Android Authentication in the Web World

DAVID TRUXALL, PH.D.

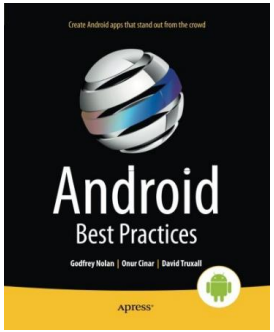
https://bit.ly/devfest-17

About Me

DXC.technology

@davetrux

spnsurvivors.org



https://davidtruxall.com

You

Are familiar with Android

Might be a web developer

Probably have secured web sites at work

Want to use an existing user identities

Not an Identity Management guru

Don’t want to invent a new identity store

Goal

Understand how web-based authentication methods work

Be able to consume them using Android

Improved understanding of security

Agenda

1. Basic security
2. HTTP basics
3. Types of web-based authentication
4. Consumption using Android
5. Certificate Pinning

Basic Security

Encoded

Hashed

Encrypted

Encoded

Hex

ASCII

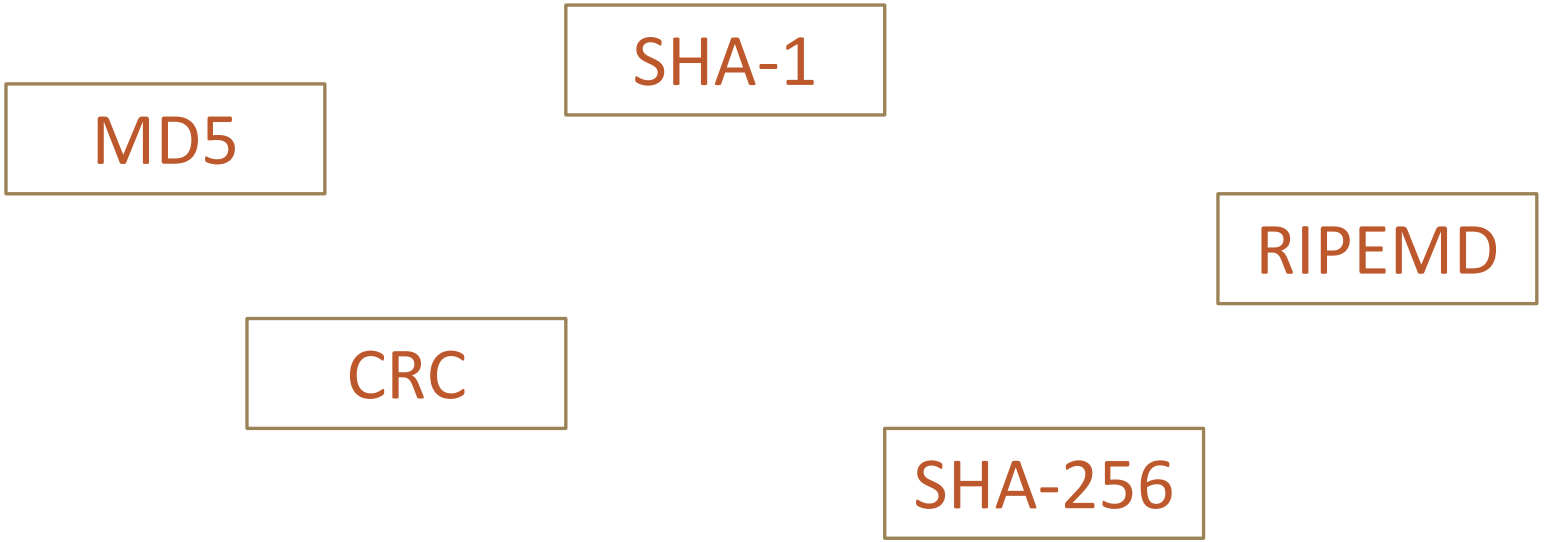
EBCDIC

Base64

Unicode

URL

Hashed



Encrypted

Symmetric/Private key

Asymmetric/Public Key

Other Terms

Nonce

Token

Security

Store the password

Send the password

Own the password

Your own encryption scheme

Security

Use transport security (SSL/TLS)

Implement sessions

Store on the server not client

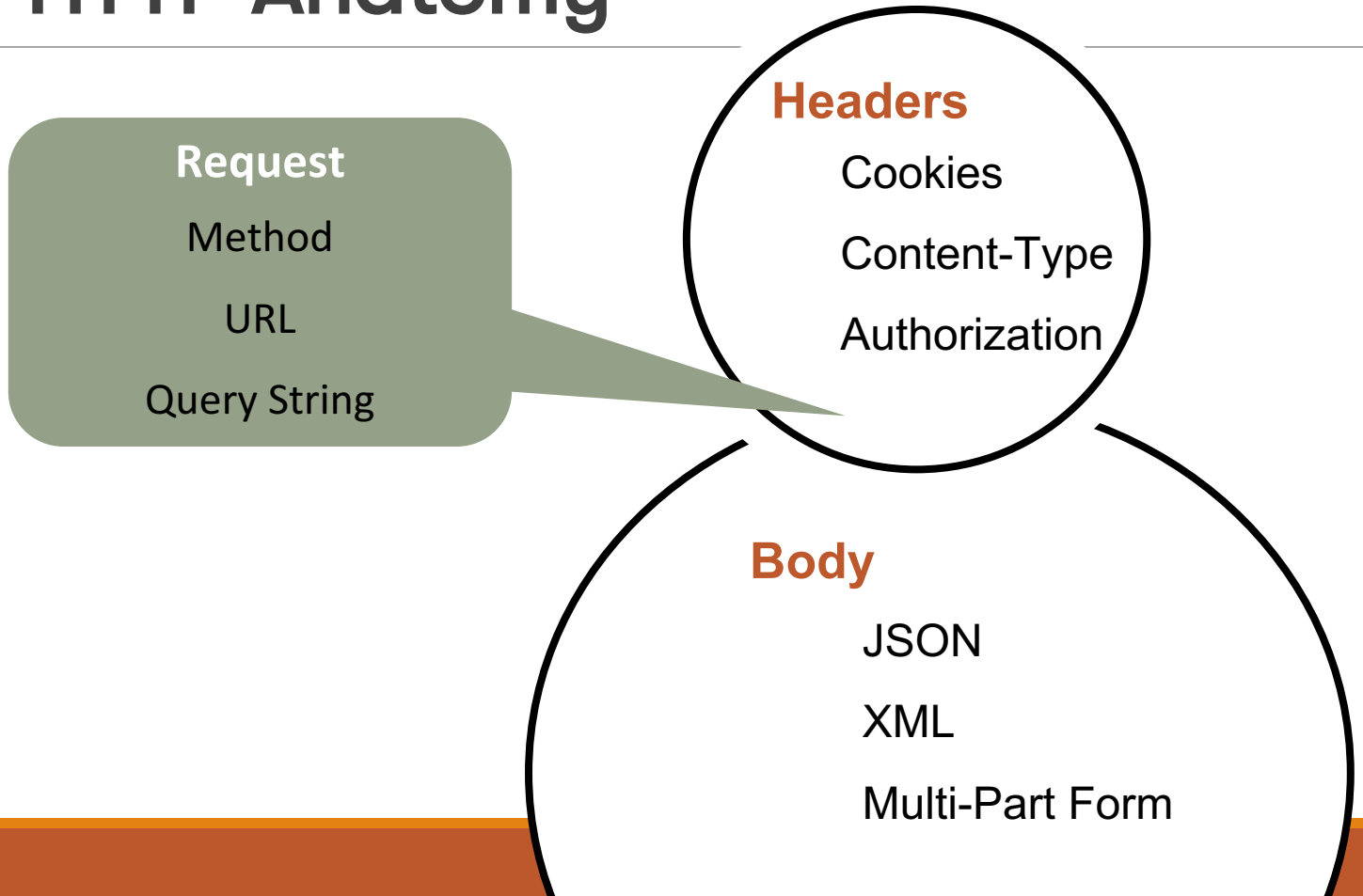
Use Certificate Pinning


https:// != SSL


TLS


SSL

HTTP Anatomy



HTTP Methods

POST

DELETE

OPTIONS

GET

PUT

HEAD

HTTP Headers

▼ **Request Headers** view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Authorization: NTLM TlRMTVNTUAADAAAAGAAYAHAAAAAYABgAiAAAAAAAAABAAAAAEgASAEAAAAAeAB4AUgAAAAA
MgAzADAANgA0AE0AQQBDAE4AMAAxAOKCTaPM89wvAAAAAAAAAAAAAAAAAAAAAP3DUycoty/Z2Pbj8ks4N29XarHbI
Cache-Control: max-age=0
Connection: keep-alive
Content-Length: 3547
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryvybIOQOGhl0ceumG
Cookie: _mkto_trk=id:352-NVO-562&token:_mch-compuware.com-1363106194471-64626

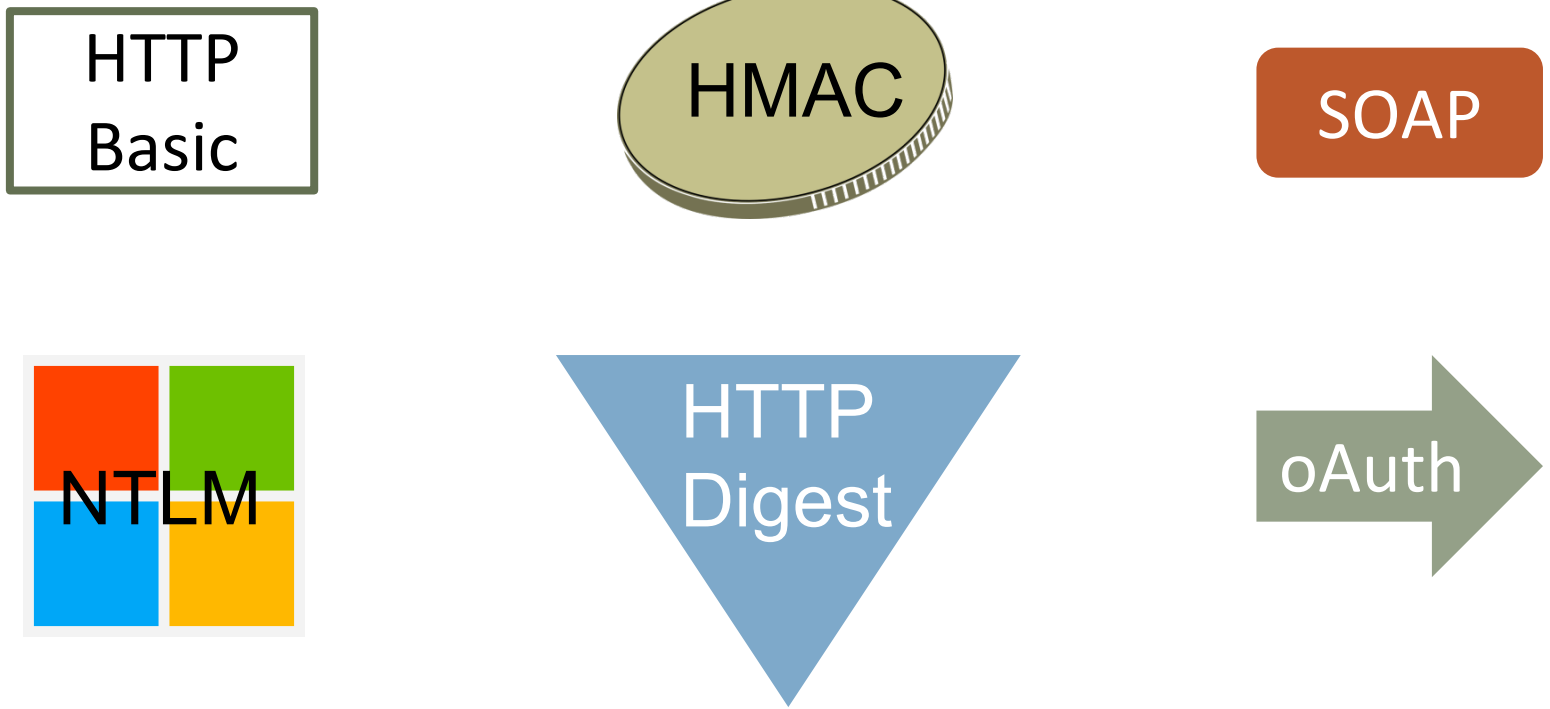
HTTP Body

▼ **Request Payload**

```
------WebKitFormBoundaryvybIOQOGhl0ceumG
Content-Disposition: form-data; name="Id"

0
------WebKitFormBoundaryvybIOQOGhl0ceumG
Content-Disposition: form-data; name="photoUpload"; filename="Red Apple.gif"
Content-Type: image/gif


------WebKitFormBoundaryvybIOQOGhl0ceumG
Content-Disposition: form-data; name="FirstName"

Test
------WebKitFormBoundaryvybIOQOGhl0ceumG
Content-Disposition: form-data; name="LastName"

Data
```

Response Codes

Web Authentication Protocols



HTTP Basic

Weakest security-wise

Clear text - Encoded

Relies on TLS

HTTP Basic

1. Concatenate username and password
2. Encode them in Base64
3. Prefix this string with ‘Basic’
4. Add as Authorization HTTP header

Authorization: Basic YWRtaW46cEBzc3cwcmQ=

HTTP Digest

Stronger than Basic

No requirement for TLS

Password not sent

Uses MD5 hashing

HTTP Digest

Server sends nonce, opaque and realm

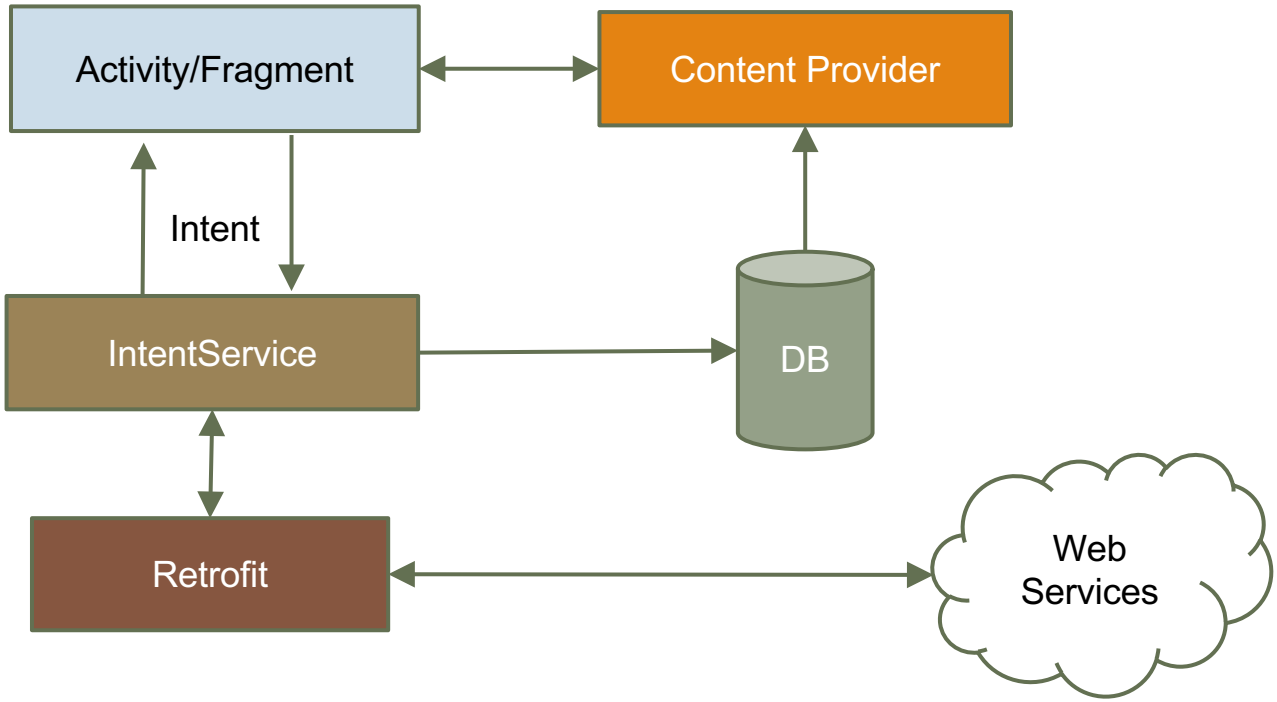
A1 = MD5(“username:realm:password”)

A2 = MD5(“method:uri”)

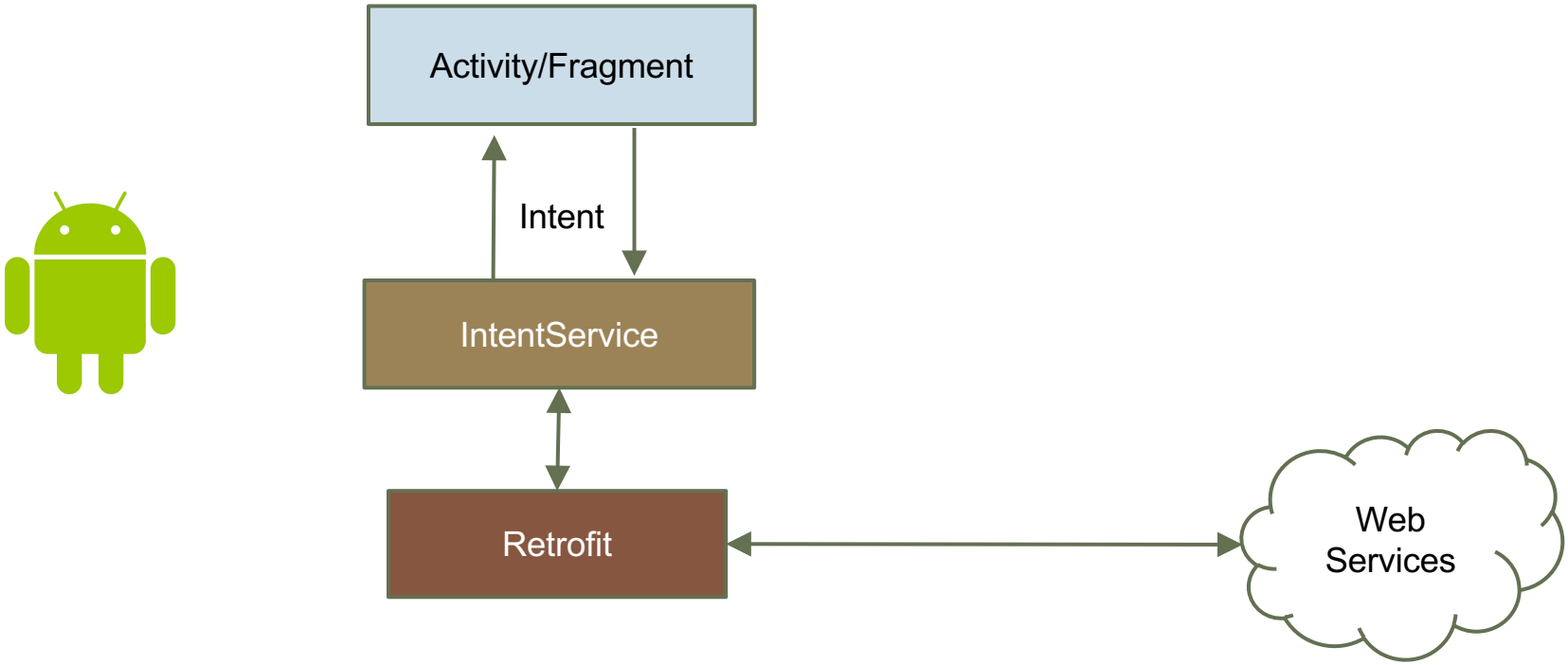
response = MD5(A1:nonce:A2)

Authorization: Digest username="%s", realm="%s", nonce="%s", opaque="%s", uri="%s", response="%s"

IntentService Pattern



IntentService Pattern



Retrofit (OkHttp)

```
public interface GitHubService {
   @GET("users/{user}/repos")
   Call<List<Repo>> listRepos(@Path("user") String user);
}
Retrofit retrofit = new Retrofit.Builder()
        .baseUrl("https://api.github.com/")
        .build();
GitHubService service = retrofit.create(GitHubService.class);

Call<List<Repo>> repos = service.listRepos(”davetrux");
```

Code

Microsoft-based

Active Directory

No requirement for TLS

Password not sent

Better than Digest

Most complicated

Apache client deprecated

Need 3rd party library

Gradle trick

oAuth

the code is more what you'd call "guidelines" than actual rules


oAuth
Google
amazon

oAuth

Web Site

Facebook, Twitter, Google, etc.

Access URL

302 Redirect to Service

Login to Service

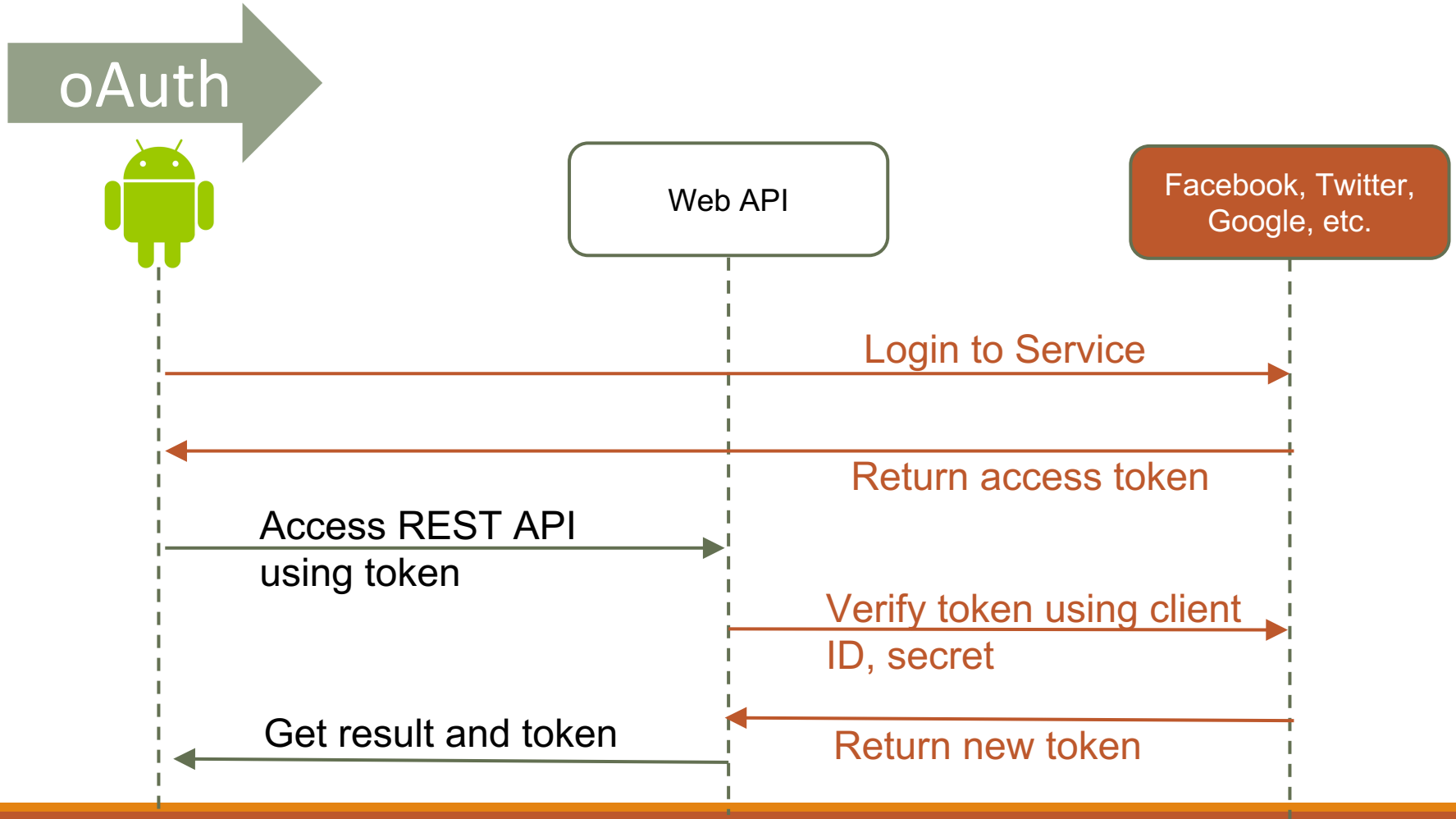
302 from Service to app w/auth code

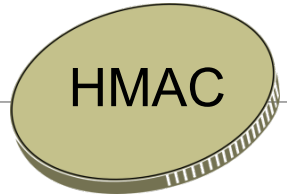
Access redirect URL

Verify auth code using client ID, secret

Return access token

Logged In


oAuth
Web API
Facebook, Twitter, Google, etc.
Login to Service
Return access token
Access REST API using token
Verify token using client ID, secret
Return new token
Get result and token



Hash-based Message Authentication Code

Guarantee authenticity of message

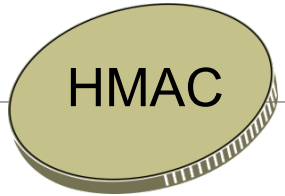
A shared secret key – both client and server

No need for SSL

AWS

Password not sent

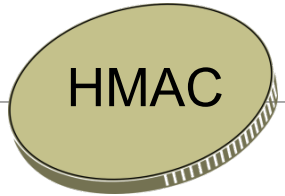
Authorization: HMAC trux:44CF006BF252F707:jZND/A/f3hSvVzXZjM2HU=



Define a request
POST /customer/ { id: 123, orders: 6, ...}

Create a signature using shared key
base64(hmac-sha1(verb + headers + content + nonce)

Add as authorization header
Authorization: HMAC userName:signature



Retrieve key from DB based on userName

Recreate signature based on request
base64(hmac-sha1(verb+ headers + content)

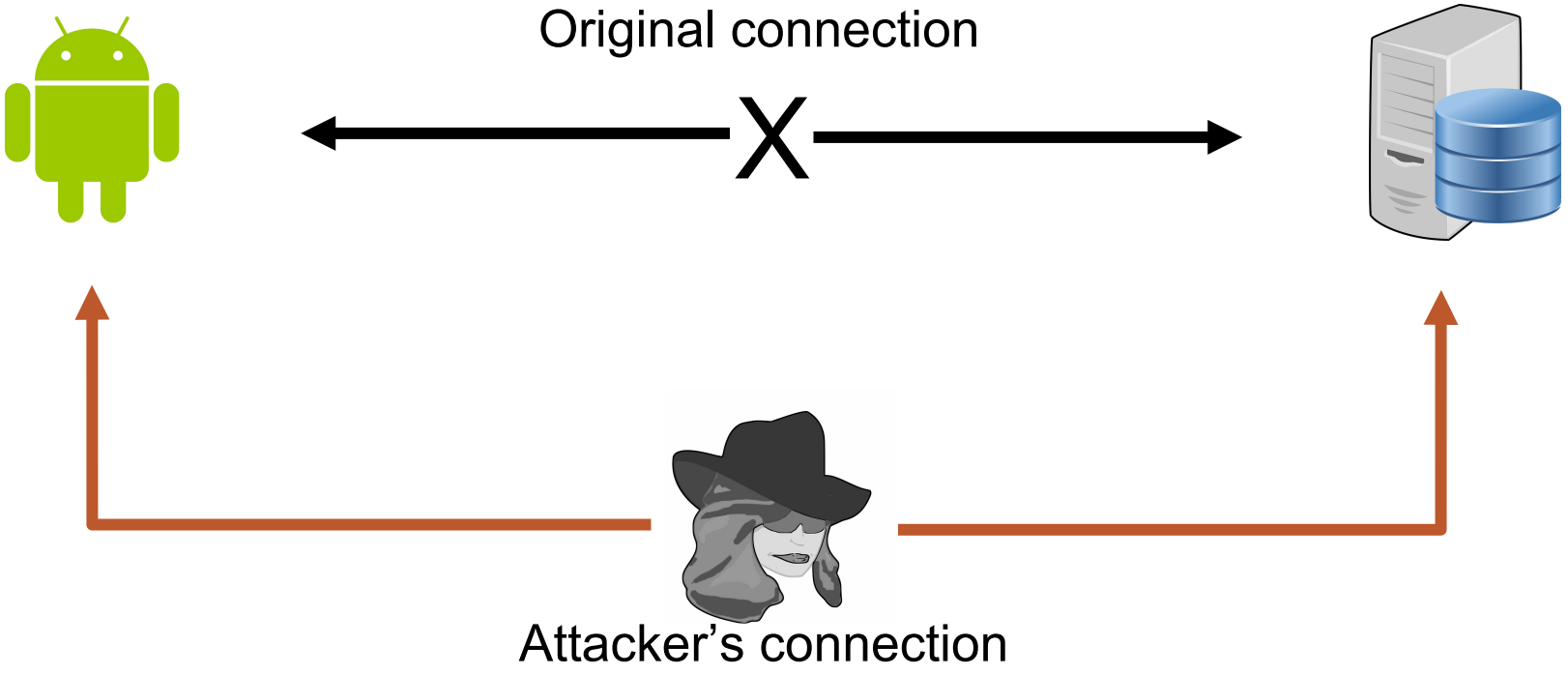
Compare signatures

No key is safe

Code

Certificate Pinning

Man in the Middle Attack



Code

Takeaway

Every authentication method has weaknesses

- Understand then choose

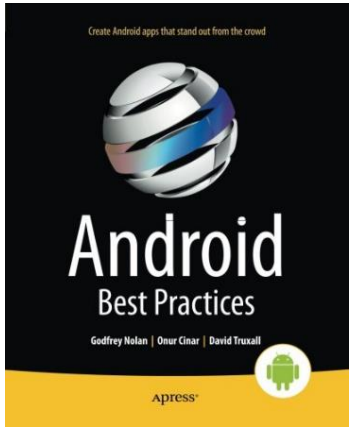
All usable by Android apps

No key is safe

Don’t re-invent the wheel

Be safe out there, use TLS and Pinning

Shameless Plug



spnsurvivors.org

https://davidtruxall.com

https://bit.ly/devfest-17