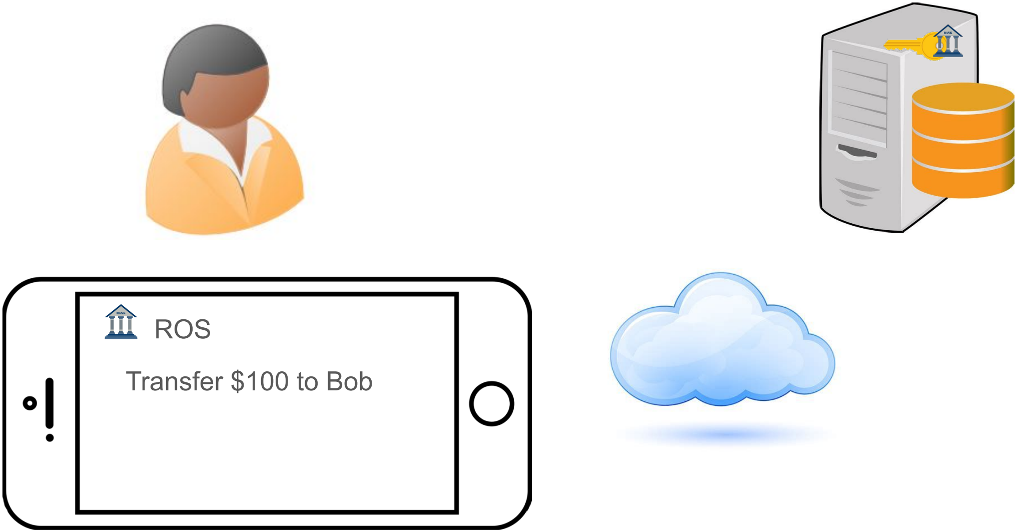
A UC analysis of Android Protected Confirmation

Maïwenn Racouchot, (joint work with M.Arapinis, T.Zacharias and D.Robin)

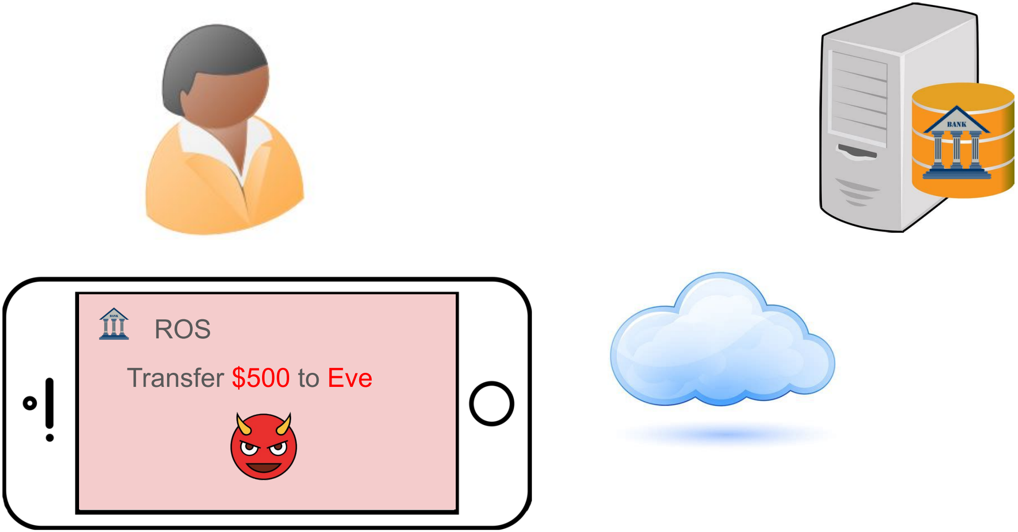
November 14, 2023

Context

Android Protected Confirmation: Use case



Android Protected Confirmation: Use case



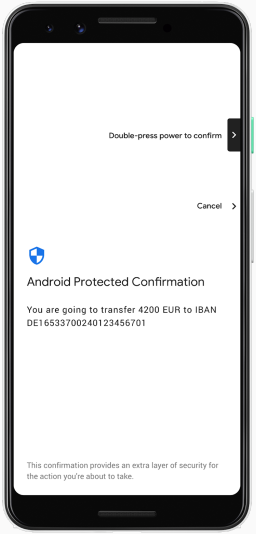
TEE

Trusted Execution Environment

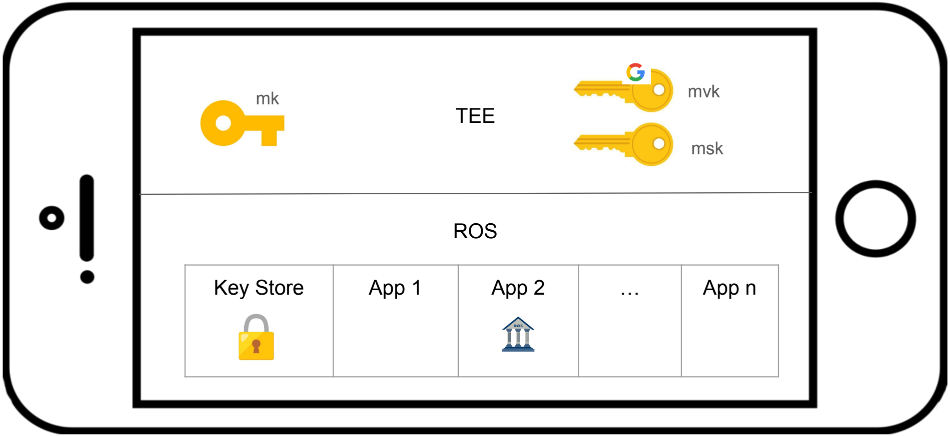
- secure area of the main processor
- can isolate code and data in memory
- protect integrity and confidentiality of what is stored inside

Limit: some application might benefit from the functionnalities of the TEE but don't have code in it.

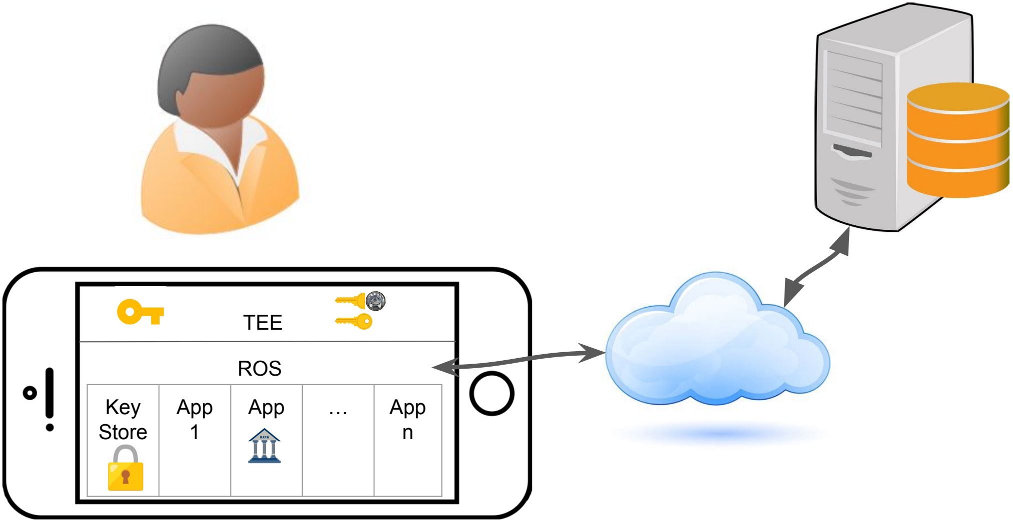
Secured channel between the TEE and the user:



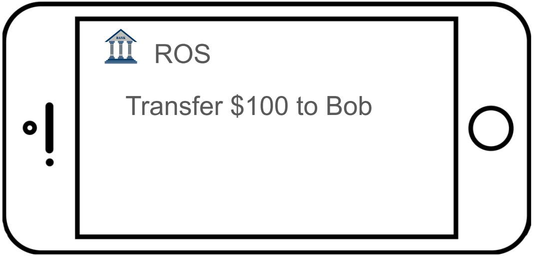
Model of the phone



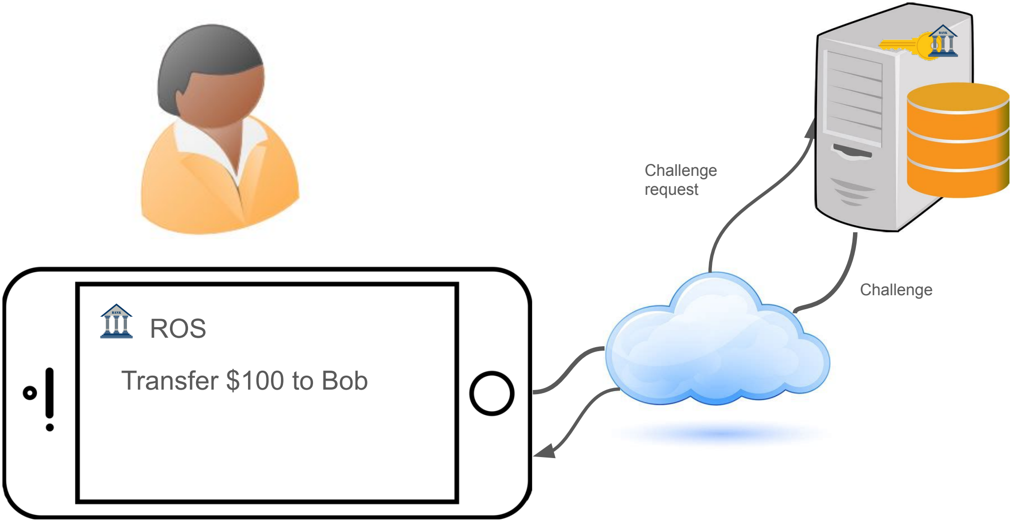
Overview of the protocol: participants



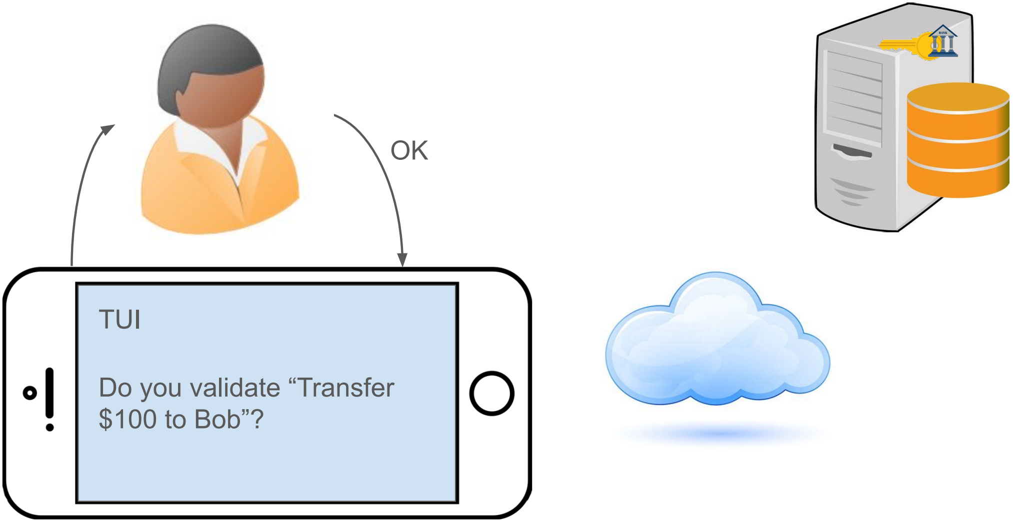
Overview of the protocol: case of use



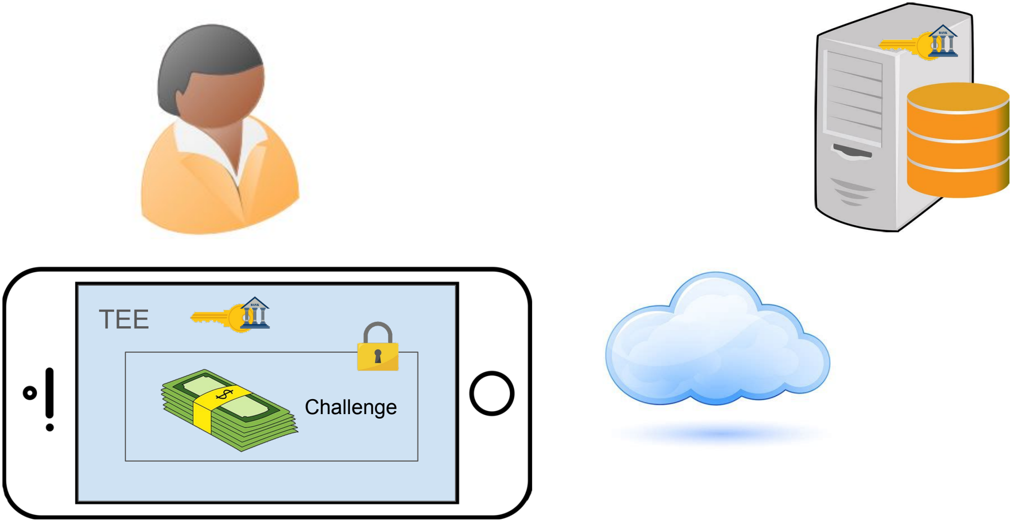
Overview of the protocol: case of use



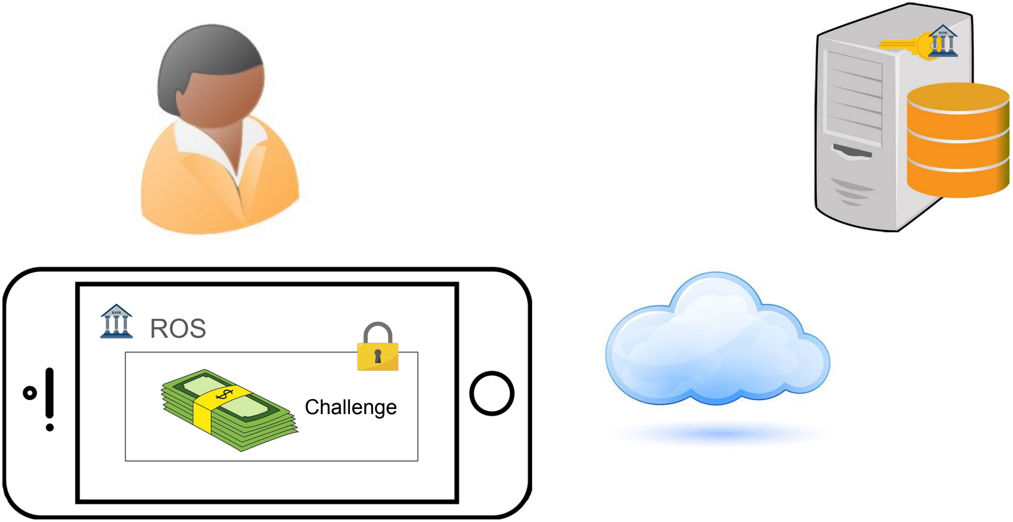
Overview of the protocol: case of use



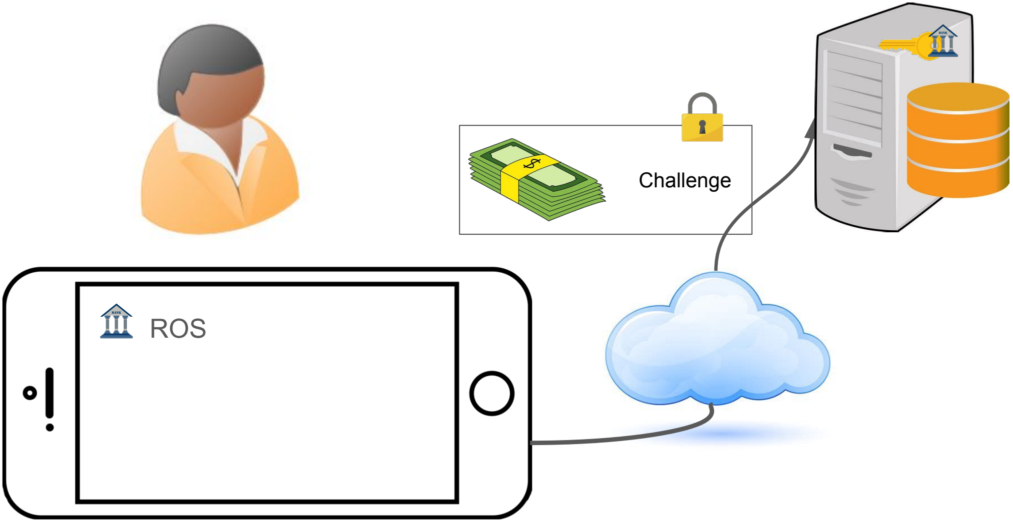
Overview of the protocol: case of use



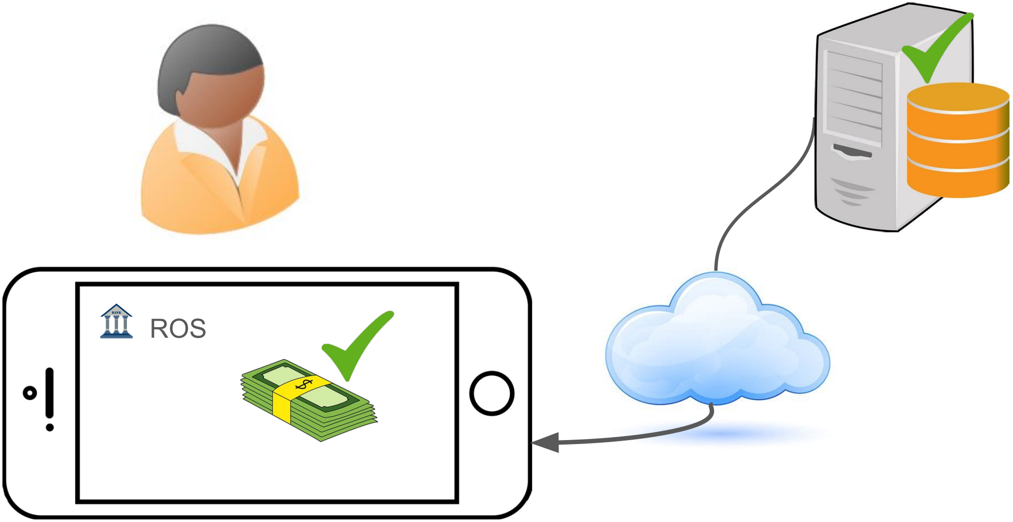
Overview of the protocol: case of use



Overview of the protocol: case of use



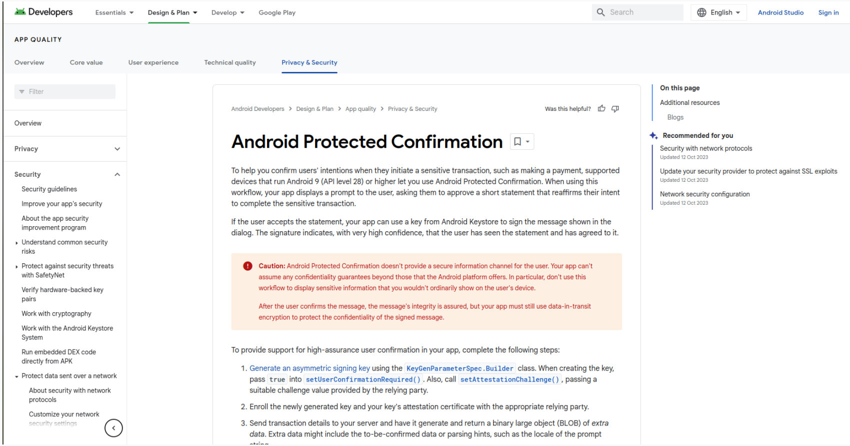
Overview of the protocol: case of use



Protocol presentation

Retrieving information on the protocol [And]

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


Developers Essentials Design & Plan Develop Google Play Search English Android Studio Sign in

APP QUALITY

Overview Core value User experience Technical quality Privacy & Security

Filter

Overview

Privacy

Security

Security guidelines

Improve your app's security

About the app security improvement program

Understand common security risks

Protect against security threats with SafetyNet

Verify hardware-backed key pairs

Work with cryptography

Work with the Android Keystore System

Run embedded DEX code directly from APK

Protect data sent over a network

About security with network protocols

Customize your network security settings

Android Developers > Design & Plan > App quality > Privacy & Security

Was this helpful?

Android Protected Confirmation

To help you confirm users' intentions when they initiate a sensitive transaction, such as making a payment, supported devices that run Android 9 (API level 28) or higher let you use Android Protected Confirmation. 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.

Caution: Android Protected Confirmation doesn't provide a secure information channel for the user. Your app can't assume any confidentiality guarantees beyond those that the Android platform offers. In particular, don't use this workflow to display sensitive information that you wouldn't ordinarily show on the user's device.

After the user confirms the message, the message's integrity is assured, but your app must still use data-in-transit encryption to protect the confidentiality of the signed message.

To provide support for high-assurance user confirmation in your app, complete the following steps:

1. Generate an asymmetric signing key using the `KeyGenParameterSpec.Builder` class. When creating the key, pass `true` into `setUserConfirmationRequired()`. Also, call `setAttestationChallenge()`, passing a suitable challenge value provided by the relying party.
2. Enroll the newly generated key and your key's attestation certificate with the appropriate relying party.
3. Send transaction details to your server and have it generate and return a binary large object (BLOB) of *extra data*. Extra data might include the to-be-confirmed data or parsing hints, such as the locale of the prompt string.

On this page

Additional resources

Blogs

Recommended for you

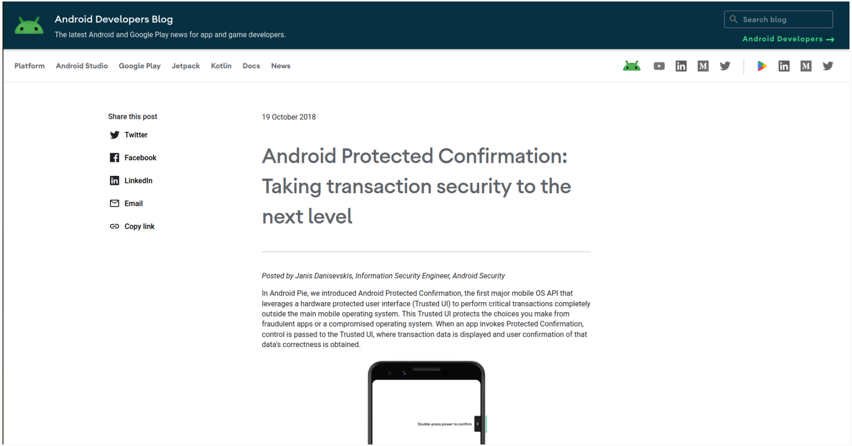
Security with network protocols
Updated 12 Oct 2023

Update your security provider to protect against SSL exploits
Updated 12 Oct 2023

Network security configuration
Updated 12 Oct 2023

Retrieving information on the protocol [Dan18]

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



Android Developers Blog

The latest Android and Google Play news for app and game developers.

Search blog

Android Developers →

Platform Android Studio Google Play Jetpack Kotlin Docs News

Share this post

Twitter

Facebook

LinkedIn

Email

Copy link

19 October 2018

Android Protected Confirmation: Taking transaction security to the next level

Posted by Janis Danisevskis, Information Security Engineer, Android Security

In Android Pie, we introduced Android Protected Confirmation, the first major mobile OS API that leverages a hardware protected user interface (Trusted UI) to perform critical transactions completely outside the main mobile operating system. This Trusted UI protects the choices you make from fraudulent apps or a compromised operating system. When an app invokes Protected Confirmation, control is passed to the Trusted UI, where transaction data is displayed and user confirmation of that data's correctness is obtained.

Double-press power to confirm

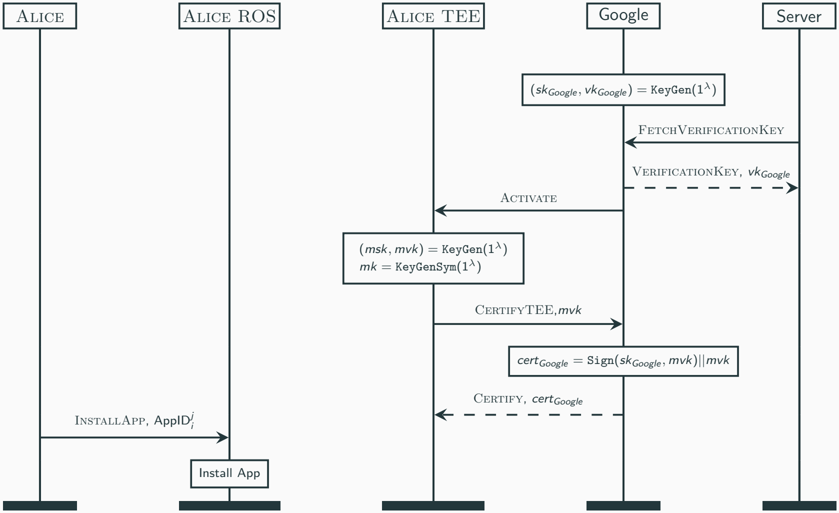
Overview of the protocol

Protocol in three phases:

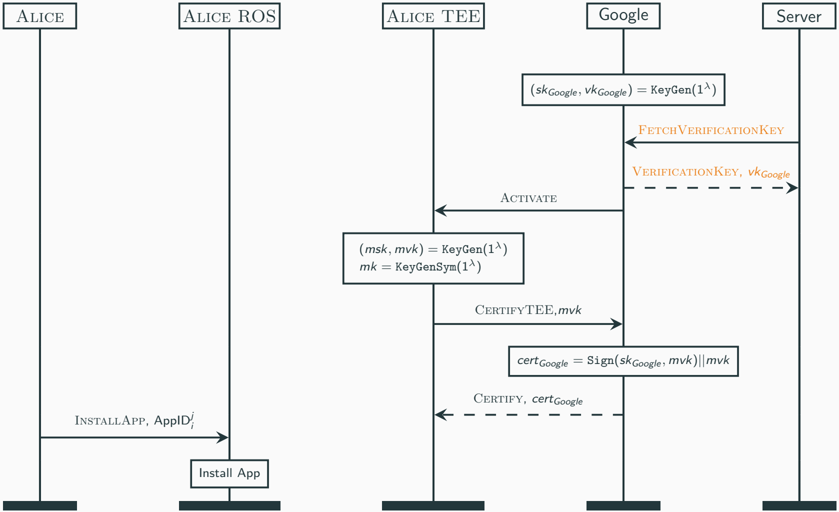
- Setup phase: certification of the TEE and the server, installation of applications on the phone
- Registration phase: generation of keys for a given application and registration on the server
- Transaction phase: verification of data by the user and transaction with the server

Setup phase

Setup phase

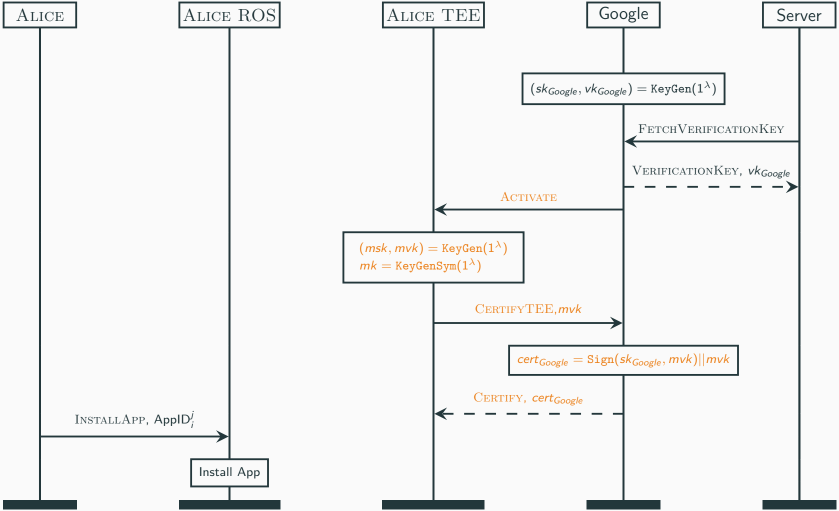

Alice
Alice ROS
Alice TEE
Google
Server
$(sk_{Google}, vk_{Google}) = \texttt{KeyGen}(1^\lambda)$
FetchVerificationKey
VerificationKey, vk_{Google}
Activate
$(msk, mvk) = \texttt{KeyGen}(1^\lambda)$
$mk = \texttt{KeyGenSym}(1^\lambda)$
CertifyTEE, mvk
$cert_{Google} = \texttt{Sign}(sk_{Google}, mvk)||mvk$
Certify, $cert_{Google}$
InstallApp, AppID_i^j
Install App

Setup phase



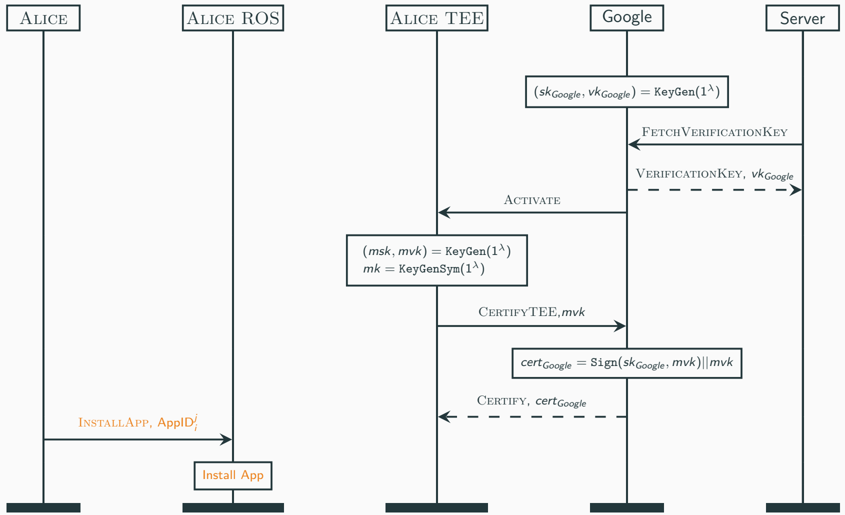
Alice
Alice ROS
Alice TEE
Google
Server
$(sk_{Google}, vk_{Google}) = \texttt{KeyGen}(1^\lambda)$
FetchVerificationKey
VerificationKey, vk_{Google}
Activate
$(msk, mvk) = \texttt{KeyGen}(1^\lambda)$
$mk = \texttt{KeyGenSym}(1^\lambda)$
CertifyTEE, mvk
$cert_{Google} = \texttt{Sign}(sk_{Google}, mvk)||mvk$
Certify, $cert_{Google}$
InstallApp, AppID_i^j
Install App

Setup phase



Alice
Alice ROS
Alice TEE
Google
Server
$(sk_{Google}, vk_{Google}) = \mathtt{KeyGen}(1^\lambda)$
FetchVerificationKey
VerificationKey, vk_{Google}
Activate
$(msk, mvk) = \mathtt{KeyGen}(1^\lambda)$
$mk = \mathtt{KeyGenSym}(1^\lambda)$
CertifyTEE, mvk
$cert_{Google} = \mathtt{Sign}(sk_{Google}, mvk)||mvk$
Certify, $cert_{Google}$
InstallApp, AppID_i^j
Install App

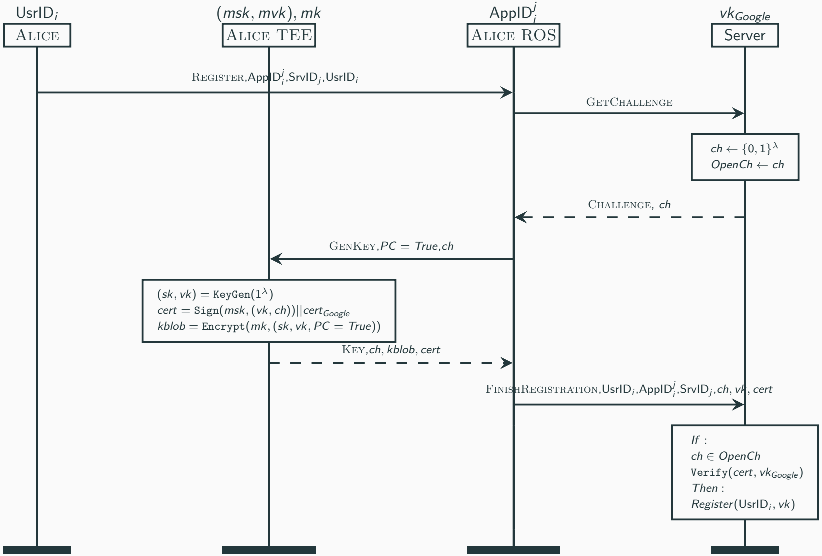
Setup phase



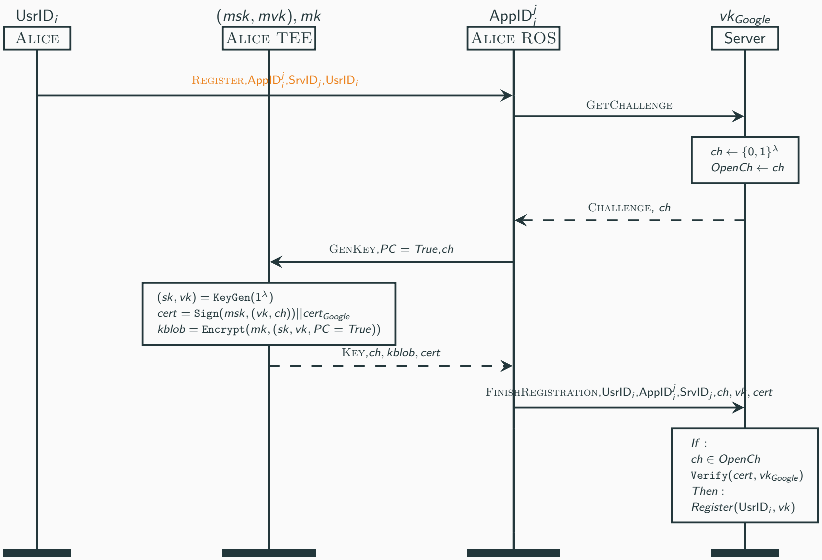
Alice
Alice ROS
Alice TEE
Google
Server
$(sk_{Google}, vk_{Google}) = \mathtt{KeyGen}(1^\lambda)$
FetchVerificationKey
VerificationKey, vk_{Google}
Activate
$(msk, mvk) = \mathtt{KeyGen}(1^\lambda)$
$mk = \mathtt{KeyGenSym}(1^\lambda)$
CertifyTEE, mvk
$cert_{Google} = \mathtt{Sign}(sk_{Google}, mvk)||mvk$
Certify, $cert_{Google}$
InstallApp, AppID_i^j
Install App

Registration phase

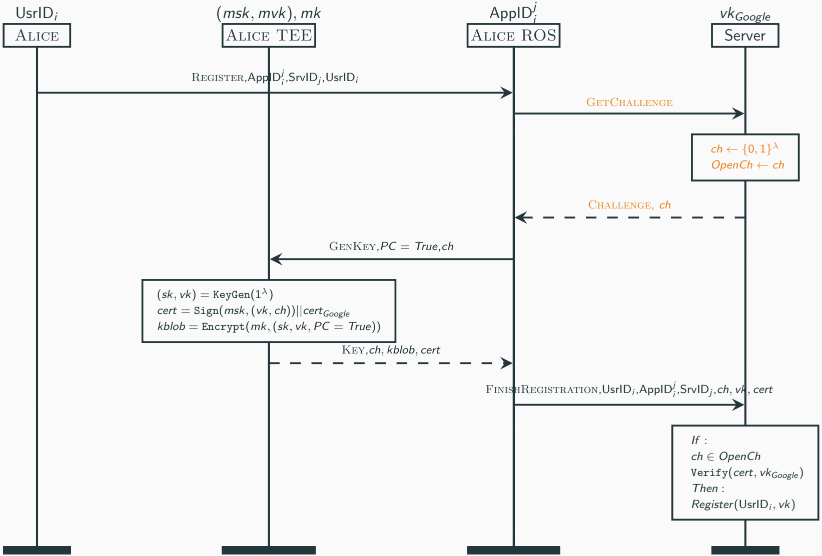
Registration phase



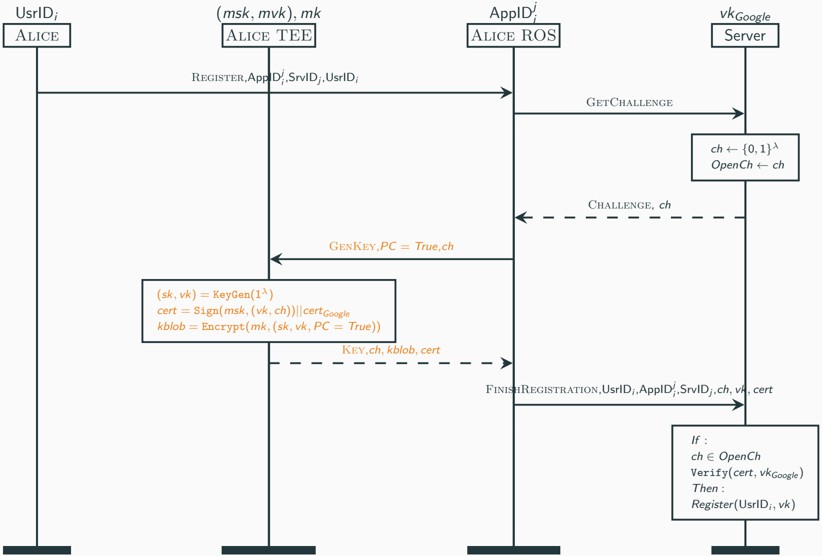
Registration phase



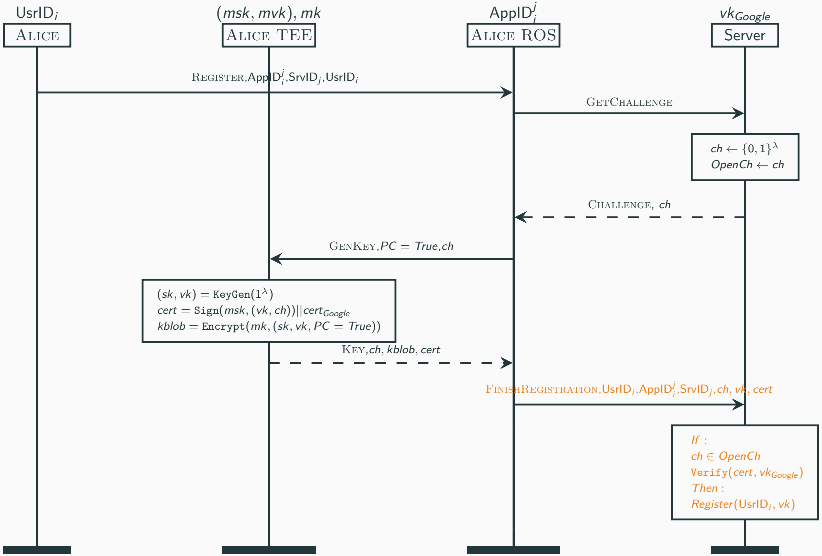
Registration phase



Registration phase

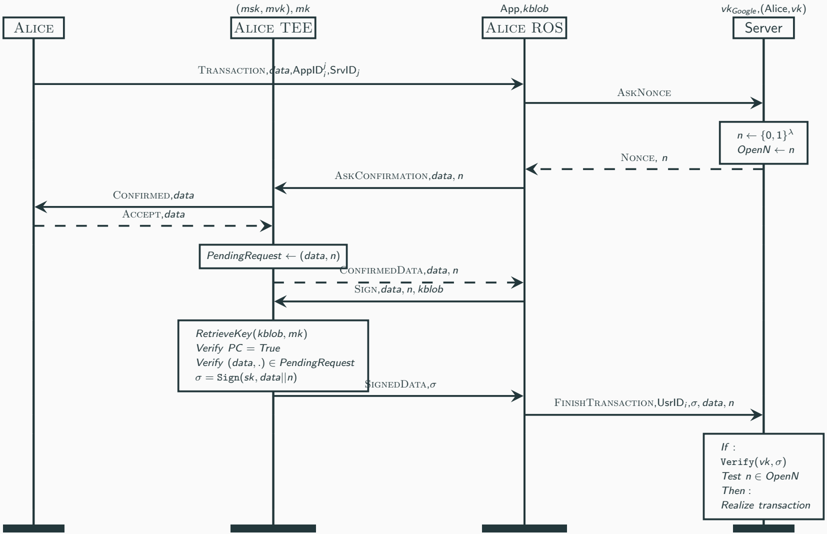


Registration phase

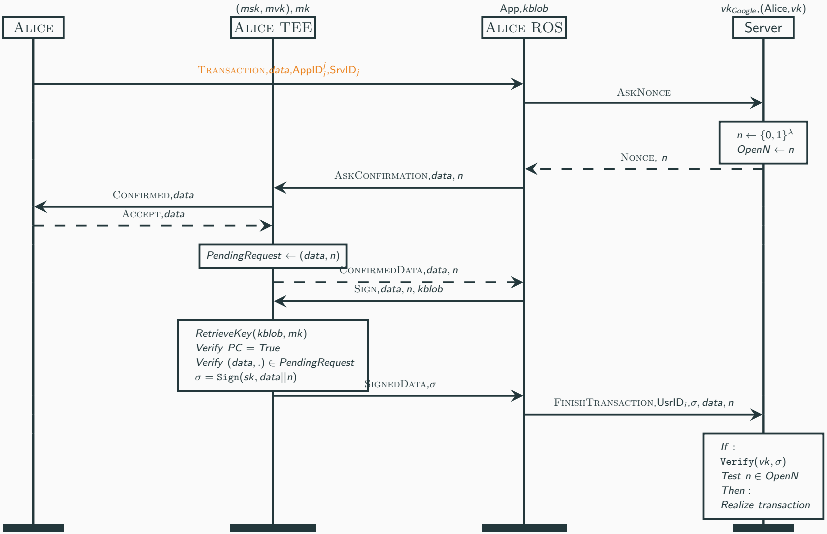


Transaction phase

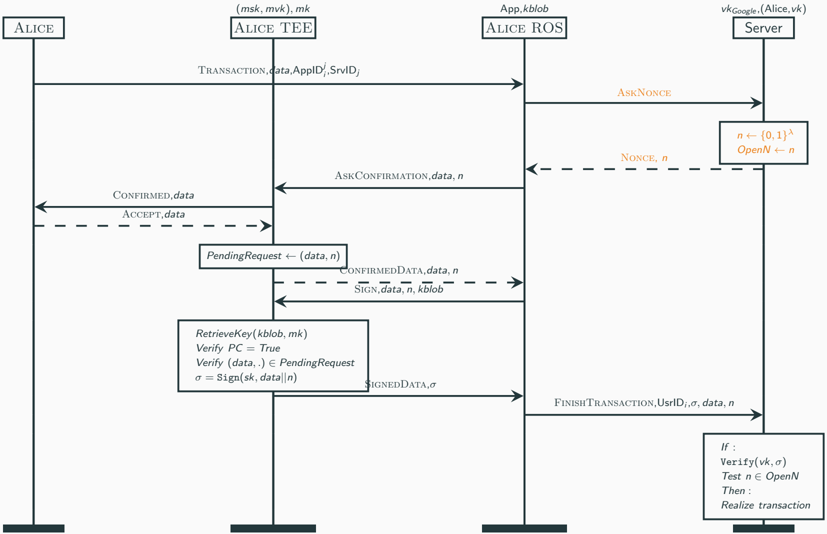
Transaction phase



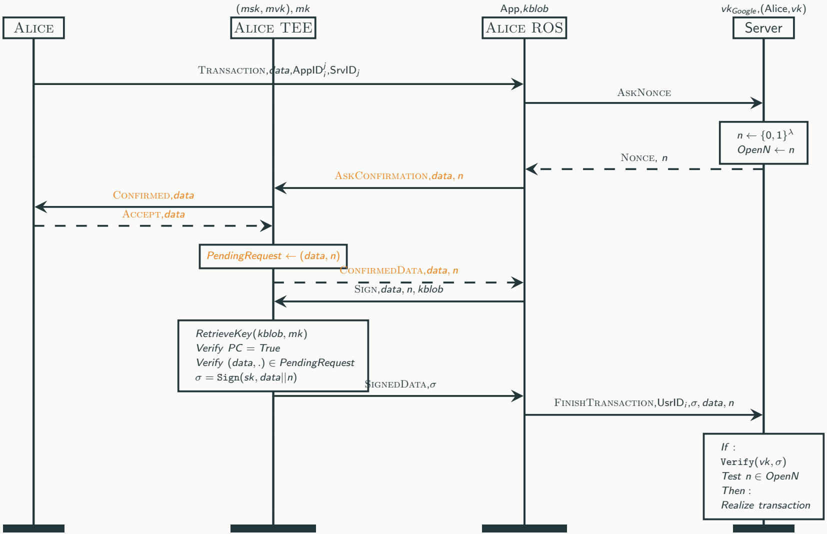
Transaction phase



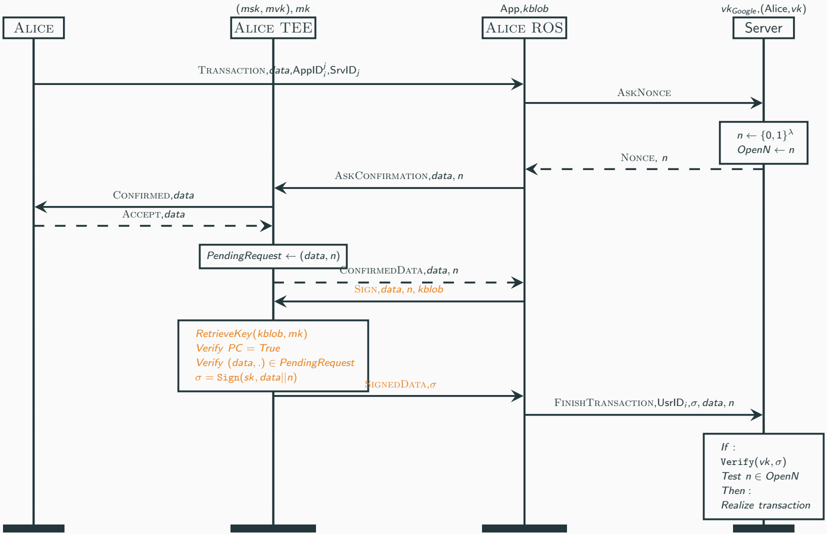
Transaction phase



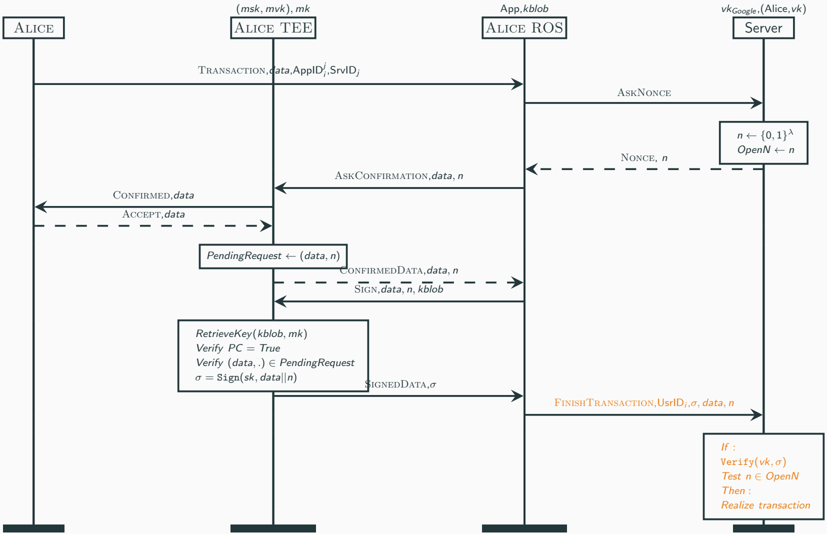
Transaction phase



Transaction phase



Transaction phase



Security analysis

Claim of the protocol

"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]

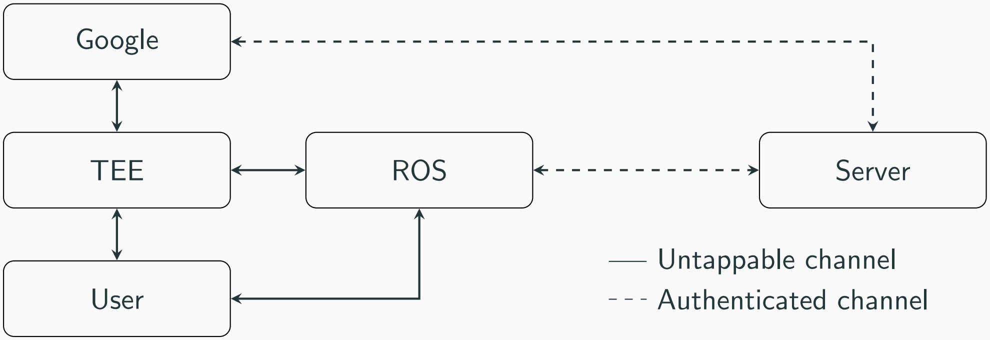
Claim of the protocol

Server accepts transaction $\implies$ user has validated the transaction.

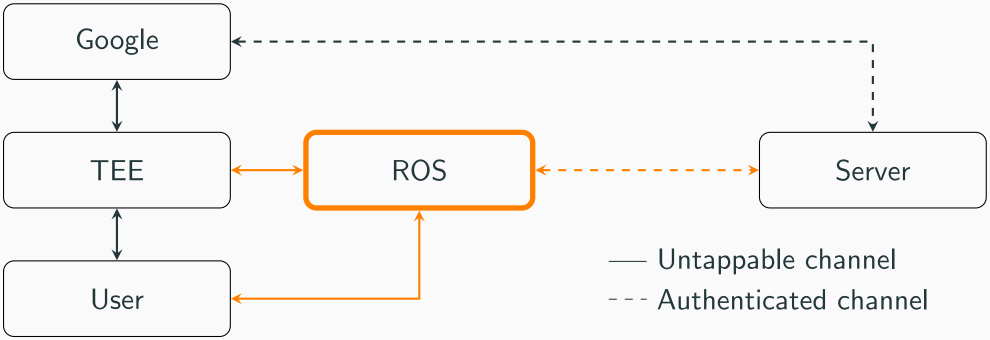
Threat model: participants

- 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



Threat model: Channels



Transaction replay: attacks and fixes

Transaction replay attack

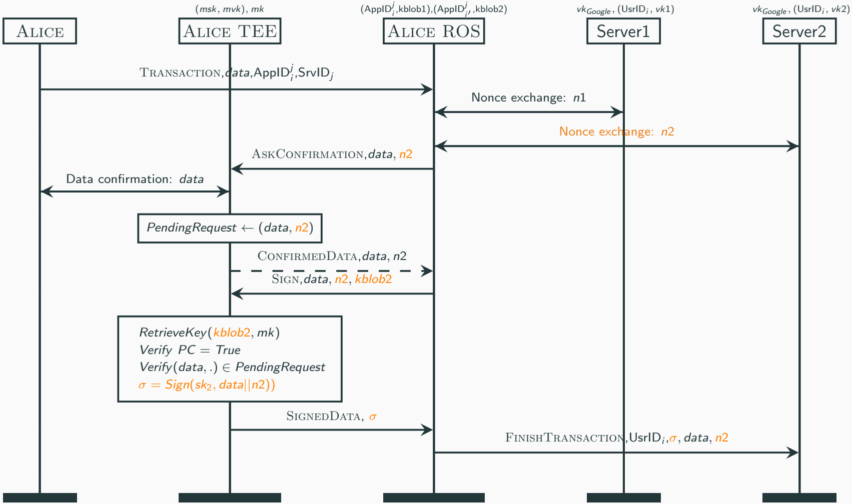
Principle

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



Implementation of the attack

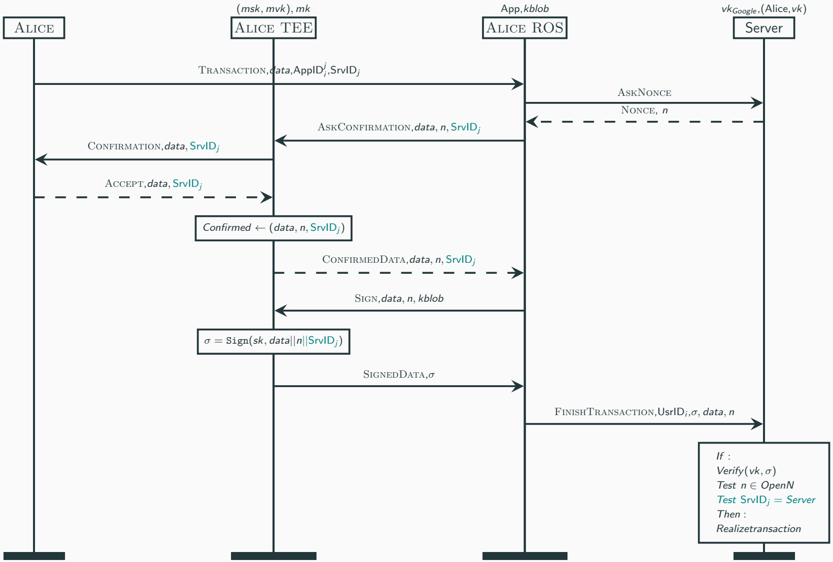
The target [AAM23]

- 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



Conclusion

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

Question?

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

Transaction phase: summary

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

Claim of the protocol

"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

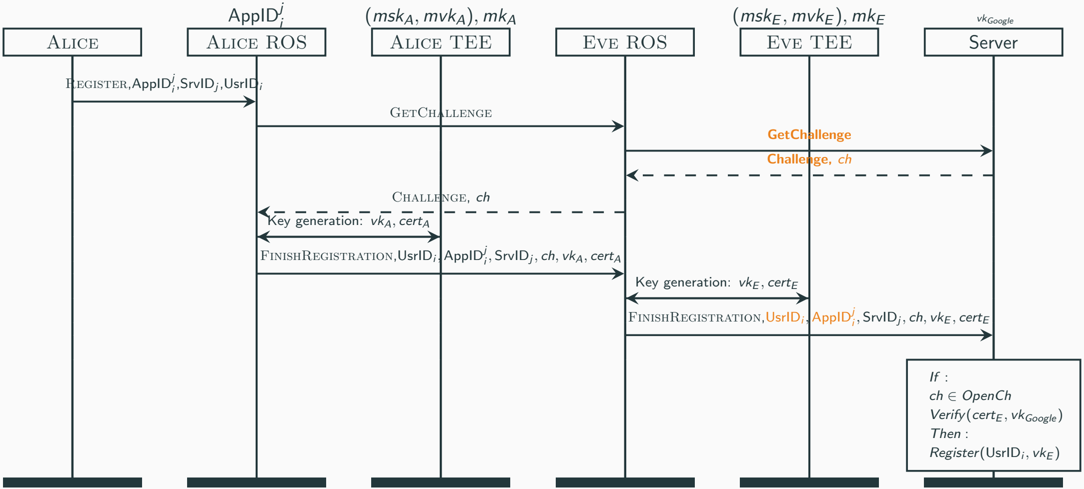
Duplication of the registration phase and Meet in the Middle

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

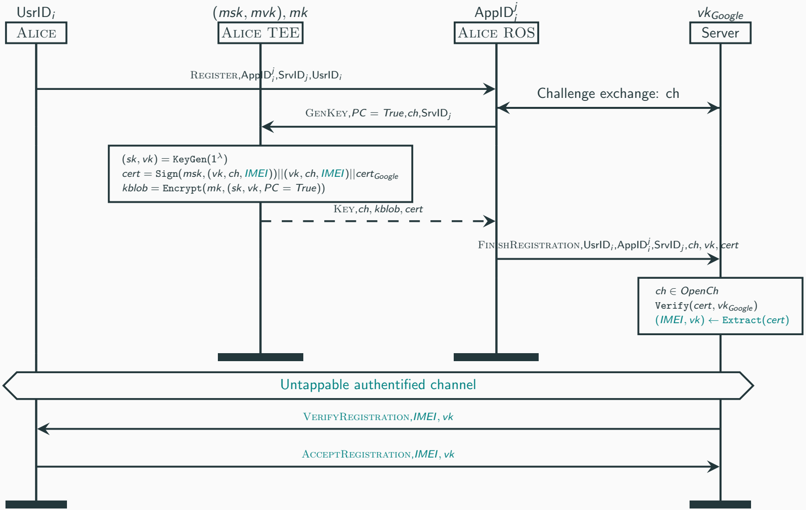
Problem

The check verifies that the signature has been made by any TEE.

Impersonation at registration



Registration phase fix



Bibliography i

APCDemo and Anti-Myon.
Apc_demo_app.
https://github.com/APCDemo/APC_Demo_App, 2023.

Android.
Android protected confirmation.
https://developer.android.com/privacy-and-security/security-android-protected-confirmation.

Bibliography ii

Janis Danisevskis.
Android protected confirmation: Taking transaction security to the next level.
https://android-developers.googleblog.com/2018/10/android-protected-confirmation.html, 2018.

David Robin.
Yubidroid.
https://www.robindar.com/yubidroid/getting-source-code, 2021.