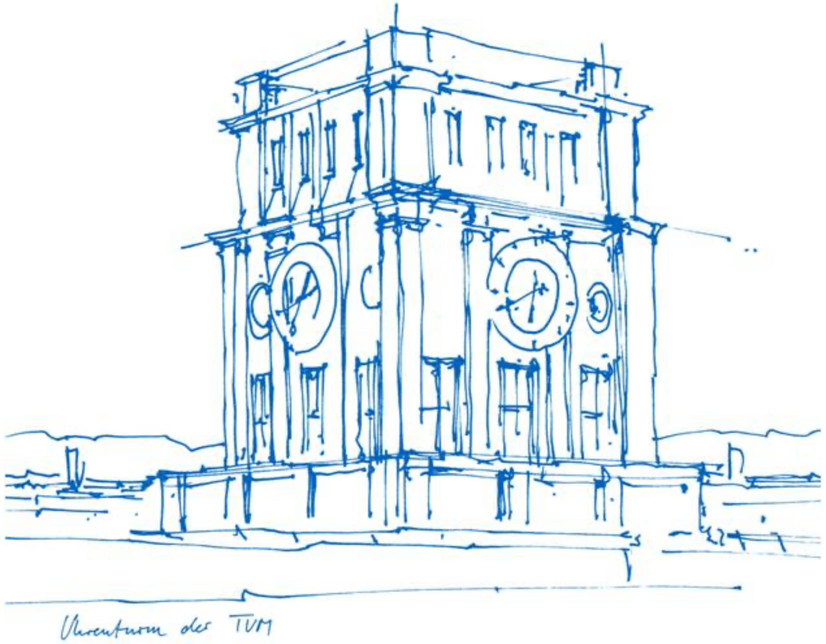
Chair of Network Architectures and Services
Department of Electrical and Computer Engineering
Technical University of Munich

A Tamper- and Fault-Resistant Certification Service

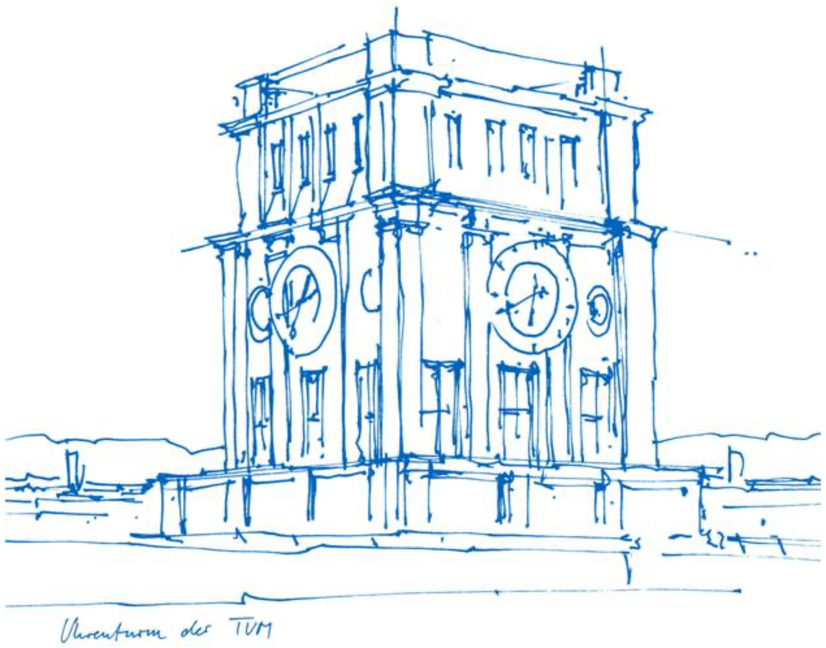
MIR3 (Garching)
17 September 2020

Filip Rezabek, Holger Kinkelin, Georg Carle
Email: {lastname}@net.in.tum.de



Motivation

- X.509 certificates bind the identity of an entity to a public key owned by that entity
  - Identities: URL of a Web site, name and e-mail address of a person, …
- Certificates are used in many cryptographic communication protocols to achieve authenticity and confidentiality
  - E.g. HTTPS, S/MIME, …
- Problem: Security of protocols relies on the correctness of the used certificates
- Goal: Create a tamper- and fault-resistant certification service
- RQ 1: How can we build a tamper-/fault-resistant system, that *authorizes* a CSR?
- RQ 2: How can we build a tamper-/fault-resistant system, that *signs* a CSR?



„How can we build a tamper-/fault-resistant *system, that authorizes a CSR?“*

HARDENING X.509 CERTIFICATE ISSUANCE USING DISTRIBUTED LEDGER TECHNOLOGY

Initially presented at NOMS 2020 - IEEE/IFIP Workshop DISSECT 2020, Budapest, Hungary, Apr. 2020

Case Study: Issuance Process of an X.509 S/MIME Certificate by DFN PKI

- DFN PKI:
  - Global CA used by German universities and other research organizations [1]
  - Audited according to ETSI EN 319 411-1 standard [2]

- Issuance process:
  - Certificate requester (CR) generates new asymmetric key pair and certificate signing request (CSR)
  - CSR contains CR's name, mail address, and public key
  - CSR is sent to Certificate Authority (CA)
  - CR must meet only one Registration Authority (RA) member of the CA in person; RA validates CSR by checking CR's identity using identity document
  - RA member authorizes CSR using some application on his computer
  - CA signs/issues certificate; certificate is delivered to CR

- Note: Certificate issuance can work different in different scenarios;
  Key idea is always to authorize the CSR by some means

Attack Vectors / Possible Error Sources

- AV1: Compromise the CA [3]
  - Attacker remotely controls CA
  - Attacker steals signing key of CA
  - ➢ Attacker can issue fraudulent certificate at will
- AV2: Compromise RA member [4]
  - Attacker remotely controls RA member's computer
  - ➢ Attacker can authorize fraudulent CSR
- AV3: Malicious RA Member
  - Attacker collaborates with RA member
  - ➢ Attacker can authorize fraudulent CSR
- AV4: Careless RA Member
  - RA member makes a mistake
  - ➢ Incorrect certificate is issued

Goals and Requirements

- Overall Goal: Create a system that improves the correct operation of a Certificate Authority and of its Registration Authorities

- Requirements on this system/solution:
  - R1: Multiparty CSR validation
    - ➢ The validation of a CSR must not depend on a single RA member
  - R2: Accountability of CSR validation
    - ➢ Collect information which CSR has been processed by which RA member
  - R3: Accountability of certificate issuance
    - ➢ Each issued certificate must be logged
  - R4: Enforcement of the certificate issuance workflow
    - ➢ Certificate issuance workflow must be carefully guided/enforced
  - R5: Tamper-resistance of workflow enforcement and collected information
    - ➢ Workflow enforcement cannot be bypassed
    - ➢ Collected meta-information cannot be changed/deleted

Brief Introduction to Hyperledger Fabric

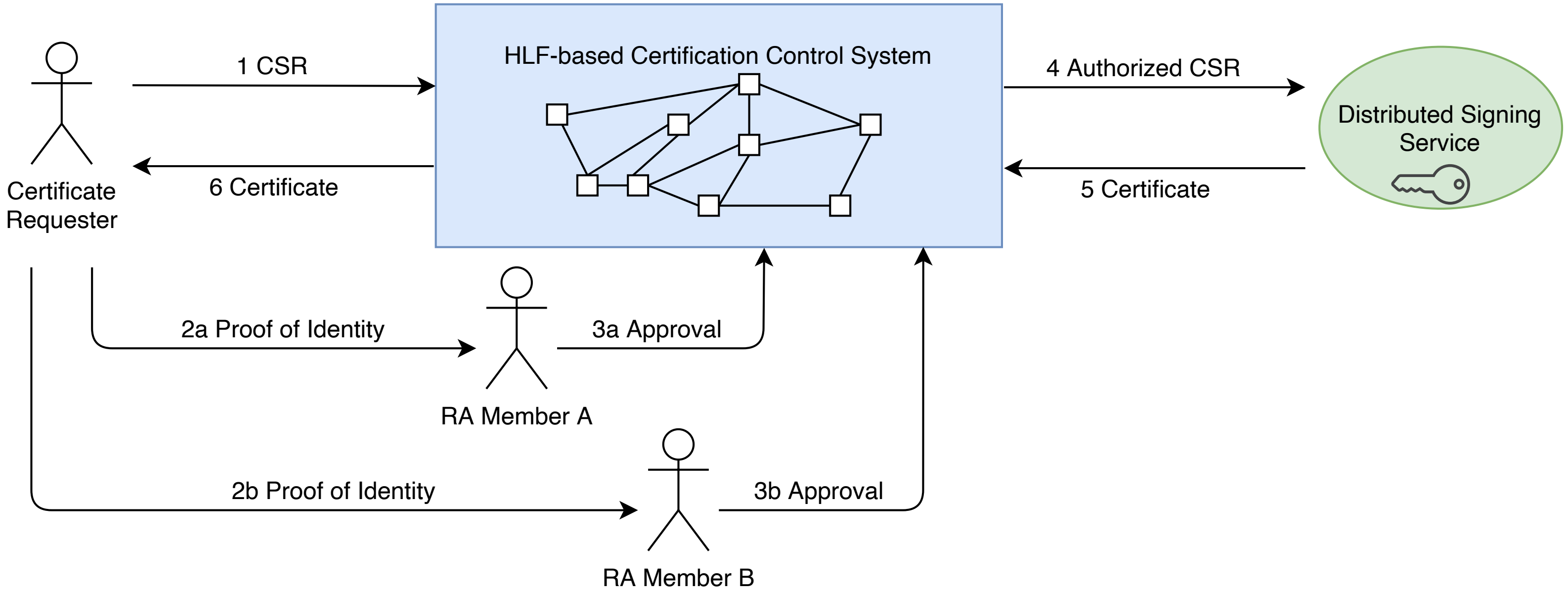
- Fabric is a *Distributed Ledger* and *Chaincode* framework

- Distributed Ledger [5, p. 17]
  - A "type of database that is spread across multiple sites"
  - "Records are stored one after the other in a continuous ledger"
  - Records "can only be added when the participants reach a quorum"

- Chaincode [6, 7] ≙ Smart Contracts known from Ethereum [8]
  - Chaincode implements business logic; CC causes side-effects in the ledger:
    - append a new data element, append an updated version of an existing data element
  - Chaincode is invoked by clients by sending a transaction into the Fabric network
  - Multiple instances of the *same* Chaincode run on *different* nodes of a Fabric network
  - Transaction must be *endorsed* by multiple Chaincode instances to change the ledger

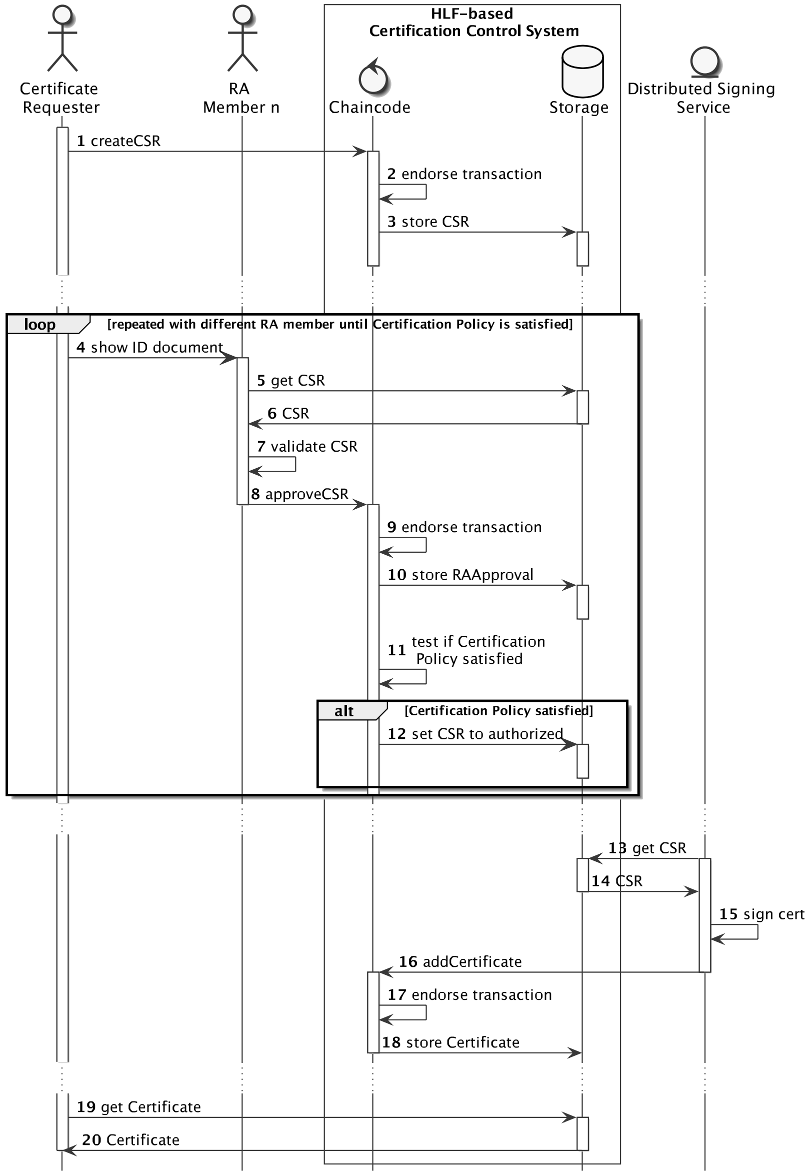
➢ Fabric offers Byzantine fault-tolerant execution of processes and a non-modifiable and non-mutable data storage

Overview



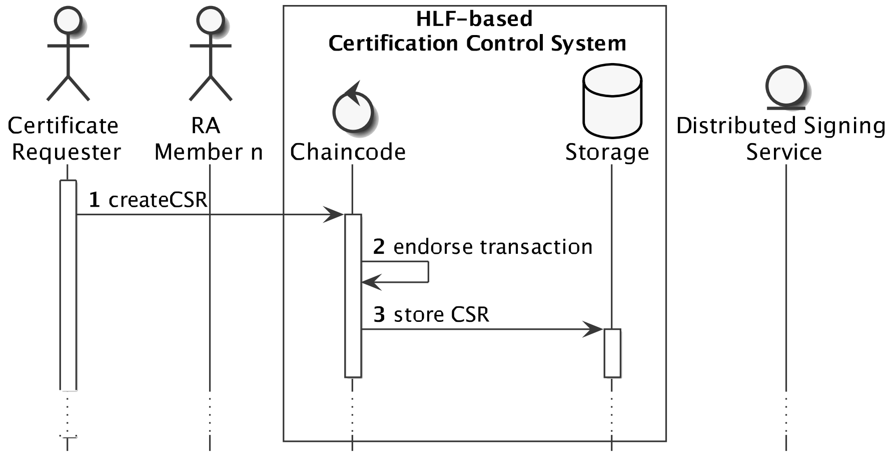
- Certification Control System (CCS) is intermediary between human entities and the actual CA
- CCS implemented on top of HLF using different transactions, Chaincode, and a user-centered data model
- CCS (i.e. tamper-resistant chaincode) authorizes CSR before signing the certificate based on
  - Approvals by RA members stored in ledger
  - Conditions specified in certification policy stored in ledger

Sequence Diagram (Overview)



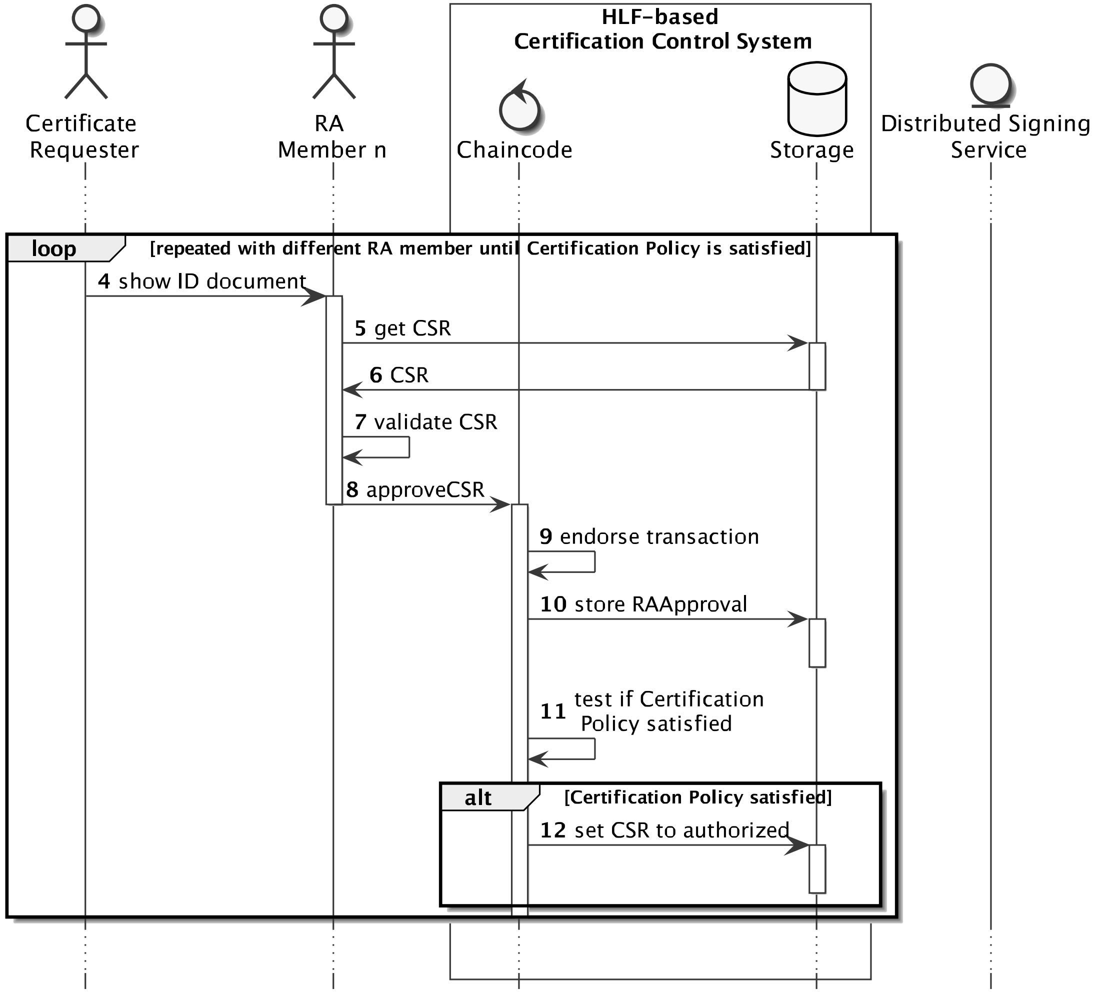
- Note:
  - Strongly simplified
  - HLF-based CCS is distributed:
    - Chaincode runs on various nodes
    - Storage is replicated on various nodes

Sequence Diagram (1/3)



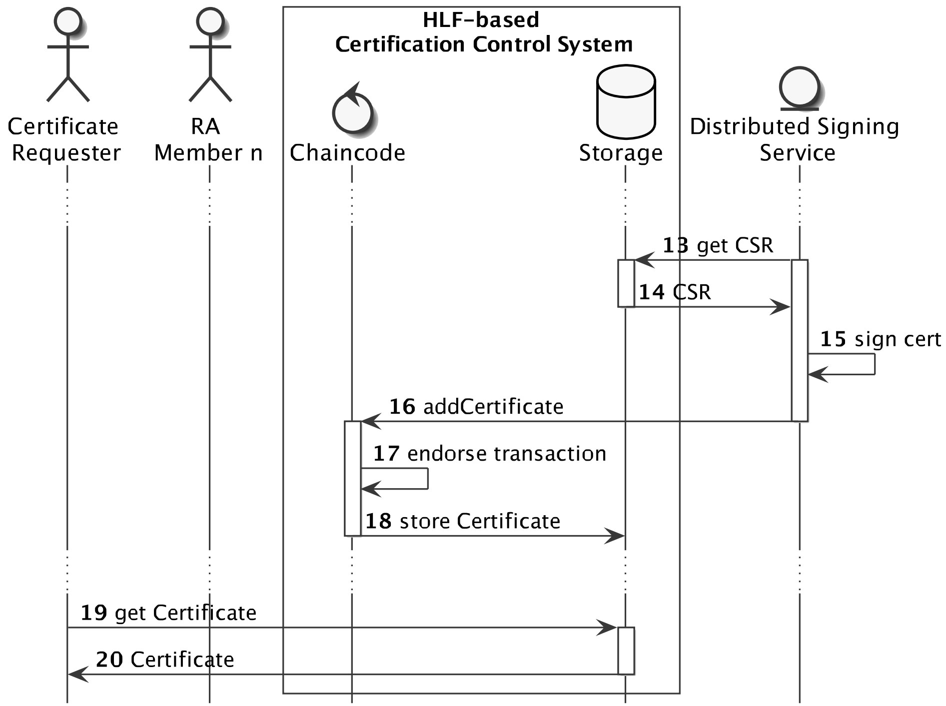
- 1 - 3: store new CSR in CSS, CSR is unauthorized `(authorized = False)`

Sequence Diagram (2/3)



- 4: CR meets with RA Member, presents ID document
- 5-6: retrieve CSR from ledger
- 7: validate identity with ID document, check if user owns mail address, etc.
- 8-10: store approval
- 11: system checks if policy is fulfilled
- 12: set CSR state to authorized `(authorized = True)`
- 4-12: repeat until CSR has been validated by a policy-defined amount of RA members

Sequence Diagram (3/3)



- 13-14: CA fetches CSR record from ledger
- 15: CA checks if `authorized = True` and signs certificate
- 16-18: store new certificate, set CSR state to issued
- 19-20: user retrieves certificate from ledger

Discussion: Fault Tolerance

- Fault = fraudulent (attack) or incorrect (mistake) certificate got issued
- Fault tolerance depends on Certification Policy
  - More CSR approvals requested → more secure → more effort
- Probability of a fault: $P_{Fault} = \frac{\binom{\#\,bad\,RA\,Members}{\#\,requested\,CSR\,approvals}}{\binom{\#\,available\,RA\,Members}{\#\,requested\,CSR\,approvals}}$ („bad“ = malicious/careless)
- Examples:
  - 10 RA members, 3 bad RA members, 3 approvals required → $P_{Fault} = 0{,}8\%$
  - 10 RA members, **4** bad RA members, 3 approvals required → $P_{Fault} = 3\%$
  - 10 RA members, 3 bad RA members, **4** approvals required → $P_{Fault} = 0\%$
- Prevents mis-issuance/increases probability that only valid certs are issued (AV 2 - 4)
  - In case of a fault, collected meta-data helps to identify bad RA members
- System cannot prevent mis-issuance of certificates if CA got compromised (AV 1)
  - But: non existent approvals make it possible to identify such certificates

Related Work

- Certificate Transparency (CT) [9]
  - CT extends the X.509 ecosystem with a public log of issued certificates
  - Clients query log to find conflicting certificates, which indicate problems (mis-issued certificates)
  - ➢ CT cannot prevent mis-issuance of certificates but helps to detect mis-issued certificates
  - ➢ Our system prevents mis-issuance of certificates and helps to detect mis-issued certificates

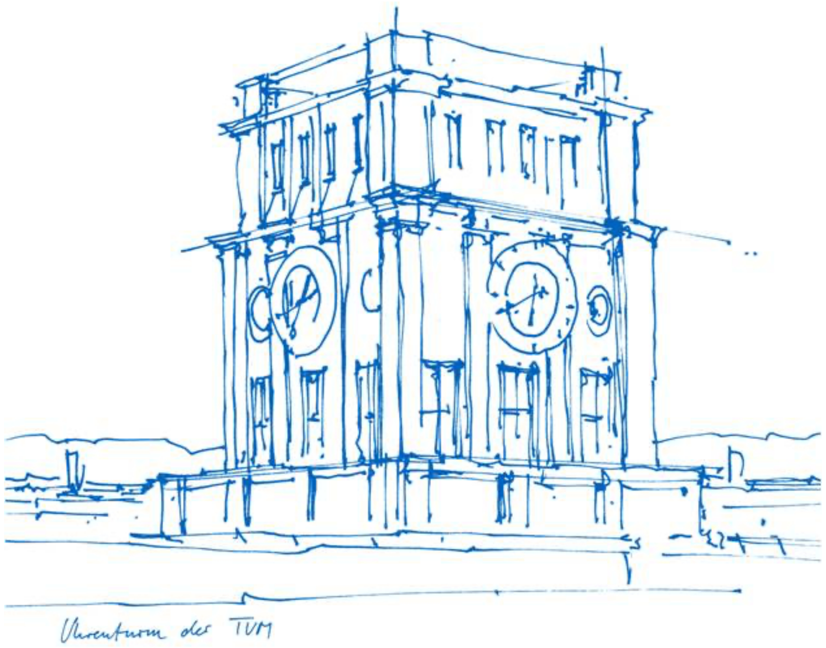
- Instant Karma PKI (IKP) [10]
  - IKP follows the idea to create financial incentives for CAs to behave well
  - Idea is that CA deposit money in an Ethereum smart contract and agree to pay a penalty when a certificate got mis-issued
  - ➢ IKP leaves out how security of CA can be improved
  - ➢ Our work provides a technical solution

Related Work (cont.)

- uPort [11], Sovrin [12]
  - Both are Blockchain-based system for self-sovereign ID management following the web-of-trust approach
  - Entities manage information about themselves; other entities can assert the correctness of this information
  - ➢ uPort and Sovrin are alternative approaches to X.509
  - ➢ Our work is an extension to the X.509 ecosystem

Conclusions

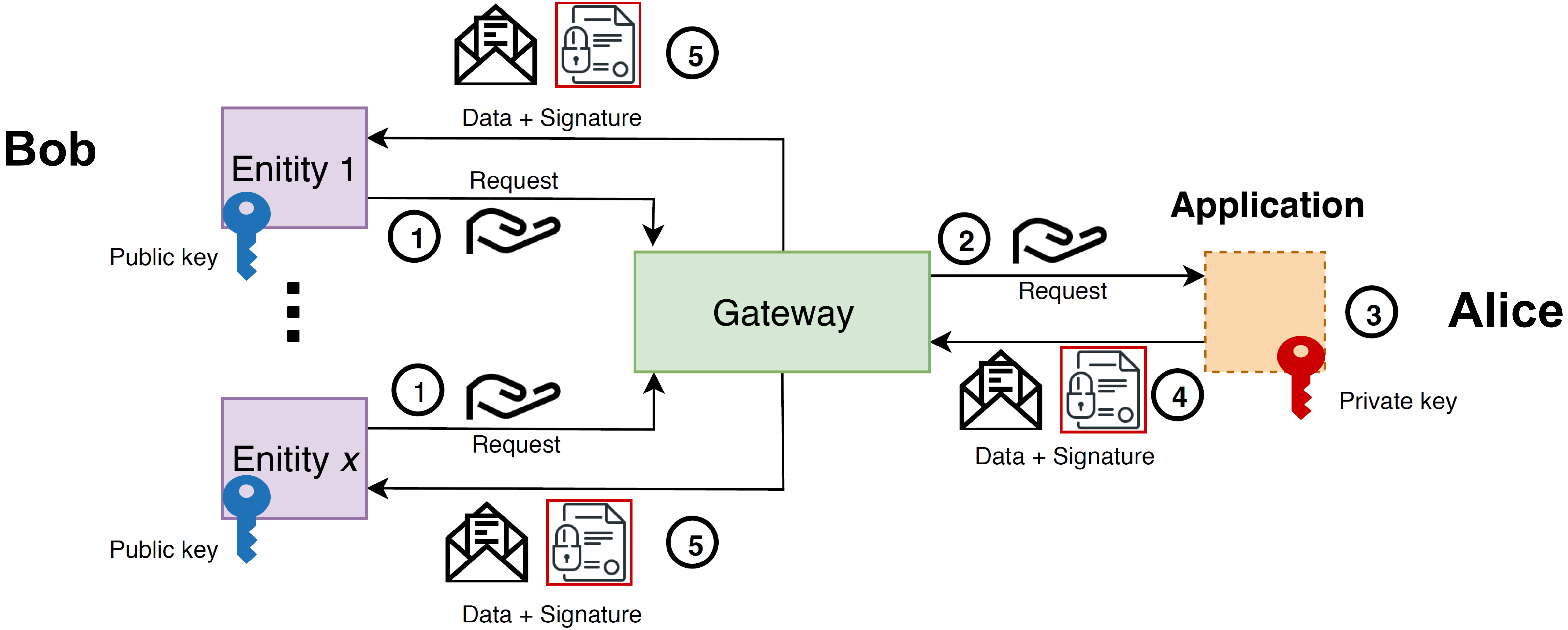
- Correctness of the certificate issuance process is crucial
- We proposed a system that enforces a policy-defined, multi-party validation and authorization workflow of CSR
- Properties:
  - Hardens the authorization process of CSR
  - Cannot prevent „direct“ attacks on CA‘s signing key
  - Collected information helps to determine the cause of mis-issued certificates
    - „Bad“ RA member
    - Direct attack on CA’s signing key



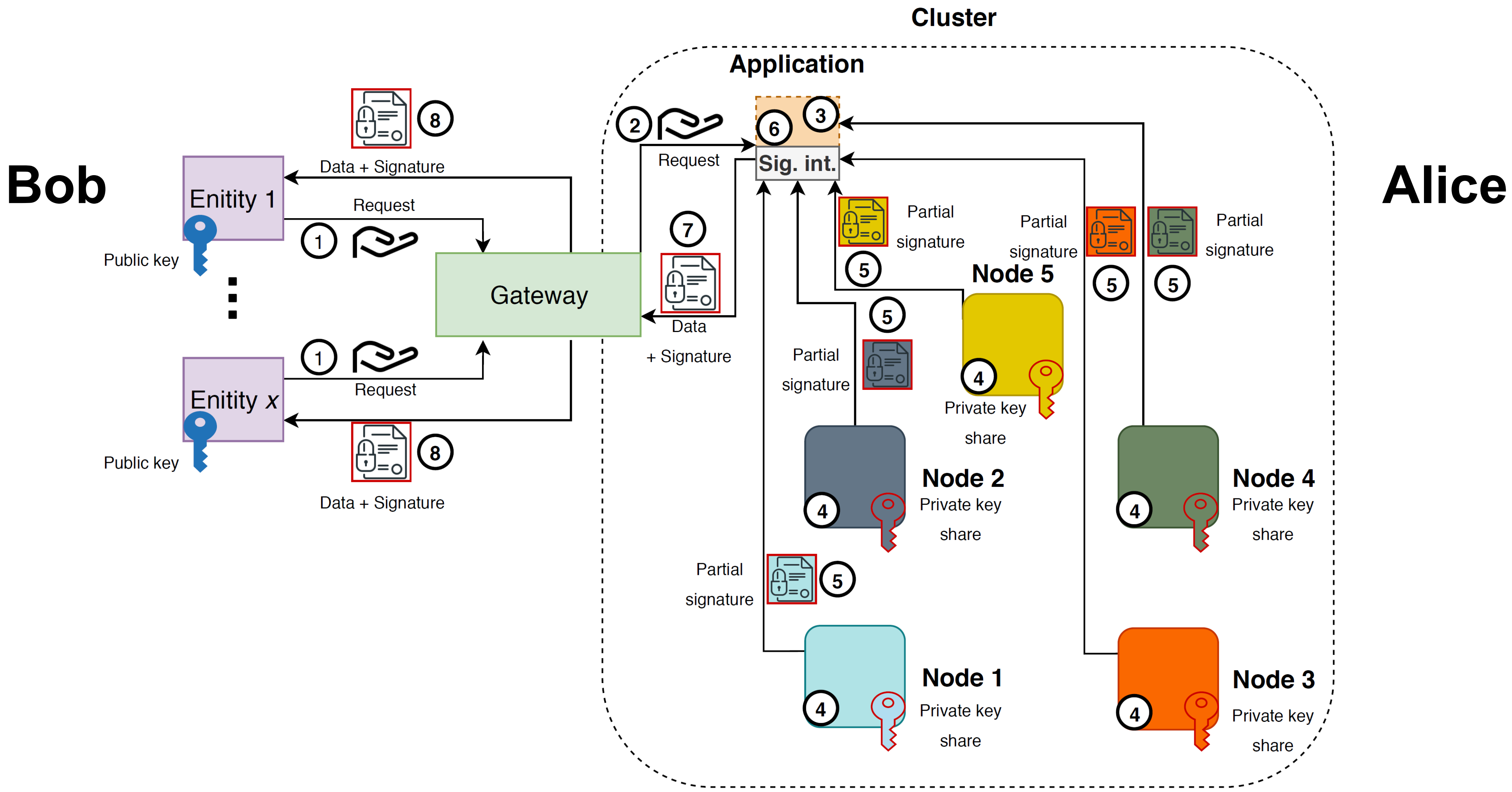
„How can we build a tamper-/fault-resistant *system, that signs a CSR?“*

VERIFIABLE SECRET SHARING AND THRESHOLD SIGNATURES FOR TAMPER-RESISTANT SIGNATURE SERVICES

Motivation by example



Approach



Cluster
Application
Bob
Alice
Enitity 1
Enitity x
Public key
Public key
Data + Signature
Request
Request
Data + Signature
Gateway
Request
Data
+ Signature
Sig. int.
Partial
signature
Partial
signature
Partial
signature
Partial
signature
Partial
signature
Node 5
Private key
share
Node 2
Private key
share
Node 4
Private key
share
Node 1
Private key
share
Node 3
Private key
share

General requirements

- Key generation and management
  - Key generation less time critical
  - Re-keying and key revoking

- Threshold signature generation - $T\ out\ of\ N$ partial signers
  - T – threshold number of partial signers
  - N – number of nodes
  - C – number of corrupted/unavailable nodes
  - $T > {}^{N}/_{2}, C \leq N - T$ (it follows $C \leq T - 1$)
  - Time constraint

- Key and signatures compatibility with X.509 standard

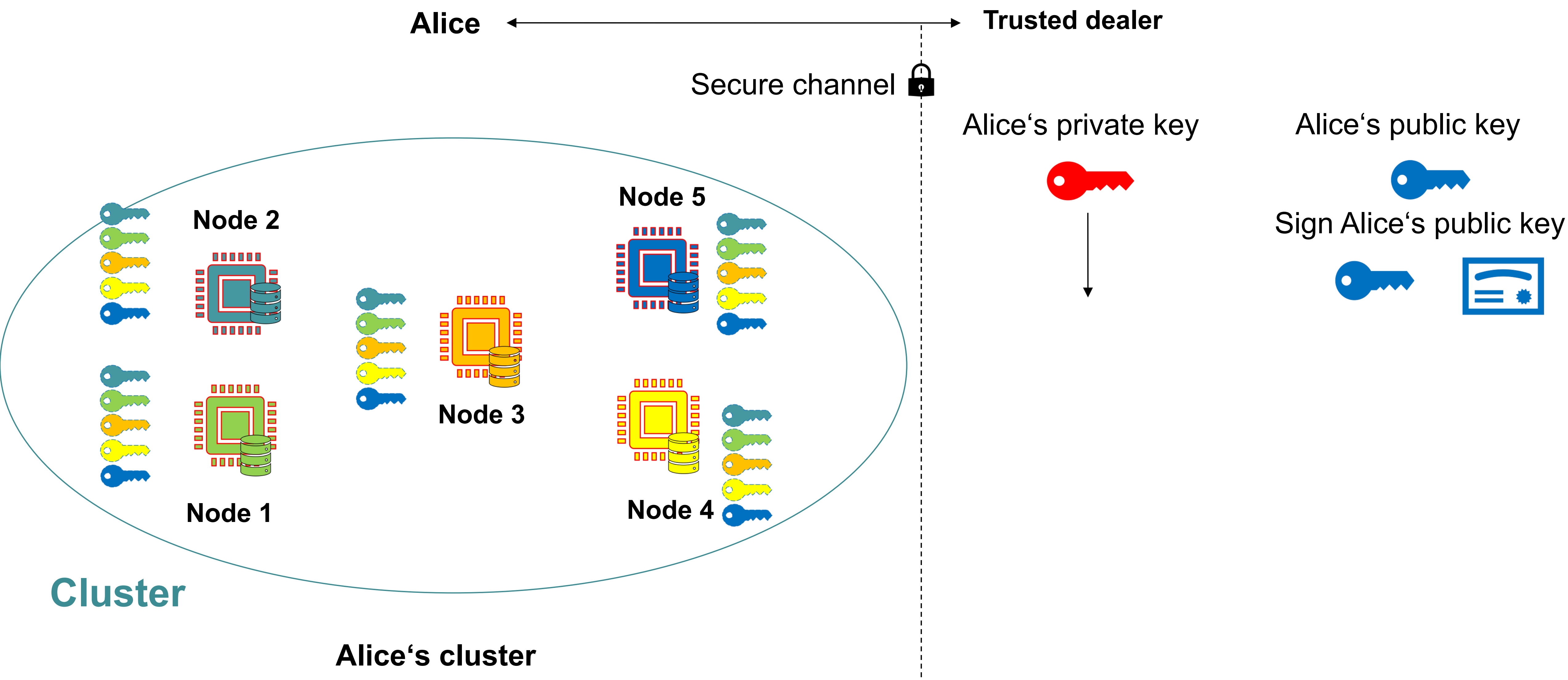
PROTOCOL OVERVIEW

Verifiable secret sharing

Alice ←——→ **Trusted dealer**

- Dealer oriented approach
- Private key and key shares are known by the dealer → trusted dealer
- Distribute an RSA private key [13]

Verifiable secret sharing



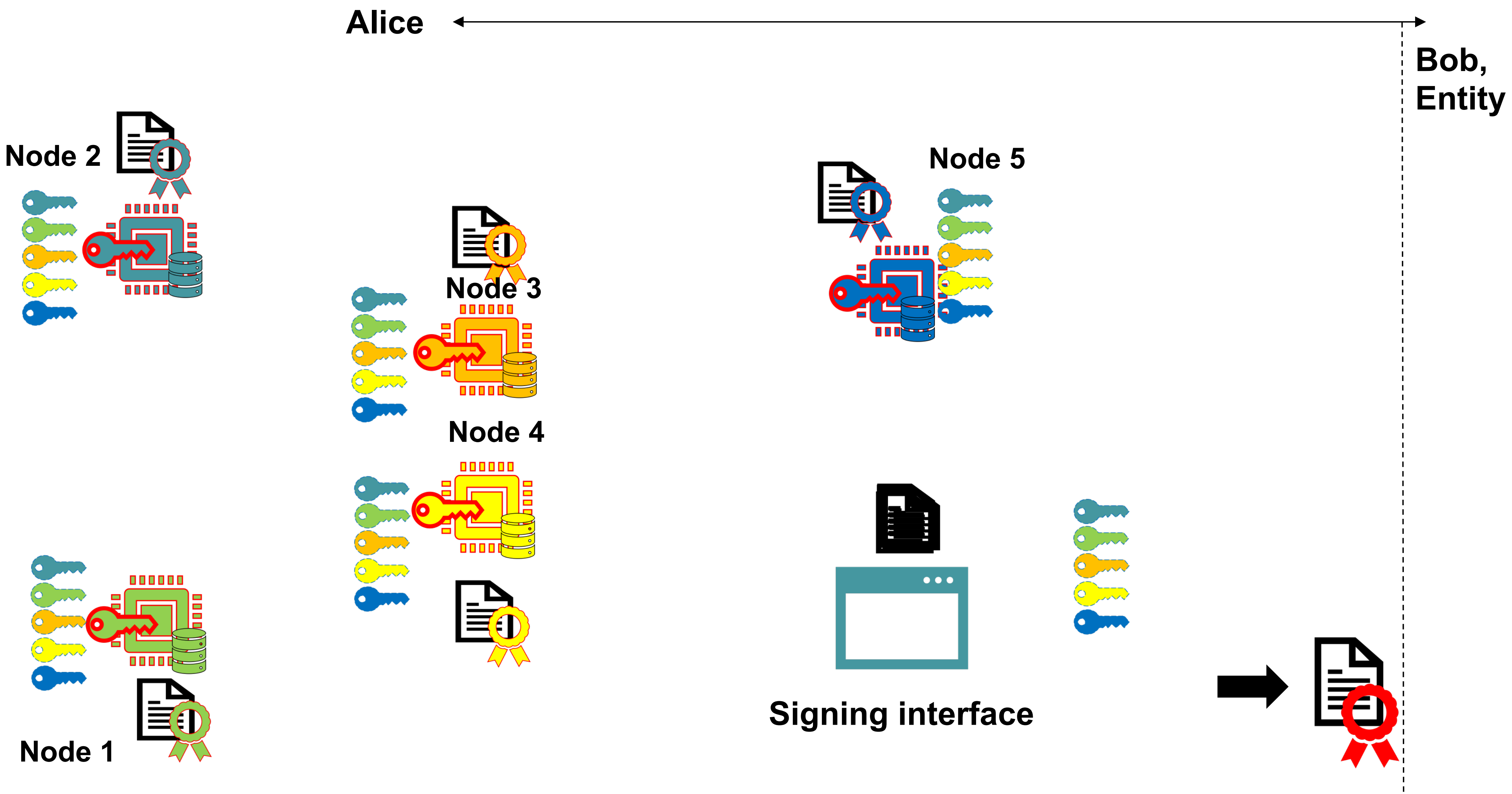
Threshold signature generation

Alice

Bob, Entity

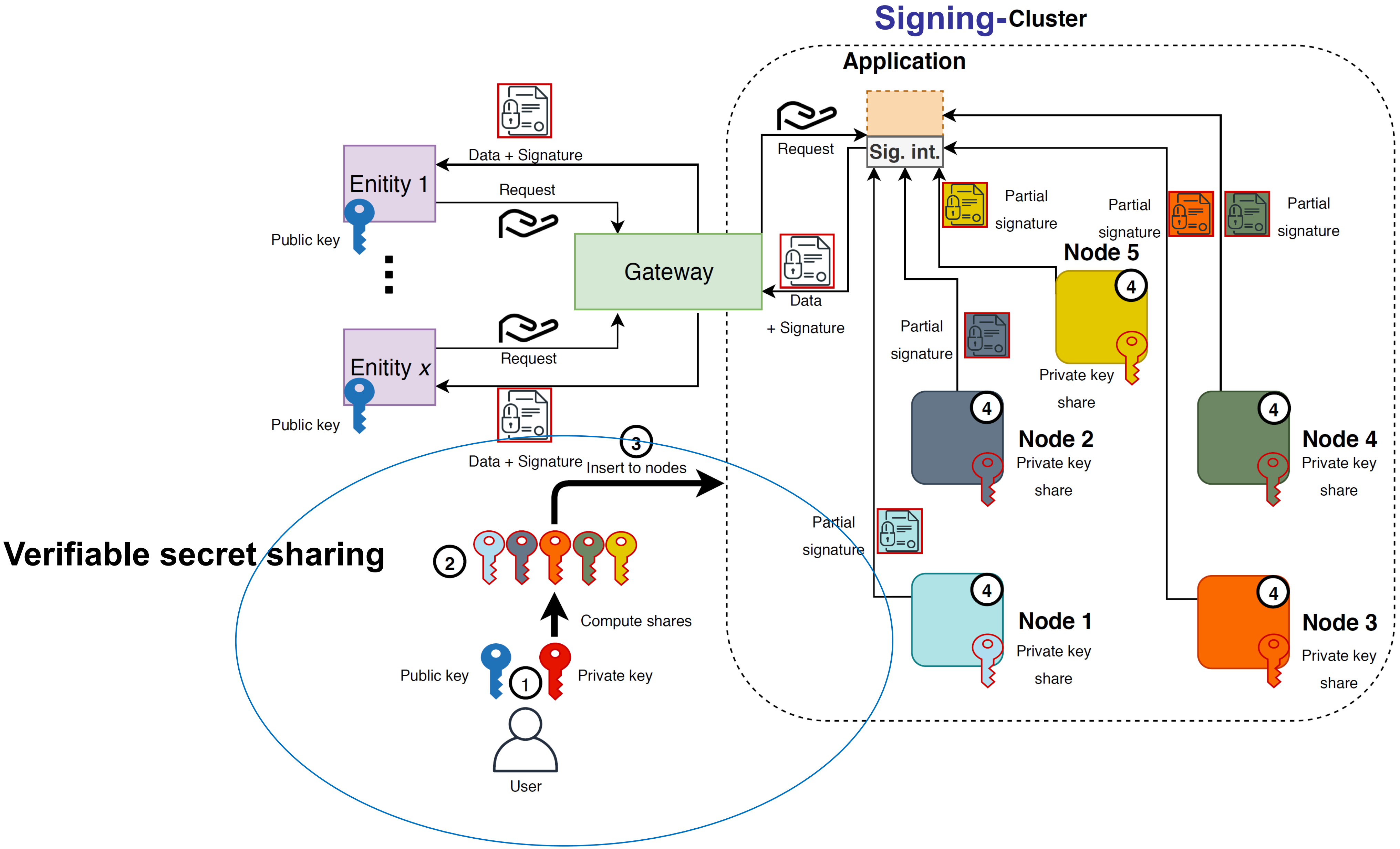
- Private key is not assembled to sign data
- Nodes and other parties are not trusted
- Communication inside of a cluster over a secure channel

Threshold signature generation



FINAL SOLUTION

Solution architecture



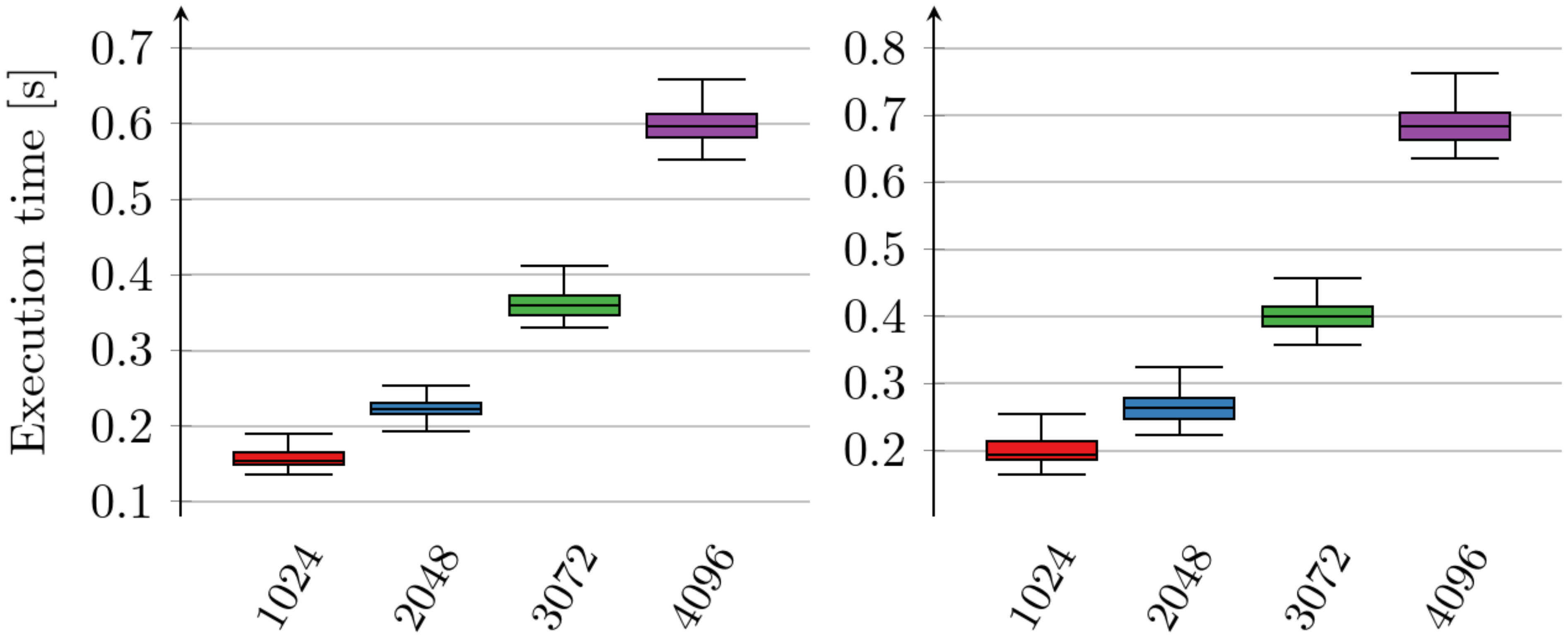
EVALUATION

Evaluation

- Two parts
  - Theoretical communication model
    - Not covered today
  - Performance evaluation
    - Present two scenarios
- Compare theoretical model with real data

Performance evaluation: Scenario I – Overall time

- Evaluate overall system performance for N = 10, $T = \text{floor}(^{N}/_{2}) + 1$, msg = 1024B

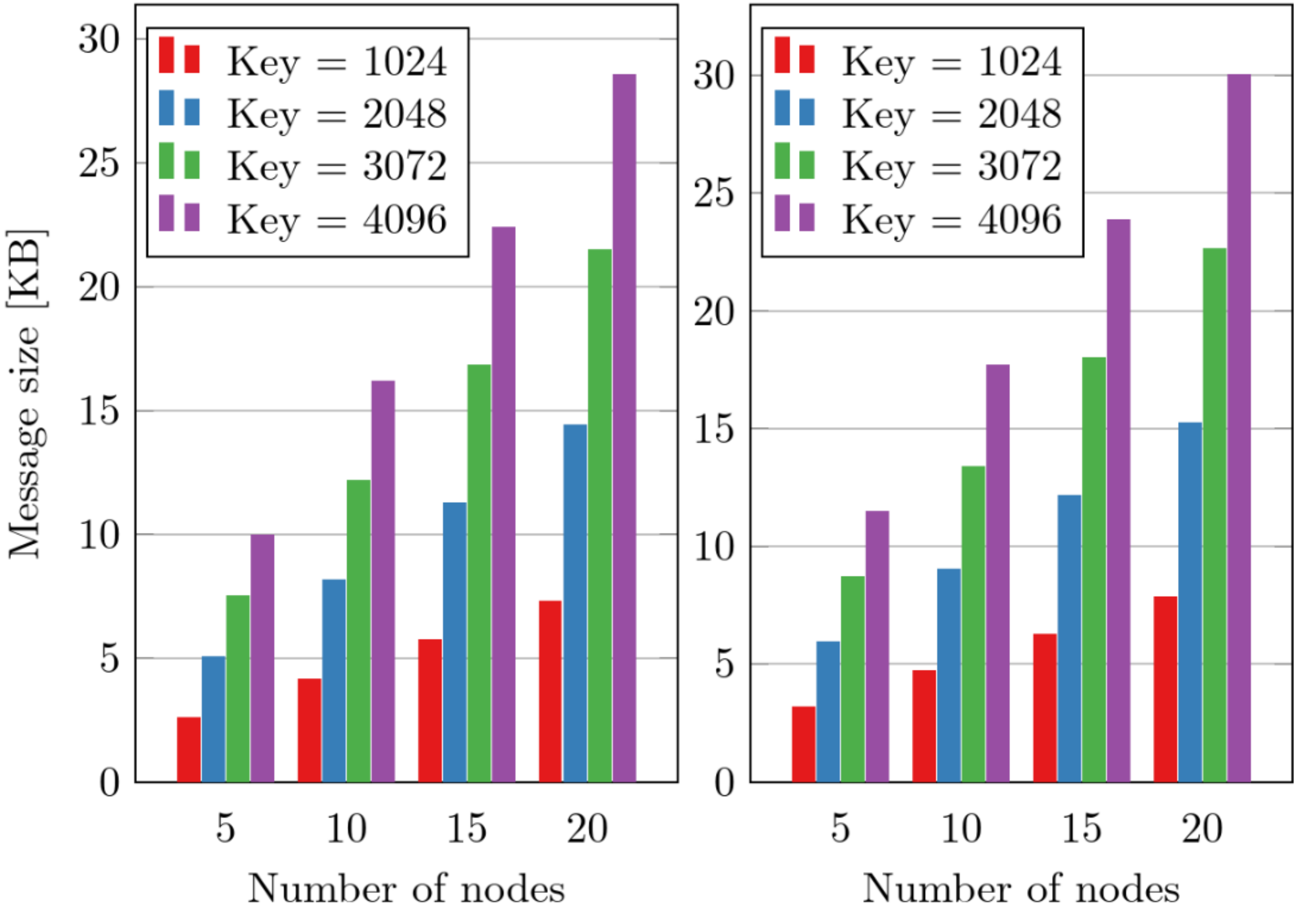


Time-to-generate

Time-to-sign

Performance evaluation: Scenario II – Message overhead

- Message size is not important – working with hashes
- Evaluate message size for $T = \text{floor}(N/2) + 1$



(a) Key material storing

(b) Response with partial signature

Performance evaluation: Overview

$$T^{kGen}_{total} = \underbrace{T^{prime}_{gen} + T^{key}_{gen} + T^{kShare}_{gen} + T^{vShare}_{gen}}_{\text{Manufacturer}} + \underbrace{\alpha^{HTTPS}_{gen}}_{\text{Communication}} + \underbrace{T^{KeyStore}_{store}}_{\text{Node}}$$

$$T^{sign}_{total} = \underbrace{T^{sShare}_{sign} + T^{proof}_{sign} + T^{KeyStore}_{read}}_{\text{Node}} + \underbrace{\alpha^{HTTPS}_{sign}}_{\text{Communication}} + \underbrace{T^{proof}_{verify} + T_{intrpl}}_{\text{Sign. interface}}$$

Time	Unit	2048
T^{kGen}_{total} *	[s]	0.23
T^{prime}_{gen}	[s]	6.55
T^{key}_{gen}	[us]	73
T^{kShare}_{gen}	[ms]	1.2
T^{vShare}_{gen}	[ms]	52.3
α^{HTTPS}_{gen}	[s]	0.14
$T^{KeyStore}_{store}$	[s]	0.02
T^{sign}_{total}	[s]	0.27
α^{HTTPS}_{sign}	[s]	0.1
$T^{KeyStore}_{read}$	[ms]	0.1
T^{sShare}_{sign}	[ms]	3.8
T^{proof}_{sign}	[ms]	10
T^{proof}_{verify}	[ms]	9.4
T_{intrpl}	[ms]	0.72

Key-generation operations

Signing operations

Performance evaluation: Conclusions

- Key size is most significant
  - Affects time in an exponential way
  - Exchange messages size in a linear way

- Number of nodes and threshold value increases time linearly

- Most significant overhead caused by communication

Next steps

- Currently looking into:
  - EC based solution – BLS, Schnorr, and ECDSA
  - Dealerless approach – EC

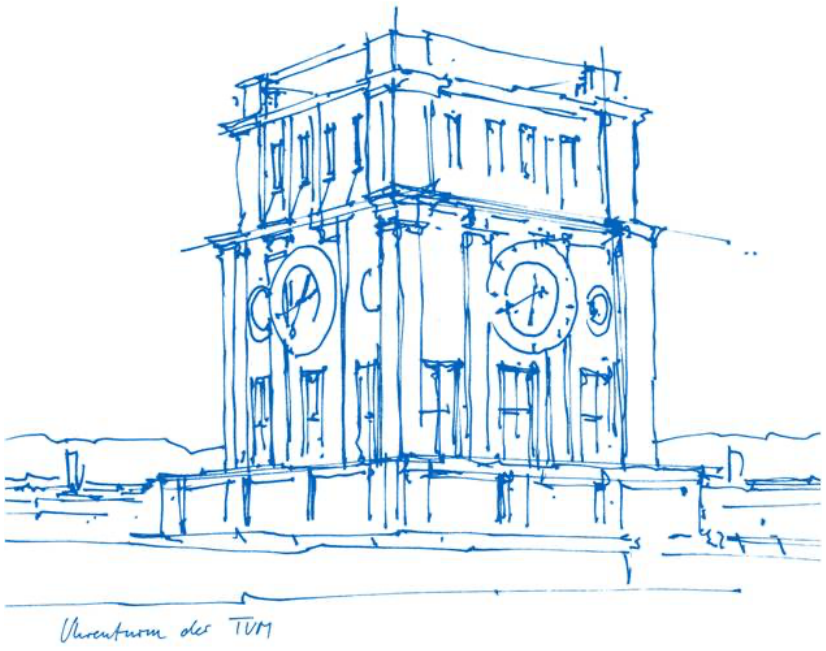
- Future work:
  - Post-quantum crypto solutions
  - Try different secure communication protocols - Mbed TLS

SUMMARY 2.0

Summary 2.0

- Increasing the quality of X.509 certificates requires two steps

1. Hardening the validation/authorization workflow of CSR
    - Multiparty authorization + Accountability
    - Enforced by tamper-proof system running on Hyperledger Fabric
    - Cannot fix attacks on CA’s signing key
2. Hardening CA’s signing key against attacks (theft and abuse)
    - Distributing keys into shares
    - Threshold crypto operations for signing



Filip Rezabek, Holger Kinkelin, Georg Carle
Email: {lastname}@net.in.tum.de

THANK YOU!

Resources

[1] DFN-Verein, “Überblick DFN-PKI,” 2019. [Online] https://www.pki.dfn.de/ueberblick-dfn-pki/, last accessed on Friday 24th April, 2020.

[2] ETSI, “Electronic Signatures and Infrastructures (ESI); Policy and security requirements for Trust Service Providers issuing certificates; Part 1: General requirements,” 2017. [Online] https://www.etsi.org/deliver/etsi_en/319400_319499/31941101/01.02.00_20/en_31941101v010200a.pdf, last accessed on Friday 24th April, 2020.

[3] Threatpost, “Final Report on DigiNotar Hack Shows Total Compromise of CA Servers,” 2012. [Online] https://threatpost.com/final-report-diginotar-hack-shows-total-compromise-ca-servers-103112/77170/, last accessed on Friday 24th April, 2020.

[4] Threatpost, “Phony SSL Certificates issued for Google, Yahoo, Skype, Others,” 2011. [Online] https://threatpost.com/phony-ssl-certificates-issued-google-yahoo-skype-others-032311/75061/, last accessed on Friday 24th April, 2020.

[5] The UK Government Chief Scientifc Adviser, “Distributed Ledger Technology,” 2008. [Online]https://www.gov.uk/government/uploads/system/uploads/attachment_data/ file/492972/gs-16-1-distributed-ledger-technology.pdf, last accessed on Friday 24th April, 2020.

[6] E. Androulaki et al., “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains,” in Proc. of the 13th EuroSys Conf., 2018.

[7] The Linux Foundation, “Hyperledger,” 2018. [Online] https://www.hyperledger.org, last accessed on Friday 24th April, 2020.

[8] Ethereum Foundation, “A Next-Generation Smart Contract and Decentralized Application Platform,” 2018. [Online] https://github.com/ethereum/wiki/wiki/White-Paper, last accessed oned on Friday 24th April, 2020.

[9] B. Laurie, A. Langley, and E. Kasper, “Certificate Transparency.” RFC 6962 (Experimental), June 2013.

[10] S. Matsumoto and R. M. Reischuk, “IKP: Turning a PKI Around with Decentralized Automated Incentives,” 2017 IEEE Symposium on Security and Privacy (SP), pp. 410–426, 2017.

[11] uPort, “uPort Website.” [Online] https://www.uport.me/, last accessed on Friday 24th April, 2020.

[12] Sovrin-Foundation, “A Protocol and Token for Self-Sovereign Identity and Decentralized Trust,” 2018. [Online] https://sovrin.org/wp-content/uploads/Sovrin-Protocol-and-Token-White-Paper.pdf, last accessed on Friday 24th April, 2020.

[13] Shoup, V. (2000). Practical Threshold Signatures. *Advances in Cryptology — EUROCRYPT 2000 Lecture Notes in Computer Science*, 207–220. doi: 10.1007/3-540-45539-6_15

[14] Jasonkresch. (2019, March 18). jasonkresch/protect. Retrieved from https://github.com/jasonkresch/protect.