ACSAC 2021

TEEKAP: Self-Expiring Data Capsule using Trusted Execution Environment (TEE)

Mingyuan Gao, Hung Dang and Ee-Chien Chang

Motivating Example

Alice (Data Owner)

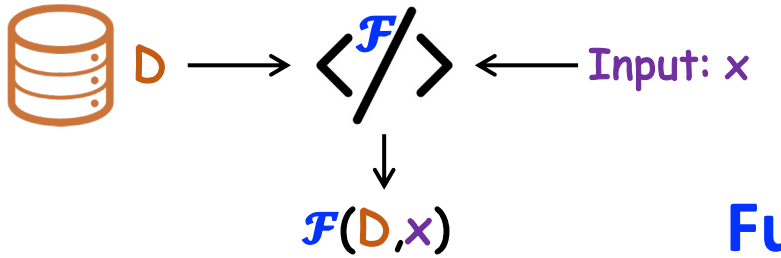
Sensitive Data D

- No involvement during the computation!

Send-and-Forget

Bob wants to do a joint computation on D using his function $\mathcal{F}$

Bob (Data User)



D can only be accessed by approved functions in a privacy-preserving way!

Functional Access

- Not allowed to see the data D
- Not allowed to use other functions to access D

- D becomes inaccessible automatically after the usage

Self-expiry

Do we have an existing solution to this?

Three related **privacy-preserving** techniques:

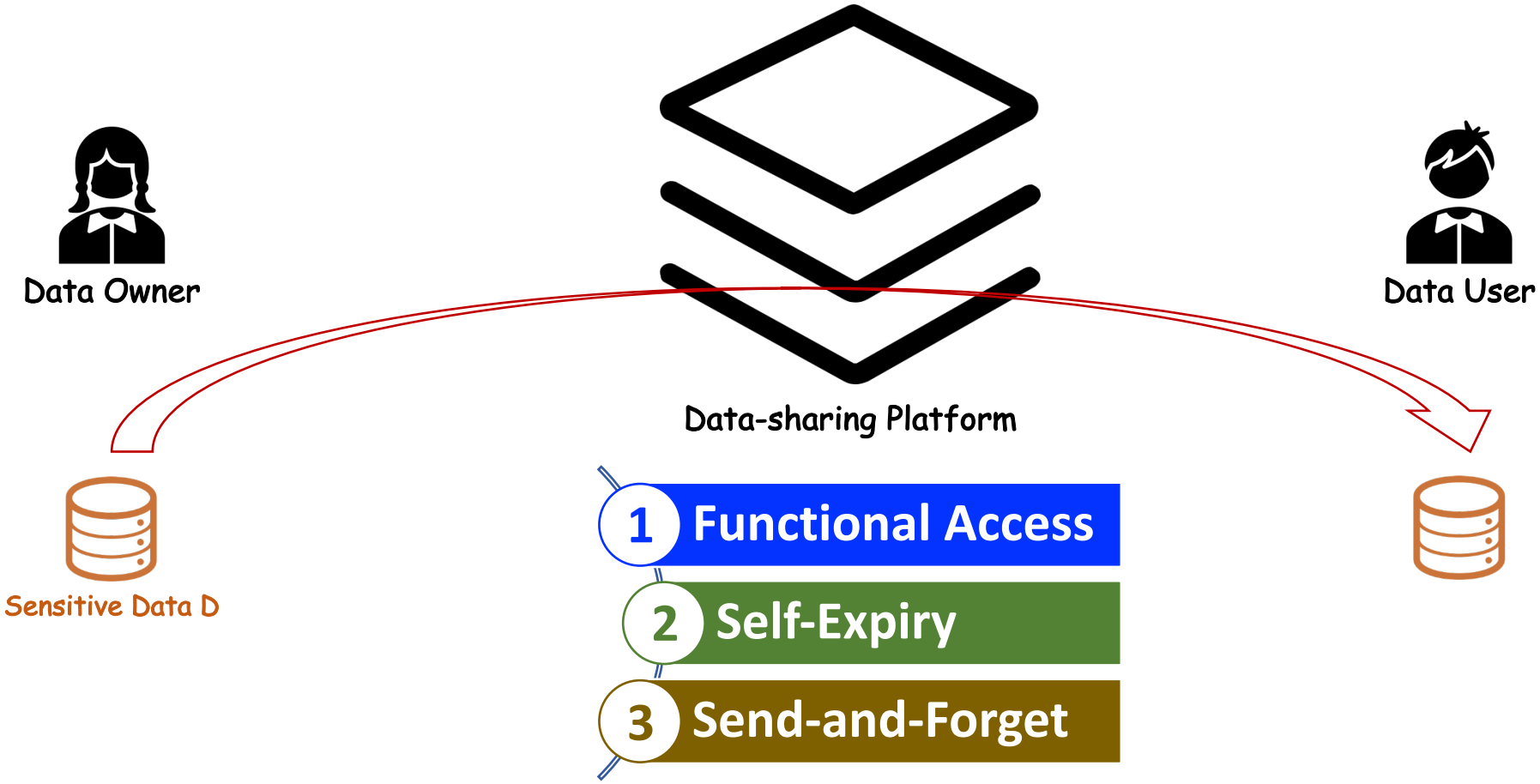
Allow users to do computations on **encrypted** data

Allow **parties** to **jointly** compute a function over their **inputs** while keeping those inputs **private**

Protect **data-in-use** using **hardware**-based Trusted Execution Environment (**TEE**)

Security Goals	Fully Homomorphic Encryption (FHE)	Secure Multi-Party Computation (MPC)	Confidential Computing (CC)
Functional Access	✗	✓	✓
Self-Expiry	✗	✗	✗
Send-and-Forget	✓	✗	✓

We propose a data-sharing platform that attains all the three security goals!

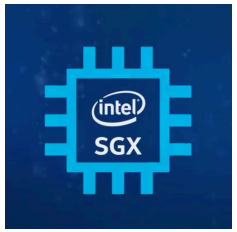


Our platform is based on the TEE technology!

Trusted Execution Environment (TEE) 101

TEE is an up-and-coming security technology.

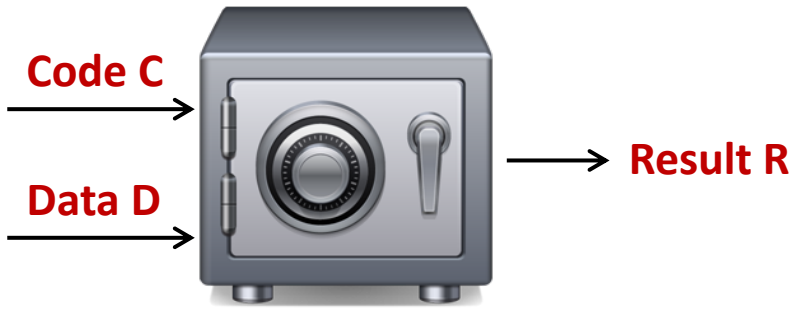
- A **vault** in the **CPU** for **sensitive** code and data, aka **secure enclave**.
- The computation in the vault is **verifiable**!



Intel SGX (2015)

AMD SEV (2017)

Apple M1 (2020)



Proof

C ran on D and produced R without anyone seeing or manipulating the computation!

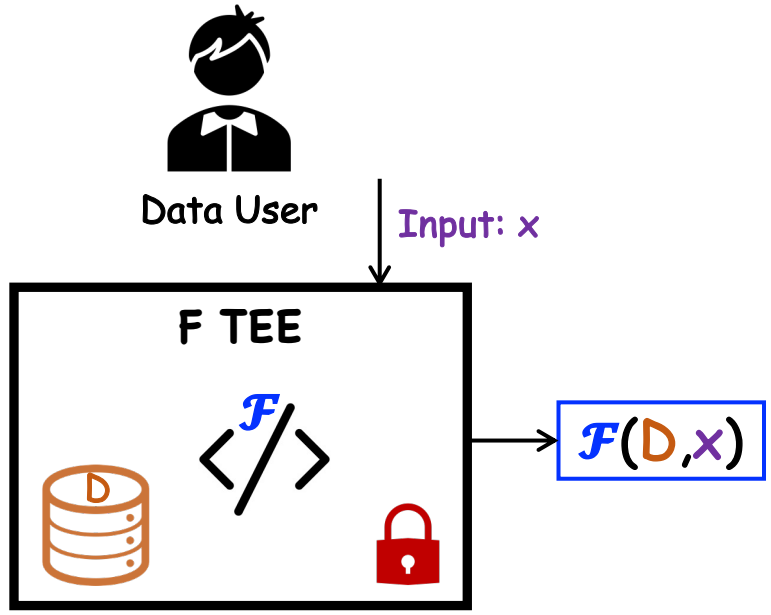
build up the platform

Enforcing functional access using TEE

TEEs allow for the **secure** and **verifiable** processing of **data** on **untrusted** machines!

Data Owner

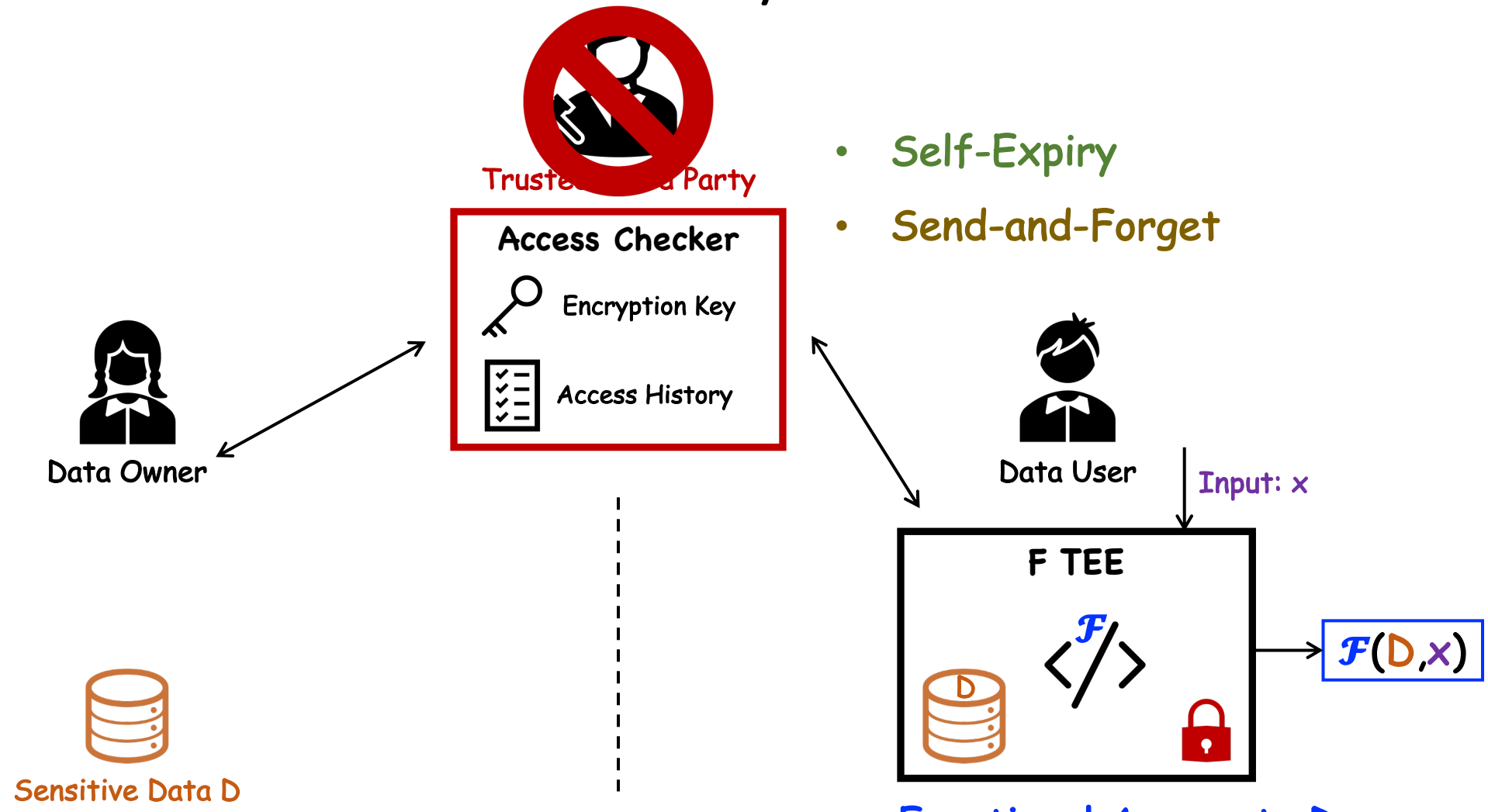
Sensitive Data D



Functional Access to D

build up the platform

Assume we have a Trusted Third Party

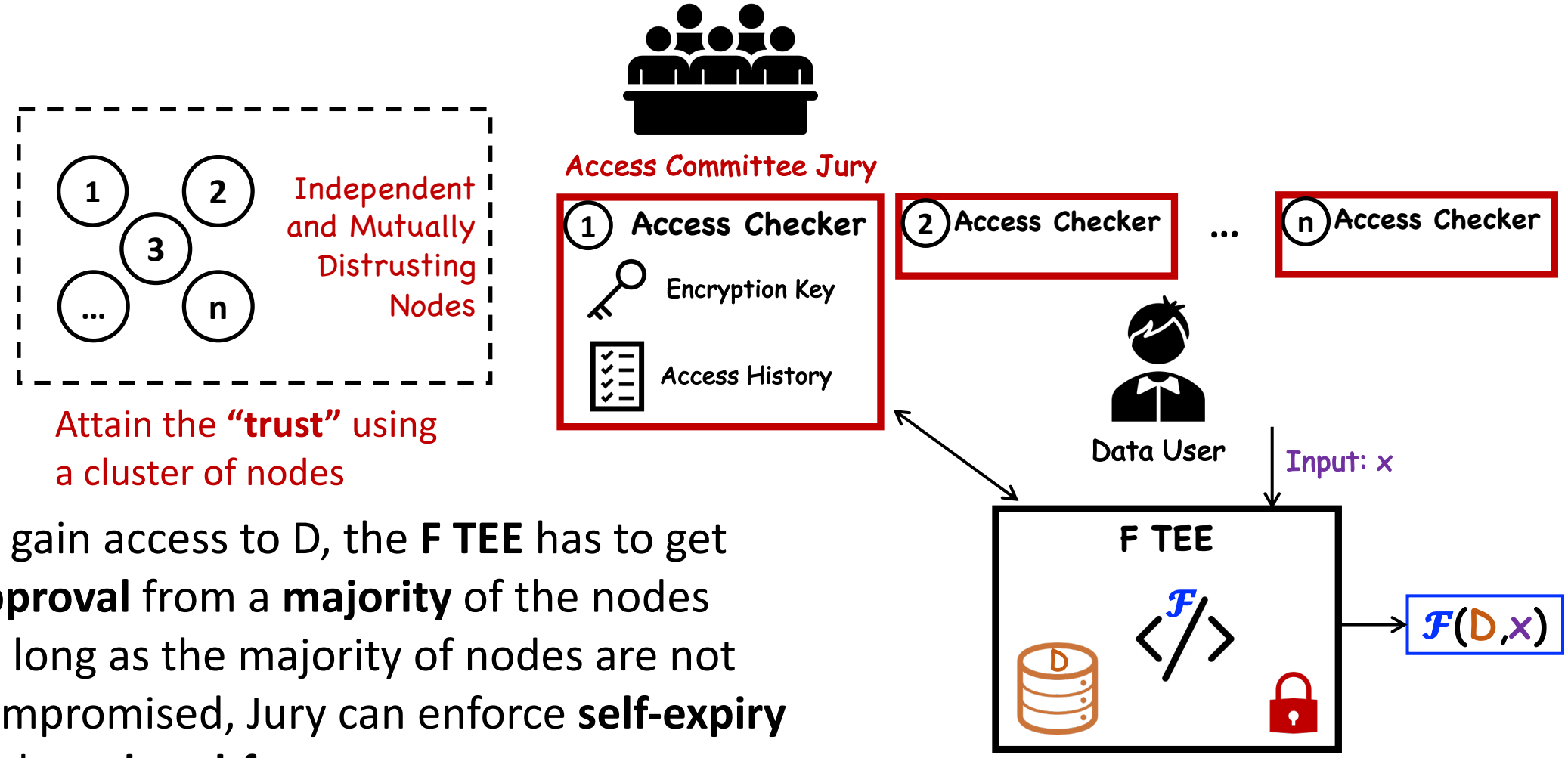


- Self-Expiry
- Send-and-Forget

- Functional Access to D

build up the platform

From Trusted Third Party to Access Committee



- To gain access to D, the **F TEE** has to get **approval** from a **majority** of the nodes
- As long as the majority of nodes are not compromised, Jury can enforce **self-expiry** and **send-and-forget**.

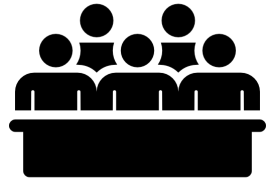
- Functional Access to D

build up the platform

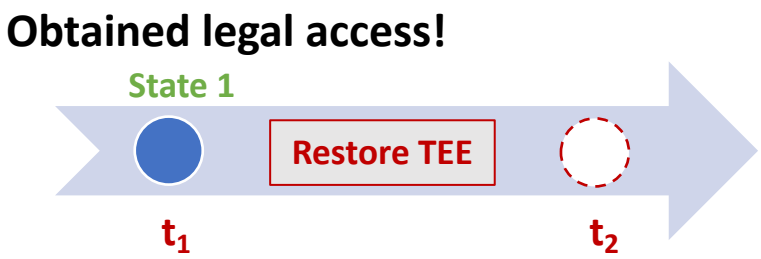
How Jury combat the rollback attacks on TEE

- Bind all the **steps** in the functional access into a single **session**,
- **Uniquely** identified by a **random number** generated by Jury!

- Self-Expiry
- Send-and-Forget



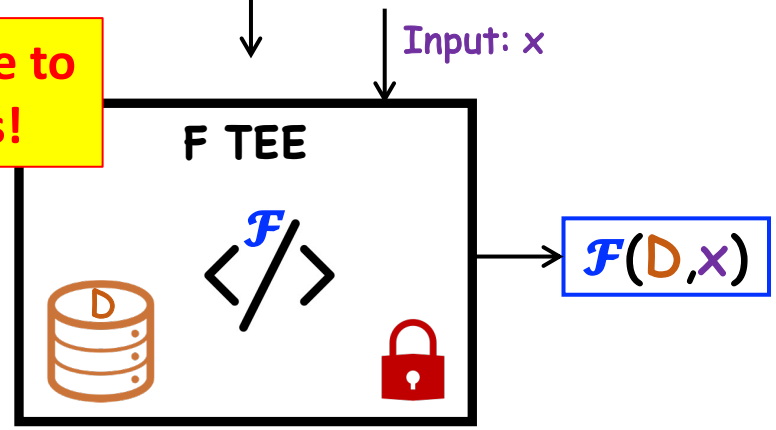
Access Committee Jury



At t_1, State 1 is sealed to disk

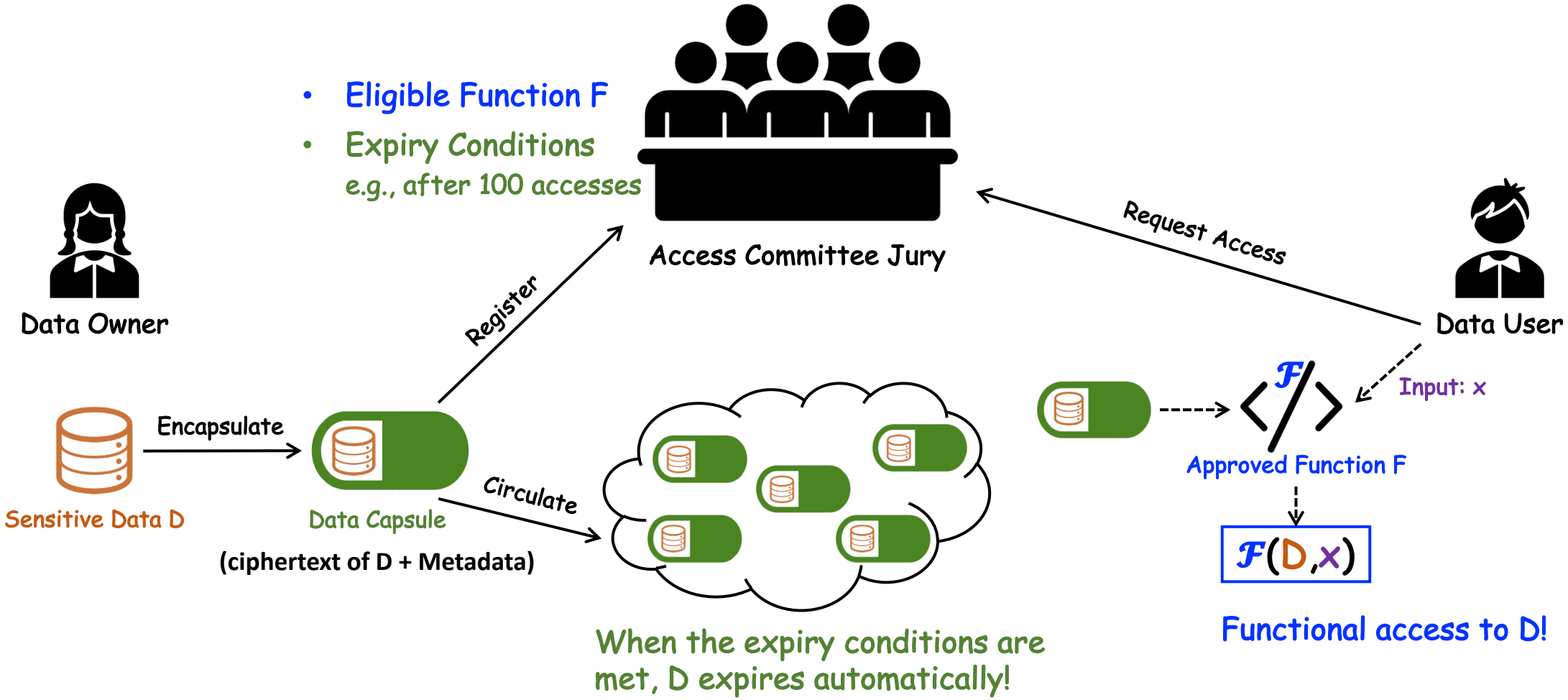
At t_2, the **restored TEE** can be fed with a different **x**

TEE is vulnerable to Rollback Attacks!



- Functional Access

TEEKAP: our data-sharing platform



Evaluation

- We built a protype using **Intel SGX**
- We conducted experiments with **realistic** deployment settings
- We focus on **latency** and **throughput** of the platform, as well as its **scalability**

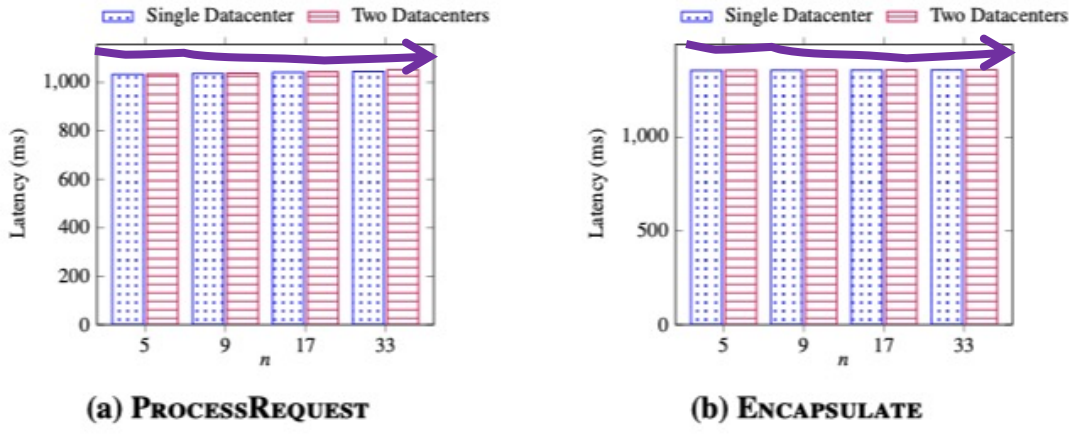


Figure 4: Latency of ProcessRequest and Encapsulate with respect to different Jury sizes (n) on Azure.

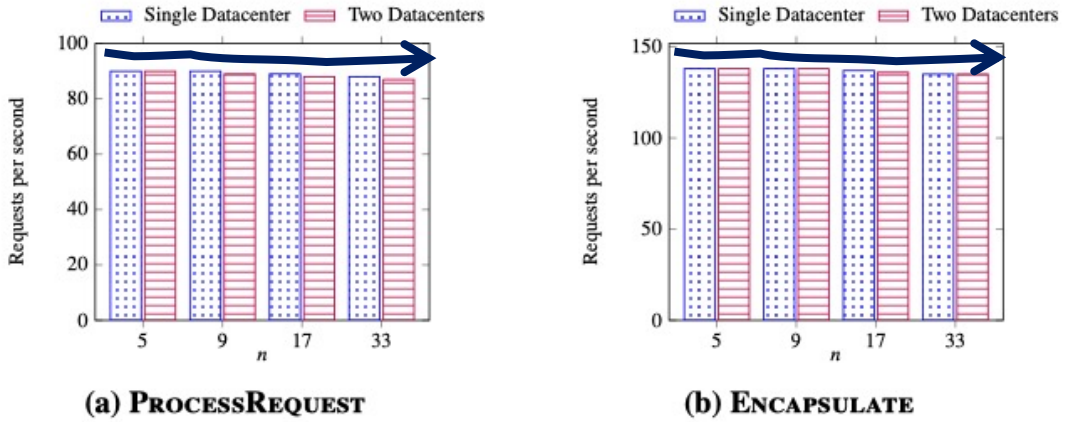


Figure 2: Throughput of ProcessRequest and Encapsulate with respect to different Jury sizes (n) on Azure.

Conclusion

- We proposed and formulated the problem of **self-expiring data encapsulation** that supports
  - Functional access
  - Generic user-defined expiry conditions

- We built a prototype system, conducted empirical experiments and demonstrated the **efficiency** of our proposal

Mingyuan Gao: mingyuan.gao@u.nus.edu

Hung Dang: hungdang@irics.vn

Ee-Chien Chang: changec@comp.nus.edu.sg