# Software and Security Engineering

Lecture 8

**Martin Kleppmann**

mk428@cam.ac.uk

*With many thanks to Ross Anderson*



205

# Public key cryptography

Allows two parties with no prior knowledge of each other to jointly establish a shared secret key over an insecure channel

Examples include Diffie-Hellman and RSA

# Diffie Hellman revision

Alice and Bob publicly agree to use p = 23, g = 5

1. Alice chooses secret integer a = 4, then
   $A \rightarrow B$: $g^a \bmod p = 5^4 \bmod 23 = 4$
2. Bob chooses secret integer b = 3, then
   $B \rightarrow A$: $g^b \bmod p = 5^3 \bmod 23 = 10$
3. Alice computes $10^4 \bmod 23 = 18$
4. Bob computes $4^3 \bmod 23 = 18$

Alice and Bob now agree the secret integer is 18

Example derived from https://en.wikipedia.org/wiki/Diffie-Hellman_key_exchange



You saw Diffie Hellman in Discrete Maths. This simple version uses a multiplicative group of integers modulo p, where p is prime and g is a primitive root modulo p The values of p and g are chosen in this way to ensure that the resulting shared secret can take on any value from 1 to *p*–1.

This protocol has a significant limitation: it is susceptible to a person-in-the-middle attack.

# Physical public key crypto with locks

- Anthony sends a message in a box to Brutus. Since the messenger is loyal to Caesar, Anthony puts a padlock on it
- Brutus adds his own padlock and sends it back to Anthony
- Anthony removes his padlock and sends it to Brutus, who can now unlock it

Is this secure?

208

Anthony wants to kill Caesar, but needs Brutus' help to do so. How can Anthony send a message to Brutus yet not let the messenger read the message? This proposal is insecure: it is vulnerable to a MITM attack, as is Diffie Hellman.

Here Anthony has shared the secret message with someone, but Anthony doesn't know who it is!

# Asymmetric public-key crypto

- Separate keys for encryption and decryption
- Publish *encryption* key widely (the "public key") allowing anyone to create an encrypted message; only holder of *decryption* key ("private key") can decode the message and read it
- Digital signatures are the other way around: only you can sign but anyone can verify
- Example: RSA

209

More on this in the Part IB Security course and Part II Cryptography course. Require knowledge at this point is as stated above and expanded on in the lecture.

Note that asymmetric public-key crypto has the same problem as Diffie-Hellman: how do you know that you have the right public key for Alice and you are not subject to a MITM attack?

# Public-key Needham-Schroeder

- Proposed in 1978:

$A \rightarrow B: \{N_A, A\}_{KB}$
$B \rightarrow A: \{N_A, N_B\}_{KA}$
$A \rightarrow B: \{N_B\}_{KB}$

- $N_A$ and $N_B$ are nonces generated by A and B respectively
- $K_A$ and $K_B$ are public keys for A and B respectively
- The idea is to use $N_A \oplus N_B$ as a shared key

Is this okay?



Once public key crypto is discovered, people then looked for ways to use it. Background: Needham went to California every summer to work at Xerox Parc. He got a preprint of the RSA paper and decided to apply it to the problem on the Xerox network computer project. Kerberos, discussed earlier, was derived from the Needham-Schroeder protocol, and in 1978 Needham proposed the following public-key variant of the protocol.

This version does not require an online server, Sam. Instead the nodes now need the long-term public keys of each other. Here, KA and KB are the *public* keys of A and B respectively, and the aim is to use these in order to derive a symmetric session key between A and B (symmetric cryptography is computationally cheaper).

# MITM attack found 18 years later

$A \rightarrow C: \{N_A, A\}_{KC}$

$C \rightarrow B: \quad \{N_A, A\}_{KB}$

$B \rightarrow C: \quad \{N_A, N_B\}_{KA}$

$C \rightarrow A: \{N_A, N_B\}_{KA}$

$A \rightarrow C: \{N_B\}_{KC}$

$C \rightarrow B: \quad \{N_B\}_{KB}$

The fix is explicitness. Put all names in all messages.



Here Charlie can pretend to be Alice when talking to Bob (line 2). Doing so means that Charlie gets $N_A$ (line 1) as well as $N_B$ (line 5) and therefore can computer the shared key between Bob and Alice. Don't beat yourself up with if you didn't spot it. It took 18 years to spot the problem as shown on this slide.

# Binding keys to principals is hard

- Physically install binding on machines
  - IPSEC, SSH
- Trust on first use; optionally verify later
  - SSH, Signal, simple Bluetooth pairing
- Use certificates with trusted certificate authority
  - Sam signs certificate to bind Alice's key with her name
  - Certificate = $sig_S\{A, K_A, Timestamp, Length\}$
  - Basis of Transport Layer Security (TLS) as used in HTTPS
- Use certificate pinning inside an app
  - Used by some smartphone apps

# Transport Layer Security (TLS)

- Uses public key cryptography and certificates to establish a secure channel between two machines
- Protocol proven correct (Paulson, 1999)
- Yet, the protocol is broken annually
- Often a large number of root certificate authorities. Are these all trustworthy?

213

Earlier versions of this protocol were called Secure Sockets Layer (SSL). There's been around one bug every year in TLS since 1999. The first series of attacks were timing attacks: look at how long it takes a server to respond and use this to determine certain bits of the key. The challenge here is that compilers and security engineers fight. Compilers attempt to make code as fast as possible, and may optimise away "make work" inserted by security engineers who are attempting to ensure constant-time execution for critical operations. There are many more technical hacks here, but these are for later courses.

Another major problem is that it's really hard to fix bugs when found. In order to change the protocol you need to make changes to both the client and the server. This is hard for the Web since you have to upgrade both all web browsers and all web servers and no single party is in control of the overall ecosystem. There are poor incentives. There are 187 root certificates installed on my Mac. Web browsers typically trust all of them, and any of these certificates may be used to license other providers with the power to create further certificates for arbitrary domains.

# DigiNotar went bust after issuing bogus certificates

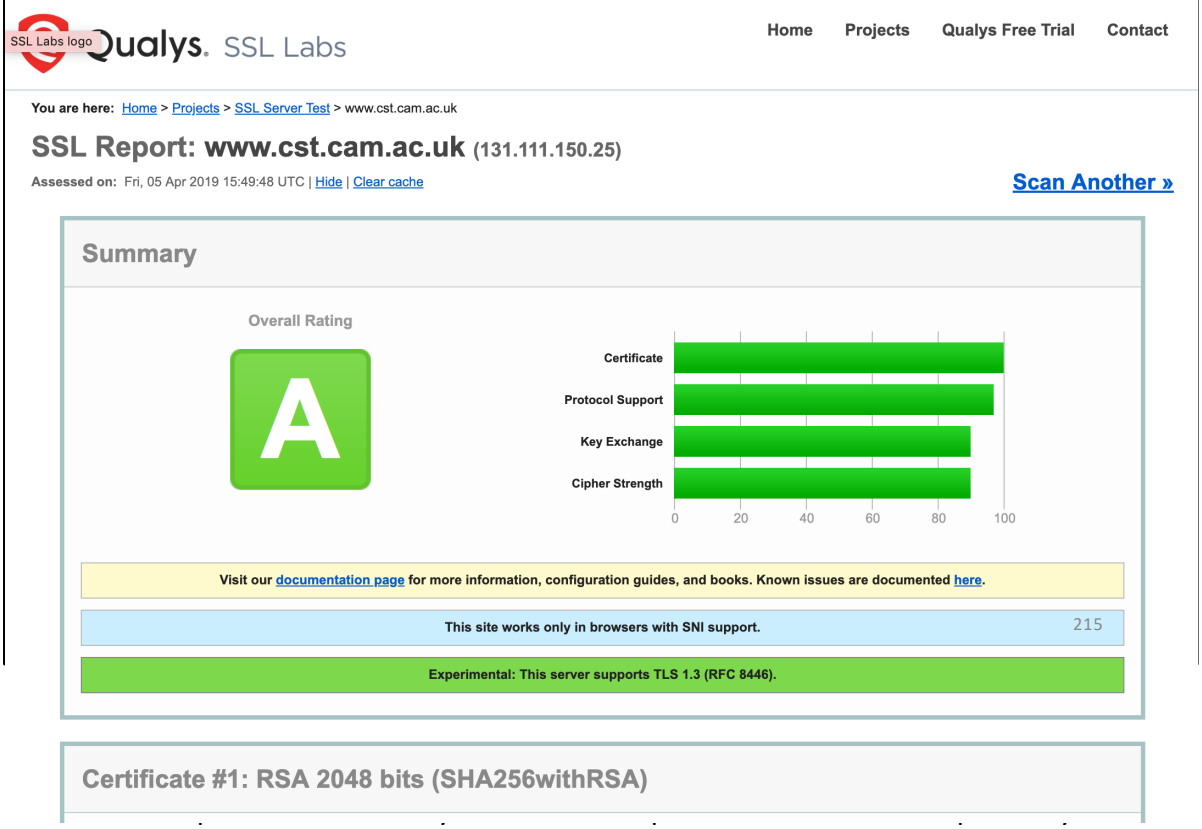
- Dutch certificate authority
- More than 300,000 Iranian Gmail users targeted
- More than 500 fake certificates issued
- Major web browsers blacklisted all DigiNotar certs



Iranian Gmail users were found to have been given fake certificates for Gmail, allowing a MITM attack to take place. Further investigation revealed that over 500 fake certificates were issued. No public investigation provides conclusive proof of all steps in the process, but the Iranian Government and the NSA have both been suggested as potential attackers. The behaviour of governments here has a significant influence on the security of everyone else. The cryptowars of the 1990s, where governments attempted to mandate exceptional access to encrypted key material, are being revisited. See: Ableson et al. Keys Under Doormats: mandating insecurity by requiring government access to all
data and communications. https://www.schneier.com/academic/paperfiles/paper-keys-under-doormats-CSAIL.pdf

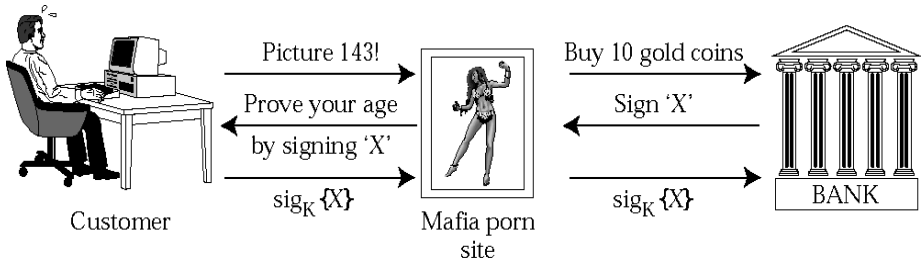
Further reading: https://en.wikipedia.org/wiki/DigiNotar

# TLS security landscape is complex



# Chosen protocol attack

The Mafia asks people to sign a random challenge as proof of age for porn sites!




216

The experience with TLS is challenging for operators because they are heavily reliant on third parties. Just as the saying goes "there is no cloud, just someone else's computer". Such reliance can be abused directly of course, but it also opens up new opportunities for the attacker. Here the mafia has repurposed (deliberately) a signing protocol to extort money from you.

# Bugs are found in and around code

- Bugs in the code
  - Arithmetic
  - Syntactic
  - Logic
  - Concurrency
- Bugs around the code
  - Code injection
  - Usability traps



Maurice Wilkes: "It suddenly occurred to me when I was at the corner of the stairs, that I would spend a large part of my life discovering bugs in my own programs."

The first documented use of the term "bug" for a technical malfunction was by Thomas Edison; In the year 1878 he mentioned the term in a private letter. This counters an oft-mentioned view that the term bug is derived from a moth getting trapped in a computer, although perhaps this latter event popularised the term. For further information, see https://en.wikipedia.org/wiki/Software_bug

# Patriot missile failures in Gulf War I

German Air Force; CC-BY-SA, Darkone, Wikipedia

Afgan National Army; PD, Davric, Wikipedia

- Failed to intercept an Iraqi Scud missile in first Gulf War on 25th February 1991
- Scud struck US barracks in Dhahran; 28 dead
- Other Scuds hit Saudi Arabia, Israel

218

The MIM-104 Patriot is a surface-to-air missile (SAM) system, the primary of its kind used by the United States Army and several allied nations. The picture on the left is a Patriot system used by the German Air Force, August 2005 (https://en.wikipedia.org/wiki/MIM-104_Patriot). The picture on the right is of a Scud missile and launcher in use by the Afgan National Army.

# Caused by arithmetic bug

- System measured time in 1/10 sec, truncated from $0.0001100110011\ldots_b$
- Accuracy upgraded as system upgraded from air-defence to anti-ballistic-missile defence
- Code not upgraded everywhere (assembly)
- Modules out by 1/3rd sec after 100h operation
- Not found in testing as spec only called for 4h tests

Lesson: Critical system failures are typically multifactorial

219

As you will know from the Numerical Analysis course, not all decimal fractions are precisely representable as binary floating-point numbers.

System was upgraded from anti-aircraft to anti-ballistic missile. This required an increase in accuracy since ballistic missiles such as the Scud travel much faster than aircraft. Unfortunately the code was not updated everywhere. This meant that different modules (some with upgraded accuracy, some not) then fell out of sync with each other, resulting in the failure of the Patriot system to effectively target Scud missiles. This problem was not caught by static analysis tools since the code was written in assembly, and therefore there was no high-level language features such as a strong type system which could have helped. Testing was also inadequate – missile defence systems are often operated continuously for hundreds of hours, yet the testing regime only called for testing over a 4-hour period. Short-term solution was to reboot Patriot every 4 hours until the underlying cause was determined.

# Syntactic bugs arise from features of the specific language

For example, in Java:

`1 + 2 + ""` evaluates to `"3"`

`"" + 1 + 2` evaluates to `"12"`

This is due to coercion from primitive integers to `java.lang.String`



Java supports implicit type conversion or *coercion* from primitive integers to Strings. This is typically helpful, however implicit type conversion interacts with implicit operator precedence in the above example, leading to different outcomes for what initially appear to be quite similar expressions. Removing all implicit type conversion may also result in (different) errors since programmers may then insert explicit type conversions which themselves might be problematic.

Further reading: Joshua Bloch and Neal Gafter, Java Puzzlers: Traps, Pitfalls, and Corner Cases, Addison-Wesley. http://www.javapuzzlers.com/

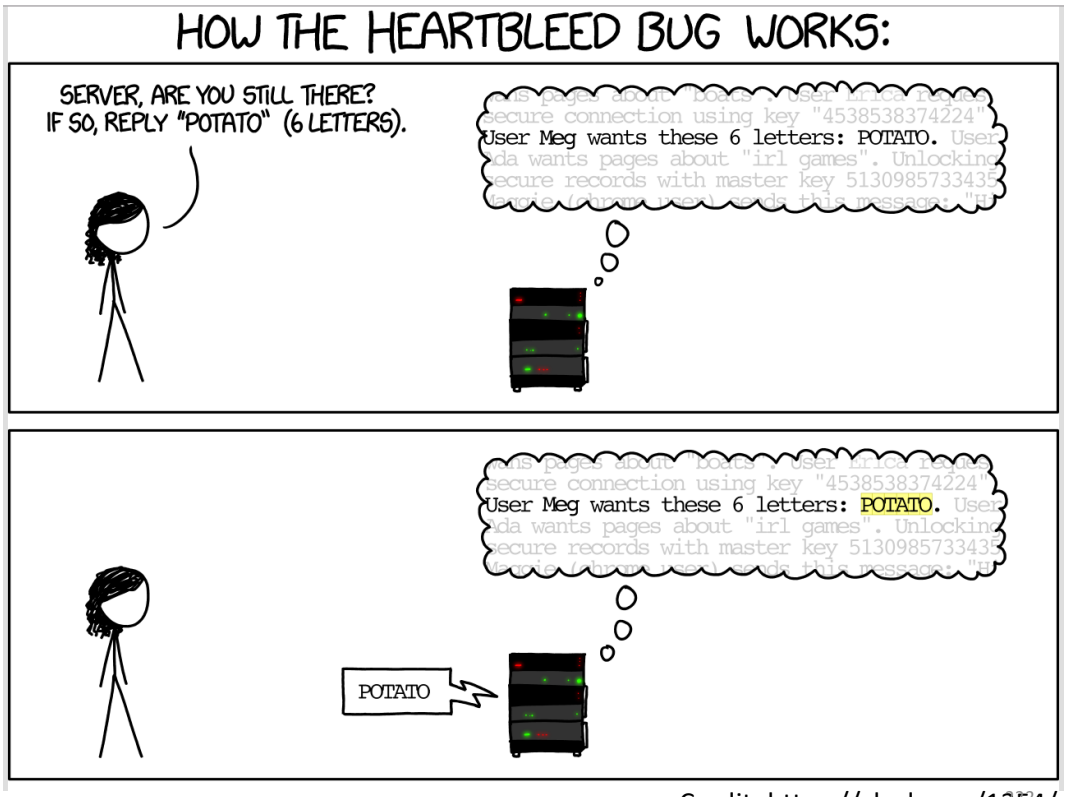
# Apple's goto fail bug (2014)

```
static OSStatus SSLVerifySignedServerKeyExchange(SSLContext *ctx,
                 bool isRsa, SSLBuffer signedParams,
                 uint8_t *signature, UInt16 signatureLen)
{
    OSStatus err;
    //...
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail; //error: this line should not exist
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    //...
    fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

221

This is a control-flow (logic) bug. Note the two consecutive lines containing "goto fail"; the second is erroneous and the control flow therefore unconditionally executes the code at the "fail" label. It's not clear how this failure was introduced. Perhaps it was an erroneous merge on a commit, either automated or manual by a user. Better unit tests might have helped.

Further reading: https://www.imperialviolet.org/2014/02/22/applebug.html




Credit: https://xkcd.com/1354/

This is another logic bug. The heartbeat feature allowed either the client or the server to ask the other party to reply with a specified message of a given length after a period of time, allowing the requesting party to know that the other was still online and available. Unfortunately the requesting party could claim the provided message was much larger than reality. This led to a buffer over-read vulnerability: the requesting party would receive their message appended with any additional contents found in the server or clients memory. The bug's name derives from heartbeat. NB: In the absence of malice, the code worked just fine.

Further reading: https://en.wikipedia.org/wiki/Heartbleed

# Heartbleed allows clients to read the contents of server memory

Therefore a malicious client could read:

- Secret keys of any TLS certificates used by server
- User creds such as email address and passwords
- Confidential business documents
- Personal data

The attack left no trace of use in server logs



The potential impact of this vulnerability is huge. Potentially the entire contents of the server's process address space were accessible.

One significant risk was for webservers connected to the public Internet. Since the attack left no trace of use in server logs, this meant that all servers needed to not only upgrade their software to fix the vulnerability, but to replace all important key material. TLS certificates in use by the server are an important example, since the private keys may have been compromised. Ideally all user passwords should have been replaced too as these may have been compromised, but the risk of this type of failure depends on details of any implementation.

# Notification and clean-up difficult

| | |
|---|---|
| 12th March 2012 | Bug introduced (OpenSSL 1.0.1) |
| 1st April 2014 | Google secretly reports vuln |
| 3rd April 2014 | Codenomicon reports vuln |
| 7th April 2014 | Fix released |
| 7th April 2014 | Public announcement |
| 9th May 2014 | 57% of website still using old TLS certificates |
| 20th May 2014 | 1.5% of 800,000 most popular websites still vulnerable |

224

The original flaw was introduced into the source code repository for OpenSSL on 31st December 2011, and was released in OpenSSL in version 1.0.1 on 12th March 2012. The bug appears to have been found by multiple people, including members of the security team at Google who produced a fix which appeared on RedHat's issue tracker on 21st March 2014. Codenomicon also discovered the problem independently and reported on 3rd April 2014. [Dates sourced from https://en.wikipedia.org/wiki/Heartbleed]

A significant issue with notification is it was essentially impossible to do so quietly: the number of servers and clients which needed fixing is simply too large. Another problem is that many server operators did not realise that they may have been compromised and therefore did not replace their certificates (potentially allowing a MITM attack on all connections, and in the absence of a version of the protocol with forward secrecy, a passive data capture followed by later processing).

A surprising outcome was that may firms decided to outsource certificates to companies like CloudFlare. This is great for the CEO who no longer gets woken up in the middle of the night with things like Heartbleed; now it's CloudFlare's problem. Unfortunately data may be less secure: encryption now runs from customer to CloudFlare, but not necessarily from CloudFlare to company actual servers unless a premium option is purchased. Of course companies don't pay the premium. So now data is backhauled across the Internet where it can be read with passive taps.

# Intel AMT Bug

- AMT allows sysadmins remote access to a machine, even when turned off (but mains power on)
- Provides full access to machine, independent of OS
- A sketch of the protocol for authentication between machine and remote party is as follows:

C $\rightarrow$ S: "Hi. I'd like to connect"

S $\rightarrow$ C: "Please encrypt *X* with our secret key"

C $\rightarrow$ S: "Here are the first *x* bytes of $\{X\}_{KCS}$"



This is a logic bug in the implementation of the protocol. The failure here occurs because the client (not the server) gets to choose how many bytes (*x*) to return, so a malicious client can choose to return zero bytes. Further reading: https://en.wikipedia.org/wiki/Intel_Active_Management_Technology

# Concurrency bug: time of check to time of use failure (TOCTOU)

Check

```
…
File file = new File(args[0]);
if(!file.canWrite())
    return;
```

Use

```
RandomAccessFile fp = new
          RandomAccessFile(file, "rw");
fp.writeChars("Some replacement text");
fp.close();
…
```

Adapted example from https://en.wikipedia.org/wiki/Time_of_check_to_time_of_use

226

In this example, a programmer is writing a program which has the setuid bit set (see Operating Systems course from last term). Therefore the programmer first checks whether the user has access to a particular file, then if true, uses the file by writing some data to it.

The bug occurs if the operating system can be coerced into performing a context switch at the red line, during which time a malicious user then (e.g. by updating symbolic links) swaps the file accessed. Then sometime later the program will write to a file which the user has specified which the user may not have write access to. This is called a *race condition*. We will see another race condition bug later in the course (Therac-25). Note that the concurrency here is not within the program itself, which has only a single thread of execution. Rather it occurs because the operating system supports multiple concurrent processes in execution.

# Clallam Bay Jail inmates perform code injection on payphones

1. Inmate typed in the number they wished to call
2. Inmate selected whether the recipient spoke Spanish or English
3. Inmate was asked to say their name; “Eve”, say
4. The phone then dialled the number and read out a recorded message in chosen language and appended inmate name to the end:

> “An inmate from Clallam Jail wishes to speak with you. Press three to accept the collect call charges. The inmate’s name is” … “Eve”

227

The hack is to select the language that the callee *doesn’t* speak (e.g. Spanish), and then state your name in Step 3 as “To hear this message in English press three”.

Lesson: remember that you are protecting the whole system, including against potentially malicious users.