Principles of Computer Security

John Rodriguez

User Input

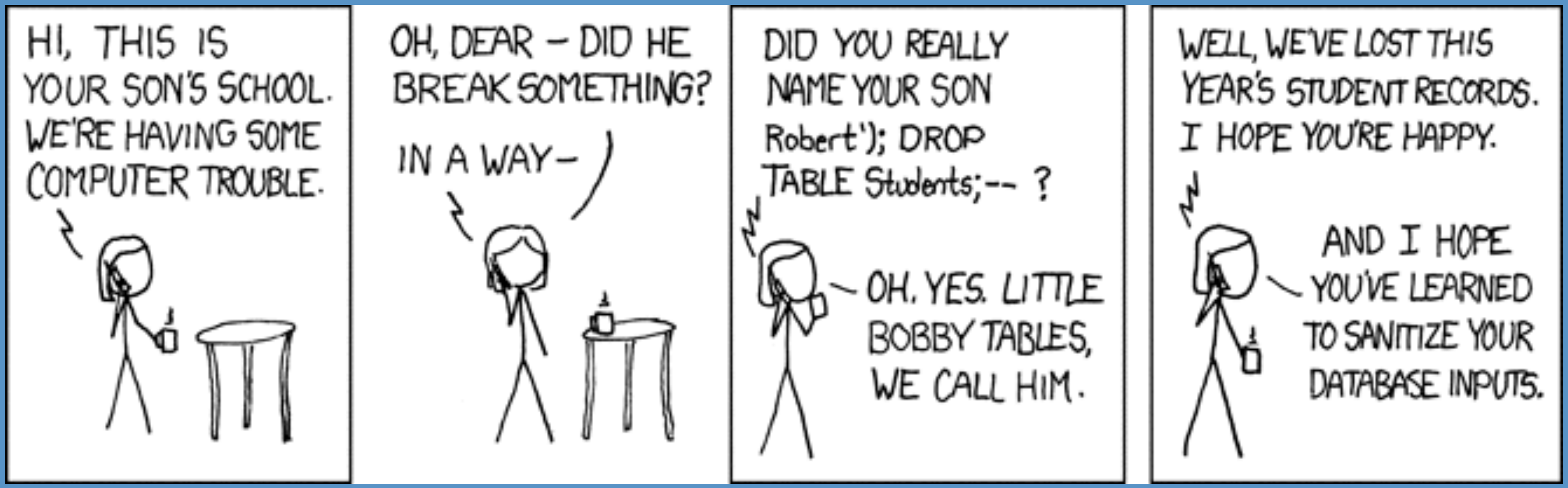
- Should I validate user input on the client, on the server, both, or none?

BOTH!

Clients should ‘fail fast’ allowing the user to correct input and avoid HTTP roundtrips that are doomed to fail

Servers should prevent a bad client from dictating what’s allowed

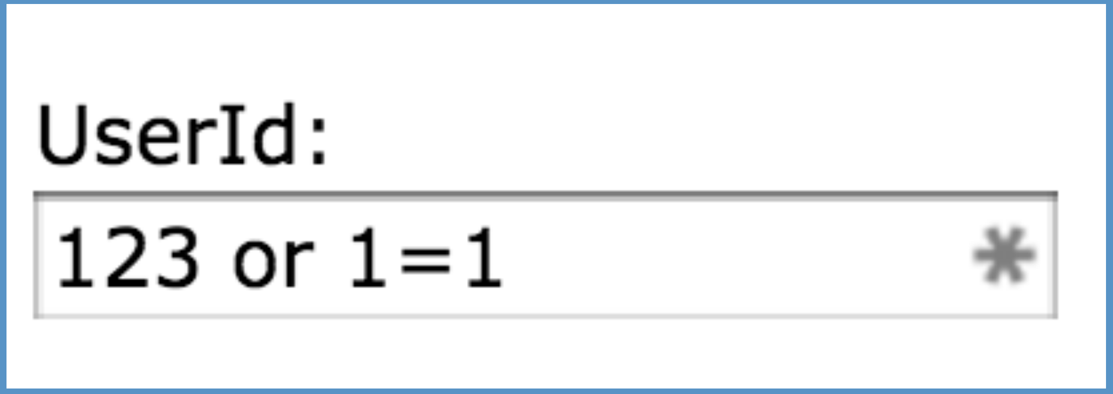
SQL Injection



The Story of 'Bobby Tables'

SQL Injection

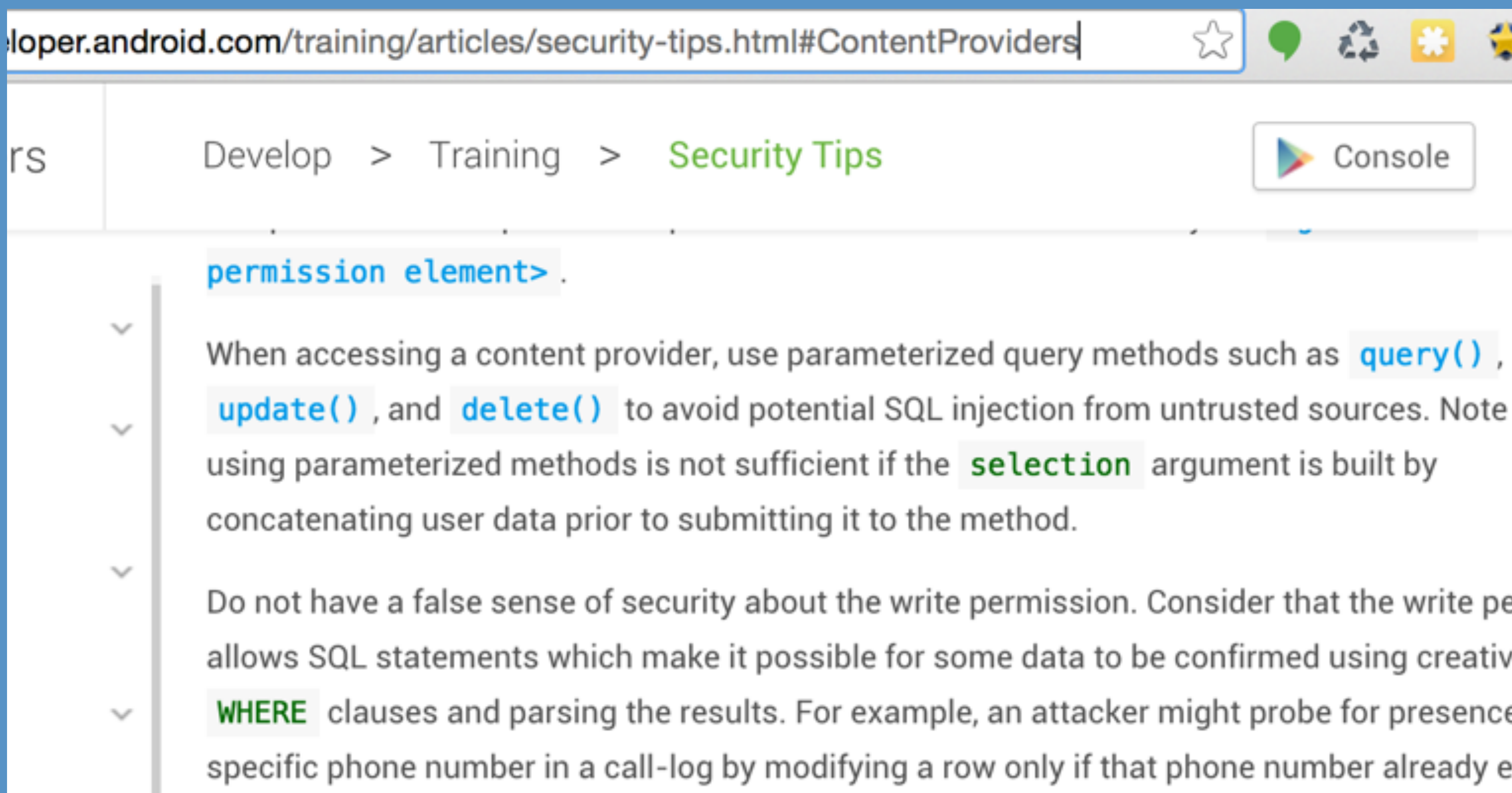
- SELECT * FROM Users WHERE UserId = 123;
- “SELECT * FROM Users WHERE UserId = “ + userID;



SQL Injection

- SELECT * FROM Users WHERE UserId = 123;
- “SELECT * FROM Users WHERE UserId = “ + userID;

SQL Injection



Cryptography

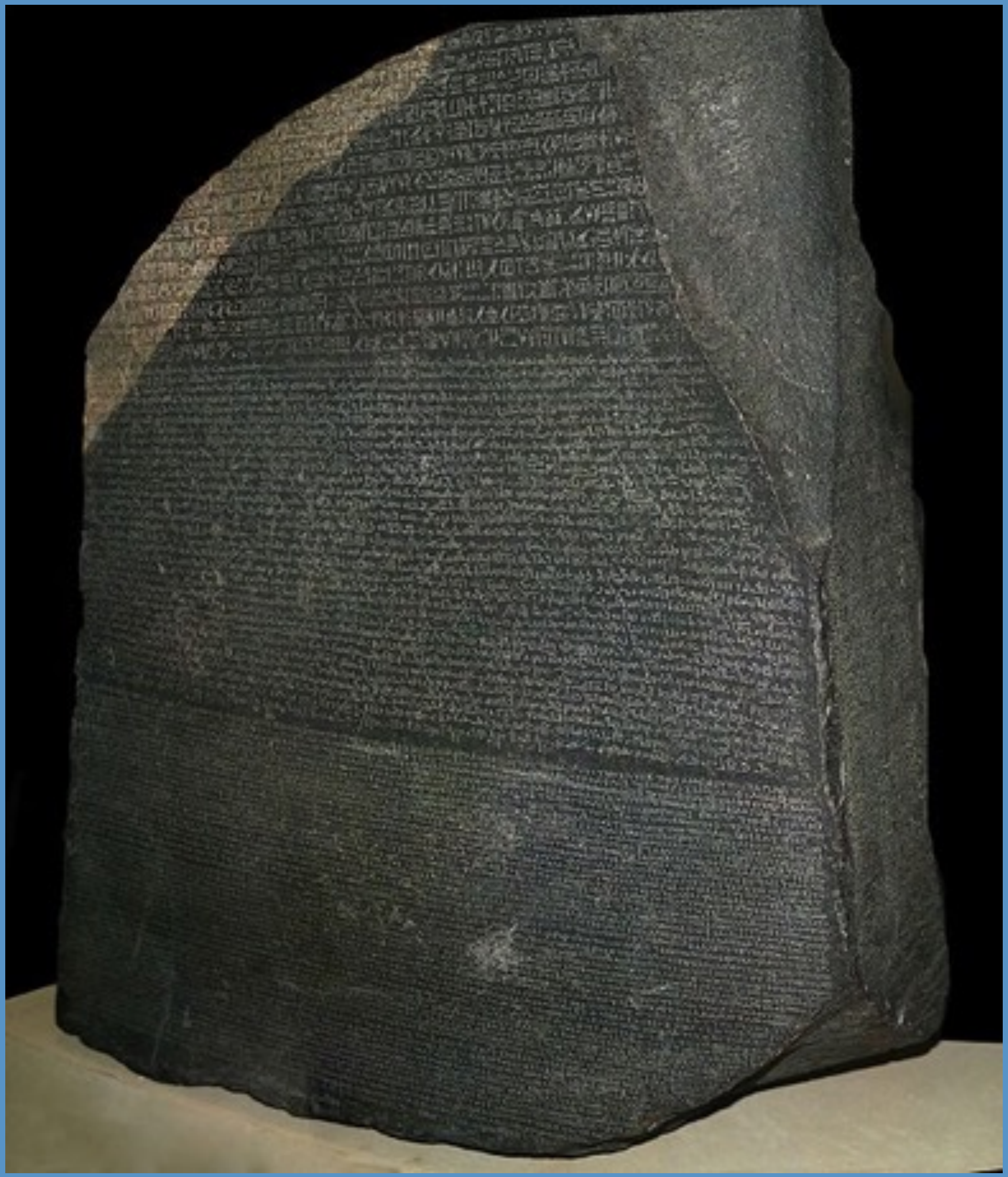
What is Cryptography?

- lit., *the art of secret* (κρυπτός, *kryptos*) *writing* (γράφειν, *graphein)*
- related but different from:
  - *cryptology (linguistic, codebreaking)*
  - *cryptanalysis (statistical analyzing)*

What is Cryptography?

- confidentiality: the ability to send information between participants in a way that prevents others from reading or intercepting it
- integrity: reassuring the recipient of a message that the message has been altered
- authentication: verifying the identity of a person or machine
- non-repudiation: assurance that someone cannot deny something

Classic Cryptography



Terms

- Plaintext: the message in its original form (aka *cleartext*)
- Ciphertext: the message in its mangled form
- Plaintext -> Ciphertext = Encryption
- Ciphertext -> Plaintext = Decryption

Basic Attacks

- Ciphertext Only
  - attacker has seen enough ciphertext to analyze or reverse engineer over time
  - susceptible to dictionary attacks if the password isn't strong
    - rather than try all 2^n possible keys (2^{32} = 4,294,967,296)
    - …try 470,000 or so English words
  - any cryptographic algorithm must be secure against this type of attack

Basic Attacks

- Known Plaintext
  - could make life easier for the attacker
  - with <plaintext, ciphertext> pairs, attacker could learn the mappings for a substantial fraction of the message
- Chosen Plaintext
  - similar but more powerful because the sample space of <plaintext, ciphertext> pairs is larger!

Shannon’s Maxim

- Frequency analysis was a powerful technique against many ciphers. Breaking a message otherwise required knowledge of the cipher used
- This made espionage, bribery, burglary, defection, etc., more attractive approaches.
- In the 19th century
    - secrecy of a cipher's algorithm is not a sensible nor practical safeguard of message security
    - Instead, security of a secret value (the *key*) alone should be sufficient for a good cipher to maintain confidentiality
    - 'the enemy knows the system’

Modern Cryptography

- based on computational difficulty
- it's not impossible to break without the key
  - try all the keys!
- consider why a combination lock is effective

Symmetric

- aka *secret key cryptography*
- sender and receiver share 1 key
- the only kind of encryption publicly known until June 1976!

- Two types
  - block ciphers - encrypt plaintext in chunks, or *blocks*
    - e.g., DES, AES, 3DES,
  - stream ciphers - encrypt plaintext one character at a time
    - e.g., RC4

Applications

- Transmitting over an insecure channel
- Secure storage on insecure media
- Authentication
  - challenge-response with shared secret
- Integrity Check
  - cryptographic checksum
  - large interbank electronic funds transfers

Block Cipher Example

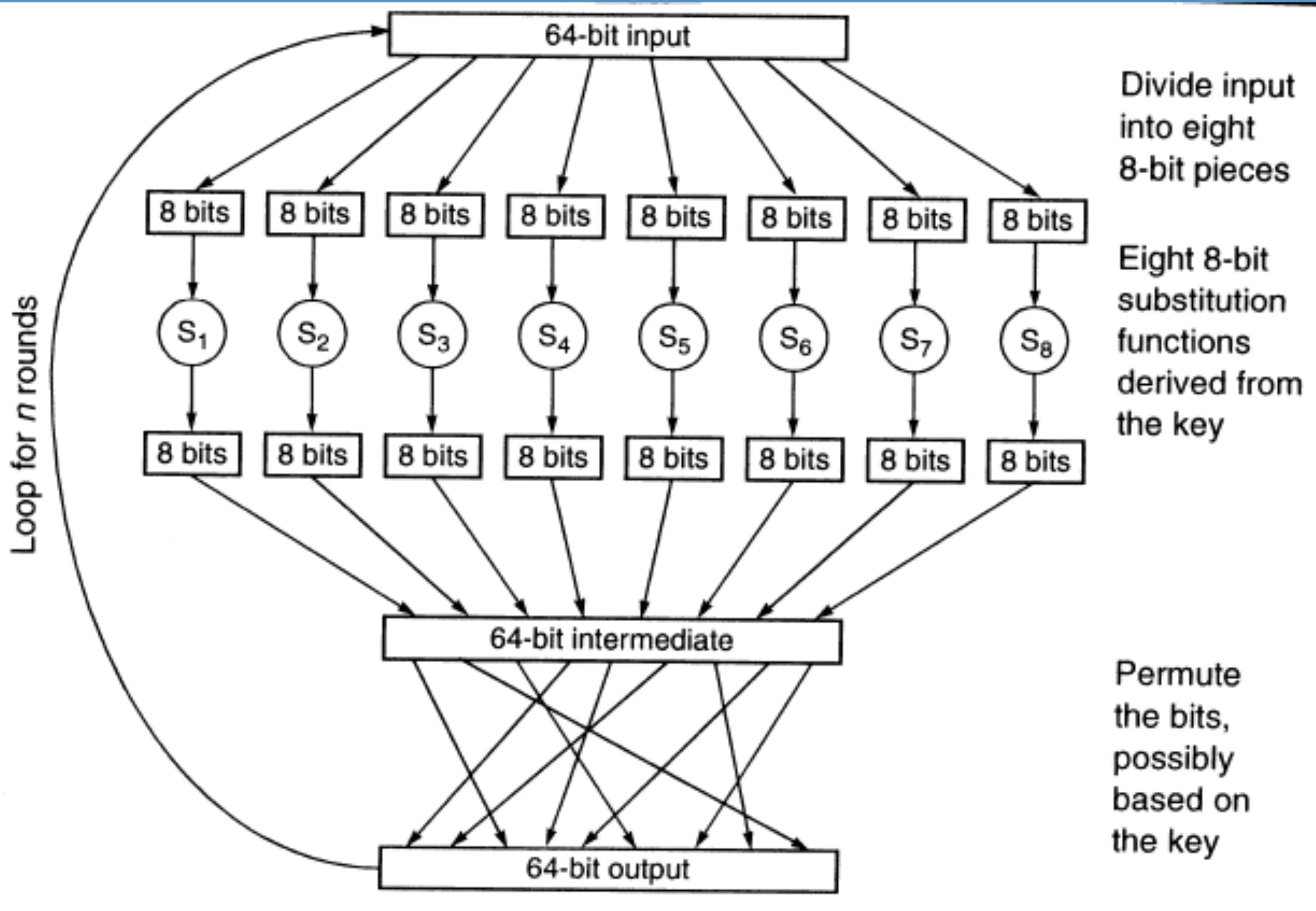


Figure 3-1. Example of Block Encryption

Block Cipher Example

- Note this algorithm only works for 64 bit inputs…
- What if your message is more than that?
- What if your message is less than that?
- We'll need to provide a *mode of operation* to this algorithm

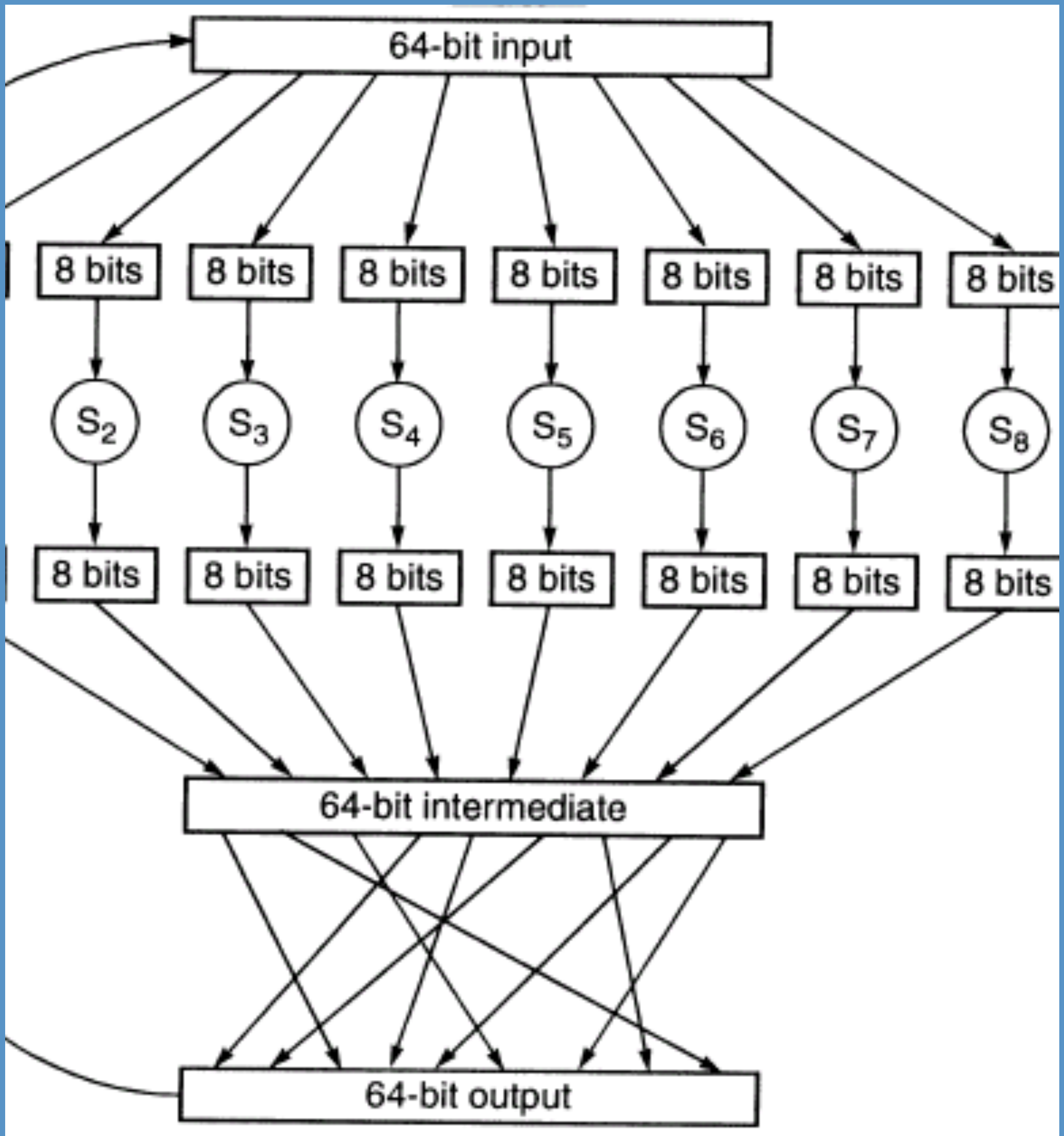


Figure 3-1. Example of Block Encryption

Mode Example

- Break message into 64-bit chunks and encrypt each chunk with the secret key?
    - Electronic Code Book (ECB)
    - subject to known ciphertext attacks!
- Instead use random numbers for each chunk
    - Cipher Block Chaining (CBC) uses an Initialization Vector (IV) as its random number
- Other examples exist…CBC most common for block modes, we won't talk about stream modes

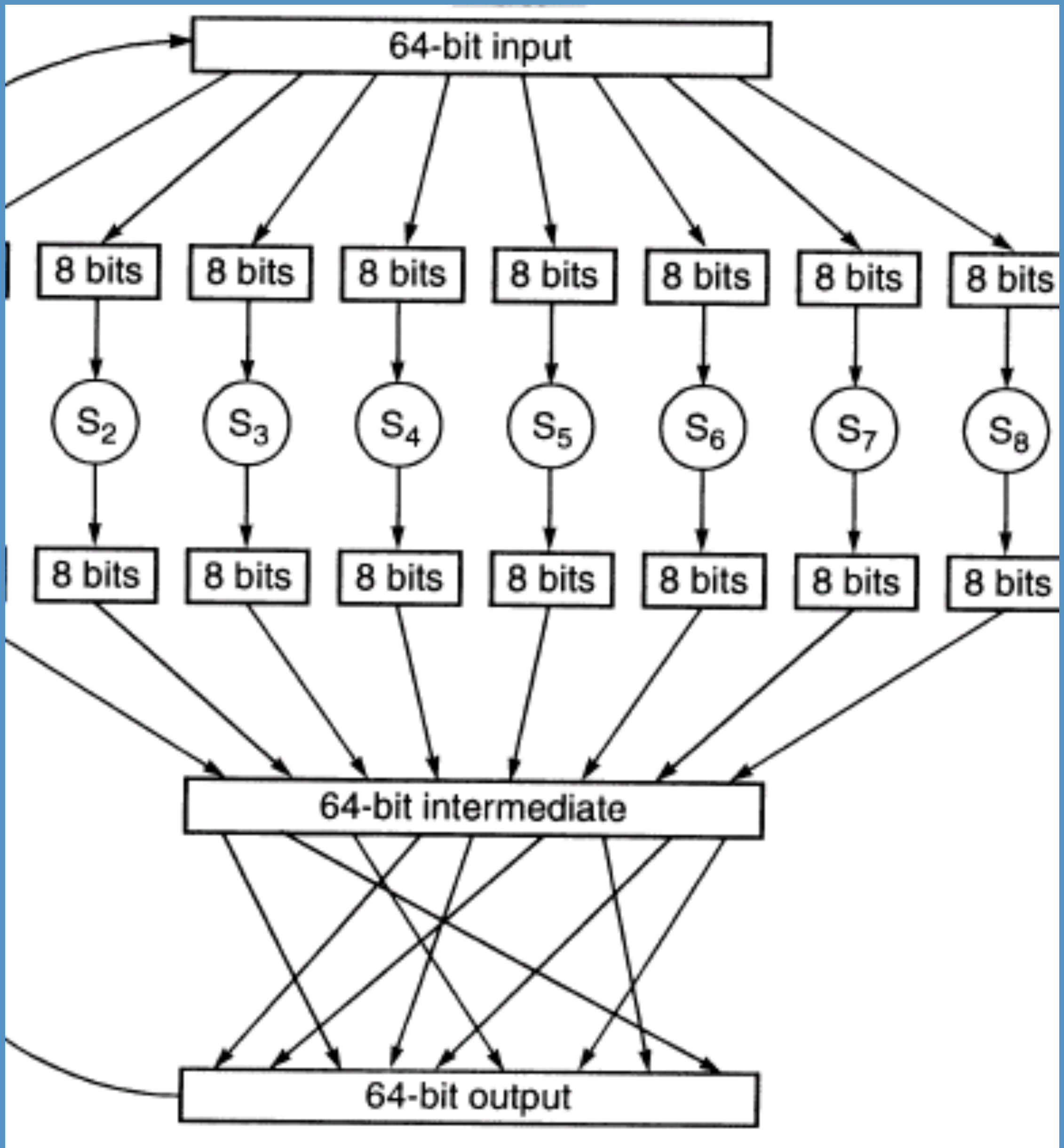


Figure 3-1. Example of Block Encryption

Using Symmetric Encryption

- Using 3DES (symmetric block cipher)
  - encrypt: openssl des3 -a -in hellow.txt -K 0 -iv 2
  - encrypt: openssl des3 -a -in hellow.txt -k password
  - decrypt: openssl des3 -d -a -in hellow.des3 -k password

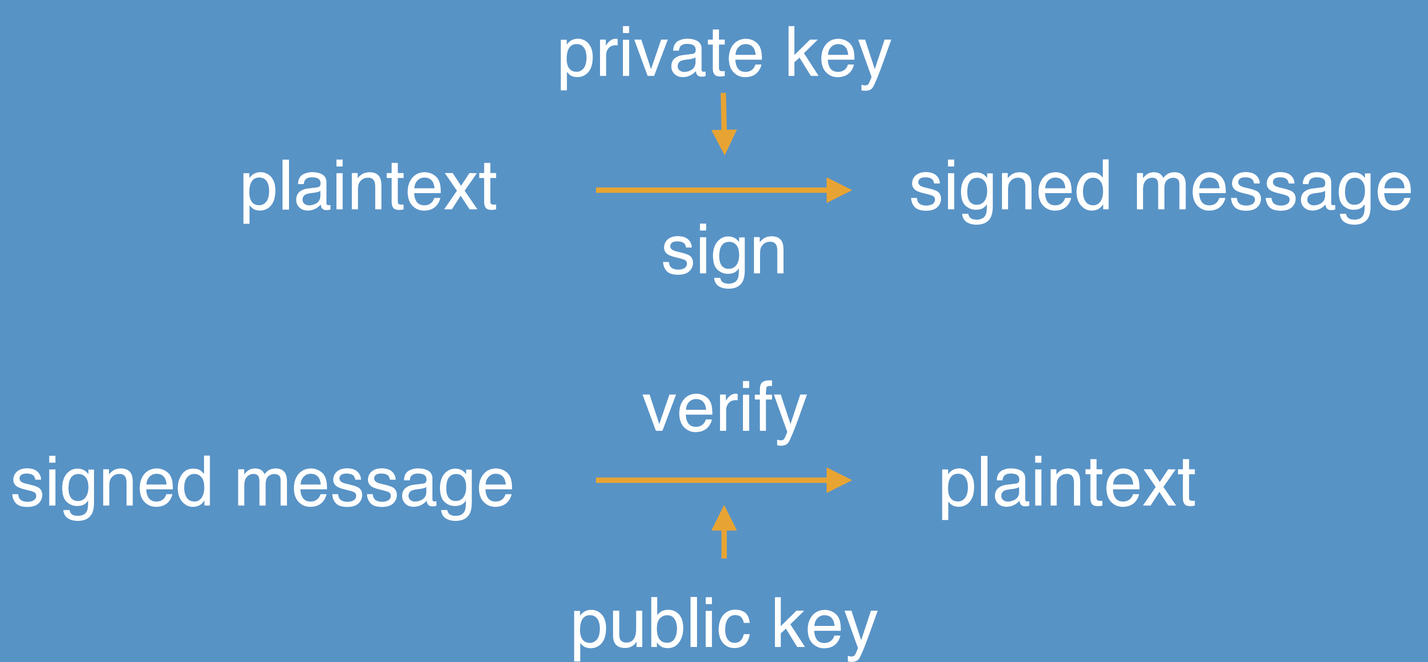
Asymmetric

- aka *public key cryptography*
- invented in 1975
- unlike symmetric, no shared key!
- each individual has 2 keys, a shared *public key* & a secret *private key*

$$\text{plaintext} \xrightarrow{\text{public key}} \text{ciphertext}$$

$$\text{ciphertext} \xrightarrow[\text{private key}]{} \text{plaintext}$$

Digital Signature



Digital Signature

- another type of integrity check
- verification of a signature only requires knowledge of the ***public key***
  - e.g., Alice signs with her ***private key***, everyone can verify that it's Alice using her ***public key***, but cannot forge her signature

Example: RSA

- Choose two large primes p and q (256 bits each). Keep them secret!
  - In class, we used $p = 31$ and $q = 13$.
- Multiply them: $p * q = n$
- $\phi(n) = (p - 1)(q - 1)$
- To generate public key
  - choose a number e that is relatively prime to $\phi(n)$ (in class, $e = 11$)
  - public key = $<e, n>$
- To generate private key
  - $d = 1 / (e \bmod \phi(n))$
  - private key = $<d, n>$

Why does this work?!

Factoring large primes is HARD!

…we hope…

Applications

- Transmitting over an insecure channel
- Secure storage over insecure media
  - added benefit: Alice can encrypt for Bob without knowing his private key!
- Authentication
  - No secret information necessary for verification!
- Digital signatures
  - provides **non-repudiation!**
  - sharing a secret key makes it impossible to verify

Symmetric vs Asymmetric

- Public key algorithms are MUCH slower than secret key algorithms…
- …as a result, we'll see a combination of the two used in practice
- Security based on public key algorithms more easily configurable

Hash Functions

- not the same as hash table hash functions, despite similar concept
- aka *message digests* or *one-way transformations*
- no key involved!
- converts message into a short fixed-length *hash*
    - e.g., MD5, SHA-1, SHA-3
- cannot reverse-engineer
    - given *h(m)*, can't determine *m* without trying all possibilities
- while many values of *m* can transform to *h(m)*, it's *computationally infeasible!*

Applications

- Password hashing
  - systems doesn't store passwords unencrypted
  - not required to *know* to verify correctness
- Message Integrity
  - keyed hash - *h(message + secret)*
  - **not secure to send** *h(message) + message*
- Message Fingerprint
  - rather than track versions using full files, track hashes
  - Git commits!

Applications

- File Download Security
  - Ensure the software you download isn't corrupted
  - https://httpd.apache.org/download.cgi#verify
- Digital Signature Efficiency
  - public key algorithms take lots of CPU time
  - sign(h(message)) more efficient than sign(message)

Using Hash Digest

- Using SHA1 (hash function):
    - openssl dgst -sha1 hellow.txt

Certificates

- Key distribution is easier with public key crypto.
- But secret key crypto is faster!
- Suppose we use public/private to exchange secret?
  - that works, but how do we trust that the public key belongs to who we think it does?
- A *Certificate Authority (CA)* generates a *certificate*, or a signed message specifying a name, an expiration date, and its public key.
  - But everyone needs to know the CA's public key to verify it…
  - A compromised CA would be disastrous!
    - https://www.schneier.com/blog/archives/2012/02/verisign_hacked.html

Certificate Generation

- User generates a public and private key-pair or is assigned a key-pair by some authority in their organization
- User requests the certificate of the CA Server
- CA responds with its certificate
    - includes its public key and its digital signature signed using its private key
- User provide information requested by the CA Server to obtain its certificate
    - e.g., email address, fingerprints, etc. that the CA needs to be certain that User claims to be who he/she is

Certificate Generation

- User sends a certificate request to the CA with his/her public key and additional information
  - signed by CA's public key
- CA gets the certificate request
  - verifies User’s identity
  - generates a certificate, binding User’s identity and public key
  - signature of CA verifies the authenticity of the Certificate
- CA issues certificate to User

PKI

- How are certificates retrieved?
  - Public Key Infrastructure (PKI)
  - Network protocol negotiation
  - Certificate Authorities
    - Mac OSX Keychain
    - Browsers
    - Android/iOS

PKI

- How are they validated?
  - Public Key Infrastructure (PKI)
  - issuer - the signer of the certificate
  - subject - the applicant
  - verifier - one who evaluates a certificate chain (aka relying party)
  - principal - one who has a public key
  - trust anchor - a public key in the chain that is trusted

Demo:
Certs on Mac OS,
Firefox, Android

Fun with Certs

- Analyzing certificates
  - echo | openssl s_client -connect www.google.com:443 2>/dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p’
- Generate self-signed certificates
  - openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout mycert.pem -out mycert.pem
- Generate certificate request
  - openssl req -new -newkey rsa:1024 -nodes -keyout mykey.pem -out myreq.pem

Certificate Revocation

- Certificates expire, but what about the “disgruntled employee scenario”?
- Solution similar to credit cards
  - Certificate is valid IF
    - valid CA signature
    - has not expired
    - not listed in the CA’s Certificate Revocation List (CRL)

Certificate Pinning

- Sometimes, CAs fail us…
  - 2011 DigiNotar attack signed fraudulent certificates for google.com
- We depend heavily on trusting the CAs of the host platform
  - Is that enough? For your app, it may not be…
- A more paranoid approach: certificate pinning
  - Take the problem on yourself
  - If the certificate of the server you're connecting to isn't among your whitelist, disallow!
  - Use sparingly, updating your server certificates will require updating your app…

Certificate Pinning

```
public CertificatePinning() {
  client = new OkHttpClient();
  client.setCertificatePinner(
      new CertificatePinner.Builder()
          .add("publicobject.com", "sha1/DmxUShsZuNiqPQsX2Oi9uv2sCnw=")
          .add("publicobject.com", "sha1/SXxoaOSEzPC6BgGmxAt/EAcsajw=")
          .add("publicobject.com", "sha1/blhOM3W9V/bVQhsWAcLYwPU6n24=")
          .add("publicobject.com", "sha1/T5x9IXmcrQ7YuQxXnxoCmeeQ84c=")
          .build());
}

public void run() throws Exception {
  Request request = new Request.Builder()
      .url("https://publicobject.com/robots.txt")
      .build();

  Response response = client.newCall(request).execute();
  if (!response.isSuccessful()) throw new IOException("Unexpected code " +

  for (Certificate certificate : response.handshake().peerCertificates())
    System.out.println(CertificatePinner.pin(certificate));
  }
```

https://github.com/square/okhttp/wiki/HTTPS#certificate-pinning

Attacks

- Denial of service
- Spoofing
- Clickjacking
- Phishing
- Decompilation
  - JADX
  - Proguard - obfuscation

Attacks

- Insecure Data Storage (PII)
- Unintended Data Leakage
- Broken Cryptography
- and more:
    - Bulletproof Android by Godfrey Nolan: https://www.youtube.com/watch?v=WMDR0Qs6WRI

References

- http://www.tldp.org/HOWTO/SSL-Certificates-HOWTO/x64.html
- https://www.openssl.org/docs/manmaster/apps/enc.html
- https://www.madboa.com/geek/openssl/
- https://httpd.apache.org/download.cgi#verify
- https://github.com/square/okhttp/wiki/HTTPS#certificate-pinning
- https://publicobject.com/2014/06/09/pinning-ssl-certificates/
- https://www.youtube.com/watch?v=WMDR0Qs6WRI

Homework

PRO TIP : review slides for sample OpenSSL commands and links

- Decrypt this text using the *des3* symmetric block cipher. Provide the command you used.
- Download Apache here and verify that its MD5 is cf4dfee11132cde836022f196611a8b7. Provide the command you used.
- Who issued this cert? To whom was it issued? Provide the commands you used.
  - *NOTE: https://blog.mozilla.org/security/2014/09/08/phasing-out-certificates-with-1024-bit-rsa-keys/*
- Use jadx to decompile one of the final project Android apps of your C4Q colleagues. Identify any interesting security holes.

Exit Ticket

- See Slack channel