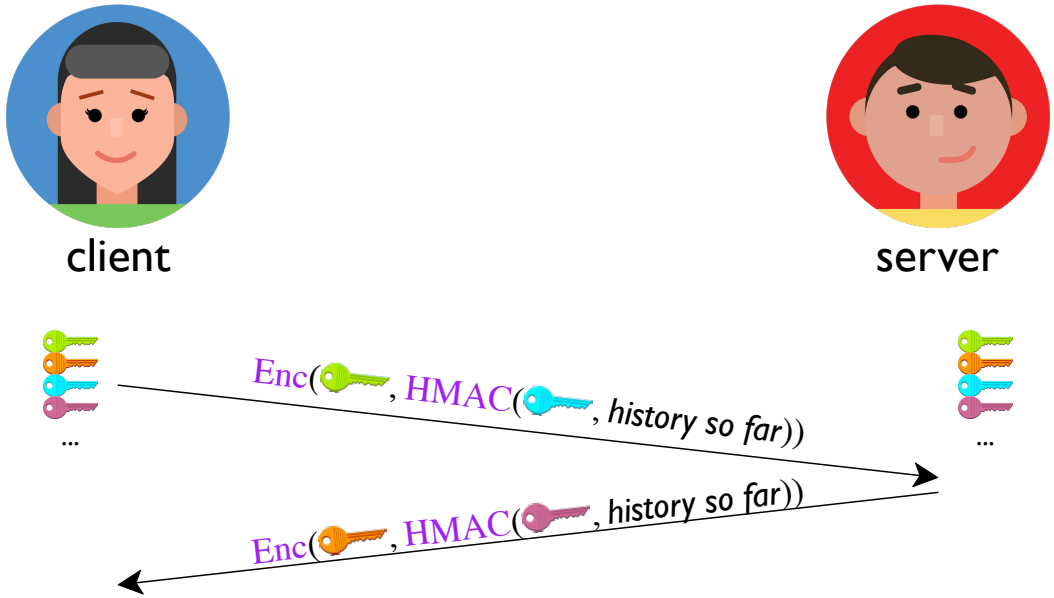
TLS Handshake Finish



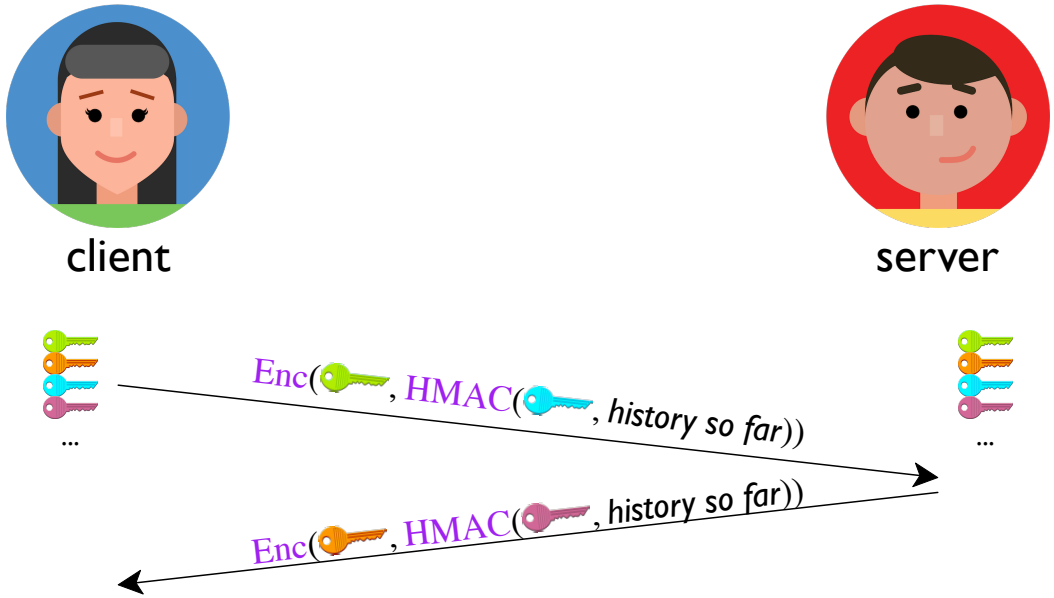
TLS Handshake Finish



TLS Handshake Finish



TLS Handshake Finish



Tagged as `ChangeCipherSpec` instead of `Handshake` messages

Needed for both authenticated and non-authenticated client modes

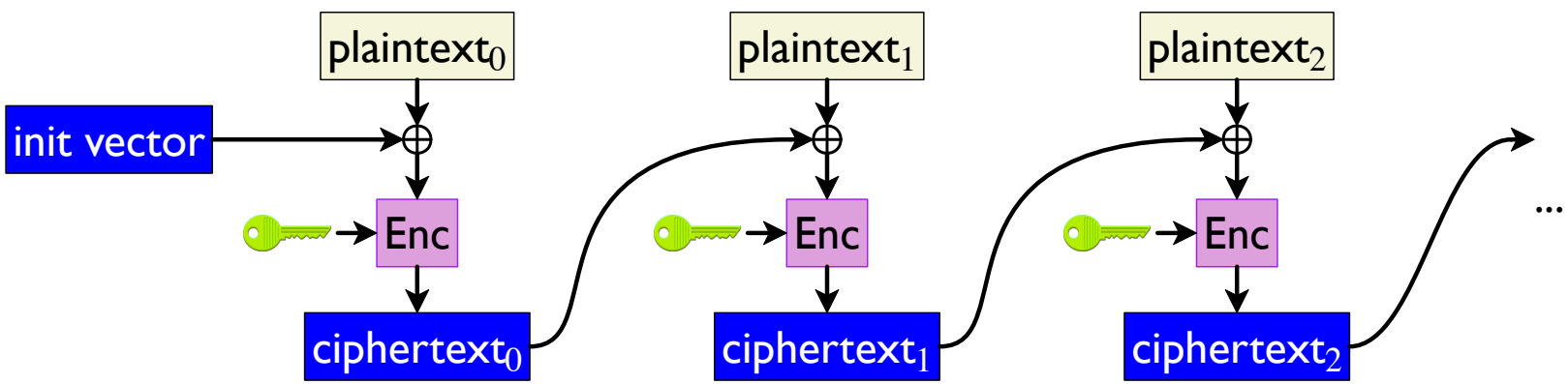
TLS Post-Handshake Packets

IP	TCP	TLS	
src:	src port:	type: ...	$\mathrm{Enc}_{\mathrm{CBC}}(\text{🔑}, \langle \boxed{\text{plaintext}}, \mathrm{HMAC}(\text{🔑}, \boxed{\text{plaintext}})\rangle)$
dest:	dest port:	version:	

TLS Post-Handshake Packets

IP	TCP	TLS	
src:	src port:	type: ...	$\mathrm{Enc}_{\mathrm{CBC}}(\text{🔑}, \langle \boxed{\text{plaintext}}, \mathrm{HMAC}(\text{🔑}, \boxed{\text{plaintext}} + \mathit{seqno})\rangle)$
dest:	dest port:	version:	

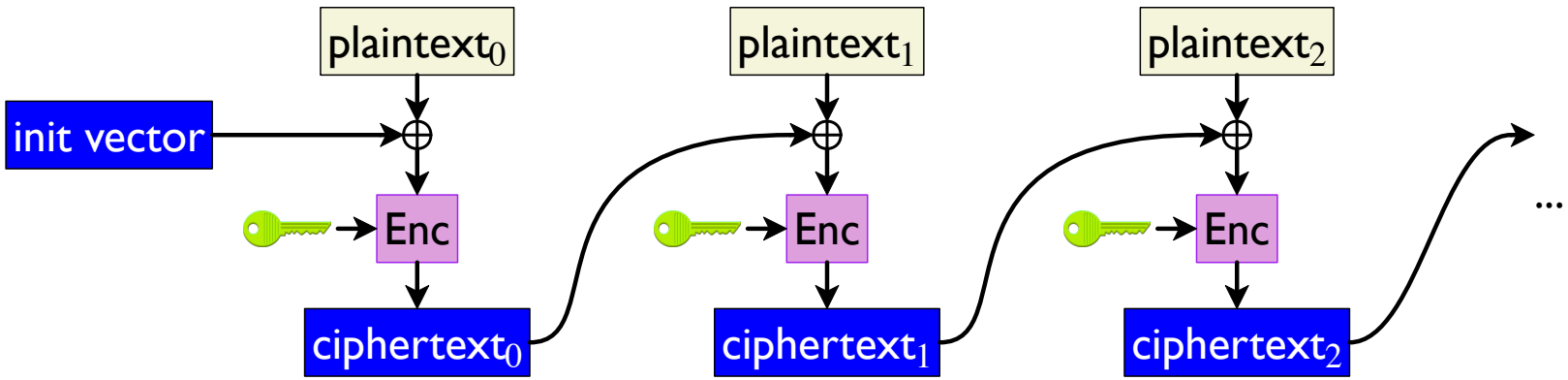
$\mathrm{Enc}_{\mathrm{CBC}}$ uses cipher-block chaining:



TLS Post-Handshake Packets

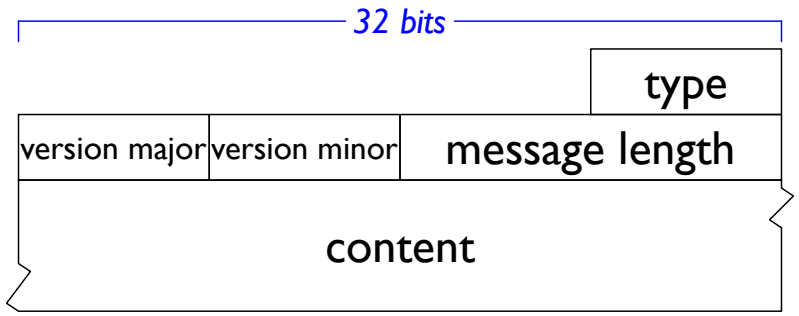
IP	TCP	TLS	
src:	src port:	type: ...	$\mathrm{Enc}_{\mathrm{CBC}}(\text{🔑}, \langle \boxed{\text{plaintext}}, \mathrm{HMAC}(\text{🔑}, \boxed{\text{plaintext}} + \mathit{seqno})\rangle)$
dest:	dest port:	version:	

$\mathrm{Enc}_{\mathrm{CBC}}$ uses cipher-block chaining:

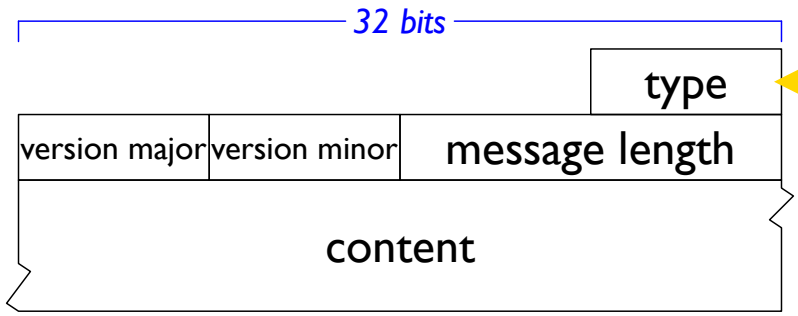


In Java, $\mathrm{Enc}_{\mathrm{CBC}}$ can be implemented by `Cipher.getInstance("AES/CBC/PKCS5Padding")` as initialized with 🔑 and init vector

TLS Record

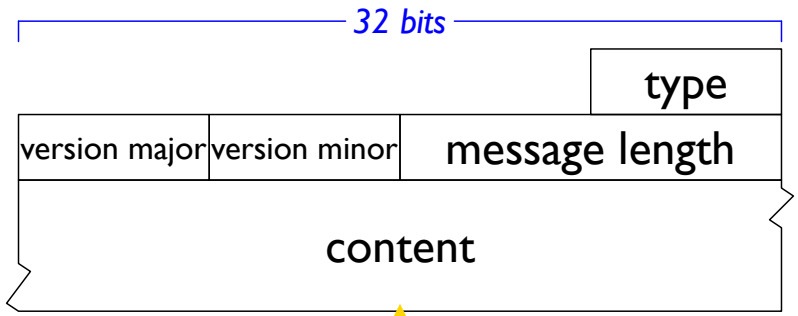

32 bits
type
version major
version minor
message length
content

TLS Record



22 = Handshake
20 = ChangeCipherSpec
23 = Application

TLS Record



For `Application` type, typically something like

$\mathrm{Enc}_{\mathrm{CBC}}(\text{🔑}, \langle \boxed{\text{plaintext}}, \mathrm{HMAC}(\text{🔑}, \boxed{\text{plaintext}} + \mathit{seqno})\rangle)$,

but it depends on the chosen cipher suite

TLS History

version	*year*	*status*
SSL 1.0	1995	flawed
SSL 2.0	1995	flawed
SSL 3.0	1996	deprecated (2015)
TLS 1.0	1999	deprecated (2021)
TLS 1.1	2006	deprecated (2021)
TLS 1.2	2008	discouraged
TLS 1.3	2018	current

Summary

Transport Layer Security (TLS): common vehicle for encrypted data streams

a.k.a. **Secure Socket Layer (SSL)**

TLS uses certificates for at least the server, optionally the client

TLS encrypts data, but not the packet headers