UNIVERSITÀ DEGLI STUDI DI UDINE

Breaking and Fixing CieID: Formal Verification of the Italian e-ID Multi-Factor Authentication

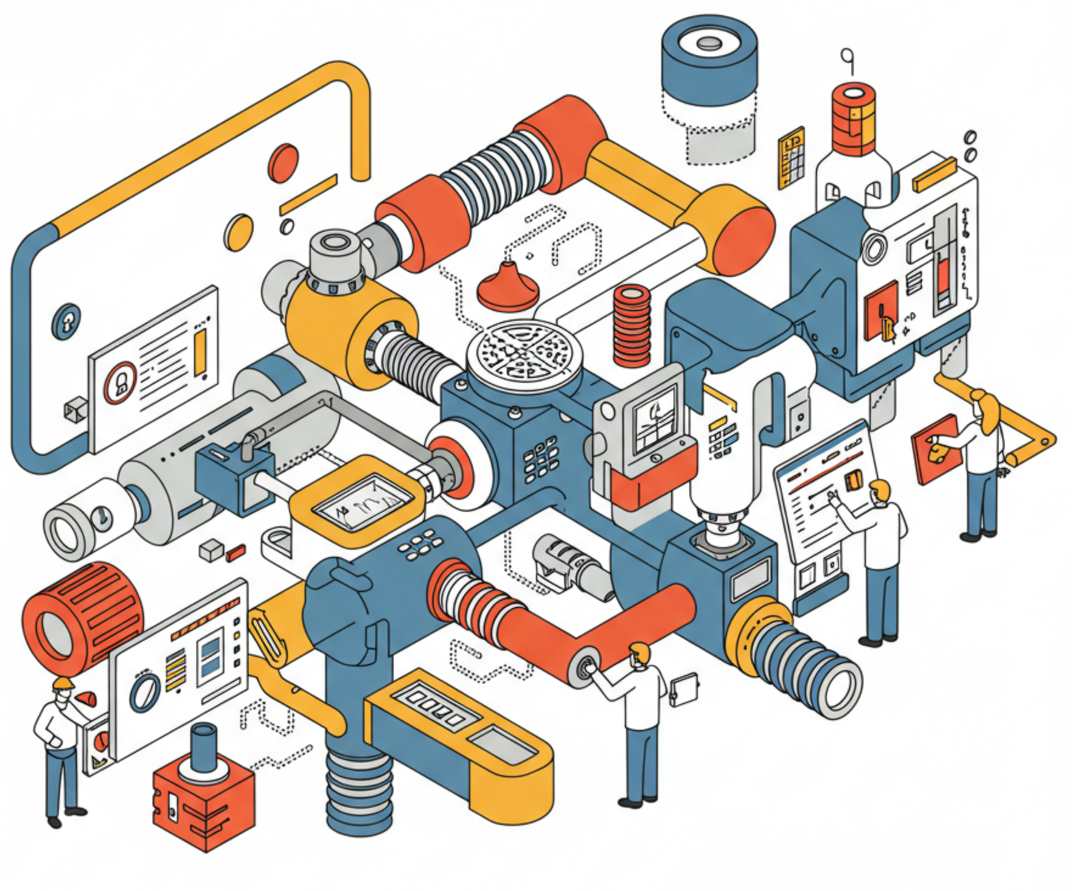
Marino Miculan

DMIF, University of Udine
marino.miculan@uniud.it
https://marino.miculan.org

DISCO, UniMIB, March 3, 2026

Modern systems are made of components

- Modern systems are complex architectures built by assembling interacting and interdependent *components*
  - Software modules
  - Hardware components
  - Physical devices (IoT, Cyber-physical systems)
  - Even people



Many theories for components and interfaces

- A component description answers the question *What does it do?*
- An interface description answers the question *How can it be used?*
- Components do not constrain the environment; interfaces do
- *Composition*: a partial function on components and interfaces, result depend on compatibility criteria
- Different theories of components and interfaces induce different composition criteria and focus on different aspects of systems
    - correctness, liveness, quantitative aspects, QoS, resilience, …
- Choice of the meta-model depends on what we want to formalize and analyze

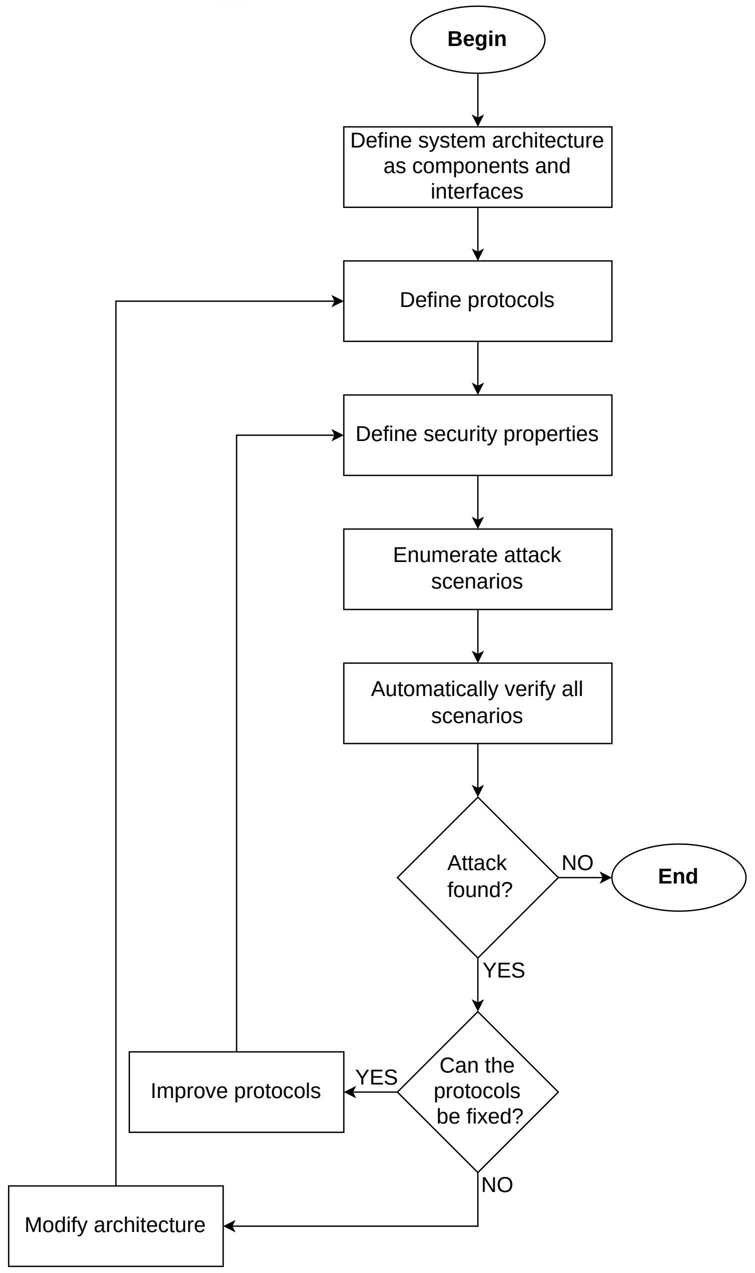
In this talk: focus on security protocols

- Security protocols are
  - ubiquitous
  - fundamental in distributed component architectures
  - error-prone ("Security protocols are three line programs that people still manage to get wrong", cit. R. Needham)
  - And there are good tools for their formal verification
- We will focus on the security aspects between components, in the symbolic Dolev-Yao model
  - Thus our interfaces and our description of components will be towards these aspects

Methodology

To analyze the security of a system in a model:

1. Identify components and interfaces (*architecture*)
2. Identify their interactions (e.g. *sequence diagrams*)
3. Formalize required properties
4. Enumerate attack scenarios
5. Analyze these scenarios using formal tools
6. If any flaws found in protocols: fix and go to 3
7. If not enough, change the architecture and go to 2


Begin
Define system architecture as components and interfaces
Define protocols
Define security properties
Enumerate attack scenarios
Automatically verify all scenarios
Attack found?
NO
End
YES
Can the protocols be fixed?
YES
Improve protocols
NO
Modify architecture

Let’s do it on a real case

- We apply this methodology, based on components and interfaces, for formally modelling and verifying MFA schemes employed in eIDAS digital identity cards (notoriously difficult to implement [Lips et al. 2020])
- Case study: the Italian **electronic identity card** (CIE), which aims to offer robust digital authentication and electronic signature capabilities
    - Effectively replacing the paper ID since 2018
    - More than 10 millions cards already issued
    - Gonna be ~40 millions in 5 years
    - Used everyday for accessing public services
    - Valid across Europe
- We will need to extend the methodology with tools for automatic generating various attack scenarios

CIE ID architecture

- The main parties involved are the CIE Identity Provider (IdP) and federated public administration Service Providers (SPs).
- The IdP, managed by the Italian Ministry of the Interior, authenticates users and generates security assertions.
- CIE provides various multi-factor protocols built upon the SAML 2.0 framework, according to three authentication levels:
    - **Level 1 (Single Factor):** Basic username and password login.
    - **Level 2 (Two Factors):** Via an one-time passcode (OTP) received via SMS, or via the CieID app on a registered device (which is the second factor).
    - **Level 3 (Two Factors with smartcard):** Via the physical CIE smartcard and PIN.

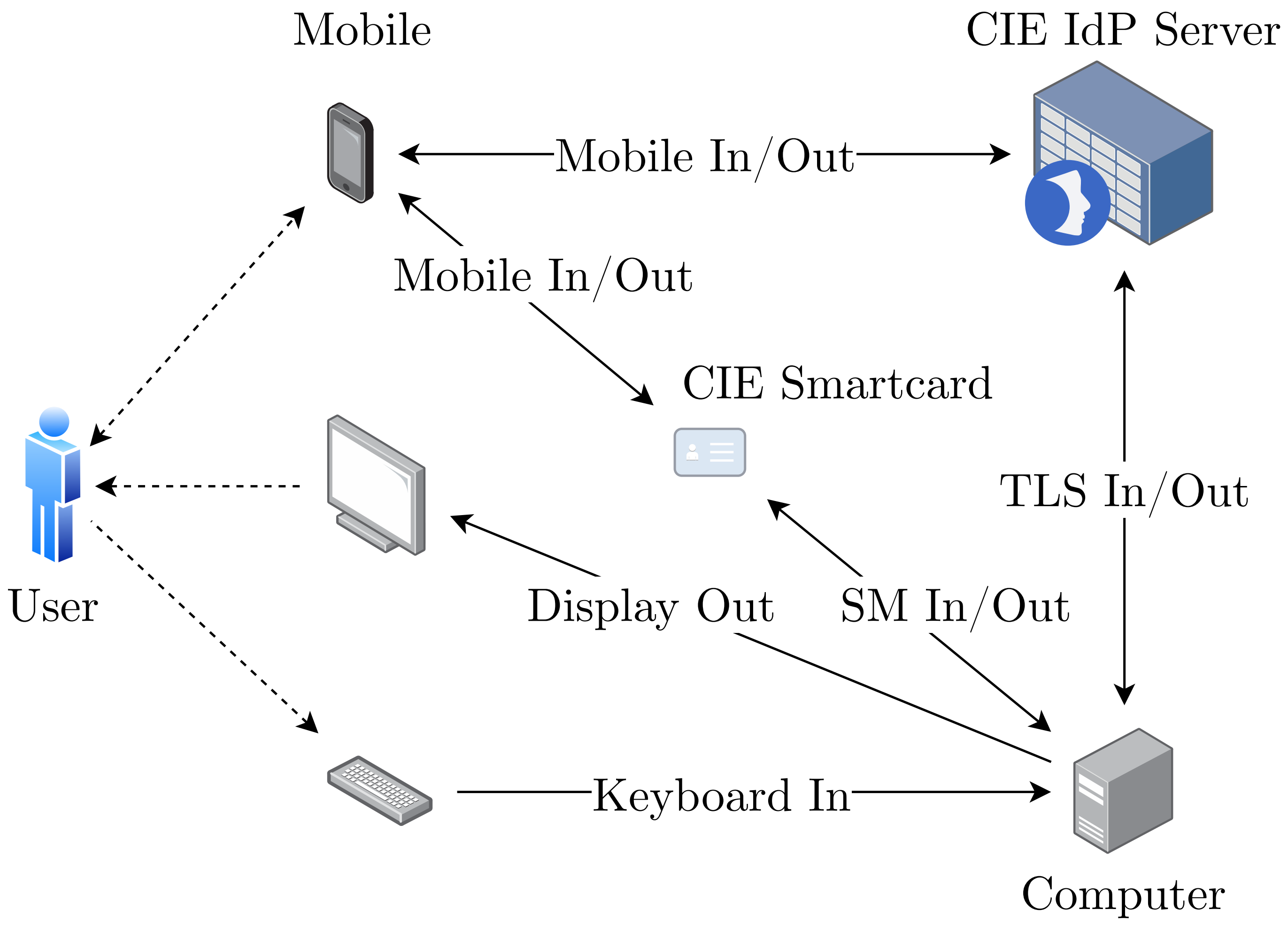
CIE Level 2 authentication protocols

CIE defines three different protocols for Level 2 authentication:

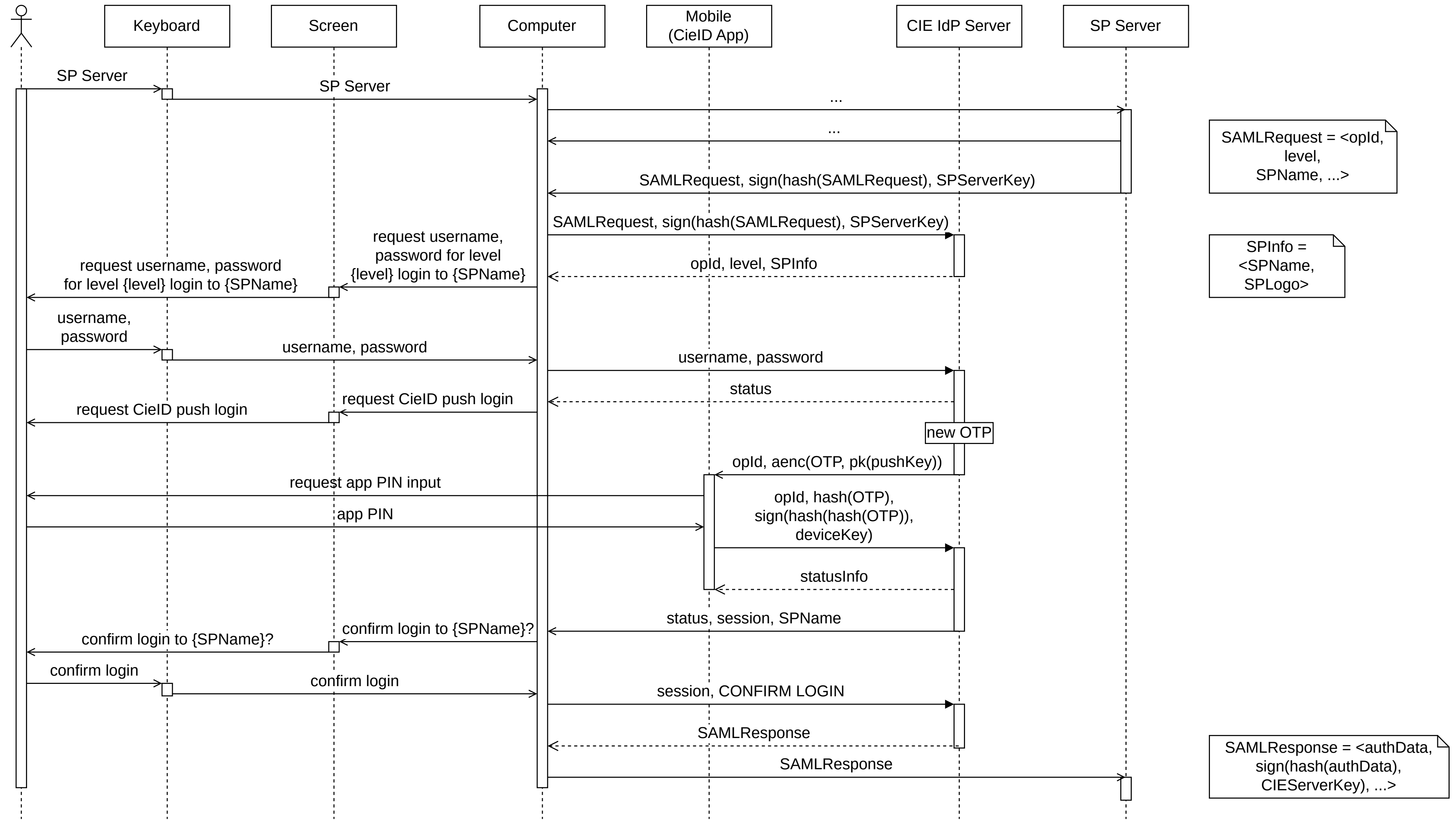
- **L2SMS**: The user enters their level 1 username and password; he is sent a OTP code via SMS; he enters the received code on the computer; and the entered code is checked by the IdP server;
- **L2PSH**: The user sends username and password; the IdP server sends a 4 digit OTP to the CieID app via a push notification; the user authorise the login using the CieID PIN; the CieID app signs a hash of the OTP code; and the IdP server validates it.
- **L2QRC**: After receiving username and password the IdP server generates a challenge, which presented as a QR code; the user scans the QR code with the CieID app and enters their PIN; the app signs a hash of the challenge using the private key of the device; and the server validates the signature.

Identifying components and interfaces of the system

- This model is obtained by reading CIE documentation, code analysis and some reverse engineering
- We abstract from the content of the specific components (see [Jacomme, Kremer 2021])
- Several interaction mechanisms, depending on the components
- Level of abstraction is important
  - Sufficient to express relevant attacks
  - The finer not necessarily the better

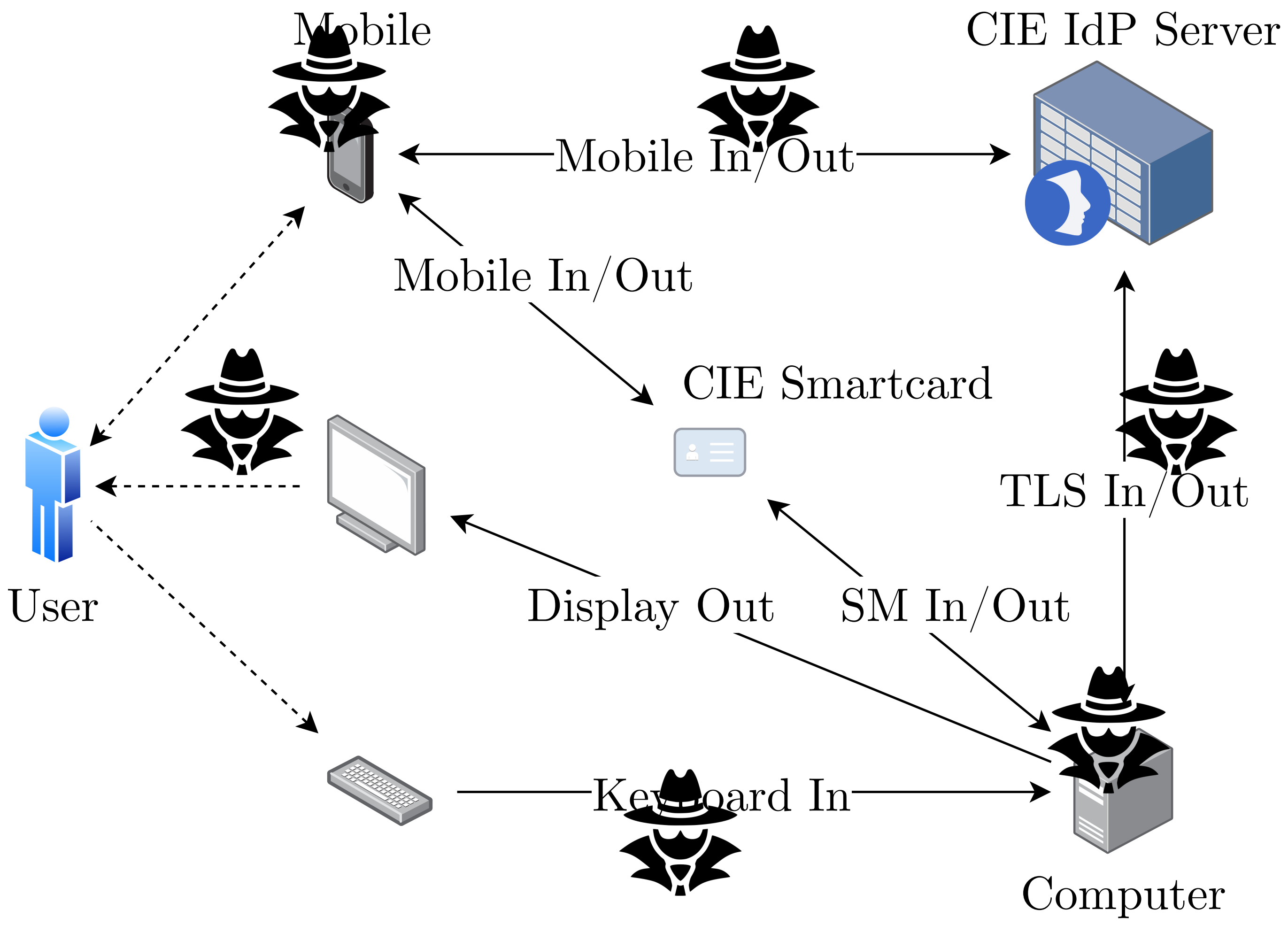


Sequence diagram for L2PSH


Keyboard
Screen
Computer
Mobile (CieID App)
CIE IdP Server
SP Server
SP Server
SP Server
...
...
SAMLRequest, sign(hash(SAMLRequest), SPServerKey)
SAMLRequest = <opId, level, SPName, ...>
SAMLRequest, sign(hash(SAMLRequest), SPServerKey)
SPInfo = <SPName, SPLogo>
opId, level, SPInfo
request username, password for level {level} login to {SPName}
request username, password for level {level} login to {SPName}
username, password
username, password
username, password
status
request CieID push login
request CieID push login
new OTP
opId, aenc(OTP, pk(pushKey))
request app PIN input
app PIN
opId, hash(OTP), sign(hash(hash(OTP)), deviceKey)
statusInfo
status, session, SPName
confirm login to {SPName}?
confirm login to {SPName}?
confirm login
confirm login
session, CONFIRM LOGIN
SAMLResponse
SAMLResponse
SAMLResponse = <authData, sign(hash(authData), CIEServerKey), ...>

Threat model

- **System compromise** by an attacker can be represented by the **level of control** that the attacker has acquired over the system's interfaces, potentially influencing data flow through inputs and outputs.
- This influence can be **partial:** the attacker may be able to read from or write to specific data streams within an interface, like its inputs or outputs.
- Many combinations → many possible attack scenarios to formalize and analyze!



Threat scenario notation

Given a system with interfaces $\{if_1, \ldots, if_n\}$, we denote the level of control exercised by an attacker as a set

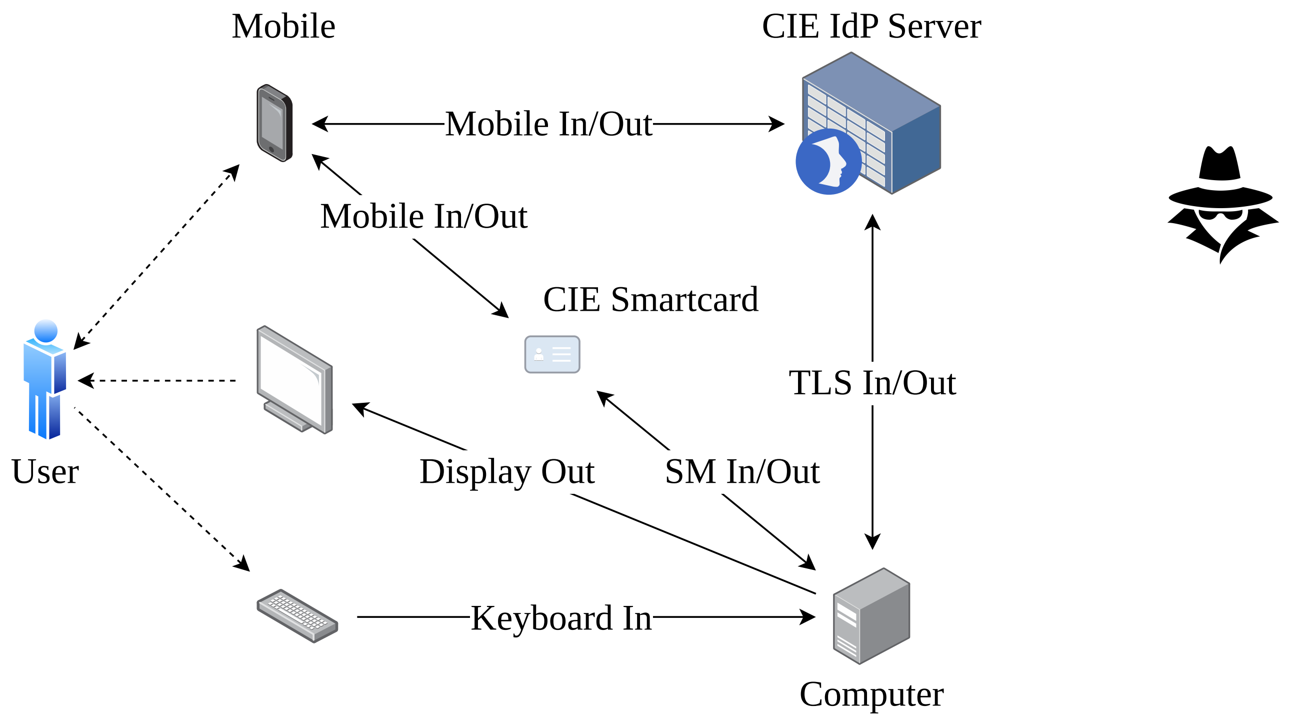
$$\{\mathscr{A}^{if_1}_{dir_1:acc_1}, \ldots, \mathscr{A}^{if_n}_{dir_n:acc_n}\}$$

where:

- $dir_i \in \{\text{in, out, io}\}$ represents the direction the attacker gains control over: the input (in), output (out) or both (io) of interface if_i
- $acc_i \in \{\text{RO,WO,RW}\}$ represents the access level acquired by the attacker on that interface and direction: read-only (RO), write-only (WO), or read-write (RW).

Threat scenarios: examples

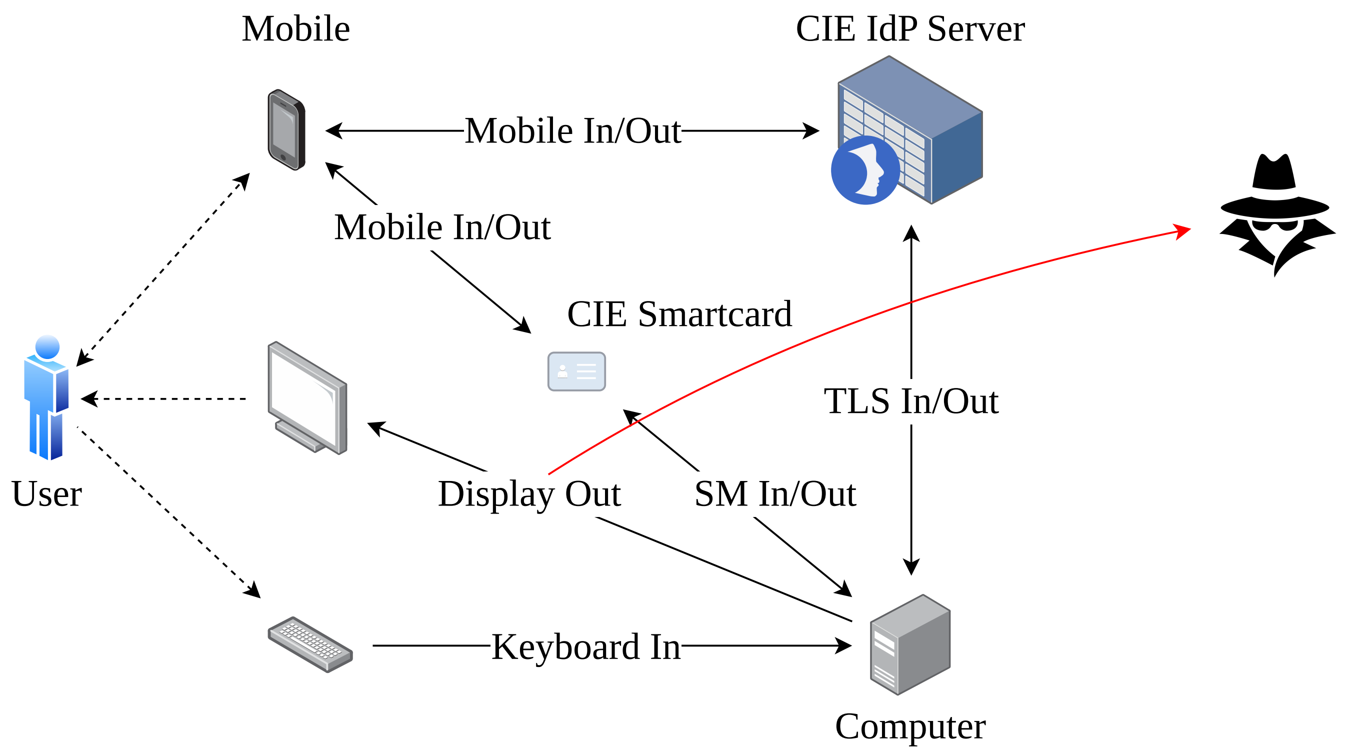
- Empty scenario: represents the uncompromised system



{}

Threat scenarios: examples

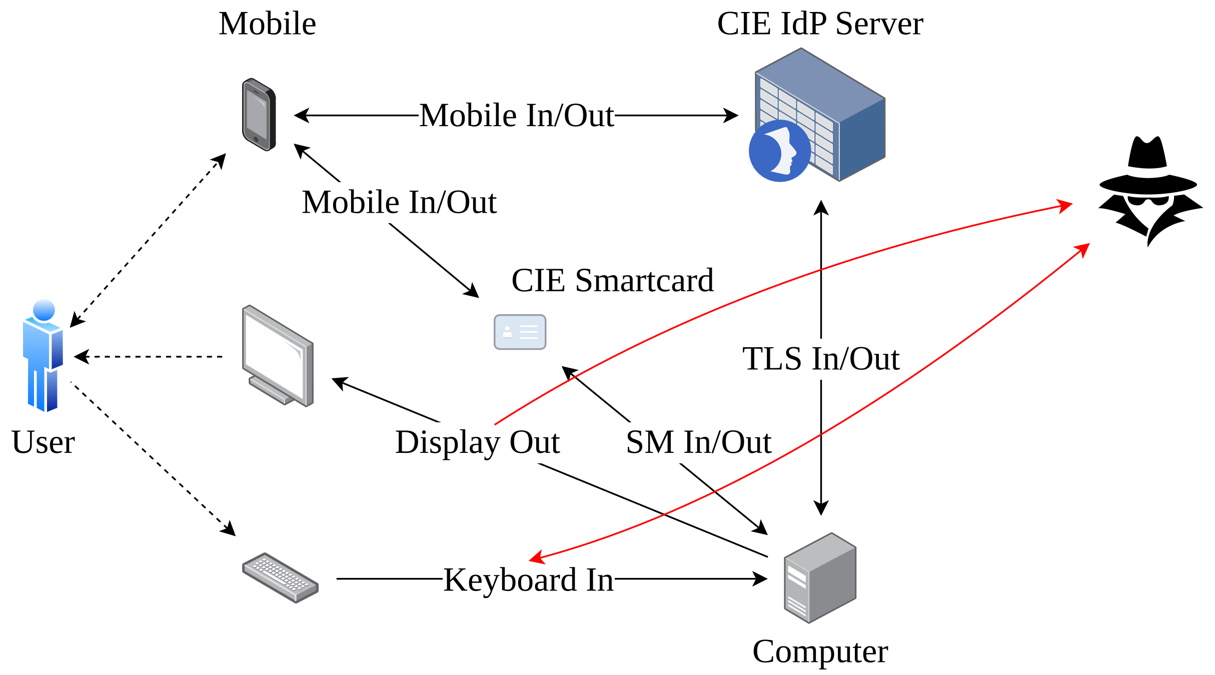
- the attacker can read what is shown on the computer's screen



$$\{\mathcal{A}^{\mathsf{displ}}_{\mathsf{out:RO}}\}$$

Threat scenarios: examples

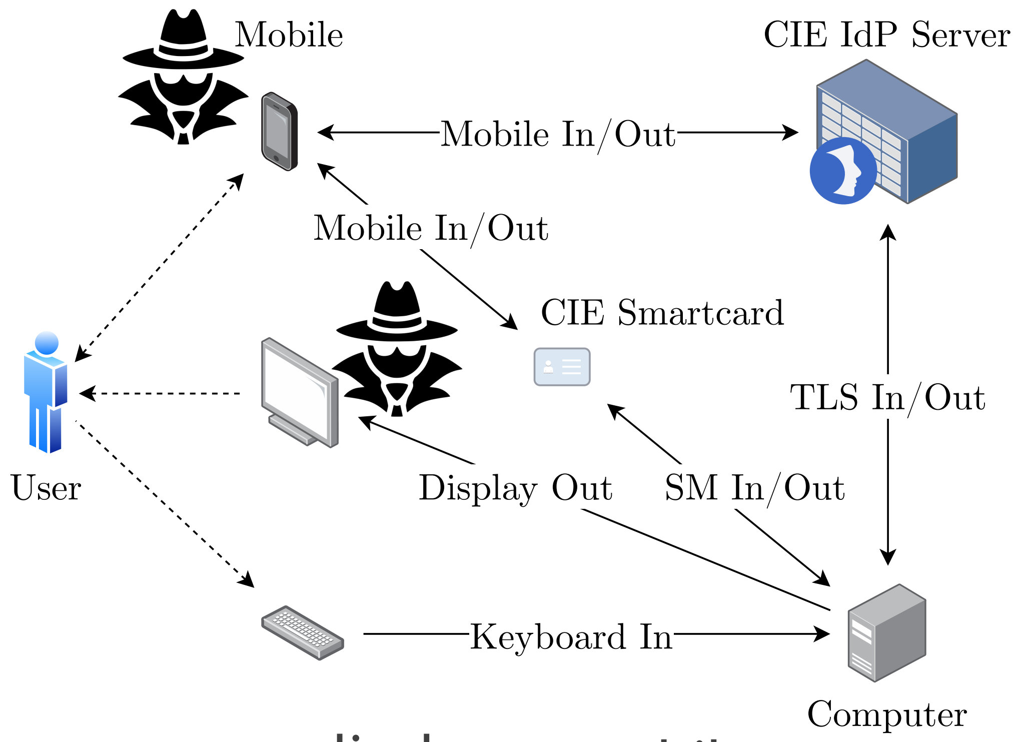
- the attacker can read what is shown on the screen, and can also read and modify the keyboard input of the computer (e.g. by means of some malware, or a USB keystroke attack (Rubber Ducky))



$$\{\mathcal{A}^{\text{displ}}_{\text{out:RO}}, \mathcal{A}^{\text{kbd}}_{\text{in:RW}}\}$$

Threat scenarios: examples

- the attacker has full control over the computer's display, to both read and manipulate what is shown on the screen, and can also passively observe messages received by the mobile device



$$\{\mathscr{A}^{\text{displ}}_{\text{out:RW}}, \mathscr{A}^{\text{mobile}}_{\text{in:RO}}\}$$

ProVerif Templates for CIE Level 2 Protocols

Each protocol L2PSH, L2SMS, L2QRC is formalized as a ProVerif *model template*:

- each possible attacker control and human threat is associated with a set of m4 preprocessor definitions inside this template
- Building a scenario means expanding these macros into the corresponding code snippets representing the vulnerability on that interface
    - e.g., if attacker has RO access to an interface, then every message that goes to that interface is also forwarded to the attacker

```
let Computer = (
          (* url, credentials provided by the user *)
          in(usb_kb_ch, (in_serv_id:id, login:bitstring, pwd:bitstring));
          _IN(`USB', (in_serv_id, login, pwd))

          (* send login credentials to server *)
          new se : TLS_session;
          out(public, (computer_id, se));

          _OUT(`TLS', (se, msg0, login, pwd))
          out(TLS(computer_id, in_serv_id), (se, msg0, login, pwd));

          _OUT(`DISPL', human_mobile_push_auth_request)
          out(display_ch, human_mobile_push_auth_request)
).
```

How many scenarios?

- All combinations may lead to a high number of attack scenarios
- An architecture with n interfaces may lead to up to 2^{4n} scenarios
  - In the CIE architecture (omitting Smartcard): 8 flows overall, 4 access levels each, yielding $4^8 = 65536$ combinations.
- How to reduce this search space? Let's observe that, usually,
  - Acquiring writing capability on a channel implies also reading capability → we can reduce the access levels to {RO,RW}
  - Gaining control over an interface's input is easier than controlling its output → an attacker that achieves RW control of an interface's output can also control its input

Strength ordering of access levels

- Formally: $\ell_1 \preceq \ell_2$ meaning “achieving the access level ℓ_1 implies achieving also the access level ℓ_2”
- Strength ordering: out:RW is stronger than both out:RO and in:RW; both out:RO and in:RW are stronger than in:RO

$$\begin{tikzcd}
 & \text{in:RW} \arrow[dr, phantom, "\prec" sloped] & & \\
\begin{array}{c}\text{out:RW}\\ = \\ \text{io:RW}\end{array} \arrow[ur, phantom, "\prec" sloped] \arrow[dr, phantom, "\prec" sloped] & & \text{in:RO} \arrow[r, phantom, "\prec"] & \top \\
 & \begin{array}{c}\text{out:RO}\\ = \\ \text{io:RO}\end{array} \arrow[ur, phantom, "\prec" sloped] & &
\end{tikzcd}$$

Strength ordering of attack scenarios

- This order can be extended to scenarios pointwise :

$$\mathcal{S}_1 \preceq \mathcal{S}_2 \iff \forall \mathcal{A}^{if}_{\ell_2} \in \mathcal{S}_2 \,.\, \exists \mathcal{A}^{if}_{\ell_1} \in \mathcal{S}_1 \,.\, \ell_1 \preceq \ell_2$$

- This means that
  - if a given protocol is proved secure in an attack scenario $\mathcal{S}_1$, then it is secure in every scenario $\mathcal{S}_2$ such that $\mathcal{S}_1 \preceq \mathcal{S}_2$;
  - Conversely, if a protocol is vulnerable in an attack scenario $\mathcal{S}_2$, then it is vulnerable in every scenario $\mathcal{S}_1$ such that $\mathcal{S}_1 \preceq \mathcal{S}_2$
- This allows to reduce the number of scenarios to consider to 1125

Further reducing attack scenarios

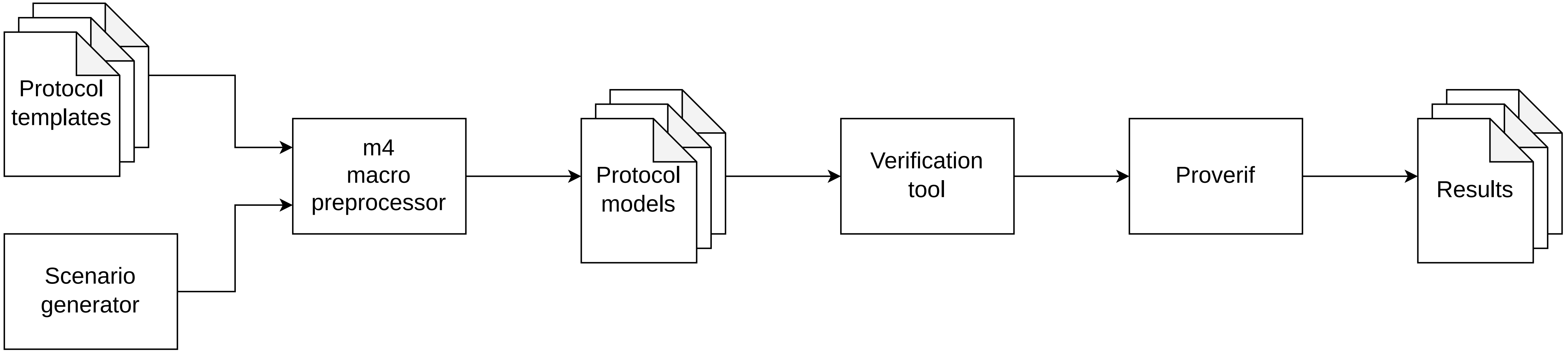
- We can further reduce the search space by noticing that for TLS channels, the only relevant access levels are io:RO and io:RW
- This reduces to 135 possible scenarios for the CIE architecture
- Then we have to consider *phishing attacks*:
  - users might be tricked into granting access to fake versions of services
  - This can be modeled by assuming that adversary has control of the target server
- Overall, we have 270 scenarios to consider (instead of 128k)

Actual numbers of theoretical and reduced scenarios

Protocol	pc-kbd	pc-displ	pc-tls	pc-sm	mob-hum	mob-sms	mob-psh	mob-tls	mob-sm	Theoretical number of scenarios	Reduced number of scenarios
L2SMS	✓	✓	✓		✓	✓				16 384	405
L2PSH	✓	✓	✓		✓		✓	✓		262 144	1215
L2QRC	✓	✓	✓		✓			✓		65 536	405
L3APP	✓	✓	✓		✓			✓	✓	1 048 576	1215
L3DSK	✓	✓	✓	✓						4096	81
Total										**1 396 736**	**3321**

Automatic verification in all scenarios

- We use the m4 macro preprocessor to instantiate the templates to actual models.
- The verification tool invokes Proverif on the models, dynamically excluding the verification of already implied scenarios.



Automatic exploration of scenario space: results

Uh, oh 😱

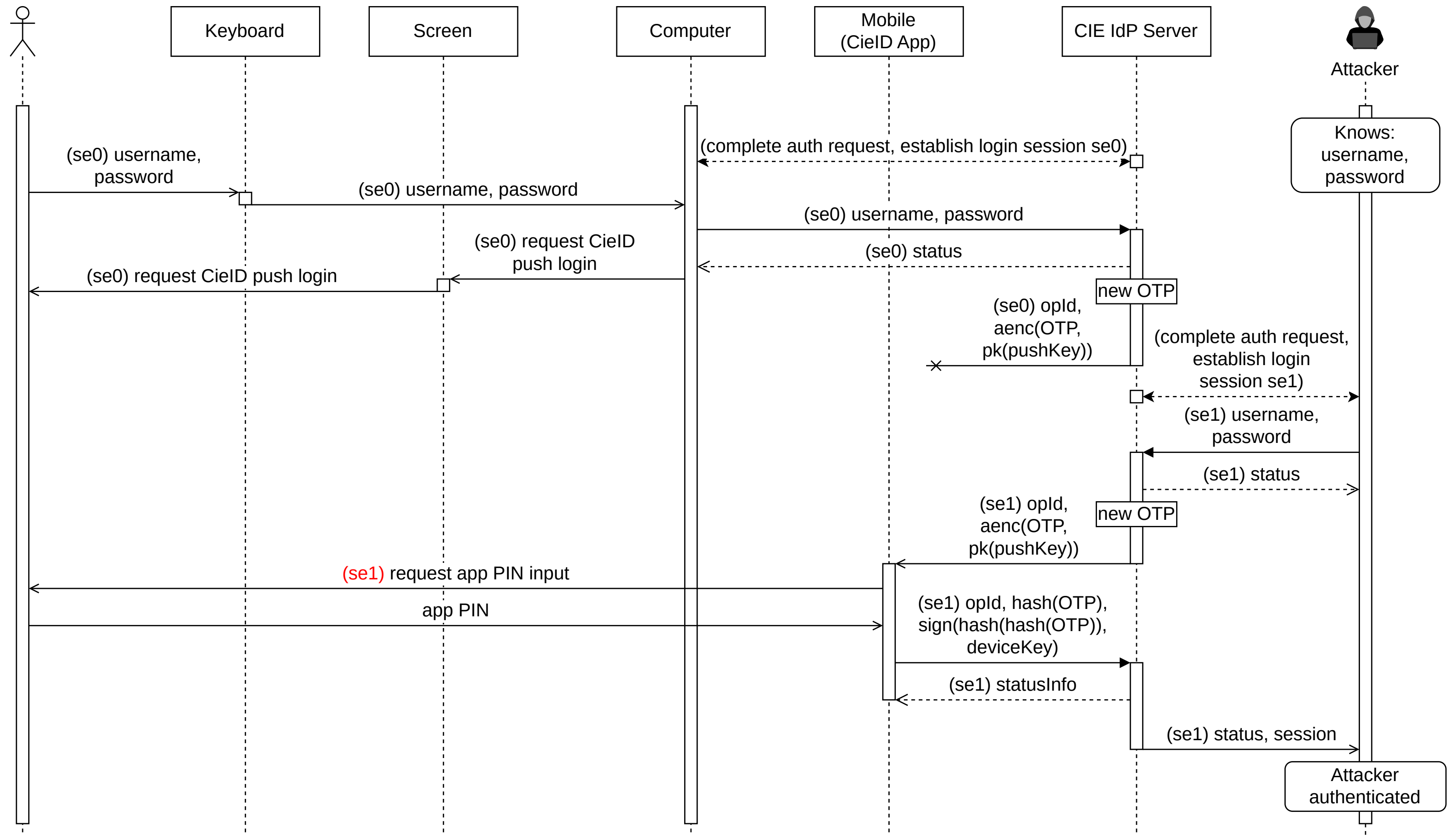
Threat Scenario					L2SMS	L2QRC	L2PS
					✓	✔	✗
				$\mathcal{A}^{\text{mobile}}_{\text{in:RO}}$	✗	✔	✘
				$\mathcal{A}^{\text{mobile}}_{\text{in:RW}}$	✘	✗	✘
			$\mathcal{A}^{\text{displ}}_{\text{out:RO}}$		✗	✔	✘
			$\mathcal{A}^{\text{displ}}_{\text{out:RW}}$		✘	✗	✘
		$\mathcal{A}^{\text{kbd}}_{\text{in:RO}}$			✗	✔	✘
		$\mathcal{A}^{\text{kbd}}_{\text{in:RW}}$			✘	✗	✘
	$\mathcal{A}^{\text{tls}}_{\text{io:RO}}$				✗	✔	✘
	$\mathcal{A}^{\text{tls}}_{\text{io:RO}}$	$\mathcal{A}^{\text{kbd}}_{\text{in:RO}}$	$\mathcal{A}^{\text{displ}}_{\text{out:RO}}$	$\mathcal{A}^{\text{mobile}}_{\text{io:RO}}$	✘	✓	✘
	$\mathcal{A}^{\text{tls}}_{\text{io:RW}}$				✘	✗	✘
PH					✗	✗	✘

Results:

- ✓ security proven
- ✔ security proven in a stronger scenario
- ✗ attack found
- ✘ attack found in a weaker scenario

Attack to L2PSH (found by ProVerif)

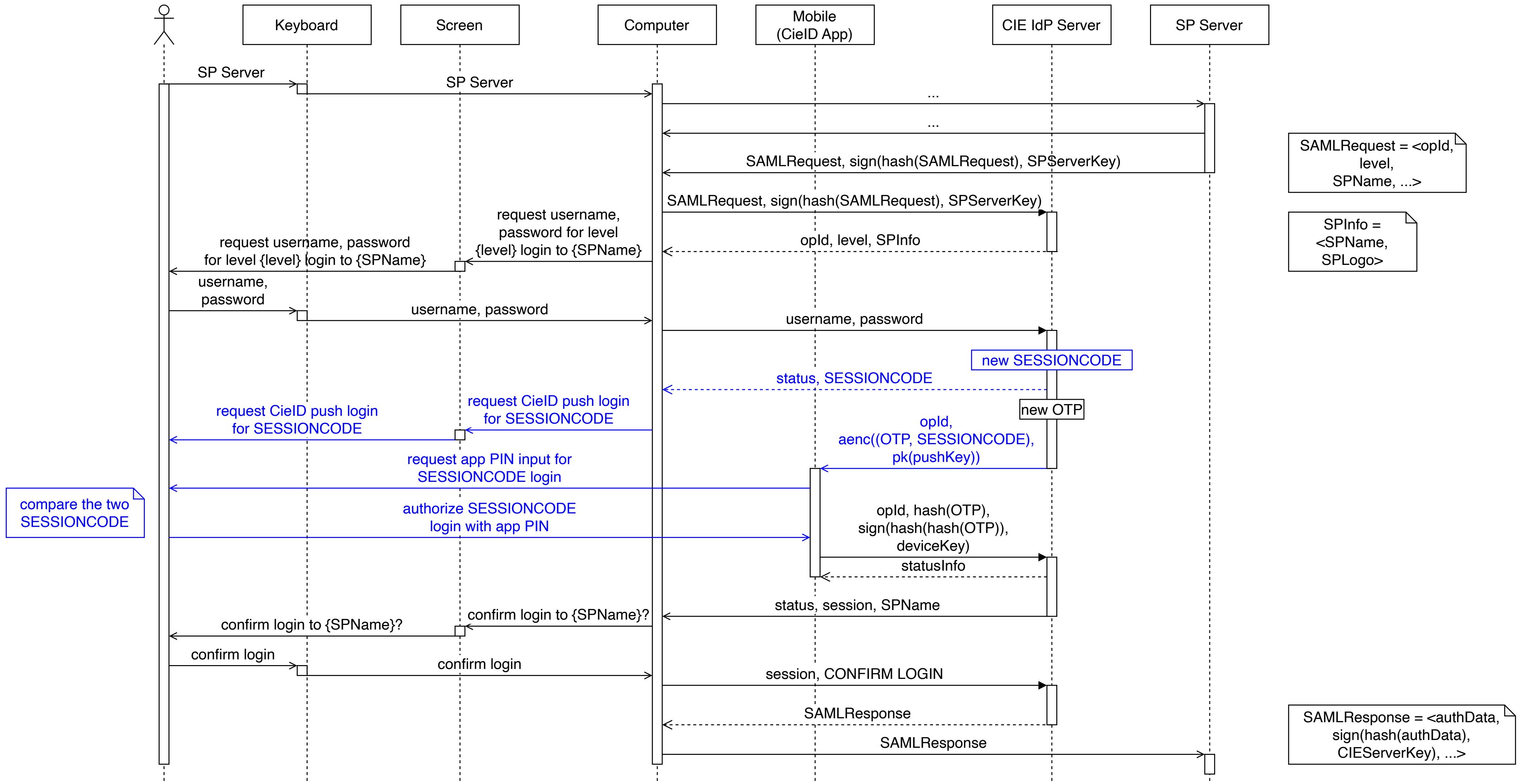
An attacker knowing only Level 1 credentials can complete a Level 2 login


Keyboard
Screen
Computer
Mobile (CieID App)
CIE IdP Server
Attacker
Knows: username, password
(complete auth request, establish login session se0)
(se0) username, password
(se0) username, password
(se0) username, password
(se0) status
(se0) request CieID push login
(se0) request CieID push login
new OTP
(se0) opId, aenc(OTP, pk(pushKey))
(complete auth request, establish login session se1)
(se1) username, password
(se1) status
new OTP
(se1) opId, aenc(OTP, pk(pushKey))
(se1) request app PIN input
app PIN
(se1) opId, hash(OTP), sign(hash(hash(OTP)), deviceKey)
(se1) statusInfo
(se1) status, session
Attacker authenticated

Fixing the L2PSH protocol: L2PSH*

- Vulnerability: there is no link between the message shown by the CieID app and the session the user is trying to initiate
- Actually implemented in a POC
  - Tested only on our CIE cards! 😏
- How to fix the protocol?
  - add some unique session identifier that the user has to check before authorizing login

Fixing the L2PSH protocol: L2PSH*


Keyboard
Screen
Computer
Mobile (CieID App)
CIE IdP Server
SP Server
SP Server
SP Server
...
...
SAMLRequest, sign(hash(SAMLRequest), SPServerKey)
SAMLRequest = <opId, level, SPName, ...>
SAMLRequest, sign(hash(SAMLRequest), SPServerKey)
SPInfo = <SPName, SPLogo>
request username, password for level {level} login to {SPName}
request username, password for level {level} login to {SPName}
opId, level, SPInfo
username, password
username, password
username, password
new SESSIONCODE
status, SESSIONCODE
new OTP
request CieID push login for SESSIONCODE
request CieID push login for SESSIONCODE
opId, aenc((OTP, SESSIONCODE), pk(pushKey))
request app PIN input for SESSIONCODE login
compare the two SESSIONCODE
authorize SESSIONCODE login with app PIN
opId, hash(OTP), sign(hash(hash(OTP)), deviceKey)
statusInfo
status, session, SPName
confirm login to {SPName}?
confirm login to {SPName}?
confirm login
confirm login
session, CONFIRM LOGIN
SAMLResponse
SAMLResponse = <authData, sign(hash(authData), CIEServerKey), ...>
SAMLResponse

Analysis after fixing: L2PSH* equivalent to L2QRC

Threat Scenario	L2SMS	L2SMS*	L2PSH	L2PSH*	L2QRC
	✓	✓	✗	✓	✓
$\mathcal{A}^{\text{mobile}}_{\text{in:RO}}$	✗	✗	✗	✓	✓
$\mathcal{A}^{\text{mobile}}_{\text{in:RW}}$	✗	✗	✗	?	✗
$\mathcal{A}^{\text{mobile}}_{\text{io:RW}}$	✗	✗	✗	✗	✗
$\mathcal{A}^{\text{displ}}_{\text{out:RO}}$	✗	✓	✗	✓	✓
$\mathcal{A}^{\text{displ}}_{\text{out:RW}}$	✗	✓	✗	✗	✗
$\mathcal{A}^{\text{kbd}}_{\text{in:RO}}$	✗	✗	✗	✓	✓
$\mathcal{A}^{\text{kbd}}_{\text{in:RW}}$	✗	✗	✗	✗	✗
$\mathcal{A}^{\text{tls}}_{\text{io:RO}}$	✗	✗	✗	✓	✓
$\mathcal{A}^{\text{tls}}_{\text{io:RO}}$ $\mathcal{A}^{\text{kbd}}_{\text{in:RO}}$ $\mathcal{A}^{\text{displ}}_{\text{out:RO}}$ $\mathcal{A}^{\text{mobile}}_{\text{io:RO}}$	✗	✗	✗	✓	✓
$\mathcal{A}^{\text{tls}}_{\text{io:RW}}$	✗	✗	✗	✗	✗
NC	✴	✴	✴	✗	✴
PH	✗	✗	✗	✗	✗

Results:

- **?** ProVerif does not reach a conclusive answer, failing with "Cannot be proven" (but it is safe to assume it is ✗)
- ✴ threat scenario not applicable to the protocol

Scenarios:

- NC: no-check scenario (the user skips the verification of the session identifier)

Summary

- Methodology has been successfully applied to the analysis of a complex case (many components, including human participants)
- Extended with a threat model based on interfaces which covers hundreds (actually, thousands) of possible attack scenarios
- Automatic and systematic generation of attack scenarios
- Heuristics for reducing search space
- We have analyzed also Level 3 protocols, involving interactions with the smart-card
- Contacted IPZS, they are fixing the CieID app
- Can be ported to other eIDAS-compliant cards and protocols - and, more generally, other heterogeneous architectures

References

- [Dumortier 2017] Dumortier, J.: Regulation EU no 910/2014 on electronic identification and trust services for electronic transactions in the internal market (eIDAS regulation). In: EU Regulation of E-Commerce, pp. 256–289. Edward Elgar Publishing (2017)
- [Jacomme, Kremer, 2021] Jacomme, C., Kremer, S.: An extensive formal analysis of multi-factor authentication protocols. ACM Transactions on Privacy and Security 24, 1–34 (2021)
- [Lips et al., 2020] Lips, S., Bharosa, N., Draheim, D.: eIDAS implementation challenges: The case of Estonia and the Netherlands. In: Electronic Governance and Open Society: Challenges in Eurasia. pp. 75–89. Springer (2020)
- [Paier et al. 2024] Paier, M., Van Eeden, R., Miculan, M.: Formal Analysis of Multi-Factor Authentication Schemes in Digital Identity Cards. In: Proc. Software Engineering and Formal Methods, Springer (2024)
- [Somorovsky et al., 2011] Somorovsky, J., Heiderich, M., Jensen, M., Schwenk, J., Gruschka, N., Lo Iacono, L.: All your clouds are belong to us: security analysis of cloud management interfaces. In: Proc. 3rd ACM workshop on Cloud computing security. pp. 3-14 (2011)