A UC analysis of Android Protected Confirmation

Maïwenn Racouchot, (joint work with M.Arapinis, T.Zacharias and D.Robin)
November 14, 2023
Context
Android Protected Confirmation: Use case

Transfer $100 to Bob

ROS
Android Protected Confirmation: Use case

Transfer $500 to Eve
## Trusted Execution Environment

- secure area of the main processor
- can isolate code and data in memory
- protect integrity and confidentiality of what is stored inside

Limit: some application might benefit from the functionalities of the TEE but don’t have code in it.
Secured channel between the TEE and the user:
Model of the phone

[Diagram showing a model of a phone with TEE, ROS, Key Store, App 1, App 2, ..., App n, mk, msk, and mvk.]
Overview of the protocol: participants
Overview of the protocol: case of use

Transfer $100 to Bob

ROS
Overview of the protocol: case of use

Transfer $100 to Bob

ROS
Challenge request
Overview of the protocol: case of use

Do you validate “Transfer $100 to Bob”?
Overview of the protocol: case of use
Overview of the protocol: case of use

Challenge

ROS

Challenge
Overview of the protocol: case of use
Overview of the protocol: case of use
Protocol presentation
No RFC or detailed specification of the protocol. Information scattered over different pages.
Retrieving information on the protocol [Dan18]

No RFC or detailed specification of the protocol. Information scattered over different pages.
Overview of the protocol

Protocol in three phases:

- Setup phase: certification of the TEE and the server, installation of applications on the phone
- Registration phase: generation of keys for a given application and registration on the server
- Transaction phase: verification of data by the user and transaction with the server
Setup phase
Setup phase

Alice

Alice ROS

Alice TEE

Google

Server

$$(sk_{\text{Google}}, vk_{\text{Google}}) = \text{KeyGen}(1^\lambda)$$

FetchVerificationKey

VerificationKey, $vk_{\text{Google}}$

Activate

$$(msk, mvk) = \text{KeyGen}(1^\lambda)$$

$mk = \text{KeyGenSym}(1^\lambda)$$

CertifyTEE, $mvk$

InstallApp, AppID

Install App

CertifyTEE, $mvk$

Certify, $cert_{\text{Google}}$

$cert_{\text{Google}} = \text{Sign}(sk_{\text{Google}}, mvk) || mvk$
Setup phase

\[ \text{Alice ROS} \quad \text{Alice TEE} \quad \text{Google} \quad \text{Server} \]

\[ (sk_{\text{Google}}, vk_{\text{Google}}) = \text{KeyGen}(1^\lambda) \]

\[ \text{Activate} \]

\[ (msk, mvk) = \text{KeyGen}(1^\lambda) \]
\[ mk = \text{KeyGenSym}(1^\lambda) \]

\[ \text{CertifyTEE}, mvk \]

\[ \text{Certify}, \text{cert}_{\text{Google}} \]

\[ \text{FetchVerificationKey} \]

\[ \text{VerificationKey, } vk_{\text{Google}} \]

\[ \text{CertifyTEE}, \text{mvk} \]

\[ \text{cert}_{\text{Google}} = \text{Sign}(sk_{\text{Google}} \cdot mvk) \]

\[ \text{Certify}, \text{cert}_{\text{Google}} \]

\[ \text{InstallApp, AppID}_i \]

Install App
**Setup phase**

- **Alice ROS**
- **Alice TEE**
- **Google**
- **Server**

1. **Alice**
   - **Setup phase**
     - **Alice ROS**
     - **Alice TEE**
     - **Google**
     - **Server**

2. **Key Generation**
   - \((sk_{Google}, vk_{Google}) = \text{KeyGen}(1^\lambda)\)

3. **Fetch Verification Key**
   - **Verification Key, ** \(vk_{Google}\)

4. **Activate**
   - \((msk, mvk) = \text{KeyGen}(1^\lambda)\)
   - \(mk = \text{KeyGenSym}(1^\lambda)\)

5. **Certify TEE, mvk**
   - **Certify TEE, mvk**

6. **Certify, ** \(cert_{Google}\)
   - **Certify, cert_{Google}**

7. **Install App, AppID**
   - **Install App**

8. **Install App**
   - **Install App**
Setup phase

\[
\begin{align*}
(s_{\text{skGoogle}}, v_{\text{skGoogle}}) &= \text{KeyGen}(1^\lambda) \\
(\text{msk}, \text{mvk}) &= \text{KeyGen}(1^\lambda) \\
\text{mk} &= \text{KeyGenSym}(1^\lambda) \\
\text{cert}_{\text{Google}} &= \text{Sign}(s_{\text{skGoogle}}, \text{mvk}) || \text{mvk} \\
\text{Certify}_{\text{TEE}, \text{mvk}} \\
\text{Certify}, \text{cert}_{\text{Google}}
\end{align*}
\]
Registration phase
Registration phase

- Alice
  - UsrID: $i$
  - TEE: $(msk, mvk)$, $mk$
- Server
  - $vk_{Google}$
- AppID: $j$
- GetChallenge
  - $ch \leftarrow \{0, 1\}^\lambda$
  - $\text{OpenCh} \leftarrow ch$
- Challenge, $ch$
- KeyGen, $PC = \text{True}, ch$
- $(sk, vk) = \text{KeyGen}(\lambda)$
- $cert = \text{Sign}(msk, (vk, ch)) || cert_{Google}$
- $kblob = \text{Encrypt}(mk, (sk, vk, PC = \text{True}))$
- Key, $ch$, $kblob$, $cert$
- FinishRegistration, UsrID: $i$, AppID: $j$, SrvID: $j$
- FinishRegistration, UsrID: $i$, AppID: $j$, SrvID: $j$, $ch$, $vk$, $cert$
- If: $ch \in \text{OpenCh}$
  - Verify($cert$, $vk_{Google}$)
  - Then: Register(UsrID: $i$, $vk$)
Registration phase

\[
\text{UsrID}_i, (msk, mvk), mk \\
\text{Alice TEE} \\
\text{AppID}_j, \text{Alice ROS} \\
\text{vk}_{\text{Google}} \\
\text{Server}
\]
Alice

\((msk, mvk), mk\)

Alice TEE

\((AppID, SrvID, UsrID)\)

Alice ROS

Server

\(vk_{Google}\)

\(ch \leftarrow \{0, 1\}\)

\(OpenCh \leftarrow ch\)

If:

\(ch \in OpenCh\)

Verify(cert, vk_{Google})

Then:

Register(UsrID, vk)
Registration phase

\[
\begin{align*}
\text{Register AppID}_j, \text{SrvID}_j, \text{UsrID}_j & \quad \text{Alice TEE} \\
\text{GetChallenge} & \quad \text{Alice ROS} \\
\text{FinishRegistration, UsrID}_i, \text{AppID}_j, \text{SrvID}_j, \text{ch}, \text{vk}, \text{cert} & \quad \text{Server}
\end{align*}
\]

\[
\begin{align*}
\text{ch} & \leftarrow \{0, 1\}^\lambda \\
\text{GenKey}_{PC} & = \text{True, ch} \\
(\text{sk}, \text{vk}) & = \text{KeyGen}(1^\lambda) \\
\text{cert} & = \text{Sign}(\text{msk}, (\text{vk}, \text{ch}))) || \text{cert}_{\text{Google}} \\
\text{kblob} & = \text{Encrypt}(\text{mk}, (\text{sk}, \text{vk}, \text{PC} = \text{True}))
\end{align*}
\]

\[
\begin{align*}
\text{Key, ch, kblob, cert} & \quad \text{Alice ROS} \\
\text{If} : & \quad \text{ch} \in \text{OpenCh} \\
\text{Verify(cert, vk}_{\text{Google}}) & \quad \text{Then} : \\
\text{Register(UsrID}_i, \text{vk}) & \quad \text{Alice TEE}
\end{align*}
\]
Registration phase

Alice

UsrID

i

Alice TEE

(\(msk, mvk\), \(mk\))

Server

AppID

ji

Register, AppID, SrvID, UsrID

GetChallenge

\(ch \leftarrow \{0, 1\}^\lambda\)

\(OpenCh \leftarrow ch\)

Challenge, ch

GenKey, PC = True, ch

\((sk, vk) = \text{KeyGen}(1^\lambda)\)

\(cert = \text{Sign}(msk, (vk, ch)) || \text{cert}_{Google}\)

\(kblob = \text{Encrypt}(mk, (sk, vk, PC = True))\)

Key, ch, kblob, cert

FinishRegistration, UsrID, AppID, SrvID, ch, vk, cert

If:

\(ch \in OpenCh\)

Verify(cert, vk_{Google})

Then:

Register(UsrID, vk)
Transaction phase
Transaction phase

Alice TEE

Alice

Alice ROS

Server

Transaction, data, AppID, SrvID

AskNonce

n ← \{0, 1\}^\lambda

OpenN ← n

AskConfirmation, data, n

n

Nonce, n

AskConfirmation, data, n

Confirmed, data

Confirmed, data

Accept, data

ConfirmedData, data, n

Sign, data, n, kblob

PendingRequest ← (data, n)

RetrieveKey(kblob, mk)

Verify PC = True

Verify \{(data, ) \in \text{PendingRequest}

\sigma = \text{Sign}(sk, data|n)

SignData, \sigma

FinishTransaction, \text{UsrID}, \sigma, data, n

If:

Verify(vk, \sigma)

Test n ∈ \text{OpenN}

Then:

Realize transaction
Transaction phase

- Alice
  - Alice TEE
    - Transaction: AppID, SrvID
    - AskNonce
      - n ← \{0, 1\}^\lambda
    - Nonce, n
    - OpenN ← n
    - RetrieveKey(kblob, mk)
    - Verify PC = True
    - Verify \{data, n\} ∈ PendingRequest
    - σ = Sign(sk, data|n)
    - SignedData, σ
    - FinishTransaction, UsrID, σ, data, n
    - If:
      - Verify(vk, σ)
      - Test n ∈ OpenN
      - Then:
        - Realize transaction
  -Confirmed, data
  - Accept, data
  -PendingRequest ← (data, n)
  -ConfirmedData, data, n
  -Sign, data, n, kblob

- Alice ROS
  - App, kblob

- Server
  - vk_{Google}, (Alice, vk)
  - AskNonce
  - n ← \{0, 1\}^\lambda
  - OpenN ← n
Transaction phase

Transaction phase
**Transaction phase**

\[(\text{msk}, \text{mvk}), \text{mk}\]

\[\text{Alice TEE} \rightarrow \text{Alice ROS} \rightarrow \text{Server}\]

\[\text{Transaction}, \text{data}, \text{AppID}_i, \text{SrvID}_j\]

\[\text{AskNonce} \rightarrow n \leftarrow \{0, 1\}^\lambda \rightarrow \text{OpenN} \leftarrow n\]

\[\text{AskConfirmation}, \text{data}, n\]

\[\text{Confirm}, \text{data}\]

\[\text{Accept}, \text{data}\]

\[\text{PendingRequest} \leftarrow (\text{data}, n)\]

\[\text{RetrieveKey}(\text{kblob}, \text{mk})\]

\[\text{Verify PC} = \text{True}\]

\[\text{Verify} \{\text{data}, n\} \in \text{PendingRequest}\]

\[\sigma = \text{Sign}(sk, \text{data}|n)\]

\[\text{SignedData}, \sigma\]

\[\text{FinishTransaction}, \text{UsrID}_i, \sigma, \text{data}, n\]

\[\text{If :}\]

\[\text{Verify}(vk, \sigma)\]

\[\text{Test } n \in \text{OpenN}\]

\[\text{Then :}\]

\[\text{Realize transaction}\]
Transaction phase

\[ (\text{msk}, \text{mvk}), \text{mk} \]

\[ \text{Alice TEE} \]

\[ \text{Alice ROS} \]

\[ \text{App, kblob} \]

\[ \text{Server} \]

\[ \text{Alice} \]

\[ \text{Server} \]

Transaction, \[ \text{data, AppID, SrvID} \]

AskNonce

\[ n \leftarrow \{0, 1\}^\lambda \]

OpenN \[ \leftarrow n \]

AskConfirmation, \[ \text{data, n} \]

CONFIRMED, \[ \text{data} \]

CONFIRMEDData, \[ \text{data, n} \]

Accept, \[ \text{data} \]

Signing, \[ \text{data, n, kblob} \]

PendingRequest \[ \leftarrow (\text{data, n}) \]

RetrieveKey(kblob, \text{mk})

Verify PC = True

Verify \{\text{data, n}\} \in \text{PendingRequest}

\( \sigma = \text{Sign}(\text{sk}, \text{data|n}) \)

SIGNEDData, \[ \sigma \]

CONFIRMEDData, \[ \text{data, n} \]

SignedData, \[ \sigma \]

FinishTransaction, \[ \text{UsrID, } \sigma, \text{data, n} \]

If:

Verify(\text{vk, } \sigma)

Test \( n \in \text{OpenN} \)

Then:

Realize transaction
Transaction phase

Alice TEE

Transaction, data, AppID, SrvID

Alice ROS

AskNonce

Server

n ← \{0, 1\}^\lambda

OpenN ← n

Alice

VK_{Alice}, (Alice, vk)

RetrieveKey(kblob, mk)

Verify PC = True

Verify \{data, n\} ∈ PendingRequest

\sigma = \text{Sign}(sk, data|n)

FinishTransaction, UsrID, data, n

If:

Verify(vk, \sigma)

Test n ∈ OpenN

Then:

Realize transaction
Security analysis
“When using this workflow, your app displays a prompt to the user, asking them to approve a short statement that reaffirms their intent to complete the sensitive transaction. If the user accepts the statement, your app can use a key from Android Keystore to sign the message shown in the dialog. The signature indicates, with very high confidence, that the user has seen the statement and has agreed to it.” [And]
Claim of the protocol

Server accepts transaction \( \Rightarrow \) user has validated the transaction.
Threat model: participants

- Alice: honnest (if not the protocol has no claim)
- TEE: honnest (hypothesis of the protocol)
- ROS: honnest until corrupted
- Server: honnest (if corrupted can perform any transaction anyway)
- Google: honnest (at least as a certification authority)
Threat model: Channels

Google

TEE

ROS

Server

User

Untappable channel
Authenticated channel
Threat model: Channels

Google

TEE

ROS

Server

User

--- Untappable channel

--- Authenticated channel
Transaction replay: attacks and fixes
## Transaction replay attack

### Principle
The user validates the data but does not check the server it is destined to.

### Problem
- The ROS can be corrupted and communicate with any server
- The nonces are not linked to the server (from the TEE perspective)
Transaction replay attack

Transaction, data, AppID, SrvID

Data confirmation: data

Nonce exchange: n1
Nonce exchange: n2

AskConfirmation, data, n2

ConfirmedData, data, n2

Sign, data, n2, kblob2

RetrieveKey (kblob2, mk)
Verify PC = True
Verify(data, .) ∈ PendingRequest
σ = Sign(sk2, data||n2))

SignedData, σ

FinishTransaction, UsrID, σ, data, n2
### Implementation of the attack

#### The target [AAM23]

- APC_Demo_APP developped by the Bern University of Applied Sciences
- Open source Android application, available on GooglePlay

#### The malicious app [Available soon]

- Based on the previous work of David [Rob21]
- Key generation adapted from APC_Demo_APP
Transaction phase fix

Alice TEE

\((\text{msk}, \text{mvk}), \text{mk}\)

Alice ROS

\(\text{App, kblob}\)

Server

\(\text{vk}_{\text{Google}}\) (Alice,\(\text{vk}\))

---

**Transaction, data, AppID, SrvID, j**

**AskConfirmation, data, n, SrvID, j**

**Accept, data, SrvID, j**

**Confirmed \(\leftarrow (\text{data}, n, \text{SrvID, j})\)**

**ConfirmedData, data, n, SrvID, j**

**Sign, data, n, kblob**

\(\sigma = \text{Sign}(\text{sk}, \text{data}||n||\text{SrvID, j})\)

**SignedData, \sigma**

**FinishTransaction, UsrID, i, \sigma, data, n**

---

If:
- Verify(\(\text{vk}, \sigma\))
- Test \(n \in \text{OpenN}\)
- Test SrvID, j = Server
- Then:
  - Realizetransaction
Conclusion
• Fixes have been proved in the UC framework
• Attacks and fixes have been discussed and accepted by Google
• The paper have been submitted to USENIX
Thank you for your attention!

Questions?
Registration phase: summary

1. User initiate the registration
2. ROS fetch a challenge from the server
3. TEE generate a pair of key for the application
4. TEE sign the verification key and the challenge
5. TEE encrypt the application key pair and store it in the KeyStore (ROS)
6. ROS sends the signed message to the server for registration
7. Server verify the challenge and the signature
1. User initiate the transaction
2. ROS fetch a nonce from the server
3. ROS sends transaction data and nonce to TEE for confirmation
4. TEE presents data to be confirmed to the user
5. TEE informs the ROS of the validation
6. ROS retrieves the encrypted app keys from the KeyStore and send them to TEE
7. TEE decipher the keys with its master key
8. TEE uses the signing key of the app to sign the data and nonce
9. TEE sends the signed message to the ROS
10. ROS send the signed data to the server
11. Server verify the signature and the nonce and realizes the transaction
“Once confirmed, your intention is cryptographically authenticated and unforgeable when conveyed to the relying party, for example, your bank. Protected Confirmation increases the bank’s confidence that it acts on your behalf, providing a higher level of protection for the transaction.” [Dan18]
Impersonation at registration: attacks and fixes
Impersonation at registration

**Principle**
Duplication of the registration phase and Meet in the Middle

4. TEE sign the verification key and the challenge
5. TEE encrypt the application key pair and store it in the KeyStore (ROS)
6. ROS sends the signed message to the server for registration
7. **Server verify the challenge and the signature**

**Problem**
The check verifies that the signature has been made by any TEE.
Impersonation at registration

\[ \text{Alice ROS} : (\text{msk}_A, \text{mvk}_A), \text{mk}_A \]
\[ \text{Alice TEE} : (\text{msk}_A, \text{mvk}_A), \text{mk}_A \]
\[ \text{Eve ROS} : (\text{msk}_E, \text{mvk}_E), \text{mk}_E \]
\[ \text{Eve TEE} : (\text{msk}_E, \text{mvk}_E), \text{mk}_E \]
\[ \text{Server} : \text{vk}_\text{Google} \]

Key generation:
- \[ \text{vk}_A, \text{cert}_A \]
- \[ \text{vk}_E, \text{cert}_E \]

Register(Alice, AppID^i_j, SrvID_j, UsrID_i)

GetChallenge

Challenge, ch

FinishRegistration(UsrID_i, AppID^i_j, SrvID_j, ch, vk_A, cert_A)

If:
- \[ ch \in \text{OpenCh} \]
- Verify(cert_E, vk_{\text{Google}})
Then:
- Register(UsrID_i, vk_E)
Registration phase fix

\[(msk, mvk), mk\]

ALICE TEE

\[Alice \text{ ROS}\]

AppID\textsubscript{j}

Server

\[vk\textsubscript{Google}\]

\[UsrID\textsubscript{i}, \text{AppID}\textsubscript{j}, \text{SrvID}\textsubscript{j}, \text{UsrID}\textsubscript{i}\]

\[\text{Register, AppID}\textsubscript{j}, \text{SrvID}\textsubscript{j}, \text{UsrID}\textsubscript{i}\]

Challenge exchange: \(ch\)

\[\text{GenKey, PC} = \text{True, ch, SrvID}_j\]

\[(sk, vk) = \text{KeyGen(1}\^\lambda)\]

cert = SIGN\(msk, (vk, ch, IMEI))||(vk, ch, IMEI)||\text{cert}_{\text{Google}}\]

\[kblob = \text{Encrypt}(mk, (sk, vk, PC = \text{True}))\]

\[\text{FinishRegistration, UsrID}_i, \text{AppID}_j, \text{SrvID}_j, ch, vk, \text{cert}\]

\[ch \in \text{OpenCh}\]

\[\text{VerifyRegistration, IMEI, vk}\]

\[\text{AcceptRegistration, IMEI, vk}\]

Untappable authentified channel
APCDemo and Anti-Myon. 

Apc_demo_app. 


Android. 

Android protected confirmation. 

Janis Danisevskis.  
**Android protected confirmation: Taking transaction security to the next level.**  

David Robin.  
**Yubidroid.**  