

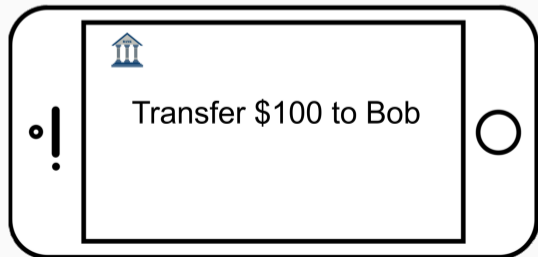
A (UC) analysis of Android Protected Confirmation

Maiwenn Racouchot (joint work with M.Arapinis, V.Danos, D.Robin and T.Zacharias)

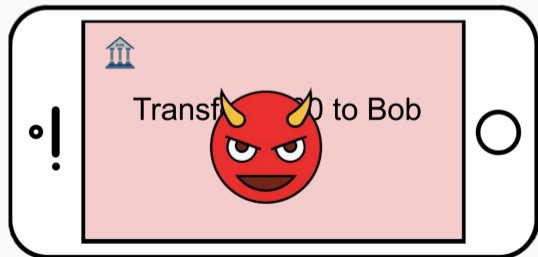
January 17, 2025

Context

Android Protected Confirmation: use case



Android Protected Confirmation: use case



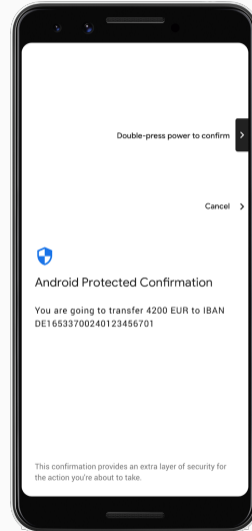
Trusted Execution Environment

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

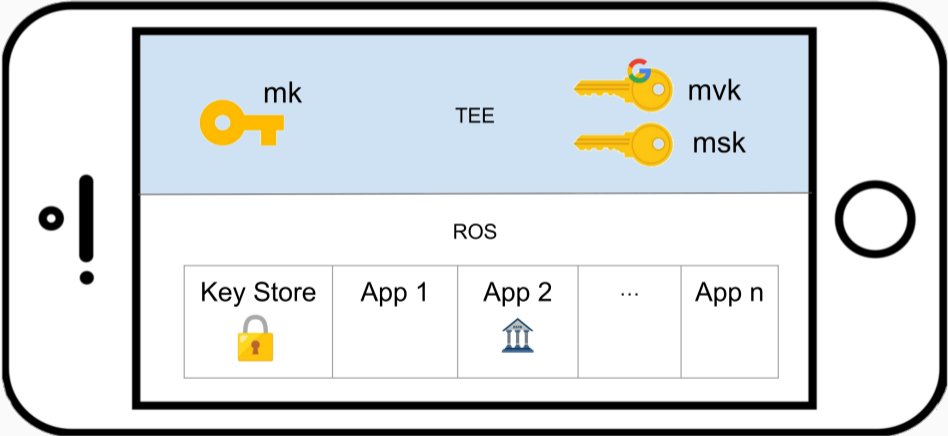
Limitation: some application might benefit from the fonctionnalités of the TEE but don't have code in it.

Trusted User Interface:

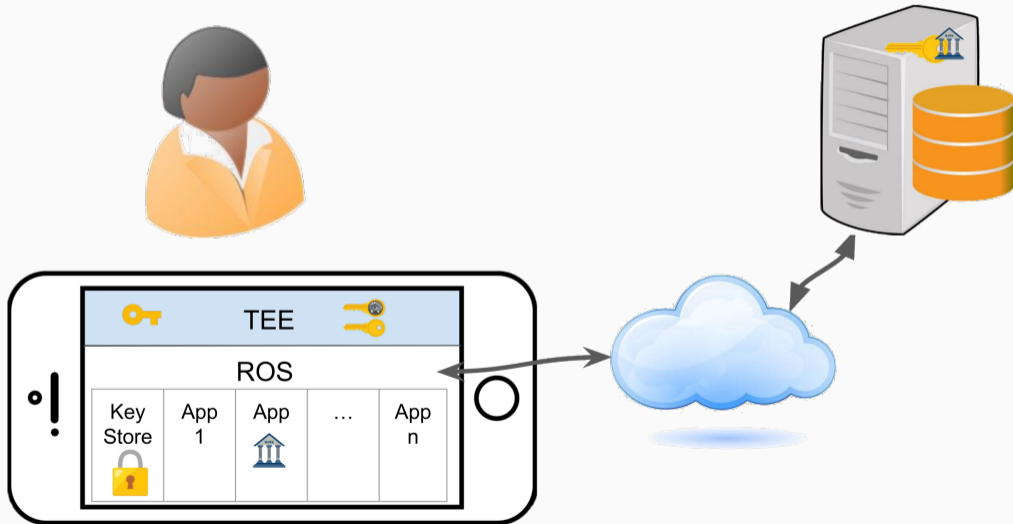
- Secure channel between the TEE and the user
- Untappable by the ROS



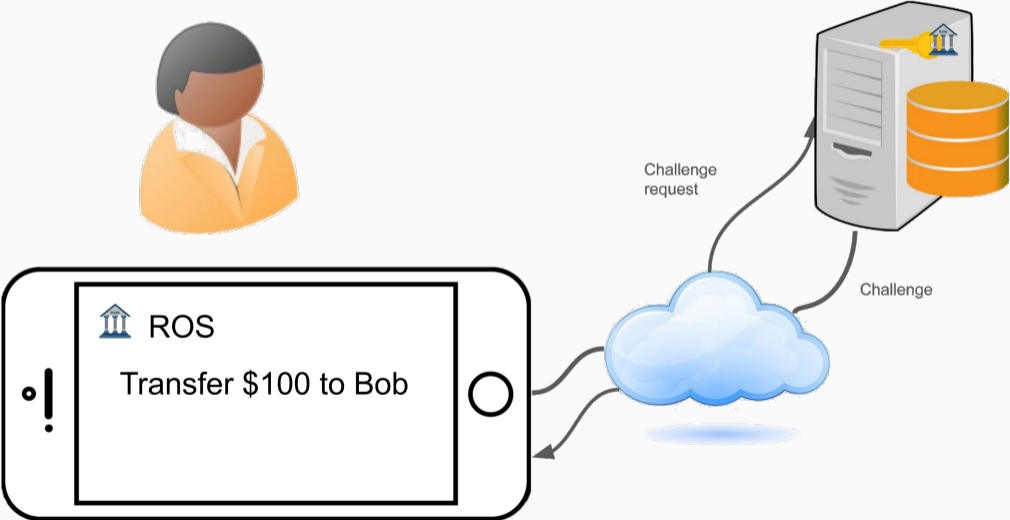
Model of the phone



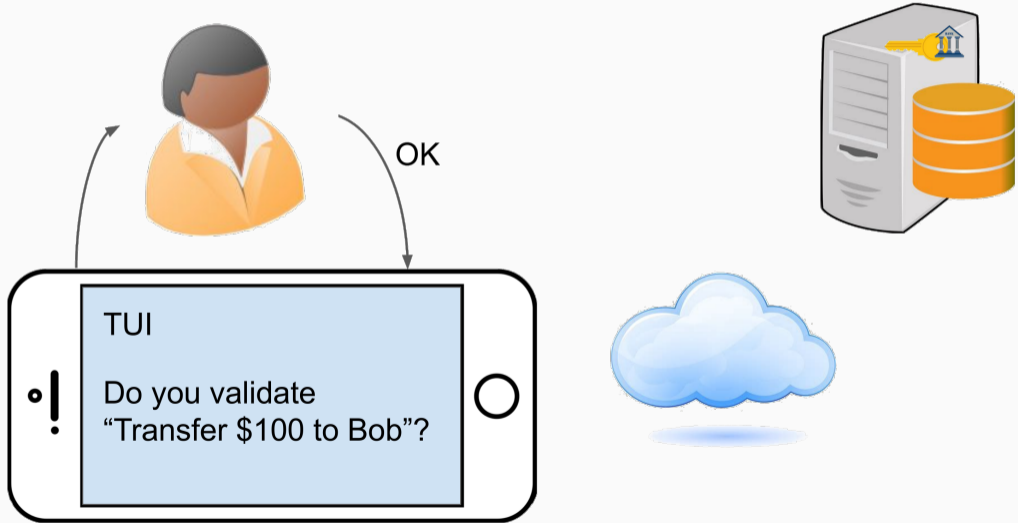
Overview of the APC protocol: participants



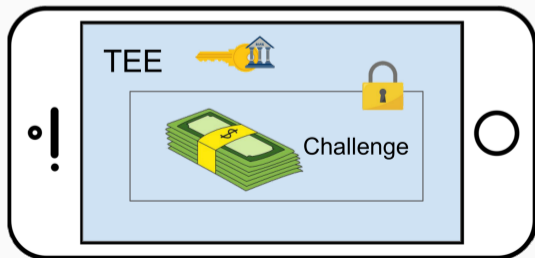
Overview of the APC protocol: use case



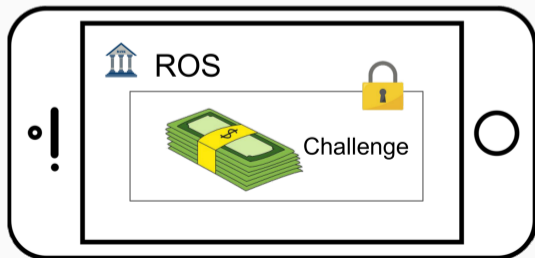
Overview of the APC protocol: use case



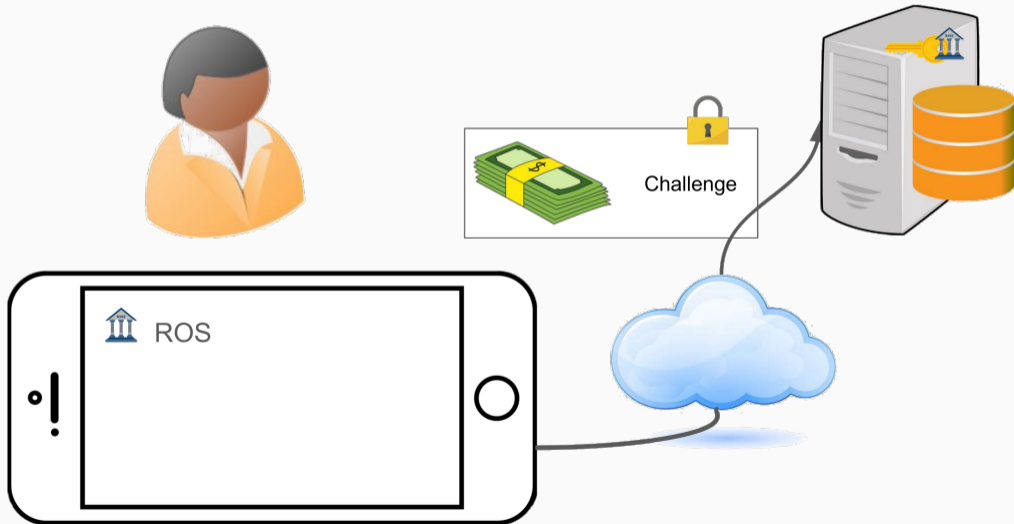
Overview of the APC protocol: use case



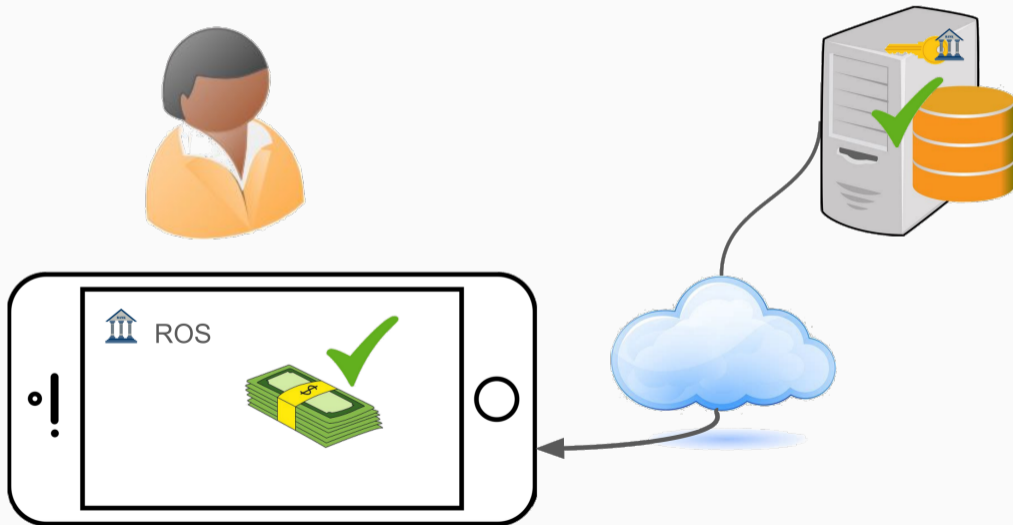
Overview of the APC protocol: use case



Overview of the APC protocol: use case



Overview of the APC protocol: use case



Protocol presentation

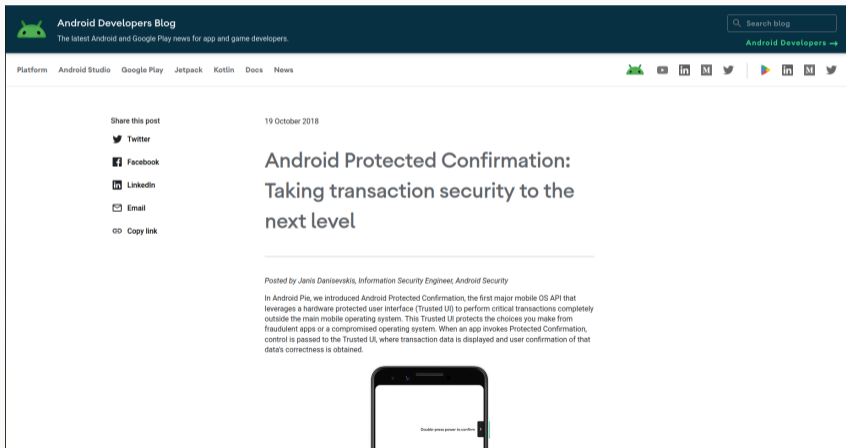
Retrieving information on the protocol [And]

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

The screenshot shows the Android Developers website interface. At the top, there's a navigation bar with 'Developers', 'Essentials', 'Design & Plan', 'Develop', and 'Google Play'. A search bar and language selector ('English') are also present. Below the navigation, there's a section for 'APP QUALITY' with tabs for 'Overview', 'Core value', 'User experience', 'Technical quality', and 'Privacy & Security'. The 'Privacy & Security' tab is active. On the left, a sidebar menu lists various security topics, with 'Security' expanded to show 'Security guidelines', 'Improve your app's security', and 'Understand common security risks'. The main content area features the article 'Android Protected Confirmation'. The article title is followed by a share icon. The text explains that this feature helps confirm users' intentions for sensitive transactions. A prominent red warning box states: '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.' Below the warning, the article lists steps to implement the feature: 1. Generate an asymmetric signing key using the `KeyGenParameterSpec.Builder` class. 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. The right sidebar contains 'On this page', 'Additional resources', and 'Recommended for you' sections.

Retrieving information on the protocol [Dan18]

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



The screenshot shows the Android Developers Blog interface. At the top, there is a dark blue header with the Android logo, the text "Android Developers Blog", and the subtitle "The latest Android and Google Play news for app and game developers." A search bar is located in the top right corner. Below the header is a navigation menu with links for Platform, Android Studio, Google Play, Jetpack, Kotlin, Docs, and News. Social media icons for YouTube, LinkedIn, Medium, and Twitter are also present.

The main content area features a post dated "19 October 2018" with the title "Android Protected Confirmation: Taking transaction security to the next level". To the left of the title are sharing options: Twitter, Facebook, LinkedIn, Email, and Copy link. Below the title, the author is identified as "Posted by Janis Danisevskis, Information Security Engineer, Android Security". The article text begins with "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."

At the bottom of the article, there is a partial view of a smartphone screen displaying the text "Double press power to confirm" next to a green checkmark icon.

Overview of the protocol

Protocol in three phases:

1. **Setup phase:** certification of the TEE, setup of the server, installation of applications on the phone

Overview of the protocol

Protocol in three phases:

1. **Setup phase:** certification of the TEE, setup of the server, installation of applications on the phone
2. **Registration phase:** generation of the application's signing key pair and registration on the server

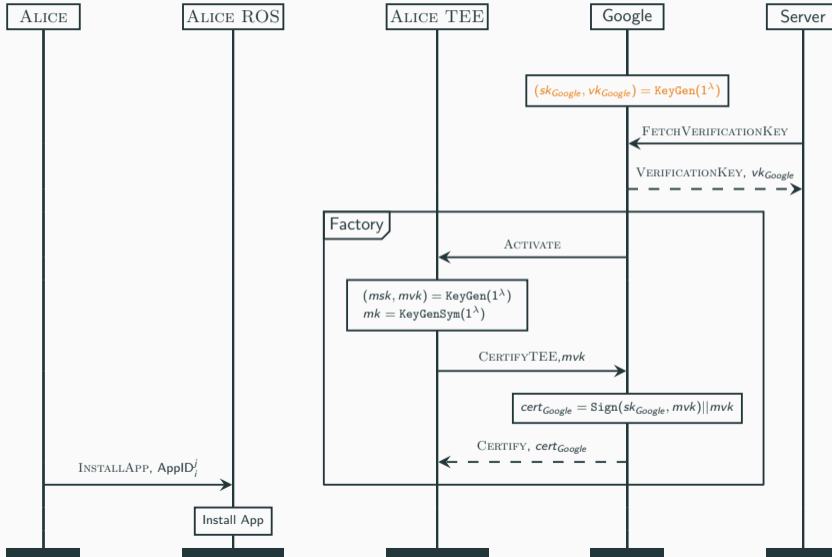
Overview of the protocol

Protocol in three phases:

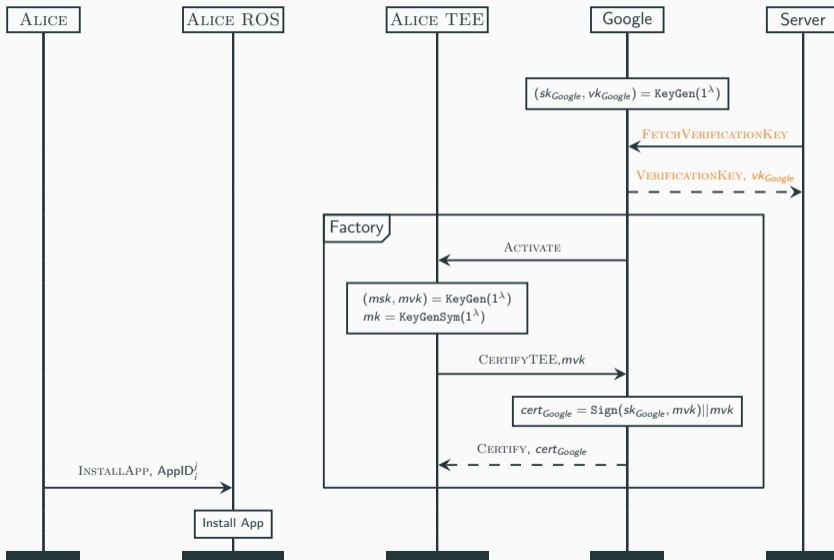
1. **Setup phase:** certification of the TEE, setup of the server, installation of applications on the phone
2. **Registration phase:** generation of the application's signing key pair and registration on the server
3. **Transaction phase:** verification of data by the user and transaction with the server

Setup phase

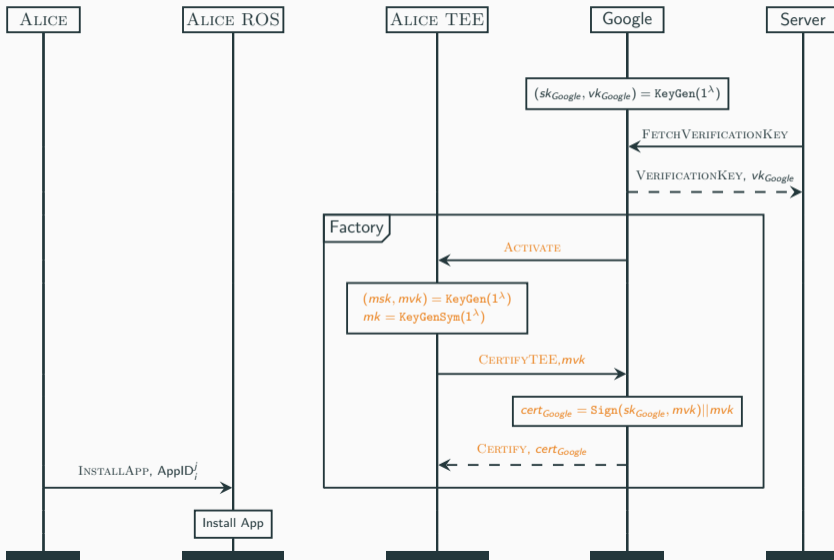
1.Setup phase



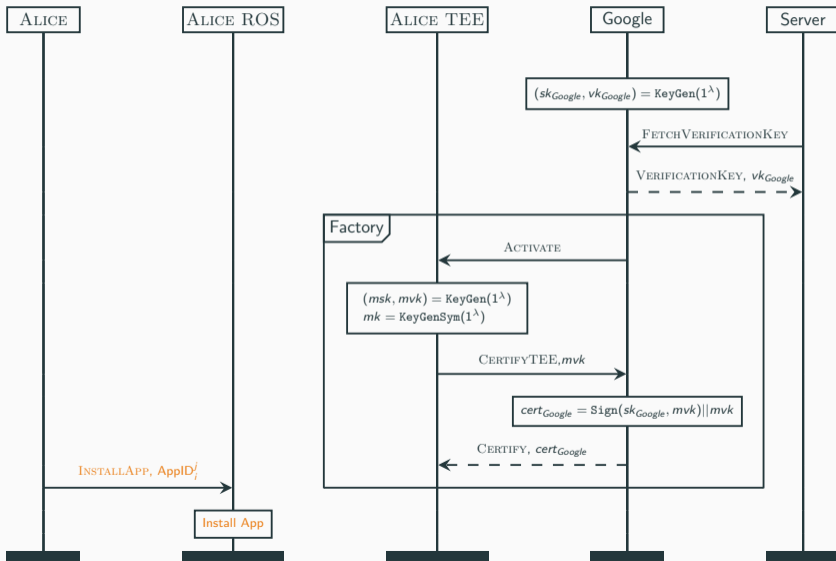
1.Setup phase



1.Setup phase

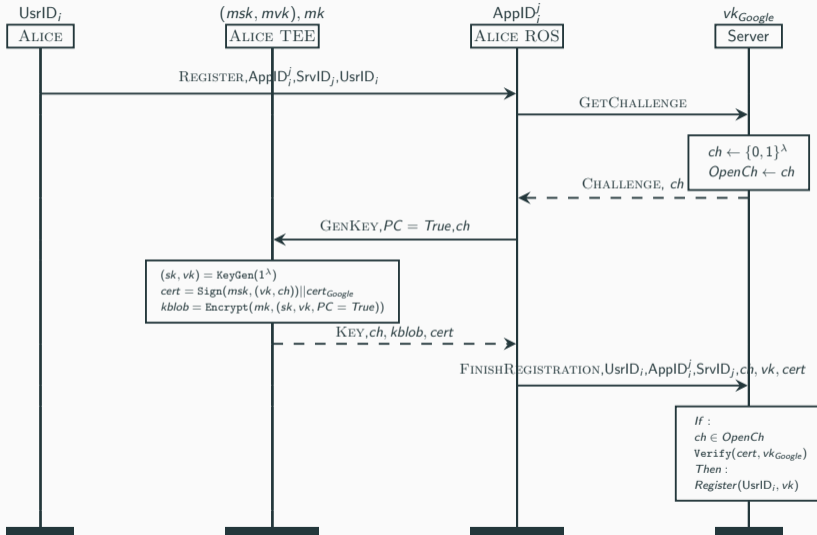


1.Setup phase

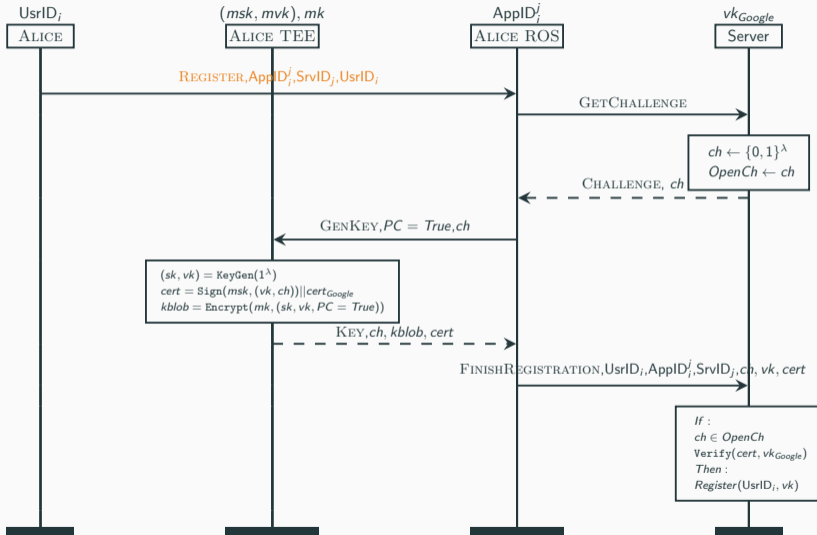


Registration phase

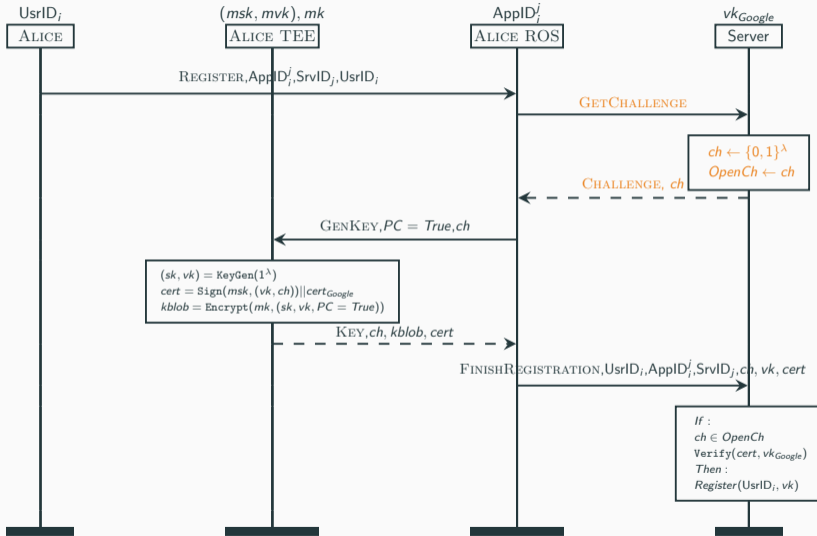
2.Registration phase



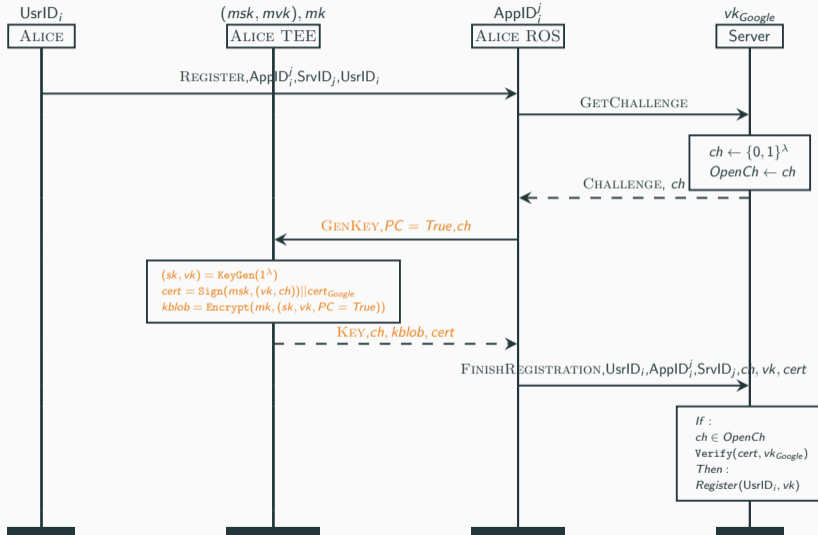
2.Registration phase



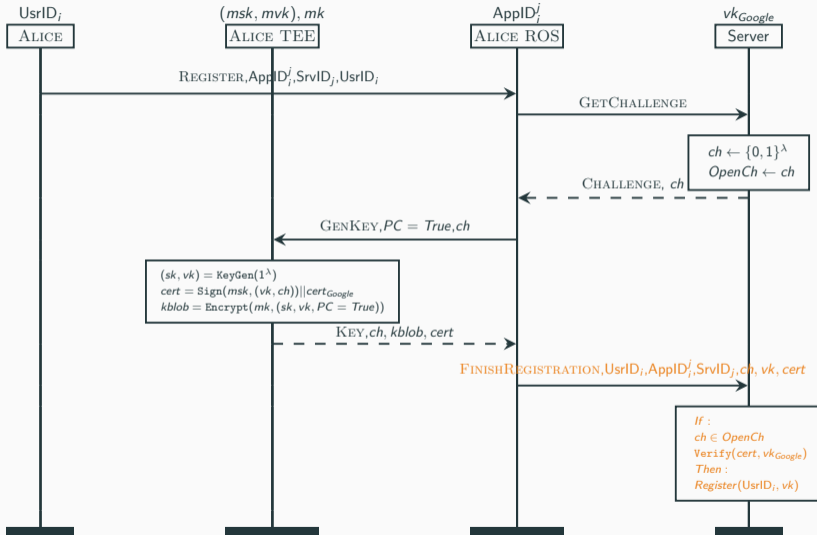
2.Registration phase



2.Registration phase

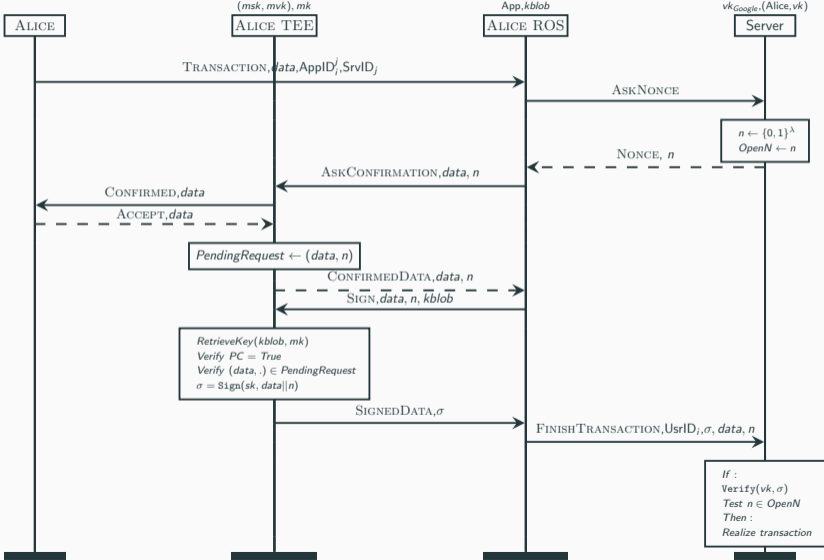


2.Registration phase

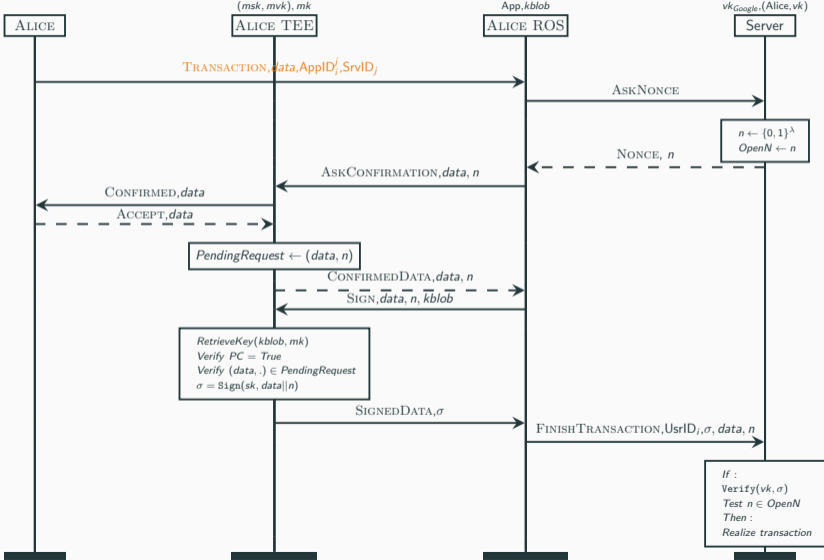


3.Transaction phase

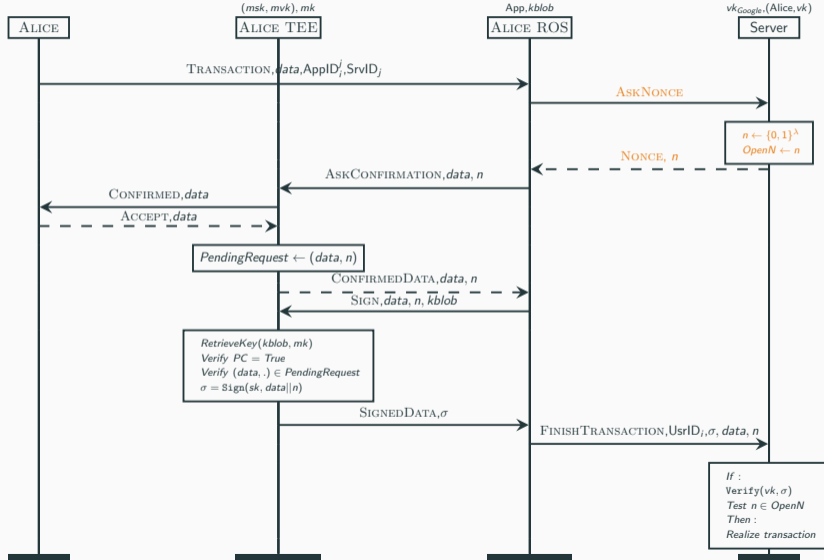
Transaction phase



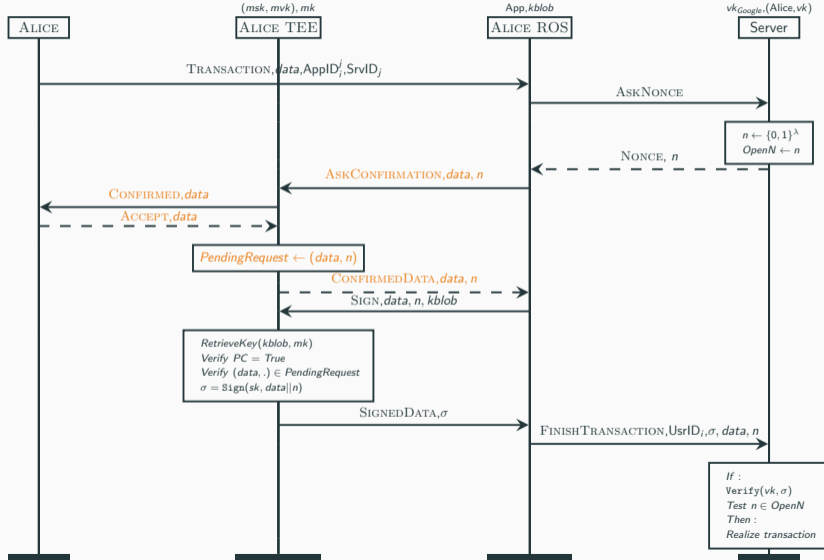
Transaction phase



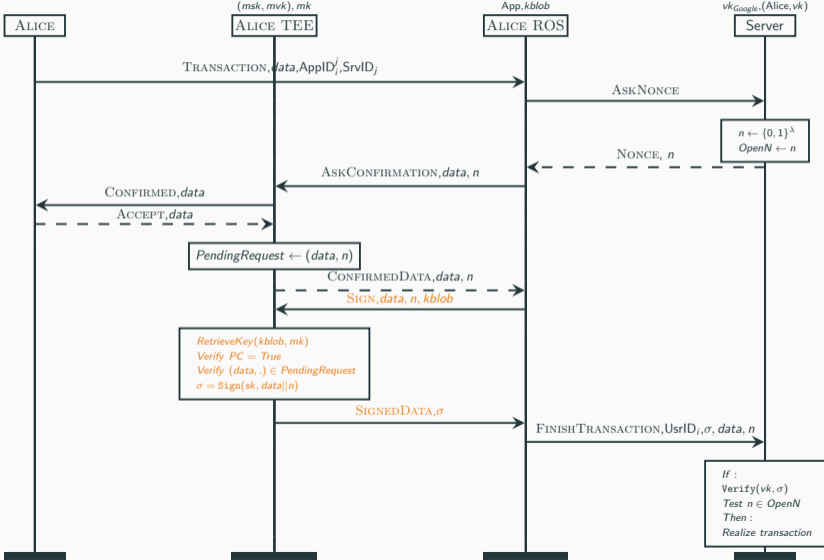
Transaction phase



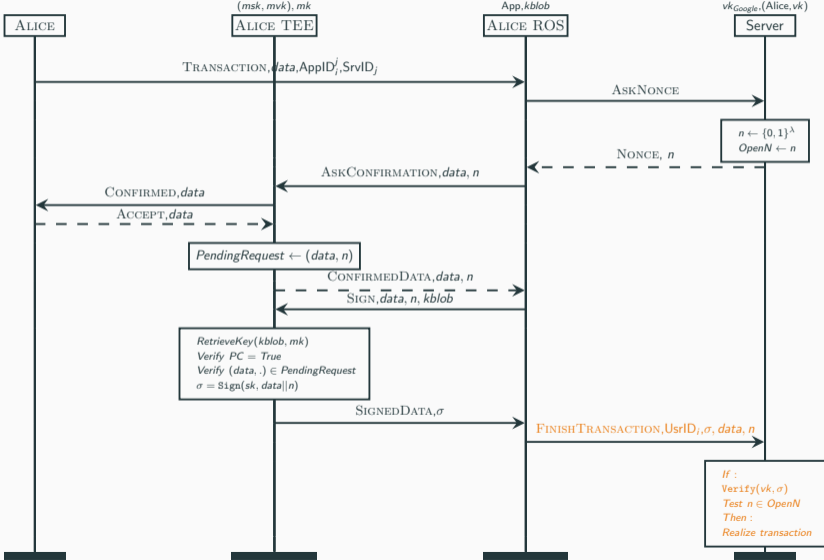
Transaction phase



Transaction phase



Transaction phase



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]

“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]

Claim of the protocol

Server accepts transaction \implies user has validated the transaction.

Threat model: participants

- Alice: honest (if not the protocol has no claim)
- TEE: honest (hypothesis of the protocol)
- ROS: honest **but corruptible**
- Server: honest (if corrupted can perform any transaction anyway)
- Google: honest (at least as a certification authority)

Threat model: Channels

Google

TEE

User

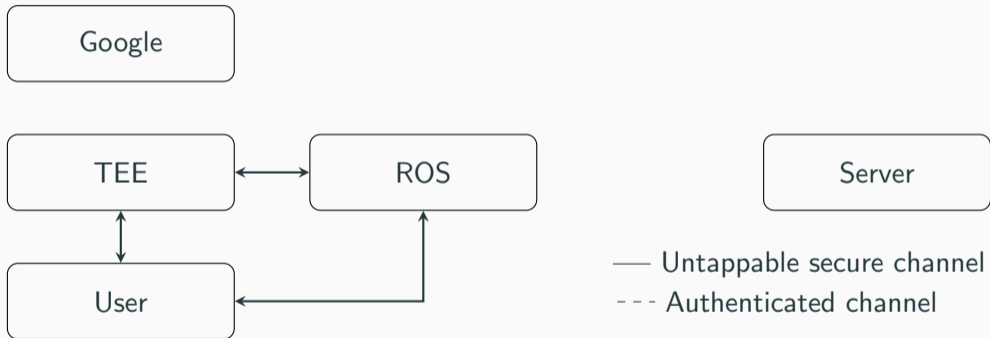
ROS

Server

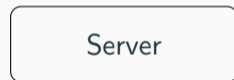
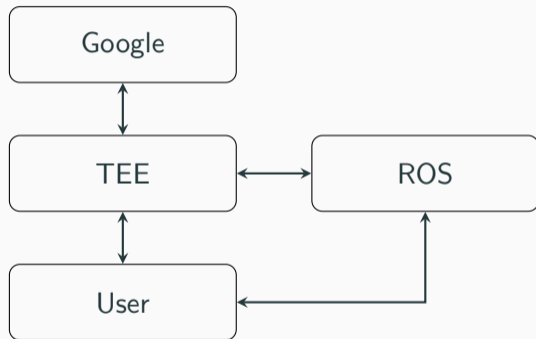
— Untappable secure channel

- - - Authenticated channel

Threat model: Channels



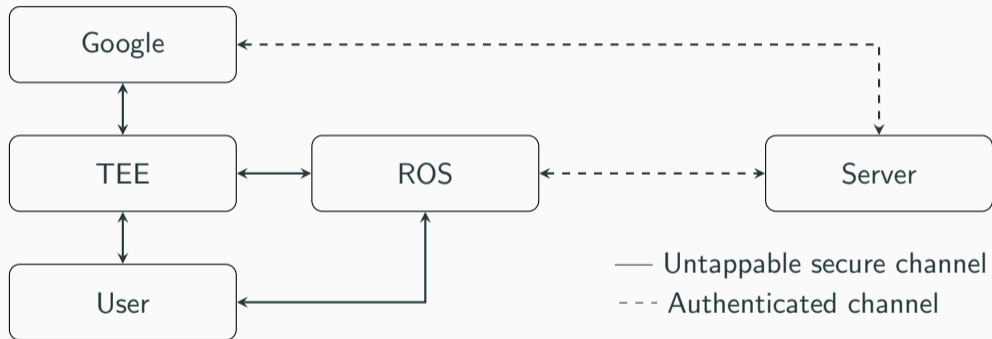
Threat model: Channels



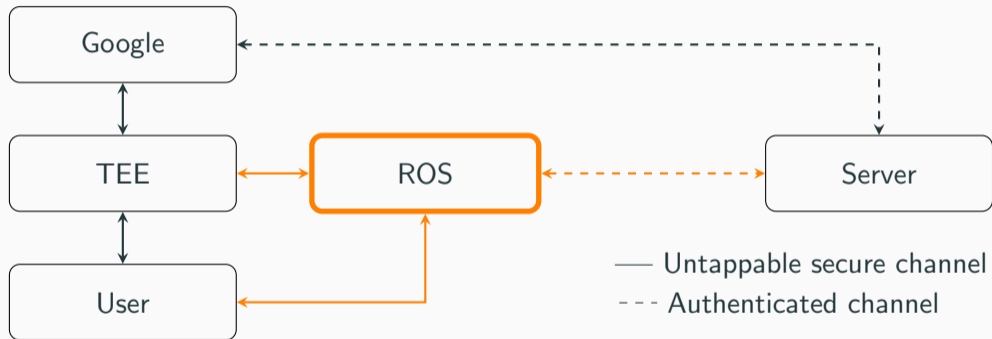
— Untappable secure channel

- - - Authenticated channel

Threat model: Channels



Threat model: Channels



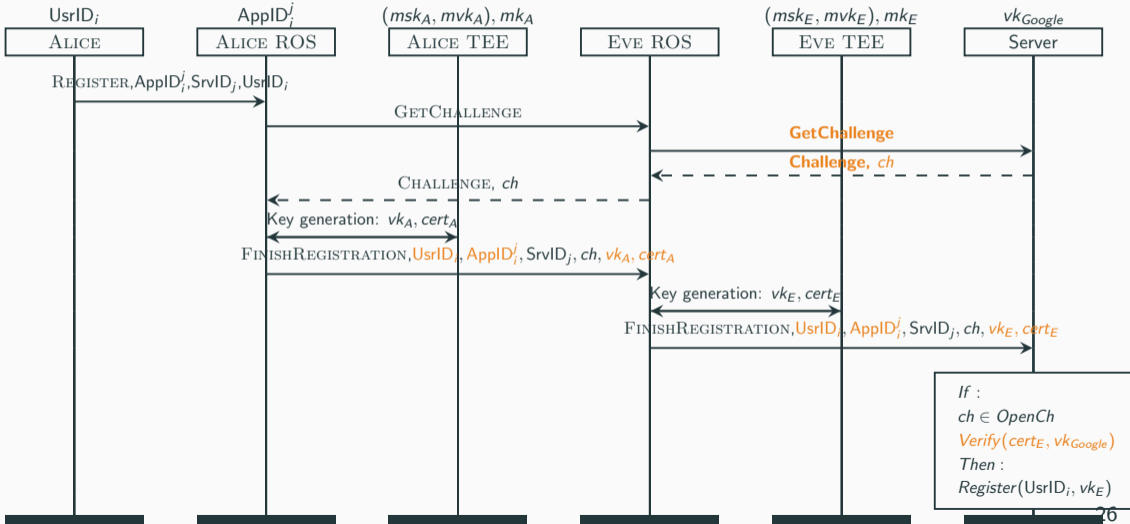
Impersonation at registration: attack and fix

Impersonation at registration (phase 2)

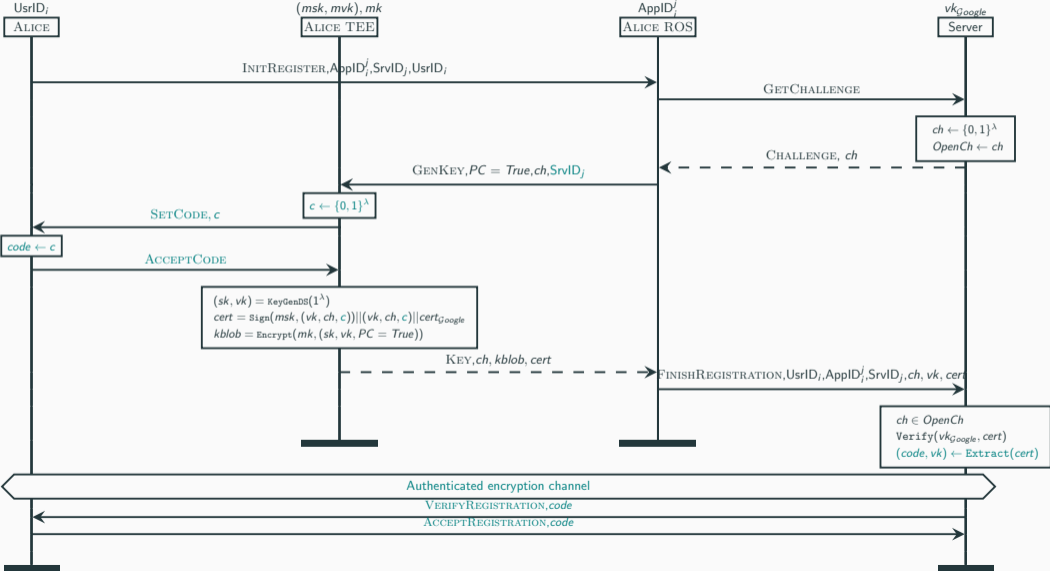
Description

- **Principle:** Duplication of the registration phase and Machine in the Middle
- **Problem:** The check verifies that the signature has been made by **any** TEE.
- **Consequence:** The attacker can register its credentials under the victim's identity (and use them to perform transactions).

Impersonation at registration (phase 2)



Registration phase fix (phase 2)



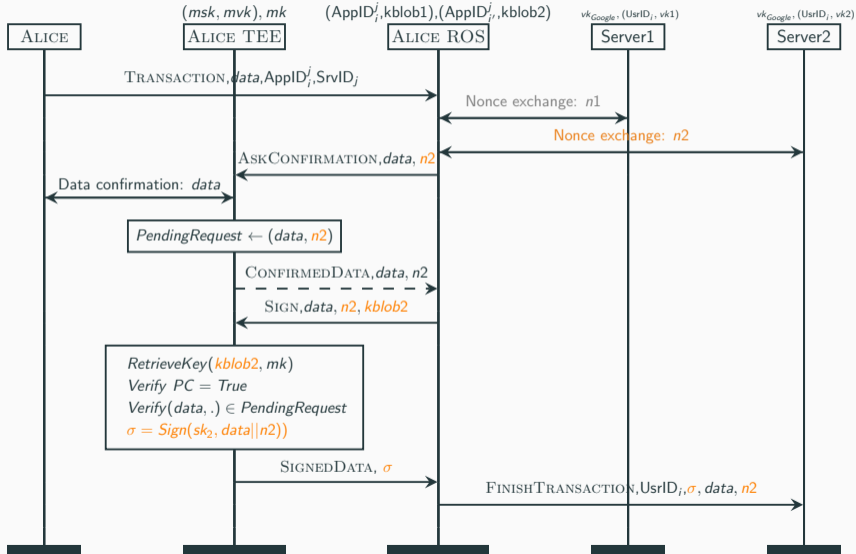
Transaction phase: attack and fix

Transaction replay attack (phase 3)

Description

- **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)
- **Consequence:** the attacker can make a honest server accept an unintended transaction.

Transaction replay attack (phase 3)



Implementation of the attack

The target [AAM23]

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



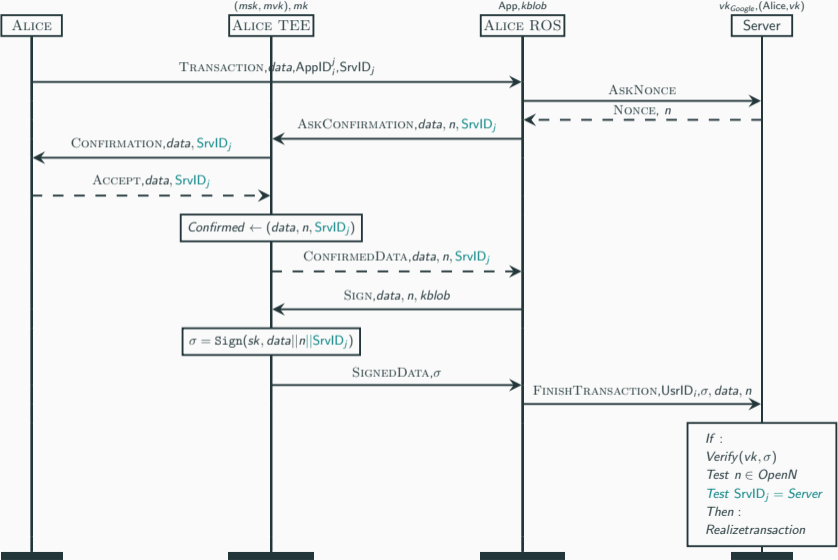
The malicious app [DKM24]

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



Demonstration!

Transaction phase fix (phase 3)



Proving security in the UC framework

The UC framework

- Computational approach
- Participants modeled by interactive Turing Machines
- Real world (protocol π) / ideal world (ideal functionality \mathcal{F}) paradigm
- Guarantee: π is secure no matter what other processes are running in parallel

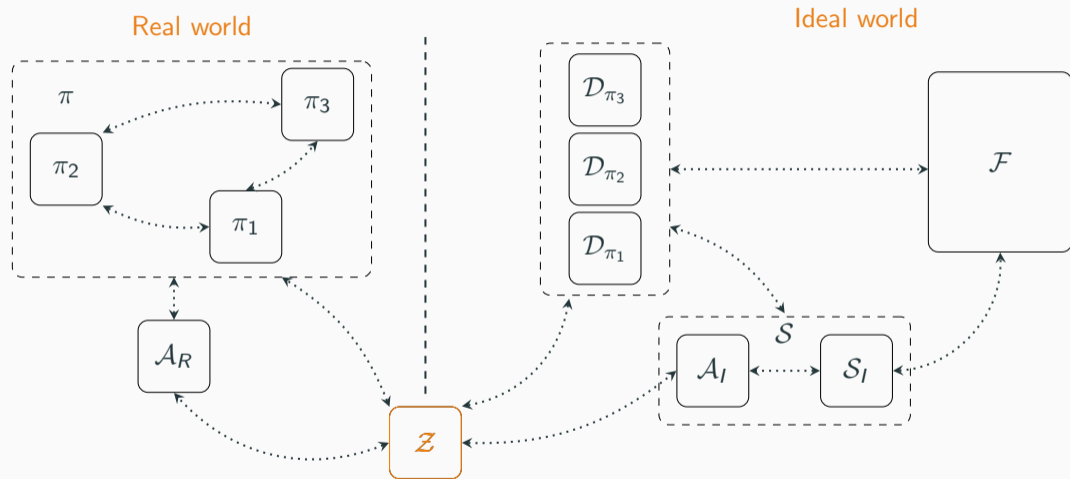
The UC framework

- Computational approach
- Participants modeled by interactive Turing Machines
- Real world (protocol π) / ideal world (ideal functionality \mathcal{F}) paradigm
- Guarantee: π is secure no matter what other processes are running in parallel

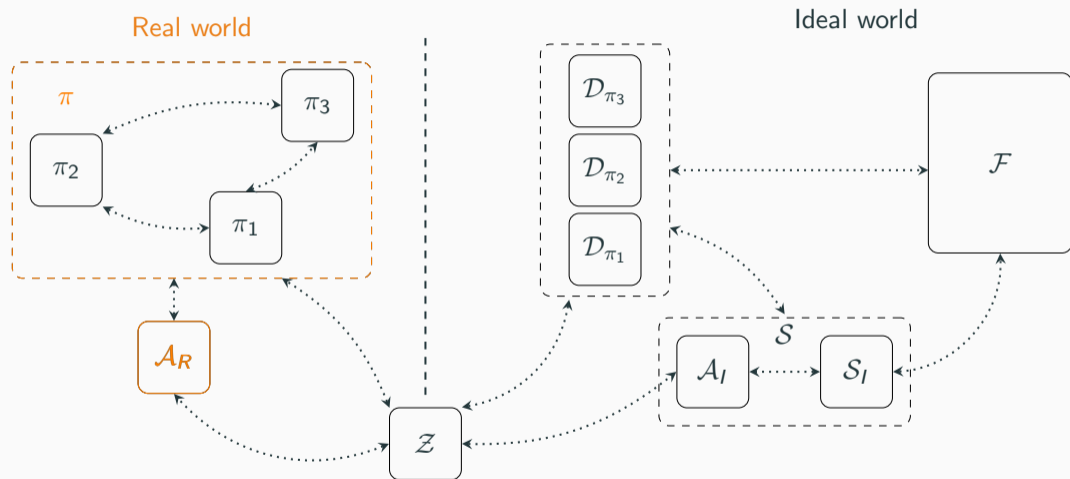
UC-realization

A protocol π is said to *UC-realize* the ideal functionality \mathcal{F} , if for every real world adversary \mathcal{A} , there exists a simulator \mathcal{S} , such that for every environment \mathcal{Z} , the distributions of $\text{EXEC}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ are computationally indistinguishable.

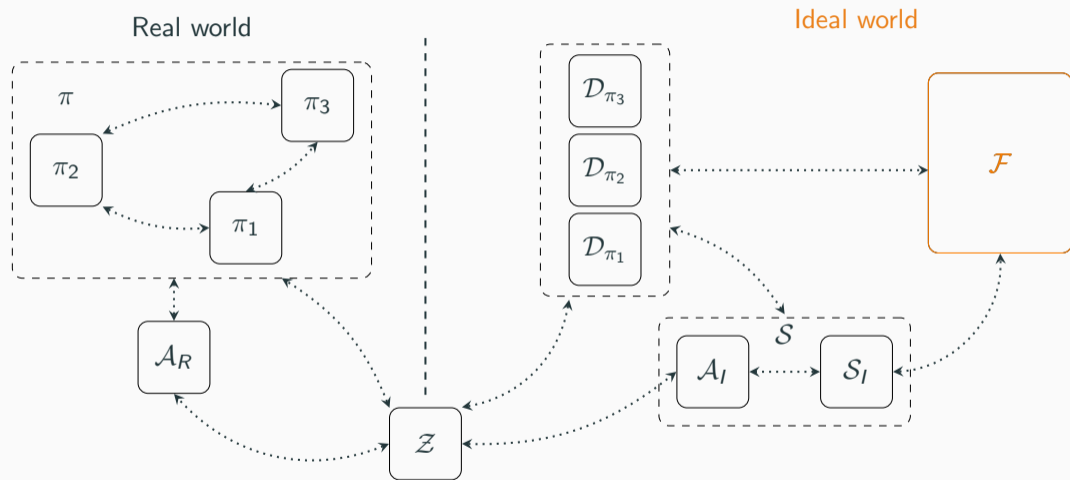
The UC framework



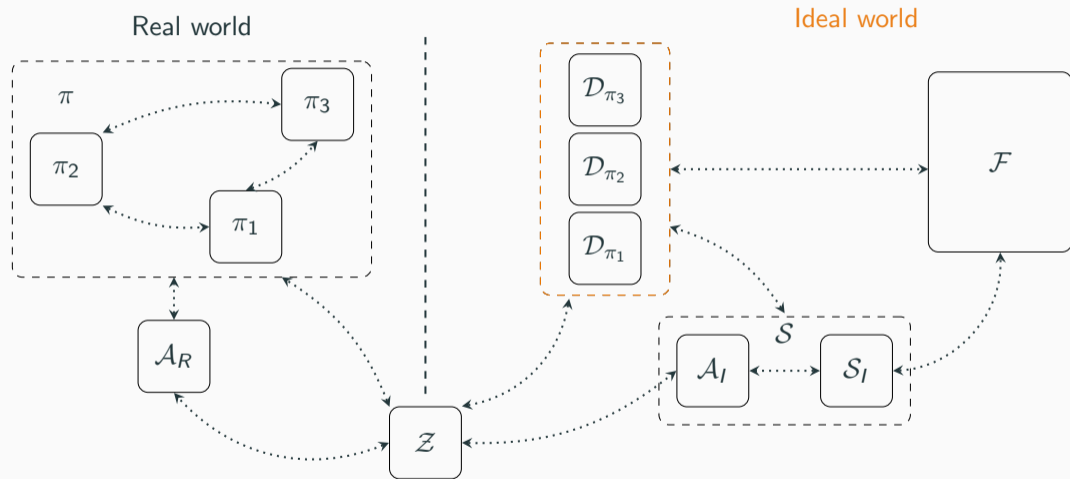
The UC framework



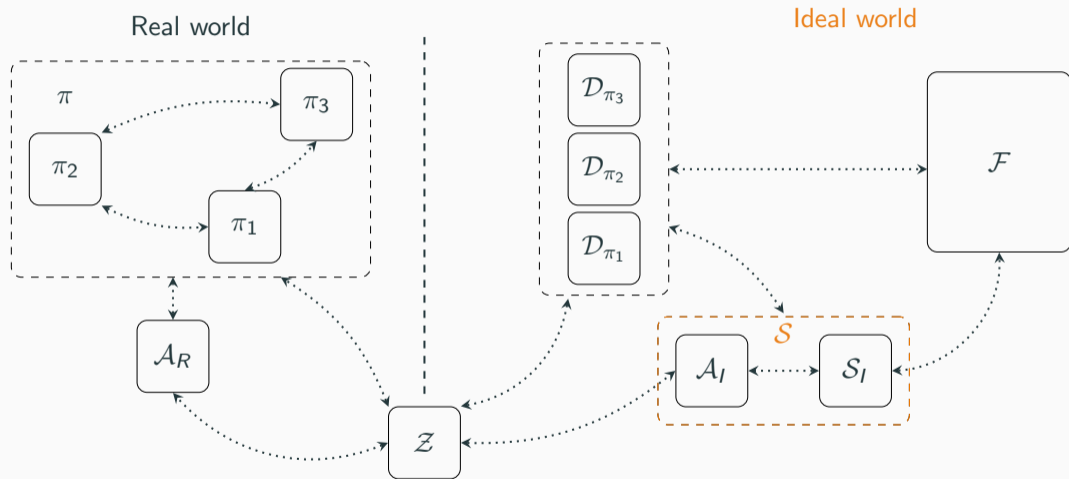
The UC framework



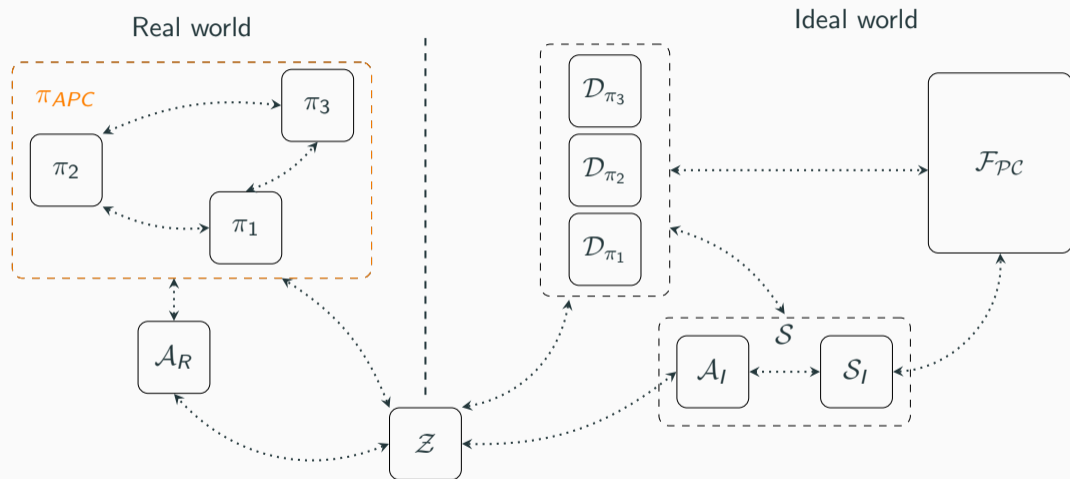
The UC framework



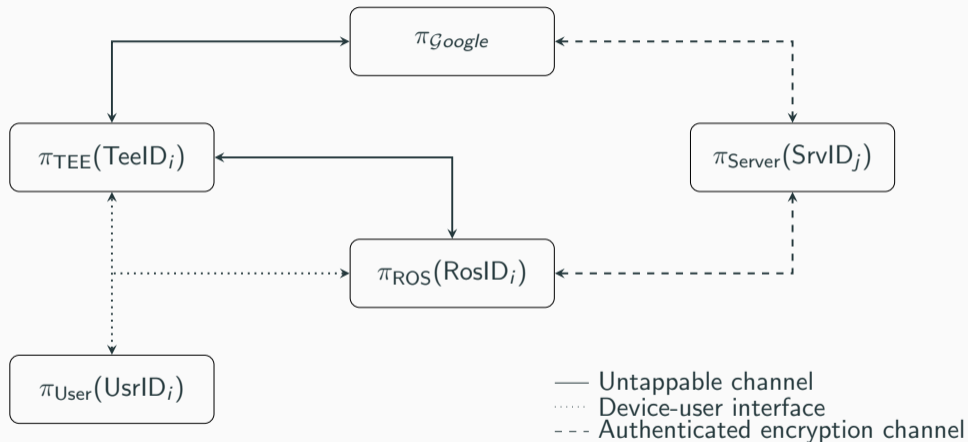
The UC framework



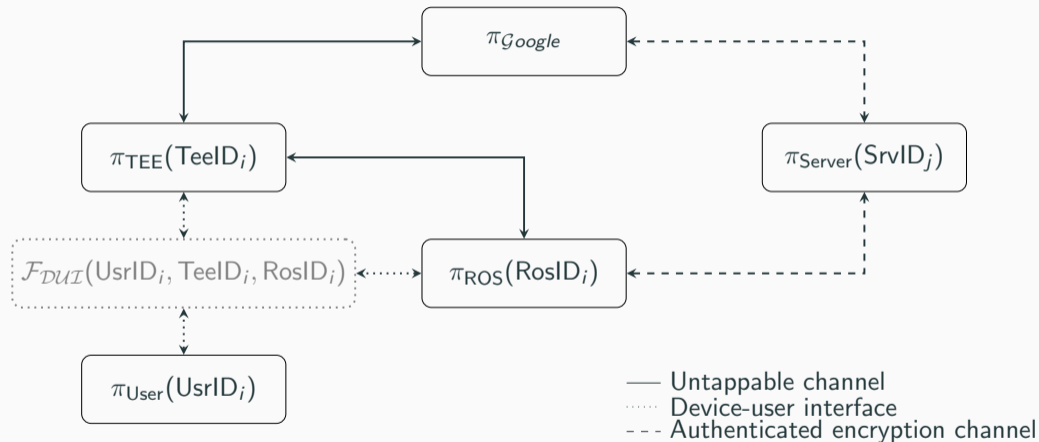
The UC framework



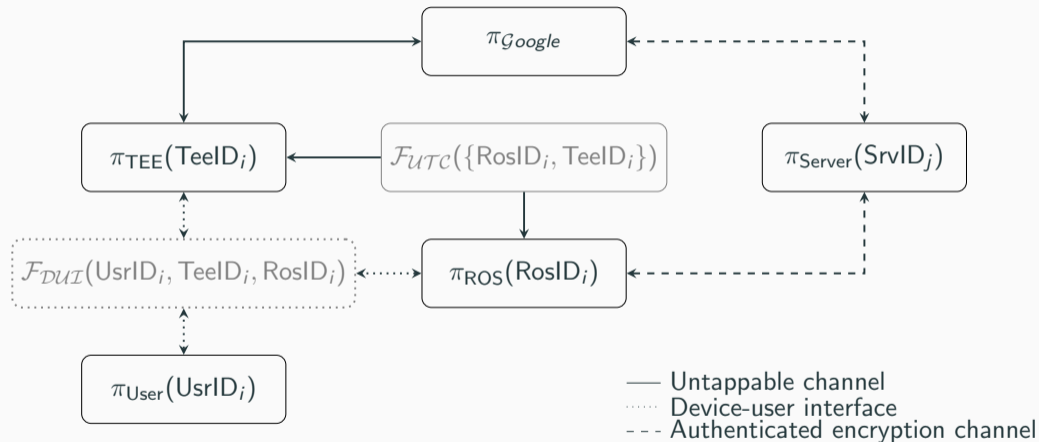
Describing the protocol



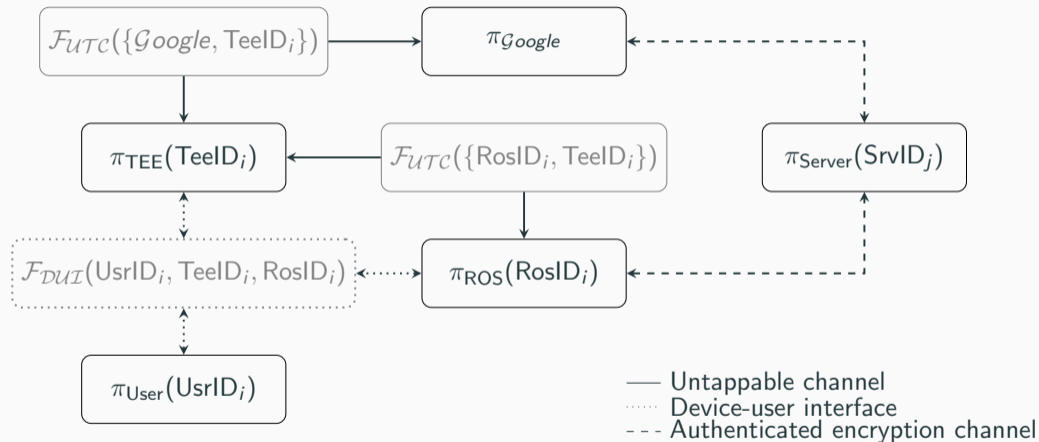
Describing the protocol



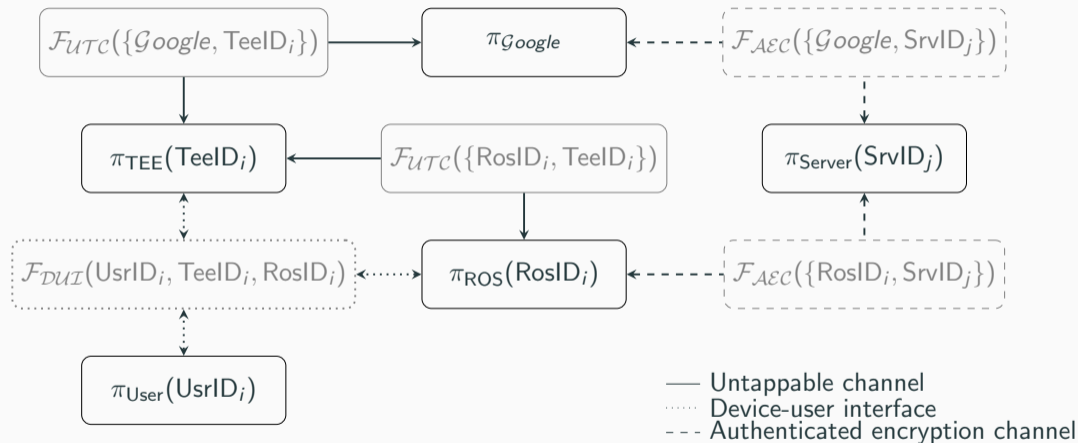
Describing the protocol



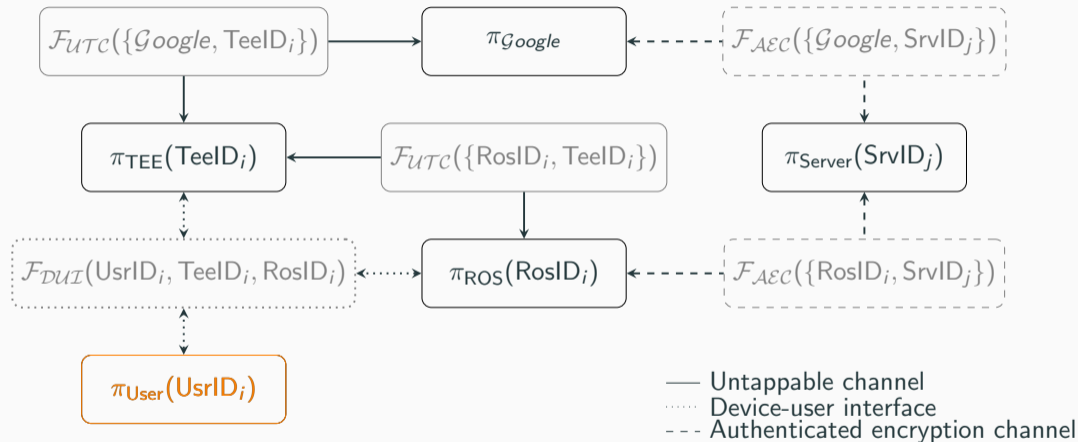
Describing the protocol



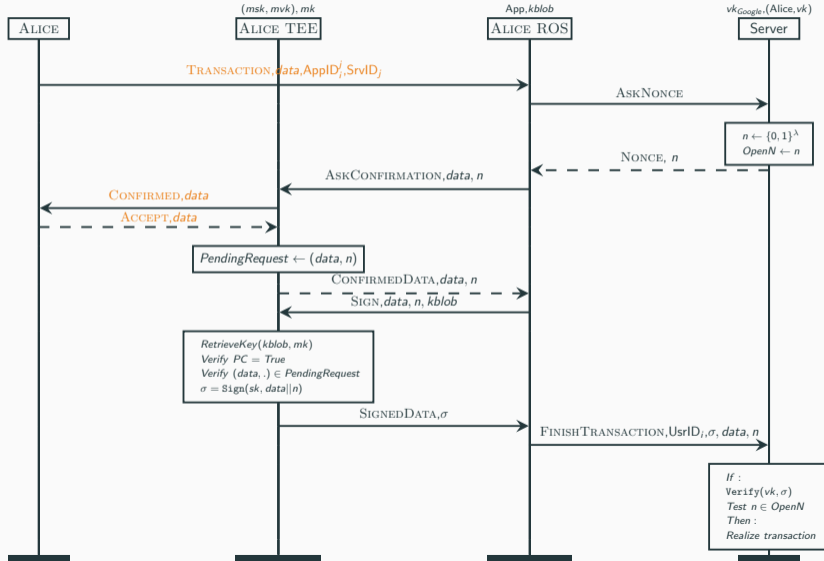
Describing the protocol



Describing the protocol



The user role: transaction phase



Modeling an agent: the user transaction phase

- Upon receiving (INITTRANSACTION , $data$, AppID_i^j , SrvID_j) from \mathcal{Z} ,
 1. store $data$ in ExpectedData ,
 2. send (SEND , (TRANSACTION , $data$, AppID_i^j , SrvID_j), RosID_i) to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.
- Upon receiving (SEND , ($ssid$, CONFIRM , $data$)) from $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$,
 1. verify that $data$ is in ExpectedData ,
 2. remove it from the list,
 3. send (SEND , ($ssid$, ACCEPT , $data$), TeelD_i) to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.

Modeling an agent: the user transaction phase

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{Z} ,
 1. store $data$ in *ExpectedData*,
 2. send $(\text{SEND}, (\text{TRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j), \text{RosID}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.
- Upon receiving $(\text{SEND}, (\text{ssid}, \text{CONFIRM}, data))$ from $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$,
 1. verify that $data$ is in *ExpectedData*,
 2. remove it from the list,
 3. send $(\text{SEND}, (\text{ssid}, \text{ACCEPT}, data), \text{TeelD}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.

Modeling an agent: the user transaction phase

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{Z} ,
 1. store $data$ in ExpectedData ,
 2. send $(\text{SEND}, (\text{TRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j), \text{RosID}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.

- Upon receiving $(\text{SEND}, (\text{ssid}, \text{CONFIRM}, data))$ from $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$,
 1. verify that $data$ is in ExpectedData ,
 2. remove it from the list,
 3. send $(\text{SEND}, (\text{ssid}, \text{ACCEPT}, data), \text{TeelD}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.

Modeling an agent: the user transaction phase

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{Z} ,
 1. store $data$ in $ExpectedData$,
 2. send $(\text{SEND}, (\text{TRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j), \text{RosID}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.

- Upon receiving $(\text{SEND}, (\text{ssid}, \text{CONFIRM}, data))$ from $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$,
 1. verify that $data$ is in $ExpectedData$,
 2. remove it from the list,
 3. send $(\text{SEND}, (\text{ssid}, \text{ACCEPT}, data), \text{TeelD}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.

Modeling an agent: the user transaction phase

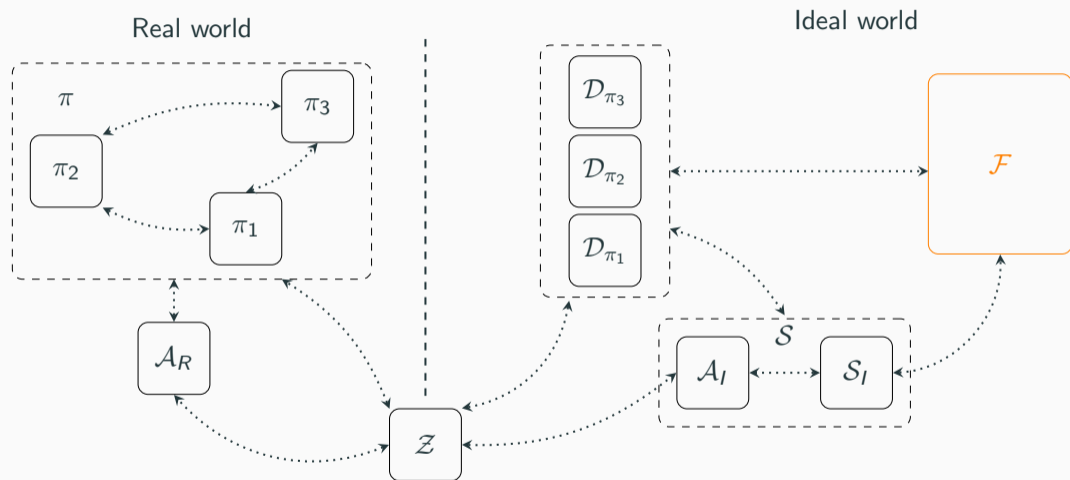
- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{Z} ,
 1. store *data* in *ExpectedData*,
 2. send $(\text{SEND}, (\text{TRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j), \text{RosID}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.
- Upon receiving $(\text{SEND}, (\text{ssid}, \text{CONFIRM}, data))$ from $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$,
 1. verify that *data* is in *ExpectedData*,
 2. remove it from the list,
 3. send $(\text{SEND}, (\text{ssid}, \text{ACCEPT}, data), \text{TeelD}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.

Modeling an agent: the user transaction phase

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{Z} ,
 1. store $data$ in ExpectedData ,
 2. send $(\text{SEND}, (\text{TRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j), \text{RosID}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.

- Upon receiving $(\text{SEND}, (\text{ssid}, \text{CONFIRM}, data))$ from $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$,
 1. verify that $data$ is in ExpectedData ,
 2. remove it from the list,
 3. send $(\text{SEND}, (\text{ssid}, \text{ACCEPT}, data), \text{TeelD}_i)$ to $\mathcal{F}_{DUI}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$.

The UC framework



Goal of the UC functionality

Server accepts transaction \implies user has validated the transaction.

Goal of the UC functionality

Server accepts transaction \implies user has validated the transaction.

$$\begin{aligned} &(\text{TRANSACTIONACCEPTED}, \text{UsrID}_i, \text{data}) \implies \\ &(\text{INITTRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j) \end{aligned}$$

Ideal functionality: transaction phase, corrupted ROS

- Upon receiving $(\text{INITTRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ from UsrID_i ,
 1. If $(\text{UsrID}_i, \text{SrvID}_j)$ is in *RegisteredApps*, then store $(\text{UsrID}_i, \text{SrvID}_j, \text{data})$ to *PendingTransaction*.
 2. Send $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ to \mathcal{S} .
- Upon receiving $(\text{TRANSACTIONALLOWED}, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{S} , if $(\text{UsrID}_i, \text{SrvID}_j, \text{data})$ is in *PendingTransaction*, then:
 1. Remove $(\text{UsrID}_i, \text{SrvID}_j, \text{data})$ from *PendingTransaction*.
 2. Send $(\text{TRANSACTIONACCEPTED}, \text{UsrID}_i, \text{data})$ to SrvID_j .

Ideal functionality: transaction phase, corrupted ROS

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from UsrID_i ,
 1. If $(\text{UsrID}_i, \text{SrvID}_j)$ is in *RegisteredApps*, then store $(\text{UsrID}_i, \text{SrvID}_j, data)$ to *PendingTransaction*.
 2. Send $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, data)$ to \mathcal{S} .
- Upon receiving $(\text{TRANSACTIONALLOWED}, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{S} , if $(\text{UsrID}_i, \text{SrvID}_j, data)$ is in *PendingTransaction*, then:
 1. Remove $(\text{UsrID}_i, \text{SrvID}_j, data)$ from *PendingTransaction*.
 2. Send $(\text{TRANSACTIONACCEPTED}, \text{UsrID}_i, data)$ to SrvID_j .

Ideal functionality: transaction phase, corrupted ROS

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from UsrID_i ,
 1. If $(\text{UsrID}_i, \text{SrvID}_j)$ is in *RegisteredApps*, then store $(\text{UsrID}_i, \text{SrvID}_j, data)$ to *PendingTransaction*.
 2. Send $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, data)$ to \mathcal{S} .
- Upon receiving $(\text{TRANSACTIONALLOWED}, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{S} , if $(\text{UsrID}_i, \text{SrvID}_j, data)$ is in *PendingTransaction*, then:
 1. Remove $(\text{UsrID}_i, \text{SrvID}_j, data)$ from *PendingTransaction*.
 2. Send $(\text{TRANSACTIONACCEPTED}, \text{UsrID}_i, data)$ to SrvID_j .

Ideal functionality: transaction phase, corrupted ROS

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from UsrID_i ,
 1. If $(\text{UsrID}_i, \text{SrvID}_j)$ is in *RegisteredApps*, then store $(\text{UsrID}_i, \text{SrvID}_j, data)$ to *PendingTransaction*.
 2. Send $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, data)$ to \mathcal{S} .
- Upon receiving $(\text{TRANSACTIONALLOWED}, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{S} , if $(\text{UsrID}_i, \text{SrvID}_j, data)$ is in *PendingTransaction*, then:
 1. Remove $(\text{UsrID}_i, \text{SrvID}_j, data)$ from *PendingTransaction*.
 2. Send $(\text{TRANSACTIONACCEPTED}, \text{UsrID}_i, data)$ to SrvID_j .

Ideal functionality: transaction phase, corrupted ROS

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from UsrID_i ,
 1. If $(\text{UsrID}_i, \text{SrvID}_j)$ is in *RegisteredApps*, then store $(\text{UsrID}_i, \text{SrvID}_j, data)$ to *PendingTransaction*.
 2. Send $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, data)$ to \mathcal{S} .
- Upon receiving $(\text{TRANSACTIONALLOWED}, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{S} , if $(\text{UsrID}_i, \text{SrvID}_j, data)$ is in *PendingTransaction*, then:
 1. Remove $(\text{UsrID}_i, \text{SrvID}_j, data)$ from *PendingTransaction*.
 2. Send $(\text{TRANSACTIONACCEPTED}, \text{UsrID}_i, data)$ to SrvID_j .

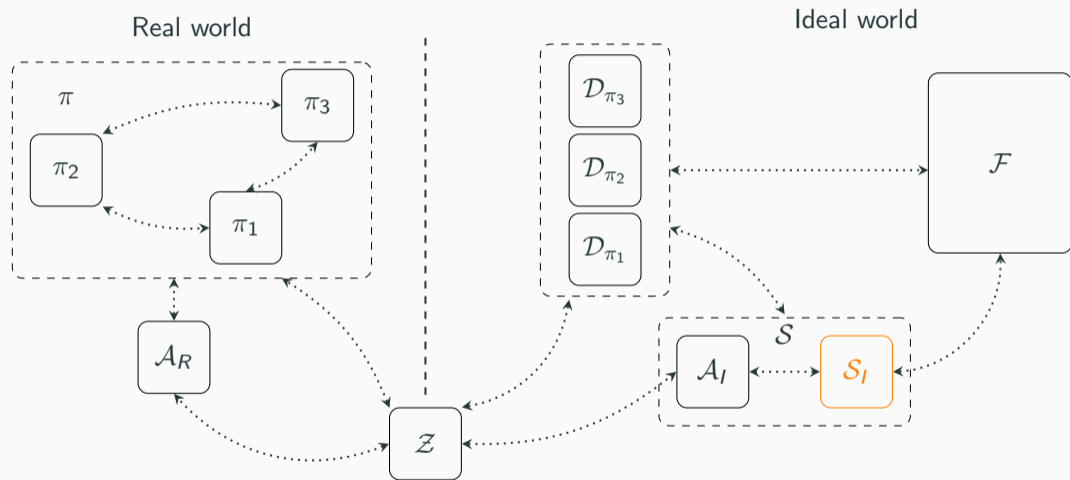
Ideal functionality: transaction phase, corrupted ROS

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from UsrID_i ,
 1. If $(\text{UsrID}_i, \text{SrvID}_j)$ is in *RegisteredApps*, then store $(\text{UsrID}_i, \text{SrvID}_j, data)$ to *PendingTransaction*.
 2. Send $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, data)$ to \mathcal{S} .
- Upon receiving $(\text{TRANSACTIONALLOWED}, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{S} , if $(\text{UsrID}_i, \text{SrvID}_j, data)$ is in *PendingTransaction*, then:
 1. Remove $(\text{UsrID}_i, \text{SrvID}_j, data)$ from *PendingTransaction*.
 2. Send $(\text{TRANSACTIONACCEPTED}, \text{UsrID}_i, data)$ to SrvID_j .

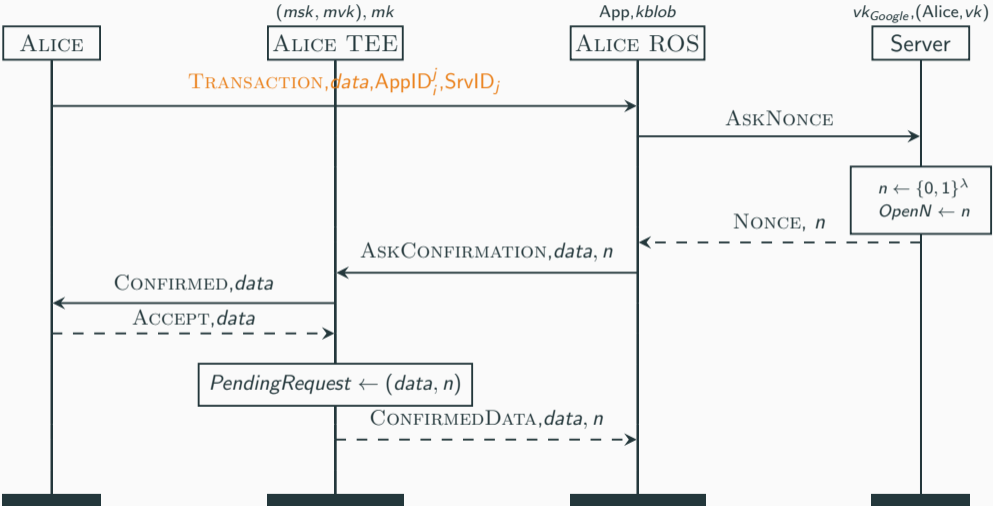
Ideal functionality: transaction phase, corrupted ROS

- Upon receiving $(\text{INITTRANSACTION}, data, \text{AppID}_i^j, \text{SrvID}_j)$ from UsrID_i ,
 1. If $(\text{UsrID}_i, \text{SrvID}_j)$ is in *RegisteredApps*, then store $(\text{UsrID}_i, \text{SrvID}_j, data)$ to *PendingTransaction*.
 2. Send $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, data)$ to \mathcal{S} .
- Upon receiving $(\text{TRANSACTIONALLOWED}, \text{AppID}_i^j, \text{SrvID}_j)$ from \mathcal{S} , if $(\text{UsrID}_i, \text{SrvID}_j, data)$ is in *PendingTransaction*, then:
 1. Remove $(\text{UsrID}_i, \text{SrvID}_j, data)$ from *PendingTransaction*.
 2. Send $(\text{TRANSACTIONACCEPTED}, \text{UsrID}_i, data)$ to SrvID_j .

The UC framework



Introduction to the simulator



Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving ($\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data}$) from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

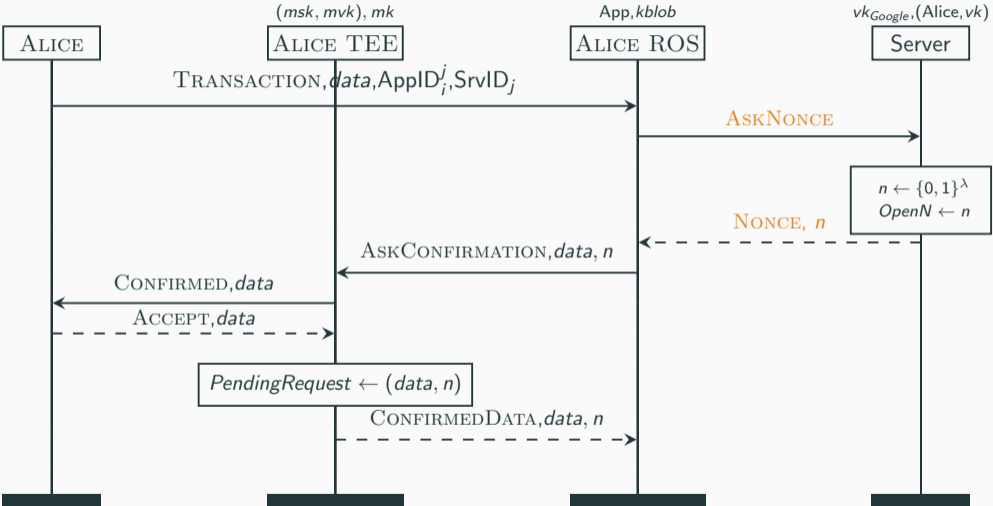
Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

Introduction to the simulator



Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - **check is SrvID_j is in *ActivatedServer***, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

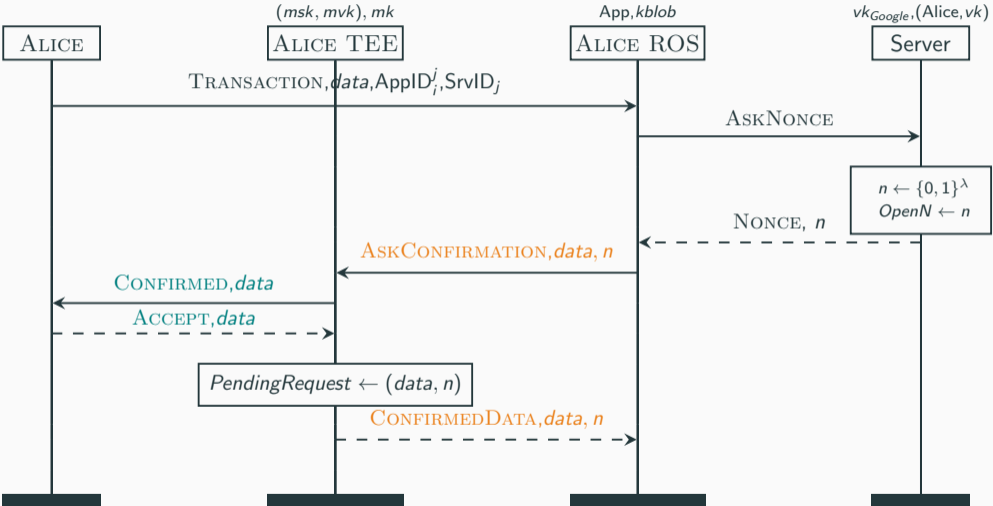
Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

Introduction to the simulator



Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

Simulating the protocol: extract of the corrupted transaction phase

- Upon receiving $(\text{ALLOWTRANSACTION}, \text{AppID}_i^j, \text{RosID}_i, \text{UsrID}_i, \text{SrvID}_j, \text{data})$ from $\mathcal{F}_{\mathcal{PC}}$,
 - store $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*
 - send $(\text{TRANSACTION}, \text{data}, \text{AppID}_i^j, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{DUI}}(\text{UsrID}_i, \text{TeelD}_i, \text{RosID}_i)$
- On behalf of $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$ and SrvID_j , upon receiving (ASKNONCE) from \mathcal{A} , then
 - check if SrvID_j is in *ActivatedServer*, if not ignore the rest;
 - draw a random $n \xleftarrow{\$} \{0, 1\}^\lambda$ and store (SrvID_j, n) in *PendingNonce*
 - send (NONCE, n) to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{AEC}}(\{\text{RosID}_i, \text{SrvID}_j\})$.
- On behalf of TeelD_i and $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$, upon receiving $(\text{ASKCONFIRMATION}, \text{data}, n, \text{SrvID}_j)$ from \mathcal{A} , if there is $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ in *PendingAllowTransaction*,
 - remove $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ from *PendingAllowTransaction* and add $(\text{AppID}_i^j, \text{RosID}_i, \text{SrvID}_j, \text{data})$ to *ValidatedData*
 - send $(\text{CONFIRMEDDATA}, \text{data}, n, \text{SrvID}_j)$ to \mathcal{A} as if \mathcal{S} was $\mathcal{F}_{\mathcal{UTC}}(\{\text{RosID}_i, \text{TeelD}_i\})$

Π_{APC} does not UC-realizes \mathcal{F}

(would accept a transaction with the wrong server id)

$\Pi_{APC,fix}$ does UC-realizes \mathcal{F}

Conclusion

Results

- Two attacks on the deployed protocol APC
 1. Impersonation at registration attack
 2. Transaction phase attack (PoC!)
- Fixes of both attacks have been proved in UC
- Google acknowledged our findings

↔ Paper under submission at Euro S&P

Thank you for your attention !

Questions?



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.](https://developer.android.com/privacy-and-security/security-android-protected-confirmation)

-  Janis Danisevskis.
Android protected confirmation: Taking transaction security to the next level.
`https://android-developers.googleblog.com/2018/10/android-protected-confirmation.html`, 2018.
-  Jannik Dreier, Steve Kremer, and Racouchot Maiïwenn.
Additional resources, 2024.
-  David Robin.
Yubidroid.
`https://www.robindar.com/yubidroid/getting-source-code`, 2021.