Cryptographic protocols

Or how to use maths to keep secrets

Vanessa Teague, March 2016
vjteague@unimelb.edu.au

Short bio

- I did my bachelor's degree here at UniMelb (in maths and CS)
- I did my PhD at Stanford Uni in California
- I am interested in using cryptography for large complicated computations in which you don't trust all the participants
- My favourite research application is electronic elections
  - In which there's a fair argument for not trusting anyone
- I also waste a lot of time writing op ed pieces about how it shouldn't have been done

Chapter 1: Public Key cryptography

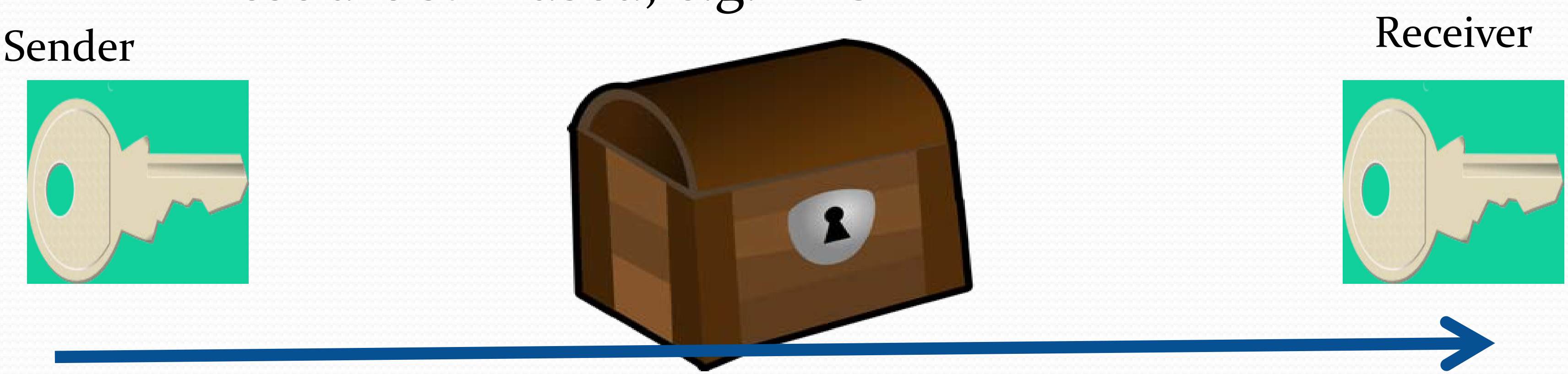
Or how to send secret messages to people you haven't met

What’s cryptography?

- Sending messages that are secret from everyone but the intended recipient
- The sender has to “hide” the message for sending, so nobody else can understand it
  - This is called **encrypting**
- The receiver has to “un-hide” and recover the message
  - This is called **decrypting**
- **Public key cryptography is one of the greatest ideas in computer science ever**

Before public-key cryptography

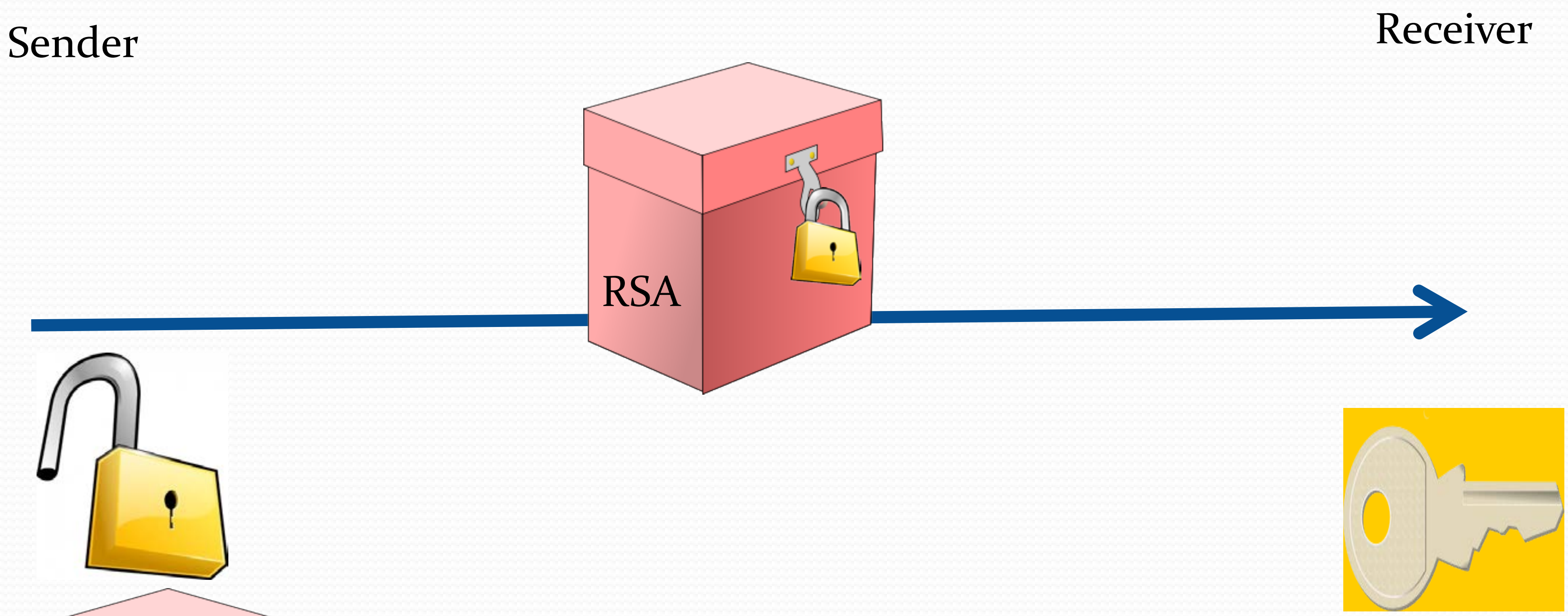
- There was **secret-key cryptography**
- Both the sender and the receiver had to agree on the secret key in advance
  - They had to "meet" somehow
- Encrypting and decrypting used the same key
  - These are still used, e.g. AES

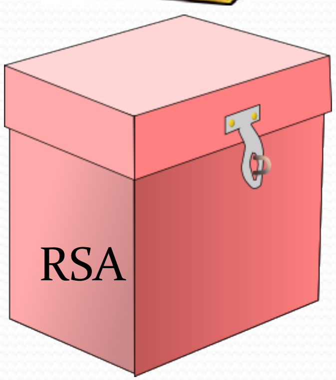


What’s public-key cryptography?

- The receiver generates two keys:
  - a public key e (for encrypting), and
  - a private key d (for decrypting)
- She publicises the public key e
  - People use this for encrypting messages
- She keeps the private key d secret
  - She uses this for decrypting messages

Picture of public-key cryptography





The matching orange colour is supposed to show that this key goes with this padlock.

Example: RSA (for the mathematically inclined)

- The receiver thinks of two large prime numbers p,q
  - About 300 digits long
  - She multiplies them together to get $N=pq$
  - She generates the public key e (almost any e will do)
  - She publicises (N, e). This is her full public key.
- To encrypt message m, compute
  - $m^e \bmod N$
  - (This means take the remainder when m^e is divided by N)
- The receiver can decrypt because she knows p and q
  - Take my word for this for now – it's not supposed to be obvious
  - Nobody else can factorise N – the computation takes too long

Example: RSA (for the even more mathematically inclined)

- The receiver thinks of two large prime numbers p,q
  - About 300 digits long
  - She multiplies them together to get N=pq
  - She generates the public key e (almost any e will do as long as it's coprime to (p-1)(q-1))
  - She publicises (N, e). This is her full public key.
- To encrypt message m,
  - Pad m with a carefully chosen random string r
  - Compute $(m \,||\, r)^e$ mod N
  - (This means take the remainder when $(m \,||\, r)^e$ is divided by N)
- The receiver can decrypt because she knows p and q
  - Take my word for this for now – it's not supposed to be obvious
    - But if you look up the Wikipedia RSA page at See http://en.wikipedia.org/wiki/RSA_(algorithm)
    - and the Euler-Fermat Theorem, you'll be able to figure it out.
  - Nobody else can factorise N. The computation takes too long
    - Strictly speaking, breaking RSA has never been shown to be as difficult as factorising N, but nobody has found a faster way to do it either

The Chinese remainder theorem

- https://en.wikipedia.org/wiki/RSA_%28cryptosystem%29#Using_the_Chinese_remainder_algorithm

- Q: How long does e need to be?
  - A: Not very long, because padding m with random junk ensures that $m^e \bmod N$ is always many times larger than N. Choosing $e = 1 + 2^{16} = 65{,}537$ is popular because it makes m^e easy to compute quickly. There are subtle reasons why very small e is insecure.
  - If you're really interested, see http://crypto.stanford.edu/~dabo/abstracts/RSAattack-survey.html

What is that good for?

- Exchanging a secret key for secret-key cryptography
  - The sender
    - generates a secret key,
    - encrypts a message with the secret key,
    - encrypts the secret key with the receiver’s public key, and
    - sends the encrypted message and the encrypted key.
  - The receiver
    - Uses her private key to decrypt the secret key
    - Uses the secret key to decrypt the message
- This is (almost but not quite) how SSL/TLS works, when you get a comforting little lock at the bottom of your screen before you send your credit card number
- Exercise: draw a picture of this protocol, using boxes, padlocks and keys

What else is that good for?

- Lots of people sending to the same receiver
- e.g. in electronic voting, everyone sends their vote to the Electoral Commission
    - Encrypted with the Commission’s public key

Python cryptography library

- From https://cryptography.io/en/latest/hazmat/primitives/asymmetric/rsa/
- This is a "Hazardous Materials" module. You should **ONLY** use it if you're 100% absolutely sure that you know what you're doing because this module is full of land mines, dragons, and dinosaurs with laser guns.
- from cryptography.hazmat.backends import default_backend
- from cryptography.hazmat.primitives.asymmetric import rsa
- key=rsa.generate_private_key(public_exponent=65537, key_size=2048, backend=default_backend())
- Key.public_key().public_numbers()
- Key.private_numbers().p
- Key.private_numbers().q

Python RSA (encryption)

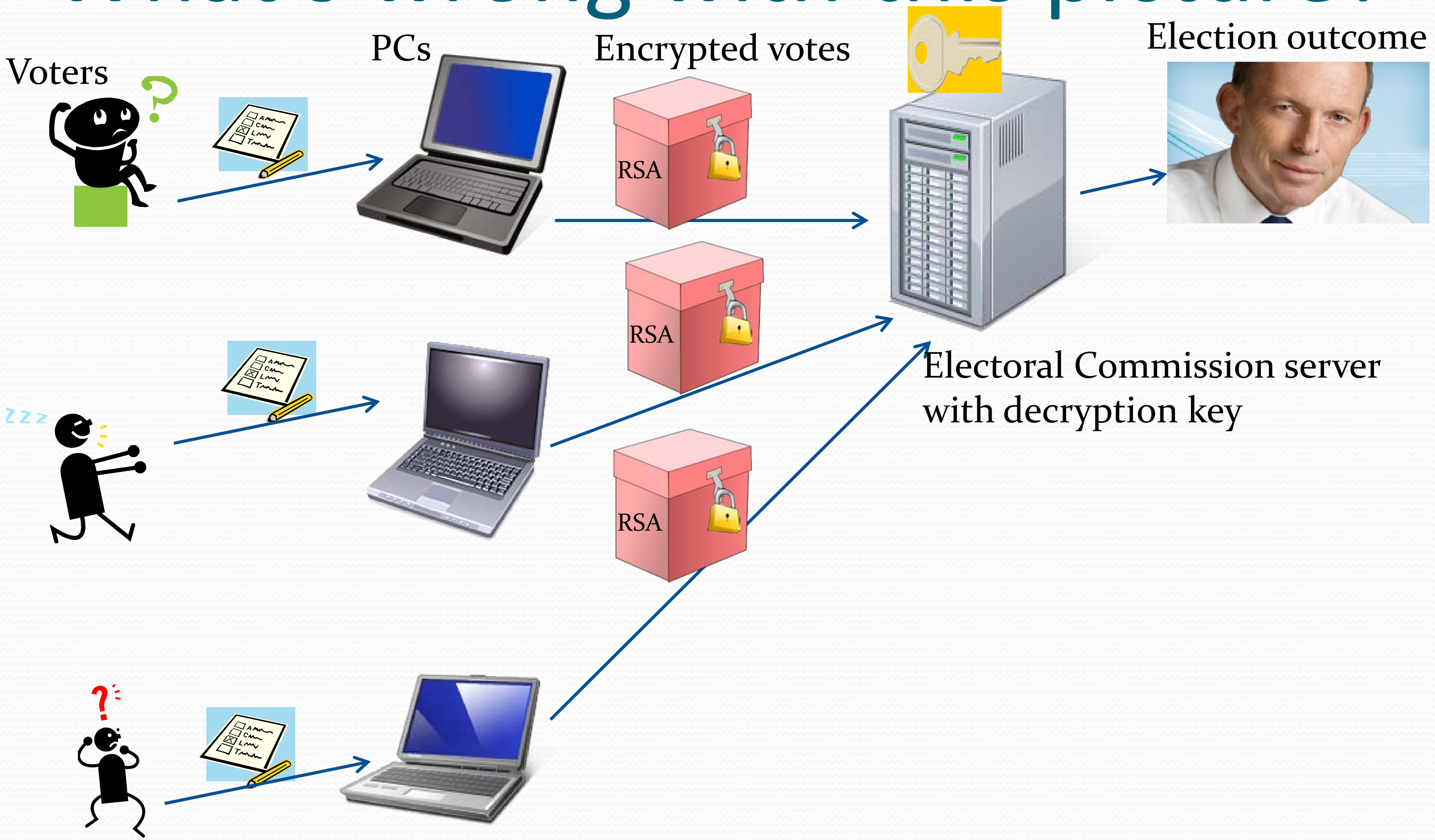
- private_key=key.private_numbers()
- public_key=key.public_key()
- from cryptography.hazmat.primitives.asymmetric import padding
- from cryptography.hazmat.primitives import hashes
- message = b"encrypted data“
- ciphertext = public_key.encrypt(message,
-     padding.OAEP(
-         mgf=padding.MGF1(algorithm=hashes.SHA1()),
-         algorithm=hashes.SHA1(),
-         label=None
-     )
-)
- ** note: SHA1 is outdated, but more recent things like SHA256/512 aren’t supported.

- Plaintext=key.decrypt(
- ... ciphertext,
- ... padding.OAEP(...
  mgf=padding.MGF1(algorithm=hashes.SHA1()), ...
  algorithm=hashes.SHA1(), ... label=None
  - ...)
- ...)

Chapter 2: Internet voting

Or how to use maths to save democracy

What's wrong with this picture?



Answers from the class

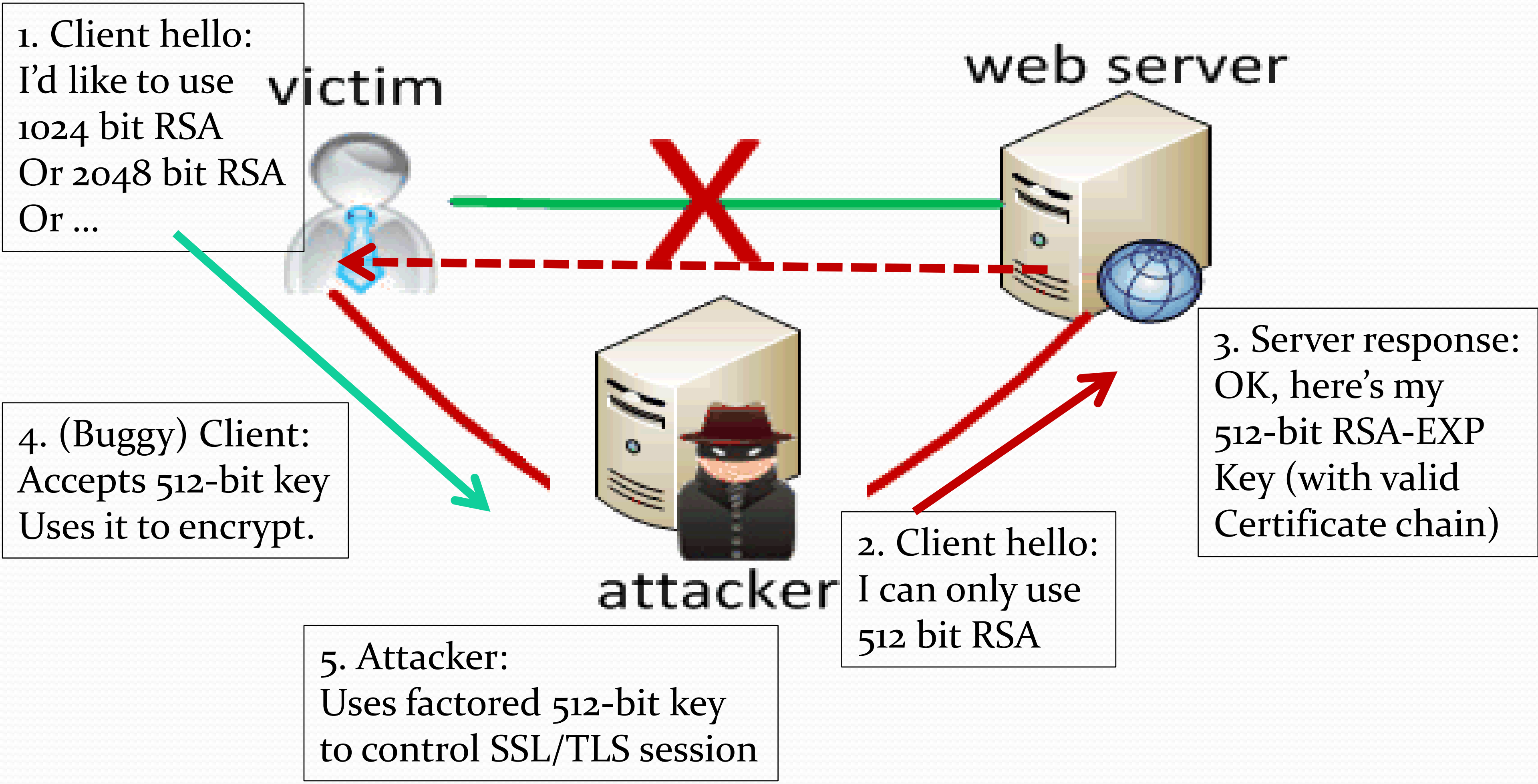
Security Requirements

- Verifiability, so that no-one can manipulate the output
  - Only eligible voters vote
    - At most once
  - Voters should get evidence that their vote was
    - Cast as they intended
    - Counted as cast
  - Everyone gets evidence votes were properly tallied
- Privacy, so coercers can't manipulate the inputs
  - Even if the voter tries to prove how they voted (receipt-freeness)
- Achieving both is hard, especially for remote voting
- I don’t know how to solve the problem completely, and neither does anybody else

Factoring RSA Export keys (FREAK)

- First some history:
  - In ancient times (around the 1990s) the US government restricted the export of strong crypto, in particular of RSA using more than 512 bit keys.
    - Web servers and clients within the US could use strong RSA parameters;
    - Software made outside the US was (obviously) not bound by the restriction, but
    - Software produced in the US but exported outside was restricted to this “Export grade” crypto
  - So lots of servers (and clients) maintained the option to use “export grade” crypto, just in case they had to communicate with a restricted computer
  - Unfortunately, many still do (or did until very recently)
    - Many servers used the same 512-bit key over and over again.
  - 512-bit “export grade” RSA now costs about $100 to break running overnight on Amazon’s EC2 cloud. (https://www.cis.upenn.edu/~nadiah/projects/faas/)

FREAK – intercepting SSL/TLS key establishment



NSW iVote Internet voting security

- The iVote internet voting system was trusted recently in the NSW state election for the return of 280,000 electronic ballots
- During the election period, Alex Halderman and I found a serious security hole that left votes open to manipulation and privacy breach using the FREAK attack
- We notified the Australian CERT, who notified the NSW Electoral Commission, who fixed it, but by then 66,000 votes had been cast
- The final margin of the last seat in the NSW Legislative Council was 3177 votes