FIGI Security Clinic

FIGI FINANCIAL INCLUSION GLOBAL INITIATIVE

Application Security Framework for DFS

Kevin Butler

4-5 December 2019
#financialinclusion

Sponsored by

BILL & MELINDA GATES foundation

Organized by

Committee on Payments and Market Infrastructures
BANK FOR INTERNATIONAL SETTLEMENTS

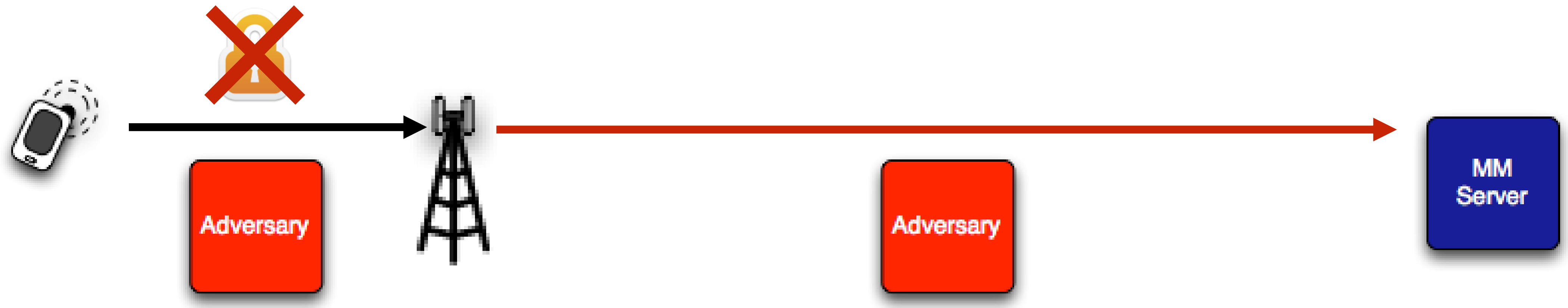
WORLD BANK GROUP

ITU

Why Have an Application Security Framework for DFS?

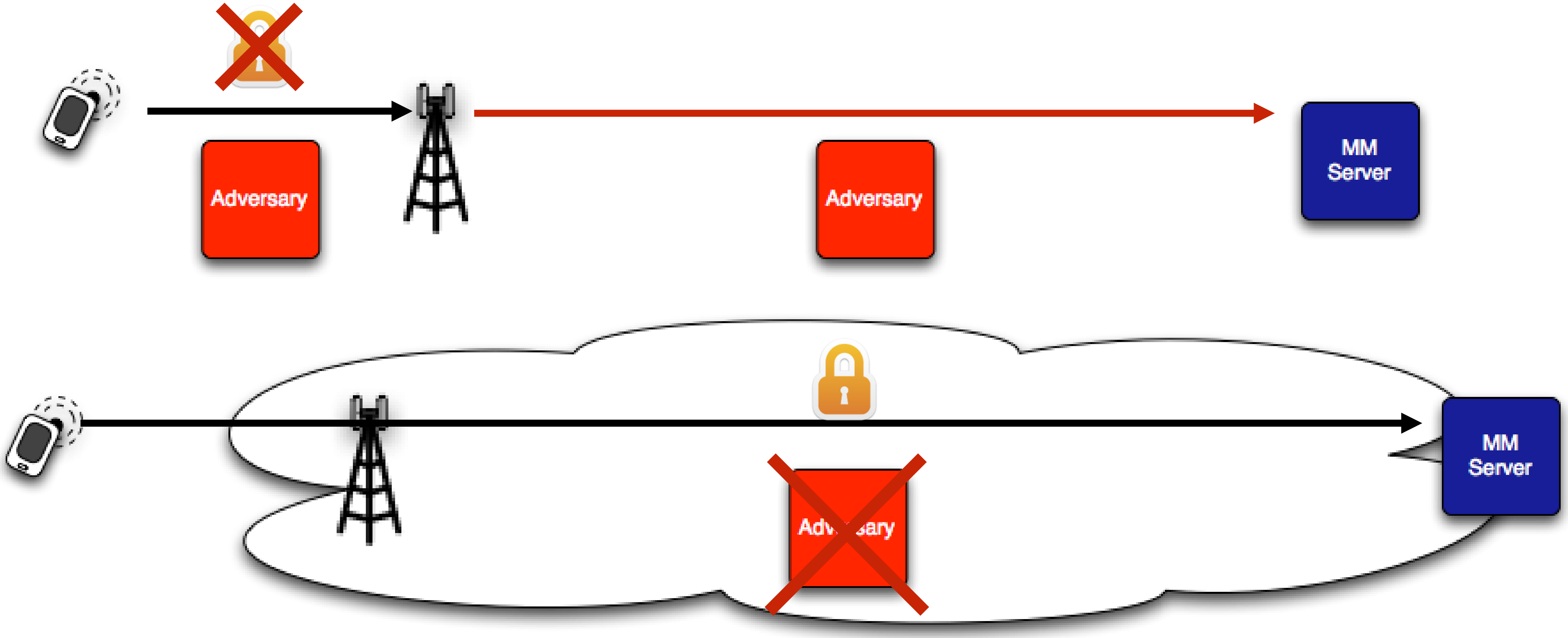
- Smartphones are a tremendous driver of innovation and adoption
  - Much of the most compelling new developments in computing has been in the smartphone area over the past 10 years
- Smartphones have only recently become a conduit for conducting financial activity
  - But growth has been explosive
- Mobile applications have great potential for providing improved security
  - But the threat surface is larger and the consequences more dire

Security Advantages of Smartphones



- Featurephones on 2G networks face vulnerabilities in a number of areas throughout the MNO's network.

Security Advantages of Smartphones



- Smartphones support end-to-end security

Other Smartphone Security Features

- Secure hardware
  - Secure boot
  - Platform measurement
  - Trusted execution envrionments

- Strong authentication
  - Biometric security

Template for Application Security Best Practices

- General best practices for a mobile money smartphone application security framework
- Draws upon GSMA study on mobile money best practices, ENISA smartphone security development guidelines, State Bank of Pakistan mobile payment applications security framework
- Template can be used as input to an app security policy by DFS providers
- Considerations: device and application integrity, communication security and certificate handling, user authentication, secure data handling, secure application development

Device and Application Integrity

- Assure the integrity of mobile devices prior to engaging in sensitive data transactions
  - The safest devices are those that have not been “jailbroken” or “rooted” since it may not be possible to assess their security
  - Circumvents platform integrity routines such as secure boot
  - Applications should use mobile platform services to determine that they and the underlying platform have not been modified
- Remove extraneous code that may have been added to the application during testing or features not designed for the target platform prior to deploying app in production
- Assure the integrity of the app on the server side

Communication Security

- Ensure that apps are using standardized encryption libraries
- Current best practice is minimum TLS 1.2
  - Why TLS? Widely-deployed, protocol openly available, not patent-encumbered, free libraries in every OS
- Older libraries have issues
  - Susceptibility to attacks (Heartbleed, DROWN, POODLE, BEAST, etc)
- TLS v1.3 is becoming available
  - Better performance and security but will likely take a few years before we see very wide adoption

TLS Ciphersuites

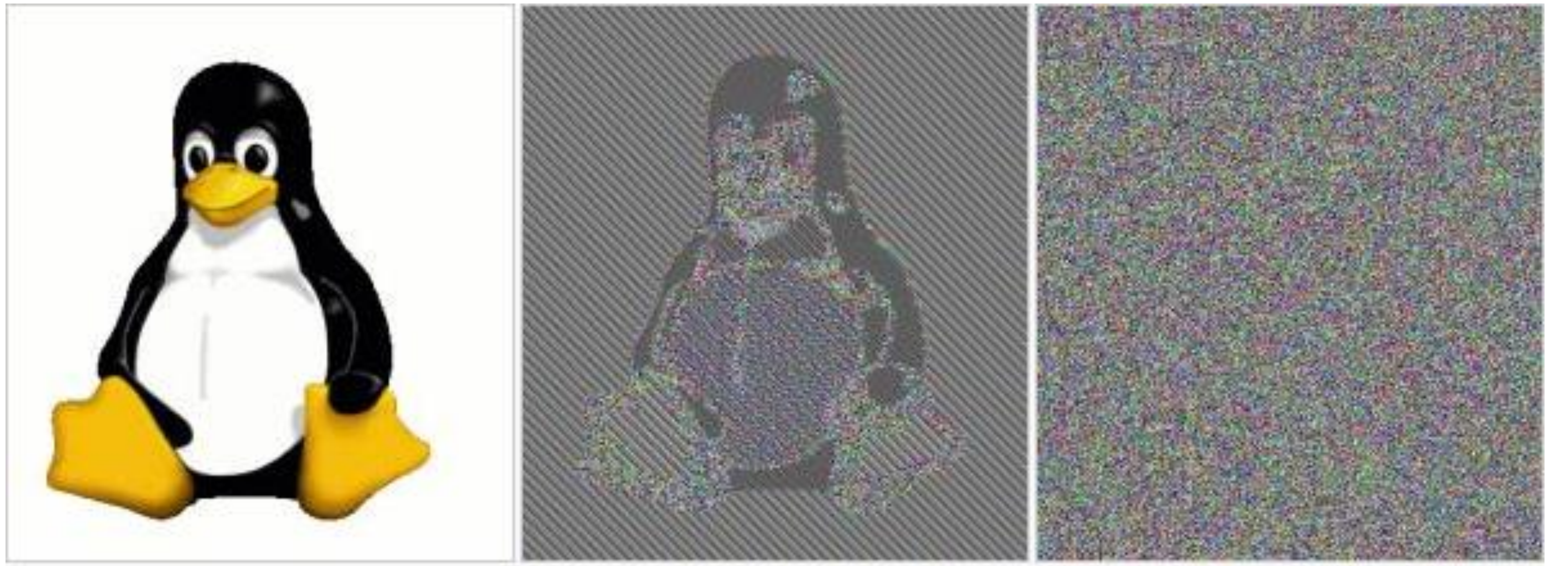
- There are many different algorithms used for authentication, encryption, and integrity
- Not all of them are considered secure for a modern deployment – especially if the server supports TLS < 1.2
- Modes of operation: the way that you encrypt information is important to consider
    - Encryption ciphers are generally “block ciphers” that break data into chunks (blocks)
    - The way that these blocks are linked together can be good or bad for security
- One recommended mode: `TLS_DHE_RSA_WITH_AES_128_CBC_SHA`

Example: Electronic Code Book

- ECB is fast and parallelizable; blocks are encrypted independently of each other
- Why is this bad?

Example: Electronic Code Book

- ECB is fast and parallelizable; blocks are encrypted independently of each other
- Why is this bad?



Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

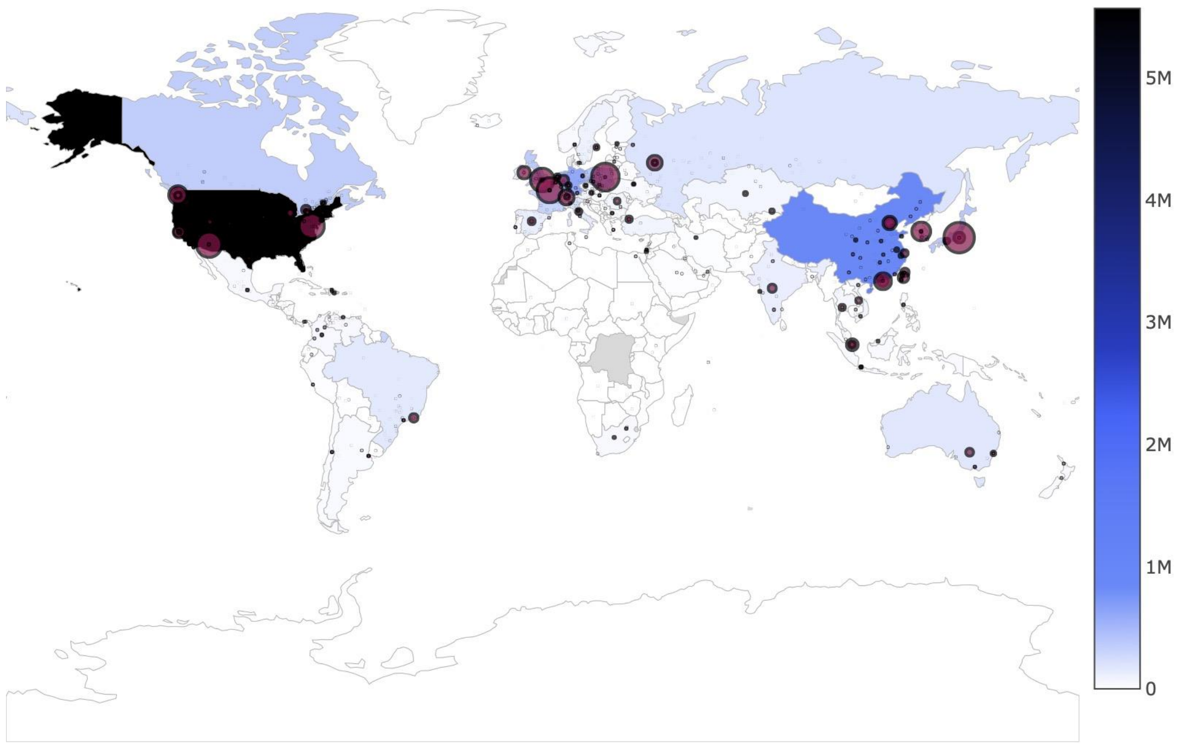
Authenticated Encryption

- Encryption assures **confidentiality** but does not assure **integrity**
- Require message authentication codes (MACs) and corresponding hashing algorithm for integrity
  - Some hashing ciphers are good, e.g., SHA-256
  - Some hashing ciphers are not, e.g., MD5, SHA-1 (no longer recommended)
- New modes of encryption in TLS 1.2 and 1.3 provide *authenticated encryption*
  - These provide both confidentiality and integrity without needing to explicitly add MACs
  - GCM can provide this - but not much client adoption yet

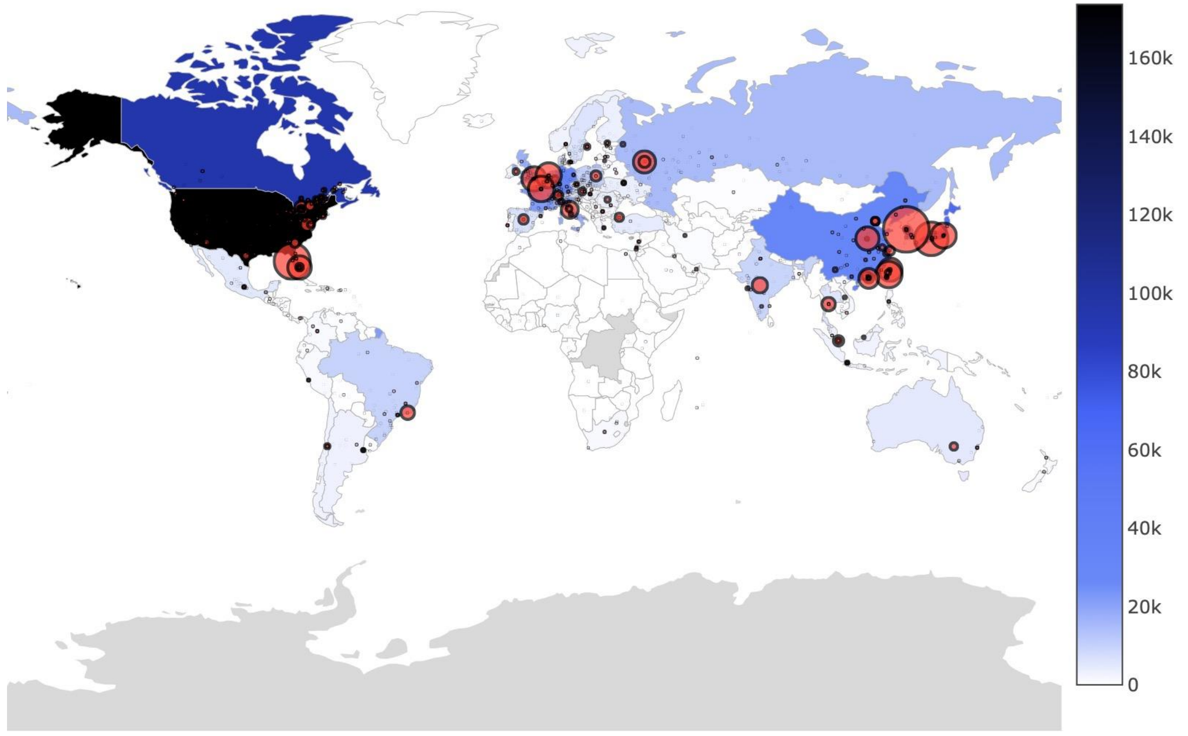
Encryption Ciphers

- Important to assure that the ciphers used for encryption are themselves secure
- Strike balance between performance and security
  - AES-128 is fine for now, AES-256 might be overkill
  - But that might not be the case a few years down the road
  - Need to keep informed about the strength of crypto ciphers
- 3DES has been officially deprecated by US NIST
- Is it still being used?

DES Usage (Frost et al., 2019)



3DES



DES-56

- Over 40% of queried TLS servers supported some form of DES
- Even DES-40 was found on some servers!

Key Negotiation

- The way in which session keys are negotiated can also be insecure
- Many modes of Diffie-Hellman negotiation can be used
- However, ephemeral DH is more secure than fixed, and especially anonymous DH
  - Characterised as `TLS_DHE_` in the set of available negotiation algorithms
- Anonymous DH (`TLS_DH_anon`) are susceptible to MITM attacks
  - DH always needs to be authenticated!

TLS on Clients

- All versions of Android beginning with API level 16+ have support for TLS 1.2
  - Corresponds to Android 4.1 Jelly Bean, released July 2012
- TLS 1.2 enabled by default for API level 20+
  - Corresponds to Android 5.0 Lollipop, released November 2014
  - SHA-256 only supported as API option for API level 20+
- Optimally at least this level of Android should be the supported to ensure TLS 1.2 is employed by clients

Certificate Handling: Server

- Offer certificates that have been signed by CAs, rather than self-signed certificates
  - Have a contingency plan if the CA fails or is no longer trustworthy
- Do not offer expired TLS certificates for clients to verify
- Lifetime of issued certificates should be limited to 825 days
- Extended validation (EV) certificates are likely not necessary
  - Standard certificates are sufficient

Certificate Handling: Client

- Smartphone OSes properly check to see that new TLS connections include a valid X.509 certificate
  - But many applications override this check
  - Likely to silence errors during testing
- Bypassing certificate validation and server authentication should never be done in production code
- Trusted CAs should be stored securely by the client
  - E.g., in an Android `KeyStore` object for storing the set of trusted Cas and a `TrustManagerFactory` object to store them in
- Certificate pinning can prevent being presented a fake server name later
  - Use APIs such as Android's `CertificatePinner`

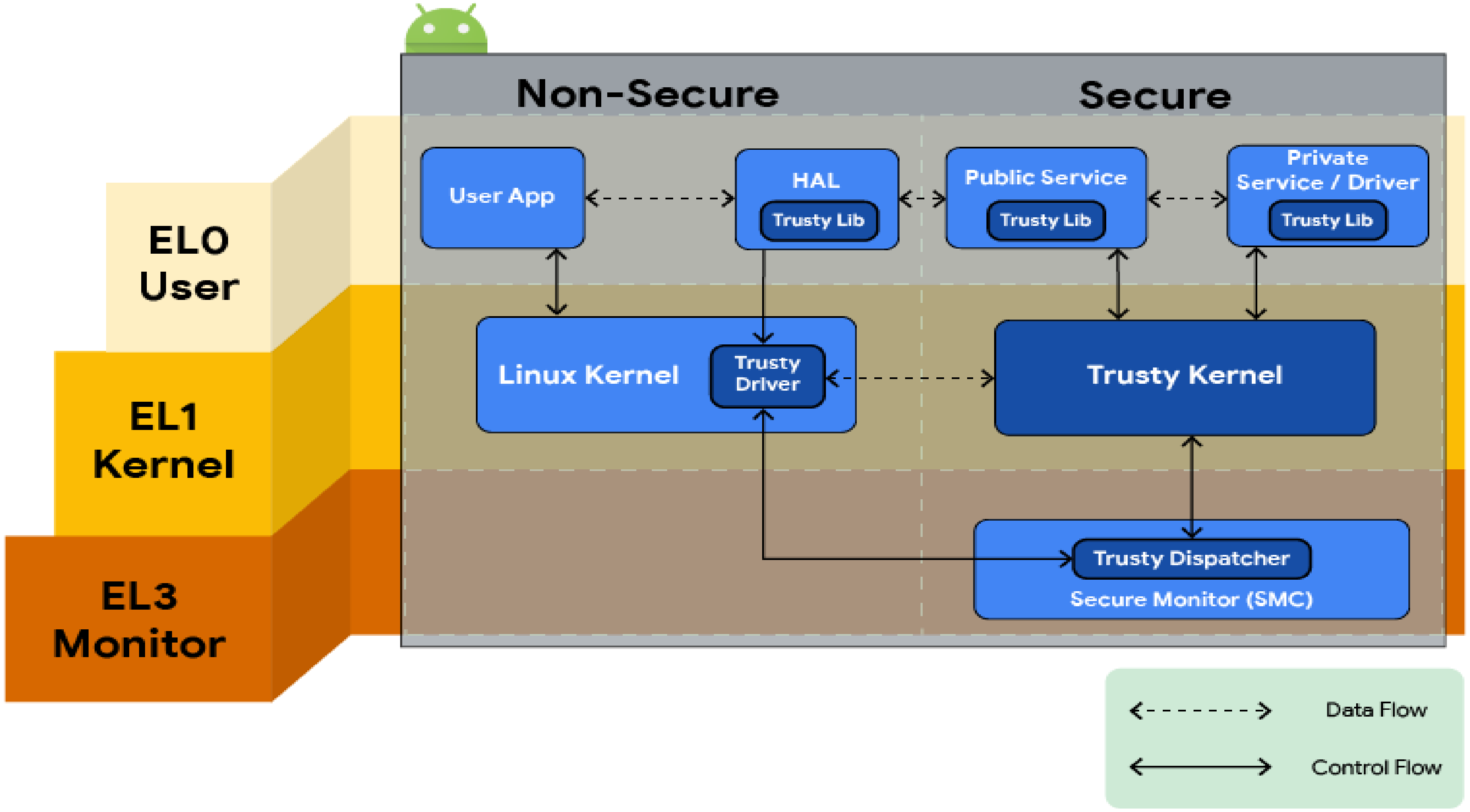
User Authentication

- PINs and passwords should not be easily guessable
  - Weak credentials should not be allowed
- However, ***do not*** force the user to constantly change their password!
- Multi-factor authentication is strongly encouraged before performing financial or other sensitive transactions
- Avoid the use of SMS as an authenticator
  - OTPs can be intercepted by SS7 hijacks
  - Use smartphone authenticators instead (e.g., Authy)
  - Fingerprint API in Android in `FingerprintManager` class

Secure Data Handling on Client

- Modern Android devices have many mechanisms to support more trustworthy data handling
- Android KeyStore can keep data either encrypted on main device or within *trusted execution environment (TEE)*
- API call `KeyInfo.isInsideSecureHardware()` available for Android 6.0 and greater
- Android 8.0 provider interface to retrieve ID attestation directly from secure hardware
  - `ATTESTATION_ID_SERIAL` and `ATTESTATION_ID_IMEI` retrieve device serial number and IMEI of all radios from TEE

Trusted Execution Environment



- AOSP provides Trusty OS in TEE, other manufacturers provide their own (Trustonic Kinibi, Qualcom QSEE most commonly seen)

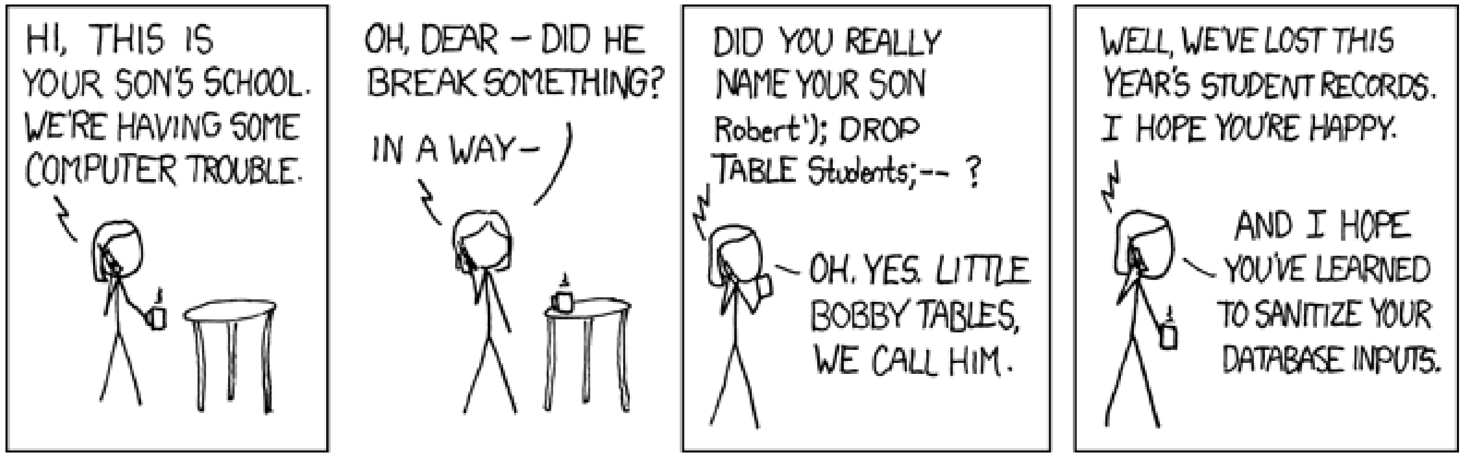
Secure Data Handling Recommendations

- Use the Android KeyStore framework
- Use TEEs when they are available on devices
- Delete confidential information from caches and memory and avoid general exposure of information
  - E.g., storing the secret key on the stack
- Use fine-grained permissions to restrict data sharing
- Do not hard-code sensitive information such as passwords or keys into the application source code

FIGI › FINANCIAL INCLUSION GLOBAL INITIATIVE

Input Validation

- Avoid storing data in external storage
  - It becomes available to all applications so no integrity should be assumed
  - Perform strong input validation if you do this
- Validate any input coming from the client to be stored in databases
  - Do this to avoid SQL injection attacks


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH, YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Secure Application Development

- Develop applications according to industry-accepted secure coding practices and standards
- Assure a means of securely updating applications and assure that all dependent libraries and modules are secure
  - Provide updates when required
- Have code independenty assessed and tested by internal and/or external code review teams

Summary

- Good practices revolve around basic security principles
- The template provides a starting point for ensuring principles of authentication, access, control, integrity, and confidentiality are maintained
- Privacy out of scope but also an important factor
- Technologies change so specific recommendations might also change but the long-term trends are clear
- Be vigilant!