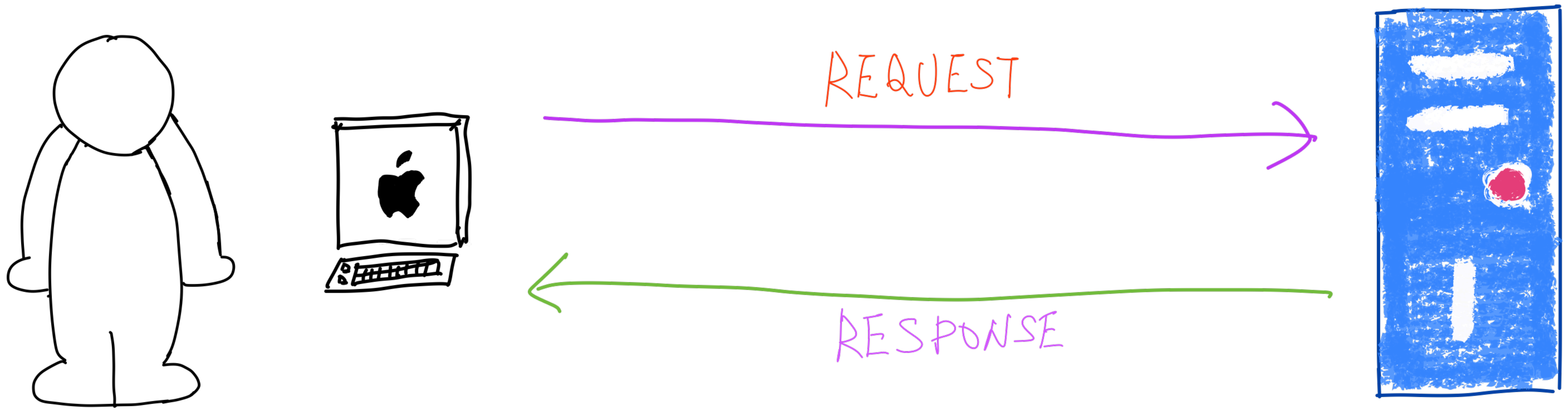
SSL Handshake

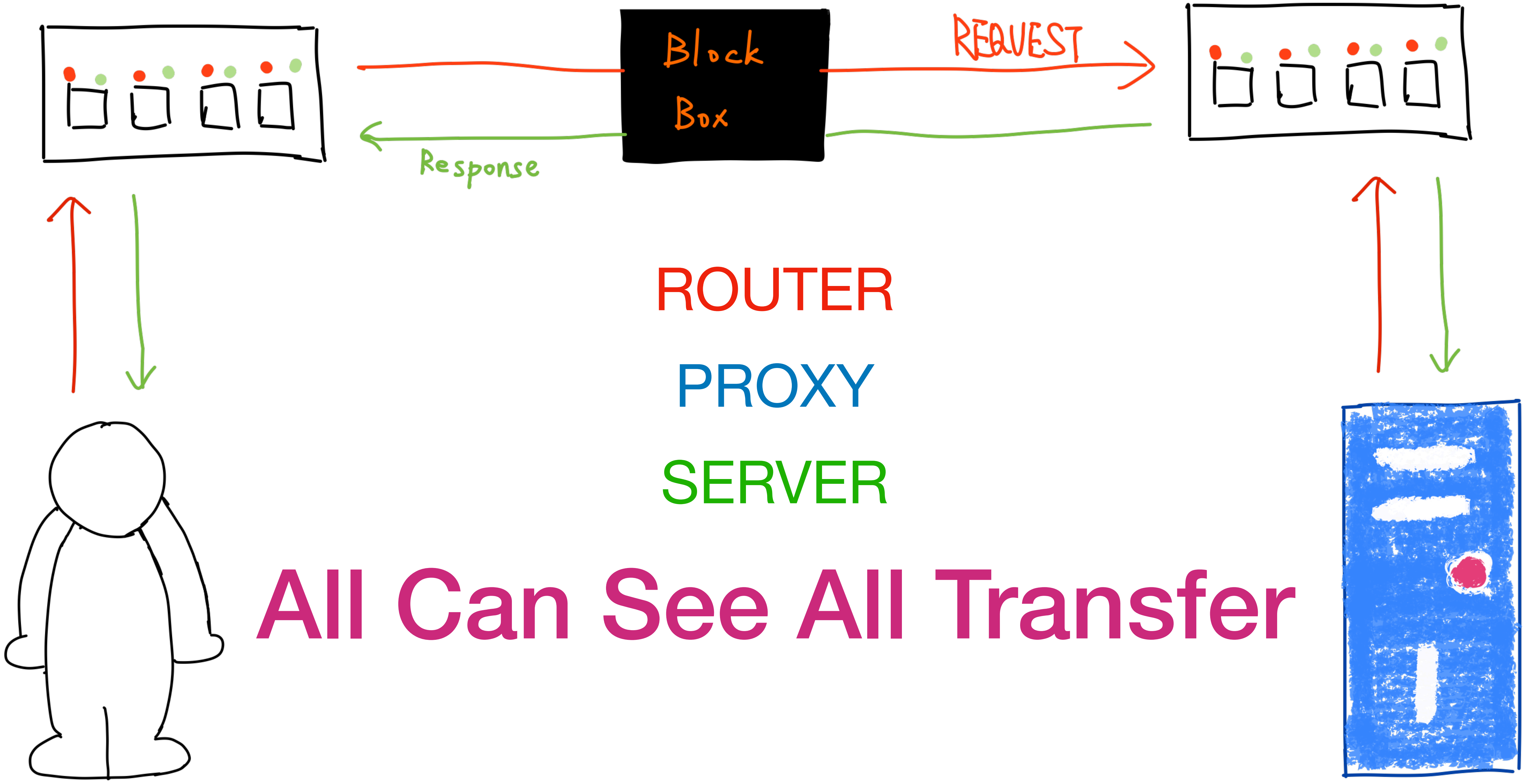
iOS@Taipei
2019-07-16
Nick Lin

什麼是網路?

網路看起來是這個樣子



事實上是這樣



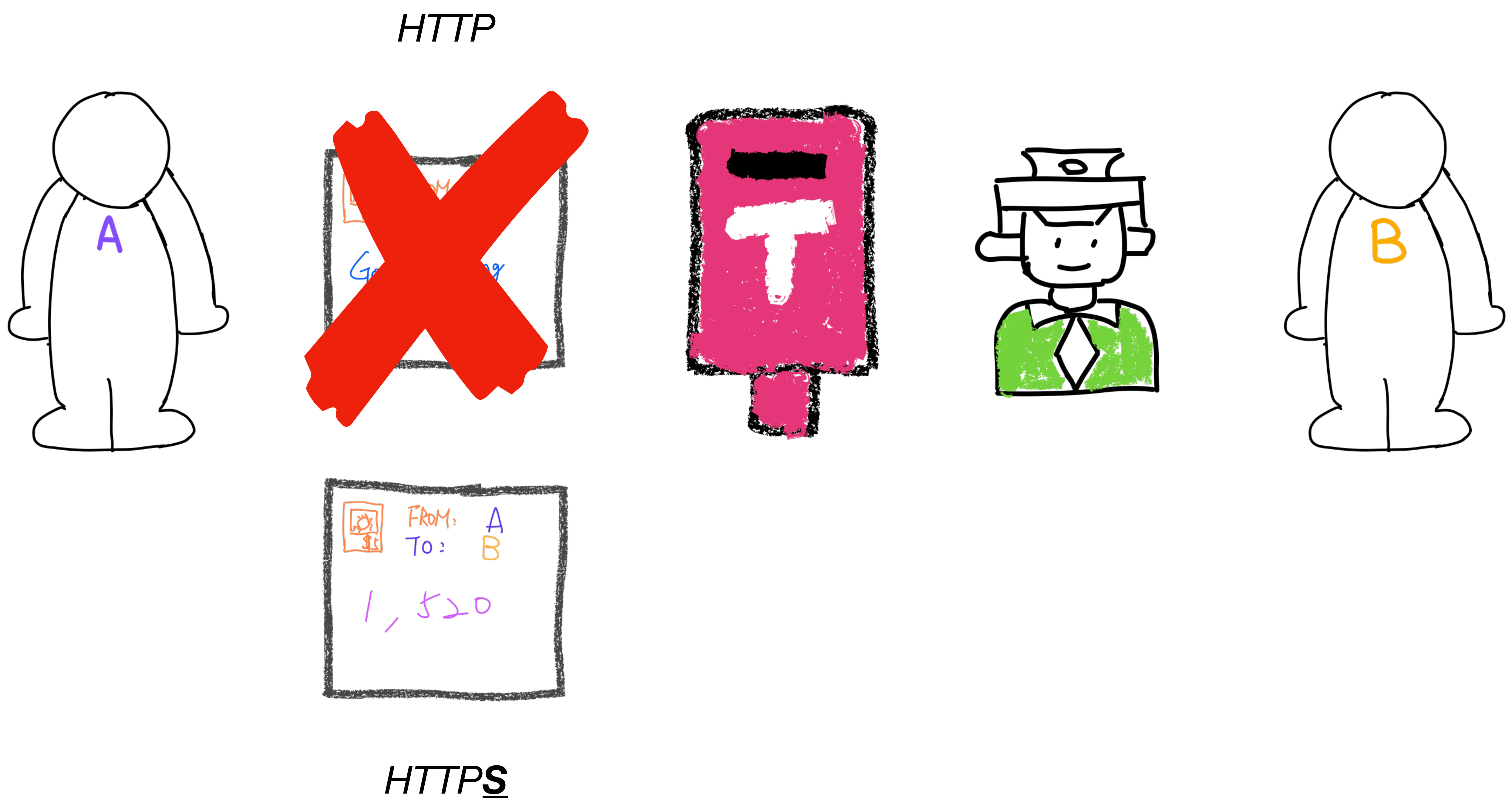
ROUTER

PROXY

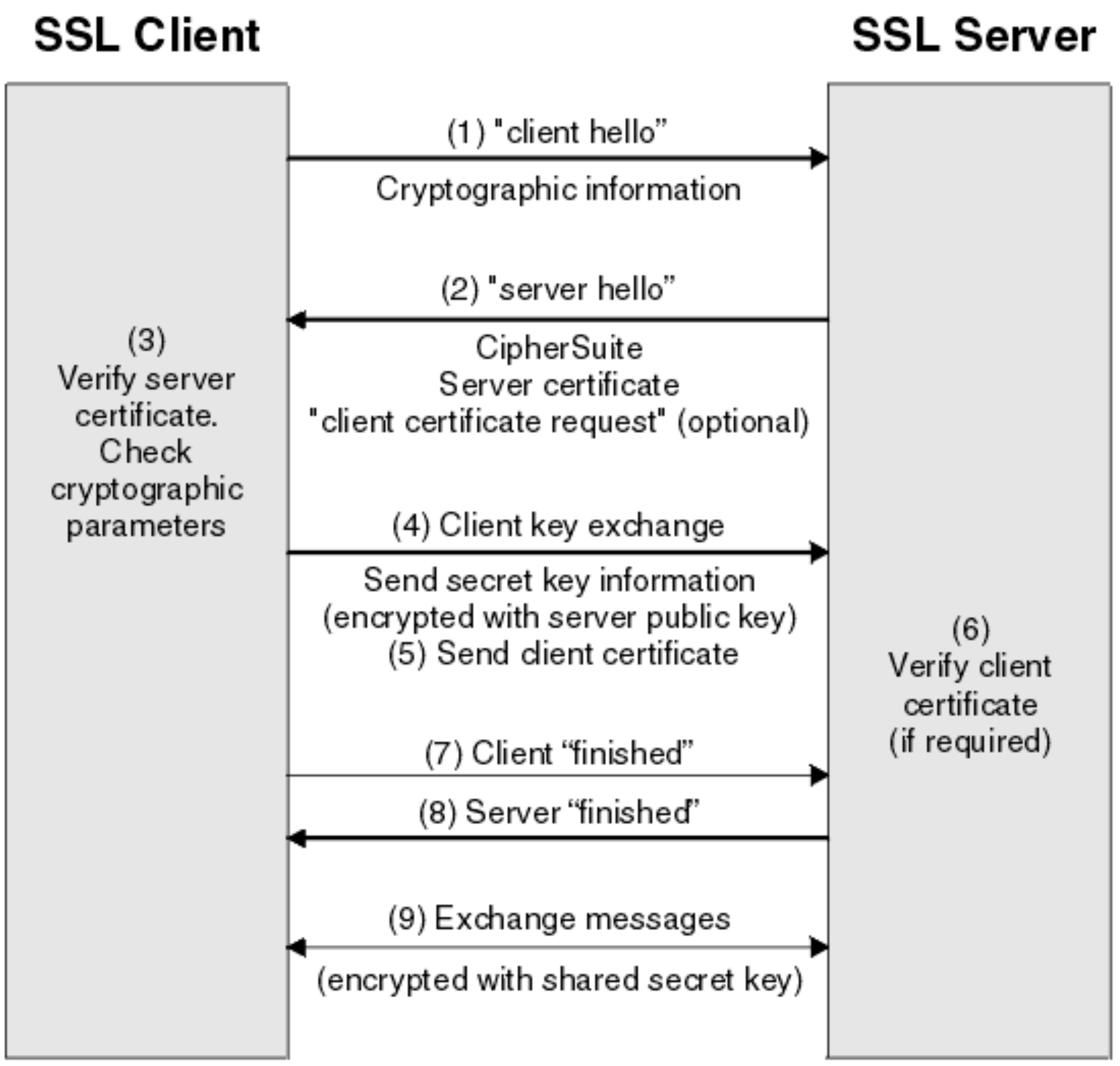
SERVER

All Can See All Transfer

舉個 🌰



SSL Handshake Flow

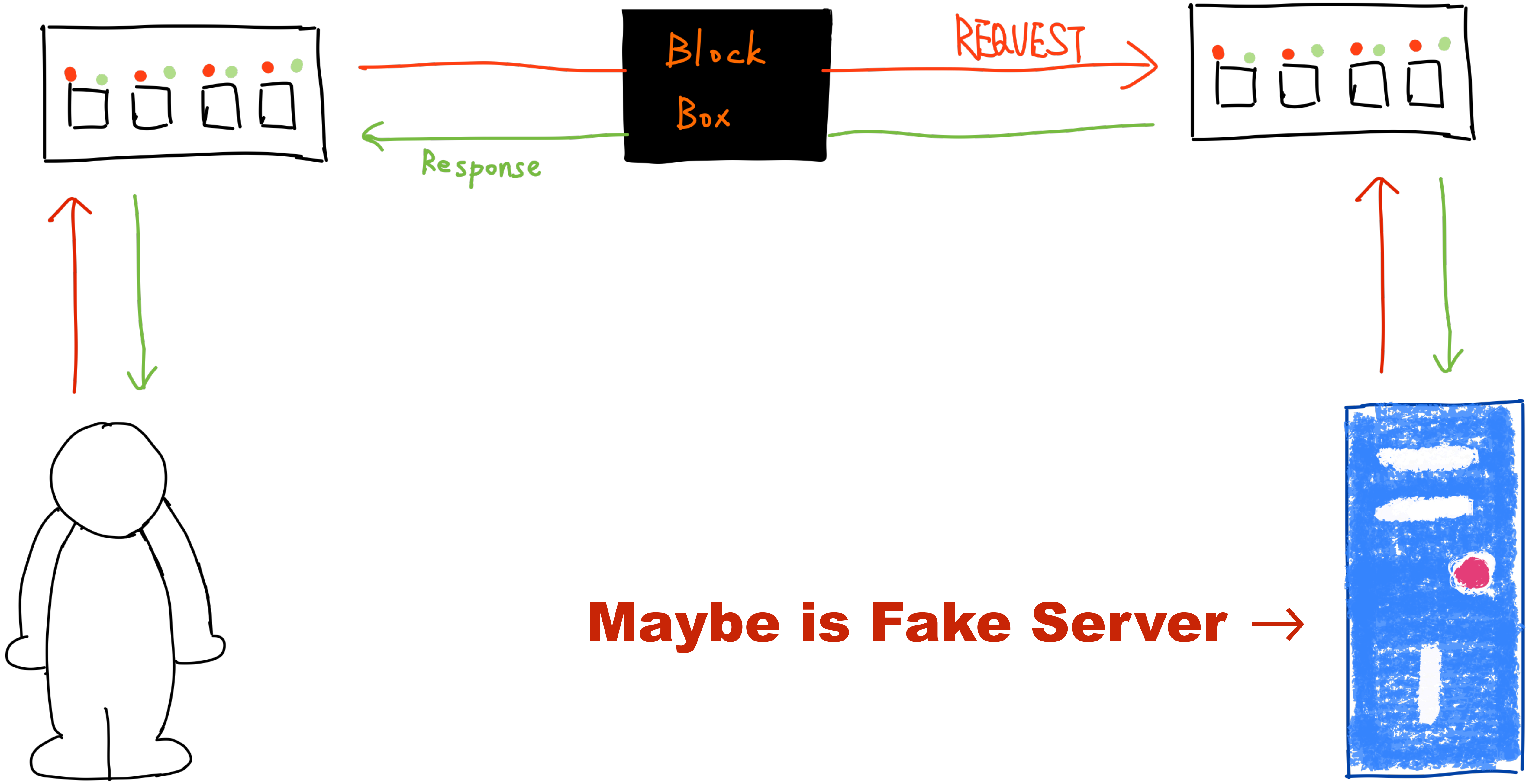

SSL Client
SSL Server
(1) "client hello"
Cryptographic information
(2) "server hello"
CipherSuite
Server certificate
"client certificate request" (optional)
(3)
Verify server certificate.
Check cryptographic parameters
(4) Client key exchange
Send secret key information
(encrypted with server public key)
(5) Send client certificate
(6)
Verify client certificate
(if required)
(7) Client "finished"
(8) Server "finished"
(9) Exchange messages
(encrypted with shared secret key)

圖片來源：*https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm*

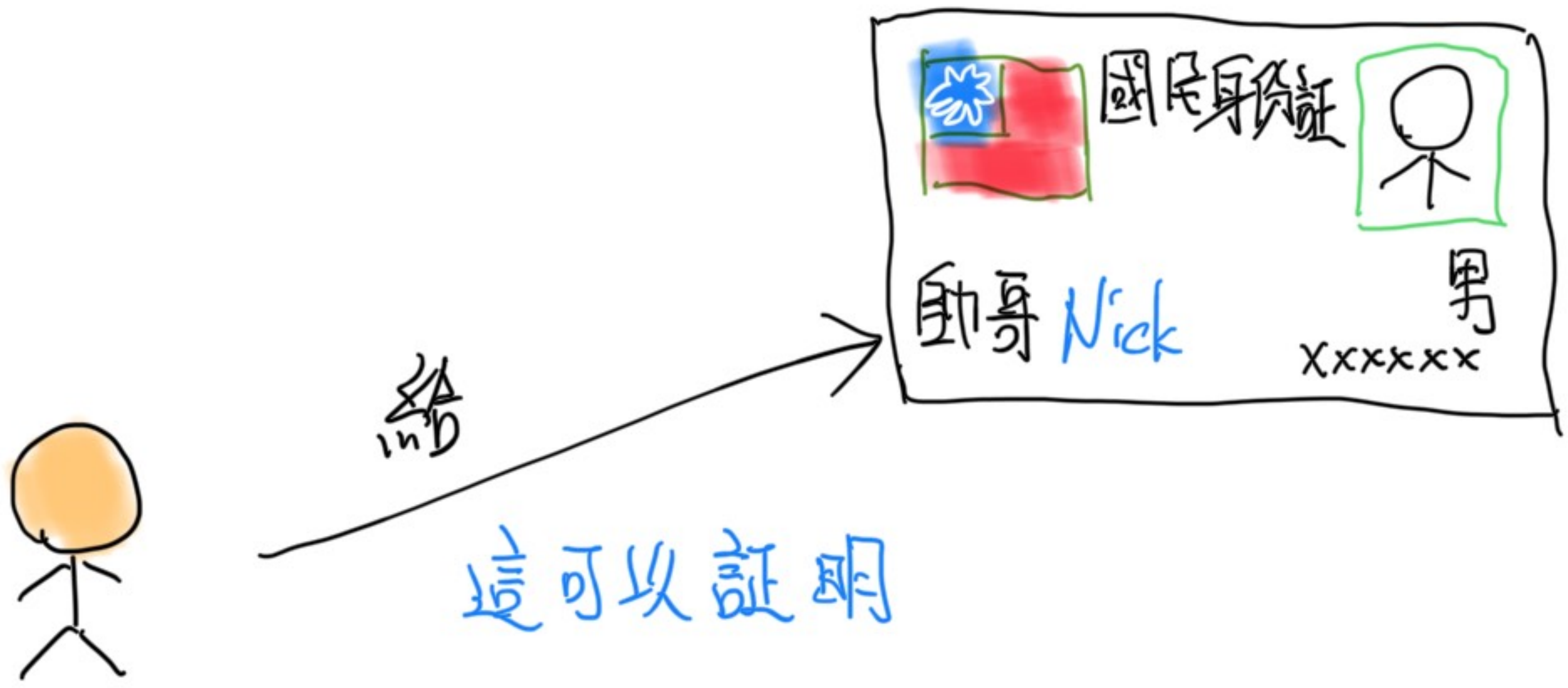
This section does not attempt to provide full details of the messages exchanged during the SSL handshake. In overview, the steps involved in the SSL handshake are as follows:

- The SSL or TLS client sends a “client hello” message that lists cryptographic information such as the SSL or TLS version and, in the client's order of preference, the CipherSuites supported by the client. The message also contains a random byte string that is used in subsequent computations. The protocol allows for the “client hello” to include the data compression methods supported by the client.
- The SSL or TLS server responds with a “server hello” message that contains the CipherSuite chosen by the server from the list provided by the client, the session ID, and another random byte string. The server also sends its digital certificate. If the server requires a digital certificate for client authentication, the server sends a “client certificate request” that includes a list of the types of certificates supported and the Distinguished Names of acceptable Certification Authorities (CAs).
- The SSL or TLS client verifies the server's digital certificate. For more information, see How SSL and TLS provide identification, authentication, confidentiality, and integrity.
- The SSL or TLS client sends the random byte string that enables both the client and the server to compute the secret key to be used for encrypting subsequent message data. The random byte string itself is encrypted with the server's public key.
- If the SSL or TLS server sent a “client certificate request”, the client sends a random byte string encrypted with the client's private key, together with the client's digital certificate, or a “no digital certificate alert”. This alert is only a warning, but with some implementations the handshake fails if client authentication is mandatory.
- The SSL or TLS server verifies the client's certificate. For more information, see How SSL and TLS provide identification, authentication, confidentiality, and integrity.
- The SSL or TLS client sends the server a “finished” message, which is encrypted with the secret key, indicating that the client part of the handshake is complete.
- The SSL or TLS server sends the client a “finished” message, which is encrypted with the secret key, indicating that the server part of the handshake is complete.
- For the duration of the SSL or TLS session, the server and client can now exchange messages that are symmetrically encrypted with the shared secret key.

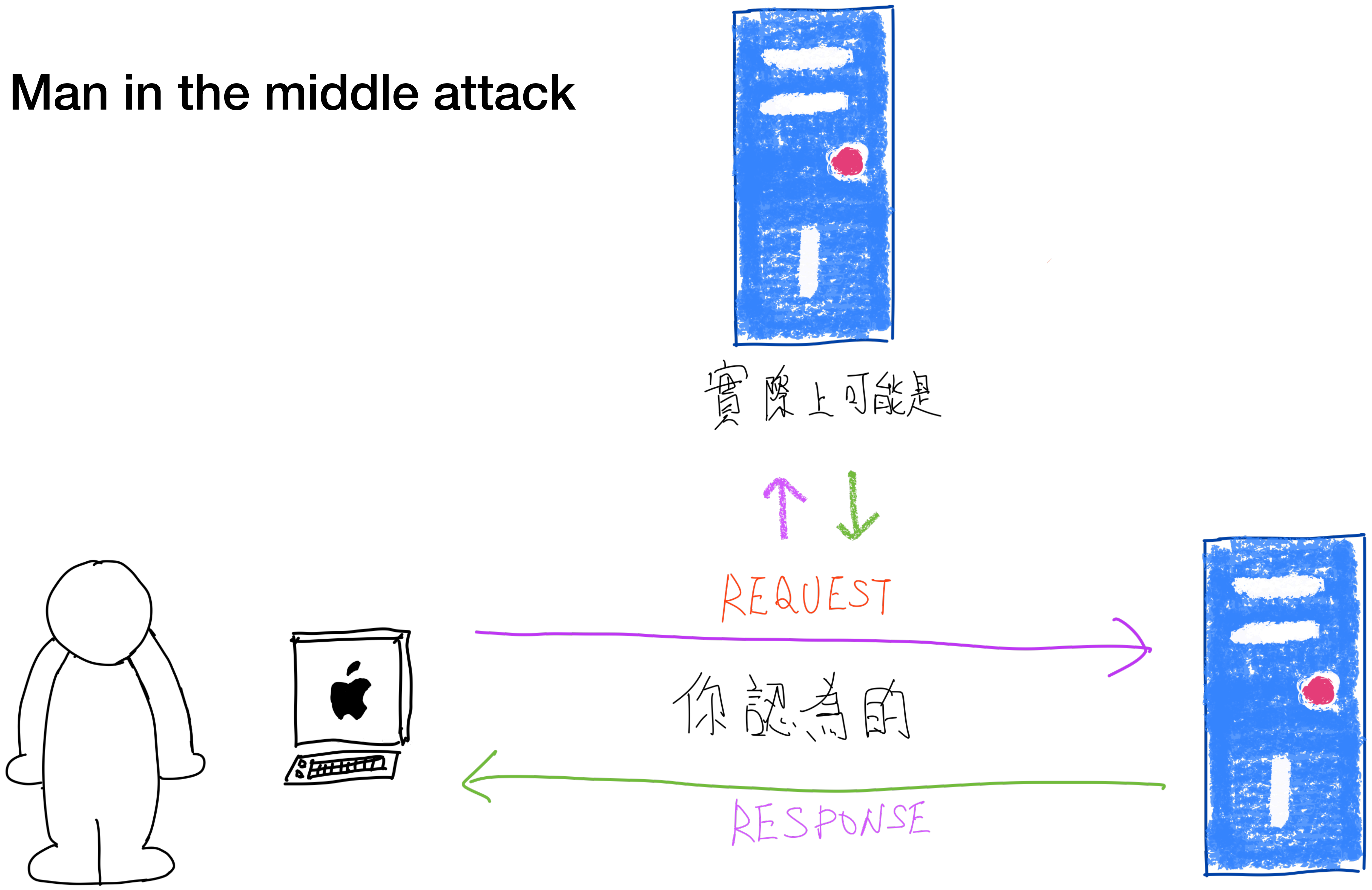
資料來源：*https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/sy10660_.htm*

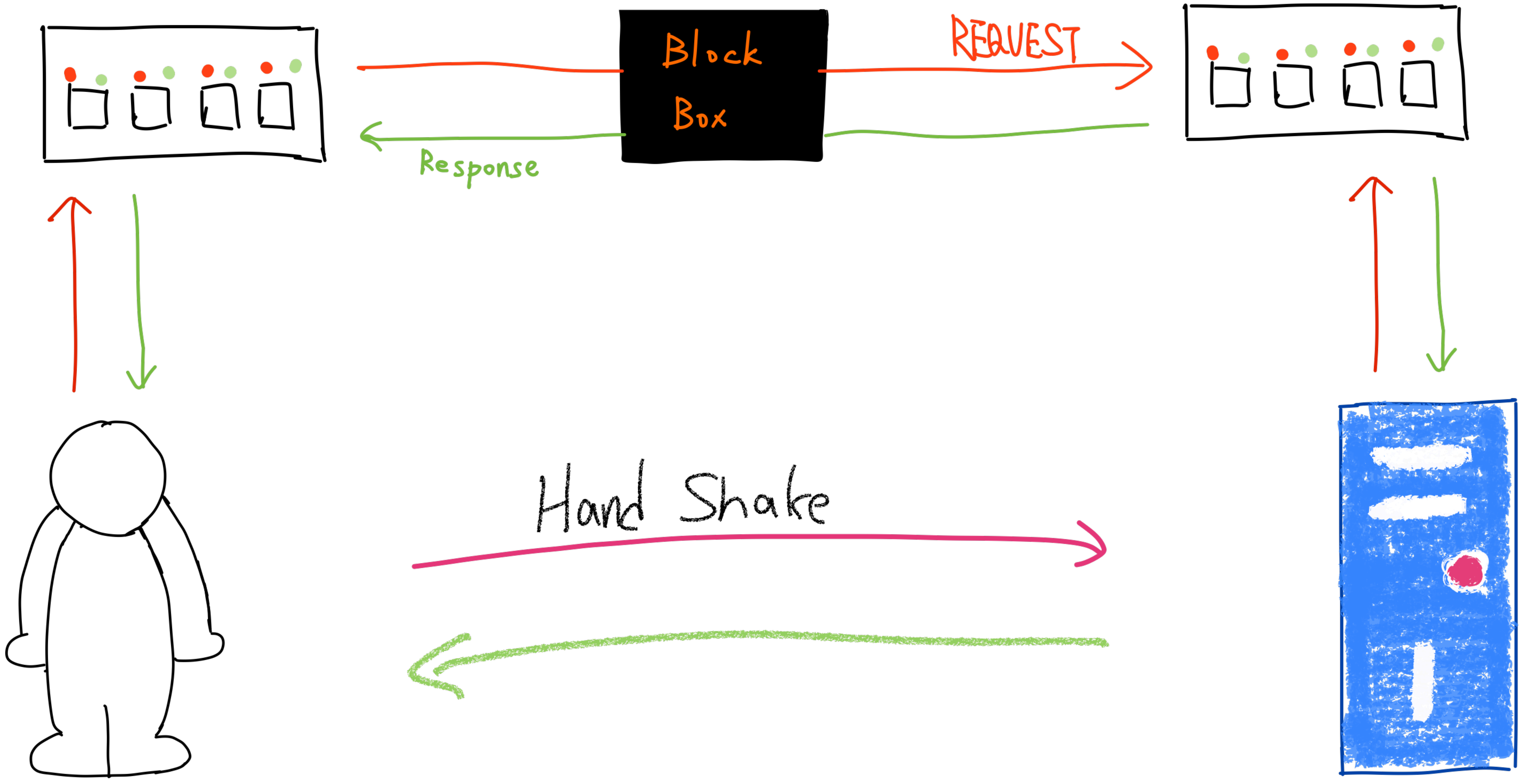


Certificate Authority


國民身份証
帥哥 Nick
男
XXXXXX
給
這可以証明

Man in the middle attack




Block Box
REQUEST
Response
Hand Shake

怎樣確保是對的那台伺服器

1. 驗證 CA 是否是正確配發
2. 驗證 CA 是否是正確的
    1. 比對檔案相同
    2. 比對 Hash 值相同

要在哪裡驗

URLSessionDelegate

```swift
public func urlSession(_ session: URLSession,
                       didReceive challenge: URLAuthenticationChallenge,
                       completionHandler: @escaping (URLSession.AuthChallengeDisposition, URLCredential?) -> Void) {

    guard let serverTrust = challenge.protectionSpace.serverTrust else {
        completionHandler(.cancelAuthenticationChallenge, nil)
        return
    }

    let shaDic: [String: String] = [:]

    let trust: (_ serverTrust: SecTrust, _ host: String) -> Bool = { serverTrust, host -> Bool in
        var trust: Bool = false
        let policy = SecPolicyCreateSSL(true, host as CFString)
        SecTrustSetPolicies(serverTrust, policy)
        var result = SecTrustResultType.invalid
        let status = SecTrustEvaluate(serverTrust, &result)
        if status == errSecSuccess {
            let unspecified = SecTrustResultType.unspecified
            let proceed = SecTrustResultType.proceed
            trust = result == unspecified || result == proceed
        }
        guard trust,
            let localeSha1 = shaDic[host],
            let certificate = SecTrustGetCertificateAtIndex(serverTrust, 0) else {
                return trust
        }
        let remoteCertificateData = SecCertificateCopyData(certificate) as Data
        let remoteSha1 = remoteCertificateData.sha1
        return remoteSha1 == localeSha1
    }

    if trust(serverTrust, challenge.protectionSpace.host) {
        let credential: URLCredential = URLCredential(trust: serverTrust)
        completionHandler(.useCredential, credential)
    } else {
        completionHandler(.cancelAuthenticationChallenge, nil)
    }
}
```

Q & A