Cryptographic protection of ML models

@vixentael

COSSACK
LABS

@vixentael

Head of customer solutions, Security software engineer at Cossack Labs.

Focused on applied crypto, building e2ee protocols, and secure software development.

vixentael.dev

Things we won't talk about

COSSACK LABS

Adversarial networks, adversarial attacks

https://arxiv.org/abs/1712.09665

ML "Unlearning"

http://yinzhicao.org/unlearning/UnlearningOakland15.pdf

Malware inside ML networks

https://arxiv.org/abs/2107.08590

Deserialization bug in TensorFlow -> arbitrary code execution

https://portswigger.net/daily-swig/deserialization-bug-in-tensorflow-machine-learning-framework-allowed-arbitrary-code-execution

What we will talk about

COSSACK
LABS

Protecting IP

TensorFlow (a bit)

Application-level encryption

HPKE-like scheme, DRM-like approach

Integrating cryptography with traditional security controls

cossacklabs.com/case-studies/ai-ml-ip-protection/

@vixentael

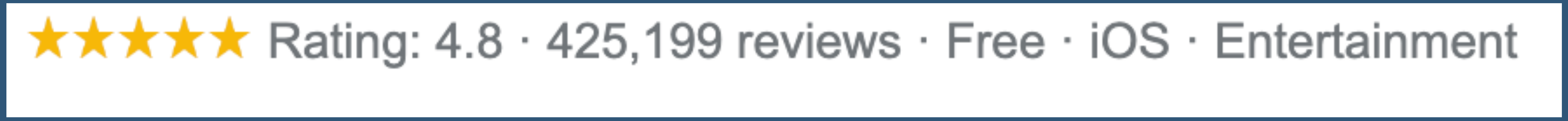
Let’s start

Protecting unique IP (ML models) against leakage and misuse

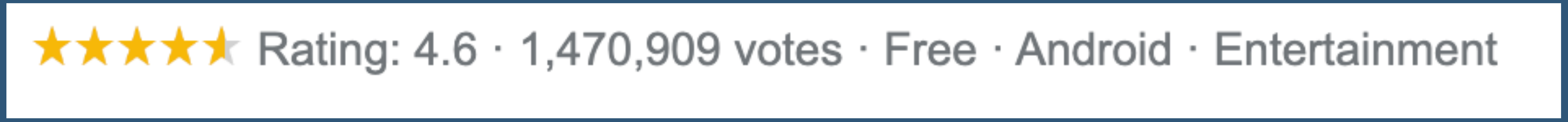
COSSACK LABS

Extremely popular AI/ML application.
ML models everywhere.

They wanted to switch ML execution from backend-side to client-side to decrease load and improve service.



★★★★★ Rating: 4.8 · 425,199 reviews · Free · iOS · Entertainment



★★★★½ Rating: 4.6 · 1,470,909 votes · Free · Android · Entertainment

IP -> backend

Before

Client app sends request.
Backend executes ML model.
Backend sends ready result.
Repeat every time.

IP -> backend, client-side

COSSACK LABS

Before

Client app sends request.
Backend executes ML model.
Backend sends ready result.
Repeat every time.

After

Client app sends request.
Backend generates unique Individual ML model (IML).
Backend sends IML to client app.
Client app stores and executes IML locally.
IMLs are unique per user (1 app - 1 user - N models).

@vixentael

Business risks of decision

R1	leakage of IP	loss of IP, competitor advantage, investments into updating ML model. *Losing 1 IML is not a problem, losing many IML is.*
R2	broken apps, clones apps, API fraud	abuse of infrastructure, revenue loss, abuse of IP, competitor advantage, reputation risks

Tech stack

Native mobile apps:

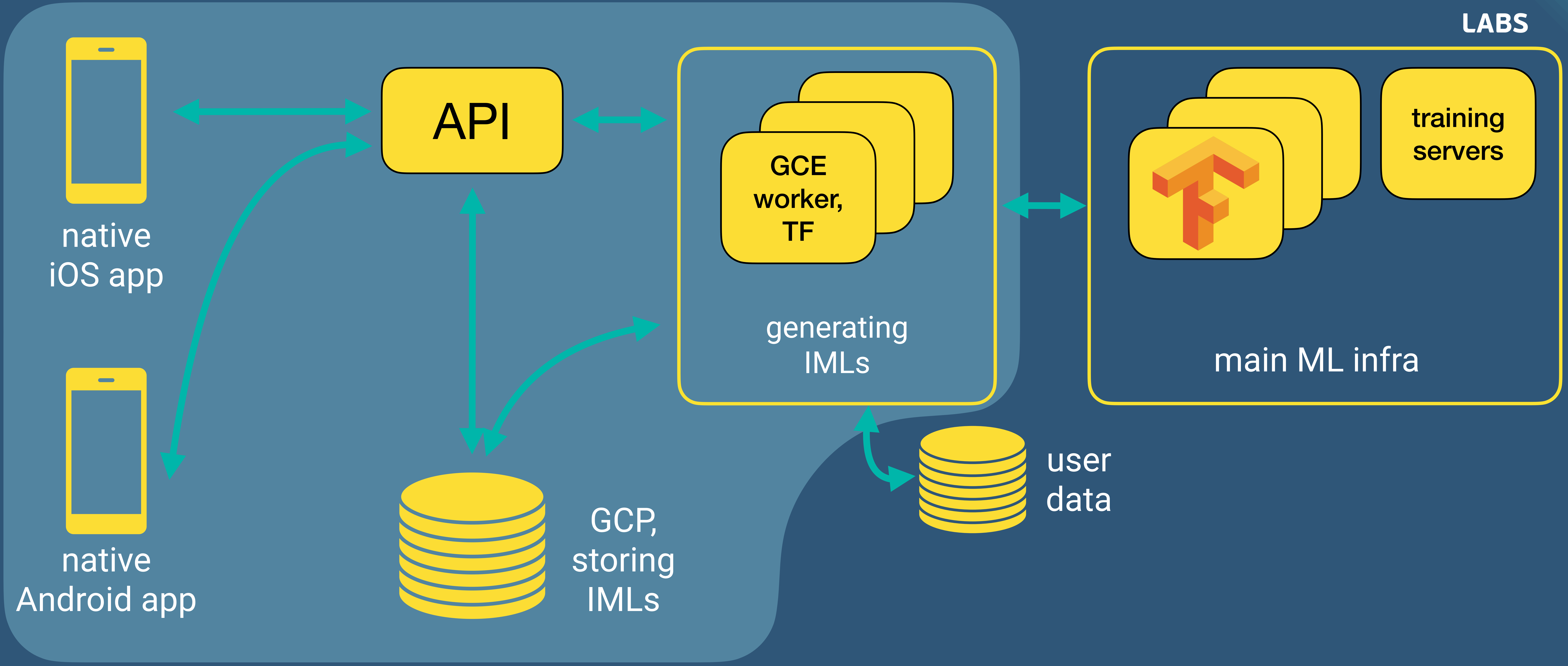
iOS – Swift/ObjC + CoreML
Android – Kotlin + TensorFlow Lite

python backend, TensorFlow

GCP: workers (GCE), storage,
KMS, DBs, Firebase authN

IML dataflow

COSSACK LABS



@vixentael

IML lifecycle

COSSACK LABS

GCE worker	generate IML, send to GCP storage	*memory, transit*
GCP storage	store IML file	*storage, transit*
API	URL on IML file	*transit*
mobile app	download from GCP storage, save locally as file, unpack & execute IML	*transit, storage, memory*

@vixentael

Threat modelling [simplified]

API leakage via API, credential leakage, abuse of IML generation pretending to be a paying user

Cloud storage collect from storage, find in backups, find in logs

Transit leakage / eavesdropping, client-server passive MitM, client-server active MitM

Mobile app extract IML via RE, crowdsourcing, automation of broken apps, malicious 3rd party libs

Threat modelling [simplified]

API

Cloud storage

Transit

Mobile app

Spoofing	Authenticity
Tampering	Integrity
Disclosure	Confidentiality
DoS	Availability

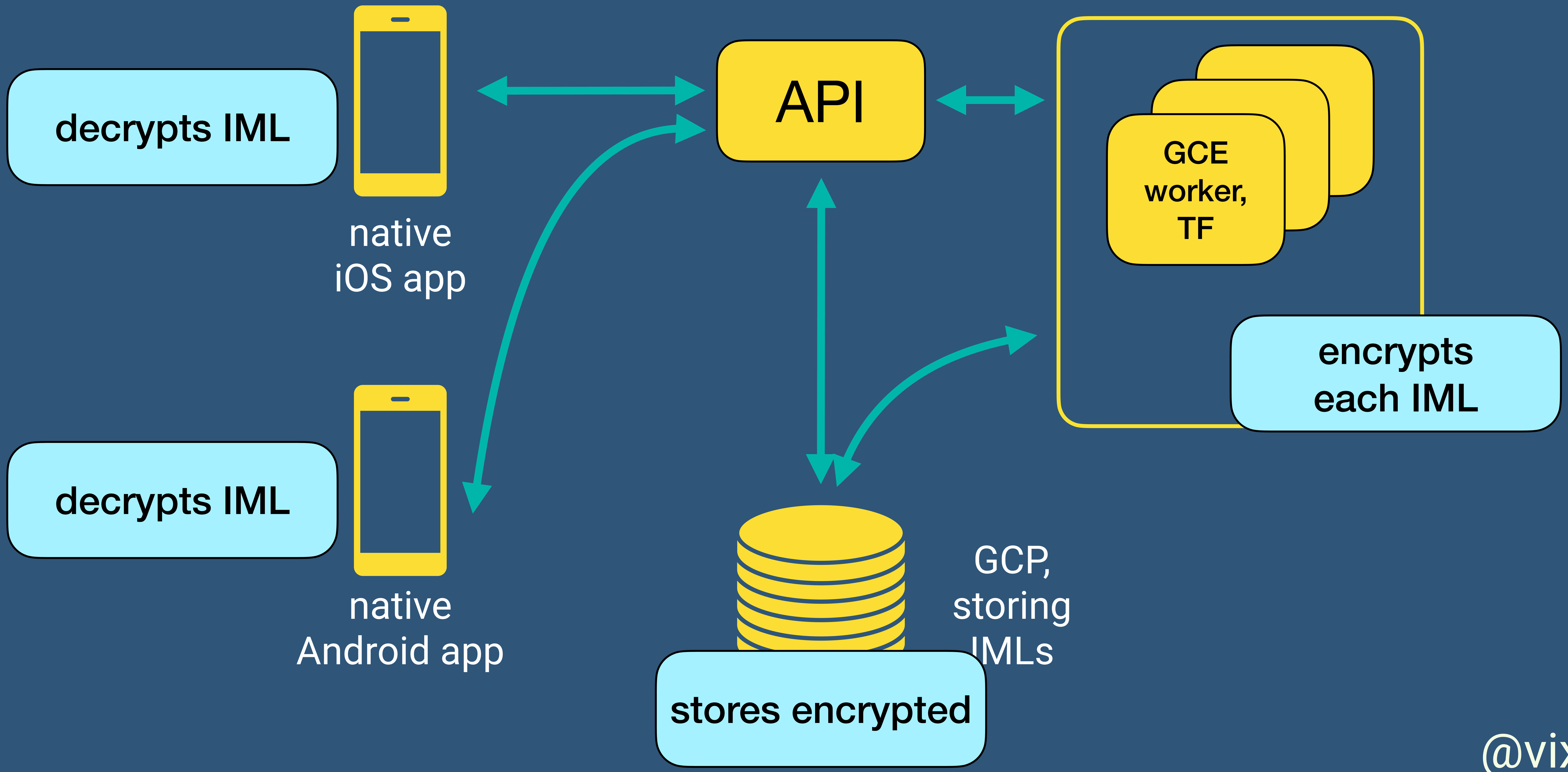
Let’s use cryptography!

What is ML model

ML model – output of ML algorithm. A file. With model data and procedure/algorithm. Layers with weights.

From security perspective – a file :)

Encryption layer



COSSACK LABS

@vixentael

Encryption layer: requirements

COSSACK LABS

1. Minimize the lifetime of plaintext IMLs
2. Minimize the chance of accumulating IMLs
3. Fast, smooth, without complicated crypto
4. Easy key management, without PKI
5. Works across 3+ platforms

Encryption layer: solutions

1. Minimize the lifetime of plaintext IMLs => encrypt after generation, decrypt before usage
2. Minimize the chance of accumulating IMLs => use unique keys per IML
3. Fast, smooth, without complicated crypto => AES-256-GCM + ECDH
4. Easy key management, without PKI => ephemeral keys
5. Works across 3+ platforms => Themis crypto lib

IML encryption & decryption

COSSACK LABS

GCE worker, TF

1. Generate keypair. Send app.publicKey to backend.

2. Generate keypair. Use server.privateKey and app.publicKey to derive sharedKey (ECDH).

3. Generate random DEK.

4. Encrypt IML using DEK, AES-256-GCM.

5. Encrypt DEK using sharedKey, AES-256-GCM.

6. Send { encryptedIML, encryptedDEK, server.publicKey }.

7. Receive. Use app.privateKey and server.publicKey to derive sharedKey.

8. Decrypt DEK, decrypt IML.

@vixentael

IML format

```
{
        "data": base64_str(encrypted_IML),
        "key": base64_str(encrypted_DEK),
        "public_key": server_ephemeral_public_key,
        "version": MODEL_VERSION,
        "layers": {
            // additional ML layers encryption
        }
}
```

IML encryption (backend side)

```
import pythemis

server_keypair = GenerateKeyPair(KEY_PAIR_TYPE.EC)
s_private_key = server_keypair.export_private_key()
s_public_key = server_keypair.export_public_key()

DEK = GenerateSymmetricKey()
cell = SCellSeal(DEK)
encrypted_IML = cell.encrypt(IML, userID)

secure_message = SMessage(s_private_key, app_public_key)
encrypted_DEK = secure_message.wrap(DEK)

send: { encrypted_IML, encrypted_DEK, s_public_key }
```

github.com/cossacklabs/themis

IML decryption (iOS side)

COSSACK LABS

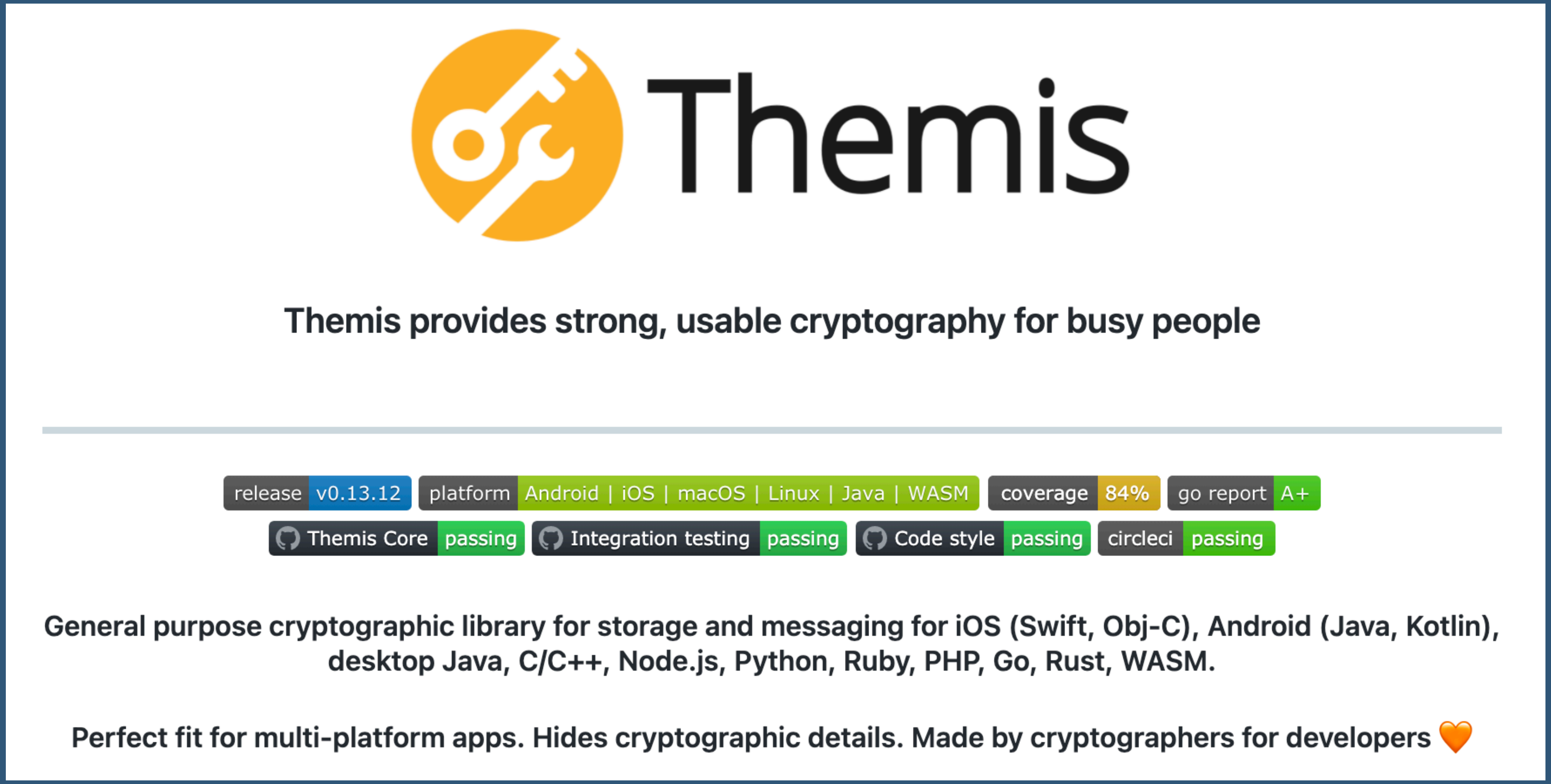
```
import themis

let keypair = TSKeyGen(algorithm: .EC)!
let appPrivateKey = keypair.privateKey!
let appPublicKey = keypair.publicKey!

let secureMessage = TSMessage(inEncryptModeWithPrivateKey: appPrivateKey,
peerPublicKey: serverPublicKey)!
let DEK = try? secureMessage.unwrapData(encryptedDEK)

let cell = TSCellSeal(key: DEK)!
let IML = try? cell.decrypt(encryptedIML, userID)
```

github.com/cossacklabs/themis

@vixentael

Crypto engine: Themis

COSSACK
LABS



Themis

Themis provides strong, usable cryptography for busy people

release v0.13.12 | platform Android | iOS | macOS | Linux | Java | WASM | coverage 84% | go report A+

Themis Core passing | Integration testing passing | Code style passing | circleci passing

General purpose cryptographic library for storage and messaging for iOS (Swift, Obj-C), Android (Java, Kotlin), desktop Java, C/C++, Node.js, Python, Ruby, PHP, Go, Rust, WASM.

Perfect fit for multi-platform apps. Hides cryptographic details. Made by cryptographers for developers 🧡

boring crypto

same API across 14 platforms

hidden crypto-details

recommended by OWASP

tons of docs

github.com/cossacklabs/themis

@vixentael

Application-level encryption

COSSACK LABS

Encryption process happening within application context, triggered by application.

ALE could work together with data-at-rest encryption and data-in-transit encryption.

ALE could be client-side, server-side, end-to-end, etc.

infoq.com/articles/ale-software-architects/

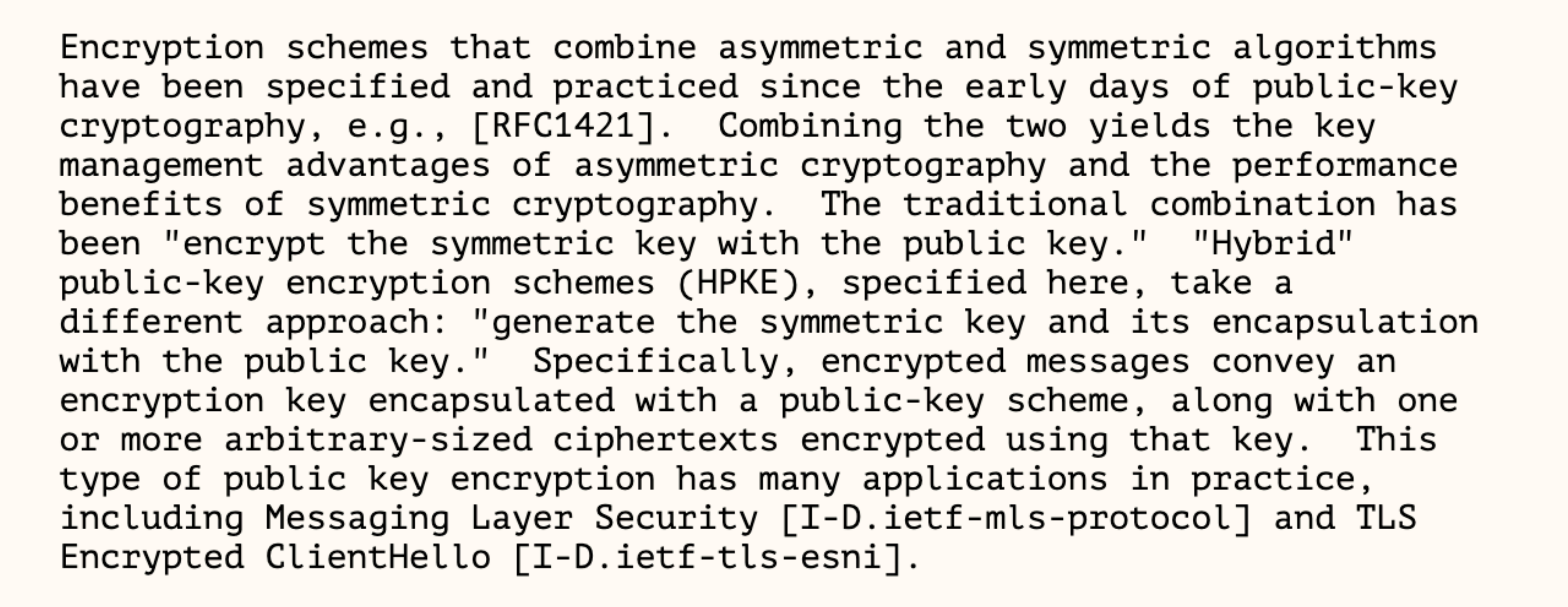
encryption controls / events	transit (TLS)	disk / FS	TDE / DB encryption	ALE	E2EE
physical access to servers	⛔	✅	✅	✅	✅
MitM	✅	⛔	⛔	✅	✅
privileged DB access	⛔	⛔	⛔	✅	✅
privileged system access	⛔	⛔	⛔	Depends	✅
backups, logs, snapshots	⛔	⛔	Few	✅	✅

infoq.com/articles/ale-software-architects/

@vixentael

Hybrid Public Key Encryption (HPKE)

encrypt data with symmetric key using AEAD;
encapsulate symmetric key with public key scheme



Encryption schemes that combine asymmetric and symmetric algorithms have been specified and practiced since the early days of public-key cryptography, e.g., [RFC1421]. Combining the two yields the key management advantages of asymmetric cryptography and the performance benefits of symmetric cryptography. The traditional combination has been "encrypt the symmetric key with the public key." "Hybrid" public-key encryption schemes (HPKE), specified here, take a different approach: "generate the symmetric key and its encapsulation with the public key." Specifically, encrypted messages convey an encryption key encapsulated with a public-key scheme, along with one or more arbitrary-sized ciphertexts encrypted using that key. This type of public key encryption has many applications in practice, including Messaging Layer Security [I-D.ietf-mls-protocol] and TLS Encrypted ClientHello [I-D.ietf-tls-esni].

RFC describes approach used before and implies standardization.

datatracker.ietf.org/doc/draft-irtf-cfrg-hpke/

Lightweight key management

COSSACK LABS

1. Lightweight key management – server generates ephemeral keypair each time, no need for PKI.

2. NIST SP 800-57 – sorry, ephemeral keys FTW.

3. Store client-side public key in the user database to “pin” devices, or use ephemeral keypairs too.

4. Server authenticity problem – solve by server attestation, TLS pinning.

5. Mobile app storage problem – use Keychain/KeyStore, encrypt keys by SecureEnclave.

@vixentael

Crypto defense in depth:

1. Re-encrypt IML on device on receiving (AES-256-GCM).

```
let new_DEK = TSGenerateSymmetricKey()
let cell = TSCellSeal(key: new_DEK)!
let encrypted_IML_ID = try? cell.encrypt(IML)
```

Crypto defense in depth:

COSSACK LABS

1. Re-encrypt IML on device on receiving (AES-256-GCM).

```
let new_DEK = TSGenerateSymmetricKey()
let cell = TSCellSeal(key: new_DEK)!
let encrypted_IML_ID = try? cell.encrypt(IML)
```

=> to un-link server keys, to re-encrypt IML purely based on device keys

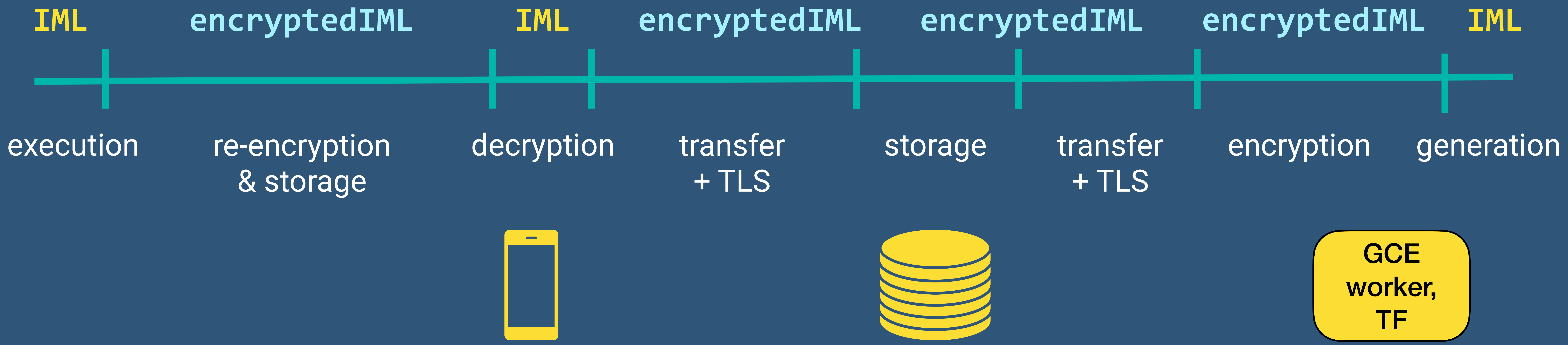
Store re-encryption keys in Keychain/KeyStore.

Bonus points for biometrics binding.

Crypto defense in depth:

COSSACK LABS

2. IMLs are encrypted after generation for storage, then using TLS for transport, then re-encrypted on device.



IML · encryptedIML · IML · encryptedIML · encryptedIML · encryptedIML · IML

execution · re-encryption & storage · decryption · transfer + TLS · storage · transfer + TLS · encryption · generation

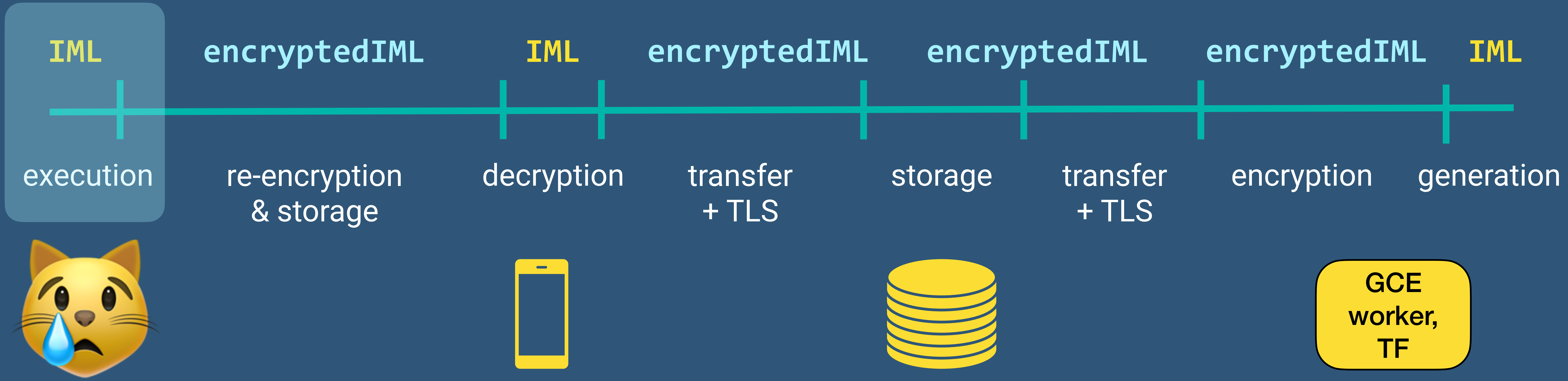
GCE worker, TF

@vixentael

Crypto defense in depth:

COSSACK
LABS

2. IMLs are encrypted after generation for storage, then using TLS for transport, then re-encrypted on device.



IML

encryptedIML

IML

encryptedIML

encryptedIML

encryptedIML

IML

execution

re-encryption
& storage

decryption

transfer
+ TLS

storage

transfer
+ TLS

encryption

generation

GCE
worker,
TF

@vixentael

Crypto defense in depth:

COSSACK LABS

3. In-memory encryption.

CoreML requires plaintext model file when loads.

@vixentael

Crypto defense in depth:

COSSACK LABS

3. In-memory encryption.

CoreML requires plaintext model file when loads.

=> create MLCustomLayer with encrypted weights, decrypt before load to shader (CPU)

=> create custom shader function to obfuscate weights before execution on shader (GPU)

(also see Apple docs on encrypting ML models that are parts of app bundle)

@vixentael

Performance considerations

COSSACK LABS

1. GPU shaders have limited cache memory, can’t run “normal” ciphers.
2. Use fast crypto: ECC & AES-GCM.
3. Crypto adds performance penalty, but AES-GCM has hardware support everywhere. No noticeable UX penalty.
4. Some Android devices are extremely slow, but if the device can render ML with 50-60 FPS, it can run crypto fast.
5. Generating IMLs and encrypting them might be still faster than executing server-side ML for each request.

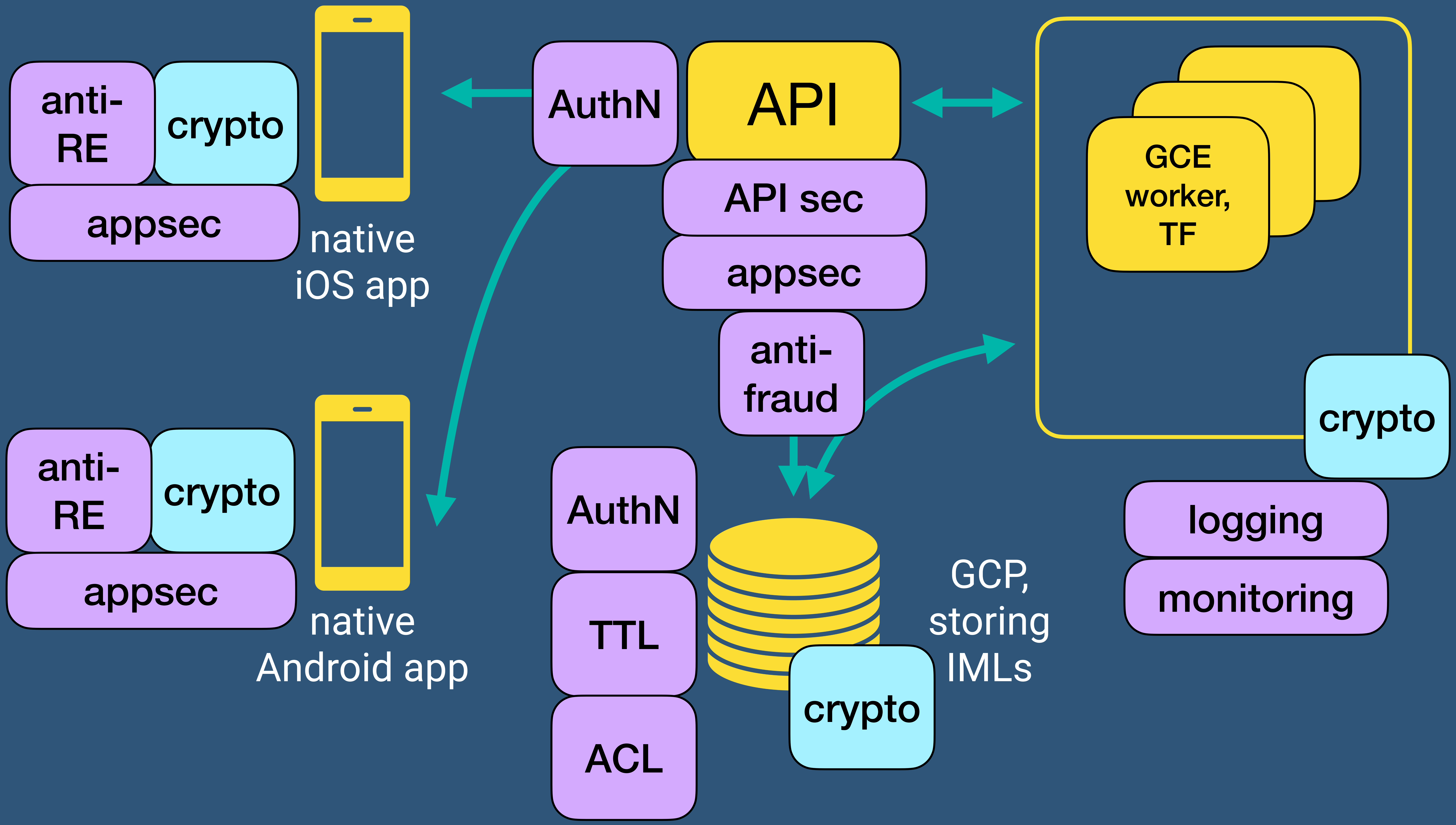
Overlapped security controls

COSSACK LABS

✅
1. Encryption to protect IMLs globally during the whole dataflow.
2. Whatever is the attack vector, there is a defense layer.
3. For most popular attack vectors, we want as many independent defenses as possible.

Crypto is more useful when integrated with traditional security controls.

Integration with other security controls



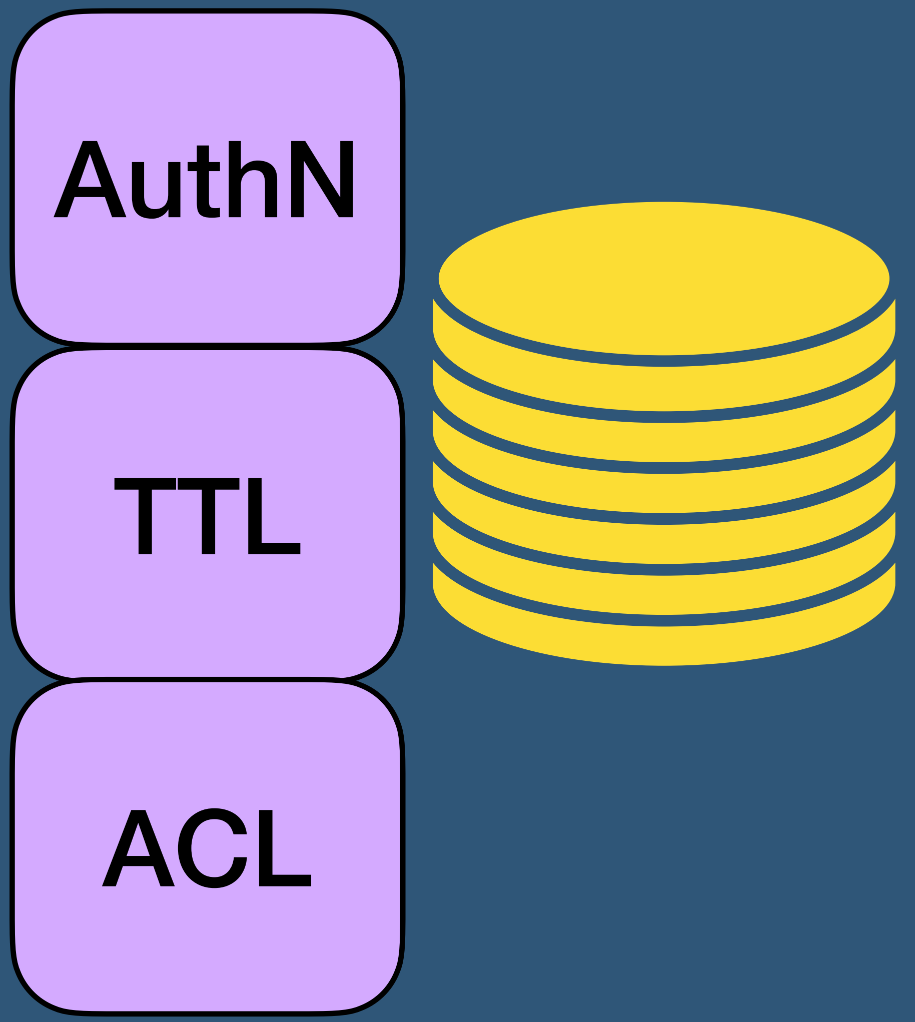
COSSACK LABS

@vixentael

Cloud storage security 101

COSSACK LABS

1. IMLs are stored min time – apps are expected to grab their IML quickly.
2. URL TTL (expire after mins).
3. URL authentication & access control.
4. Clean up IML files (every hour).
5. Do not backup IMLs.
6. URLs are not logged.
7. Monitoring of access errors.

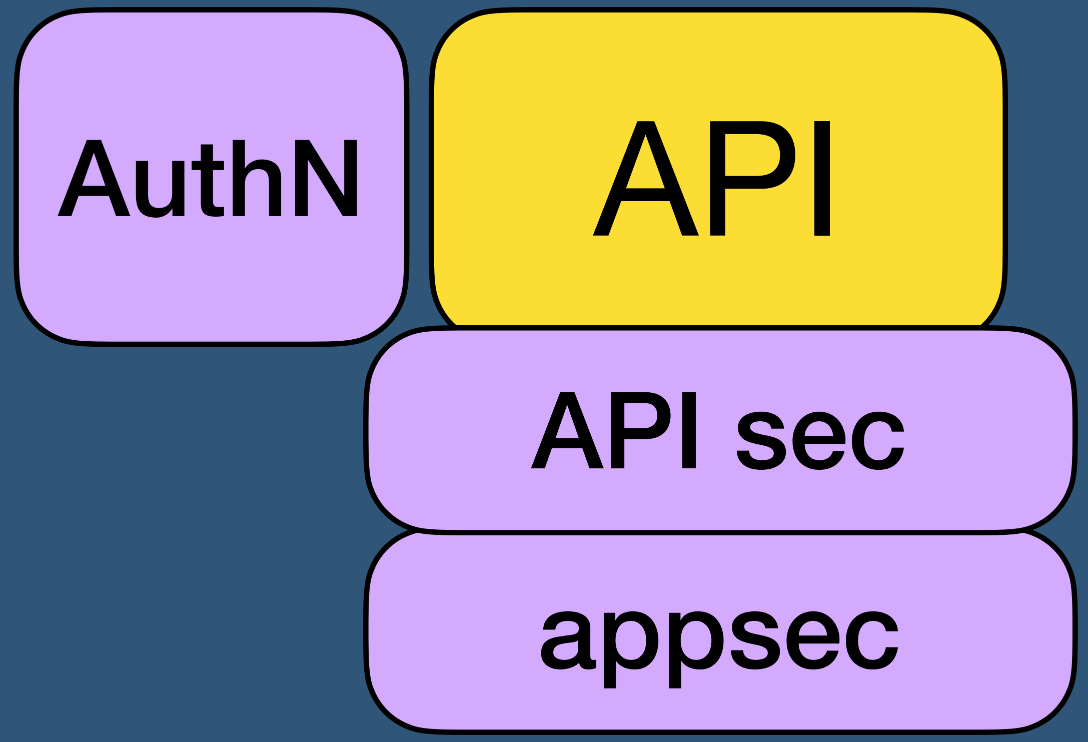


(also see OWASP WSTG-CONF-11)

@vixentael

API protection 101

1. User authN, IMLs are available only after successful authN.
2. API limits, requests throttling, firewalling.
3. IML request limits – after N model requests, server returns error.

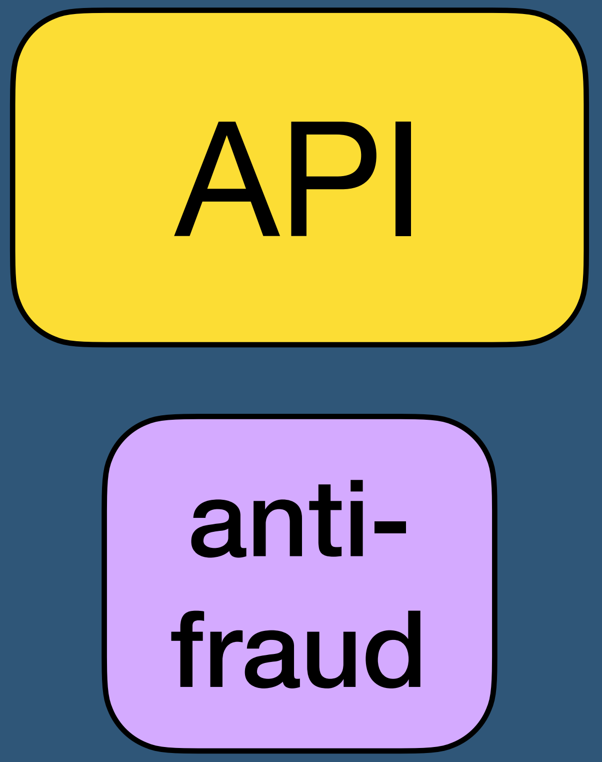


(also see OWASP ASVS :))

@vixentael

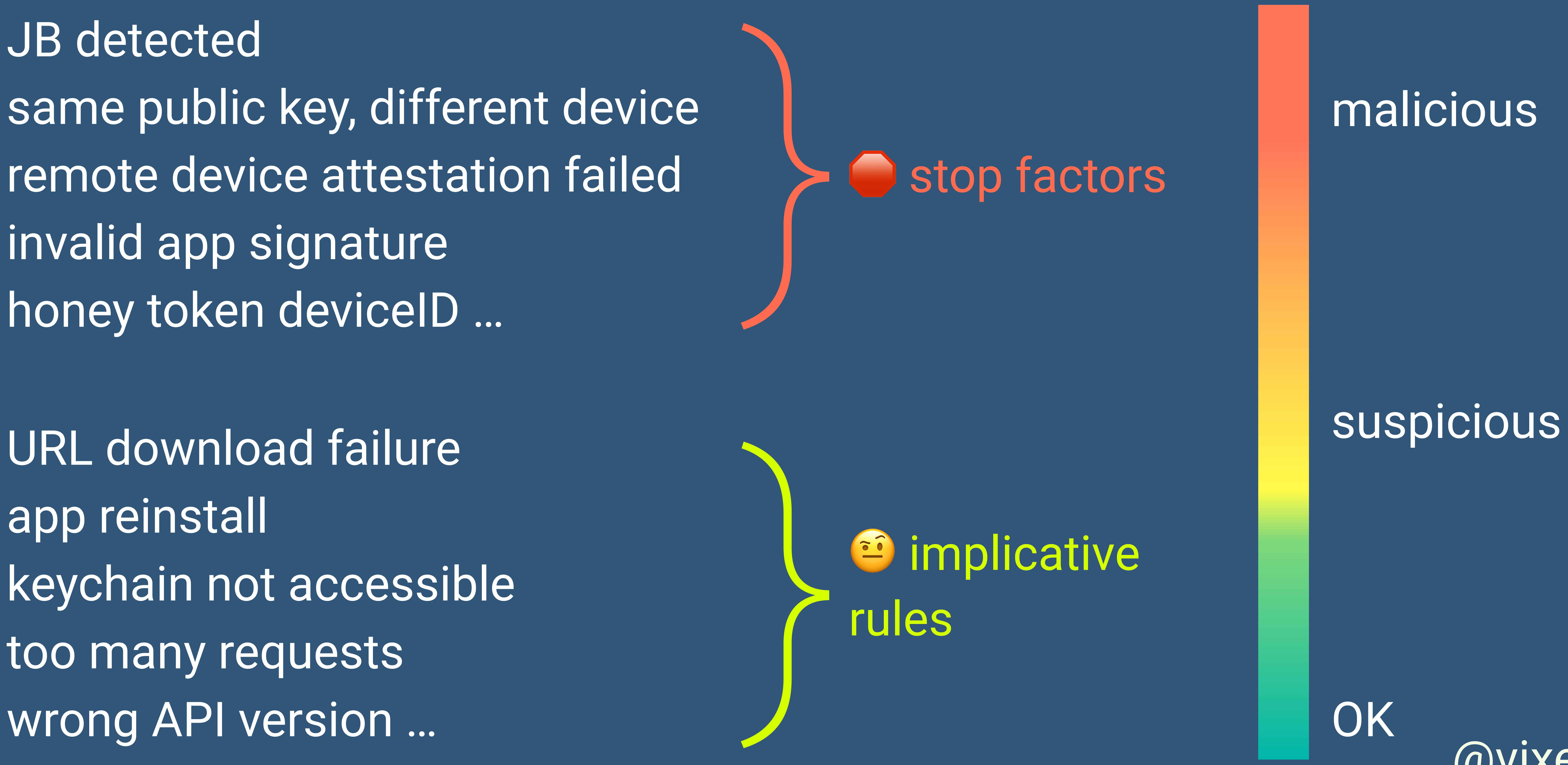
Anti-fraud system 201

1. Limit access to IML based on user behaviour.
2. Gather events from mobile apps and from server side.
3. Calculate user scoring based on events (“stop-factors”, rules).
4. User scoring: OK, suspicious, malicious.
5. Block malicious, limit suspicious.



COSSACK LABS

Anti-fraud system 201



JB detected
same public key, different device
remote device attestation failed
invalid app signature
honey token deviceID …

🛑 stop factors

URL download failure
app reinstall
keychain not accessible
too many requests
wrong API version …

🤨 implicative rules

malicious

suspicious

OK

@vixentael

Remote device attestation

COSSACK LABS

1. Use as part of user authN.
2. Use as source for anti-fraud system.
3. Block apps installed not from stores.

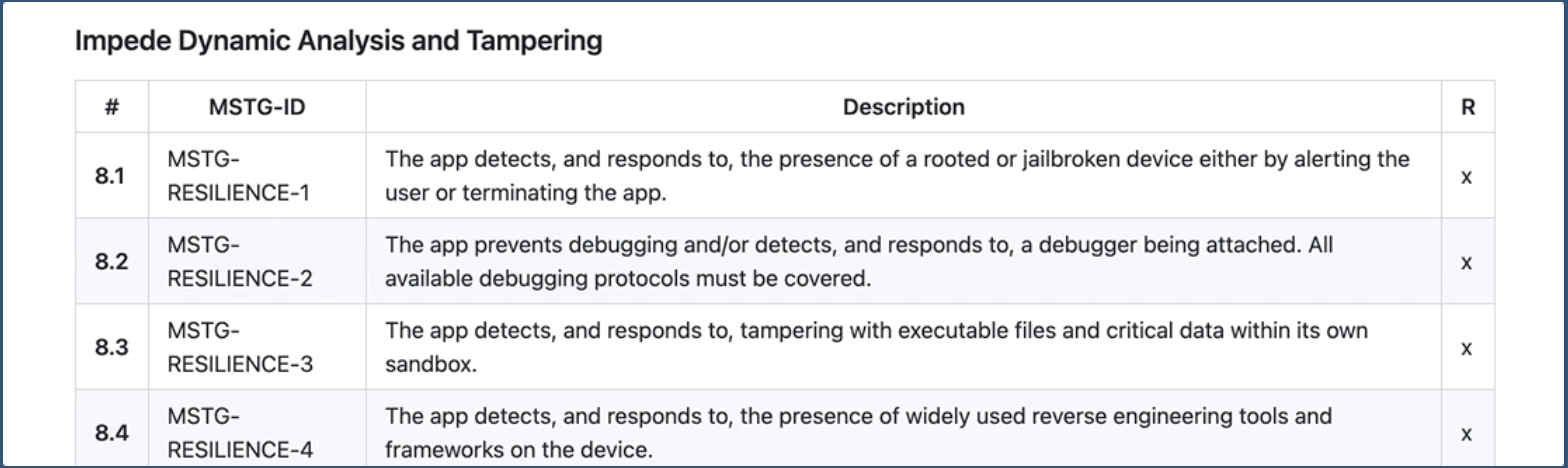
Apple DeviceCheck

developer.apple.com/documentation/devicecheck

Android SafetyNet

developer.android.com/training/safetynet/attestation

Anti-reverse engineering mobile apps



Impede Dynamic Analysis and Tampering

#	MSTG-ID	Description	R
8.1	MSTG-RESILIENCE-1	The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.	x
8.2	MSTG-RESILIENCE-2	The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.	x
8.3	MSTG-RESILIENCE-3	The app detects, and responds to, tampering with executable files and critical data within its own sandbox.	x
8.4	MSTG-RESILIENCE-4	The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.	x

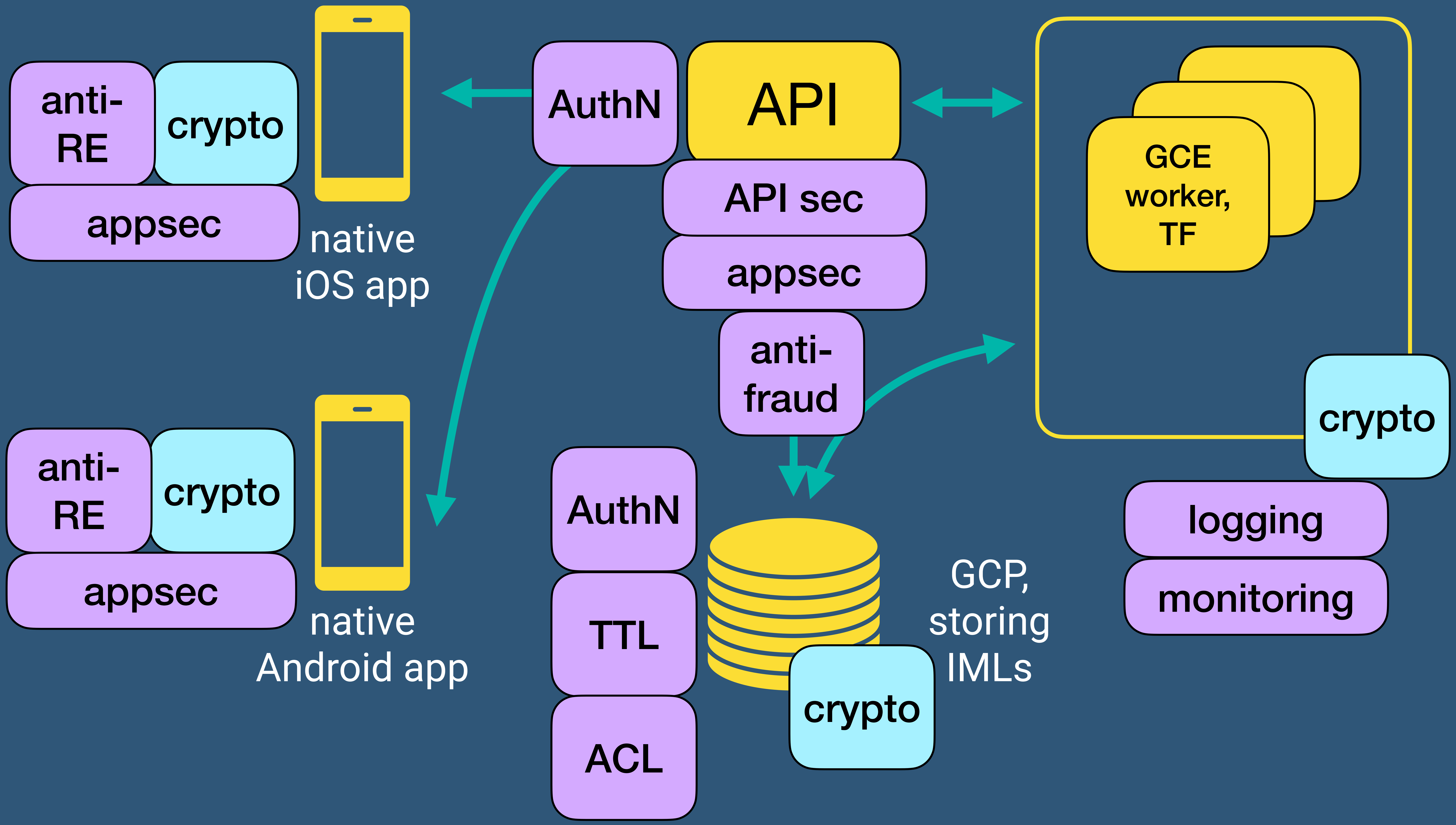
(also see OWASP MASVS-R)

Special improvements for ML models

COSSACK LABS

1. Watermarks.
2. Custom ML layers.
3. Model binding (ML models that work only with custom data -> non-general purpose ML models, no risks to steal).

Integration with other security controls



COSSACK LABS

@vixentael

Overlapped security controls

COSSACK LABS

✅ 1. Encryption to protect IMLs globally during the whole dataflow.

✅ 2. Whatever is the attack vector, there is a defense layer.

✅ 3. For most popular attack vectors, we want as many independent defenses as possible.

Failure of a single security control is a question of time.

Failure of a security system is a question of design.

COSSACK LABS

Use cryptography; don’t learn it

vixentael.dev/talks/use-crypto-dont-learn-it/

Application Level Encryption for Software Architects, by @9gunpi

infoq.com/articles/ale-software-architects/

Cryptographically signed audit logs

cossacklabs.com/blog/crypto-signed-audit-logs.html

React Native security: things to keep in mind, by @julepka

cossacklabs.com/blog/react-native-app-security.html

COSSACK LABS

@vixentael

vixentael.dev

cossacklabs.com

cossacklabs.com/whitepapers

cossacklabs.com/blog