CASE STUDY: HOW WE'VE SECURED A FINANCIAL MOBILE APP USING A CLOUD COMPUTING BACKEND WITHOUT PENALIZING THE USER EXPERIENCE

Anderson Dadário, Cândido Sales, Márcio Rosa

¿Quiénes somos?

Anderson Dadario

Founder of Gauntlet.io, a platform to identify vulnerabilities in web apps, servers and source code, manage them and also take action. Besides that he is an official instructor for (ISC)² CISSP and CSSLP.

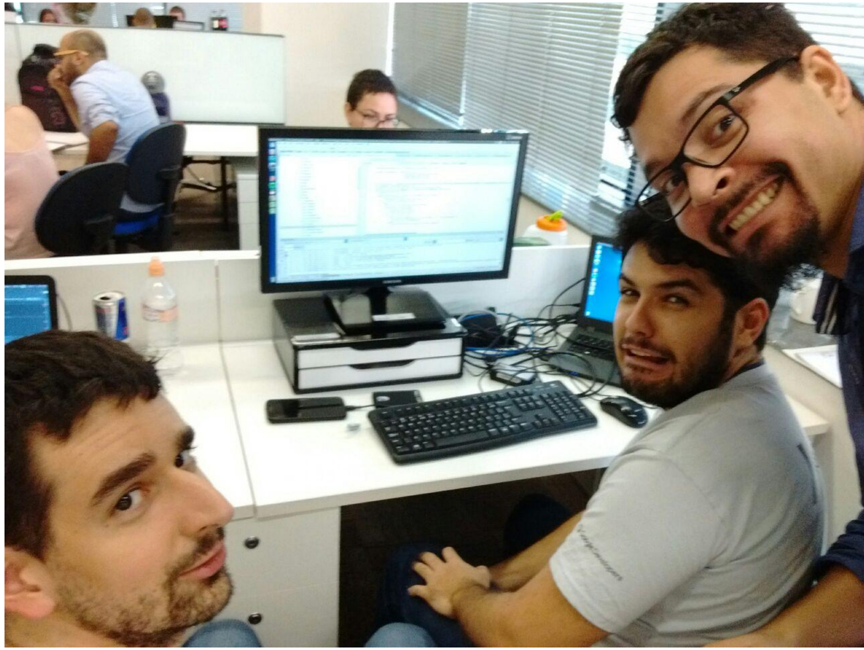
Cândido Sales

I work at Brasil/CT, and was responsible for translating mockups to functionalities, designing and implementing the application's security layer in the Android version.

Special thanks to Márcio Rosa

Because he:

- Implemented security controls for Backend;
- Implemented certificate automation with custom key pair for TLS Pinning;
- Spared Mr. Cândido for this talk;
- Made this presentation possible.



Bank Level Security

- We worry about security …
- But how to protect information and systems from skilled and tenacious attackers?

The Challenge

- “Sign up” happens in the mobile app, 100% digital - no agencies
- Client needs to talk to Server
- App needs to:
  - Store Credit Card
  - Allow Payments in the app
- App process/transmits/stores:
  - Personal Identifiable Information (PII)
  - Personal Financial Information (PFI)

Mapping Threat Actors

- Social Engineer
- Android Application Reverse Engineer
- Attacker from China (all internet attacks to backend only)
- Attacker with malicious app installed
- Attacker with access to user’s email
- Attacker with access to user’s mobile phone SMSes
- Attacker in the same network as the victim’s phone (network attacks only)
- Attacker with victim’s locked phone (robbery)
- Attacker with digital forensics capabilities, laboratory and budget
- Attacker with victim’s unlocked phone (away for bathroom)
- Attacker with O.S. admin user (root)

Main Threats

1. Data Leak, Integrity and Availability
2. Mobile Phone Robbery
3. Mobile Phone Data Dump
4. Mobile Number Clone
5. Man-in-the-middle
6. Payment without user’s consent
7. Reverse Engineering
8. Backend Attacks

Common Pitfalls

1. Think that attackers cannot access your mobile app’s source code
2. Forget about non-web vulnerabilities, e.g., mobile number clone
3. Security bugs on the backend

Time is limited

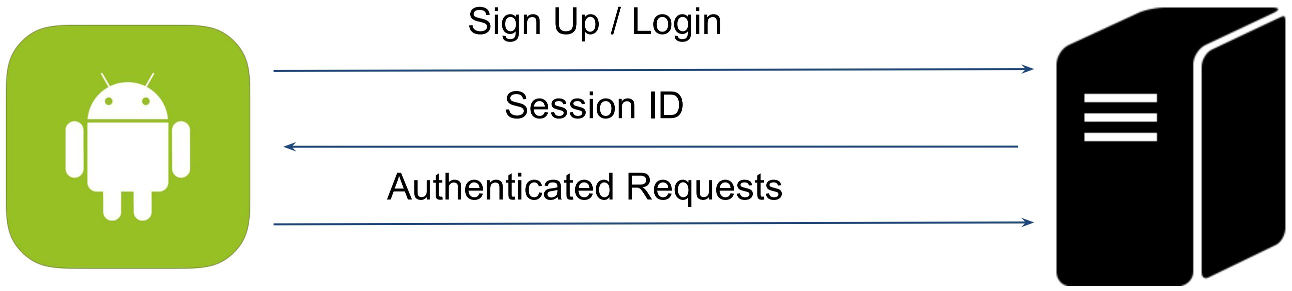
We will narrow our explanation to the designed

Authentication Scheme.

We, of course, had to implement more things, e.g., security flows, perform vulnerability assessment, validate libraries, verify dependencies against CVEs, etc.

Most Popular Architecture

» User installs mobile app



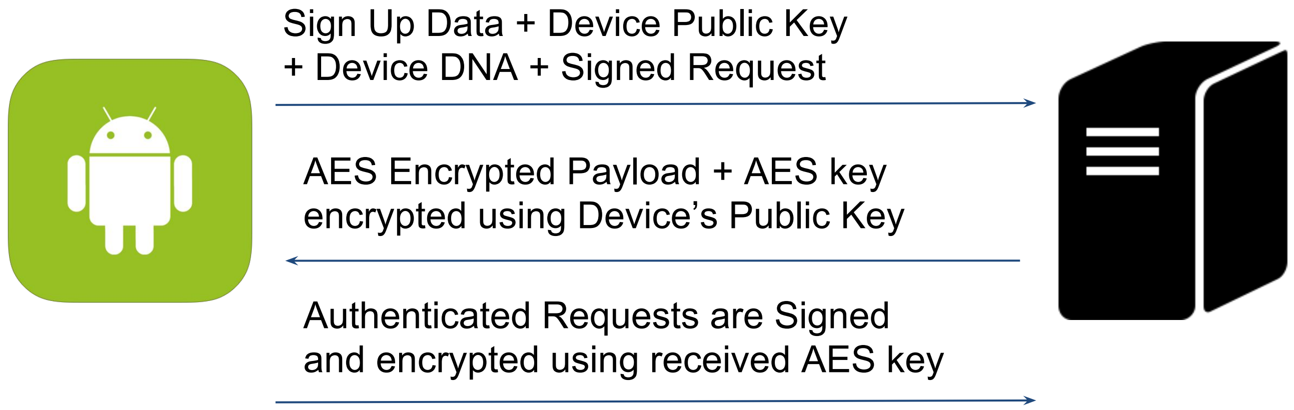
- All communications happen over HTTPS

What’s missing?

- Device authentication
- TLS Certificate / Public Key Pinning
- TLS deployment hardening
- No hard-coded secrets in source code
- Backend security
- Secure storage for session token
- Opportunities to implement additional controls (Defense in Depth)

Bank Level Architecture

- User installs mobile app
- **App generates a RSA keypair and stores on Android KeyStore**


Sign Up Data + Device Public Key + Device DNA + Signed Request
AES Encrypted Payload + AES key encrypted using Device's Public Key
Authenticated Requests are Signed and encrypted using received AES key

But implementing that is tough.

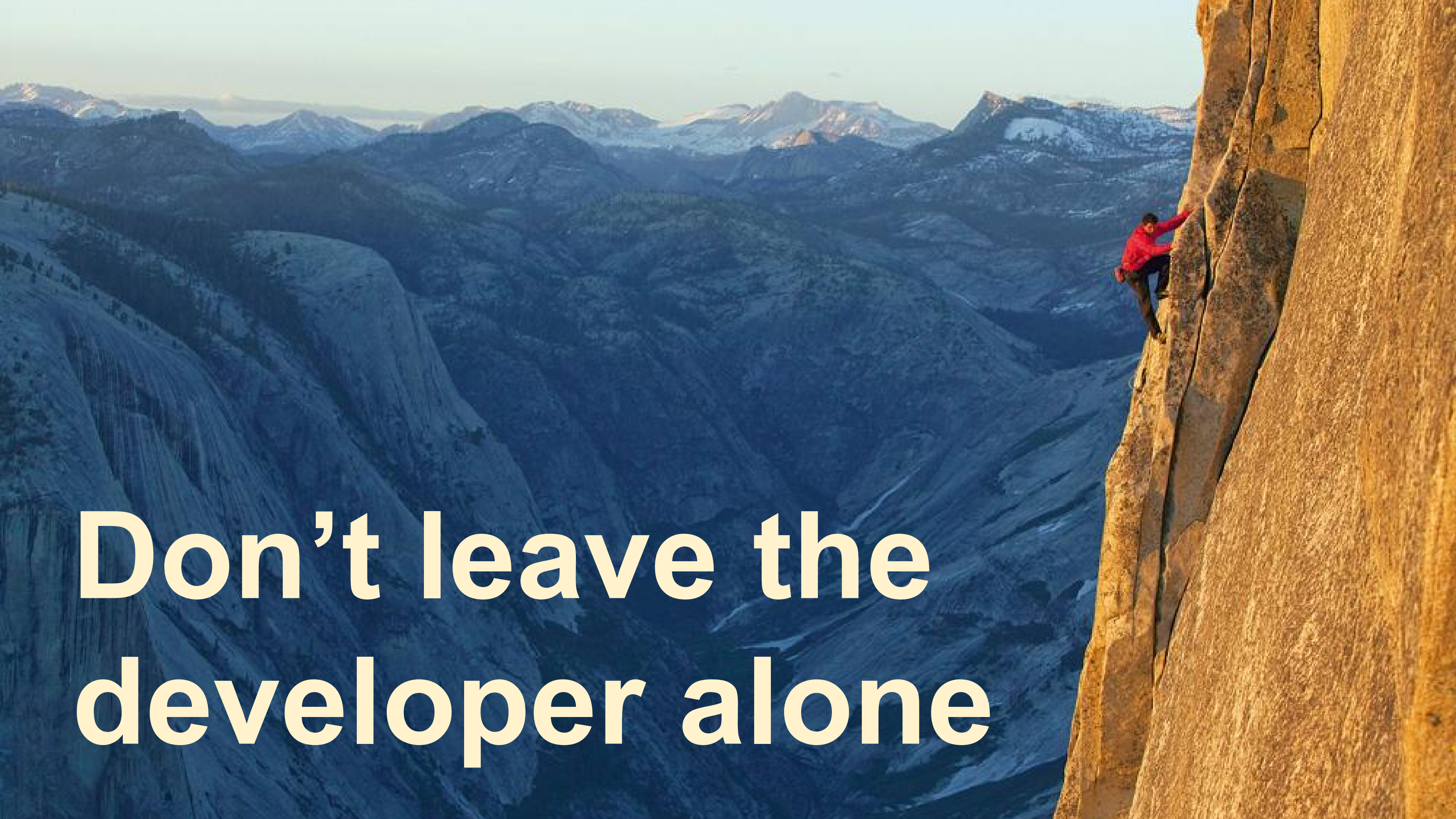
- Backend Team Challenges
- Android Team Challenges
- Awareness Challenges

Backend Challenges

- Propagate requests to services behind Mobile API Gateway
  - It should abstract cryptography
  - Lots of validations are required
    - validate request signature
    - decrypt AES payload
    - identify whether device is register / is trusted
    - validate session token
    - then we consider it a legitimate request
  - On top of all validations, performance is a must!
- Find reliable JWT library as JWT specification has some drawbacks;
- Libsodium is too heavy as a dependency to be used as a secure PRNG, but there aren’t better alternatives from what we’ve researched.

Android Challenges

- Cryptography support begins to become more “decent” only in recent Android API versions, necessary to use the BouncyCastle library
- Bouncycastle and its version problems
- Stackoverflow for crypto isn’t that mature
- Android Limitations (when compared to iOS in cryptography)
- Make software compatible to many Android versions
- Incompatibility to generate a JWT (Java) readable by the server (JavaScript)
- ProGuard implementation
- Android KeyStore from API 18 (4.3v)
- Create a secure storage mechanism for Android versions 4.1, 4.2 and 4.3.



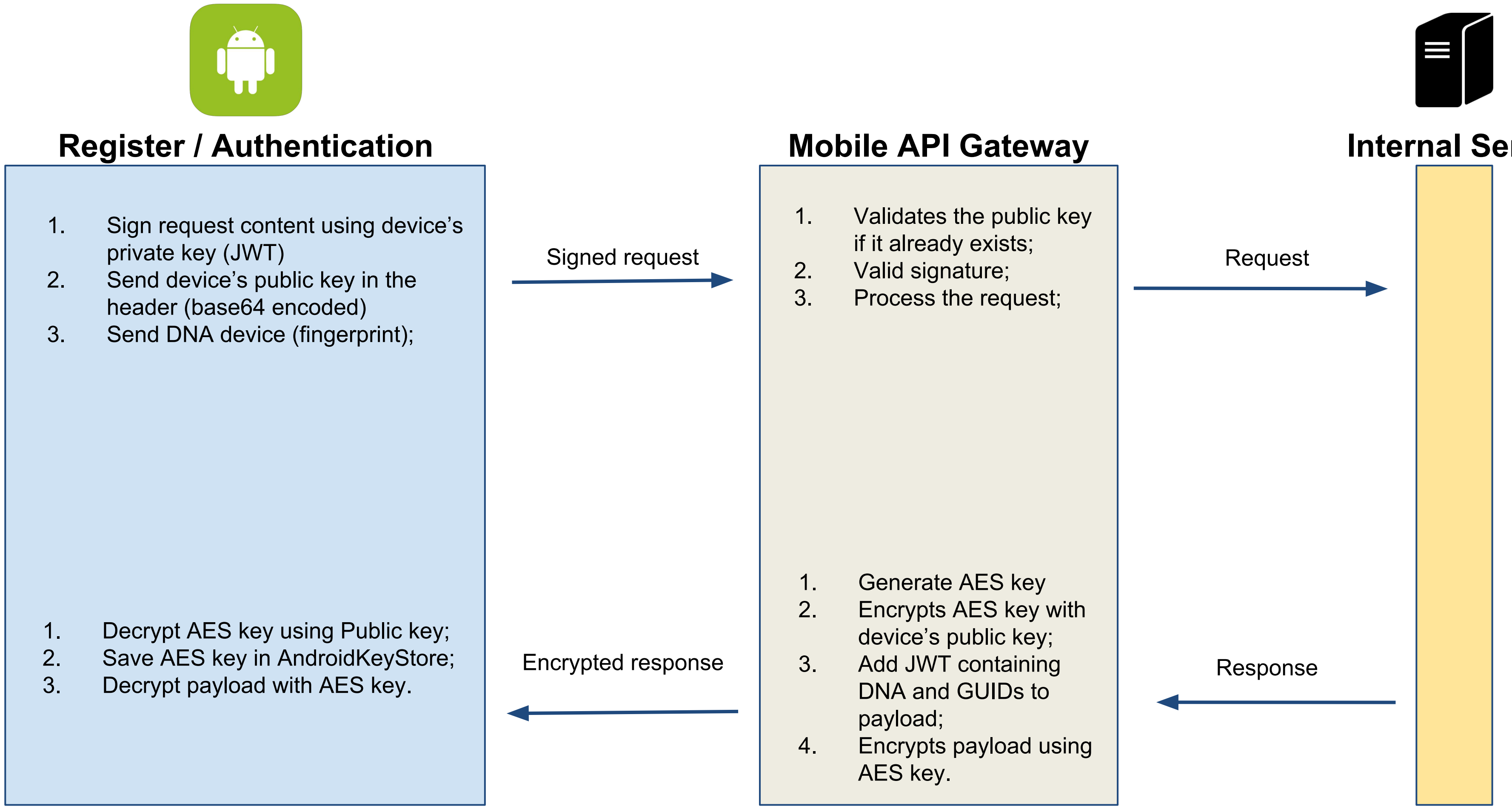
Don’t leave the developer alone

1st Step

1. Create RSA key pair
2. Securely store keys
   a. (4.3+): AndroidKeyStore
   b. (4.3-): Save them encrypted on Cache (SharedPreferences)

Mobile API Gateway

Internal Server



Register / Authentication

1. Sign request content using device’s private key (JWT)
2. Send device’s public key in the header (base64 encoded)
3. Send DNA device (fingerprint);

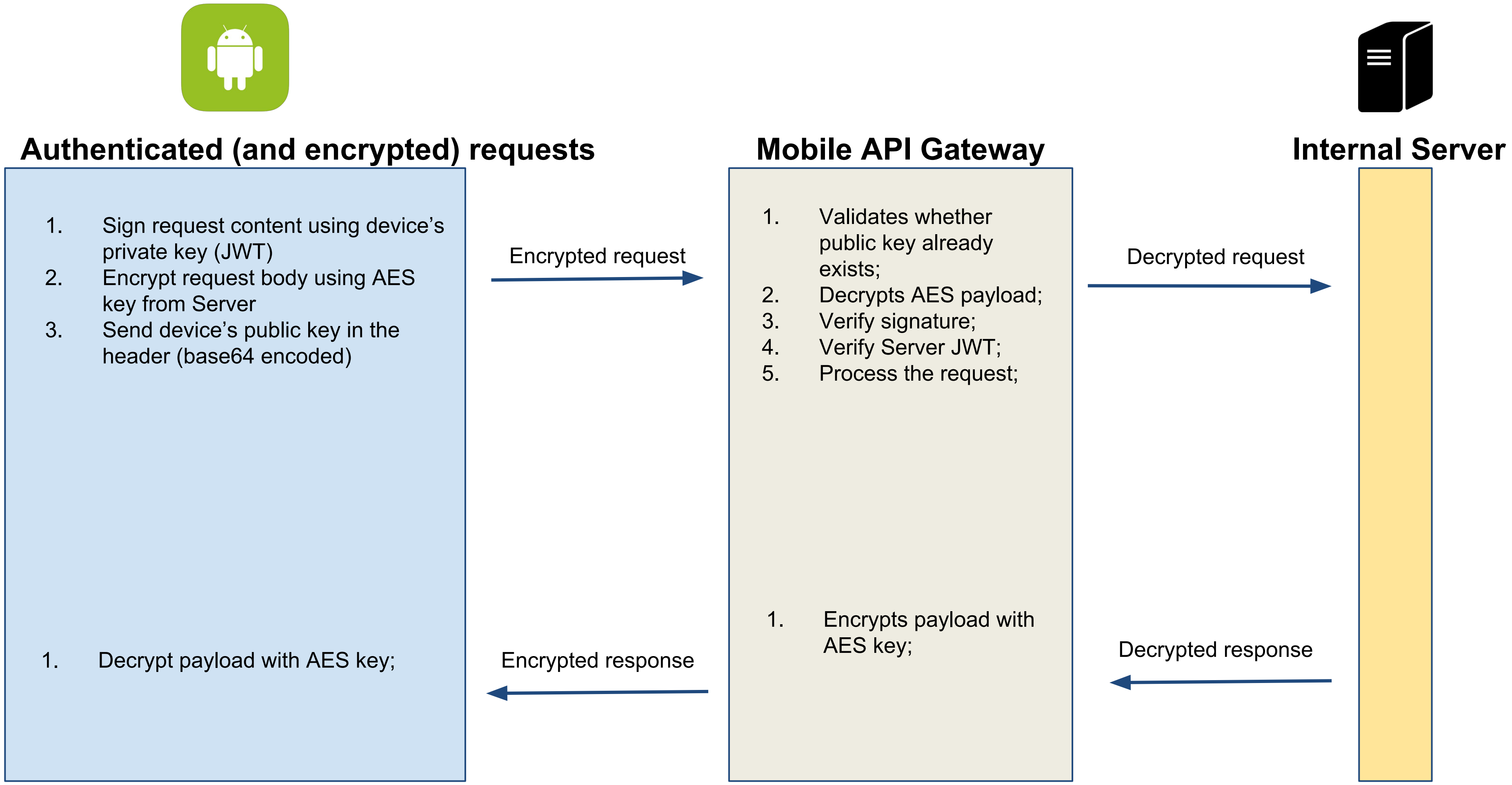
1. Decrypt AES key using Public key;
2. Save AES key in AndroidKeyStore;
3. Decrypt payload with AES key.

Mobile API Gateway

1. Validates the public key if it already exists;
2. Valid signature;
3. Process the request;

1. Generate AES key
2. Encrypts AES key with device’s public key;
3. Add JWT containing DNA and GUIDs to payload;
4. Encrypts payload using AES key.

Internal Server



1. Sign request content using device's private key (JWT)
2. Encrypt request body using AES key from Server
3. Send device's public key in the header (base64 encoded)

1. Validates whether public key already exists;
2. Decrypts AES payload;
3. Verify signature;
4. Verify Server JWT;
5. Process the request;

1. Encrypts payload with AES key;

1. Decrypt payload with AES key;

More security items for the app

- Obfuscation (Proguard / Dexguard);
  - Kerkhoff Principle;
- You can implement an administrative control (EULA) to warn Anti-Reversing;
- Check the execution environment (jailbreak / rooted device);
- Check if the app has not been tampered with (SafetyNet Attestation API);
- HTTPS (TLS 1.2+);
- SSL Pinning;
- Disable copy / print screen;
- Disable clipboard access;
- Remove app logs in production mode (use Timber);
- **Avoid saving sensitive data on android;**

Awareness / Education Challenges

- Cryptography is not teached properly to security professionals, let alone developers understand such topic (nothing new in here, but it’s really a problem);
- Topics like (a)symmetric cryptography, request signature, AEAD, IV, Key and PRNG usually take days to be understood;
- In addition, the difficulty of programmers to understand in practice the creation of encrypted streams.

Zooming in

Why generate a RSA pair?

- To identify the device;
- Public Key is sent when a user sign up;
- Digest (SHA256) of Public Key is sent along every request to serve as an ID;
- Private Key never leaves the device. It’s stored in Android KeyStore;
- Private Key is used to sign every request;
- Minimum key size to use: 2048 bits.

Was AES really necessary?

It’s a defense-in-depth mechanism given how many vulnerabilities showed up in OpenSSL. Even leaked documents from CIA states in their “DO & DON’TS” for Malware Developers to don’t trust OpenSSL either.

- After signing up a user from a mobile request, or every time a user signs in, the server generates an AES key;
- Server encrypts the AES key using mobile’s public key;
- The rest of the payload is encrypted using AES.

And AES what? AES GCM for Encryption & Authentication. Every ciphertext is tailored to a certain user and device.

Json Web Token (JWT) Role

We used 2 Json Web Tokens in this architecture:

- One from Android to Server, generated for every request:
  - Used to sign a request
  - Uses RS256 (using Private Key from Device)
- One from Server to Android, generated after a sign in only:
  - Serve as a Session ID
  - Uses HS256 (using Server's pepper)

Actually add a JWT library may not be necessary to generate a request signature, but it made the implementation easier. Otherwise we would have to develop our own signature generator.

Request Signing. Why?

- Authenticate Device
- Identify valid credentials from non trusted device
  - Starts a flow to register new device

Why use Amazon as a backend?

We relied on Amazon mainly for:

- Availability and
- How practical it is to set up the infrastructure.

There we used:

- EC2 (instances) and
- ECS to manage containers in such instances.

Every application is isolated in a Docker container.

Does Public Key Pinning worth the trouble?

Absolutely!

Few lines of code on the Android client, the public key used in our HTTPS certificate and success.

However we were using Amazon ELB, which would work great with Amazon Certificate Manager (generates and renew certificates automatically), but Amazon may change the public key, thus we could not rely on Amazon.

We used a combination of Custom Key Pair + Certbot to generate free certificates using our own key pair.

References

- Android Key Store https://developer.android.com/training/articles/keystore.html
- Json Web Token (JWT) https://jwt.io/
- Certbot https://certbot.eff.org/
- Android Encryption Limitations https://blog.cryptographyengineering.com/2016/11/24/android-n-encryption/
- ProGuard https://developer.android.com/studio/build/shrink-code.html
- Just 'Implement AES' is a very bad advice https://dadario.com.br/just-implement-aes-is-a-very-bad-advice/
- Generating Secure Random Numbers https://paragonie.com/blog/2016/05/how-generate-secure-random-numbers-in-various-programming-languages#nodejs-csprng
- Protecting against Security Threats with SafetyNet https://developer.android.com/training/safetynet/index.html
- https://speakerdeck.com/mseclab/android-key-management
- https://books.nowsecure.com/secure-mobile-development
- https://github.com/ashishb/android-security-awesome
- https://speakerdeck.com/rafaeltoledo/seguranca-no-android-1
- https://speakerdeck.com/manoelaranda/android-security-owasp-tips
- https://speakerdeck.com/yakivmospan/how-to-secure-data-in-android
- https://speakerdeck.com/mutexkid/keeping-android-secrets-secure-with-fingerprint-authentication-and-the-keystore

Thank you

Anderson Dadario

@andersonmvd
https://dadario.com.br

Márcio Rosa

@protonss4fun

Cândido Sales

@candidosales
https://candidosalesg.wordpress.com/