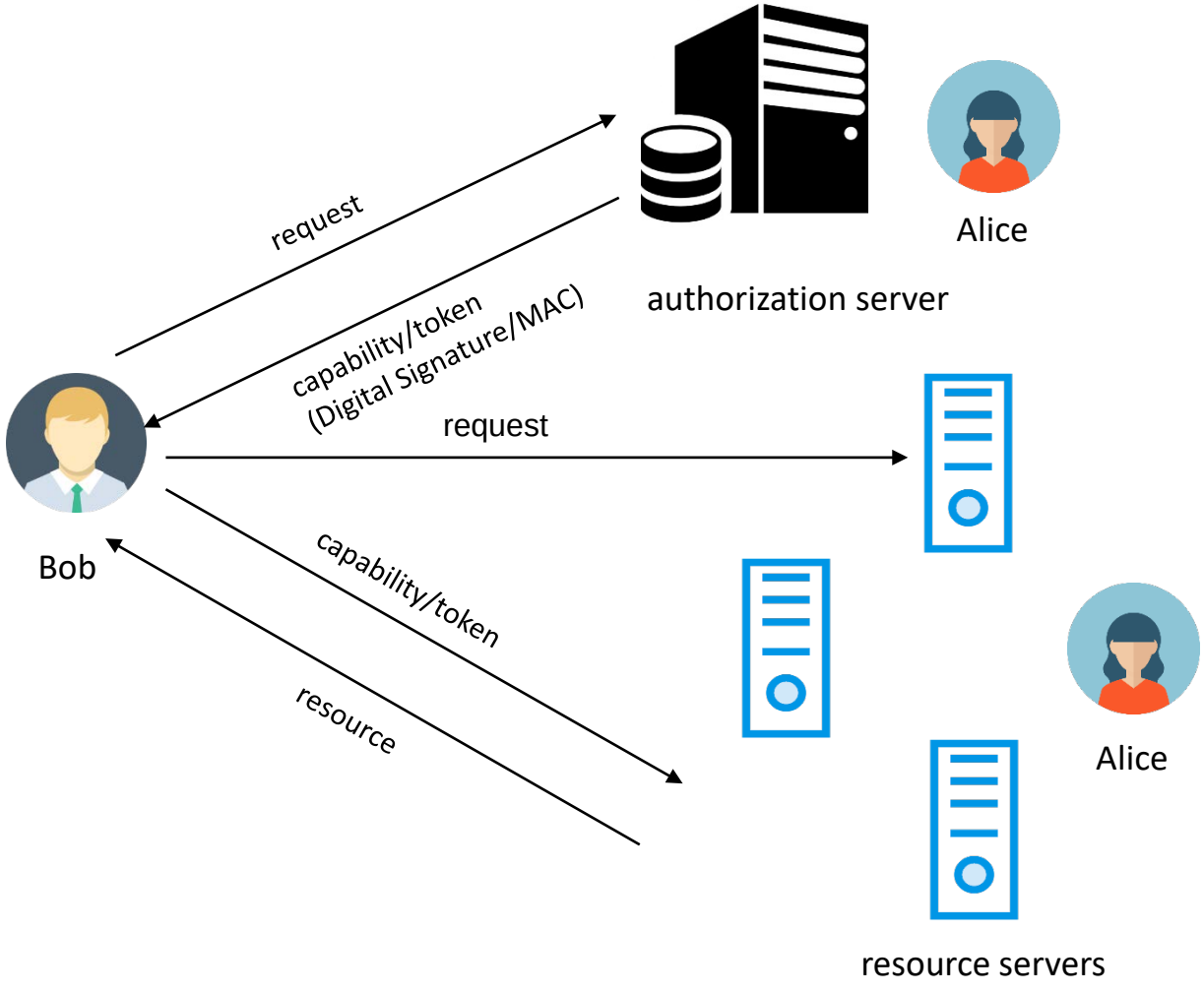
A Capability-based System to Enforce Context-aware Permission Sequences

Shuai Li

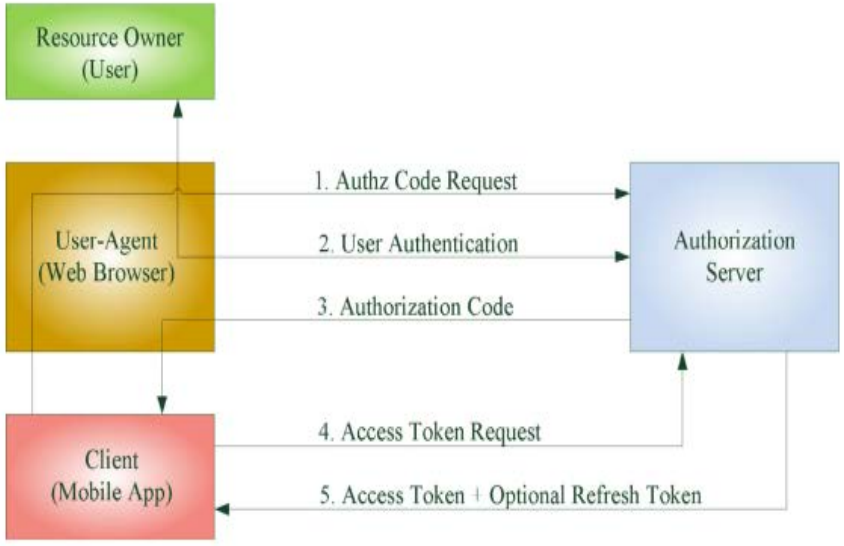
Capability-based decentralized authorization

Protocols include OAuth 2.0 [H,2012], UMA [MCMH, 2016], ICAP [G,1989]



OAuth and Proof of Possession Token

Typical OAuth flow



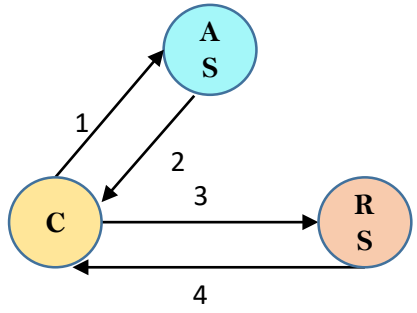
Two Legged OAuth*

1. $C \rightarrow AS: ID_C, \text{credentials}, ID_{RS}$
2. $AS \rightarrow C: Token$
3. $C \rightarrow RS: ID_C, Token$

$$Token = \left(t, auth_k(t)\right)$$

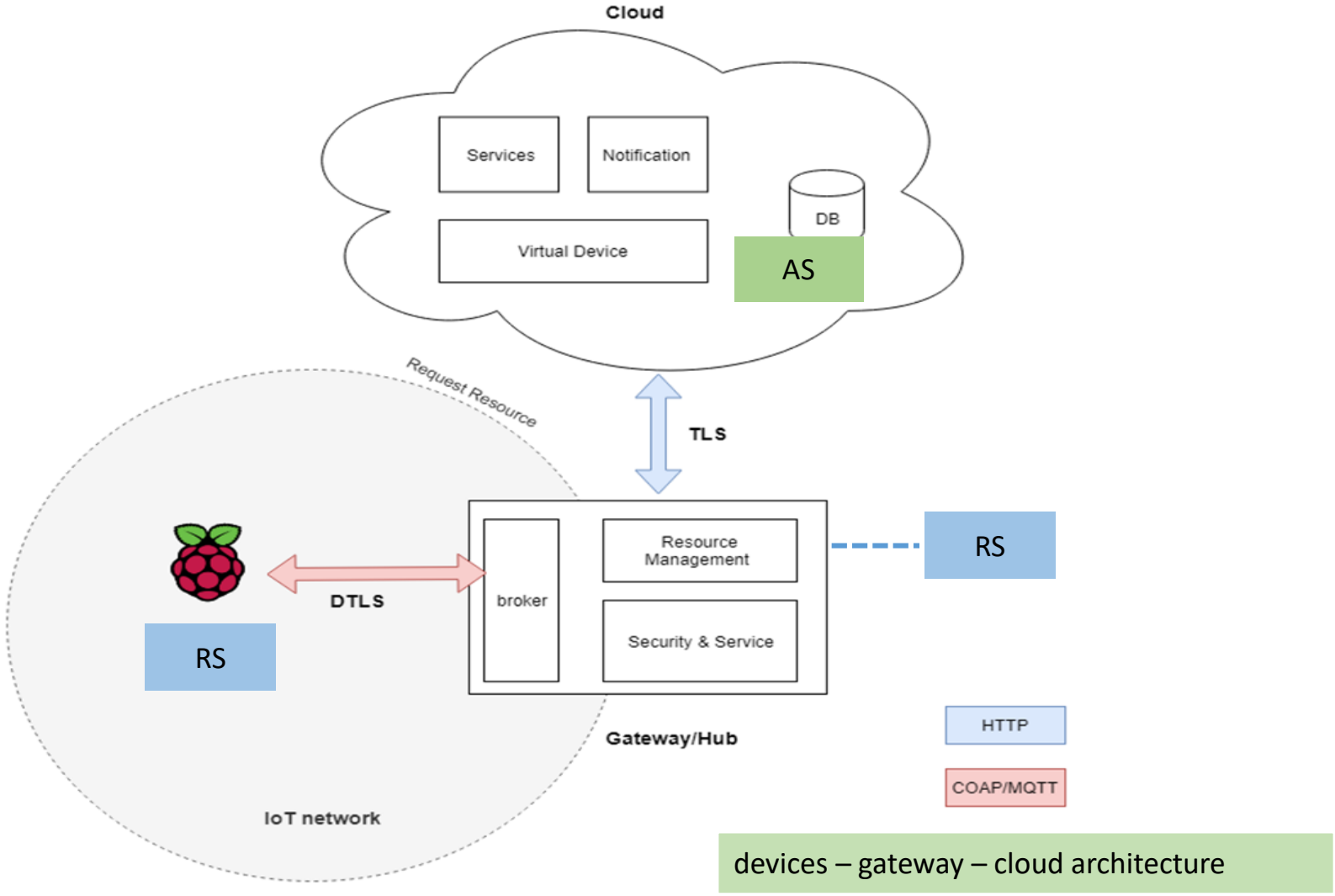
$$t = (ID_C, ID_{RS}, P, exp)$$

P: permissions granted to the C
Exp: expired time
K: secret key



* Simpler flow by eliminating the redirection

Capability-based authorization solution for IoT



devices – gateway – cloud architecture

Access patterns in IoT

- Access depends on locally verifiable conditions
  - Time of request
  - IP address of Bob
  - Supported by systems in [GPR,2013], [HJMS, 2016]

- Access depends on environmental situations
  - The camera is open only when the user is not home

- Permissions are allowed only when the client follows an ordered permission sequence
  - The visitor can access the printer for no more than 5 times
  - The devices are accessible only when the user follows a particular sequence [TFS, 2018]

- A combination of the above patterns
  - The children can watch the TV no more than two times during a day, with no more than 1 hour for each access

Our contributions

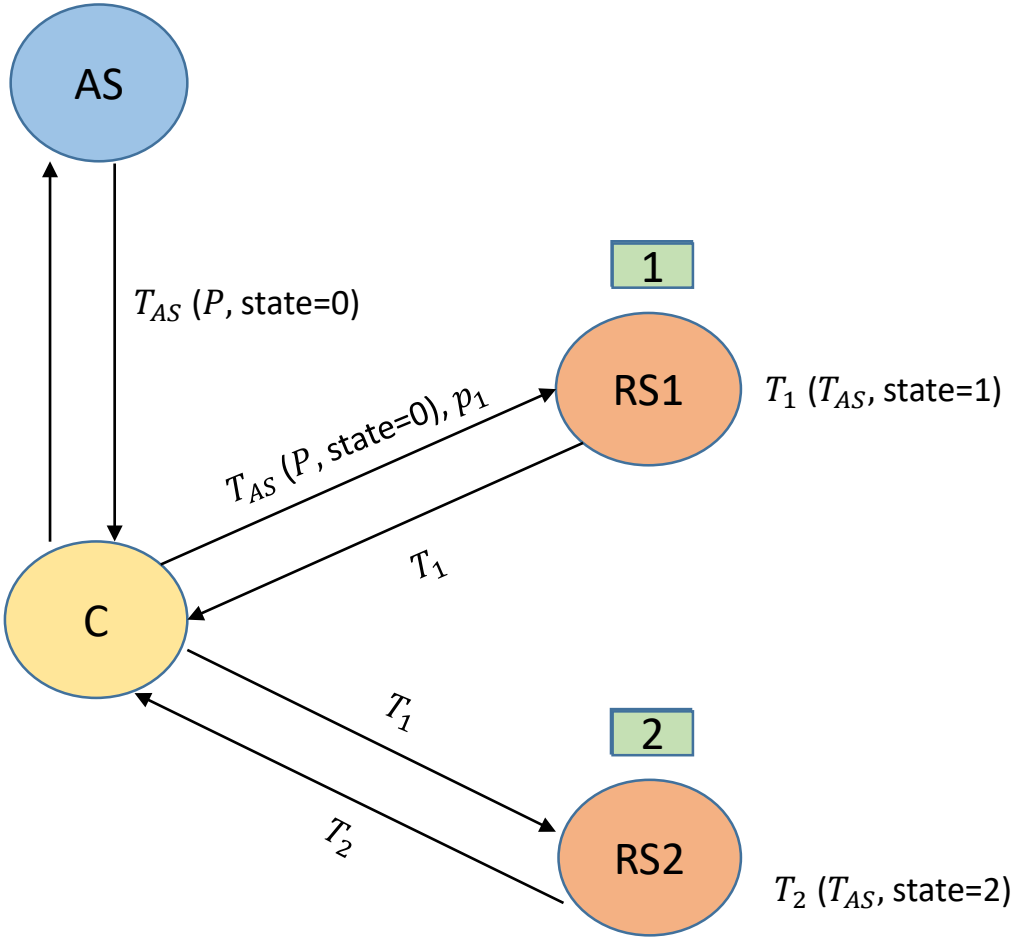
- Theoretical
  - Proposed an efficient method of enforcing permission sequences with proof
    - HCAP supports history-based access control [TFS, 2018]
    - Less overhead, context-aware
  - Enforcing environmental context/situations
    - Used ESO in [SST, 2018] as a protected point of contact for evaluating environmental situation
    - Efficient interactions with ESO

- Griffin System Design
  - Incorporated our model in the OAuth framework using proof-of-possession token and attribute-based access control model
  - Deployed Griffin for sharing patient health data

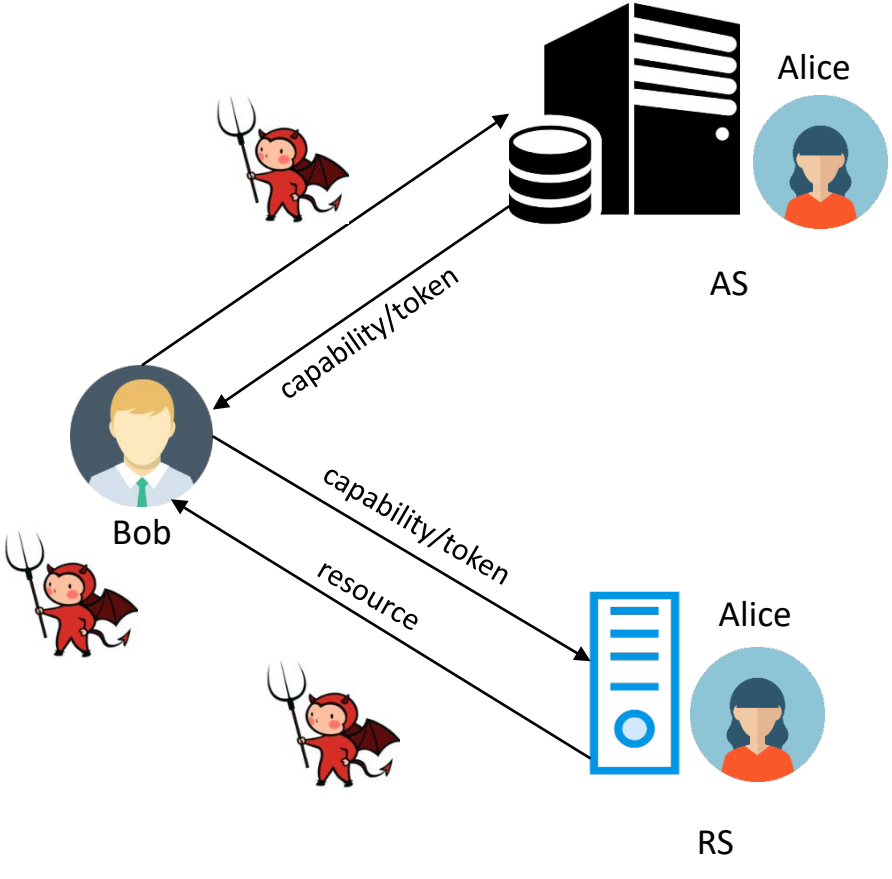
- Performance Evaluation
  - Competitive performance compared with OAuth 2.0

Permission Sequence

$P: p_1 p_2 p_3 p_4 \ldots p_n$

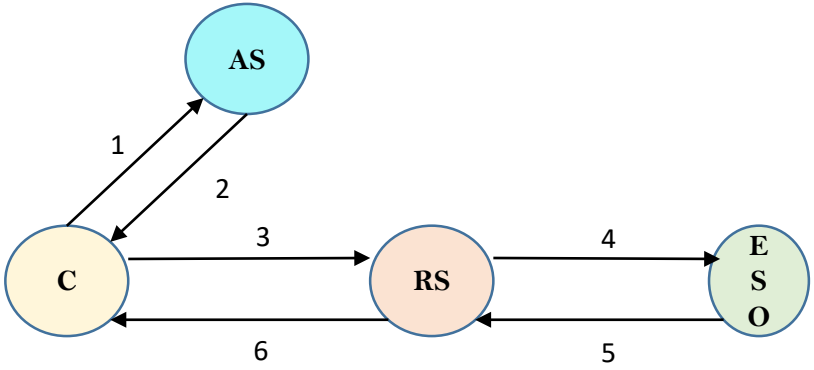


Adversary model and Attacks



- Capability forgery and tampering
  - Digital Signature
- Capability theft
  - Proof of possession capability
- Client Impersonation
  - Public-key based client authentication
- Replay attack
  - Proof of safety property

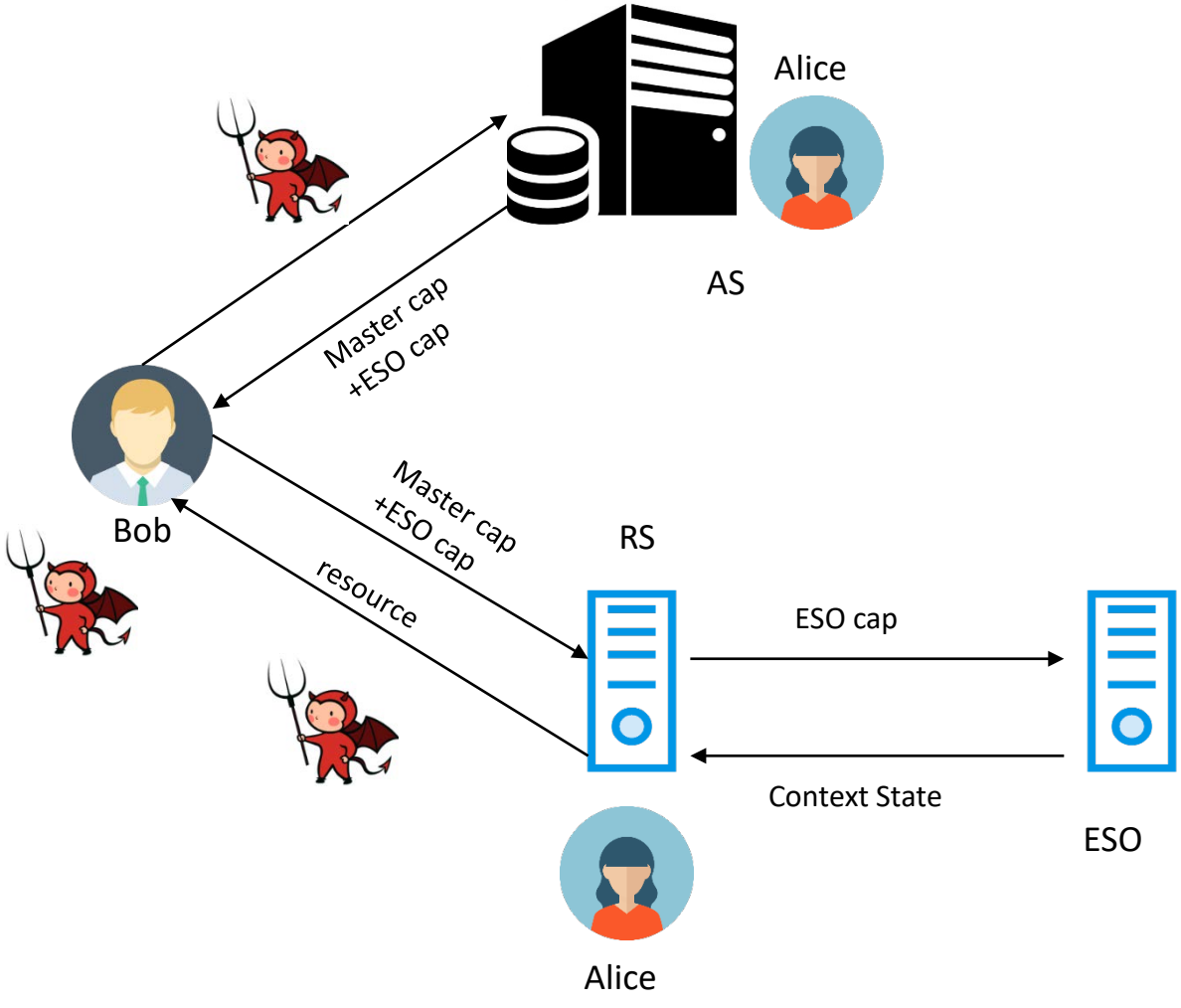
Context Awareness



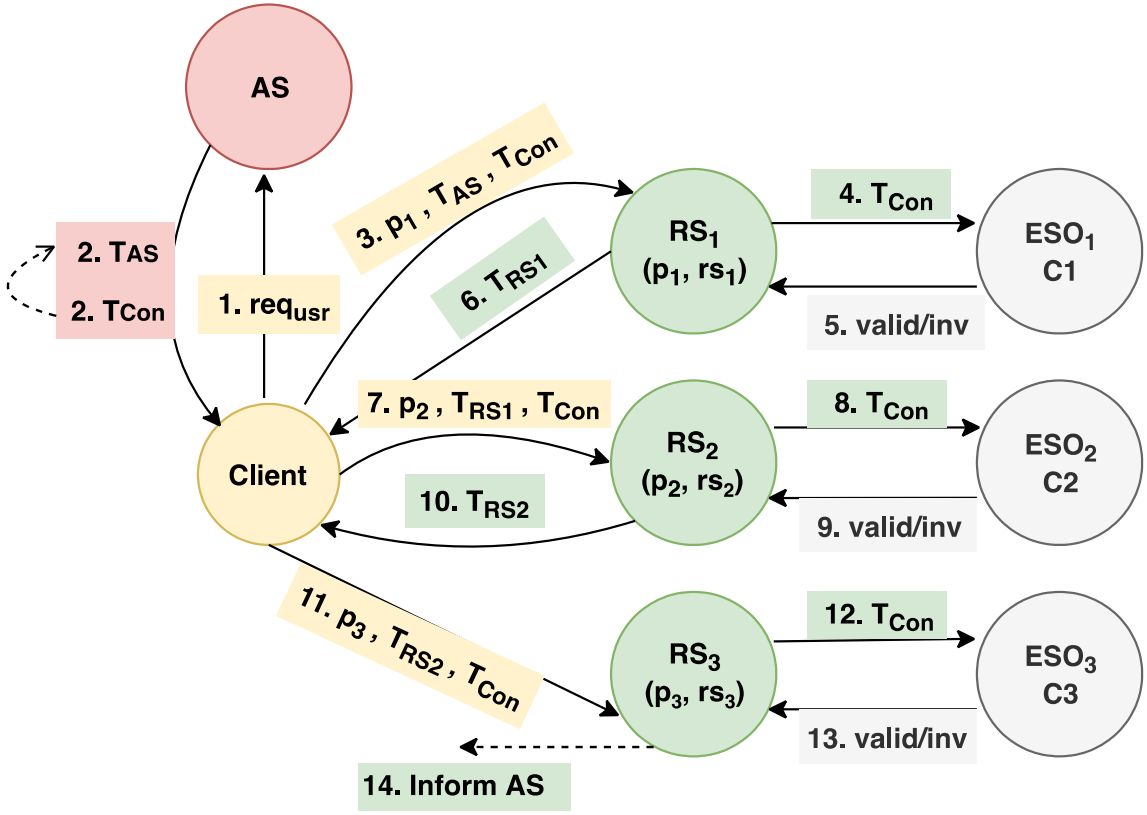
ESO: environmental situation oracle [SST, 2018]

1. Request master capability and ESO capability
2. Get capabilities
3. Request for service by presenting capabilities together
4. Request for situation state using ESO capability
5. Return ESO state Y/N
6. Provide service/return failure

Adversary model and Attacks

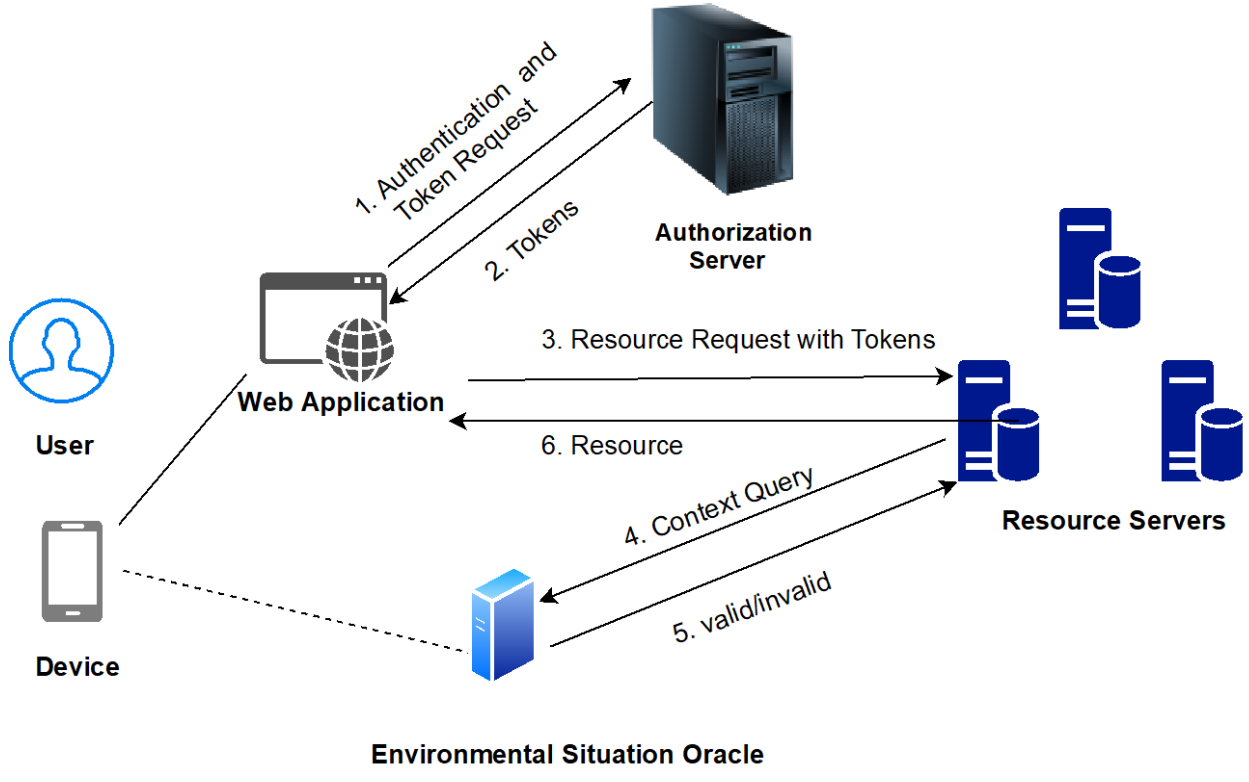


Context-aware permission sequence



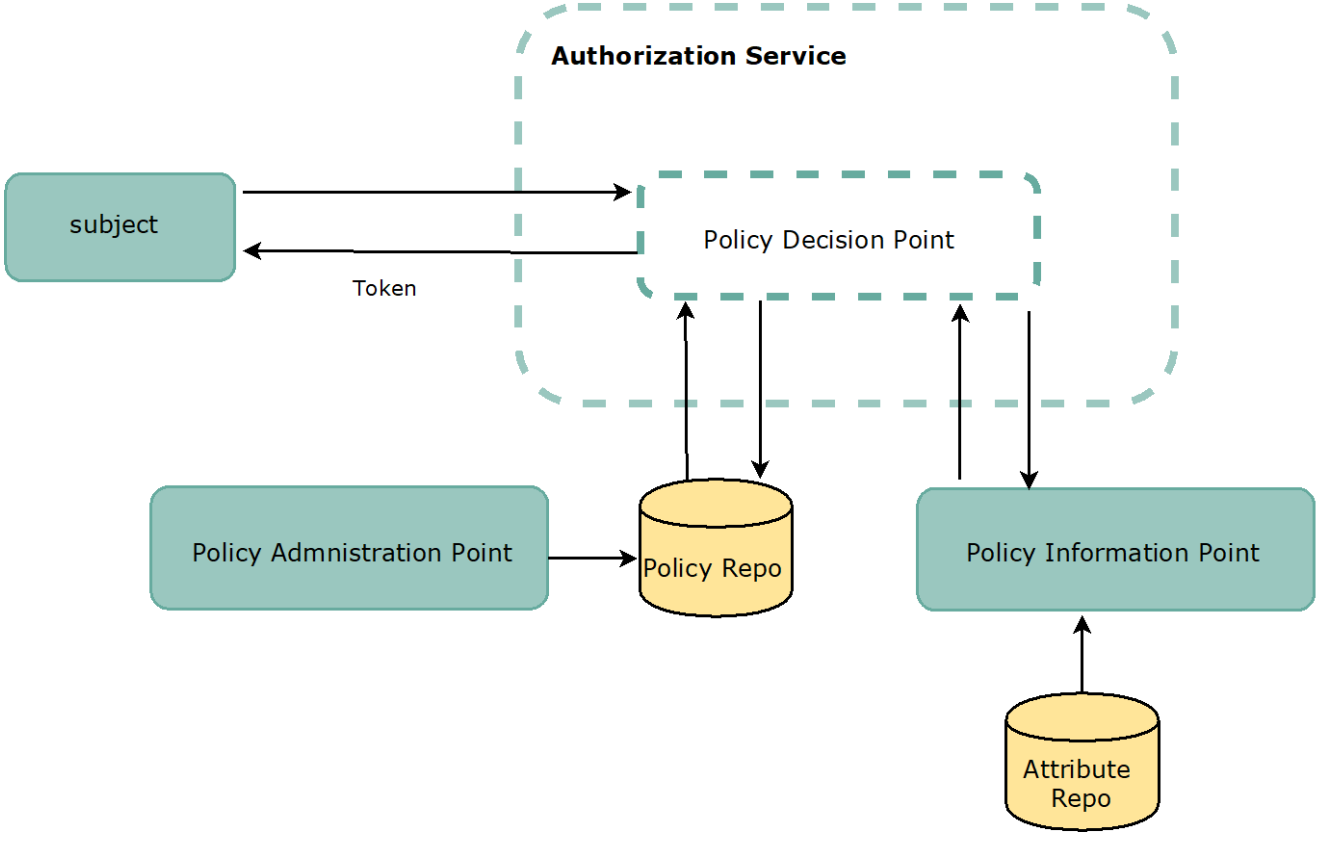
- ⟶ fast revocation
- ⟶ verifiable integrity
- ⟶ inability to use the capability other than the intended possessor.
- ⟶ inability to violate the permission sequence by replaying capabilities.

Griffin System Design (OAuth Extension)

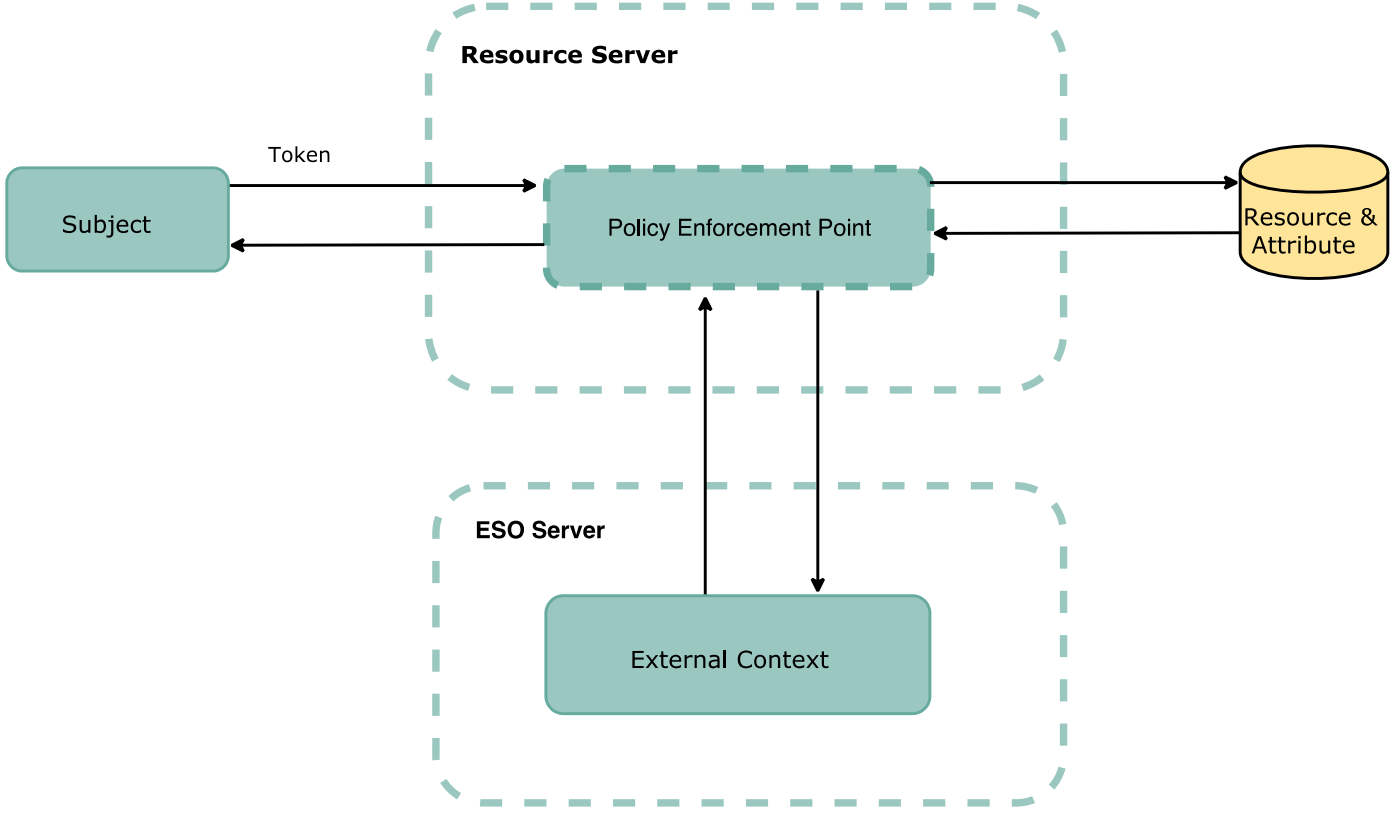


- ➞ Reusable AS: Node.js, Express, MongoDB, ABAC, JWT
- ➞ RS and ESO: Node.js, Express, MongoDB
- ➞ Middleware : can be added into any RS restful API to enable Griffin compliant RS.
- ➞ Client application: Postman, Apache JMeter

Griffin system implementation architecture



Griffin system implementation architecture



Use Case Demonstration in Health Care

Policy Language Model:

<policy> ::= <rule> | <rule> <policy>

<rule> ::= <subjectAttributes>

<objectAttributes>

<authorization>

<actionAttributes>

<environmentContext>

<Default>

Rules: All Nurse and Doctor practitioners in the Cardiology Department can view the Medical Records of Heart Patient only when they are at the hospital

```
{
  "rules": {
    "SubjectAttribute": {
        "role":["Doctor","Nurse"],
        "department":"Cardiology"
    },
    "ObjectAttribute": {
        "resourceType":["Heart"]
    },
    "authorization": "permit",
    "actionAttribute": {
      "actions": ["view"]
    },
    "environmentContext": ["clientlocationhospital"],
    "Default": {
      "authorization": "Deny"
    }
  }
}
```

Future Work

- Privacy-preserving capability-based system to enforce context-aware permission sequence
  - The Clients interact with the AS and the RS under real identities. The identity of the end user is disclosed to the AS to receive a capability, and to the RS to obtain the resource.
  - Solution: Pseudonym system, anonymous credential system [CL, 2001]
  - Alice obtains the capability from the AS without revealing anything more than the fact that she owns the pseudonym.
  - We also want to allow Alice to be anonymous when proving the possession of this capability to the RS.
  - [CL, 2001] proposed a anonymous credential system that can be incorporated with our system.
  - How to incorporate the credential into the capability?

References

- [TFS, 2018] L. Tandon, P. W. Fong, and R. Safavi-Naini. Hcap: A history-based capability system for iot devices. In Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, pages 247–258. ACM, 2018.
- [GPR,2013] S. Gusmeroli, S. Piccione, and D. Rotondi. A capability-based security approach to manage access control in the internet of things. Mathematical and Computer Modelling, 58(5-6):1189–1205, 2013.
- [HJMS, 2016] J. L. Hernández-Ramos, A. J. Jara, L. Marín, and A. F. Skarmeta Gómez. Dcapbac: embedding authorization logic into smart things through ecc optimizations. International Journal of Computer Mathematics, 93(2):345–366, 2016.
- [SST, 2018] R. Schuster, V. Shmatikov, and E. Tromer. Situational access control in the internet of things. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 1056–1073. ACM, 2018.
- [CL, 2001] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymouscredentials with optional anonymity revocation. In International Conference on the Theory and Applications of Cryptographic Techniques, pages 93–118. Springer, 2001.

Performance Evaluation

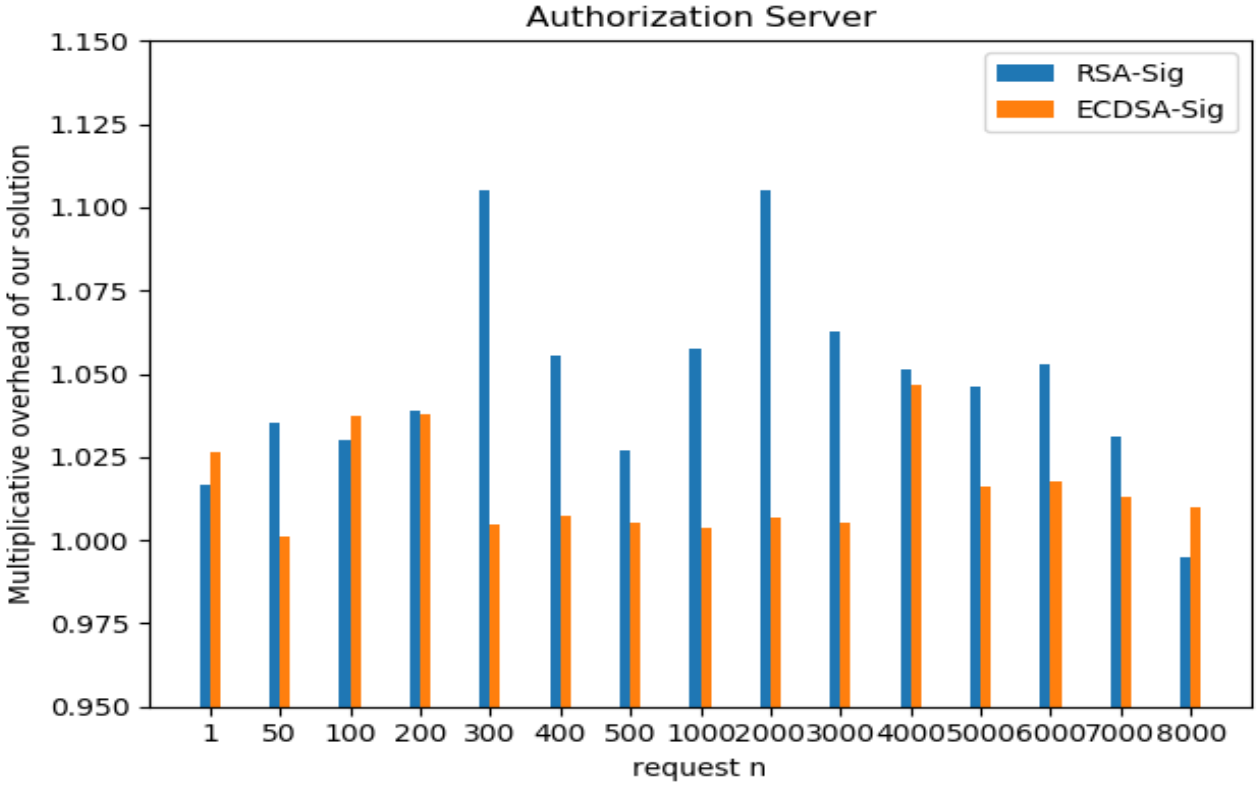
Goal:

- What is the load of our implemented AS, RS?
- How much is the overhead compared to OAuth 2.0 ?

Methodology:

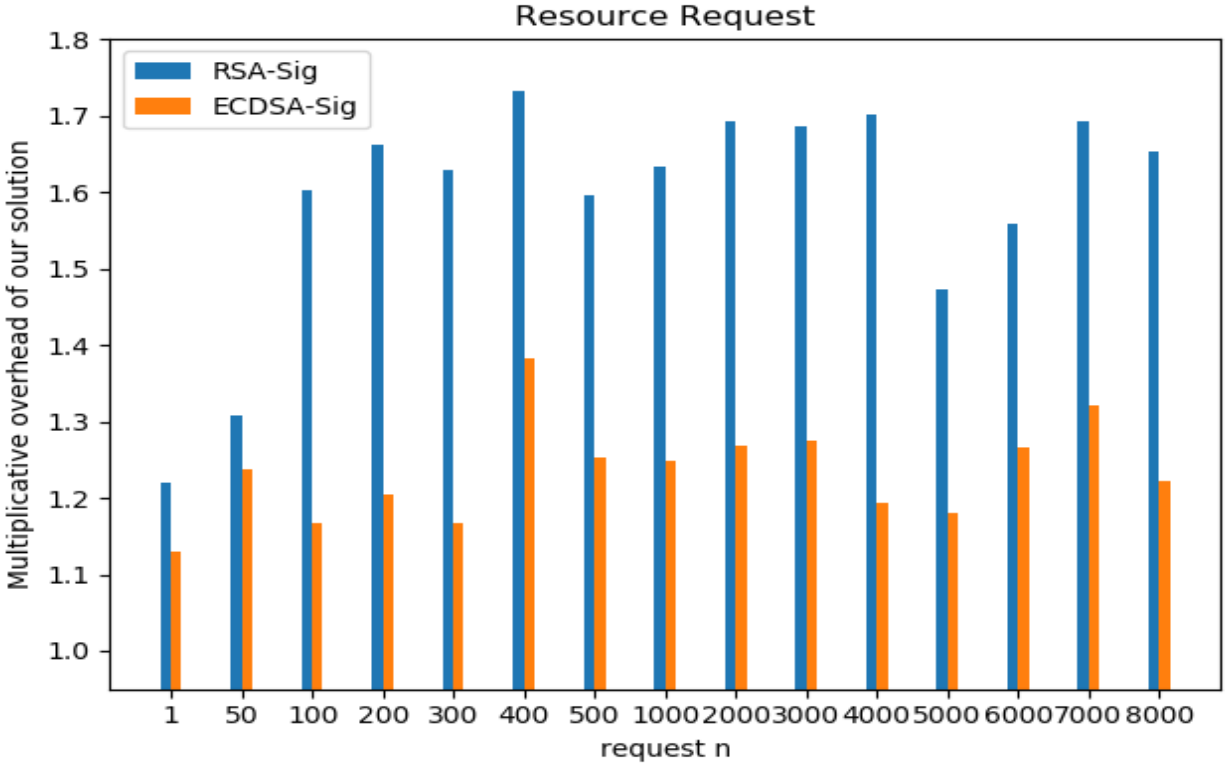
- N requests (till 8000) are sent to the AS and the RS in one second
- Measure the average round trip time for all the requests AND record the error rate.
- Single request to record the time-breakdown
- Conduct the experiment in Griffin and OAuth under two prevalent token signature algorithms.
  - RSA SHA 256 3072 , ECDSA P-256 SHA 256
- All communication channels are secured by TLSv1.2.
- Test is automated using Apache JMeter

Multiplicative overhead of AS: RTT in our solution compared with OAuth 2.0



- OAuth starts to have error when it hits 7000 requests with smaller error rate (0.84% for RSA-Sig, 0.72% for ECDSA-Sig).
- Our system starts to generates error when it hits 7000 requests (1.31% for RSA-Sig, 0.75% for ECDSA-Sig).
- the overhead is less than 5% in the ECDSA-Sig setting and 12% in the RSA-Sig setting.

Multiplicative overhead of RS: RTT in our solution compared with OAuth 2.0



- OAuth starts to have error when it hits 7000 with smaller error rate (1.31% for RSA-Sig, 0.75% for ECDSA-Sig).
- Our system generates error when it hits 7000 request (5.46% for RSA-Sig, 4.00% for ECDSA-Sig).
- The overhead for RSA-Sig and ECDSA-Sig is a small constant.

Publications

- S. Li, R. Safavi-Naini and P. W. Fong—A Capability-based System to Enforce Context-aware Permission Sequence, in review of the 2020 ACM ASIA Conference on Computer and Communications Security.

- S. Avizheh, R. Safavi-Naini and S. Li—Secure Logging with Security against Adaptive Crash Attack, Proceedings of the 2019 International Symposium on Foun- dations & Practice of Security (FPS), Nov 2019.

- T. T. Doan, R. Safavi-Naini, S. Li, S. Avizheh, M. Venkateswarlu K., and P. W. Fong—Towards a Resilient Smart Home (received best paper award), Proceedings of the 2018 ACM SIGCOMM Workshop on IoT Security and Privacy (IoT S&P), Aug 2018

Reference monitor and security automaton

- The reference monitor is the abstract machine mediating all the access requests. It authenticates the subject and evaluates the request against policies and additional information referred by the policies.

- The security automaton interprets the security policy and server as the basis of an enforcement mechanism in Execution Monitoring (EM). EM includes security kernels, reference monitors and other OS enforcement mechanisms.

Security automaton

They are similar to non-deterministic finite state automaton. Formally, a security automaton is defined by:

- a countable set Q of automaton states,
- a countable set $Q_0 \subseteq Q$ of initial automaton states,
- a countable set I of input symbols, and
- a transition function, $\delta: (Q \times I) \rightarrow 2^Q$

To process a sequence $s_1 s_2 s_3$ ... of input symbols. The current state Q of the security automaton starts equal to Q_0 and the sequence is read one symbol at a time. As each input symbol S_i is read, the security automaton changes Q to

$$\bigcup_{q \in Q'} \delta(q, s_i)$$

If Q' is the empty set, the input is rejected; otherwise, the input is accepted.

Context-aware protocol

1. $C \rightarrow AS: ID_C, ID_{RS}, R$

2. $AS \rightarrow C: T_{AS}, T_{CON}$

$$T_{AS} = \left(t, Sign_{AS}(t)\right)$$
$$t = (ID_C, scope_1, Lifetime1)$$
$$scope_1 = \left(\text{p}, ID_{RS}, Context(ID_C, ID_{RS}, property)\right)$$

$$T_{CON} = \left(t', Sign_{AS}(t')\right)$$
$$t' = (H(T_{AS}), scope_2, Lifetime2)$$
$$scope_2 = \left(ID_{RS1}, ID_{ESO1}, \text{“}read\text{”}, Context(ID_C, ID_{RS}, property)\right)$$

3. $C \rightarrow RS: ID_C, T_{AS}, T_{CON}$

3. $RS \rightarrow ESO: state\ request, T_{CON}$

4. $ESO \rightarrow RS: state, T_{CON}, Sign_{ESO}(state, T_{CON})$

5. *If the property is satisfied, RS provides the service to the client.*

Context-aware permission sequence

1. $C \rightarrow AS: ID_C, ID_{RS}, R$

2. $AS \rightarrow C: T_{AS}, T_{CON}$

$$T_{AS} = (P, state)$$
$$P = \begin{pmatrix} (ID_{RS1}, p1, context1), (ID_{RS2}, p2, context2), \\ (ID_{RS3}, p3, context3), (ID_{RS1}, p1, context1) \end{pmatrix}$$

$$T_{CON} = (H\ (T_{AS}), scope_2)$$
$$scope_2 = \begin{pmatrix} (ID_{RS1}, ID_{ESO1}, \text{“}read\text{”}, context1), (ID_{RS2}, ID_{ESO2}, \text{“}read\text{”}, context2), \\ (ID_{RS3}, ID_{ESO3}, \text{“}read\text{”}, context3), (ID_{RS1}, ID_{ESO1}, \text{“}read\text{”}, context1)) \end{pmatrix}$$

.

3. $C \rightarrow RS1: ID_C, T_{AS}, T_{CON}$

4. $RS1 \rightarrow ESO1: T_{CON}$

5. $ESO1 \rightarrow RS1: state$

6. *If the property is satisfied, RS1 provides the service to the client, generates* T_{RS1}*, and updates local state.*
   Client visit the next RS by passing T_{RS1}, T_{CON}. $Steps\ 3-6\ are\ repeated\ until\ the\ last\ RS.$