
# Android Fingerprint

## Esame di Sistemi Biometrici

A cura di Gianluca Scatigna

[Matricola 718633]

ANNO ACCADEMICO 2019-2020

# Sommario

# Introduzione

Il sensore di impronte digitali è una tecnologia che identifica e autentica le impronte digitali di un individuo al fine di concedere o negare l'accesso a un sistema informatico o una struttura fisica. Questa tecnologia biometrica è più sicura di un passcode di poche cifre e la notevole usabilità ne favorisce l'adozione. Tuttavia, la scarsa mutabilità di questo dato biometrico pone importanti sfide nella sua gestione sicura soprattutto nei dispositivi mobili più comuni, gli smartphone. Il crescente utilizzo degli smartphone in diversi settori come l'assistenza sanitaria elettronica [1], le transazioni commerciali e personali [2] e il settore bancario [3], vede le impronte digitali svolgere un ruolo essenziale nella sicurezza e nella verifica dell'identità dell'utente. L'obiettivo di questa ricerca è quello di fornire lo stato dell'arte sulla tecnologia del fingerprint sui dispositivi mobili con sistema operativo Android. E' stato oggetto di studio l'intero flusso di autenticazione biometrica dal livello più alto, l'interfaccia utente, sino ai meccanismi hardware che costituiscono le fondamenta della sicurezza. E' indispensabile, inoltre, riformulare il threat model poiché per loro natura e utilizzo i dispositivi mobili pongono nuove sfide nella sicurezza. Per completare la ricerca è stata realizzata una applicazione Android che implementa i meccanismi di sicurezza visti nella teoria.

# Adozione delle biometrie

L'uso delle biometrie come mezzo di autenticazione è sempre più frequente. Ciò si verifica a causa dell'aumento dell'utilizzo di dispositivi mobili come smartphone, laptop e tablet.
I due leader degli OS per dispositivi mobili più diffusi, Apple e Google, hanno investito molto nell'implementazione all'interno dei loro sistemi delle biometrie. I vendor dei dispositivi, invece, hanno abbassato i costi rendendo la tecnologia del fingerprint uno standard per tutte le fasce di prezzo.
La tecnologia biometrica è utilizzata in molte applicazioni e settori come in quello bancario e finanziario (ad esempio, per l'autenticazione degli utenti per accedere alle cassette di sicurezza), nell'industria manifatturiera / commerciale (ad esempio, per la registrazione delle presenze), nelle forze dell'ordine (ad esempio, per identificare i sospetti durante le indagini sui reati), nell'assistenza sanitaria (ad esempio, per l'identificazione dei pazienti) e nella sicurezza nazionale e di frontiera[4].
Il pagamento elettronico continua a svolgere un ruolo importante nel futuro della tecnologia finanziaria e del commercio elettronico [5], soprattutto in considerazione della maggiore adozione di dispositivi mobili. Per pagamento elettronico ci si riferisce ai pagamenti effettuati utilizzando una forma di tecnologia dell'informazione e della comunicazione [6] e può includere carte di credito, carte di debito, pagamenti mobili e pagamenti web. Nonostante questa crescita, le questioni di sicurezza e privacy rimangono centrali nell'adozione dei sistemi di pagamento elettronico. Abrazhevich [5] suggerisce che i sistemi di pagamento elettronico tradizionali presentano alcune limitazioni nel contesto del commercio elettronico.
L'autenticazione biometrica per il pagamento è stata suggerita come alternativa ai metodi tradizionali di autenticazione del pagamento [7]. In effetti, gli esperti di FinTech hanno previsto che il pagamento elettronico tradizionale come le carte di credito basate su chip e PIN verrà eventualmente sostituito con il pagamento basato sull'autenticazione biometrica e mobile [8]. A supporto di questa tesi già oggi alcuni sistemi di pagamento come PayPal hanno integrato nella loro app mobile l'autenticazione biometrica.
Cohn [9] in un sondaggio rileva che il 92% dei consumatori del Regno Unito e il 69% dei consumatori statunitensi preferiscono che banche, società di carte di credito, operatori sanitari e organizzazioni governative adottino tecnologie biometriche, rispetto ad altre misure di protezione come lettori di smart card, secure token o password / PIN per verificare in sicurezza le loro identità. Questo numero crescente indica che gli utenti riconoscono la necessità di metodi di autenticazione più sicuri.
Per gli individui, alcuni dei maggiori vantaggi nell'utilizzo dell'autenticazione biometrica includono oltre una maggiore sicurezza e protezione anche un potenziale rischio minore di perdita e furto delle carte di credito [7]. L'aumento dell'autenticazione biometrica per il pagamento è stato associato a minori problemi di sicurezza (ad esempio prevenzione del furto di identità) e maggiore praticità [10], [11].
Un sondaggio del 2013, su un campione di 1924 individui diviso tra Stati Uniti, UK e Germania, ha rilevato che il 65% dei consumatori tedeschi e il 46% dei consumatori statunitensi non si fidano delle aziende online che utilizzano sistemi di autenticazione legacy con nome

utente e password[12]. In altre parole, i consumatori tendono a fidarsi delle aziende online e dei siti web che forniscono procedure di autenticazione più solide.
Uno studio del 2018, condotto su un campione di 94 studenti, per quanto sia un numero molto esiguo, conferma la tendenza della preferenza delle biometrie ai metodi tradizionali[4].
I risultati dimostrano che il fingerprint è percepito come più sicuro rispetto ai metodi di autenticazione tradizionali con carta di credito o carta di credito + PIN. In particolare, l'autenticazione con biometrie è risultata quella preferita tra coloro che non pongono particolare attenzione ai problemi di sicurezza.
Considerando l'aumento del numero di consumatori statunitensi colpiti da furti di identità e frodi con carte di credito [13], [14], si potrebbe ipotizzare che le vittime di questi tipi di crimini possano diventare futuri utilizzatori dell'autenticazione biometrica.

# Threat model dei dispositivi mobili

I dispositivi mobili, considerati quasi una estensione del nostro corpo, sono un ricco contenitore di dati sensibili a causa del loro utilizzo.
Il Threat model per i dispositivi mobili è diverso da quelli comunemente usati per i sistemi operativi desktop o server per due ragioni principali:

1. I dispositivi mobili possono essere facilmente smarriti, rubati o condivisi tra diversi utenti.
2. I dispositivi mobili solitamente sono connessi a reti pubbliche non trusted.

Pertanto, si presume che i dispositivi siano direttamente accessibili agli avversari o che si trovino in loro prossimità fisica come parte esplicita del threat model. Ciò include la perdita o il furto, ma anche la possibilità che più utenti (benigni ma potenzialmente curiosi) condividano un dispositivo (come uno smartphone o un tablet).

**Accesso fisico e prossimale**

Deriviamo le minacce specifiche (Threat) se l'accesso è fisico o prossimale:

T1. I dispositivi spenti sotto il completo controllo fisico di un attaccante, ad esempio controlli di frontiera o controlli doganali.
T2. Dispositivi con schermo bloccato sotto il completo controllo fisico di un attaccante, ad esempio attaccanti che tentano di esfiltrare i dati per ulteriori furti di identità.
T3. Dispositivi con schermo sbloccato (condiviso) sotto il controllo di un utente autorizzato ma diverso, ad esempio abuso del partner intimo, sottomissione volontaria a un controllo di frontiera o controllo doganale.
T4. Dispositivi (schermo bloccato o sbloccato) in prossimità fisica di un attaccante (con la presunta capacità di controllare tutti i canali di comunicazione radio disponibili, inclusi cellulare, WiFi, Bluetooth, GPS, NFC e FM), ad esempio attacchi diretti tramite Bluetooth [15]. Sebbene l'NFC possa essere considerato una categoria separata rispetto ad altri attacchi radio prossimali a causa della scala della distanza, lo includiamo ancora nella classe di minaccia di prossimità invece che di controllo fisico.

**Minacce a livello di rete**

Se si considera che la comunicazione di rete sia sotto il controllo completo di un attaccante allora, per semplicità, le minacce a livello di rete sono due:

T5. Intercettazioni passive e analisi del traffico, inclusi i dispositivi di tracciamento all'interno o attraverso le reti, ad esempio in base all'indirizzo MAC o altri identificatori di rete del dispositivo.
T6. Manipolazione attiva del traffico di rete, ad esempio MITM su connessioni TLS. Queste due minacce sono diverse da [T4] (attacchi radio prossimali) in termini di scalabilità degli

attacchi. Il controllo di un singolo punto di controllo in una grande rete può essere utilizzato per attaccare un gran numero di dispositivi, mentre gli attacchi radio prossimali (ultimo salto) richiedono la vicinanza fisica ai dispositivi di destinazione dispositivo.

**Esecuzione di codice non trusted**

Una differenza fondamentale rispetto ad altri sistemi operativi mobili è che Android consente intenzionalmente (con il consenso esplicito degli utenti finali) l'installazione dell'applicazione da fonti arbitrarie e non impone il controllo delle app da parte di un'istanza centrale. Ciò implica vettori di attacco su più livelli:

T7. Abuso di API supportate dal sistema operativo con intenti dannosi, ad esempio spyware.
T8. Sfruttamento di bug nel sistema operativo, ad esempio kernel, driver o servizi di sistema [16] [17].
T9. Abuso di API supportate da altre app installate sul dispositivo [18, pag. 2018-21066].
T10. Esecuzione di codice non trusted dal web (cioè JavaScript) senza consenso esplicito.
T11. Sistema di imitazione o altre interfacce utente di app per confondere gli utenti, ad esempio per inserire PIN / password in un app dannosa[19].
T12. Lettura di contenuti dal sistema o da altre interfacce utente di app, ad esempio per eseguire lo screen-scraping di dati riservati da un'altra app [20] [21].
T13. Iniezione di eventi di input nel sistema o in altre interfacce utente dell'app [22].

**Elaborazione di contenuto non trusted**

Oltre a eseguire direttamente codice non trusted, i dispositivi elaborano un'ampia varietà di dati non trusted, incluse le strutture dati più complesse come i media. Questo porta direttamente a minacce riguardanti l'elaborazione di dati e metadati:

T14. Sfruttamento del codice che elabora i contenuti non trusted nel sistema operativo o nelle app, ad esempio nelle librerie multimediali [23]. Questa può essere sia una superficie di attacco locale che remota, a seconda di dove vengano presi i dati di input.
T15. Abuso di identificatori univoci per attacchi mirati (che possono verificarsi anche su reti trusted), ad esempio utilizzando un numero di telefono o un indirizzo e-mail per lo spam o la correlazione con altri set di dati, comprese le posizioni geofisiche.

# Diffusione di Android

Android è, al momento, il sistema operativo per utenti finali più diffuso. Con oltre 2 miliardi di dispositivi attivi mensilmente nel 2017 [24] e una tendenza generale verso l'uso mobile dei servizi Internet, Android è ora l'interfaccia più comune per l'interazione degli utenti globali con i servizi digitali. Tra i diversi form factors (inclusi telefoni, tablet, dispositivi indossabili, TV, IOT, automobili e categorie per usi più speciali) esiste una vasta, e ancora in crescita, gamma di casi d'uso dalla comunicazione al consumo di media e intrattenimento, per la finanza, salute e sensori / attuatori fisici. Molte di queste applicazioni sono sempre più critiche per la sicurezza e la privacy e Android, come sistema operativo, deve fornire garanzie sufficienti e appropriate a utenti e sviluppatori.
Android è un sistema operativo incentrato sull'utente finale. L'ovvia implicazione è che, come sistema operativo consumer, deve essere utile per gli utenti e attraente per gli sviluppatori. L'attenzione per l'utente finale implica che le interfacce utente e i flussi di lavoro debbano essere sicuri per impostazione predefinita e richiedano un intento esplicito per qualsiasi azione che potrebbe compromettere la sicurezza o la privacy. Ciò significa che il sistema operativo non deve delegare decisioni sulla sicurezza o sulla privacy tecnicamente dettagliate a utenti non esperti che non sono sufficientemente qualificati o esperti per prenderle.
L'ecosistema Android è immenso e le statistiche più recenti, dal 2018 ad oggi, lo vedono contendersi il primato, come OS più diffuso a livello globale, con i sistemi Windows, rispettivamente al 35% e 40% [25]. Inoltre, ci sono centinaia di diversi OEM (produttori di apparecchiature originali, ovvero produttori di dispositivi) che producono decine di migliaia di dispositivi Android in diversi form factors [26]. Alcuni di questi OEM non hanno competenze tecniche dettagliate ma si affidano a ODM (Original Device Manufacturers) per lo sviluppo di hardware e firmware, occupandosi quindi solo dell'imballaggio o di rinominare i dispositivi con il proprio marchio. Soltanto i dispositivi forniti con l'integrazione dei servizi Google necessitano della certificazione del firmware, mentre i dispositivi basati semplicemente su AOSP (Android Open Source Project) possono essere realizzati senza autorizzazione o registrazione. Pertanto, non esiste un unico registro che elenchi tutti gli OEM e l'elenco è in continua evoluzione con lo sviluppo continuo di nuovi concetti hardware.
Tuttavia, i dispositivi che utilizzano Android come nome di marchio per pubblicizzare la loro compatibilità con le app Android devono superare la Compatibility Test Suite (CTS).

# Autenticazione

Sui dispositivi mobili il metodo di autenticazione principale prevede il lockscreen, una schermata di blocco. Essa è un evidente compromesso tra sicurezza e usabilità: da un lato, gli utenti sbloccano i dispositivi per interazioni brevi (10-250 secondi) circa 50 volte al giorno in media e anche fino a 200 volte in casi eccezionali [27], [28] e il lockscreen è ovviamente un ostacolo all'interazione immediata con il dispositivo [29], [30]. D'altra parte, i dispositivi privi di una schermata di blocco sono immediatamente soggetti ad abusi da parte di utenti non autorizzati ([T1] - [T3]) e il sistema operativo non può applicare in modo affidabile il consenso dell'utente senza autenticazione. Pertanto, è di fondamentale importanza che la schermata di blocco raggiunga un ragionevole equilibrio tra sicurezza e usabilità. A tal fine, le recenti versioni di Android utilizzano un modello di autenticazione a più livelli in cui un meccanismo di autenticazione basato su fattori di conoscenza può essere supportato da modalità più convenienti che sono funzionalmente vincolate in base al livello di sicurezza che forniscono. La comodità aggiuntiva offerta da un tale modello aiuta a promuovere l'adozione di lockscreen e consente di beneficiare sia dei vantaggi immediati della sicurezza di un lockscreen sia di funzionalità come la crittografia basata su file che si basano sulla presenza di una credenziale fornita dall'utente.
A partire da Android 7.x il 77% dei dispositivi con sensori di impronte digitali ha un lockscreen sicuro abilitato contro il 50% dei dispositivi privi di fingerprint che non ne fanno uso. A partire da Android 9.0, il modello di autenticazione suddivide le modalità in tre livelli:

1. Le modalità di autenticazione primaria sono limitate ai fattori di conoscenza e per impostazione predefinita includono PIN, Pattern e password. L'autenticazione primaria fornisce l'accesso a tutte le funzioni del dispositivo.
2. Le modalità di autenticazione secondaria sono biometrie "forti", come definito dai loro tassi di SAR e IAR.
3. Le modalità di autenticazione terziaria sono i dati biometrici deboli che non superano la soglia di spoofabilità o modalità alternative come lo sblocco con un dispositivo Bluetooth associato.

# Requisiti dei sensori in Android

Per essere considerate compatibili con Android, le implementazioni dei dispositivi devono soddisfare i requisiti presentati nel documento CDD (Compatibility Definition Document) Android[31]. Il CDD di Android 10 valuta la sicurezza di un'implementazione biometrica utilizzando la **sicurezza dell'architettura** e l'indice di **spoofability**:

- **Sicurezza architettonica**: esprime la resilienza della pipeline biometrica alla compromissione del kernel o della piattaforma. Una pipeline è considerata sicura se le compromissioni del kernel e della piattaforma non conferiscono la capacità di leggere dati biometrici raw o di iniettare dati sintetici nella pipeline al fine di influenzare la decisione di autenticazione.
- **Spoofability**: è misurato dal tasso di accettazione dello spoofing (SAR) del biometrico. Il SAR è una metrica introdotta in Android 9 per misurare la resilienza di un biometrico contro un aggressore dedicato.

Le performance dell'autenticazione biometrica solitamente sono misurate con due metriche ampiamente utilizzate nel machine learning[32]:

- **False Accept Rate (FAR)**: misura quanto spesso un modello accetta erroneamente un input errato scelto a caso.
  Il FAR esprime un problema di sicurezza.
- **False Reject Rate (FRR)**: misura quanto spesso un modello classifica l'input valido come non corretto.
  Il FRR esprime un problema di usabilità.

Entrambe le metriche sono ottime nella misurazione dell'accuracy e della precision di un modello di machine learning o biometrico se applicato su sample randomici. Tuttavia, entrambe le metriche non considerano un attaccante attivo come parte del threat model e di conseguenza non forniscono informazioni sufficienti per valutare quanto questo resista agli attacchi mirati. Con Android 8.1 sono state introdotte due nuove metriche che considerano un attaccante in maniera esplicita:

- **Spoof Accept Rate (SAR):** misura la probabilità che un modello biometrico accetti un campione valido noto in precedenza. Ad esempio, con lo sblocco vocale questo misurerebbe le possibilità di sbloccare il dispositivo di un utente utilizzando un campione registrato di loro che dice: "Ok, Google".

  Chiamiamo tali attacchi **Spoof Attacks.**

- **Imposter Accept Rate (IAR):** misura la probabilità che un modello biometrico accetti input che imitano un buon campione noto. Ad esempio, nel meccanismo di sblocco vocale questo misurerebbe la frequenza con cui qualcuno sblocchi il dispositivo che cerca di imitare la voce di un utente (utilizzando toni e accenti simili) riesca a sbloccare il proprio dispositivo. Chiamiamo tali attacchi **Imposter Attacks.**

Da Android 11, un sensore biometrico è classificato come **Class 3** (Strong), **Class 2** (Weak), o **Class 1** (Convenience) in base alla combinazione di SAR, IAR e sicurezza architetturale.
Nei casi in cui non esiste un imposter attack, come per il fingerprint, si considerano FAR e SAR.
Tutti i sensori sono classificati di default come Class 1. Il sensore dovrà obbligatoriamente soddisfare alcuni requisiti per essere classificato come Class 2 o Class 3.
In generale, i requisiti sono organizzati in maniera gerarchica [33] ove la Classe 1 contiene i requisiti più semplici che dovranno essere soddisfatti anche da Class 2 e Class 3.

Requisiti di Class 1 più rilevanti:

- FAR > 0.002%
- FRR < 10%
- Le biometrie prima di poter essere abilitate devono notificare l'utente con un messaggio di warning dei possibili rischi derivanti dal loro uso.
- Sospensione dell'autenticazione per 30 secondi dopo cinque tentativi falliti.
- Prima di aggiungere nuove biometrie è necessario stabilire la chain of trust ovvero la conferma o l'aggiunta di nuove credenziali (PIN/pattern/password) rese sicure dal TEE (Trusted Execution Environment).
- La rimozione dell'account utente o il factory reset deve comportare la cancellazione di tutti i dati biometrici.
- L'utente deve autenticarsi con il meccanismo primario di autenticazione (PIN/pattern/password) dopo un periodo di 4 ore di inattività o tre tentativi di autenticazione biometrica falliti o 24 ore di attività per dispositivi nati con Android 10, 72 ore per i device aggiornati successivamente.

Requisiti di Class 2 più rilevanti:

- Il matching biometrico deve avvenire in un ambiente di esecuzione isolato al di fuori di Android e dello spazio Kernel, come il TEE o su un chip con un canale sicuro verso l'ambiente di esecuzione isolato.

- I dati devono essere cifrati e crittograficamente autenticati cosicché sia garantita la loro integrità e non possano essere alterati al di fuori dell'ambiente di esecuzione isolato.

Requisiti di Class 3 più rilevanti:

- SAR < 7%

- L'implementazione del keystore deve essere di tipo hardware-backed.

- L'utente deve autenticarsi con il meccanismo primario di autenticazione (PIN/pattern/password) ogni 72 ore.

Da una rapida comparazione tra le diverse classificazioni si evince che:

- Il SAR sia la metrica determinante, in particolare il range 0-7% per la Class 3. Tale intervallo è stato definito come standard essendo il più comune tra tutte le implementazioni. Il range verrà ulteriormente ridotto al miglioramento dei sensori e degli algoritmi di classificazione.
- L'autenticazione con biometrie, anche se Class 3, non è considerata come meccanismo primario di autenticazione affidata invece a knowledge factors come PIN, Pattern e Password.
- La sicurezza dei dati sensibili e le operazioni crittografiche sono eseguite in un ambiente di esecuzione isolato chiamato TEE (Trusted Execution Environment).

## Flusso di autenticazione

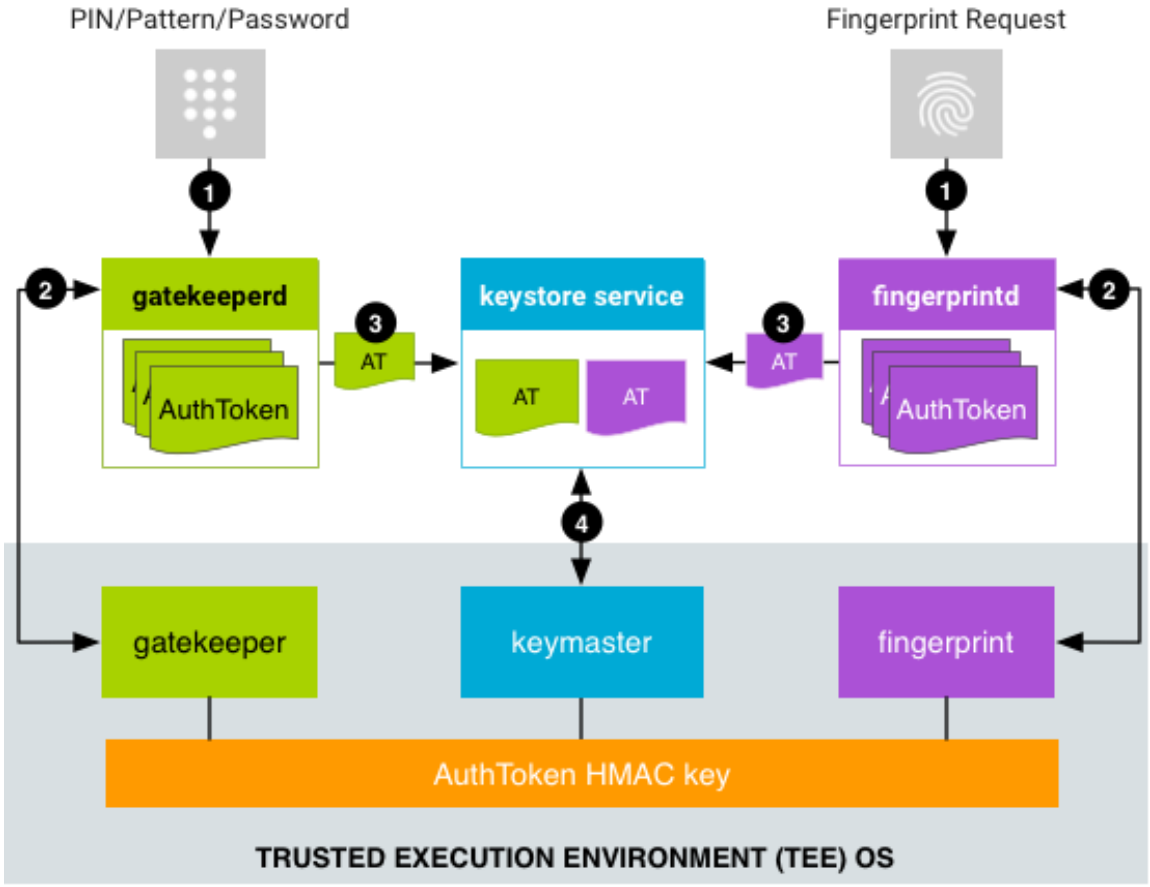


*Figure 1 Flusso di autenticazione*

### I. Device Boot - AuthToken HMAC key generation

Ad ogni avvio di un dispositivo [34], **la chiave AuthToken HMAC (**Figure 1**)** deve essere generata e condivisa con tutti i componenti TEE (**Gatekeeper**, **Keymaster** e trustlet biometrici supportati). Pertanto, per una maggiore protezione dagli attacchi di replay, la chiave HMAC deve essere generata in modo casuale ogni volta che il dispositivo si riavvia.
La chiave HMAC non è mai disponibile al di fuori del TEE e il protocollo di condivisione con gli altri componenti è una implementazione dipendente dalla piattaforma. Se un sistema operativo TEE non dispone di un meccanismo di comunicazione interno tra processi (IPC) e deve trasferire i dati tramite il sistema operativo non trusted, ovvero Android, tale trasferimento deve essere eseguito tramite un protocollo di scambio di chiavi sicuro.
Il sistema operativo **Trusty**, eseguito parallelamente ad Android, è un esempio di TEE, ma è possibile utilizzare altri TEE. Trusty utilizza un sistema IPC interno per comunicare direttamente tra Keymaster e Gatekeeper o il trustlet biometrico appropriato.
**La chiave HMAC è memorizzata esclusivamente nel Keymaster**; Fingerprint e Gatekeeper richiedono la chiave dal Keymaster per ogni utilizzo ma non ne memorizzano mai il valore.

## II. Enrollment

Al primo avvio del dispositivo dopo un ripristino delle impostazioni di fabbrica, tutti gli autenticatori sono pronti a ricevere le registrazioni delle credenziali dall'utente. Un utente deve inizialmente registrare un PIN o Pattern o Password con Gatekeeper. Questa registrazione iniziale crea un **SID (User Secure Identifier) a 64 bit** generato in modo casuale che funge da identificatore per l'utente e da token di associazione per il materiale crittografico dell'utente [34]. Questo SID utente è associato crittograficamente alla password dell'utente; le autenticazioni validate da Gatekeeper producono AuthTokens che contengono il SID utente per quella password.
Un utente che desidera modificare una credenziale deve presentare una credenziale esistente. Se una credenziale esistente è verificata correttamente, il SID utente associato alla credenziale esistente è trasferito alla nuova credenziale, consentendo all'utente di continuare ad accedere alle chiavi dopo aver modificato una credenziale. Se un utente non presenta una credenziale esistente, la nuova credenziale viene registrata con un SID utente completamente casuale. L'utente può accedere al dispositivo, ma le chiavi create con il vecchio SID utente sono perse definitivamente. Questa operazione è nota come *registrazione non trusted* .
In circostanze normali, il framework Android non consente una registrazione non trusted, quindi la maggior parte degli utenti non vedrà mai questa funzionalità. Tuttavia, la reimpostazione forzata della password, da parte di un amministratore del dispositivo o di un utente malintenzionato, potrebbe causare ciò.

## III. Autenticazione

La fase di autenticazione avviene esclusivamente dopo le fasi di “Device Boot - AuthToken HMAC key generation” e “Enrollment”:

- Tutti i componenti TEE condividono una chiave AuthToken HMAC che utilizzano per autenticare i messaggi degli altri.
- La registrazione delle credenziali ha generato uno user SID.

Il flusso di autenticazione prosegue come mostrano gli step della Figure 1:

1. Un utente fornisce un metodo di autenticazione e il servizio associato effettua una richiesta al suo daemon:
    - Per PIN, Pattern e password, LockSettingsService invia una richiesta a gatekeeperd.
    - I flussi di autenticazione basati sulla biometria dipendono dalla versione di Android:
        i. Sui dispositivi con Android 8.x e versioni precedenti, FingerprintService effettua una richiesta a fingerprintd.
        ii. Sui dispositivi con Android 9 e versioni successive, BiometricPrompt effettua una richiesta al daemon biometrico appropriato (ad esempio, fingerprintd per le impronte digitali o faced per il volto) utilizzando la classe Biometric Manager appropriata, come FingerprintManager o FaceManager.

      Indipendentemente dalla versione, l'autenticazione biometrica avviene in modo asincrono dopo l'invio della richiesta.

2. Il daemon invia i dati alla sua controparte che genera un AuthToken:

    - Per l'autenticazione con PIN / Pattern / Password il gatekeeperd invia l'hash al Gatekeeper nel TEE.
      Se l'autenticazione nel TEE ha esito positivo allora il Gatekeeper nel TEE invia un AuthToken contenente il SID utente corrispondente (firmato con la chiave HMAC AuthToken) alla sua controparte nel sistema operativo Android.
    - Per l'autenticazione dell'impronta digitale, fingerprintd ascolta gli eventi relativi alle impronte digitali e invia i dati a Fingerprint nel TEE.
      Se l'autenticazione nel TEE ha esito positivo, Fingerprint nel TEE invia un AuthToken (firmato con la chiave HMAC AuthToken) alla sua controparte nel sistema operativo Android.
    - Per altre autenticazioni biometriche, il daemon biometrico appropriato ascolta l'evento biometrico e lo invia al componente TEE biometrico appropriato.

3. Il daemon riceve un AuthToken firmato e lo passa al servizio keystore tramite un'estensione dell'interfaccia Binder del servizio keystore. (gatekeeperd notifica il servizio keystore sia quando il dispositivo è nuovamente bloccato sia quando la password del dispositivo è cambiata).

4. Il servizio keystore passa gli AuthTokens al Keymaster che li verifica utilizzando la chiave condivisa con Gatekeeper e il componente TEE biometrico supportato. Il Keymaster rilascia la chiave per un intervallo di tempo limitato che ha inizio con il timestamp nel token prima di richiedere nuovamente l’autenticazione.

Nota: gli AuthTokens sono invalidati al riavvio di un dispositivo.

# Fingerprint HIDL

Android utilizza il Fingerprint Hardware Interface Definition Language (HIDL) per connettersi alla specifica libreria del vendor e al sensore hardware[35].

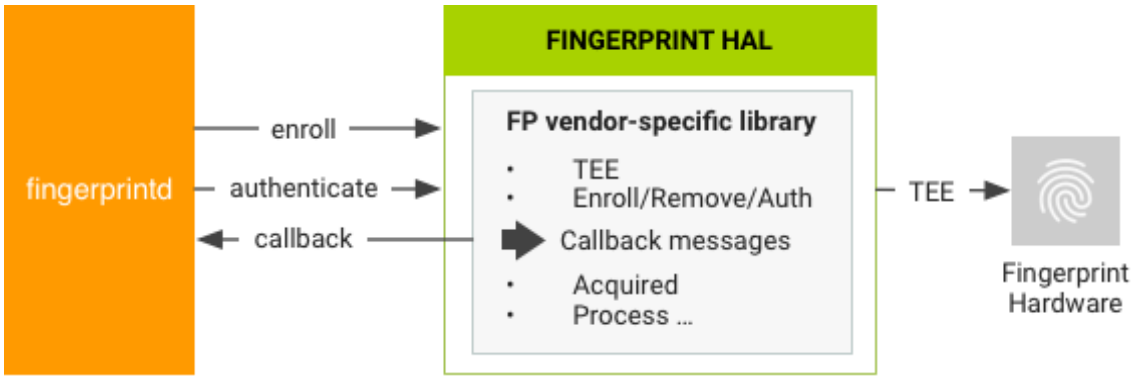


*Figure 2 Interazione del daemon fingerprint con la libria del vendor*

Nel caso del fingerprint, il daemon fingerprintd comunica con il **Fingerprint HAL**, un'implementazione C / C ++ dell'interfaccia HIDL IBiometricsFingerprint.
Il Fingerprint HAL contiene la libreria specifica del vendor per comunicare con l'hardware specifico del dispositivo, utilizzando il protocollo di comunicazione richiesto dal TEE.
**FingerprintService** e **fingerprintd** effettuano call tramite il Fingerprint HAL alla libreria specifica del fornitore per registrare le impronte digitali ed eseguire altre operazioni.

## Linee guida per l'implementazione

Le seguenti linee guida per le Fingerprint HAL sono progettate per garantire che i dati relativi alle impronte digitali **non** siano **diffusi** e siano invece **rimossi** nel momento in cui l'utente è rimosso dal dispositivo:

- I raw data o derivati delle impronte digitali (ad esempio, templates) non devono essere mai accessibili all'esterno del driver del sensore o del TEE. Se l'hardware supporta un TEE, l'accesso all'hardware deve essere limitato al TEE e protetto da una policy SELinux. Il canale Serial Peripheral Interface (SPI) deve essere accessibile solo al TEE e deve essere presente una policy SELinux esplicita su tutti i file del dispositivo.
- L'acquisizione, l'enrollment e il riconoscimento delle impronte digitali devono avvenire all'interno del TEE.
- Solo la forma crittografata dei dati dell'impronta digitale può essere archiviata nel file system, anche se il file system stesso è crittografato.
- I Fingerprint templates devono essere firmati con una chiave privata specifica del dispositivo. Per Advanced Encryption Standard (AES), almeno un template deve essere firmato con il percorso assoluto del file system, il gruppo e l'ID del dito in modo tale che i file del template non siano utilizzabili su un altro dispositivo o per chiunque non sia l'utente originale.

- Le implementazioni devono utilizzare il percorso del file system fornito dalla funzione **setActiveGroup()** per archiviare i template crittografati o fornire un modo per cancellare tutti i dati relativi ai template se l'utente venisse rimosso.
- Il rooting **non deve** essere in grado di compromettere i dati biometrici.

## TrustZone: la tecnologia abilitante per il TEE

Il flusso di autenticazione di Android richiede che alcuni componenti come Gatekeeper, Fingerprint e Keymaster siano eseguiti all'interno della TEE (Trusted Execution Enviroment). Questa scelta architetturale è motivata dal fatto che i driver hardware forniti dai vendor dei SoC (System on a Chip) rappresentano la causa più comune delle vulnerabilità kernel in Android [36]. Inoltre il progressivo aggiornamento di Android negli anni si è concentrato sull'irrigidimento dello userspace Android rendendo il kernel, di contro, un target ancor più attraente per attacchi di privilege escalation[37].
Il TEE, realizzato attraverso il meccanismo hardware TrustZone, è al di sotto del kernel e consente di eseguire in maniera sicura le applicazioni critiche con una TCB (Trusted computing base) di diversi ordini di grandezza inferiore rispetto all'OS locale, in questo caso Android.
Per definizione da parte dell'Orange Book il TCB è: *"the totality of protection mechanisms within it, including hardware, firmware, and software, the combination of which is responsible for enforcing a computer security policy."[38]*
Tipicamente una TCB è formata dal kernel OS, da servizi privilegiati e librerie.
La grandezza di una TCB determina la sua superficie di attacco poiché la quantità di codice e le interazioni complesse tra i componenti interni aumentano le potenziali vulnerabilità.
D'altra parte, le TCB di grandi dimensioni tendono ad essere più esposte agli attaccanti per via di un maggior numero di porte aperte sfruttabili come punti di ingresso. Le applicazioni che si basano su piattaforme software così complesse, inevitabilmente, ereditano le potenziali carenze di sicurezza del TCB sottostante[39].
Il meccanismo TrustZone, che fornisce il TEE, permette di affrontare il problema dell'ingrandimento delle TCB. Più specificamente, un TEE è costituito da un ambiente isolato in cui le applicazioni considerate trusted possono essere eseguite senza l'interferenza del sistema operativo locale (non trusted), in questo caso specifico Android. Le proprietà di sicurezza di un TEE garantiscono la confidentiality e l'integrity dei calcoli che avvengono al suo interno. Inoltre, per imporre l'esecuzione isolata, un'astrazione TEE definisce i meccanismi per il provisioning sicuro di codice e dati (comprese le chiavi crittografiche) nel TEE e nei canali trusted[40] per recuperare i risultati delle computazioni ed errori.
È anche comune per un TEE l'accesso a uno spazio di archiviazione sicuro privato e consentire ad entità remote di verificare l'integrità e l'autenticità dell'ambiente TEE attraverso un protocollo di attestazione remota [41].

# Arm TrustZone

L’Arm TrustZone è un insieme di estensioni di sicurezza hardware introdotte nei processori applicativi Arm (Cortex-A) nel 2004. Più recentemente, TrustZone è stato adattato per coprire la nuova generazione di microcontrollori Arm (Cortex-M). TrustZone segue un approccio alla sicurezza SoC e CPU a livello di sistema.
Questa tecnologia è incentrata sul concetto di domini di protezione denominati secure world e normal world. Il software eseguito dal processore è eseguito in maniera esclusiva su uno dei due stati[41]. Entrambi i mondi sono completamente isolati dall'hardware e concessi privilegi non uniformi, con il software non protetto cui è impedito di accedere direttamente alle risorse del secure world. Questa forte separazione tra i mondi imposta dall'hardware apre nuove opportunità per la protezione di applicazioni e dati. In particolare, vincolando il sistema operativo a operare entro i confini del normal world, le applicazioni critiche possono risiedere all'interno del secure world senza la necessità di fare affidamento sul sistema operativo per la protezione.
Da diversi anni, TrustZone è ampiamente disponibile sui dispositivi mobili. Sfortunatamente, nonostante tutto il suo potenziale per migliorare la sicurezza, questa tecnologia è rimasta in uno stato di relativa oscurità per molto tempo [42], [43]. I produttori di SoC abilitati per TrustZone erano in qualche modo riluttanti a divulgare dettagli tecnici, spesso richiedendo agli sviluppatori di firmare accordi di non divulgazione (NDA) [43] prima che qualsiasi dettaglio relativo all'architettura potesse essere loro divulgato.
Di conseguenza, per quasi dieci anni, TrustZone è stato utilizzato principalmente dai produttori di dispositivi per monetizzare servizi protetti proprietari, la cui sicurezza era difficile da esaminare a causa della loro natura chiusa. Tuttavia, negli ultimi anni, abbiamo assistito a un crescente interesse per TrustZone sia dal mondo accademico che industriale. TrustZone è stato utilizzato in molti progetti di ricerca accademica e prodotti commerciali, fornendo le basi di sicurezza per sistemi come Samsung Knox [44], Keystore di Android [45] e OP-TEE [46].
I progetti esistenti abbracciano vari domini applicativi, in particolare mobile [47], [48], industriale [49], [50], automobilistico [51] e aerospaziale [52], e sono rilasciati in base a diverse politiche di licenza (open source e proprietarie) . Una dinamica comunità open source è maturata, contribuendo allo sviluppo di vari progetti basati su TrustZone e organismi di standardizzazione, come GlobalPlatform, hanno lavorato alla definizione di specifiche API comuni per promuovere l'interoperabilità attraverso soluzioni basate su TrustZone.
I processori ARM condividono la maggior parte dei mercati mobili ed embedded, alimentando oltre il 60% di tutti i dispositivi embedded e 4,5 miliardi di smartphone. Arm ha ulteriormente esteso il supporto TrustZone per i dispositivi più piccoli di fascia bassa, che Arm stessa stima raggiungerà quasi 1 trilione entro il 2035 [53]. Si prevede che una tale pletora di dispositivi interconnessi genererà e scambierà notevoli quantità di dati con contenuti critici per la sicurezza e sensibili alla privacy, che tenderanno ad attrarre la criminalità informatica.
Le architetture TEE possono essere distinte in due tipi in base al programma trusted eseguito nel mondo protetto: kernel TEE o servizio TEE. Nel primo caso, il programma implementa un set base di funzioni del sistema operativo per gestire più istanze TEE, ciascuna delle quali ospita una particolare applicazione [54]. Il kernel trusted è responsabile: della gestione della

memoria del secure world, dell’applicazione della protezione della memoria per ogni TEE, della gestione della comunicazione tra TEE e sistema operativo e della fornitura di un'API alle applicazioni TEE. Invece, i servizi TEE implementano una funzione specifica e non richiedono alcuna logica del sistema operativo di basso livello per gestire la propria memoria e le comunicazioni cross-world. Per evitare interferenze reciproche, è possibile distribuire un solo servizio TEE sul dispositivo mentre i kernel TEE consentono l'esecuzione di più applicazioni in istanze TEE indipendenti. Tuttavia, uno svantaggio dei kernel TEE è che normalmente dipendono da TCB più grandi rispetto ai sistemi in cui viene distribuito un singolo servizio TEE[41].

## Compromissione del Kernel

Consideriamo esplicitamente la possibilità di una compromissione del kernel (ad esempio attaccando direttamente alcune interfacce del kernel basate sull'accesso fisico [T2] - [T4] o concatenando più bug dello user space per raggiungere le superfici del kernel in [T8]), configurazioni errate (es. con policy SELinux errate o eccessivamente permissive [34]), o bypass (es. modificando la boot chain per avviare un kernel diverso con policy di sicurezza disattivate) come parte del threat model per alcuni scenari selezionati. Con un kernel compromesso, Android non soddisfa più i requisiti di compatibilità e molte delle garanzie di sicurezza e privacy per utenti e app non sono più valide. Tuttavia, possiamo ancora difenderci da alcune minacce anche sotto questo presupposto:
Keymaster implementa il keystore Android in TEE per proteggere l'archiviazione delle chiavi crittografiche e l'utilizzo in caso di compromissione del kernel in fase di esecuzione [45]. Anche con un kernel completamente compromesso, un utente malintenzionato non può leggere il contenuto della chiave memorizzato in Keymaster. Le app possono richiedere esplicitamente che le chiavi siano archiviate in Keymaster, ovvero vincolate all'hardware, ed essere accessibili solo dopo l'autenticazione dell'utente, questa legata a Gatekeeper / Weaver, o richiedere certificati di attestazione per verificare queste proprietà della chiave [55].
A partire da Android 9.0 sui dispositivi Google (Pixel 3), è stata rinominata come Strongbox una implementazione specifica hardware-backed del keystore Android, resistente alle manomissioni (Tamper Resistant Hardware - TRH) per un migliore isolamento. Questo mitiga [T1] e [T2] da attacchi sofisticati, ad es. contro cold boot memory attacks [56] o bug hardware come Spectre / Meltdown [57], [58], Rowhammer [59] o Clkscrew [60] che consentono l'escalation dei privilegi anche dal kernel al TEE. Dal punto di vista dell'hardware, l'AP (Application Processor) avrà sempre una superficie di attacco notevolmente più ampia rispetto all'hardware protetto dedicato.
Si noti che solo la memorizzazione e l'utilizzo di chiavi in TEE o TRH non risolve completamente il problema del renderle inutilizzabili nell'ipotesi di una compromissione del kernel: se un attaccante ottenesse l'accesso alle interfacce di basso livello per comunicare direttamente con Keymaster o Strongbox, potrebbe usarlo come un oracolo per le operazioni crittografiche che richiedono la chiave privata. Per questo motivo le chiavi possono essere vincolate all'autenticazione e / o richiedere la verifica della presenza dell'utente, ad es. premendo un pulsante hardware rilevabile dal TRH per garantire che le chiavi non vengano utilizzate in background senza il consenso dell'utente.

# App Android

Per completare questa ricerca è stata sviluppata una app Android implementando i meccanismi di sicurezza visti nella teoria. In breve, l'applicazione consente di cifrare e decifrare una stringa di testo richiedendo l'autenticazione dell'utente mediante il fingerprint.
Per gestire le chiavi crittografiche è stato scelto l'AndroidKeyStore, una implementazione, specifica per Android delle API Keystore di Java. Ad esempio, utilizzando tale keystore non è possibile inserire chiavi nel keystore in maniera hardcoded in coerenza con i principi visti nella teoria ovvero che il loro ciclo di vita deve avvenire all'interno del TEE.

Per generare in maniera sicura le chiavi è prevista la classe: **android.security.keystore.KeyGenParameterSpec.Builder.**

Tale classe permette di specificare tutti i parametri della chiave tra cui:

- Un alias per identificare la chiave. Il mapping con il valore segreto è compito esclusivo del Keystore.
- La dimensione della chiave
- Il purpose della chiave: cifratura e / o decifratura
- Algoritmo di cifratura

L'applicazione fa uso dell'algoritmo di cifratura simmetrica AES.
Le nuove versioni di Android hanno aggiunto nuovi parametri più specifici il cui utilizzo combinato garantisce la massima sicurezza:

1. **setUserAuthenticationRequired (boolean required)**

   Il rilascio della chiave avviene solo se l'utente è stato autenticato da cui consegue che la chiave può essere generata solo se il lock screen è attivo ed è invalidata se disabilitato.

2. **SetInvalidatedByBiometricEnrollment(boolean invalidateKey)**

   La registrazione di una nuova fingerprint invalida le chiavi già registrate. Uno scenario che esprime l'utilità di questa proprietà è il furto di un dispositivo già sbloccato su cui l'attaccante potrebbe registrare la propria fingerprint.

3. **setUserAuthenticationParameters(int timeout, int type)**

   - timeout: indica l'intervallo di tempo di rilascio della chiave, il valore 0 implica l'autenticazione per ogni singolo utilizzo.
   - type: specifica il tipo di credenziale valido per l'autenticazione se conoscitivo (PIN, Pattern, Password) e/o biometriche.

Il flusso di autenticazione tuttavia può essere bypassato da script di penetration testing se non vi è uso dell'oggetto **BiometricPrompt.CryptoObject** che contiene il riferimento della chiave del Keystore da rilasciare.
Il flusso di autenticazione corretto prevede che l'utente autenticato con successo triggera il metodo **onAuthenticationSucceeded()** tuttavia se quest'ultimo è implementato in maniera incorretta consente di passare un CryptoObject nullo a tale metodo che validerà in qualsiasi caso l'autenticazione.
La soluzione per una autenticazione sicura prevede:

1. La generazione della chiave dovrebbe avere tutti i parametri sopra elencati.
2. L'inizializzazione del cipher object con la chiave generata.
3. La creazione di un CryptoObject usando il cipher object.
4. L'implementazione della chiamata onAuthenticationSucceeded() in modo che recuperi il CryptoObject dal metodo BiometricPrompt.AuthenticationResult e lo usi per recuperare il cipher object che cifra/decifra il dato.

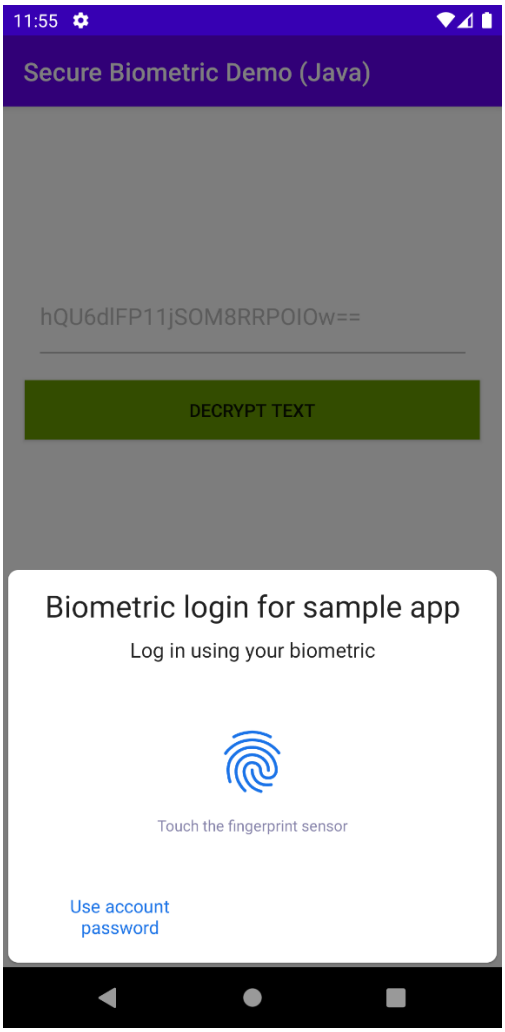


*Figure 3 Applicazione con autenticazione sicura*

# Conclusioni

L’adozione sempre più frequente delle biometrie in contesti diversi e la natura aperta di Android ha imposto la definizione di API che abbassassero il carico di lavoro per gli sviluppatori sul lato security. A prova di ciò, la realizzazione dell’applicazione per questa ricerca non ha richiesto conoscenze di alto livello poiché i parametri di default coprono la maggior parte dei casi d’uso e la documentazione risulta esaustiva. Abbiamo dimostrato però come alcuni errori implementativi potrebbero causare gravi vulnerabilità, come il bypass dell’autenticazione. Si auspica che le prossime versioni di Android definiscano metodi più rigidi che semplifichino ancora l’implementazione mantenendo al contempo una sicurezza elevata. Inoltre, si auspica che i meccanismi di sicurezza basati su chip hardware dedicati, come Strongbox, diventino uno standard per tutte le fasce di prezzo dei dispositivi mobili.

# Bibliografia


[1] E. G. Spanakis, M. Spanakis, A. Karantanas, e K. Marias, «Secure access to patient's health records using SpeechXRays a mutli-channel biometrics platform for user authentication», in *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, ago. 2016, pagg. 2541–2544, doi: 10.1109/EMBC.2016.7591248.

[2] S. S. Das e S. J. Debbarma, «Designing a Biometric Strategy (Fingerprint) Measure for Enhancing ATM Security in Indian e-banking System», *Int. J. Inf. Commun. Technol. Res.*, pagg. 197–203, 2011.

[3] R. Donida Labati e F. Scotti, «Fingerprint», in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg e S. Jajodia, A c. di Boston, MA: Springer US, 2011, pagg. 460–465.

[4] O. Ogbanufe e D. J. Kim, «Comparing fingerprint-based biometrics authentication versus traditional authentication methods for e-payment», *Decis. Support Syst.*, vol. 106, pagg. 1–14, feb. 2018, doi: 10.1016/j.dss.2017.11.003.

[5] D. Abrazhevich, *Electronic Payment Systems: a User-Centered Perspective and Interaction Design*. Dennis Abrazhevich, 2004.

[6] «A finite mixture logit model to segment and predict electronic payments system adoption — Arizona State University». https://asu.pure.elsevier.com/en/publications/a-finite-mixture-logit-model-to-segment-and-predict-electronic-pa (consultato ott. 19, 2020).

[7] R. Clodfelter, «Biometric technology in retailing: Will consumers accept fingerprint authentication?», *J. Retail. Consum. Serv.*, vol. 17, n. 3, pagg. 181–188, mag. 2010, doi: 10.1016/j.jretconser.2010.03.007.

[8] «Fintech Experts Say Mobile And Biometric Authentication to Replace PINs Within Five Years», *Fintech Finance*, nov. 23, 2015. https://www.fintechf.com/01-news/fintech-experts-say-mobile-and-biometric-authentication-to-replace-pins-within-five-years/ (consultato ott. 19, 2020).

[9] M. Cohn, «Biometrics: Key to securing consumer trust», *Biom. Technol. Today*, vol. 15, n. 3, pagg. 8–9, mar. 2007, doi: 10.1016/S0969-4765(07)70082-6.

[10] T. G. Zimmerman *et al.*, «Retail applications of signature verification», 2004, vol. 5404, pagg. 206–214, doi: 10.1117/12.542747.

[11] Q. Tao e R. Veldhuis, «Biometric authentication system on mobile personal devices», *IEEE Trans. Instrum. Meas.*, vol. 59, n. 4, pagg. 763–773, 2010, doi: 10.1109/TIM.2009.2037873.

[12] «Moving Beyond Passwords Consumer Attitudes on Online Authentication». Consultato: ott. 08, 2020. [In linea]. Available at: https://www.ponemon.org/local/upload/file/NokNokWP_FINAL_3.pdf.

[13] «Identity Theft Tops FTC's Consumer Complaint Categories Again in 2014», *Federal Trade Commission*, feb. 27, 2015. https://www.ftc.gov/news-events/press-releases/2015/02/identity-theft-tops-ftcs-consumer-complaint-categories-again-2014 (consultato ott. 19, 2020).

[14] «2016 Identity Fraud: Fraud Hits an Inflection Point | Javelin». https://www.javelinstrategy.com/coverage-area/2016-identity-fraud-fraud-hits-inflection-point (consultato ott. 19, 2020).

[15] «BlueBorne Information from the Research Team - Armis Labs», *Armis*. https://www.armis.com/blueborne/ (consultato ott. 19, 2020).

[16] «CVE - CVE-2017-13177». https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-13177 (consultato ott. 19, 2020).

[17] «Android Security Bulletin—June 2018», *Android Open Source Project*. https://source.android.com/security/bulletin/2018-06-01 (consultato ott. 19, 2020).
[18] «NVD - CVE-2018-21066». https://nvd.nist.gov/vuln/detail/CVE-2018-21066 (consultato ott. 19, 2020).
[19] *Financial cryptography and data security*. New York, NY: Springer Berlin Heidelberg, 2017.
[20] G.-J. Ahn, Association for Computing Machinery, e Association for Computing Machinery, A c. di, *Proceedings of the 21st ACM Conference on Computer and Communications Security: November 3 - 7, 2014, Scottsdale, Arizona, USA*. New York, NY: ACM, 2014.
[21] I. Stojmenovic, Z. Cheng, e S. Guo, A c. di, *Mobile and Ubiquitous Systems: Computing, Networking, and Services: 10th International Conference, MOBIQUITOUS 2013, Tokyo, Japan, December 2-4, 2013, Revised Selected Papers*, 1st ed. 2014. Cham: Springer International Publishing : Imprint: Springer, 2014.
[22] Y. Fratantonio, C. Qian, S. P. Chung, e W. Lee, «Cloak and Dagger: From Two Permissions to Complete Control of the UI Feedback Loop», in *2017 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, mag. 2017, pagg. 1041–1057, doi: 10.1109/SP.2017.39.
[23] «CERT/CC Vulnerability Note VU#924951». https://www.kb.cert.org (consultato ott. 19, 2020).
[24] «Google_Android_Security_2017_Report_Final.pdf». Consultato: ott. 19, 2020. [In linea]. Available at: https://source.android.com/security/reports/Google_Android_Security_2017_Report_Final.pdf.
[25] «Operating System Market Share Worldwide», *StatCounter Global Stats*. https://gs.statcounter.com/os-market-share (consultato ott. 19, 2020).
[26] «Only play in your comfort zone: interaction methods for improving security awareness on mobile devices | SpringerLink». https://link.springer.com/article/10.1007/s00779-015-0840-5 (consultato ott. 19, 2020).
[27] «Diversity in smartphone usage | Proceedings of the 8th international conference on Mobile systems, applications, and services». https://dl.acm.org/doi/10.1145/1814433.1814453 (consultato ott. 19, 2020).
[28] D. Hintze, P. Hintze, R. D. Findling, e R. Mayrhofer, «A Large-Scale, Long-Term Analysis of Mobile Device Usage Characteristics», *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, n. 2, pagg. 1–21, giu. 2017, doi: 10.1145/3090078.
[29] D. Hintze, R. D. Findling, M. Muaaz, S. Scholz, e R. Mayrhofer, «Diversity in locked and unlocked mobile device usage», in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, New York, NY, USA, set. 2014, pagg. 379–384, doi: 10.1145/2638728.2641697.
[30] D. Hintze, R. D. Findling, S. Scholz, e R. Mayrhofer, «Mobile Device Usage Characteristics: The Effect of Context and Form Factor on Locked and Unlocked Usage», in *Proceedings of the 12th International Conference on Advances in Mobile Computing and Multimedia*, New York, NY, USA, dic. 2014, pagg. 105–114, doi: 10.1145/2684103.2684156.
[31] «Android Compatibility Definition Document», *Android Open Source Project*. https://source.android.com/compatibility/cdd (consultato ott. 19, 2020).
[32] «Better Biometrics in Android P», *Android Developers Blog*. https://android-developers.googleblog.com/2018/06/better-biometrics-in-android-p.html (consultato set. 28, 2020).

[33] «Android 11 Compatibility Definition», *Android Open Source Project*. https://source.android.com/compatibility/android-cdd (consultato ott. 12, 2020).
[34] «Authentication», *Android Open Source Project*. https://source.android.com/security/authentication (consultato ott. 19, 2020).
[35] «Fingerprint HIDL», *Android Open Source Project*. https://source.android.com/security/authentication/fingerprint-hal (consultato ott. 15, 2020).
[36] «LSS2018.pdf». Consultato: ott. 19, 2020. [In linea]. Available at: https://events19.linuxfoundation.org/wp-content/uploads/2017/11/LSS2018.pdf.
[37] «Android- protecting the kernel.pdf». Consultato: ott. 19, 2020. [In linea]. Available at: https://events.static.linuxfound.org/sites/events/files/slides/Android-%20protecting%20the%20kernel.pdf.
[38] «Trusted computing base», *Wikipedia*. feb. 09, 2020, Consultato: ott. 15, 2020. [In linea]. Available at: https://en.wikipedia.org/w/index.php?title=Trusted_computing_base&oldid=939971956.
[39] «Can we make operating systems reliable and secure? - IEEE Journals & Magazine». https://ieeexplore.ieee.org/document/1631939 (consultato ott. 19, 2020).
[40] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, e J. M. McCune, «Trustworthy Execution on Mobile Devices: What Security Properties Can My Mobile Platform Give Me?», in *Trust and Trustworthy Computing*, vol. 7344, S. Katzenbeisser, E. Weippl, L. J. Camp, M. Volkamer, M. Reiter, e X. Zhang, A c. di Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pagg. 159–178.
[41] S. Pinto e N. Santos, «Demystifying Arm TrustZone: A Comprehensive Survey», *ACM Comput. Surv.*, vol. 51, n. 6, pagg. 1–36, feb. 2019, doi: 10.1145/3291047.
[42] J. Winter, «Trusted computing building blocks for embedded linux-based ARM trustzone platforms», in *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, New York, NY, USA, ott. 2008, pagg. 21–30, doi: 10.1145/1456455.1456460.
[43] J. Winter, «Experimenting with ARM TrustZone – Or: How I Met Friendly Piece of Trusted Hardware», in *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, giu. 2012, pagg. 1161–1166, doi: 10.1109/TrustCom.2012.157.
[44] «SamsungKnoxSecuritySolution.pdf». Consultato: ott. 19, 2020. [In linea]. Available at: https://images.samsung.com/is/content/samsung/p5/global/business/mobile/SamsungKnoxSecuritySolution.pdf.
[45] «Android keystore system | Android Developers». https://developer.android.com/training/articles/keystore (consultato ott. 19, 2020).
[46] «Open Portable Trusted Execution Environment», *OP-TEE*. https://www.op-tee.org/ (consultato ott. 19, 2020).
[47] K. Kostiainen, J.-E. Ekberg, N. Asokan, e A. Rantala, «On-board credentials with open provisioning», in *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, New York, NY, USA, mar. 2009, pagg. 104–115, doi: 10.1145/1533057.1533074.
[48] H. Sun, K. Sun, Y. Wang, J. Jing, e H. Wang, «TrustICE: Hardware-Assisted Isolated Computing Environments on Mobile Devices», in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, giu. 2015, pagg. 367–378, doi: 10.1109/DSN.2015.11.
[49] A. Fitzek, F. Achleitner, J. Winter, e D. Hein, «The ANDIX research OS — ARM TrustZone meets industrial control systems security», in *2015 IEEE 13th International*

*Conference on Industrial Informatics (INDIN)*, lug. 2015, pagg. 88–93, doi: 10.1109/INDIN.2015.7281715.

[50] S. Pinto, T. Gomes, J. Pereira, J. Cabral, e A. Tavares, «IIoTEED: An Enhanced, Trusted Execution Environment for Industrial IoT Edge Devices», *IEEE Internet Comput.*, vol. 21, n. 1, pagg. 40–47, gen. 2017, doi: 10.1109/MIC.2017.17.

[51] S. W. Kim, C. Lee, M. Jeon, H. Y. Kwon, H. W. Lee, e C. Yoo, «Secure device access for automotive software», in *2013 International Conference on Connected Vehicles and Expo (ICCVE)*, dic. 2013, pagg. 177–181, doi: 10.1109/ICCVE.2013.6799789.

[52] R. Liu e M. Srivastava, «PROTC: PROTeCting Drone's Peripherals through ARM TrustZone», in *Proceedings of the 3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, New York, NY, USA, giu. 2017, pagg. 1–6, doi: 10.1145/3086439.3086443.

[53] P. Sparks, «The route to a trillion devices», pag. 14, 2017.

[54] N. Santos, H. Raj, S. Saroiu, e A. Wolman, «Using ARM trustzone to build a trusted language runtime for mobile applications», *ACM SIGARCH Comput. Archit. News*, vol. 42, n. 1, pagg. 67–80, feb. 2014, doi: 10.1145/2654822.2541949.

[55] «KeyGenParameterSpec», *Android Developers*. https://developer.android.com/reference/android/security/keystore/KeyGenParameterSpec (consultato ott. 19, 2020).

[56] J. A. Halderman *et al.*, «Lest we remember: cold-boot attacks on encryption keys», *Commun. ACM*, vol. 52, n. 5, pagg. 91–98, mag. 2009, doi: 10.1145/1506409.1506429.

[57] P. Kocher *et al.*, «Spectre Attacks: Exploiting Speculative Execution», *ArXiv180101203 Cs*, gen. 2018, Consultato: ott. 19, 2020. [In linea]. Available at: http://arxiv.org/abs/1801.01203.

[58] M. Lipp *et al.*, «Meltdown», *ArXiv180101207 Cs*, gen. 2018, Consultato: ott. 19, 2020. [In linea]. Available at: http://arxiv.org/abs/1801.01207.

[59] V. van der Veen *et al.*, «Drammer: Deterministic Rowhammer Attacks on Mobile Platforms», in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, ott. 2016, pagg. 1675–1689, doi: 10.1145/2976749.2978406.

[60] A. Tang, S. Sethumadhavan, e S. Stolfo, «{CLKSCREW}: Exposing the Perils of Security-Oblivious Energy Management», 2017, pagg. 1057–1074, Consultato: ott. 19, 2020. [In linea]. Available at: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang.