Security on the Internet: encryption, TLS, certificates

Kevin Scrivnor

COMP 429

Spring 2023

Security

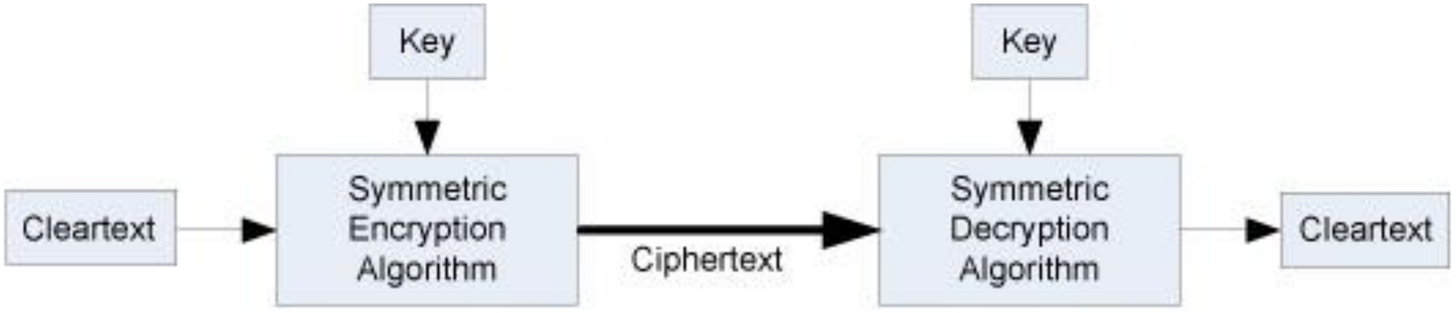
- Broad topic, see COMP 424
- In terms of networking, we are concerned with:
  - Confidentiality – information to intended users only
    - End to end security
  - Integrity – information hasn't been modified
    - Man in the middle
  - Availability – available when needed
    - Denial of Service
    - Cryptolockers
  - Authentication – are you who you say you are?
    - Certificates
  - Non-repudiation – proof of integrity
    - Digital signatures/authentication
  - Auditability - logs
    - forensics

The Real Deal

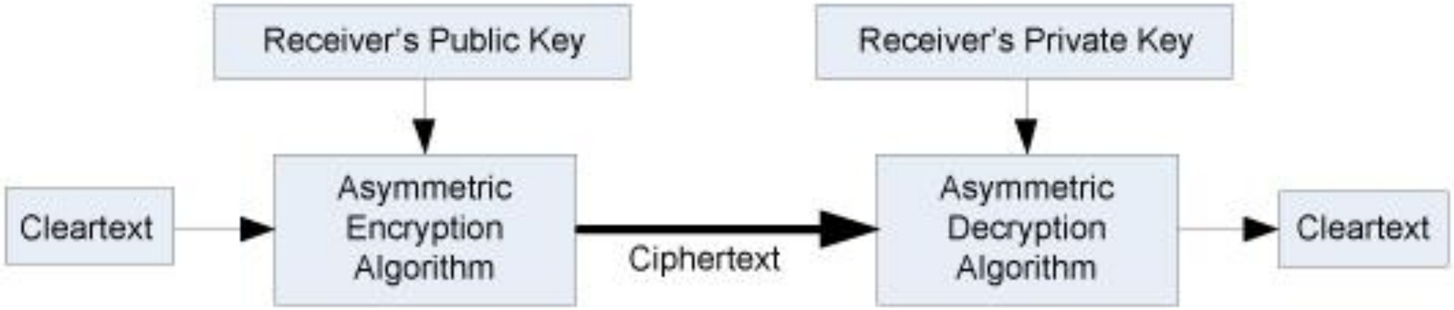
- We will go over, from a high level, the principles behind security on the Internet
- What should you focus on?

  - Construction/use of secure protocols
    - Standards exist, many times required.
    - Libraries exist, use them.
    - There is a correct way and an incorrect way for security. Which changes over time…
  - Key management
    - How are keys created, exchanged, and revoked?
    - What is a certificate?
    - Why do we trust that certificate?
  - Encryption will (eventually) be broken
    - If your encrypted data is compromised, it will eventually be decrypted.

Symmetric Key and Asymmetric Key Systems



Symmetric Key Cryptosystem



Asymmetric (Public Key) Cryptosystem

Which cryptosystem does TLS use?

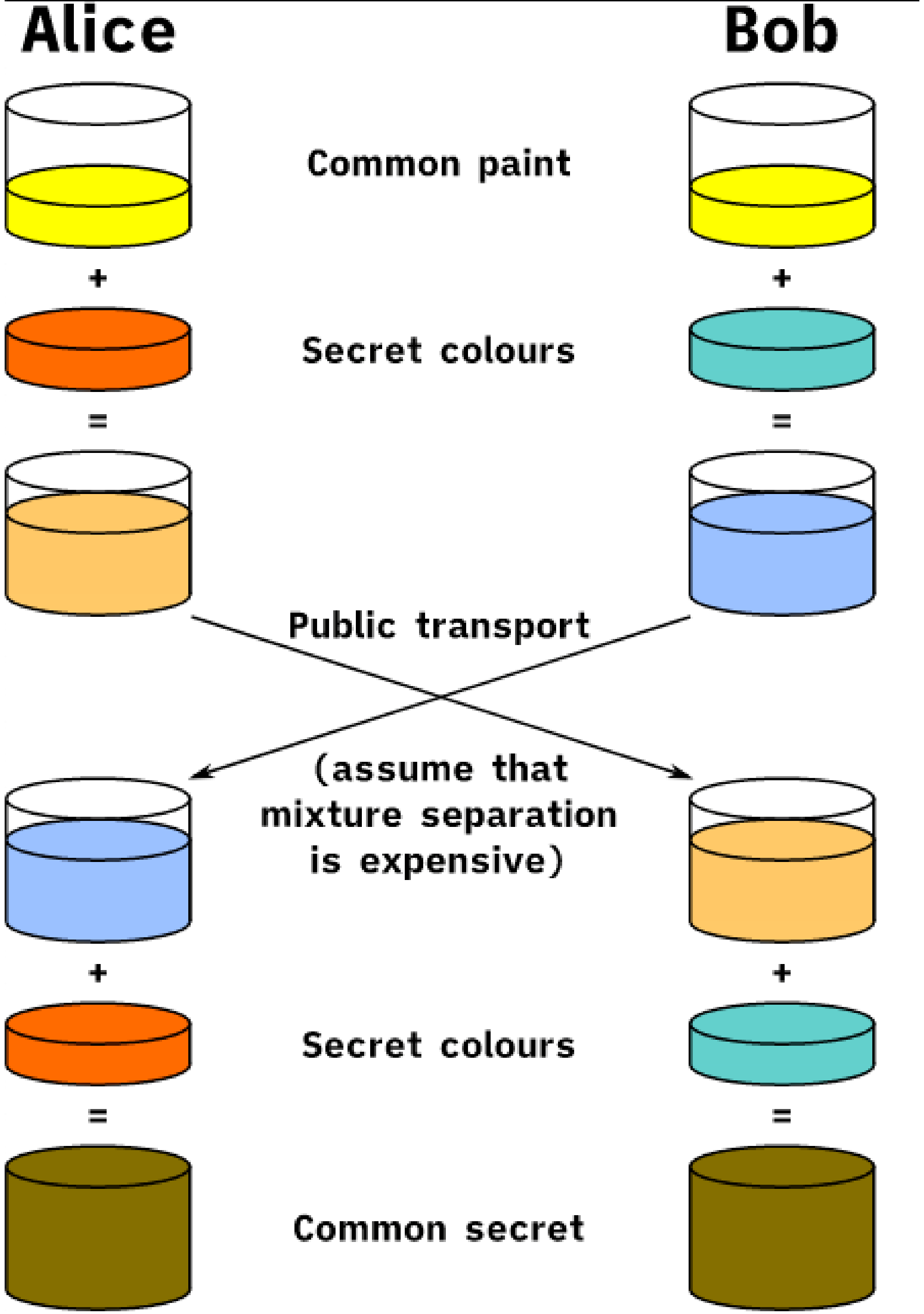
- Well, both.

1. Client contacts server over HTTPS
2. The server responds with its certificate and public key.
3. The client verifies that the certificate is legitimate.
4. The client and server negotiate a cryptographic scheme.
5. The client creates a secret password (session key) and encrypts it with the servers public key, and sends it back to the server.
6. The server decrypts the message with its private key.
7. The session key can now be used to encrypt and decrypt data transmitted during the session.

The evolving story: we want a client/server to talk to each other in some secure manner over a channel any evil person can monitor…

- How does the public key exchange work?
- What is a certificate and why do we trust it?
- How does the encryption work?
- How does a client/server “decide” what schemes to use?
- How does it all tie in to TLS?

Public Key Exchange Algorithm with paints



Challenges with Public Key Systems

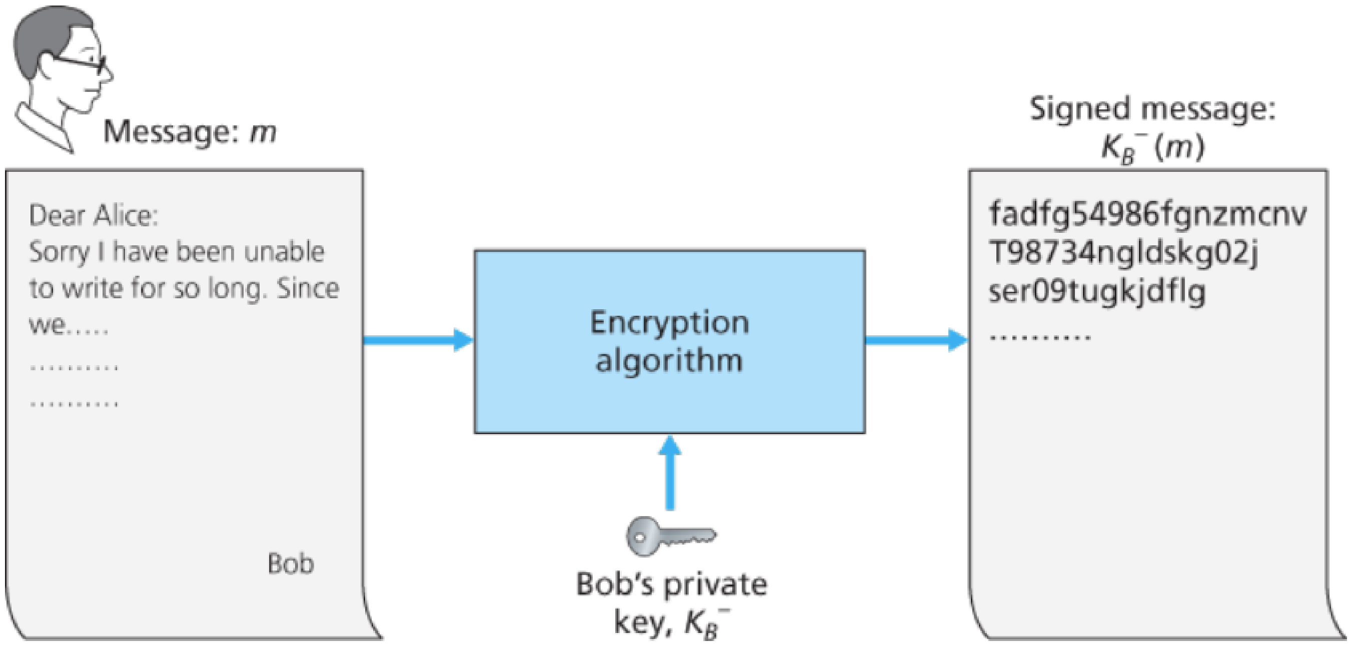
- How do we know the public key *belongs* to who we think it belongs to? Can't anyone make a public key?

- The server can "digitally sign" a "**certificate**" with its private key. Which means if you can decrypt it with the advertised public key, then it *must* be from that server.

- Yes but can't anyone make a public/private key pair?
- Two solutions:
  - Web of trust
    - Trust any keys signed by endorsers.
  - Public Key Infrastructure
    - **Certificate Authority (CA) manages creating, distribution, revoking, and updating key pairs and certs.**

Signing a Certificate

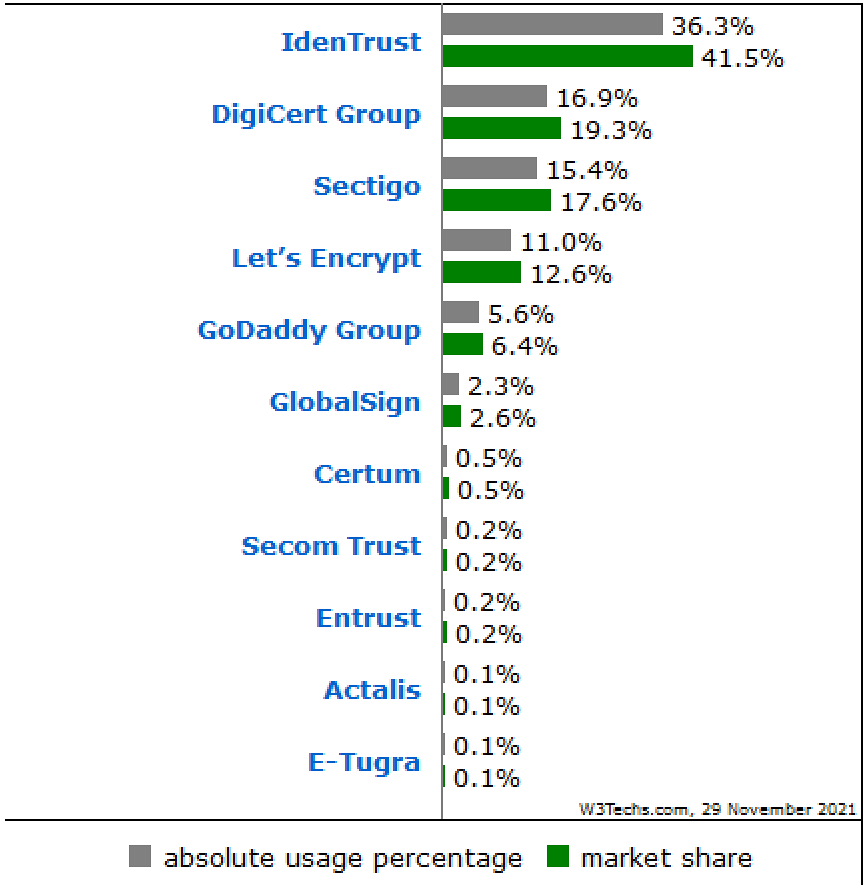
- The signed message can be decrypted using Bob's known and available public key
- Since private keys are supposed to be…private, then it is (mathematically) impossible for anyone to fake this signature



What is a Certificate Authority then?

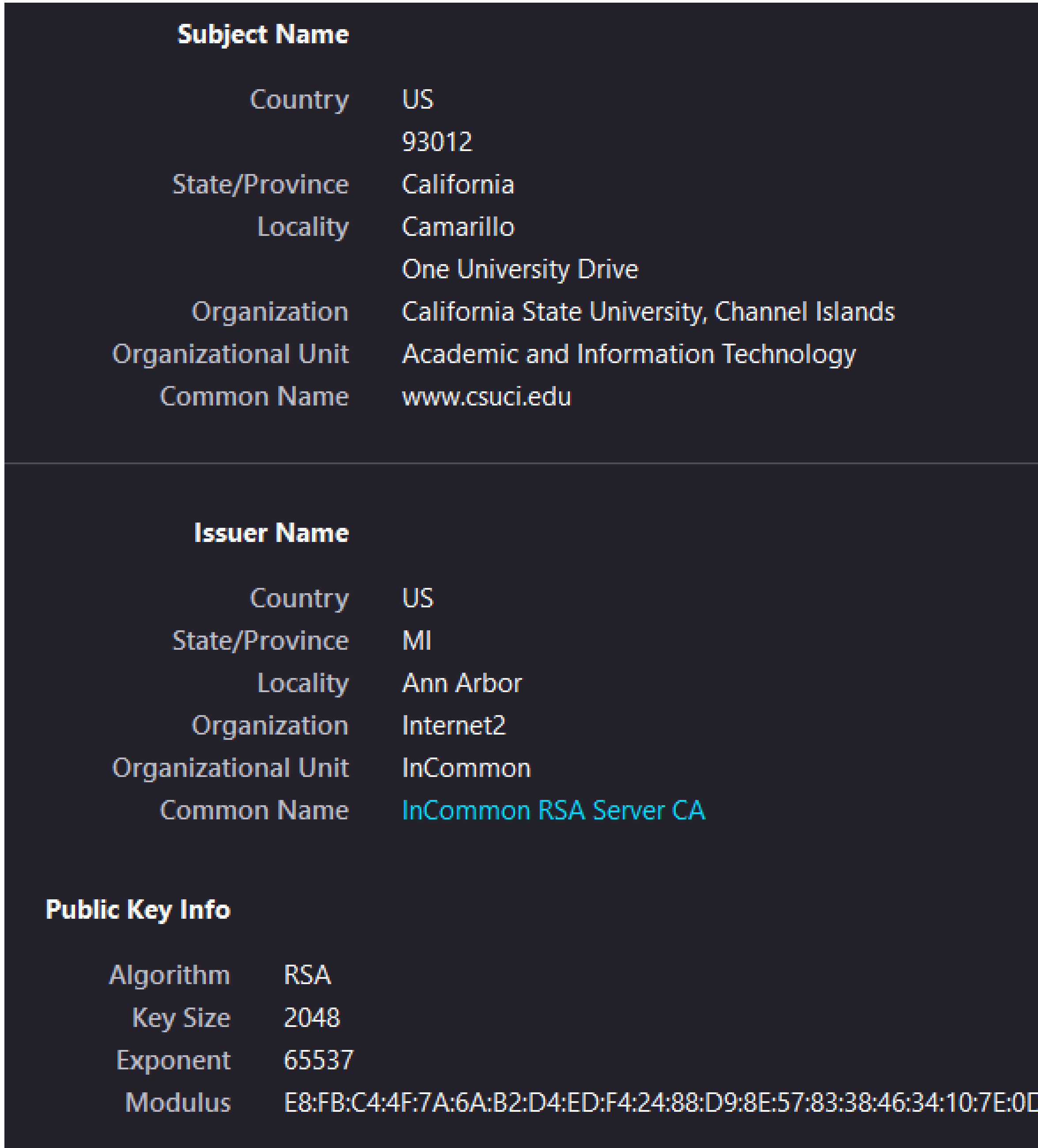
- There are a few companies who you can pay money to issue you a valid certificate.
- Those companies then distribute updated lists to all the browsers/computers, including revocation lists.

- *The idea is that we must trust these companies to only digitally sign public keys from entities THEY trust.*
  - They may require a notary or other in person verification.



What does a cert look like?

- This is *some* of the information presented in the certificate.
- The subject (www.csuci.edu). The domain csuci.edu is not on the list, so you must always include www. (try to goto https://csuci.edu)
- The issuer (the company that signed our key).
- The public key which can be used for encryption.



Ok but what does a cert really look like?


```
-----BEGIN CERTIFICATE-----
MIILCTCCCfGgAwIBAgIRAJo0qR3yxXkTKWKsHyGzq44wDQYJKoZIhvcNAQELBQAw
djELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAk1JMRIwEAYDVQQHEwlBbm4gQXJib3Ix
EjAQBgNVBAoTCUludGVybmV0MjERMA8GA1UECxMISW5Db21tb24xHzAdBgNVBAMT
FkluQ29tbW9uIFJTQSBTZXJ2ZXIgQ0EwHhcNMjAwODAxMDAwMDAwWhcNMjIwODAx
MjM1OTU5WjCB4jELMAkGA1UEBhMCVVMxDjAMBgNVBBETBTkzMDEyMRMwEQYDVQQI
EwpDYWxpZm9ybmlhMRIwEAYDVQQHEwlDYW1hcmlsbG8xHTAbBgNVBAkTFE9uZSBV
bml2ZXJzaXR5IERyaXZlMTUwMwYDVQQKEyxDYWxpZm9ybmlhIFN0YXRlIFVuaXZl
cnNpdHksIENoYW5uZWwgSXNsYW5kczEsMCoGA1UECxMjQWNhZGVtaWMgYW5kIElu
…
```

base64 (RFC 4648)

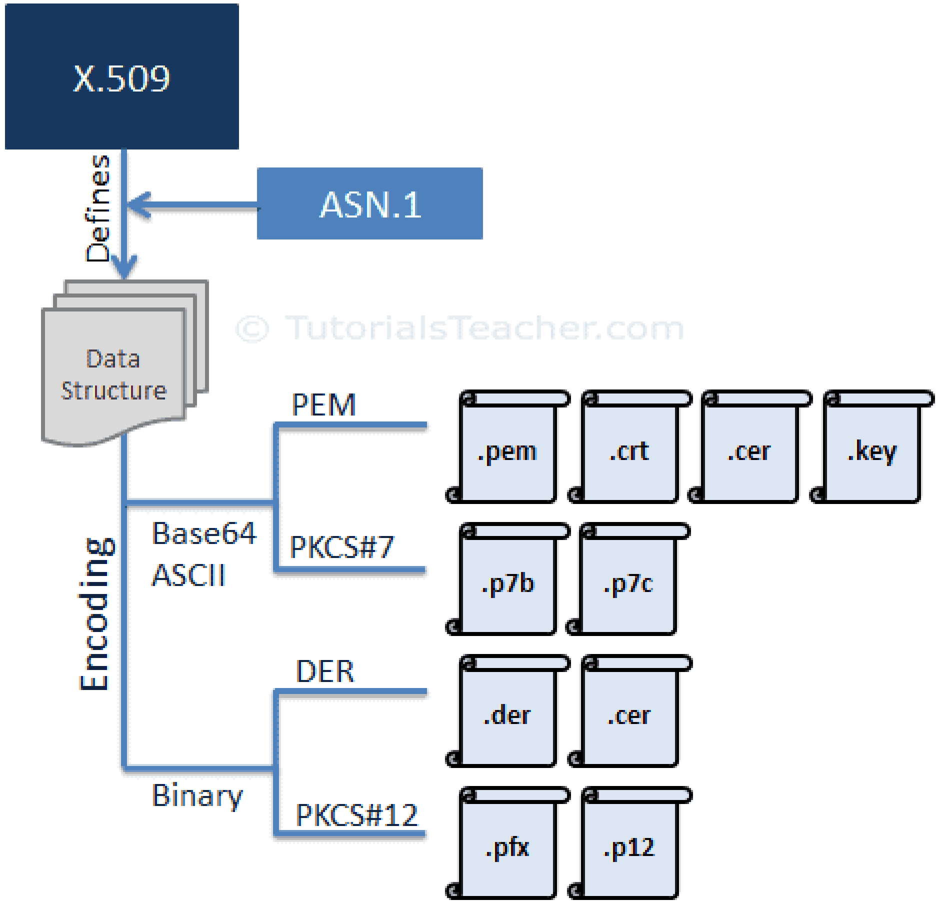
- https://datatracker.ietf.org/doc/html/rfc4648#section-4

- Encodes binary data as printable characters
- Makes it convenient to transmit across the Internet
- Easily copy + paste from an editor, send it an email, etc.

- Remember, it is an encoding, not encryption
  - WW91IHNob3VsZCBqb2luIHRoZSBORVRTRUMgY2x1Yg==

Ok but what does a cert really really look like?

- Well that was it. It is a base64 encoding that can be read by the openssl program.
- The cert shown on the previous page was in the .pem format.



How does the browser then verify the cert?

- So recall that the cert has been digitally signed by a Certificate Authority (CA).
  - (which means the CA (issuer) has encrypted a message with its private key)
- Your browser will then test the digital signature of the issuer with the known public key from that issuer. If it can decrypt the signature, then you should trust it.
- But not yet…
- If a certificate is revoked for some reason, then you must also make sure it is not on the CRL (Certificate Revocation List) from that CA.

The evolving story: we want a client/server to talk to each other in some secure manner over a channel any evil person can monitor…

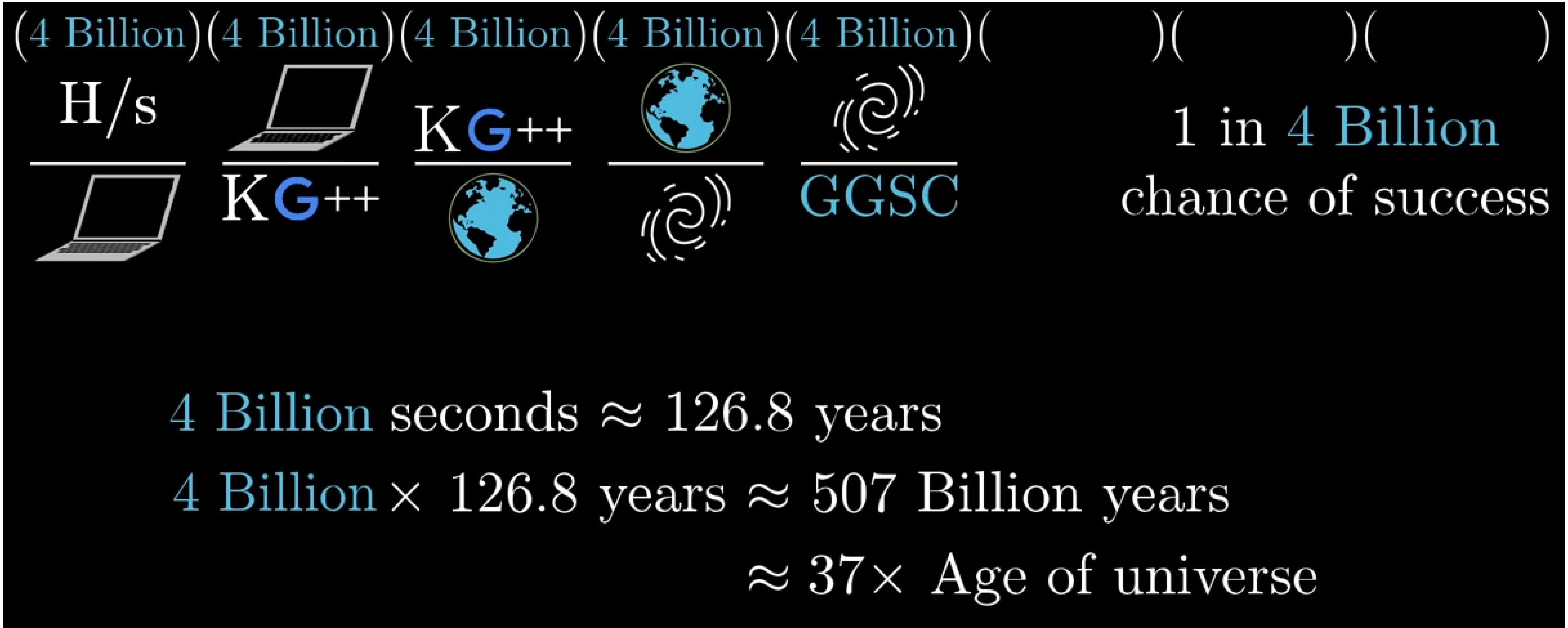
- How does the public key exchange work? Fancy math. Really good random numbers.
- What is a certificate and why do we trust it? A digital signature. We trust Certificate Authorities.
- How does the encryption work?
- How does a client/server “decide” what algorithms to use?
- How does it all tie in to TLS?

If we have a secret password, that is 256 bits, what does that really mean?

- Bits are 1's and 0's. So there are 256 1's and 0's in a string, and that is used as a secret key. Great.
- 256 doesn't sound like very much. Well, really it's 2^256… so how mind boggling large is that number?
    - 2^256 is 2^32 (8 times). 2^32 is roughly 4 billion, which is maybe a little easier to imagine.
    - So 2^256 is
        - (4 billion) (4 billion) (4 billion) (4 billion) (4 billion) (4 billion) (4 billion) (4 billion)

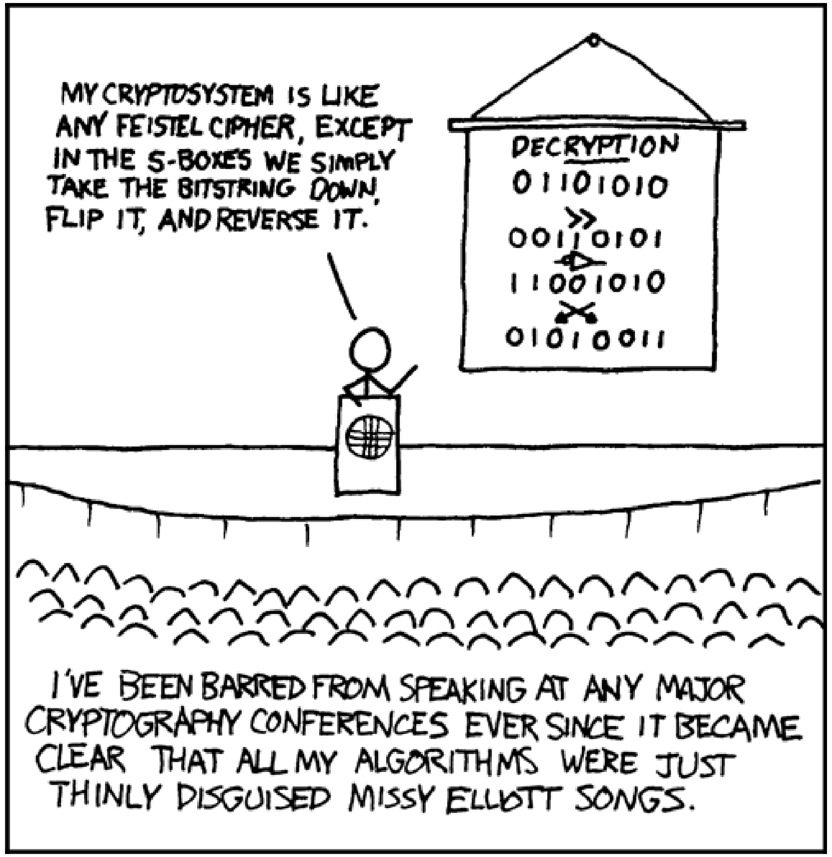
Computationally Impossible (for now)

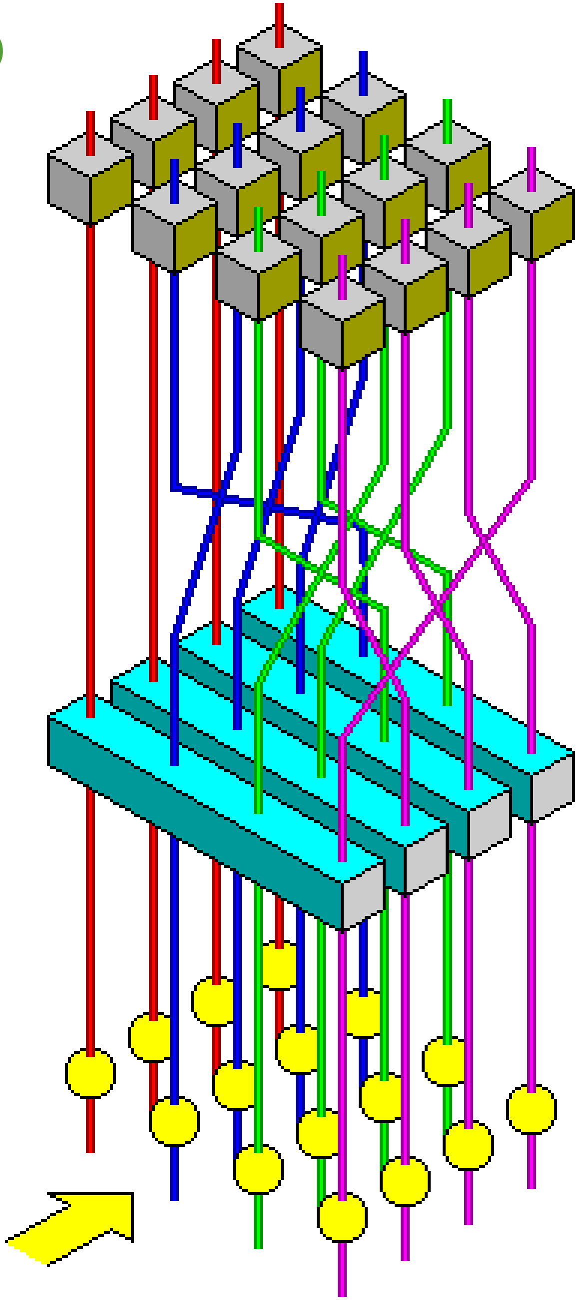
- So if you had a computer that could guess 4 billion hashes per second…
- https://www.youtube.com/watch?v=S9JGmA5_unY



So bits are secure, fine. Now what?

- Divide your data into 256 bit chunks. Perform the round the right. Then do it again 12 more times.
- To decrypt, perform the operation in reverse.





Byte Sub

Shift Row

Mix Column

Add Round Key

The evolving story: we want a client/server to talk to each other in some secure manner over a channel any evil person can monitor…

- How does the public key exchange work? Fancy math. Really good random numbers.
- What is a certificate and why do we trust it? A digital signature. We trust Certificate Authorities.
- How does the encryption work? A key length that would take a multiverse and infinite time to guess, plus shifting, adding, substituting bits over and over with this key.
- How does a client/server "decide" what algorithms to use?
- How does it all tie in to TLS?

We trust who we are talking to and we have some fancy math algorithms, which one and why?

- Well actually, which **ones**.

- Key exchange
  - Decide upon a secret key using an asymmetric algorithm.
- Cipher
  - Likely a block cipher.
  - Used for symmetric data encryption using the secret key.
- Data integrity
  - A MAC (message authentication code).
  - Ensure the encrypted data is valid (not corrupted/modified).

- *The combination of these algorithms are referred to as a cipher suite.*

Forward Secrecy

- An important and desired property in cryptography.

- **The session key generated by a set of public/private key pairs will not be derivable if *one* of the private keys is compromised in the future.**

- About 80% of TLS-enabled websites are **configured** to use forward secrecy.

Key Exchange Algorithms and Availability in TLS

Key exchange/agreement and authentication

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
RSA	Yes	Yes	Yes	Yes	Yes	No	Defined for TLS 1.2 in RFCs
DH-RSA	No	Yes	Yes	Yes	Yes	No	
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes	
ECDH-RSA	No	No	Yes	Yes	Yes	No	
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
DH-DSS	No	Yes	Yes	Yes	Yes	No	
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[51]	
ECDH-ECDSA	No	No	Yes	Yes	Yes	No	
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDH-EdDSA	No	No	Yes	Yes	Yes	No	
ECDHE-EdDSA (forward secrecy)[52]	No	No	Yes	Yes	Yes	Yes	
PSK	No	No	Yes	Yes	Yes		
PSK-RSA	No	No	Yes	Yes	Yes		
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
SRP	No	No	Yes	Yes	Yes		
SRP-DSS	No	No	Yes	Yes	Yes		
SRP-RSA	No	No	Yes	Yes	Yes		
Kerberos	No	No	Yes	Yes	Yes		
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes		
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes		
GOST R 34.10-94 / 34.10-2001[53]	No	No	Yes	Yes	Yes		Proposed in RFC drafts

Block cipher algorithms available in TLS

Cipher		Protocol version					
Algorithm	Nominal strength (bits)	SSL 2.0	SSL 3.0 [n 1][n 2][n 3][n 4]	TLS 1.0 [n 1][n 3]	TLS 1.1 [n 1]	TLS 1.2 [n 1]	TLS 1.3
AES GCM[54][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	Secure
AES CCM[55][n 5]		N/A	N/A	N/A	N/A	Secure	Secure
AES CBC[n 6]		N/A	Insecure	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A
Camellia GCM[56][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	N/A
Camellia CBC[57][n 6]		N/A	Insecure	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A
ARIA GCM[58][n 5]	256, 128	N/A	N/A	N/A	N/A	Secure	N/A
ARIA CBC[58][n 6]		N/A	N/A	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A
SEED CBC[59][n 6]	128	N/A	Insecure	Depends on mitigations	Depends on mitigations	Depends on mitigations	N/A
3DES EDE CBC[n 6][n 7]	112[n 8]	Insecure	Insecure	Insecure	Insecure	Insecure	N/A
GOST 28147-89 CNT[53][n 7]	256	N/A	N/A	Insecure	Insecure	Insecure	N/A
IDEA CBC[n 6][n 7][n 9]	128	Insecure	Insecure	Insecure	Insecure	N/A	N/A
DES CBC[n 6][n 7][n 9]	56	Insecure	Insecure	Insecure	Insecure	N/A	N/A
	40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A
RC2 CBC[n 6][n 7]	40[n 10]	Insecure	Insecure	Insecure	N/A	N/A	N/A

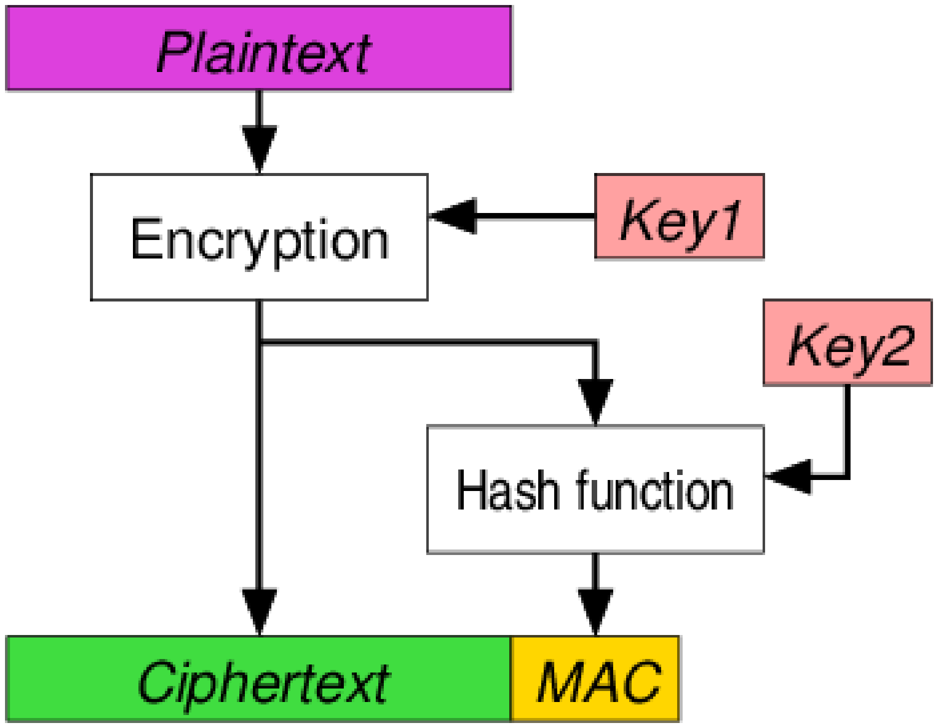
Data integrity algorithms available

Data integrity

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
HMAC-MD5	Yes	Yes	Yes	Yes	Yes	No	Defined for TLS 1.2 in RFCs
HMAC-SHA1	No	Yes	Yes	Yes	Yes	No	
HMAC-SHA256/384	No	No	No	No	Yes	No	
AEAD	No	No	No	No	Yes	Yes	
GOST 28147-89 IMIT[53]	No	No	Yes	Yes	Yes		Proposed in RFC drafts
GOST R 34.11-94[53]	No	No	Yes	Yes	Yes		

Return of the MAC

- Just one approach shown, basic idea is the same for other approaches



Cipher suite

- A cipher suite is a string defining which algorithms will be used.
- TLS 1.2 has 37 cipher suites available. Many choices, little guidance.

- TLS 1.3 only recommends 5 cipher suites:
  - PROTOCOL_CIPHER_HASH

  - TLS_AES_128_GCM_SHA256
  - TLS_AES_256_GCM_SHA384
  - TLS_CHACHA20_POLY1305_SHA256
  - TLS_AES_128_CCM_SHA256
  - TLS_AES_128_CCM_8_SHA256

- Which algorithm is missing?

How to pick?

- Not as many options for TLS 1.3, why?
- TLS, while theoretically secure, can be configured to insecure!

- Don't know what you're doing? https://ssl-config.mozilla.org/
- Why use intermediate over modern? compatibility

```
    # intermediate configuration
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-
ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384;
    ssl_prefer_server_ciphers off;
```

```
    # modern configuration
    ssl_protocols TLSv1.3;
    ssl_prefer_server_ciphers off;
```

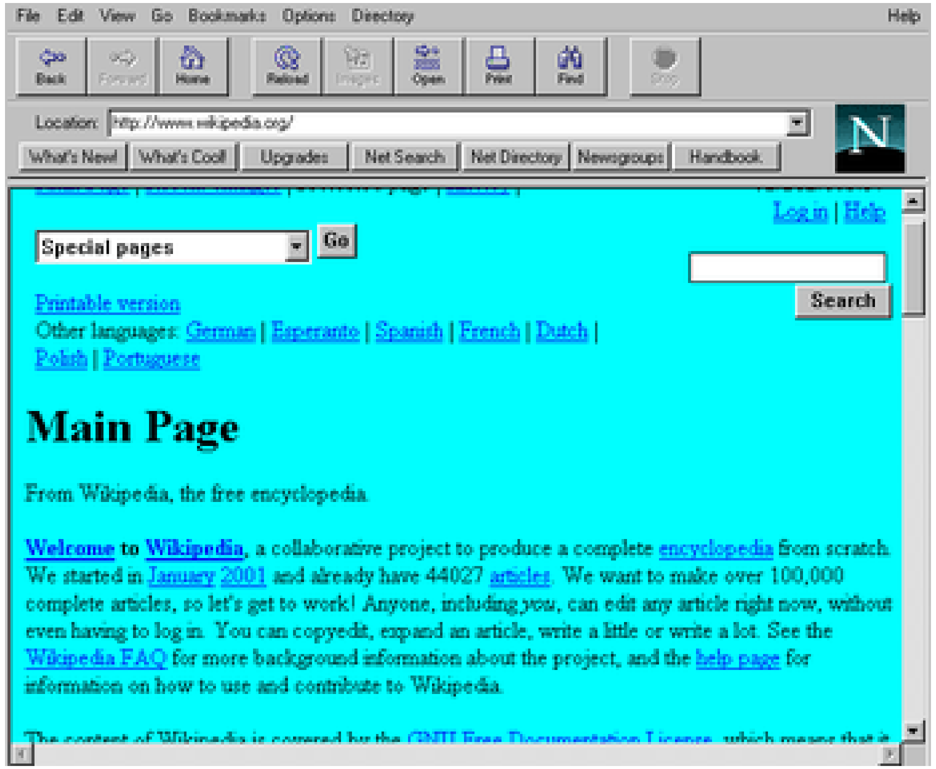
The evolving story: we want a client/server to talk to each other in some secure manner over a channel any evil person can monitor…

- How does the public key exchange work? Fancy math. Really good random numbers.
- What is a certificate and why do we trust it? A digital signature. We trust Certificate Authorities.
- How does the encryption work? A key length that would take a multiverse and infinite time to guess, plus shifting, adding, substituting bits over and over with this key.
- How does a client/server “decide” what algorithms to use? Chooses from a list described by the standard. Probably during a handshake.
- How does it all tie in to TLS?

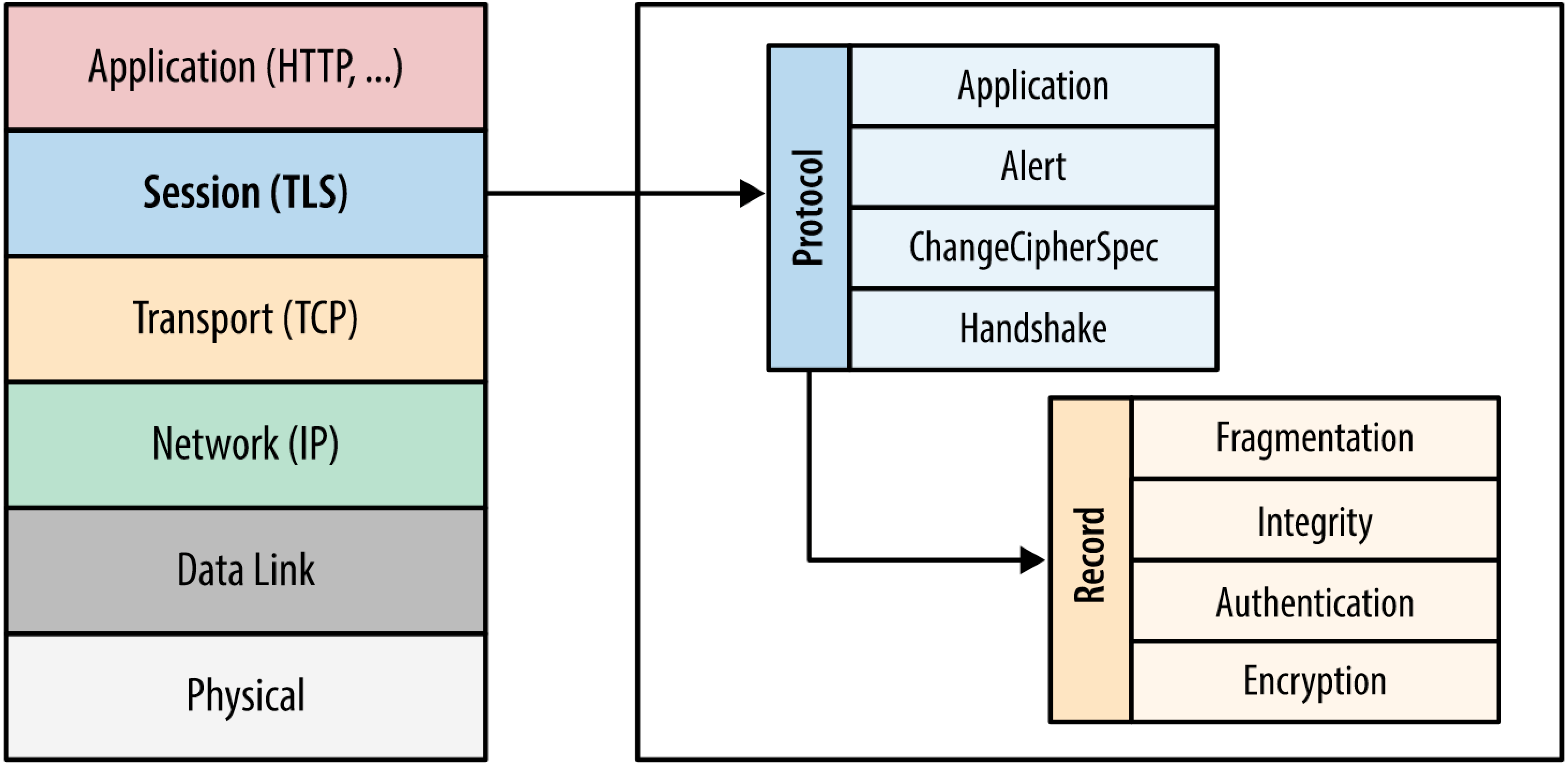
Transport Layer Security (TLS)

- TLS is not SSL (secure socket layer) as SSL is considered insecure now.
- However, SSL is often used interchangeably with TLS (openssl cmd)

- Originally developed at Netscape to secure ecommerce communications.
- SSL 1.0 not publicly released
- SSL 2.0 released in 1995, deprecated 2011
- SSL 3.0 released in 1996, deprecated 2015

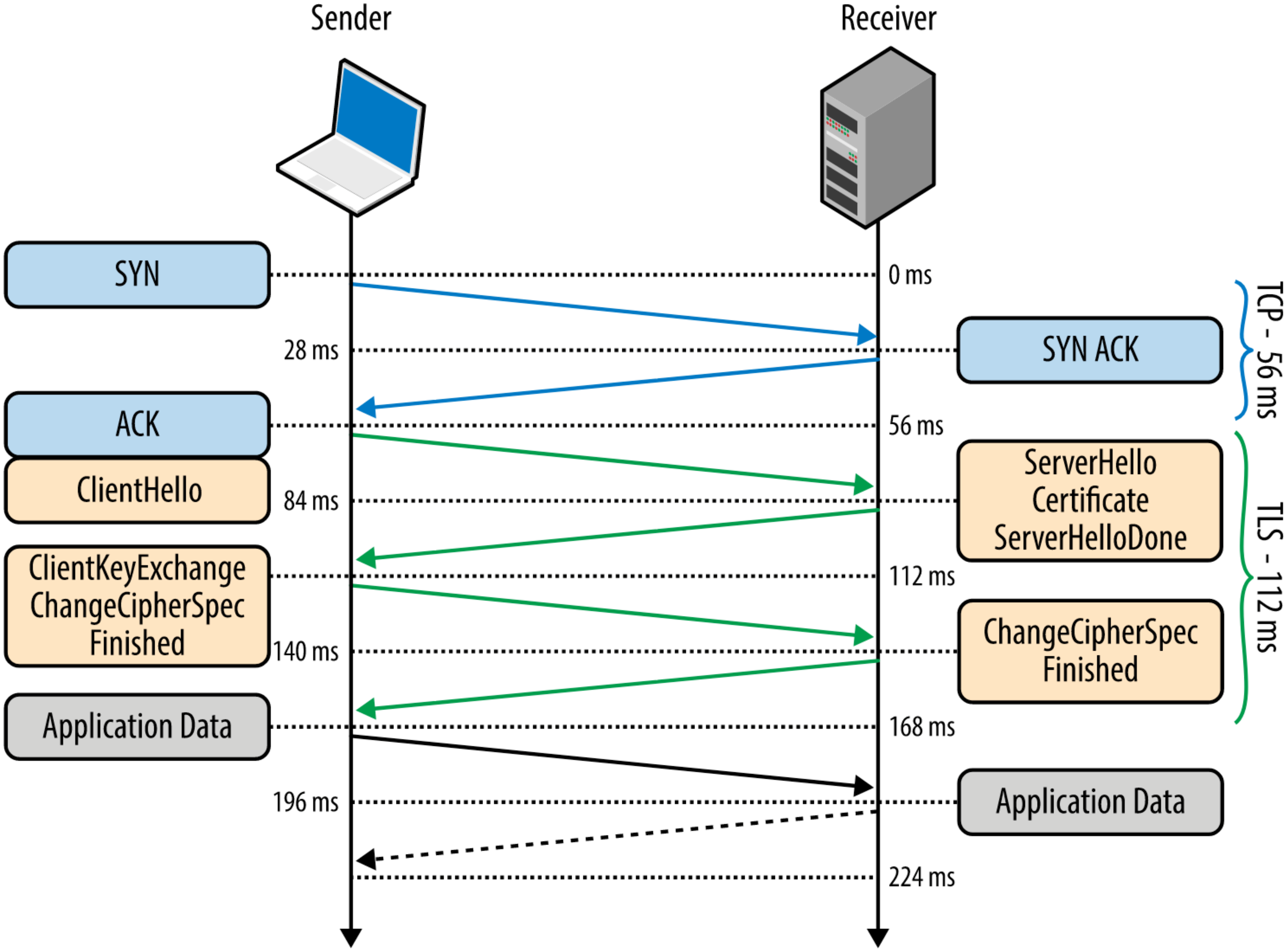


HTTPS is a combination of HTTP and TLS



Another Handshake

- Say Hello in TLS, offer cipher suites
- Respond with certificate, agree on cipher suite, perform key exchange
- Application data may now be encrypted with the agreed upon cipher



What makes TLS secure

- Generating correct certificates, validating certificates and using modern ciphers.
- A study in 2017 analyzed TLS intercepted traffic and graded the following products based on their behavior:
  - https://jhalderm.com/pub/papers/interception-ndss17.pdf

Product	Grade	Validates Certificates	Modern Ciphers	Advertises RC4	TLS Version	Grading Notes
A10 vThunder SSL Insight	F	✓	✓	Yes	1.2	Advertises export ciphers
Blue Coat ProxySG 6642	A*	✓	✓	No	1.2	Mirrors client ciphers
Barracuda 610Vx Web Filter	C	✓	✗	Yes	1.0	Vulnerable to Logjam attack
Checkpoint Threat Prevention	F	✓	✗	Yes	1.0	Allows expired certificates
Cisco IronPort Web Security	F	✓	✓	Yes	1.2	Advertises export ciphers
Forcepoint TRITON AP-WEB Cloud	C	✓	✓	No	1.2	Accepts RC4 ciphers
Fortinet FortiGate 5.4.0	C	✓	✓	No	1.2	Vulnerable to Logjam attack
Juniper SRX Forward SSL Proxy	C	✓	✗	Yes	1.2	Advertises RC4 ciphers
Microsoft Threat Mgmt. Gateway	F	✗	✗	Yes	SSLv2	No certificate validation
Sophos SSL Inspection	C	✓	✓	Yes	1.2	Advertises RC4 ciphers
Untangle NG Firewall	C	✓	✗	Yes	1.2	Advertises RC4 ciphers
WebTitan Gateway	F	✗	✓	Yes	1.2	Broken certificate validation

Fig. 3: **Security of TLS Interception Middleboxes**—We evaluate popular network middleboxes that act as TLS interception proxies. We find that nearly all reduce connection security and five introduce severe vulnerabilities. *Mirrors browser ciphers.

TLS Interception

- NAT is transparent to clients. We send traffic out and traffic comes back as if nothing was changed. This change is detectable though!
- The same can happen with TLS and a proxy server. The traffic is terminated at the proxy and then re-established to the server. This means that the proxy can inspect the traffic in plaintext.
- https://amibehindaproxy.com/
- This is good for
  - Malware analysis
  - Corporate networks enforcing acceptable use policies
  - Compliance based on legal requirements
- This is bad for
  - Privacy
  - Another point that could be misconfigured

How it works (pictorially)