Analysis

* Capped at 90% if 2 days late

Key Distribution

* How do I send keys around?
* How do I know what I am authenticating?

We don’t know how to authenticate

* Gmail phishing attack
* Can’t be sure what to trust and not what to trust

How to send keys around / how to authenticate

* Problem with authentication: How do you know it is coming from the real sender?
* Prevent replay attacks: use a NONCE

Main Challenge: How to distribute shared secrets around?

**Horton Principle:** When you authenticate something, make sure you are authenticating exactly the write thing. You have to make sure what you mean is what you say and what you say is what you mean.

* E.g. You are trying to authenticate Gitzon.com but you are really authenticating Gitzon.org.
* You have to watch out all the time that you are authenticating the right thing.

Two approaches to sharing keys

* Centralised authority based Command and Control
  + A central place where everyone stores their shared secret. (e.g. a bank)
  + You go to the bank to get the shared secret and then you use the shared secret to talk to Gitzon.
  + CONS: Single point of failure. Things can fail catastrophically if things go wrong
* Decentralised peer based web of trust: e.g. PGP, Blockchain
  + A distributed web of trust. If people you trust think something, then you will trust that thing too.

**PKI – Public Key Infrastructure**

* A centralised way of distributing public keys
* This works because even if everyone knows the public key, it doesn’t matter because someone would need the private key to know the message.
* Distributing the key is a problem because of Man-In-The-Middle attacks. PKI solves this problem.
* PKI solves this by constantly timing out.

**Certificate Authority**

* Example:
  + There are people that I trust called Certificate Authorities. They are trusted people that are very honest and wise (e.g. Verisign)
  + Brendan goes to Chris (a certificate authority) and says that he’s really Brendan, and Chris says prove it. Brendan gives a lot of proof. After some extensive checking, Chris accepts Brendan is Brendan and he gives a certificate to Brendan saying that he is Brendan. Chris signs the certificate.
  + The certificate is “signed” if
* Signing
  + Basically a signature to make sure that nobody else is forging it. Done by you.
  + Basically use a private key to encrypt signature after intensive testing to ensure that a person/ entity is who they say they are.
  + Then person who gets message with a certificate from entity uses public key of CA to prove the signature is real.
* Solution
  + When you get a web browser, it comes pre-loaded with a certificate. All the browser has to do, is to check that it’s really Brendan.
  + Can add fake certificates.
  + CA’s get paid to authenticate, so CA’s sometime have potential to do a crappy job.
* Problems
  + People can easily insert their own certificate into the browser or edit an existing certificate
  + How do we even know if the entity we are authenticating is the entity that we are expecting?

**Bruce Schnier**

* He wrote a paper called “Ten Problems with PKI”.
  + Is a good description of ten ways that PKI can fail (while still not failing according to spec)

**SSL** would typical have:

* “your” domain name
* “your” company name
* “your” address, your city, “your” state and “your” country
* The expiration date of the Certificate
* Details of the Certification Authority responsible for issuing the Certificate

**HTTPS**

* Http over SSL
* Session key and symmetric cipher for **perfect forward secrecy**
* PKI for authentication
* Mac for integrity

**Forward Secrecy**

* If bad guys are observing communication between two parties and one day they learn the shared secret.
* A compromise in one point in time will cause problems in the past. E.g. they can go back and decrypt all the past messages.
* How do we make sure that if the bad guys find out they keys, that they cannot decrypt past message?
* Public Key Cryptography (such as RSA) is expensive / slow.
  + Solves the key distribution problem, as you just put the key out in public
* HOW DO WE SOLVE THIS PROBLEM?
  + You create a public key channel between each other (slow / expensive)

You agree on the parameters for Diffey-Helman

You work out what the session key is and then you use that.

Cache Poisoning

* ARP: Address Resolution Protocol
* Works out exactly which devices are sitting on which IP address on a local network
* When a msg comes in for a particular IP, it knows which mac address it is going to
* If you can attack the switch on the network, send a message to your own machine rather than the designated mac address. Steal all messages from your neighbour and send back yourself.

**Web of trust**

* PGP: Find out more about this

Work out what are the main security focus / issues with the case before trying to solve (WHICH ONE IS IT: C . I . A ???)

A Web Server is like an Oracle which you send something and it sends you something back

Padding / Blocks / Block Ciphers **(REFER TO WEEK 7 NOTES)**

* When breaking into blocks, what happens if your message isn’t the multiple of the block size? Computers can’t deal with variable length things. It depends on having a message of exactly the right length.
* Normally, you would add some padding to the end that is irrelevant but enables the enciphering. When you decrypt the message, you just throw array the padding.
* It lets you decrypt the padding characters / whether you got them right or not and incrementally do them one at a time.
* **E.C.B: Electronic Code Book**
  + You have a message that breaks into 3 chunks, encrypt each chunk independently and concatenate them together = that is the encrypted cipher text.
  + Simple to do, easiest to do but not a good one.
  + Problem with E.C.B: If you’ve got a big plaintext with any patterns / order, it is likely that you have blocks with identical content 🡪 Identical plaintext block = identical cipher-text blocks 🡪 Easy to crypt-analyse and reverse engineer it.
  + The different blocks should be treated differently.
* **C.B.C: Cipher Block Chaining**
  + CBC: Same plaintext 🡪 different cipher-text. You can decode in parallel because you have all the cipher-text before you decrypt.
* **C.T.R: Counter Mode Encryption**
  + What you fold in each time is a counter / NONCE.
  + You encrypt the NONCE using the key and XOR it with the plaintext.
  + You are actually encrypting the NONCE rather than the plaintext.
  + However, if you have XOR-ing two things, it will have the same properties as the plaintext.
  + **Nonce/Counter 🡪 Encrypt with key 🡪 XOR encrypted Nonce/Counter with Plaintext = Ciphertext**
  + **Nonce/Counter 🡪 Encrypt with key 🡪 XOR encrypted Nonce/Counter with Ciphertext = Plaintext???  
    (^NEED CONFIRMATION ON THE REVERSE)**