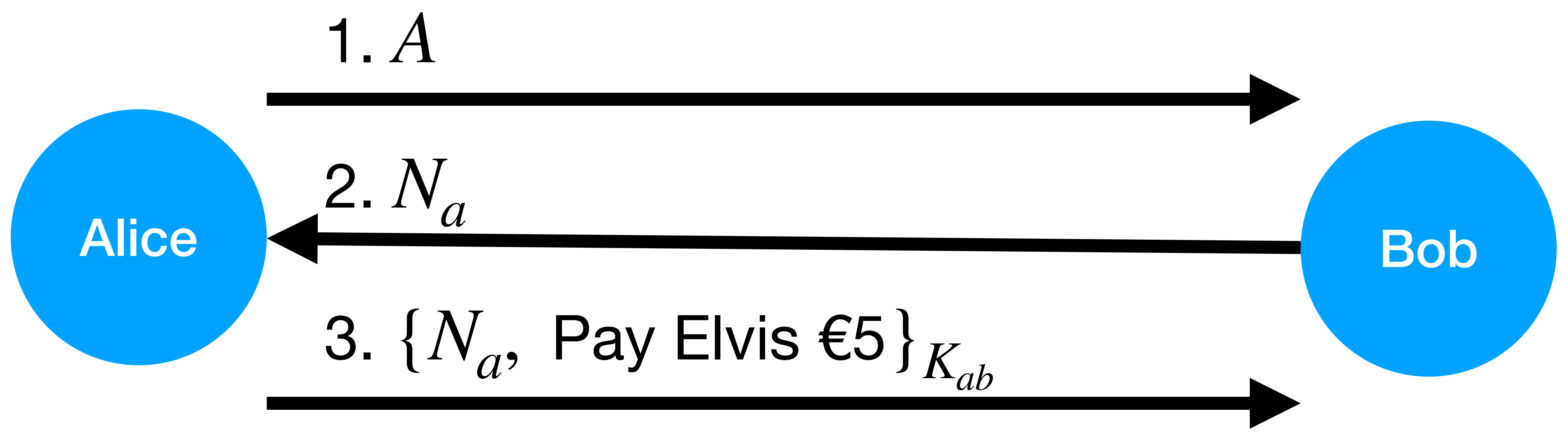
The TLS & Tor Protocols

Security & Networks

Today's Lecture

- Details of TLS
    - How it works
    - Common problems
- Tor
    - Anonymity on the internet
    - The Tor Protocol
    - Hidden servers

Last Lecture



1. $A \rightarrow B : A$

2. $B \rightarrow A : N_a$

3. $A \rightarrow B : \{N_a,$ Pay Elvis €5$\}_{K_{ab}}$

$E_X(\_)$ means public key encryption

The Needham-Schroeder Public Key Protocol

Assume Alice and Bob know each others public keys, can they set up a symmetric key?

1. $A \rightarrow B : E_B(N_a, A)$
2. $B \rightarrow A : E_A(N_a, N_b)$
3. $A \rightarrow B : E_B(N_b)$

N_a and N_b can then be used to generate a symmetric key.

Goals: Alice and Bob are sure they are talking to each other and only they know the key.

An Attack Against the NH Protocol

The attacker C acts as a man-in-the-middle:

1. $A \rightarrow C : E_C(N_a, A)$

    1) $C(A) \rightarrow B : E_B(N_a, A)$

    2) $B \rightarrow C(A) : E_A(N_a, N_b)$

2. $C \rightarrow A : E_A(N_a, N_b)$

3. $A \rightarrow C : E_C(N_b)$

    3) $C(A) \rightarrow B : E_B(N_b)$

Corrected Version

A very simple fix:

1. $A \rightarrow B : E_B(N_a, A)$

2. $B \rightarrow A : E_A(N_a, N_b, B)$

3. $A \rightarrow B : E_B(N_b)$

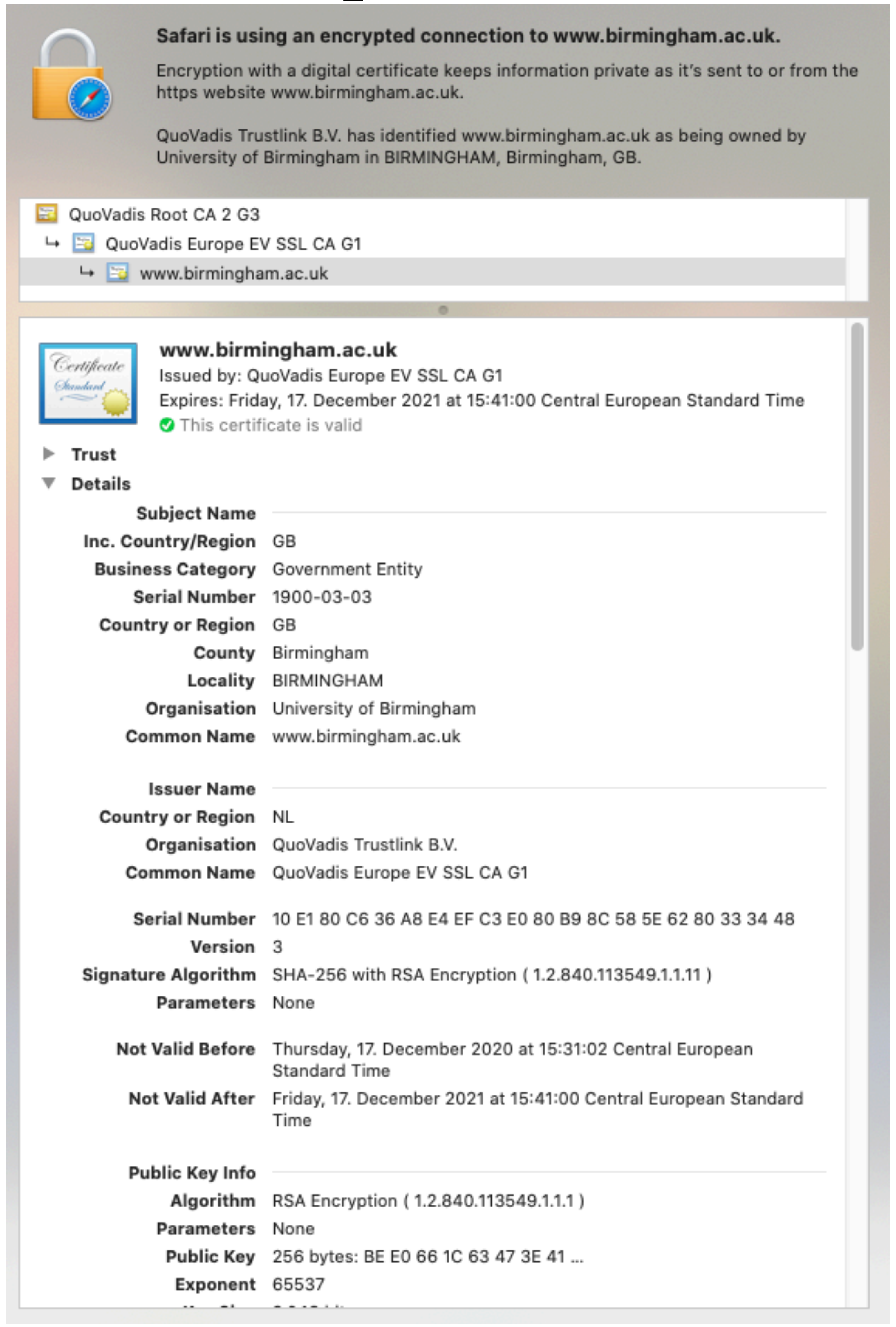
The SSL/TLS Protocol

- The Secure Sockets Layer (SSL) protocol was renamed to Transport Layer Security (TLS) protocol.
- It provides encrypted socket communication and authentication, based on public keys.
- It may use a range of ciphers (RSA, DES, DH,…)
    - These are negotiated at the start of the run.

X.509 Standard for Certificates

- X.509 certificates contain a subject, subject’s public key, issuer name, etc.
- The issuer signs the hash of all the data
- To check a certificate, I hash all the data and check the issuers public key.
- If I have the issuer’s public key and trust the issuer, I can then be sure of the subject’s public key.

Example Cert.



Safari is using an encrypted connection to www.birmingham.ac.uk.

Encryption with a digital certificate keeps information private as it's sent to or from the https website www.birmingham.ac.uk.

QuoVadis Trustlink B.V. has identified www.birmingham.ac.uk as being owned by University of Birmingham in BIRMINGHAM, Birmingham, GB.

QuoVadis Root CA 2 G3
↳ QuoVadis Europe EV SSL CA G1
↳ www.birmingham.ac.uk

www.birmingham.ac.uk
Issued by: QuoVadis Europe EV SSL CA G1
Expires: Friday, 17. December 2021 at 15:41:00 Central European Standard Time
✓ This certificate is valid

▶ **Trust**
▼ **Details**

Subject Name	
Inc. Country/Region	GB
Business Category	Government Entity
Serial Number	1900-03-03
Country or Region	GB
County	Birmingham
Locality	BIRMINGHAM
Organisation	University of Birmingham
Common Name	www.birmingham.ac.uk
Issuer Name	
Country or Region	NL
Organisation	QuoVadis Trustlink B.V.
Common Name	QuoVadis Europe EV SSL CA G1
Serial Number	10 E1 80 C6 36 A8 E4 EF C3 E0 80 B9 8C 58 5E 62 80 33 34 48
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters	None
Not Valid Before	Thursday, 17. December 2020 at 15:31:02 Central European Standard Time
Not Valid After	Friday, 17. December 2021 at 15:41:00 Central European Standard Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	None
Public Key	256 bytes: BE E0 66 1C 63 47 3E 41 ...
Exponent	65537

The Internet Protocol Stack (Most of the Time)

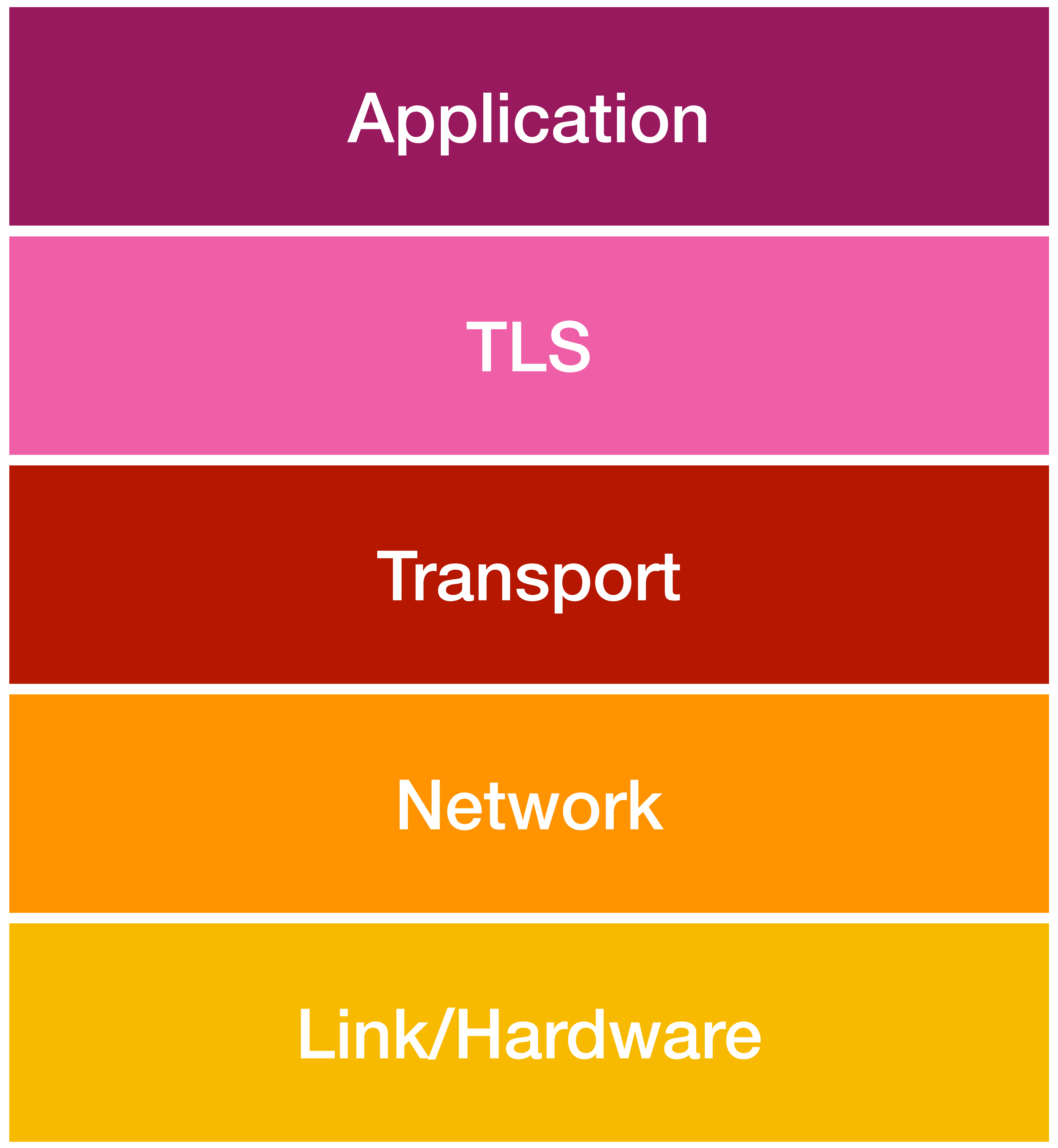
Stuff that you write	Application
TCP or UDP	Transport
IP	Network
Ethernet or 802.11	Link/Hardware

The Internet Protocol Stack with TLS

The TLS layer runs between the Application and Transport layer.

The encryption is transparent to the Application layer.

Normal TCP and IP protocols etc. can be used at the low layers.



TLS

1. $C \to S : N_C$
2. $S \to C : N_S, Cert_S$
3. $C \to S : E_S(K_{\text{seed}}), \{Hash_1\}_{K_{CS}}$
4. $S \to C : \{Hash_2\}_{K_{CS}}$

$$Hash_1 = \#(N_C, N_S, E_S(K_{\text{seed}}))$$

$$Hash_2 = \#(N_C, N_S, E_S(K_{\text{seed}}), \{Hash_1\}_{K_{CS}})$$

All previous messages are hashed and then encrypted with K_{CS} for integrity.

K_{CS} is a session key based on $N_C, N_S, K_{\text{seed}}$

TLS-DHE

A variant uses Diffie-Hellman for *forward secrecy*

i.e., if someone gets the server's key later, they can't go back and break a recording of the traffic.

1. $C \to S : N_C$
2. $S \to C : N_S, g^x, Cert_S, Sign_S(\#(N_C, N_S, g^x))$
3. $C \to S : g^y, \{\#(\text{All previous messages})\}_{K_{CS}}$
4. $S \to C : \{\#(\text{All previous messages})\}_{K_{CS}}$

K_{CS} is a session key based on N_C, N_S, g^{xy}.

Cipher Suites

Cipher Suites with encryption and authentication:

```
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
...
```

Cipher Suites with just authentication:

```
SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
```

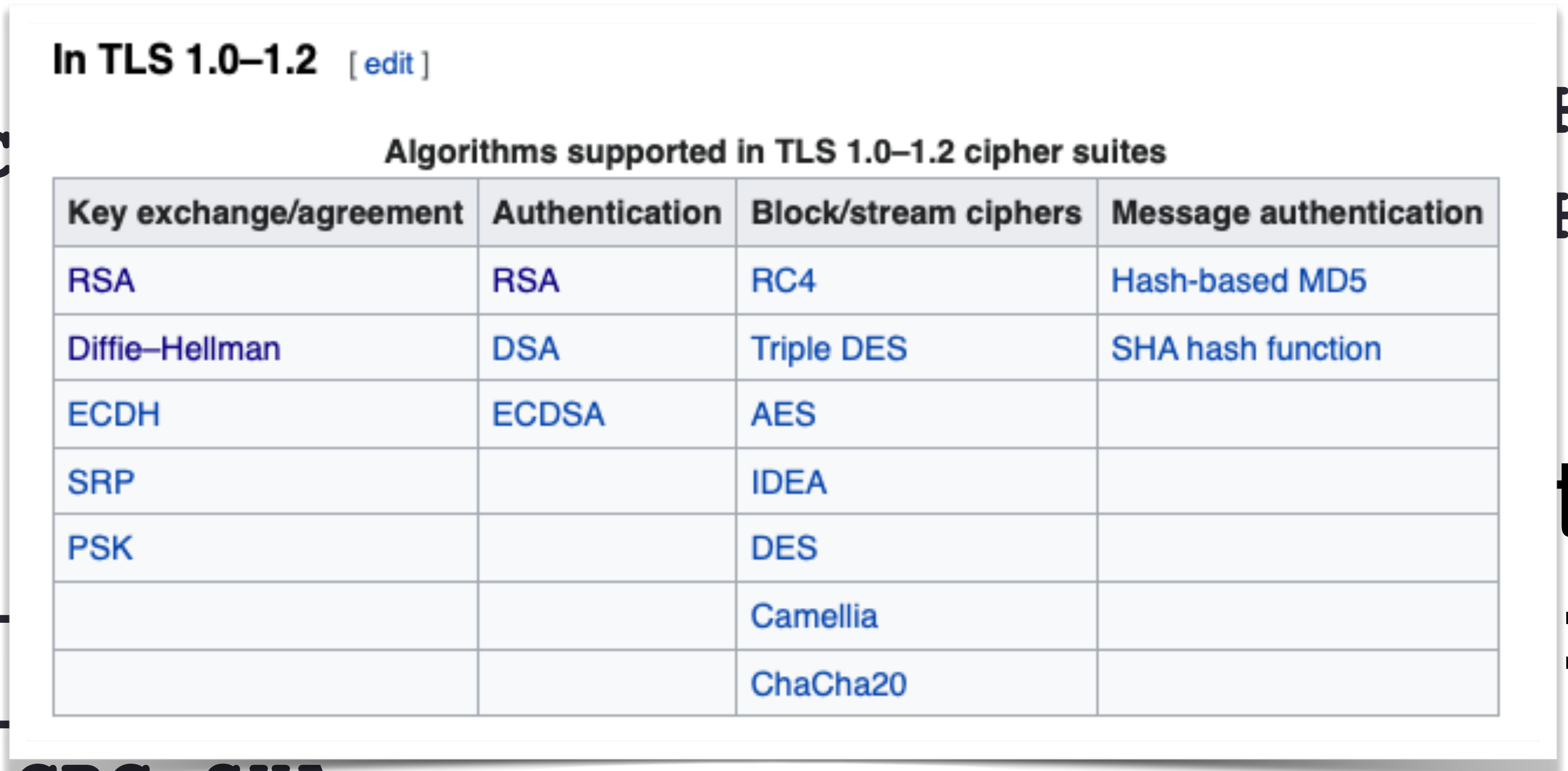
Cipher Suites with just encryption:

```
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_WITH_AES_128_CBC_SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA
```

Cipher Suites

Cipher Suites with encryption and authentication:

```
SSL_RSA_WITH_3DES_EDE_CBC
SSL_RSA_WITH_DES_CBC_SHA
SSL_RSA_WITH_RC4_128_MD5
SSL_RSA_WITH_RC4_128_SHA
TLS_DHE_DSS_WITH_AES_128_
TLS_DHE_DSS_WITH_AES_256_
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
...
```



In TLS 1.0–1.2 [edit]

Algorithms supported in TLS 1.0–1.2 cipher suites

Key exchange/agreement	Authentication	Block/stream ciphers	Message authentication
RSA	RSA	RC4	Hash-based MD5
Diffie–Hellman	DSA	Triple DES	SHA hash function
ECDH	ECDSA	AES	
SRP		IDEA	
PSK		DES	
		Camellia	
		ChaCha20	

Cipher Suites with just authentication:

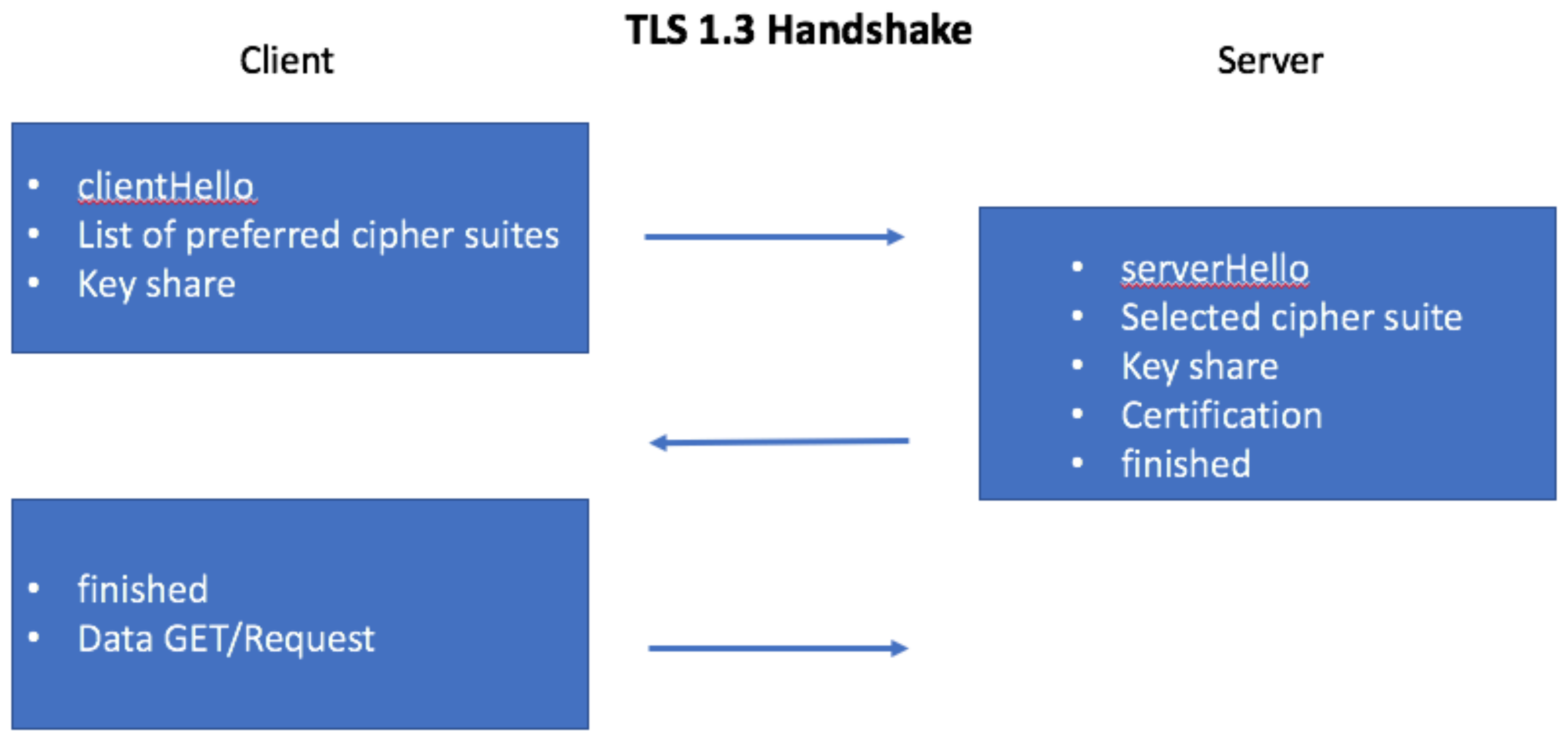
```
H_NULL_MD5
H_NULL_SHA
```

tes with just

:

```
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_WITH_AES_128_CBC_SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA
```

Cipher Suites Handshake



https://en.wikipedia.org/wiki/Cipher_suite

TLS Demo

- Websites
- Wireshark

Weaknesses in TLS

- Configuration weaknesses:
  - Cipher downgrading
  - Self-signed certificates
- Direct attack against implementations:
  - Apple’s goto fail bug
  - LogJam attack
  - HeartBleed

Cipher downgrading attack

Client supports:

```
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
```

Server supports:

```
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSL_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
```

Ciphers are listed in the order of preference.
What cipher will be used?

BUT...

Cipher downgrading attack

Client supports:

~~TLS_DHE_DSS_WITH_AES_256_CBC_SHA~~
~~TLS_DHE_DSS_WITH_AES_128_CBC_SHA~~
~~SSL_RSA_WITH_3DES_EDE_CBC_SHA~~
SSL_RSA_WITH_DES_CBC_SHA

Server supports:

~~TLS_DHE_DSS_WITH_AES_128_CBC_SHA~~
~~SSL_RSA_WITH_3DES_EDE_CBC_SHA~~
SSL_RSA_WITH_DES_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5
TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA

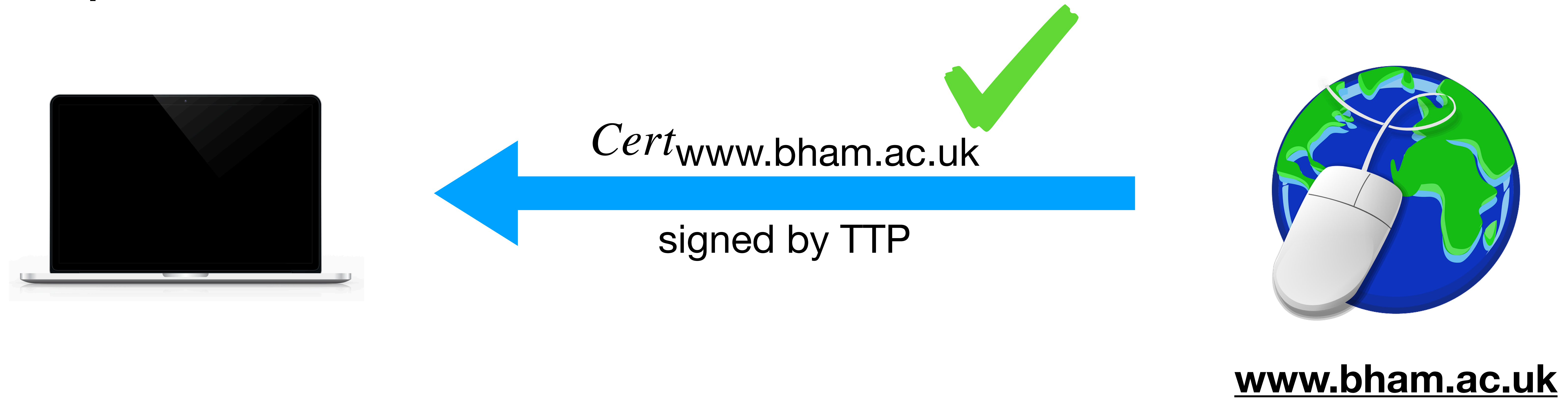
The cipher suite messages are not authenticated!

An attacker that owns the network can remove strong ciphers.

If both client and server support a weak cipher, then an attack can force its use.

Self-signed Certificates

- Maintaining a set of certificates is hard (especially on apps and IoT devices).
- It's much easier just to accept any certificate (or certificates that sign themselves).
- What's the problem?

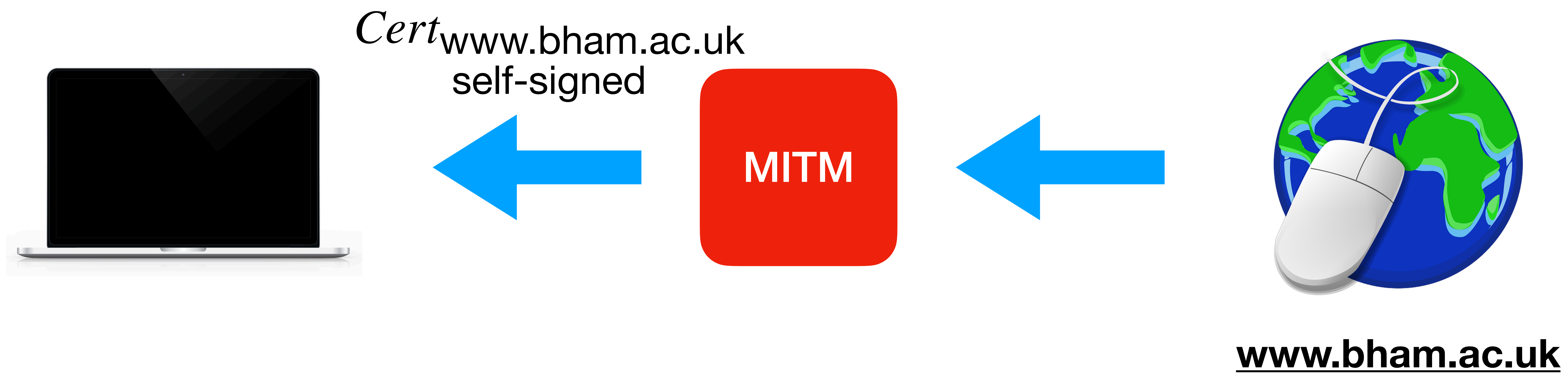


Self-signed Certificates

- Maintaining a set of certificates is hard (especially on apps and IoT devices).

- It’s much easier just to accept any certificate (or certificates that sign themselves).

- What’s the problem?



Self-signed Certificates

- Maintaining a set of certificates is hard
  (especially on apps and IoT devices).

- It’s much easier just to accept any certificate
  (or certificates that sign themselves).

- If the client accepts the self-signed certificates, then it’s easy to man-in-the-middle.

- This has been shown to happen a lot in devices and code that use TLS!

Apple's Implementation of TLS

```c
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer signedParams,
                                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus        err;
    SSLBuffer       hashOut, hashCtx, clientRandom, serverRandom;
    uint8_t         hashes[SSL_SHA1_DIGEST_LEN + SSL_MD5_DIGEST_LEN];
    SSLBuffer       signedHashes;
    uint8_t            *dataToSign;
     size_t            dataToSignLen;

     signedHashes.data = 0;
    hashCtx.data = 0;

    clientRandom.data = ctx->clientRandom;
    clientRandom.length = SSL_CLIENT_SRVR_RAND_SIZE;
    serverRandom.data = ctx->serverRandom;
    serverRandom.length = SSL_CLIENT_SRVR_RAND_SIZE;


     if(isRsa) {
          /* skip this if signing with DSA */
          dataToSign = hashes;
          dataToSignLen = SSL_SHA1_DIGEST_LEN + SSL_MD5_DIGEST_LEN;
          hashOut.data = hashes;
          hashOut.length = SSL_MD5_DIGEST_LEN;

          if ((err = ReadyHash(&SSLHashMD5, &hashCtx)) != 0)
              goto fail;
          if ((err = SSLHashMD5.update(&hashCtx, &clientRandom)) != 0)
              goto fail;
          if ((err = SSLHashMD5.update(&hashCtx, &serverRandom)) != 0)
              goto fail;
          if ((err = SSLHashMD5.update(&hashCtx, &signedParams)) != 0)
              goto fail;
          if ((err = SSLHashMD5.final(&hashCtx, &hashOut)) != 0)
              goto fail;
     }
     else {
          /* DSA, ECDSA - just use the SHA1 hash */
          dataToSign = &hashes[SSL_MD5_DIGEST_LEN];
          dataToSignLen = SSL_SHA1_DIGEST_LEN;
     }
```

```c
     hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
    hashOut.length = SSL_SHA1_DIGEST_LEN;
    if ((err = SSLFreeBuffer(&hashCtx)) != 0)
        goto fail;

    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

     err = sslRawVerify(ctx,
                        ctx->peerPubKey,
                        dataToSign,                /* plaintext */
                        dataToSignLen,             /* plaintext length */
                        signature,
                        signatureLen);
     if(err) {
         sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                     "returned %d\n", (int)err);
         goto fail;
     }

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;

}
```

http://opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c

Apple's TLS-DHE

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
  goto fail;
... other checks ...
fail:
  ... buffer frees (cleanups) ...
  return err;
```

1. $C \to S : N_C$

2. $S \to C : N_S, g^x, Cert_S, Sign_S(\#(N_C, N_S, g^x))$

3. $C \to S : g^y, \{\#(\text{All previous messages})\}_{K_{CS}}$

4. $S \to C : \{\#(\text{All previous messages})\}_{K_{CS}}$

K_{CS} is a session key based on N_C, N_S, g^{xy}.

Apple's TLS-DHE

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
  goto fail;
... other checks ...
fail:
  ... buffer frees (cleanups) ...
  return err;
```

1. $C \rightarrow S : N_C$

2. $S \rightarrow C : N_S, g^x, Cert_S, True$

3. $C \rightarrow S : g^y, \{\#(\text{All previous messages})\}_{K_{CS}}$

4. $S \rightarrow C : \{\#(\text{All previous messages})\}_{K_{CS}}$

K_{CS} is a session key based on N_C, N_S, g^{xy}.

Major issues

- iOS fixed days before macOS!
- Why didn't tests pick this up?
- Compiler should have warned of unreachable code.
- Bad programming style: no brackets, goto:

I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

http://xkcd.com/292/

Cipher Suites

- What if one side supports a weak cipher suite but the other does not?
- Generally considered safe.
- Browser developers removed all weak ciphers, some remained in servers.
- This depends on different cipher suites being incompatible, e.g.:
  `SSL_RSA_WITH_DES_CBC_SHA` and
  `TLS_DHE_DSS_WITH_AES_256_CBC_SHA`

LogJam

- The Snowden leaks revealed that the NSA regularly MITMed TLS.
- How could they be doing this? Someone had missed something.
- These people figured it out: https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf
- A weak Diffie Hellman key is compatible with a strong Diffie Hellman key!

Diffie-Hellman

- Alice and Bob pick random numbers r_A and r_B and find “$t_A = g^{r_A} \mod p$” and “$t_B = g^{r_B} \mod p$”

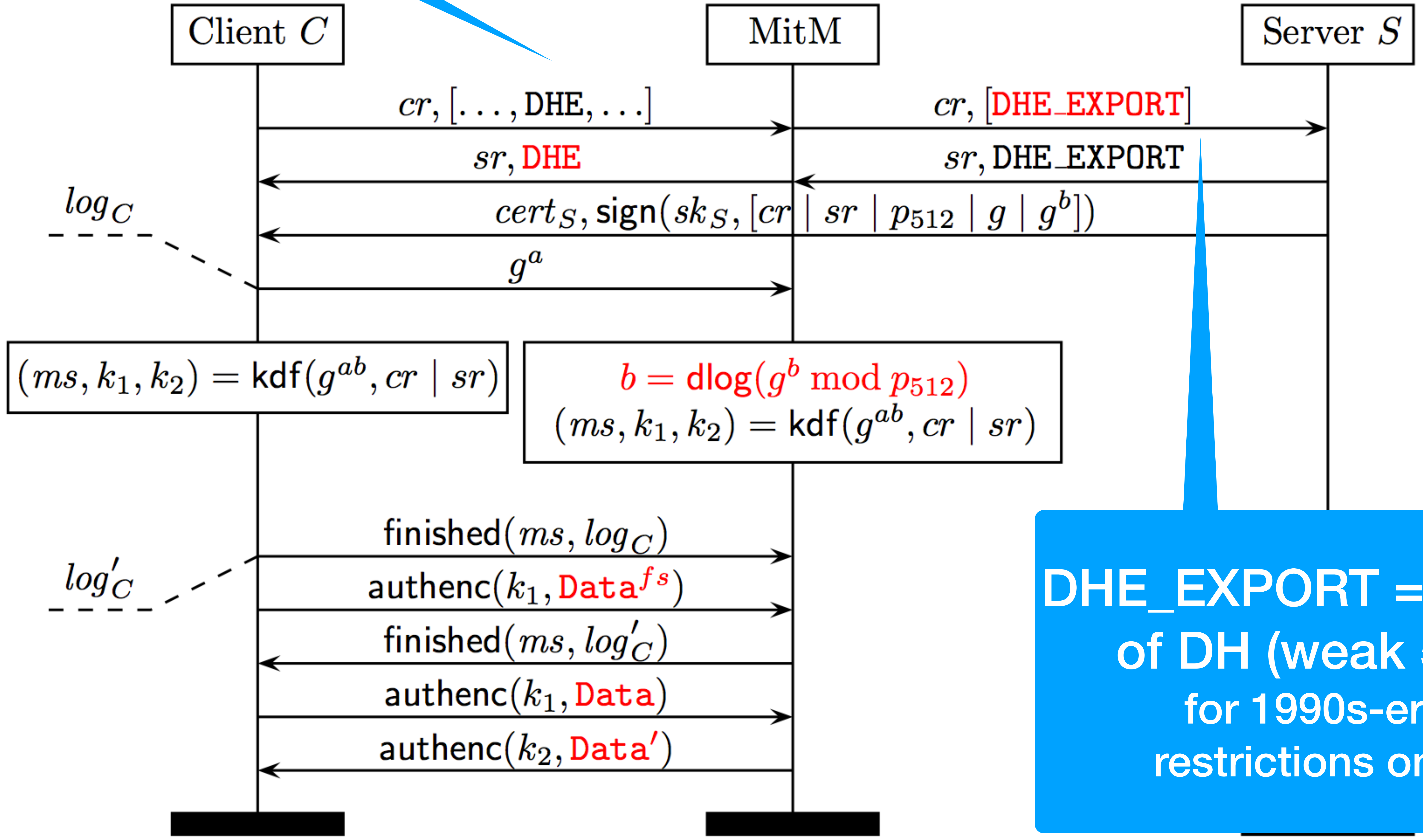
- The protocol just exchanges these numbers:

  1. $A \rightarrow B : t_A$
  2. $B \rightarrow A : t_B$

- Alice calculates “${t_B}^{r_A} \mod p$” and Bob “${t_A}^{r_B} \mod p$”, receiving the key: $K = g^{r_A r_B} \mod p$

Attack Diagram from paper

DHE = ephemeral DH (strong prime)

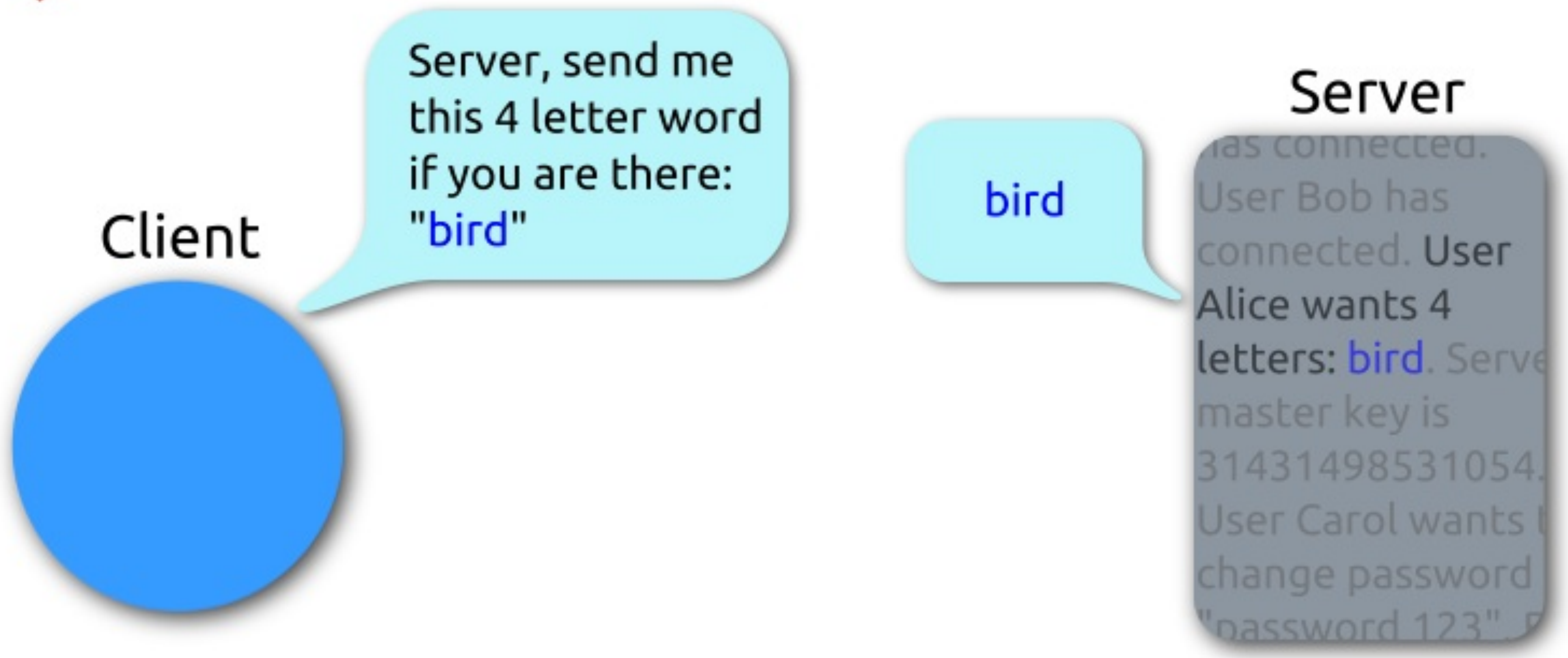

Client C
MitM
Server S
$cr, [\ldots, \mathrm{DHE}, \ldots]$
cr, [DHE_EXPORT]
sr, DHE
sr, DHE_EXPORT
log_C
$cert_S, \mathsf{sign}(sk_S, [cr \mid sr \mid p_{512} \mid g \mid g^b])$
g^a
$(ms, k_1, k_2) = \mathsf{kdf}(g^{ab}, cr \mid sr)$
$b = \mathsf{dlog}(g^b \bmod p_{512})$
$(ms, k_1, k_2) = \mathsf{kdf}(g^{ab}, cr \mid sr)$
log'_C
$\mathsf{finished}(ms, log_C)$
$\mathsf{authenc}(k_1, \mathtt{Data}^{fs})$
$\mathsf{finished}(ms, log'_C)$
$\mathsf{authenc}(k_1, \mathtt{Data})$
$\mathsf{authenc}(k_2, \mathtt{Data}')$

DHE_EXPORT = "export version" of DH (weak 512-bit prime) for 1990s-era U.S. export restrictions on cryptography

HeartBleed

- A programming error in OpenSSL
- Introduced in 2012, made public in 2014.
- Rumours it was being exploited.
- TLS client can request a “heart beat” from the server to make sure the connection is still open.
- This memory could contain the server’s key.

Heartbeat – Normal usage


Server, send me this 4 letter word if you are there: "bird"
Client
Server
bird
as connected. User Bob has connected. User Alice wants 4 letters: bird. Serve master key is 31431498531054. User Carol wants t change password "password 123".

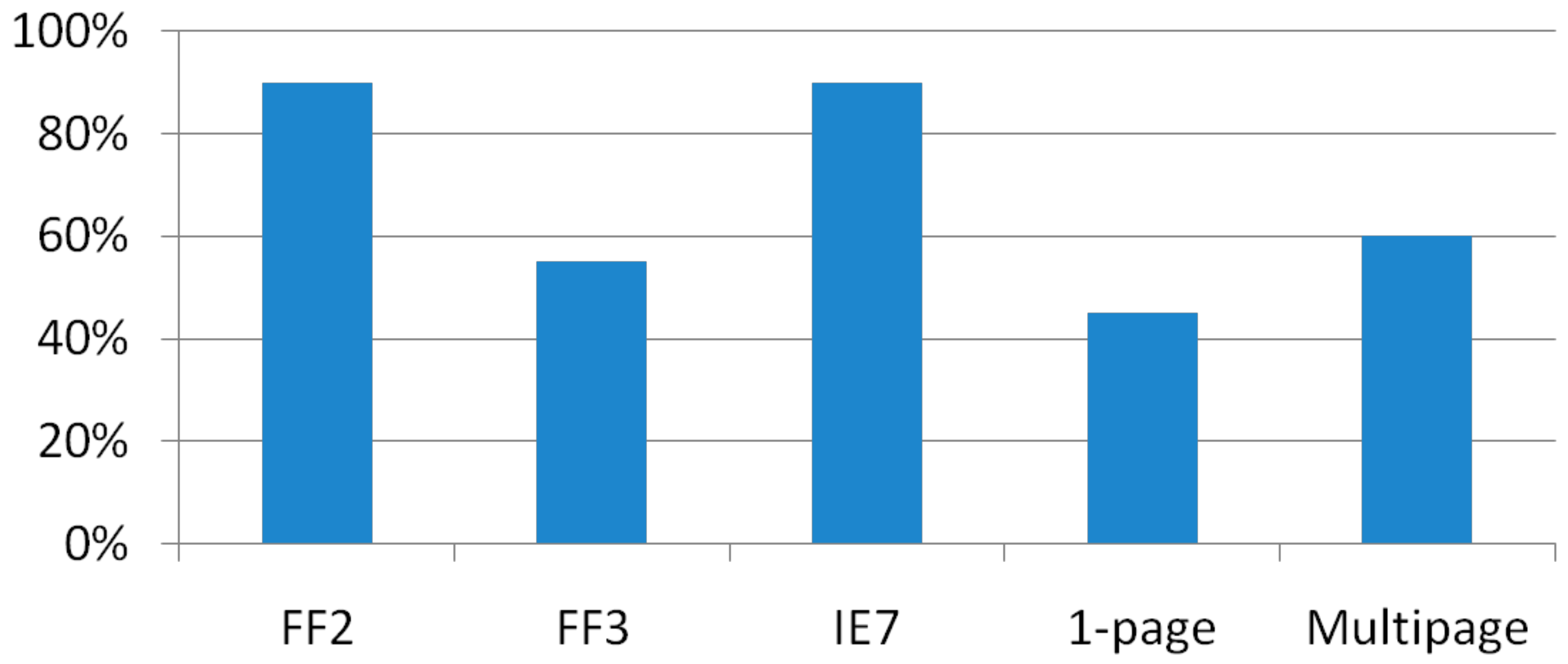
From Wikipedia

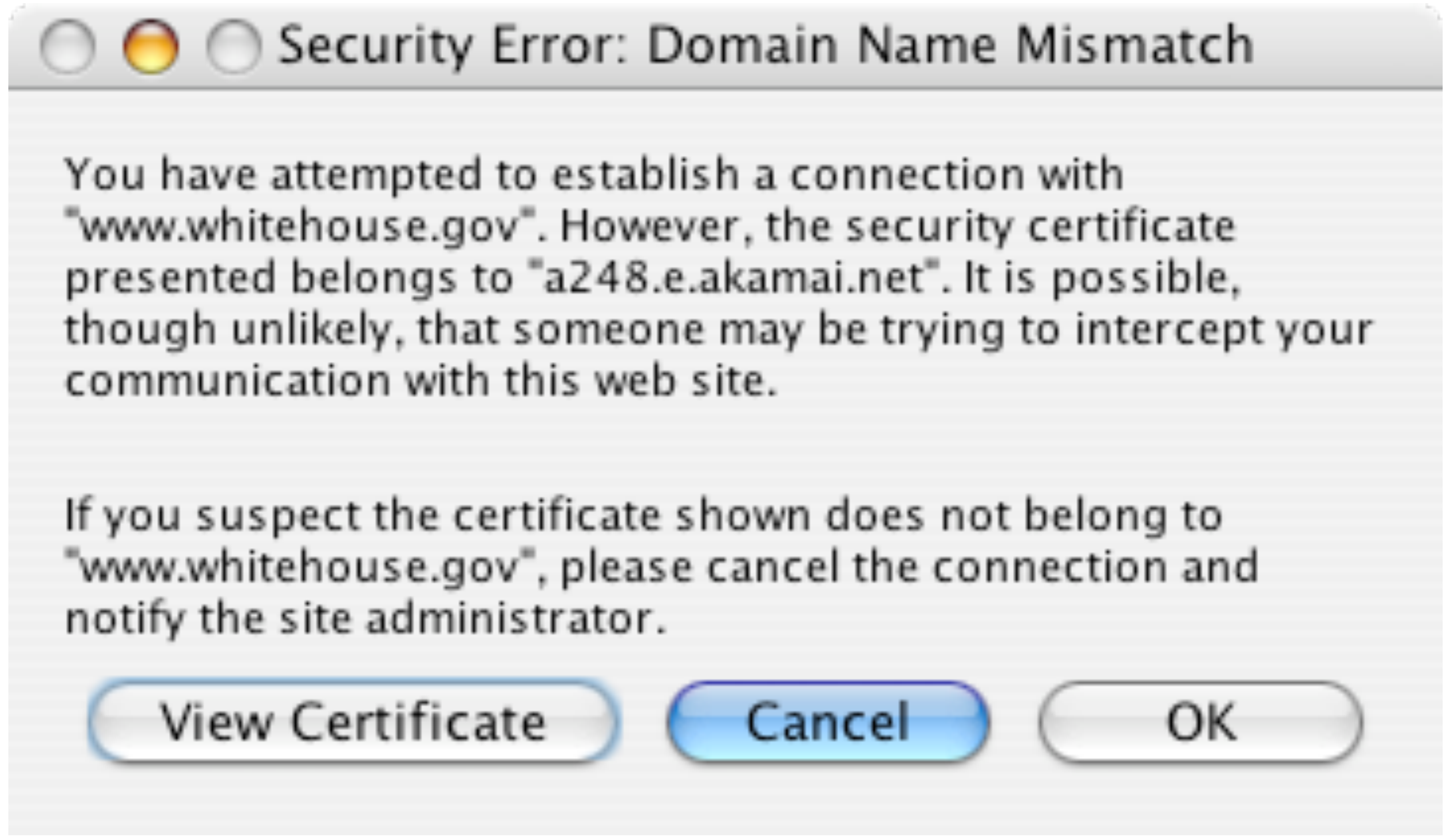
TLS 1.3

- Newest standard, ratified August 2018
- Removes obsolete cryptographic protocols
- Simplified handshake ⇒ efficiency gain
- Forward secrecy mandatory
- Intercepting TLS connections now only possible as active attacker performing MITM attack

Cranor et al.'s Crying Wolf: "An Empirical Study of SSL Warning Effectiveness"

People that ignored warnings:

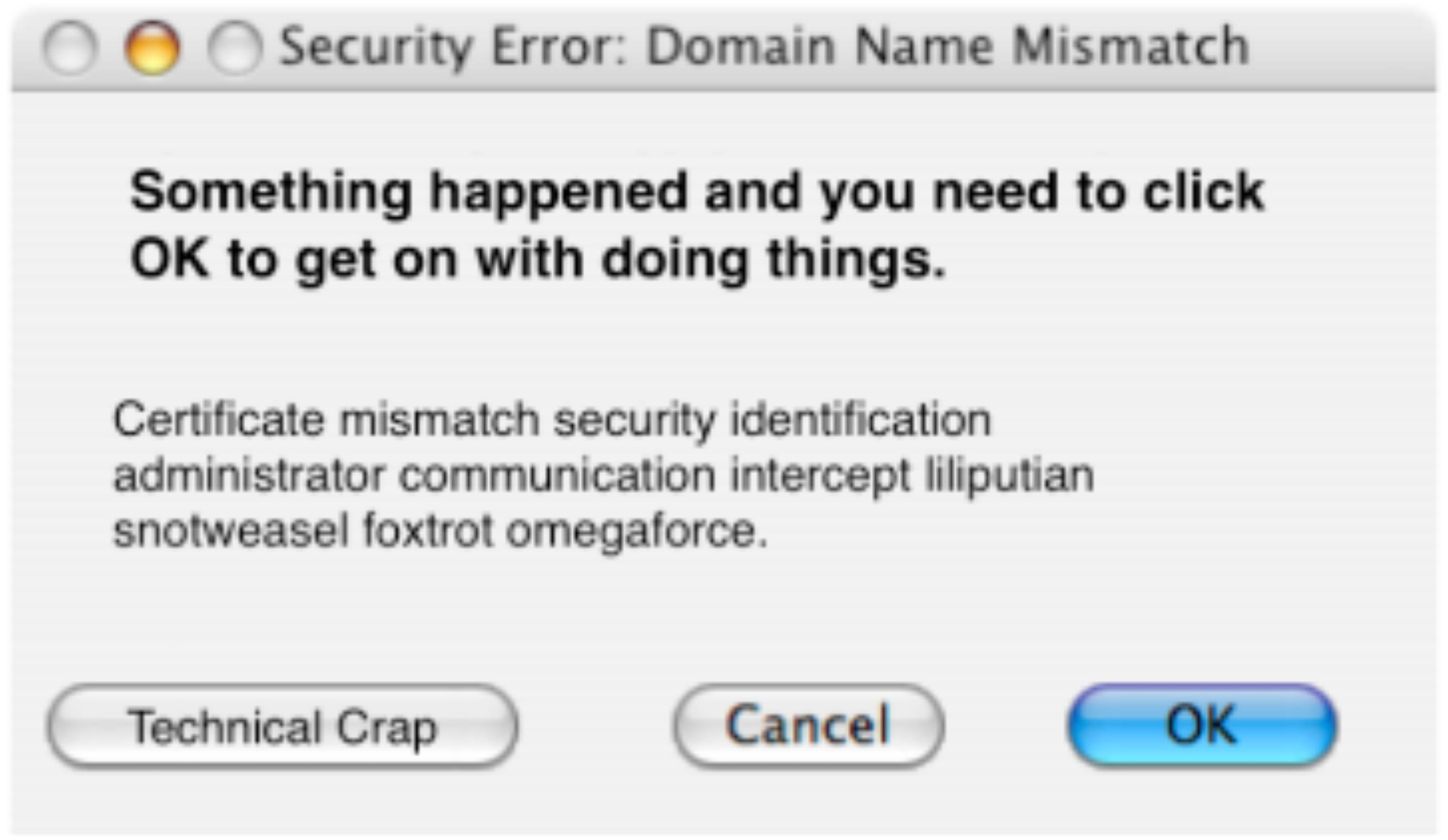



Security Error: Domain Name Mismatch

You have attempted to establish a connection with "www.whitehouse.gov". However, the security certificate presented belongs to "a248.e.akamai.net". It is possible, though unlikely, that someone may be trying to intercept your communication with this web site.

If you suspect the certificate shown does not belong to "www.whitehouse.gov", please cancel the connection and notify the site administrator.

Image courtesy of Johnathan Nightingale


Security Error: Domain Name Mismatch

Something happened and you need to click OK to get on with doing things.

Certificate mismatch security identification administrator communication intercept liliputian snotweasel foxtrot omegaforce.

Image courtesy of Johnathan Nightingale

Checking servers

- There are many insecure TLS servers on the internet.
- The most common problems are support for weak ciphers and old unpatched code.
- SSL labs provide a useful testing tool:
  - https://www.ssllabs.com/ssltest/index.html

Today's Lecture

- Details of TLS
    - How it works
    - Common problems
- Tor
    - Anonymity on the internet
    - The Tor Protocol
    - Hidden servers

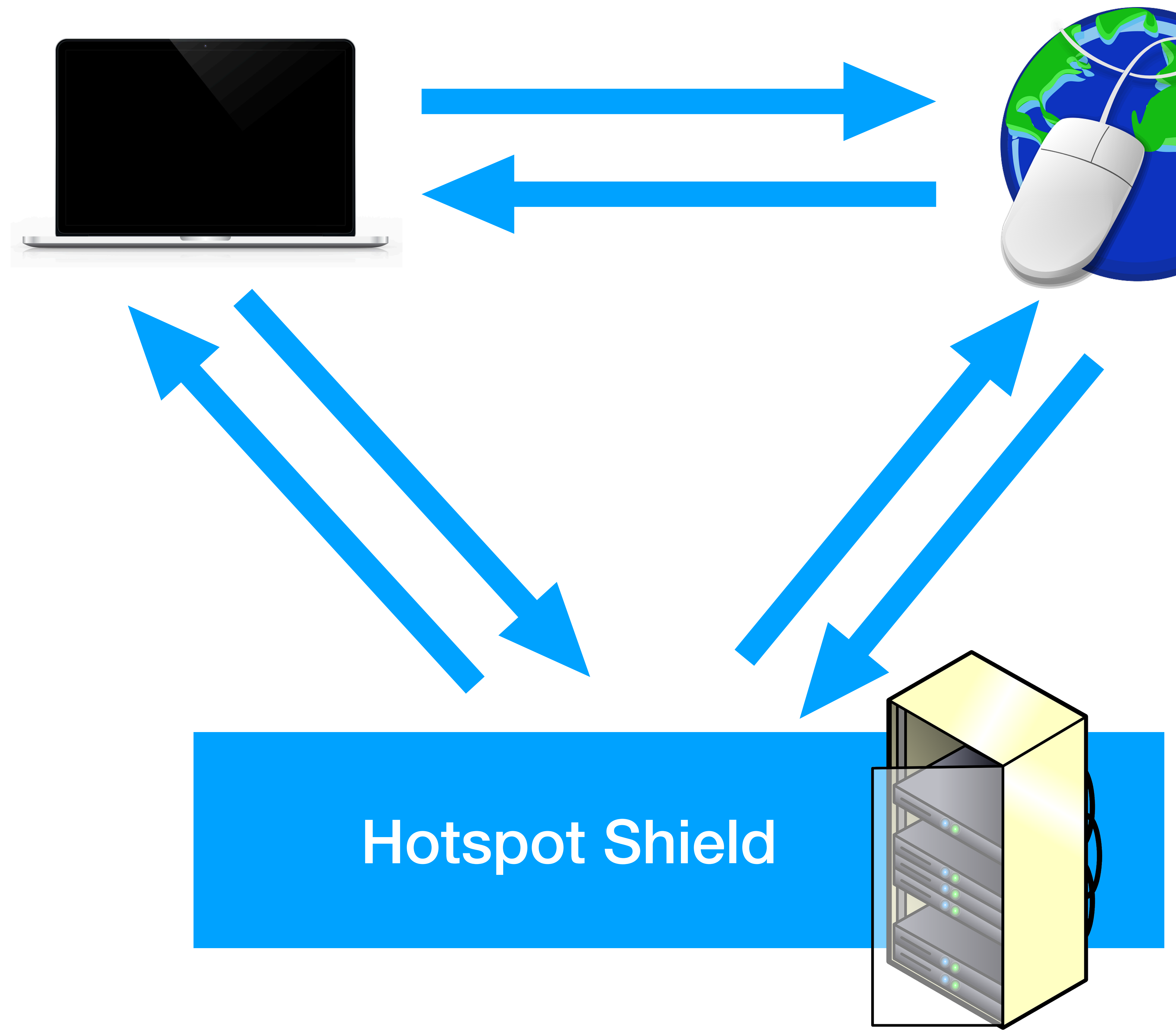
“You have zero privacy anyway, get over it.”

–Scott McNealy, former CEO of SUN Microsystems

“With your permission, you give us more information about you, about your friends, and we can improve the quality of our searches. We don't need you to type at all. We know where you are. We know where you've been. We can more or less know what you're thinking about.”

–Eric Schmidt, former CEO of Google

Proxy:
Hotspot Shield VPN



- An internet connection reveals your IP number.
- VPNs promise “Anonymity”.
- Connection made via their servers.
- The intended recipient server never see’s your IP address.

Virtual Private Networks

- VPNs securely connect you to another network.
- e.g., you can connect to the school’s printers via the school’s VPN.
- Secured with certificates and encryption, e.g., TLS or IPSec.

Virtual Private Networks For Anonymity

- To get some anonymity, you can route all your traffic via the VPN.
    - Server thinks you are the VPN provider
    - ISP only sees the connection to the VPN
    - A global observer can probably link your connections.
- There is **no anonymity** to the VPN provider.

Virtual Private Networks For Anonymity

- Quiz!
- Suppose you're connected to a public WiFi hotspot and browse to the website "https://bham.ac.uk" using a VPN.
- What information does the **WiFi** provider have about you?
  - …your WiFi's outgoing IP address ✓
  - …that you are connected to the VPN ✓
  - …your VPN's outgoing IP address ✗
  - …that you are browsing to "https://bham.ac.uk" ✗
  - …the contents of your communication with the website ✗

Virtual Private Networks For Anonymity

- Quiz!
- Suppose you're connected to a public WiFi hotspot and browse to the website "https://bham.ac.uk" using a VPN.
- What information does the **VPN** provider have about you?
  - …your WiFi's outgoing IP address ✓
  - …that you are connected to the VPN ✓
  - …your VPN's outgoing IP address ✓
  - …that you are browsing to "https://bham.ac.uk" ✓
  - …the contents of your communication with the website ✗

Virtual Private Networks For Anonymity

- Quiz!
- Suppose you're connected to a public WiFi hotspot and browse to the website "https://bham.ac.uk" using a VPN.
- What information does the **website** provider have about you?
  - …your WiFi's outgoing IP address ✗
  - …that you are connected to the VPN ✗ **Might infer that from the IP address**
  - …your VPN's outgoing IP address ✓
  - …that you are browsing to "https://bham.ac.uk" ✓
  - …the contents of your communication with the website ✓

Onion Routing

- You get the best anonymity by routing your traffic via a number of proxies.
- Onion Routing ensures that your message really is routed via the proxies you want.
- The Tor network is using this protocol https://www.torproject.org/

Tor: Onion Routing

- Each proxy only learns the IP of the proxy before it and the proxy after it.
- The public key of each proxy is known.
- Source IP is visible to the first node, destination IP is visible to the last node.
- User picks 3 proxies (entry, middle, and exit node) and is anonymous as long as they aren’t all corrupt.

Tor: Onion Routing

m
Tor Node 1
Tor Node 2
Tor Node 3

Tor: Onion Routing

m
Tor Node 1
Tor Node 2
Tor Node 3

Tor: Onion Routing

m
Tor Node 1
Tor Node 2
Tor Node 3

Tor: Onion Routing

m
Tor Node 1
Tor Node 2
Tor Node 3

Tor: Onion Routing

m
Tor Node 1
Tor Node 2
Tor Node 3

Tor: Onion Routing

Tor Node 1
Tor Node 2
Tor Node 3
$E_{T_1}(g^x)$
$g^y, H(g^{xy})$
$\{T_2, E_{T_2}(g^z)\}_{g^{xy}}$
$E_{T_2}(g^z)$
$g^w, H(g^{zw})$
$\{g^w, H(g^{zw})\}_{g^{xy}}$
$\{\{T_3, E_{T_3}(g^u)\}_{g^{zw}}\}_{g^{xy}}$
$\{T_3, E_{T_3}(g^u)\}_{g^{zw}}$
$E_{T_3}(g^u)$
$g^v, H(g^{uv})$
$\{g^v, H(g^{uv})\}_{g^{zw}}$
$\{\{g^v, H(g^{uv})\}_{g^{zw}}\}_{g^{xy}}$
$\{\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$
$\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}$
$\{Server, m\}_{g^{uv}}$
m
r
$\{r\}_{g^{uv}}$
$\{\{r\}_{g^{uv}}\}_{g^{zw}}$
$\{\{\{r\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$

Tor: Onion Routing

Tor Node 1
Tor Node 2
Tor Node 3
$E_{T_1}(g^x)$
$g^y, H(g^{xy})$
$\{T_2, E_{T_2}(g^z)\}_{g^{xy}}$
$E_{T_2}(g^z)$
$g^w, H(g^{zw})$
$\{g^w, H(g^{zw})\}_{g^{xy}}$
$\{\{T_3, E_{T_3}(g^u)\}_{g^{zw}}\}_{g^{xy}}$
$\{T_3, E_{T_3}(g^u)\}_{g^{zw}}$
$E_{T_3}(g^u)$
$g^v, H(g^{uv})$
$\{g^v, H(g^{uv})\}_{g^{zw}}$
$\{\{g^v, H(g^{uv})\}_{g^{zw}}\}_{g^{xy}}$
m
$\{\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$
$\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}$
$\{Server, m\}_{g^{uv}}$
m
r
$\{r\}_{g^{uv}}$
$\{\{r\}_{g^{uv}}\}_{g^{zw}}$
$\{\{\{r\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$

Tor: Onion Routing

Tor Node 1
Tor Node 2
Tor Node 3
$E_{T_1}(g^x)$
$g^y, H(g^{xy})$
$\{T_2, E_{T_2}(g^z)\}_{g^{xy}}$
$E_{T_2}(g^z)$
$g^w, H(g^{zw})$
$\{g^w, H(g^{zw})\}_{g^{xy}}$
$\{\{T_3, E_{T_3}(g^u)\}_{g^{zw}}\}_{g^{xy}}$
$\{T_3, E_{T_3}(g^u)\}_{g^{zw}}$
$E_{T_3}(g^u)$
m
$g^v, H(g^{uv})$
$\{g^v, H(g^{uv})\}_{g^{zw}}$
$\{\{g^v, H(g^{uv})\}_{g^{zw}}\}_{g^{xy}}$
$\{\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$
$\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}$
$\{Server, m\}_{g^{uv}}$
m
r
$\{r\}_{g^{uv}}$
$\{\{r\}_{g^{uv}}\}_{g^{zw}}$
$\{\{\{r\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$

Tor: Onion Routing

Tor Node 1
Tor Node 2
Tor Node 3
$E_{T_1}(g^x)$
$g^y, H(g^{xy})$
$\{T_2, E_{T_2}(g^z)\}_{g^{xy}}$
$E_{T_2}(g^z)$
$g^w, H(g^{zw})$
$\{g^w, H(g^{zw})\}_{g^{xy}}$
$\{\{T_3, E_{T_3}(g^u)\}_{g^{zw}}\}_{g^{xy}}$
$\{T_3, E_{T_3}(g^u)\}_{g^{zw}}$
m
$E_{T_3}(g^u)$
$g^v, H(g^{uv})$
$\{g^v, H(g^{uv})\}_{g^{zw}}$
$\{\{g^v, H(g^{uv})\}_{g^{zw}}\}_{g^{xy}}$
$\{\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$
$\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}$
$\{Server, m\}_{g^{uv}}$
m
r
$\{r\}_{g^{uv}}$
$\{\{r\}_{g^{uv}}\}_{g^{zw}}$
$\{\{\{r\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$

Tor: Onion Routing

Tor Node 1
Tor Node 2
Tor Node 3
$E_{T_1}(g^x)$
$g^y, H(g^{xy})$
$\{T_2, E_{T_2}(g^z)\}_{g^{xy}}$
$E_{T_2}(g^z)$
$g^w, H(g^{zw})$
$\{g^w, H(g^{zw})\}_{g^{xy}}$
$\{\{T_3, E_{T_3}(g^u)\}_{g^{zw}}\}_{g^{xy}}$
$\{T_3, E_{T_3}(g^u)\}_{g^{zw}}$
$E_{T_3}(g^u)$
m
$g^v, H(g^{uv})$
$\{g^v, H(g^{uv})\}_{g^{zw}}$
$\{\{g^v, H(g^{uv})\}_{g^{zw}}\}_{g^{xy}}$
$\{\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$
$\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}$
$\{Server, m\}_{g^{uv}}$
m
r
$\{r\}_{g^{uv}}$
$\{\{r\}_{g^{uv}}\}_{g^{zw}}$
$\{\{\{r\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$

Tor: Onion Routing

Tor Node 1
Tor Node 2
Tor Node 3
$E_{T_1}(g^x)$
$g^y, H(g^{xy})$
$\{T_2, E_{T_2}(g^z)\}_{g^{xy}}$
$E_{T_2}(g^z)$
$g^w, H(g^{zw})$
$\{g^w, H(g^{zw})\}_{g^{xy}}$
$\{\{T_3, E_{T_3}(g^u)\}_{g^{zw}}\}_{g^{xy}}$
$\{T_3, E_{T_3}(g^u)\}_{g^{zw}}$
$E_{T_3}(g^u)$
$g^v, H(g^{uv})$
$\{g^v, H(g^{uv})\}_{g^{zw}}$
$\{\{g^v, H(g^{uv})\}_{g^{zw}}\}_{g^{xy}}$
r
$\{\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$
$\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}$
$\{Server, m\}_{g^{uv}}$
m
r
$\{r\}_{g^{uv}}$
$\{\{r\}_{g^{uv}}\}_{g^{zw}}$
$\{\{\{r\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$

Tor: Onion Routing

Tor Node 1
Tor Node 2
Tor Node 3
$E_{T_1}(g^x)$
$g^y, H(g^{xy})$
$\{T_2, E_{T_2}(g^z)\}_{g^{xy}}$
$E_{T_2}(g^z)$
$g^w, H(g^{zw})$
$\{g^w, H(g^{zw})\}_{g^{xy}}$
$\{\{T_3, E_{T_3}(g^u)\}_{g^{zw}}\}_{g^{xy}}$
$\{T_3, E_{T_3}(g^u)\}_{g^{zw}}$
$E_{T_3}(g^u)$
r
$g^v, H(g^{uv})$
$\{g^v, H(g^{uv})\}_{g^{zw}}$
$\{\{g^v, H(g^{uv})\}_{g^{zw}}\}_{g^{xy}}$
$\{\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$
$\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}$
$\{Server, m\}_{g^{uv}}$
m
r
$\{r\}_{g^{uv}}$
$\{\{r\}_{g^{uv}}\}_{g^{zw}}$
$\{\{\{r\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$

Tor: Onion Routing

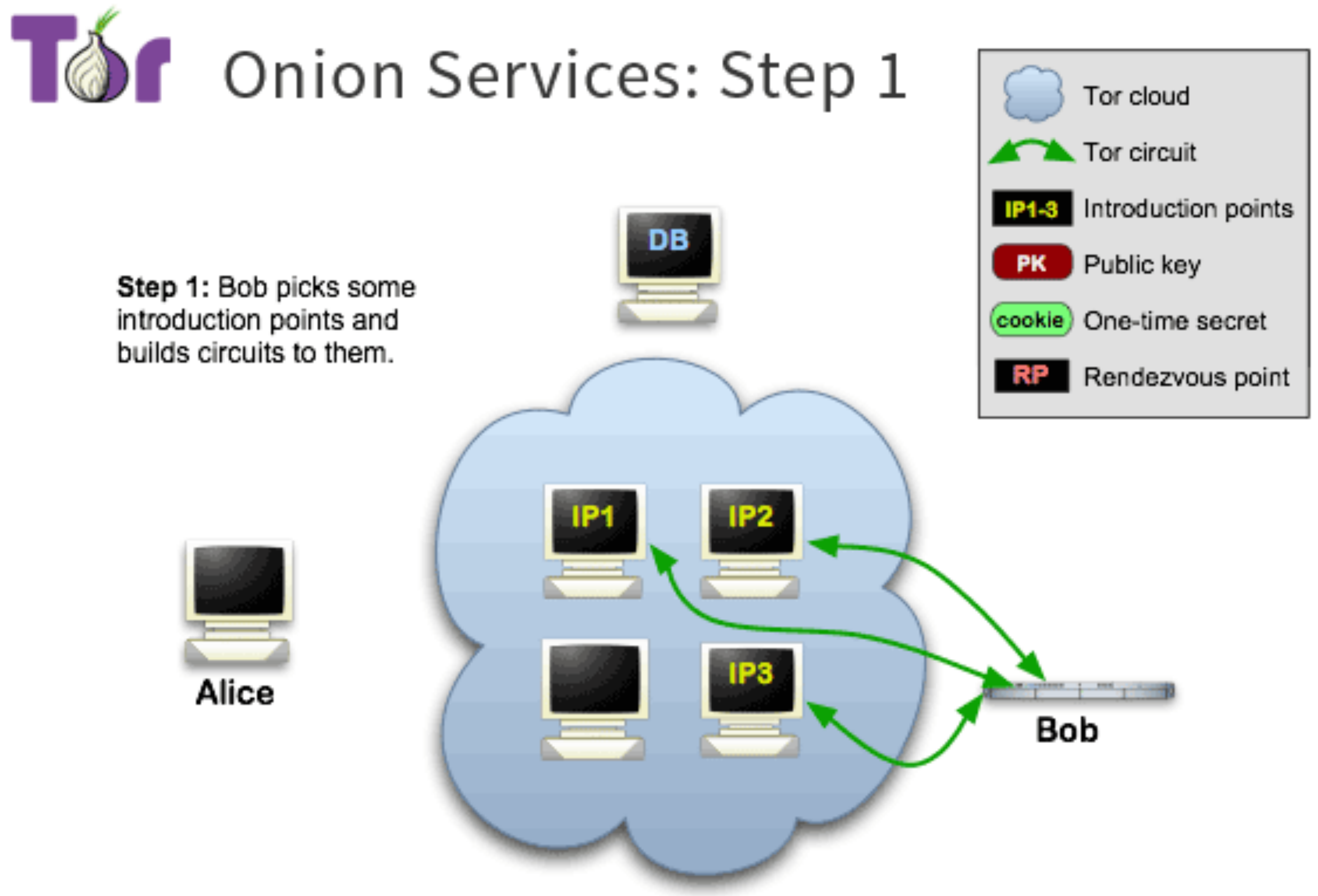
Tor Node 1
Tor Node 2
Tor Node 3
$E_{T_1}(g^x)$
$g^y, H(g^{xy})$
$\{T_2, E_{T_2}(g^z)\}_{g^{xy}}$
$E_{T_2}(g^z)$
$g^w, H(g^{zw})$
$\{g^w, H(g^{zw})\}_{g^{xy}}$
$\{\{T_3, E_{T_3}(g^u)\}_{g^{zw}}\}_{g^{xy}}$
$\{T_3, E_{T_3}(g^u)\}_{g^{zw}}$
$E_{T_3}(g^u)$
r
$g^v, H(g^{uv})$
$\{\{g^v, H(g^{uv})\}_{g^{zw}}\}_{g^{xy}}$
$\{g^v, H(g^{uv})\}_{g^{zw}}$
$\{\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$
$\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}$
$\{Server, m\}_{g^{uv}}$
m
r
$\{r\}_{g^{uv}}$
$\{\{r\}_{g^{uv}}\}_{g^{zw}}$
$\{\{\{r\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$

Tor: Onion Routing

Tor Node 1
Tor Node 2
Tor Node 3
$E_{T_1}(g^x)$
$g^y, H(g^{xy})$
$\{T_2, E_{T_2}(g^z)\}_{g^{xy}}$
$E_{T_2}(g^z)$
$g^w, H(g^{zw})$
$\{g^w, H(g^{zw})\}_{g^{xy}}$
$\{\{T_3, E_{T_3}(g^u)\}_{g^{zw}}\}_{g^{xy}}$
$\{T_3, E_{T_3}(g^u)\}_{g^{zw}}$
$E_{T_3}(g^u)$
$g^v, H(g^{uv})$
$\{g^v, H(g^{uv})\}_{g^{zw}}$
$\{\{g^v, H(g^{uv})\}_{g^{zw}}\}_{g^{xy}}$
r
$\{\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$
$\{\{Server, m\}_{g^{uv}}\}_{g^{zw}}$
$\{Server, m\}_{g^{uv}}$
m
r
$\{r\}_{g^{uv}}$
$\{\{r\}_{g^{uv}}\}_{g^{zw}}$
$\{\{\{r\}_{g^{uv}}\}_{g^{zw}}\}_{g^{xy}}$

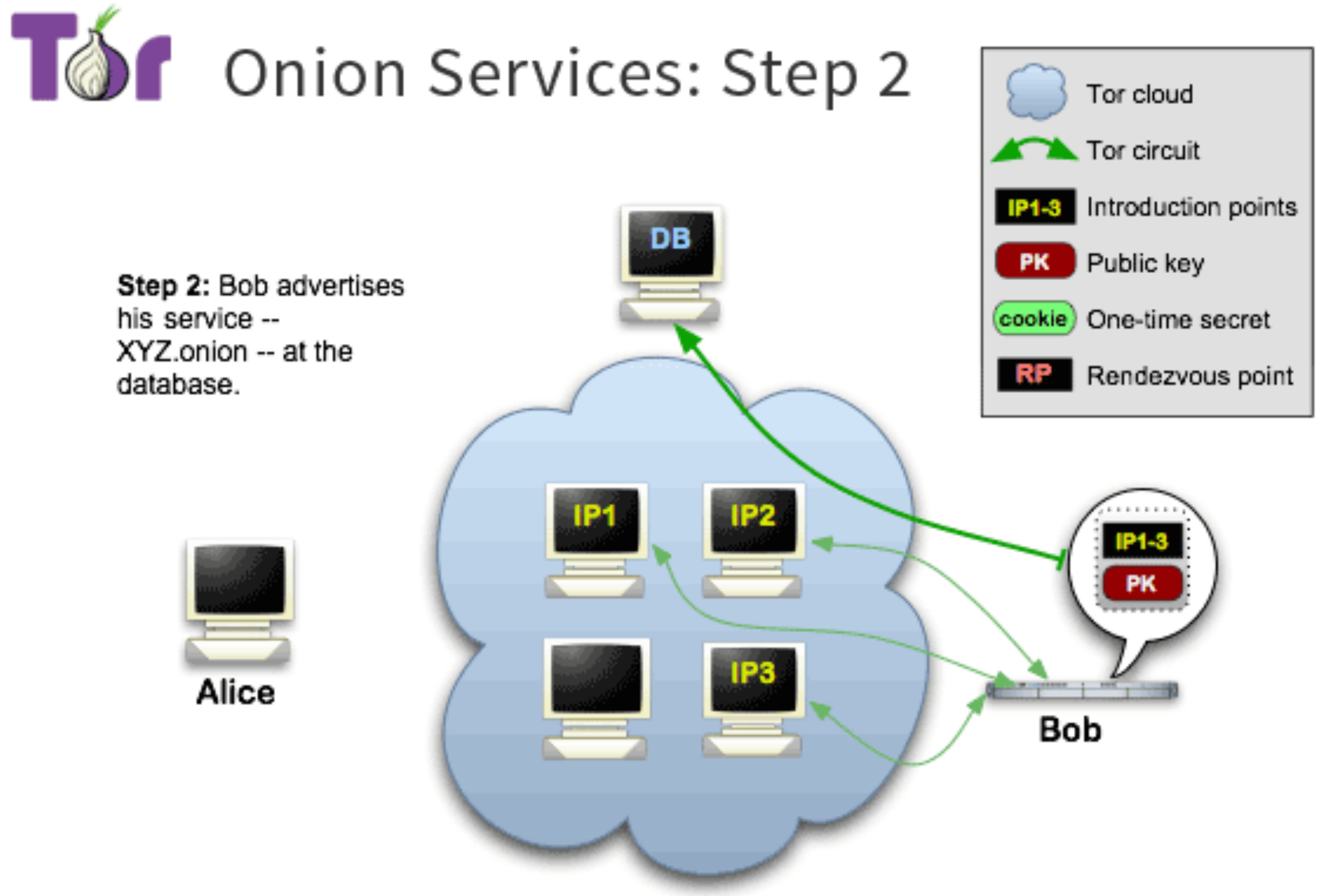
Hidden Servers

- Tor hidden servers hide the server from you.
- See: https://www.torproject.org/docs/hidden-services.html.en



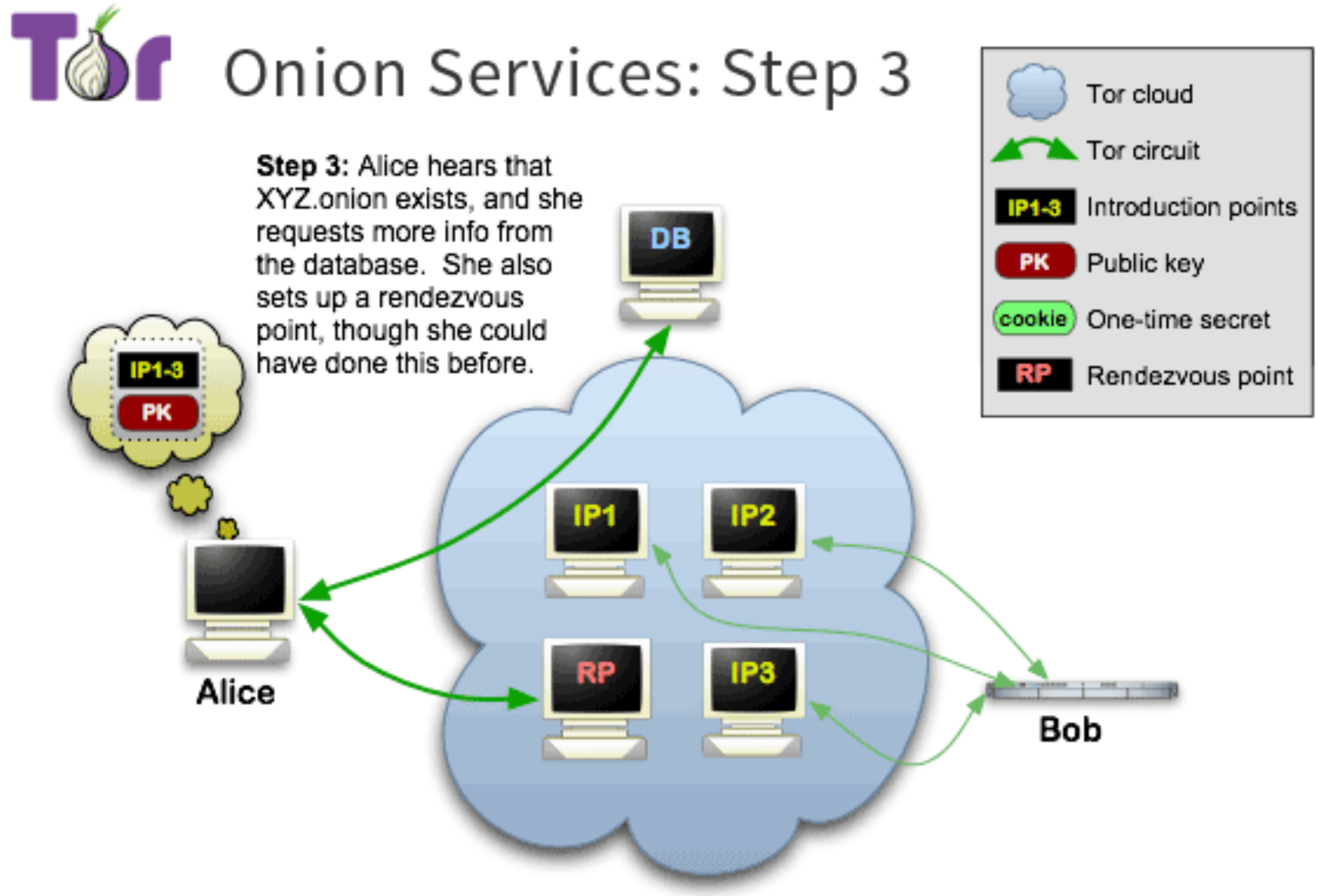
Hidden Servers

- Tor hidden servers hide the server from you.
- See: https://www.torproject.org/docs/hidden-services.html.en



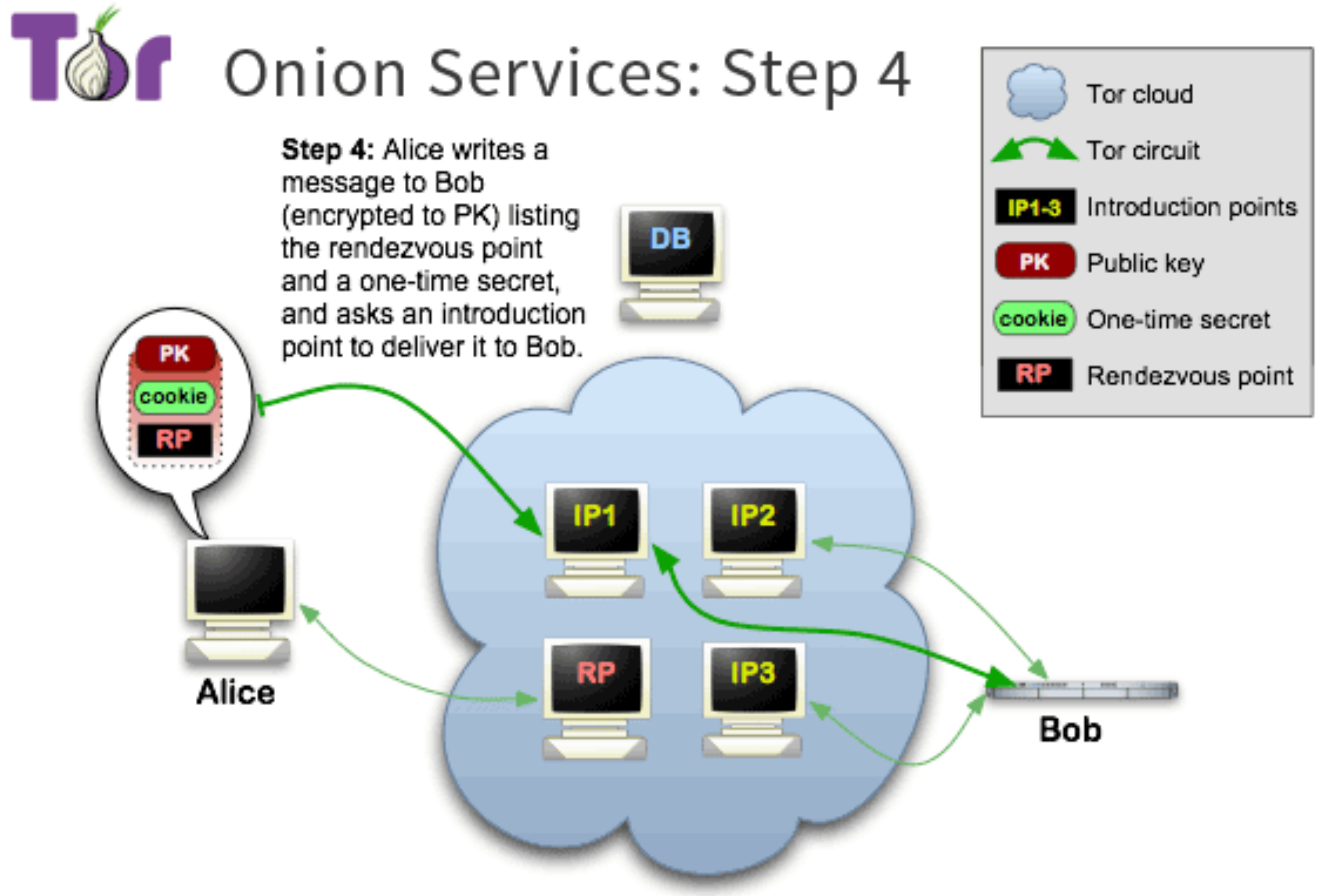
Hidden Servers

- Tor hidden servers hide the server from you.
- See: https://www.torproject.org/docs/hidden-services.html.en


Onion Services: Step 3
Step 3: Alice hears that XYZ.onion exists, and she requests more info from the database. She also sets up a rendezvous point, though she could have done this before.
Tor cloud
Tor circuit
IP1-3 Introduction points
PK Public key
cookie One-time secret
RP Rendezvous point
IP1-3
PK
DB
IP1
IP2
RP
IP3
Alice
Bob

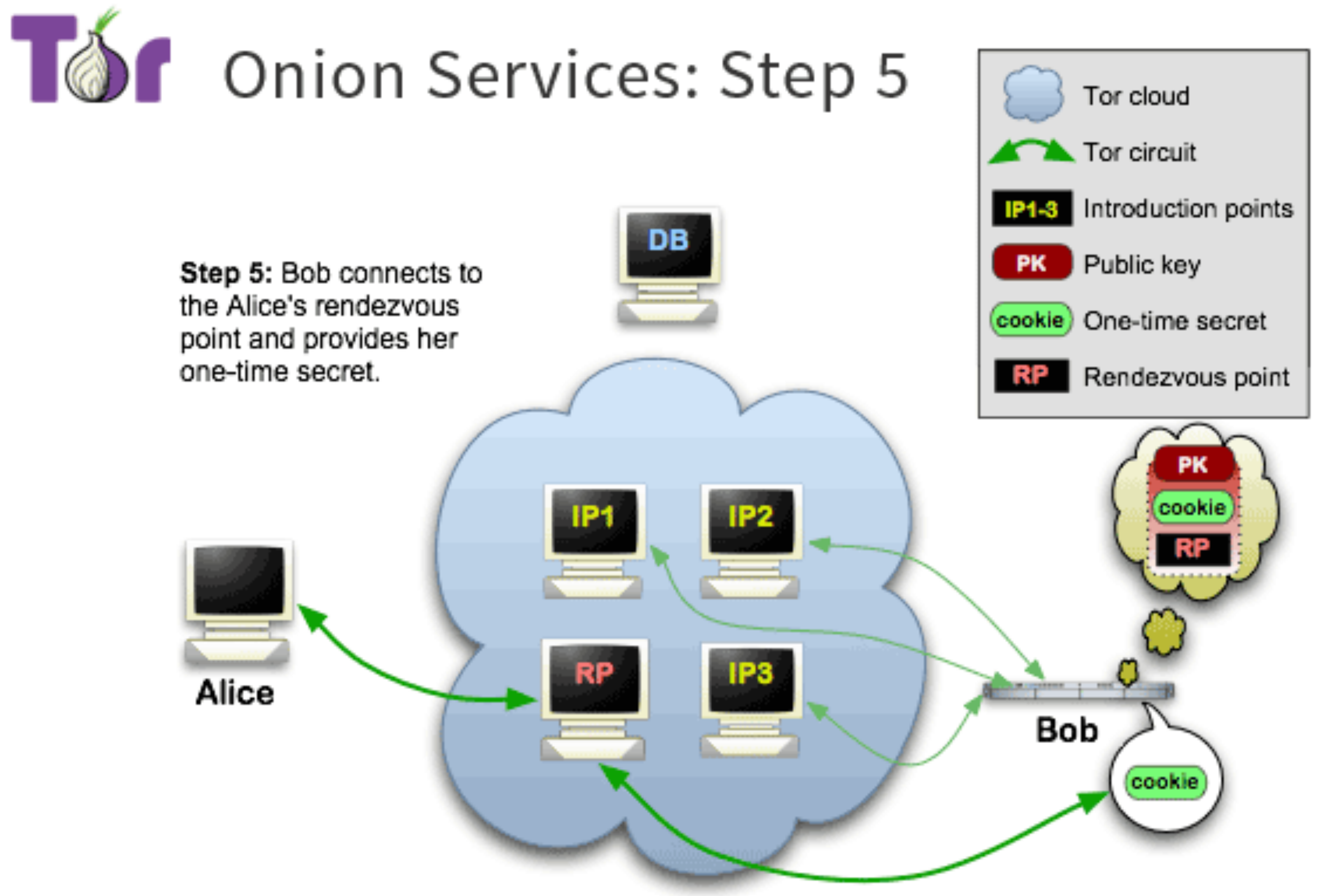
Hidden Servers

- Tor hidden servers hide the server from you.
- See: https://www.torproject.org/docs/hidden-services.html.en


Tor
Onion Services: Step 4
Step 4: Alice writes a message to Bob (encrypted to PK) listing the rendezvous point and a one-time secret, and asks an introduction point to deliver it to Bob.
Tor cloud
Tor circuit
IP1-3 Introduction points
PK Public key
cookie One-time secret
RP Rendezvous point
DB
PK
cookie
RP
IP1
IP2
RP
IP3
Alice
Bob

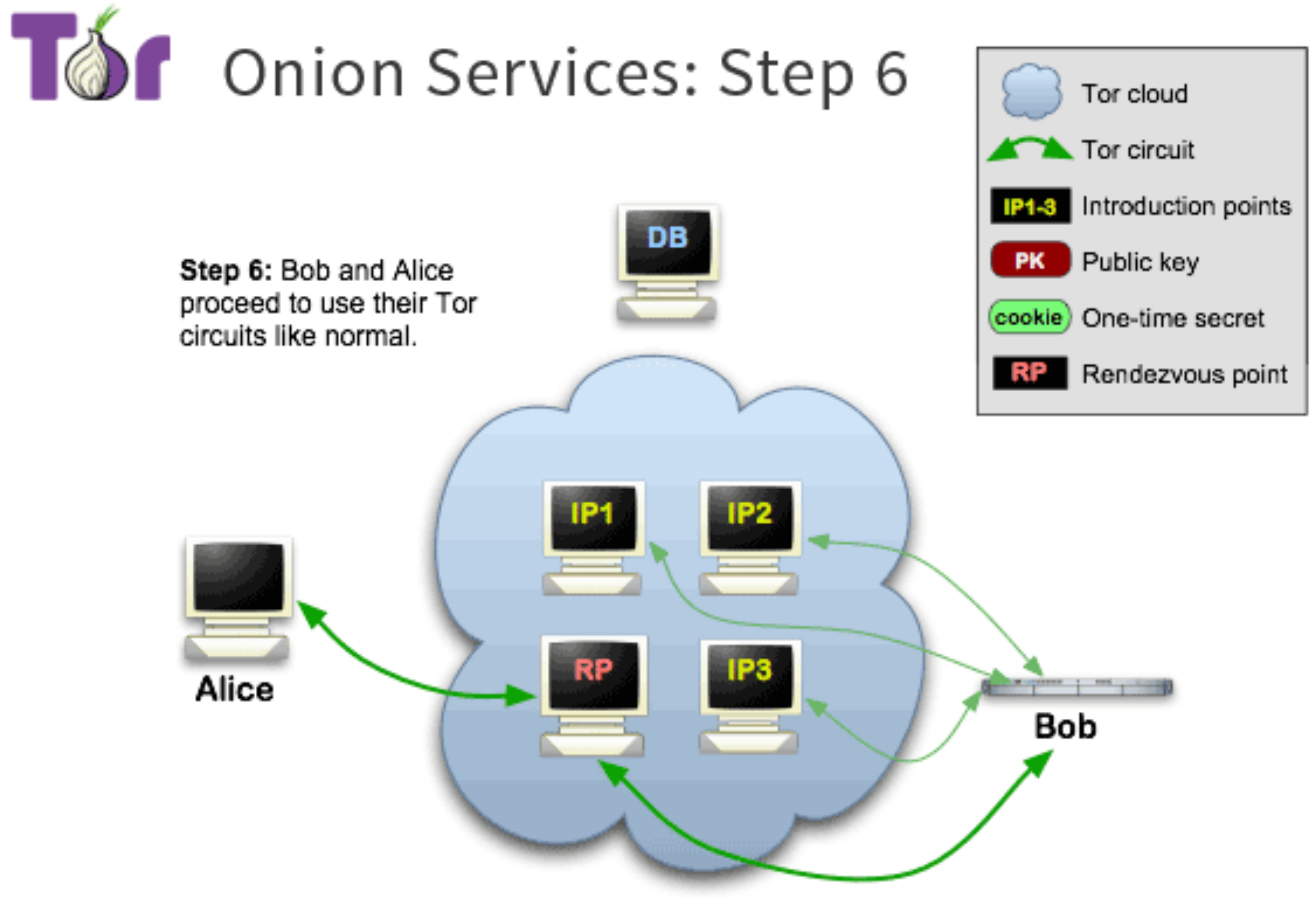
Hidden Servers

- Tor hidden servers hide the server from you.
- See: https://www.torproject.org/docs/hidden-services.html.en



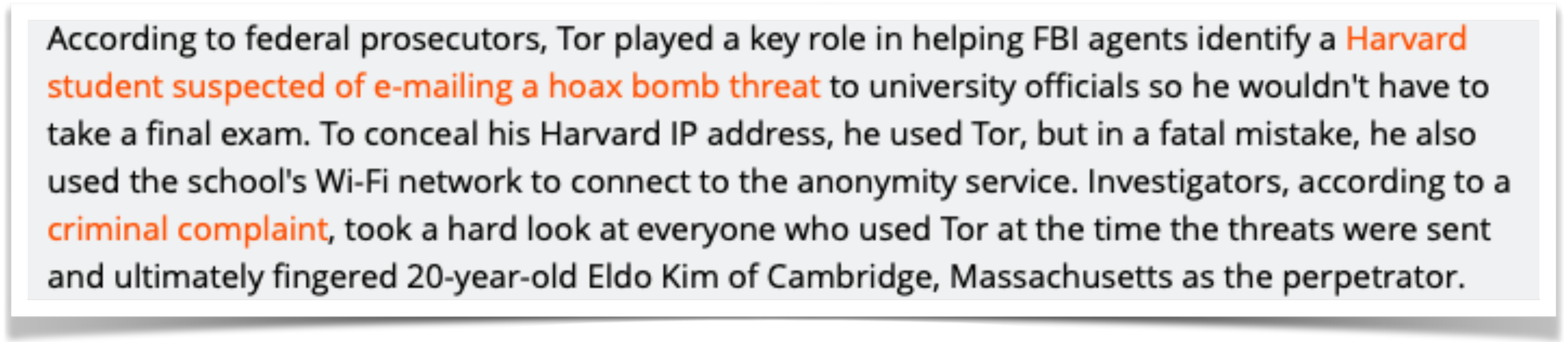
Hidden Servers

- Tor hidden servers hide the server from you.
- See: https://www.torproject.org/docs/hidden-services.html.en

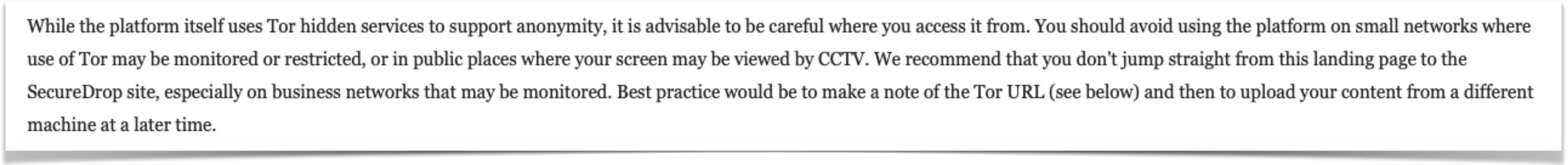


Anonymity does not equal security

- The Webservers can still be attacked.
  - e.g., FBI attacked criminal websites running on Tor and implanted malware.
- Poor use of Tor also


According to federal prosecutors, Tor played a key role in helping FBI agents identify a Harvard student suspected of e-mailing a hoax bomb threat to university officials so he wouldn't have to take a final exam. To conceal his Harvard IP address, he used Tor, but in a fatal mistake, he also used the school's Wi-Fi network to connect to the anonymity service. Investigators, according to a criminal complaint, took a hard look at everyone who used Tor at the time the threats were sent and ultimately fingered 20-year-old Eldo Kim of Cambridge, Massachusetts as the perpetrator.

  - http://arstechnica.com/security/2013/12/use-of-tor-helped-fbi-finger-bomb-hoax-suspect/
  - See warning on: https://www.theguardian.com/securedrop


While the platform itself uses Tor hidden services to support anonymity, it is advisable to be careful where you access it from. You should avoid using the platform on small networks where use of Tor may be monitored or restricted, or in public places where your screen may be viewed by CCTV. We recommend that you don't jump straight from this landing page to the SecureDrop site, especially on business networks that may be monitored. Best practice would be to make a note of the Tor URL (see below) and then to upload your content from a different machine at a later time.

Today’s Lecture

- Details of TLS
  - How it works
  - Common problems
- Tor
  - Anonymity on the internet
  - The Tor Protocol
  - Hidden servers