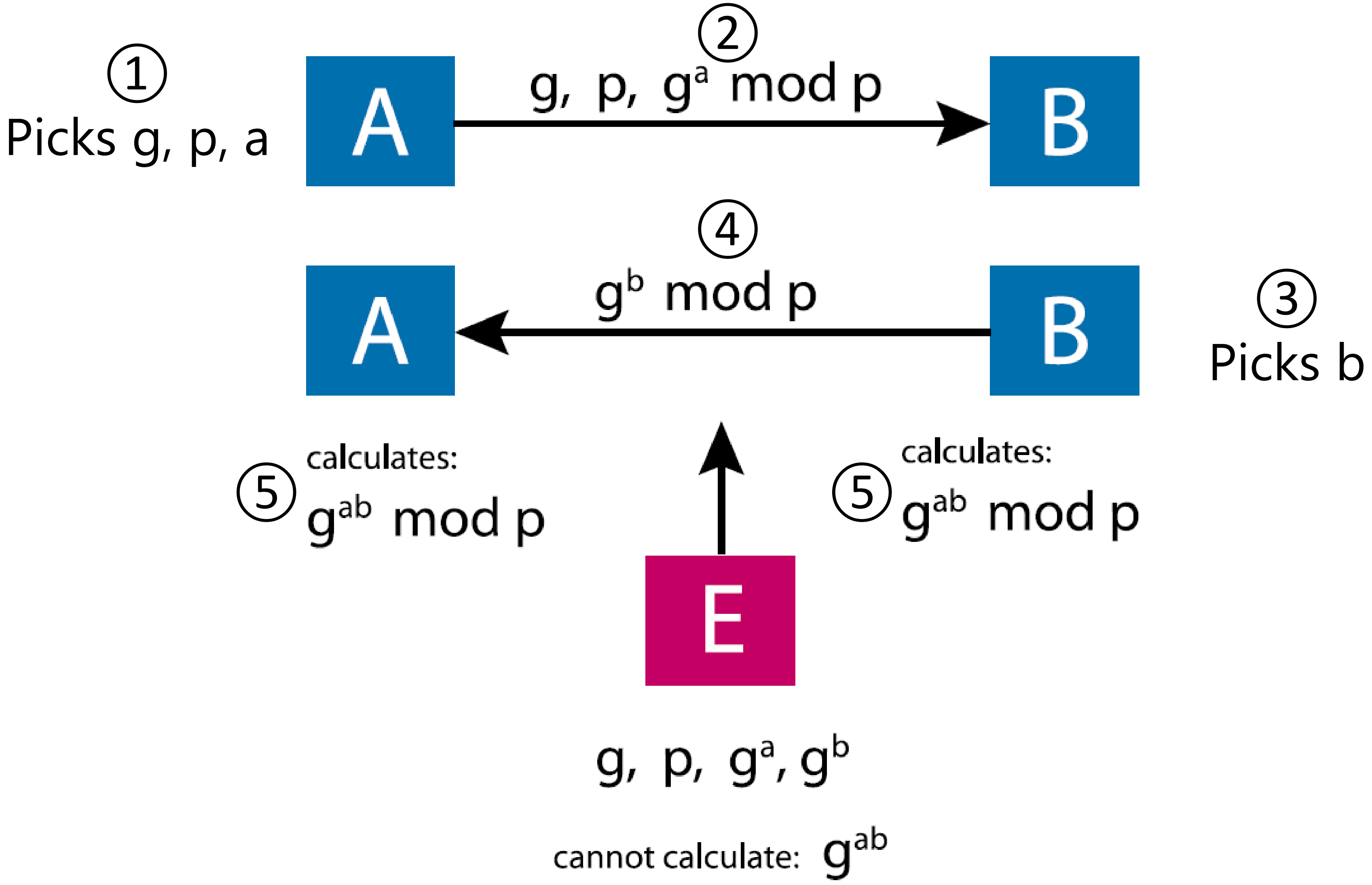
Week 6

DH attacks, Certificates, SSL/TLS

THE UNIVERSITY OF SYDNEY

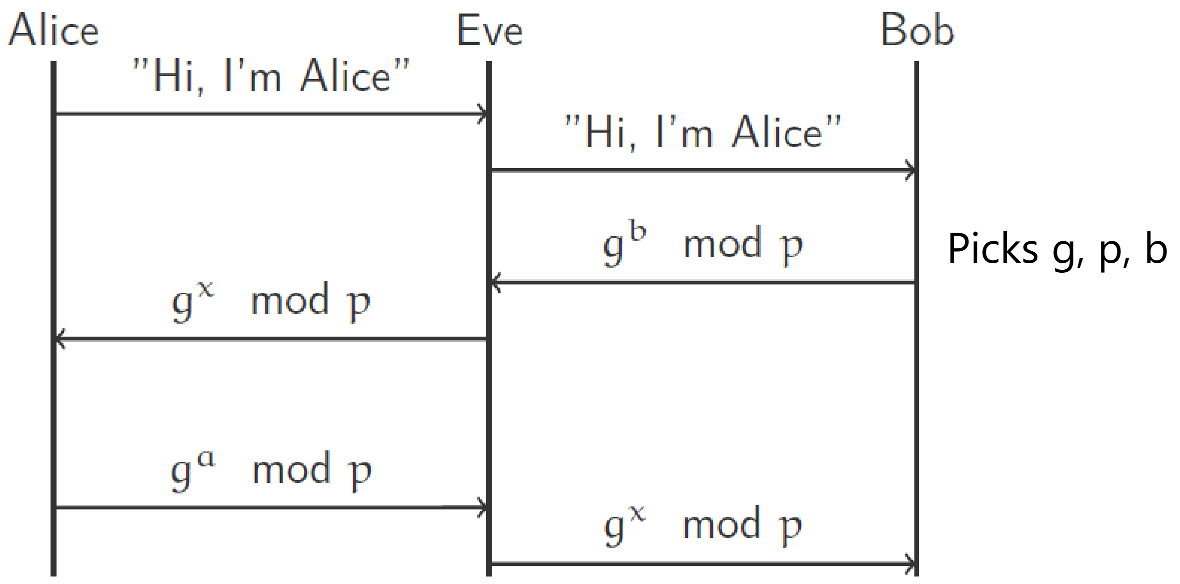
Review: Diffie–Hellman Key Exchange



① Picks g, p, a
A
② g, p, g^a mod p
B
④ g^b mod p
A
B
③ Picks b
⑤ calculates: g^{ab} mod p
⑤ calculates: g^{ab} mod p
E
g, p, g^a, g^b
cannot calculate: g^{ab}

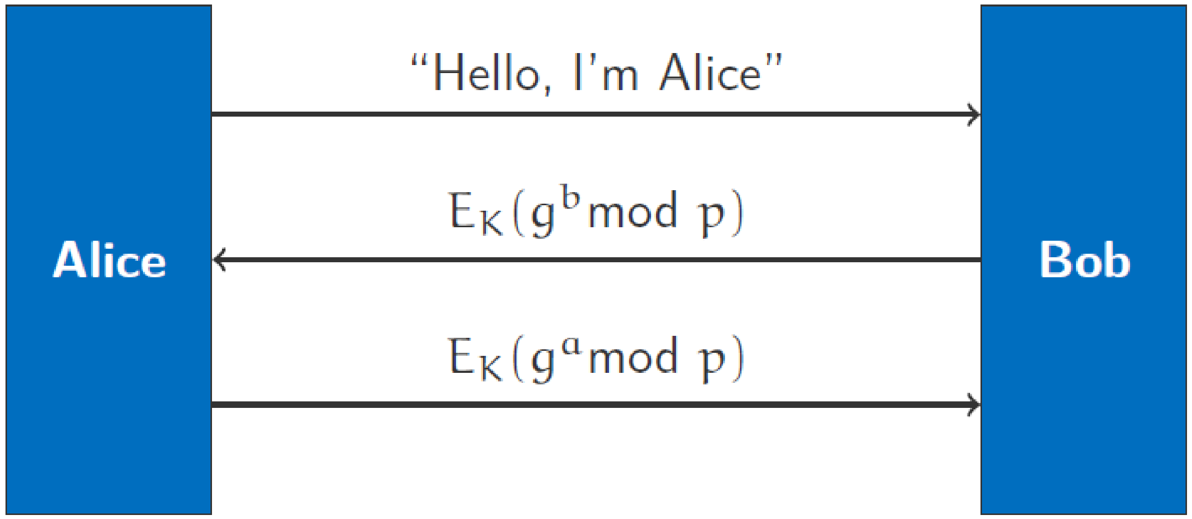
Diffie-Hellman Man-in-the-Middle (MITM) Attack

- Suppose Alice (A) and Bob (B) are performing Diffie–Hellman key exchange, but a third party, Eve (E) owns the channel and can **intercept traffic.**



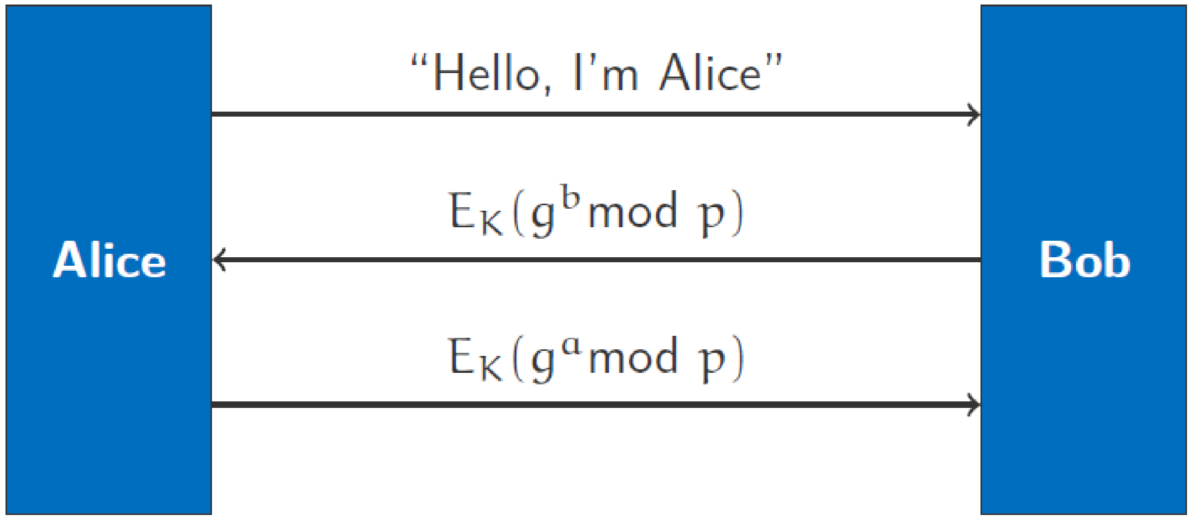
  - Shared secret for Alice/Eve is $g^{ax} \bmod p$.
  - Shared secret for Eve/Bob is $g^{bx} \bmod p$.
  - Eve owns the channel and neither Alice nor Bob know!

EKE – Encrypted Key Exchange



- The shared secret is $g^{ab} \bmod p$. A (possibly low entropy) shared key K is used to form the high entropy shared secret.
- Prevents eavesdropping, and active MITM attacks on DH. Maintains **forward secrecy**.
- What is the problem?

EKE – Encrypted Key Exchange



- The shared secret is $g^{ab} \bmod p$. A (possibly low entropy) shared key K is used to form the high entropy shared secret.
- Prevents eavesdropping, and active MITM attacks on DH. Maintains **forward secrecy**.
- What is the problem? Alice needs to have a shared key, K, with Bob prior to the key exchange.

Definitions

- A protocol is said to have perfect forward secrecy if disclosure of long-term keys does not compromise past (short term) sessions.
  - Ephemeral keys (such as are generated during Diffie–Hellman) give this automatically.
  - Encrypting everything directly with an RSA public key does not have forward secrecy.
- A protocol is vulnerable to a known-key attack if disclosure of past session keys allows an attacker to compromise future session keys (including actively impersonating).

Needham–Schroeder Protocol

- The Needham–Schroeder protocol facilitates key exchange using a trusted third party (TTP).
- Alice and Bob want to set up a session key for communication. All parties share a key with Trent, the TTP.
  - 1. Alice sends Trent a request to talk to Bob: (A, B, r_A), where r_A is a nonce.
  - 2. Trent sends Alice a session key, and a ticket to give to Bob:
$$E_{k_{AT}}(r_A, B, k_{AB}, E_{k_{BT}}(k_{AB}, A)).$$
  - 3. Alice sends Bob the ticket: $(E_{k_{BT}}(k_{AB}, A), E_{k_{AB}}(s_A))$.
  - 4. Bob challenges Alice: $E_{k_{AB}}(s_A - 1, r_B)$, where r_B is a nonce.
  - 5. Alice responds to Bob's challenge: $E_{k_{AB}}(r_B - 1)$.

- What if Eve gets a hold of k_{AT} somehow?

Needham–Schroeder Protocol

- Problem: Bob has no guarantee that k_{AB} is fresh. Old session keys are valuable, as they do not expire.
  - 1. Suppose Eve manages to get k_{AT} .
  - 2. She can now read all of Alice's messages and impersonate her to everyone else.
  - 3. Alice needs to revoke her key, but the only person that can make this have an effect is Trent.
  - 4. Key revocation is a major problem.

- Needham-Schroeder's problem is that it assumes all users of the system are good guys, and the goal is to keep the bad guys from getting in — the "eggshell" model.

Kerberos

- Kerberos is a protocol which builds on the Needham–Schroeder protocol and allows nodes within an insecure network to prove identity to each other.
- Kerberos provides mutual authentication of both the user and the server.
- Primarily suited for client–server model.
- Kerberos protocol is protected against eavesdropping and replay attacks.
- Extra reading: Designing an Authentication System: A Dialogue in Four Scenes. http://web.mit.edu/kerberos/www/dialogue.html

Public Key Management

THE UNIVERSITY OF SYDNEY

Public Key Management using Certification Authorities

- A public key certificate binds a public key to its owner:
  - 1. Alice sends her public key to the CA (Trent).
  - 2. The CA produces a certificate for Alice.
  - 3. Alice sends her public key and certificate to Bob.
  - 4. Bob verifies the certificate using CA's public key.
  - 5. Bob sends encrypted messages to Alice using her public key.

- Certificate = sign_{CA}[X.500: name, org, address, pub. key, expires, ...]
  - Everyone must be able to verify the CA's public key, so it is shipped with OS's, browsers, etc.
  - The CA is a trusted party: it has the ability to issue signatures to whomever.

Public Key Certificate Generation

- 1. Alice generates a public/private keypair.
- 2. Alice sends the public key to the CA.
- 3. The CA challenges Alice to see if she knows the private key.
- 4. The CA generates a certificate and sends it to Alice.

- Important Note:
- 1. The CA never learns Alice's private key.
- 2. Important for forward secrecy.
- 3. A compromise of the CA can still lead to people pretending to be Alice.

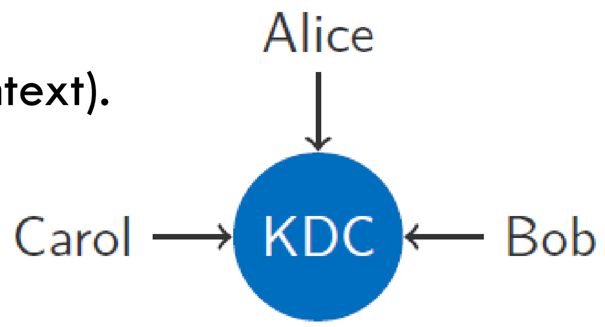
Certificate Revocation

- Alice’s certificate may need to be revoked.
  - Her private key is stolen or otherwise compromised.
  - She changes jobs (or trustworthiness).

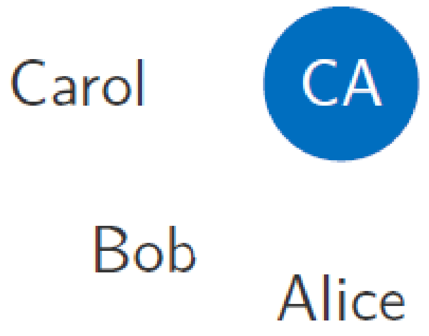
- This is a major problem with the CA system.
  - May require daily certification-validation information (slow, cumbersome).
  - Use expiration date field.
  - Use of a certificate revocation list (CRL) which is circulated (like bad credit cards.)

Trust Models

- Symmetric Keys
  - TTP must be online (used every session).
  - TTP is a juicy target (knows passwords, in plaintext).
  - No forward Secrecy.



- Asymmetric Keys
  - TTP is offline (only used in generation).
  - TTP only knows public keys.
  - TTP has forward secrecy.
  - Not as fast (e.g. SSL/TLS, PGP, ...)



Overview of SSL/TLS

Raw HTTP is Insecure

- In stock standard HTTP, everything goes across the line in plaintext.

- This leaves connections vulnerable to:
  - message tampering
  - eavesdropping on connections
  - man-in-the-middle attacks

- On top of that, there are numerous flaws in the underlying TCP/IP protocols.

Introducing SSL/TLS

- Transport Layer Security (TLS) with its predecessor Secure Socket Layer (SSL) are cryptographic protocols for secure communication. They use:
  - Asymmetric cryptography for authentication
  - Symmetric encryption for confidentiality of messages
  - MACs for integrity

- HTTP sits on top of that to produce HTTPS (HTTP Secure).
  - The only information visible is the IP address and TCP port you're connecting to, as well as the size of the messages you're sending and receiving.
  - While previously only considered for "secure operations", now it's suggested to be used everywhere by default.

TLS Protocol

- Client → Server: (SSL version, available ciphers, other info)
- Server → Client: (SSL version, selected cipher, certificate)

- Client authenticates the server (messages sent should be signed by cert).
- Client (possibly in conjunction with server, depending on cipher) creates the pre-master secret for the session, encrypts using server's cert, sends to the server.
- Random bit strings are included in the exchanged messages to prevent replay attacks.
- (Optional) Server may authenticate the client using client's certificate.
- All messages from this point are encrypted.

Where do the Certificates come from?

- How do we decide whether to trust the certificate the server sends?
  - A Certification Authority (CA) is a trusted third party that issues digital certificates.
  - Certificates for CAs are shipped with operating systems and browsers, and other software.
  - Each time a server sends a certificate to a browser, the browser will check to see if one of the CAs it trusts has signed the certificate.

- Who are these CAs?
- Small number of multinational companies, with a significant barrier to entry.
- Examples:
  - https://www.digicert.com
  - https://letsencrypt.org
  - https://www.globalsign.com/en
  - https://www.verisign.com/

Issues with CAs

- Do we actually trust them?
  - Many have poor security practices and are willing to co-operate with governments.
  - The small number of root CAs allow other CAs to sign on their behalf.
  - In 2011, 9 fraudulent certificates were obtained from Comodo for Google, Yahoo! Mail, Hotmail, Skype etc. The attackers could have performed man-in-the-middle attacks for any of the compromised websites.

- They can be expensive:
  - Cost of 128-bit and 256-bit certificates are noticeably different, even though next to no extra work goes into the latter.

- Saying “this website secured by SSL” gives a false sense of confidence.
  - Very easy for scammers to get themselves an SSL certificate for a domain they own.

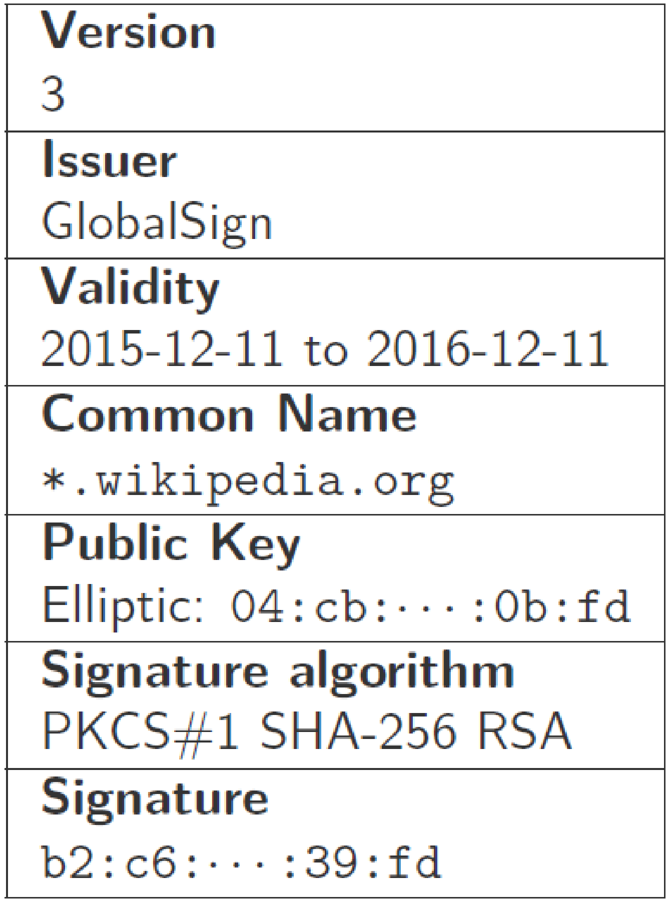
How do you acquire a certificate?

- Domain Validation: The CA determines you own the domain by one of:
  - Having you respond to an email sent to admin@, postmaster@, etc.
  - Having you publish a certain DNS TXT record.
  - … and more.
  - Issue with domain validation: do any of the above methods actually prove that you own the domain?

- Extended Validation (EV) certificates
  - CA verifies you pass a set of identity verification criteria.
  - Limits the certificate validity period to ensure the certificate information is up to date.
  - Costs more than a normal certificate.

How does it work?

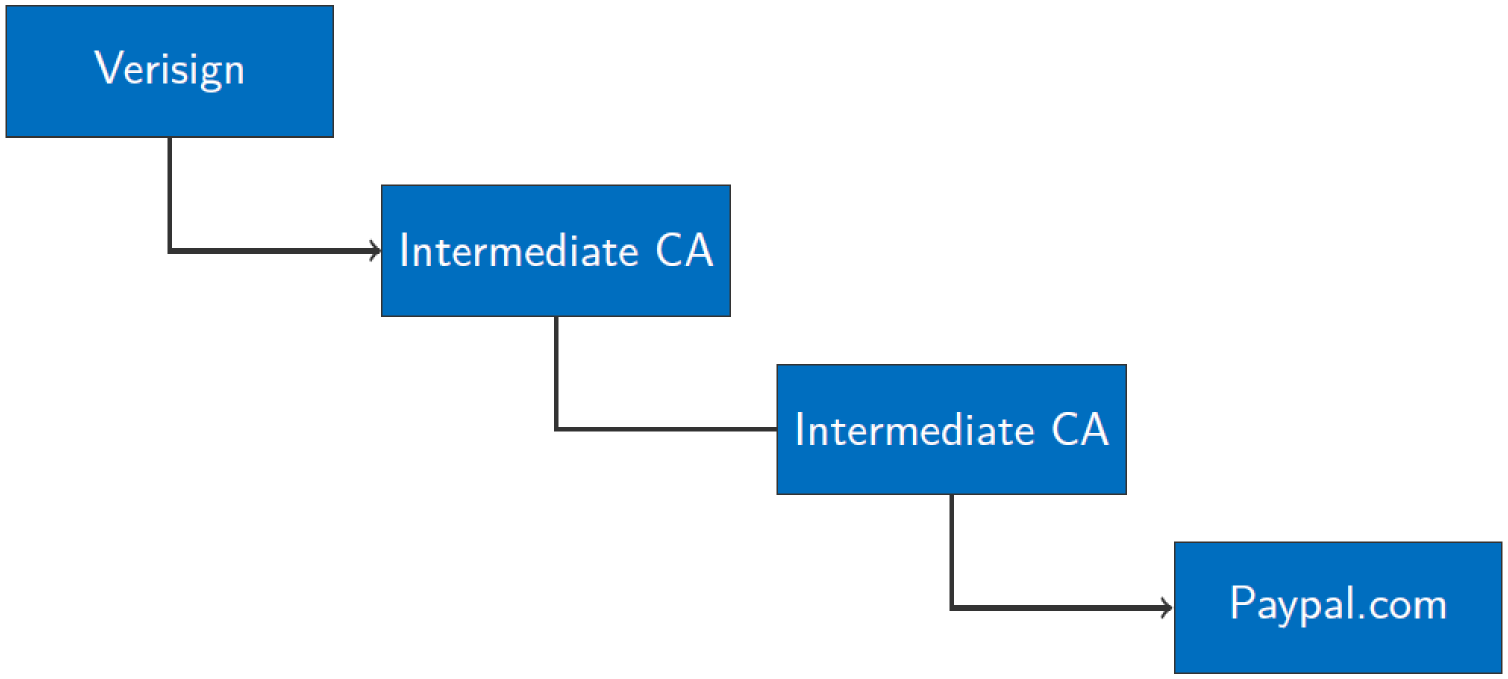
- 1. Generate a public/private keypair for your server.
- 2. Generate a Certificate Signing Request (CSR), containing domain name, public key, and other relevant details, and send this to a CA.
- 3. CA confirms any necessary details (domain validation or extended validation).
- 4. The CA signs the certificate and sends it back.
- 5. Install the certificate on your server.



Version	3
Issuer	GlobalSign
Validity	2015-12-11 to 2016-12-11
Common Name	`*.wikipedia.org`
Public Key	Elliptic: `04:cb:···:0b:fd`
Signature algorithm	PKCS#1 SHA-256 RSA
Signature	`b2:c6:···:39:fd`

Certificate Chaining

– Root CAs can give permission to other CAs to sign SSL certificates.



Certificate Revocation

- There is a method to revoke SSL certificates, called Online Certificate Status Protocol (OCSP).
  - Whenever a browser wants to confirm the validity of an SSL certificate, it makes a request to the OCSP URL embedded in the CA's signing certificate.
  - The OCSP server sends a signed response indicating whether the certificate is still valid. The signature on the response only covers some of the response data and does not cover the response status.

- A possible MITM attacker can send back any response status. The outcome of this is that an attacker can intercept an OCSP request and send a tryLater response status.
  - Because most browsers will silently ignore such OCSP statuses, the communication will proceed without raising an issue.

- Recommended viewing: the DEFCON presentation "More tricks for defeating SSL in practice" https://www.youtube.com/watch?v=ibF36Yyeehw

Attacks on SSL/TLS

THE UNIVERSITY OF SYDNEY

BEAST

– BEAST (Browser Exploit against SSL/TLS): A practical exploit taking advantage of a protocol flaw with CBC in TLS.

– Implication: If an attacker can read and inject packets quickly, then they can eavesdrop on any SSL connection using *-CBC (such as AES-CBC). This attack could be done in local networks. (e.g. Wi-Fi in shops, LAN in labs).

– Was fixed in TLS1.1 (2006) but was not widely adopted. TLS1.2 and 1.3 (current standards) are not vulnerable to BEAST.

BEAST and RC4

- Prior to adopting TLS 1.1, several workarounds to mitigate were proposed.

  - Use RC4: government standards require AES-CBC not RC4.
    - RC4 is a stream cipher, does not require CBC.
    - RC4 is quicker and cheaper than AES-CBC.
    - Later, RC4 was found to be insecure.

  - Change the block cipher mode of operation: TLS 1.0 only supported CBC mode, so this workaround was not possible.

  - Insert empty packets to consume unsafe initialisation vectors: Not documented in TLS 1.0 and caused compatibility issues with some SSL stacks.
    - Incomplete blocks are padded with random data to the block size.

CRIME (2012)

- The CRIME (Compression Ratio Info-leak Made Easy) attack is based on taking advantage of compression.
  - Rationale: make the protocol use less traffic by compressing data before encrypting it.
  - The size of the encrypted payload is visible, and compression makes size give away quite a lot about the payload.

- The above observation can be turned into an attack on protocols that compress and then encrypt.
  - It is common to have cookies storing an authentication token.
  - Cookies are sent with every request (in the HTTP headers).
  - What if the attacker controls part of that request?

CRIME (2012)

- Suppose an attacker can get a victim to send a message of the form $E_k(Compress(m \,||\, a))$ to a server, where a is controlled by an attacker, and m contains authentication data (like cookies, for example), but k and m are unknown to the attacker.

Let a = “`sessionKey=a`” $\Rightarrow Size(Ek(Compress(m \parallel a))) = 100$

a = “`sessionKey=b`” $\Rightarrow Size(Ek(Compress(m \parallel a))) = 100$

$\ldots$

a = “`sessionKey=j`” $\Rightarrow Size(Ek(Compress(m \parallel a))) = 95$

- Now the attacker knows it is likely that the string “`sessionKey=j`” occurs somewhere in the message m. Next, the attacker repeats for “`sessionKey=ja`”, “`sessionKey=jb`”, and so on.

POODLE (2014)

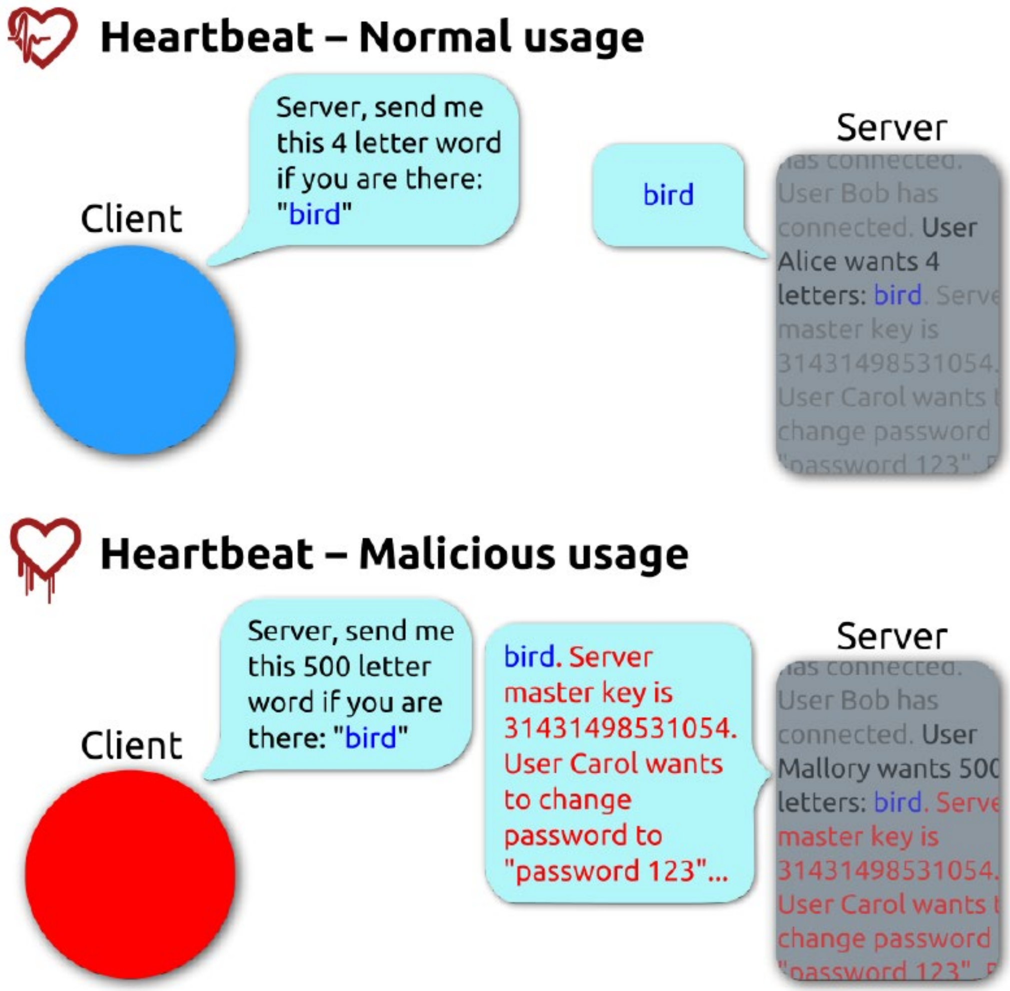
- The POODLE (Padding Oracle On Downgraded Legacy Encryption) attack is a padding oracle attack against SSLv3.
  - In CBC mode, padding bytes must be appended before encryption.
  - The last byte of the padding block is known to the attacker (a function of message length).
  - SSLv3 does not specify the contents of the other padding bytes, so the server cannot check these.
  - SSLv3 uses MAC-then-Encrypt, rather than Encrypt-then-MAC.

- The outcome is that due to poor design in padding specification and MAC order, a man-in-the-middle can get the server to decrypt an arbitrary block of data, and the server will accept the message if and only if the last byte is correct. This gives an attacker 256 guesses (on average) before recovering a byte from the message. Further reading: ImperialViolet - POODLE attacks on SSLv3

HEARTBLEED (2014)

- Unlike CRIME, BEAST, POODLE, etc., Heartbleed was not a vulnerability in the protocol, but in a specific implementation of the protocol, the OpenSSL library. Unfortunately, this library was very widely used.
  - The Heartbeat Extension (RFC 6520) of TLS allows keeping a secure connection alive even if no data is exchanged.
  - A client to send a short message to the server and the server copies back the message.
  - This is meant to increase the reliability of TLS over long-lived connections.
  - The heartbeat message is of the form (message length, message).
  - OpenSSL **assumed** that the given message length was the true message length. Heartbeat is a classic bug occurring by a lack of proper bounds checking.

HEARTBLEED (2014)


Heartbeat – Normal usage
Client
Server, send me this 4 letter word if you are there: "bird"
bird
Server
has connected. User Bob has connected. User Alice wants 4 letters: bird. Serve master key is 31431498531054. User Carol wants t change password "password 123". P
Heartbeat – Malicious usage
Client
Server, send me this 500 letter word if you are there: "bird"
bird. Server master key is 31431498531054. User Carol wants to change password to "password 123"...
Server
has connected. User Bob has connected. User Mallory wants 500 letters: bird. Serve master key is 31431498531054. User Carol wants t change password "password 123". P