Tore Frederiksen

July 2024

Zama Threshold Key Management System (TKMS)

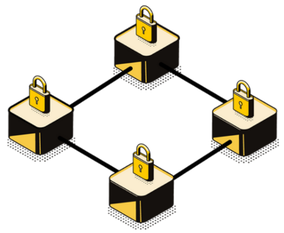
Threshold Key Generation and Decryption for fhEVM Chains

ZAMA

Motivation

EVM Blockchains

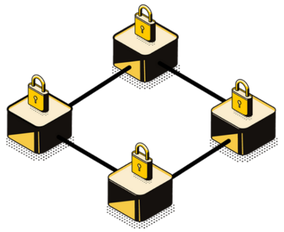
EVM Blockchain



- All transactions public
- All addresses public
- All smart contract code public
- All state variables public

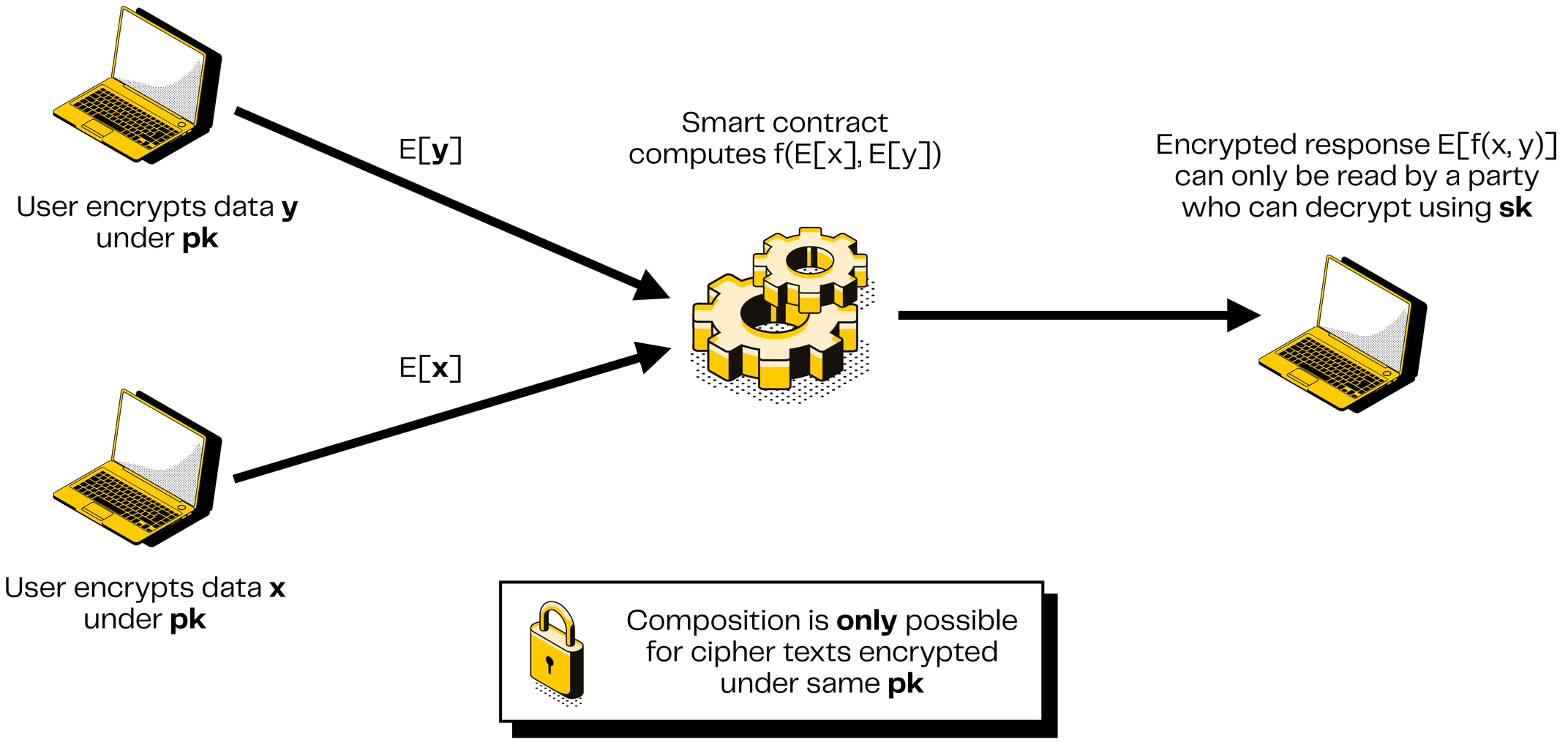
FHE

fhEVM Blockchain

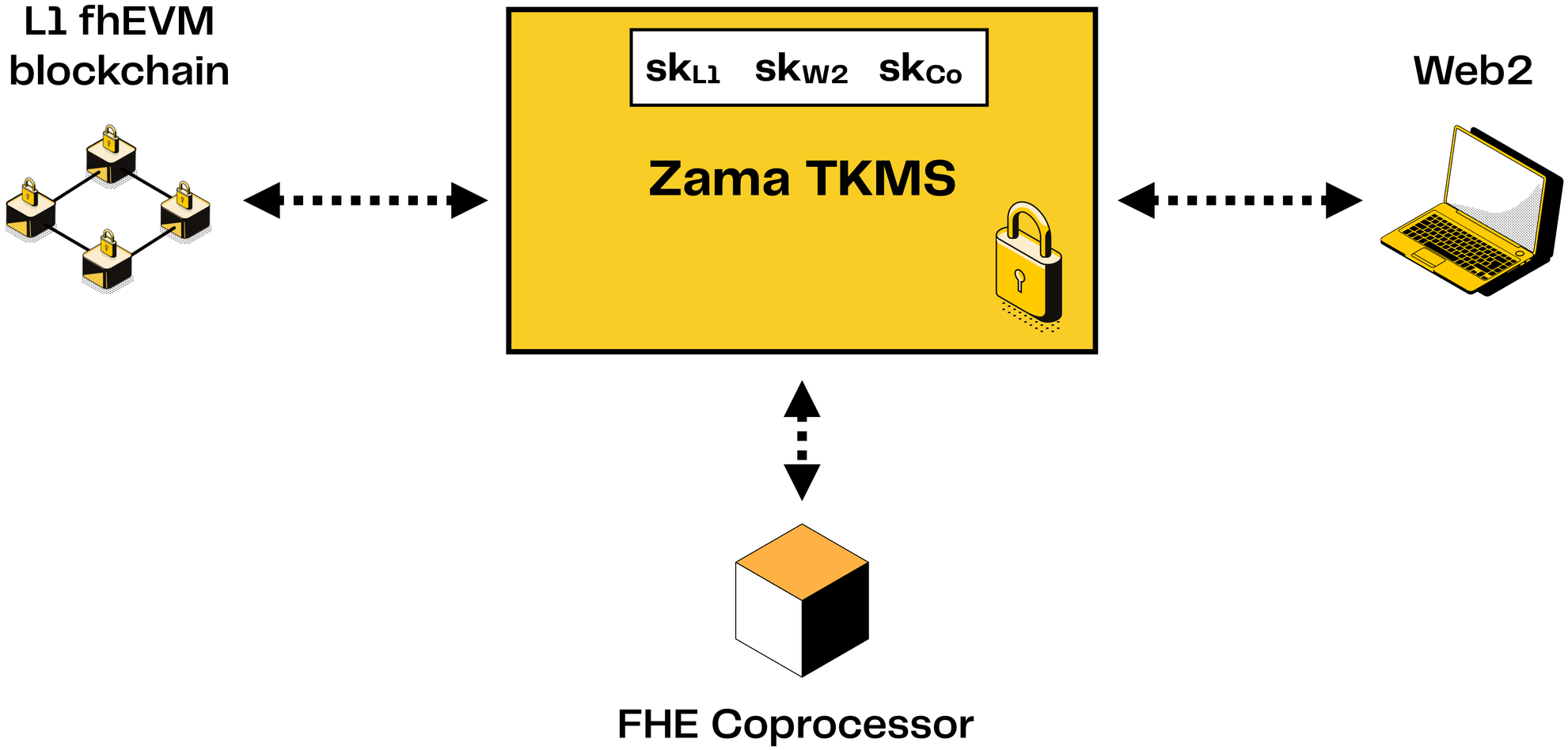


- All transactions public
- All addresses public
- All smart contract code public
- Chosen state variables **private**

Fully Homomorphic Encryption



Zama TKMS


L1 fhEVM blockchain
sk_{L1} sk_{W2} sk_{Co}
Zama TKMS
Web2
FHE Coprocessor

Zama's TKMS* enables secure and robust FHE key generation and decryption

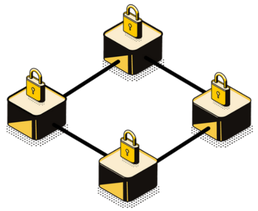
No centralized private key

Resistant to DoS attacks

Verifiable & Auditable

*a Key Management Service (KMS) is used to secure cryptographic keys

Zama's TKMS unlocks a myriad of multi-party FHE use cases



Blockchain

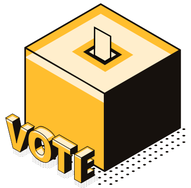
Confidential smart contracts with full composability.

Artificial Intelligence

AI model training on encrypted data from multiple parties.

Compliance & AML

Secure collaboration on KYC & AML between financial institutions.



Governance

Secure electronic voting for governments, companies and DAOs.

Defense

Secure collaboration between government agencies across borders.

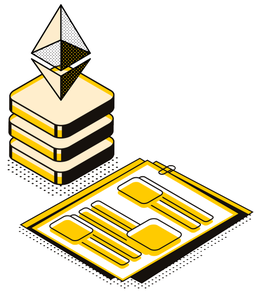
Healthcare

Confidential collaboration on research initiatives.

Zama's TKMS achieves enterprise-grade security* via a defense-in-depth approach

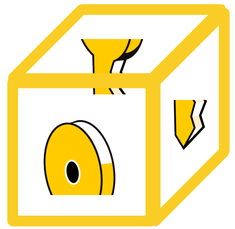
PoA Blockchain

The integrity and verifiability of decryption requests are ensured via a Proof of Authority blockchain consensus.



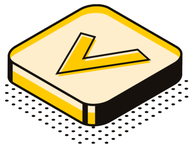
Threshold MPC

Security and liveness is guaranteed via a robust threshold MPC protocol for key generation and decryptions.



Secure Enclaves

Off-chain collusion is prevented by keeping the private key shares inside a secure enclave such as Amazon Nitro.



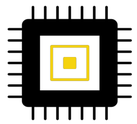
Legal Contracts

Nodes are chosen based on reputation and experience, and sign a legally binding agreement to behave honestly.

*Security audit and certification is ongoing, and will be completed soon

Keeping keys secure

How to manage a secret key

	Complexity	Cost $	Speed	Security assumption	Custom algorithms	Open source	Distributed	Built-in resilience	Malicious host protection
 Contract	**Simple**	**Cheap**	**Fast**	Legal system	✓	✓			✓
 HSM	**Simple**	Expensive	Slow	Honest Manufacturer	✗	✗	✗	✗	✓
 Enclave	**Simple**	**Cheap**	**Fast**	Honest Manufacturer	✓	✗	✗	✗	!
 MPC	Complex	**Cheap***	Slow	**Provable cryptography**	✓	✓	✓	✓	✗

Hybrid approach with security in focus

	Complexity	Custom algorithms	Cost $	Speed	Open source	Distributed	Security assumption	Malicious host protection
Zama TKMS	Complex		**Cheap***	Slow			Legal system **Provable cryptography** Honest Manufacturer	 

How we manage a secret key

Custom MPC protocol

- Provably secure
- Robust by design
- Based on top of peer-reviewed tier 1 academic research
- Custom generalizations
- Optimized for TFHE

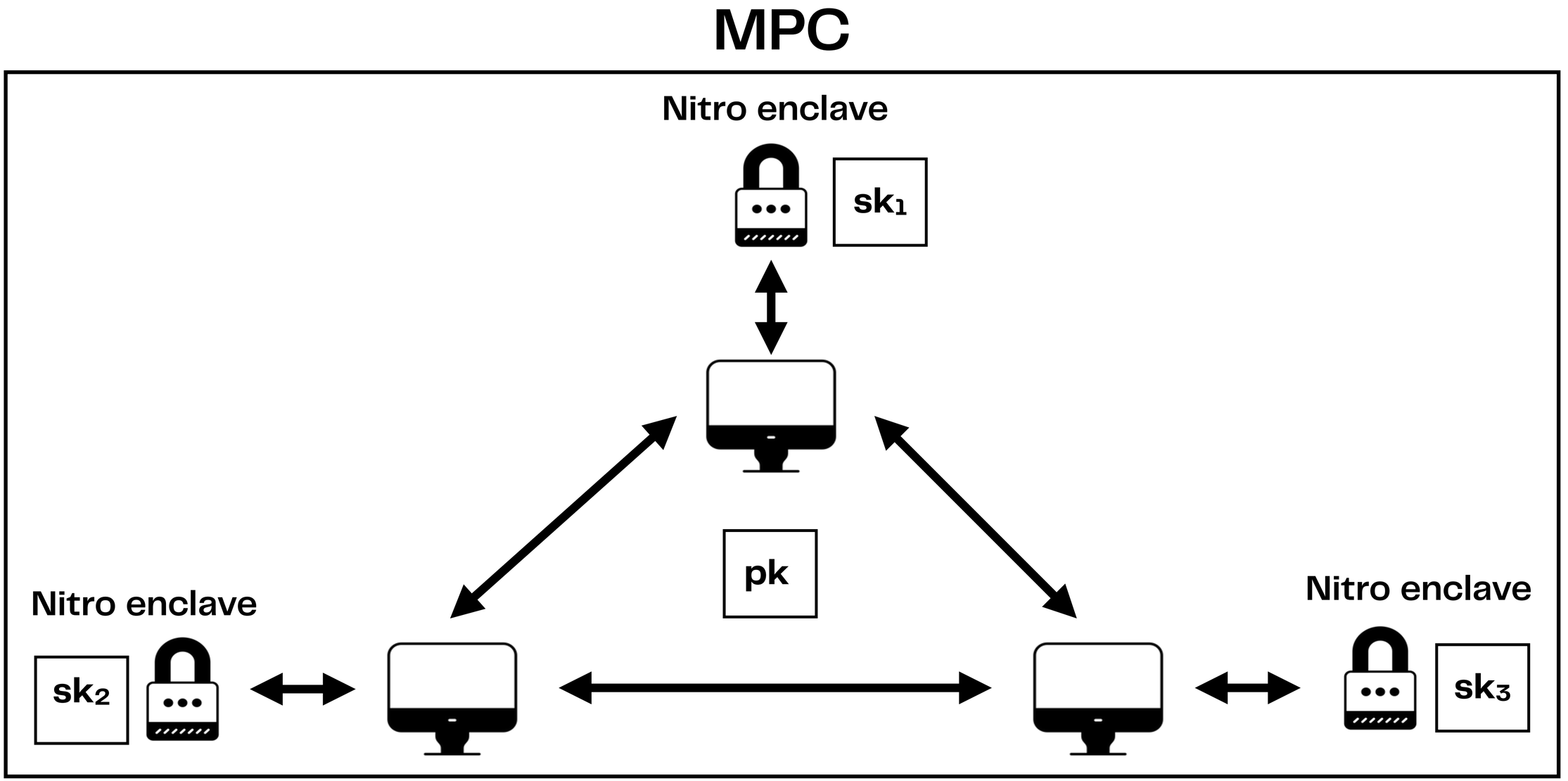
Use of Nitro enclave

- Secret shares stored in independent enclaves
- MPC protocol run between independent Nitro enclaves

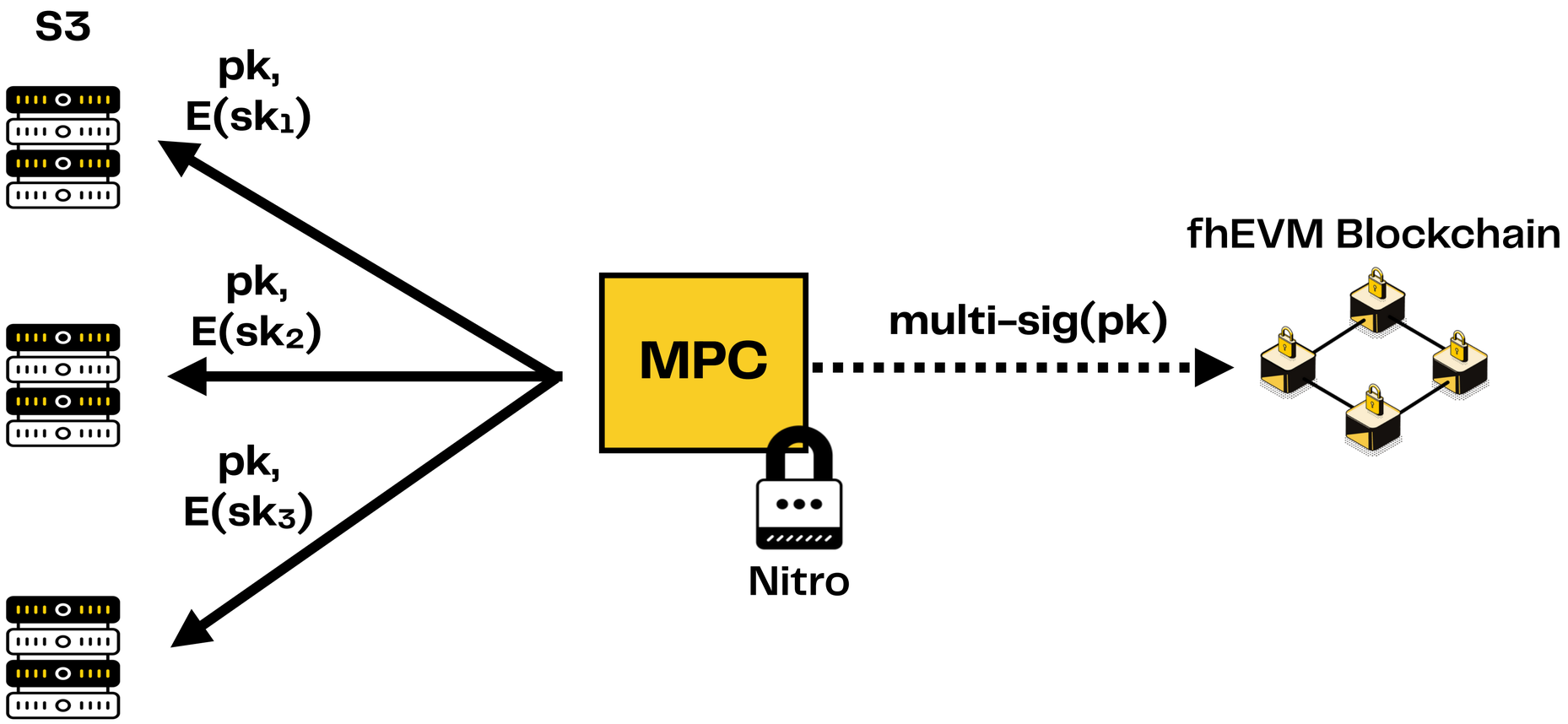
Community Contributions

- First commercial/production implementation of **robust** general MPC
- Submission for NIST standardization of threshold cryptography
- Explicit formalization MPC for TFHE key generation, decryption
- Multiple academic papers required for our construction
  - [Lib23] ZKP
  - [DDK+23] TFHE robust decryption
  - [Sma23] Secure computation model

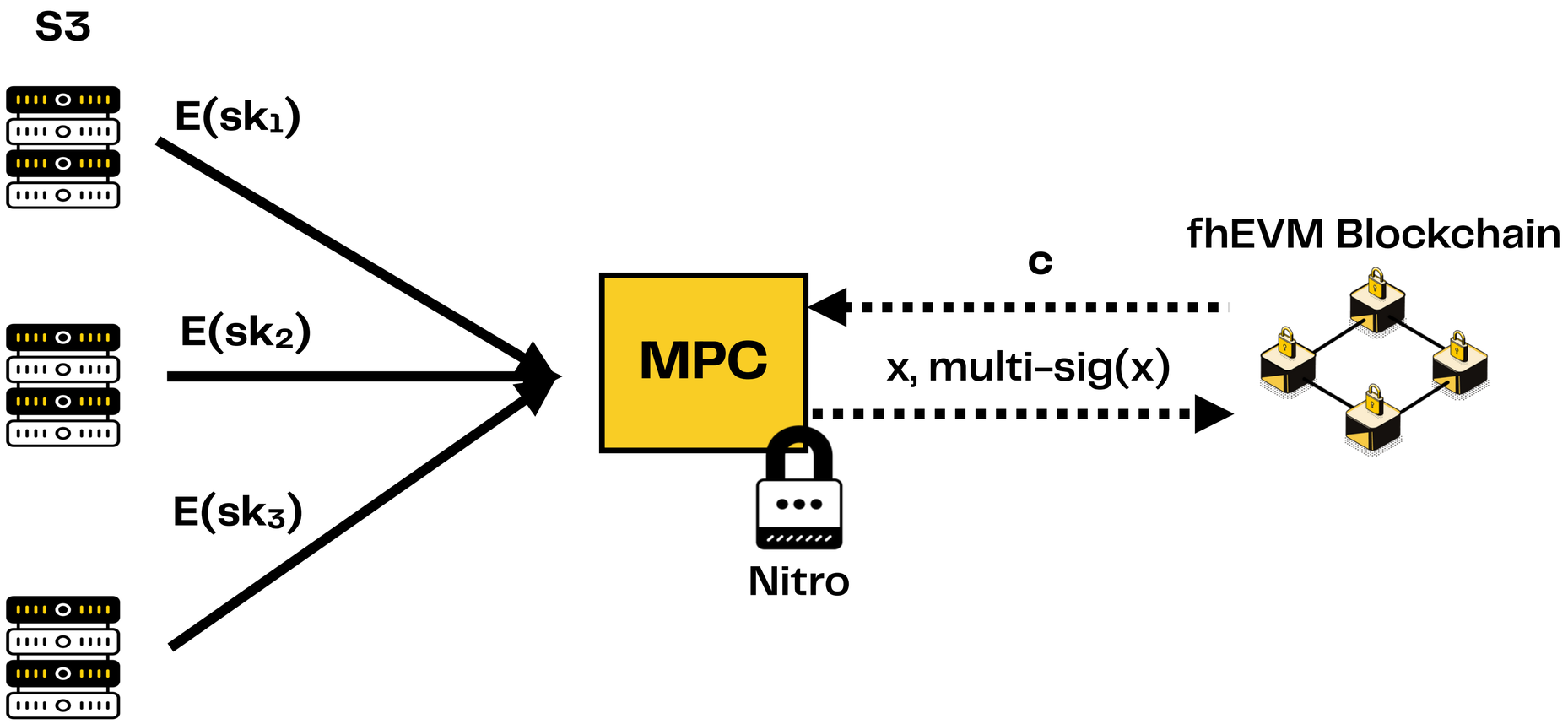
How we setup keys



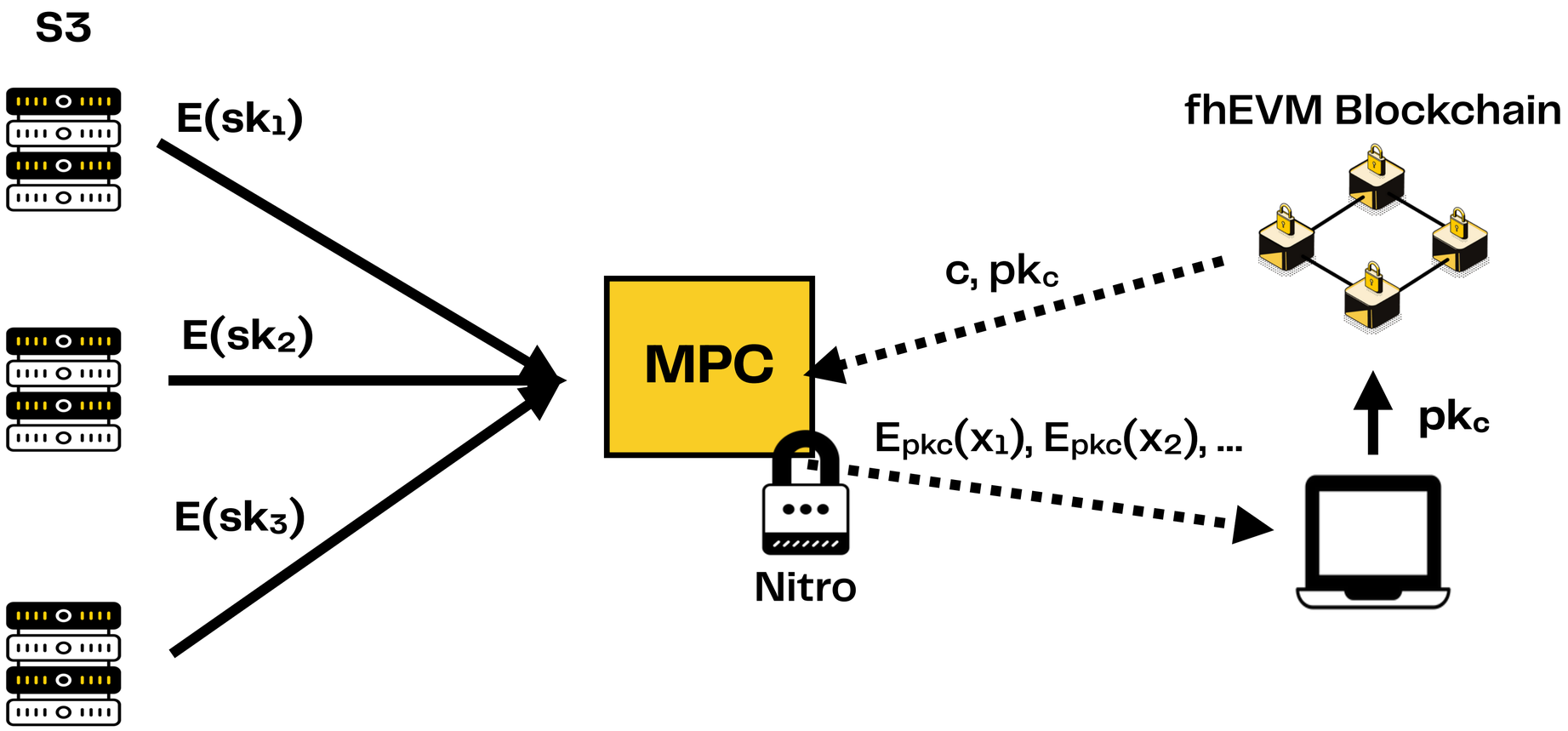
How keys are stored



How decryption works



How reencryption works



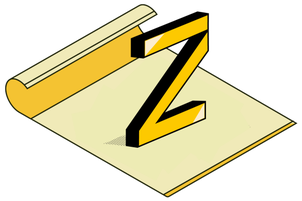
MPC protocol details

Requirements

- Sampling random bits
- Working modulo arbitrary prime-power (including 2)
- Work with arbitrary amount of parties

Features

- Robust (G.O.D.) MPC protocol
- Simulation-based security proofs
- Static, but proactive security
- PRSS [CLO+13] for Beaver triples when $\binom{n}{t}$ is small
  - n = amount of parties, t = corruption threshold
- VSS based protocol when $\binom{n}{t}$ is large



Academic Paper

https://eprint.iacr.org/2023/815.pdf

Connecting the components

Architectural challenges

- How are MPC servers paid?
- How are MPC servers updated?
- How do we audit the actions of the MPC servers?
- How do we link MPC parties to the fhEVM?
- Do we need a set of MPC servers for each fhEVM?
- Do the MPC servers need custom software for each fhEVM?

Tokens

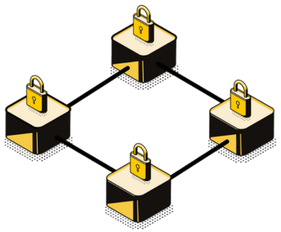
dApp, DAO

Public ledger

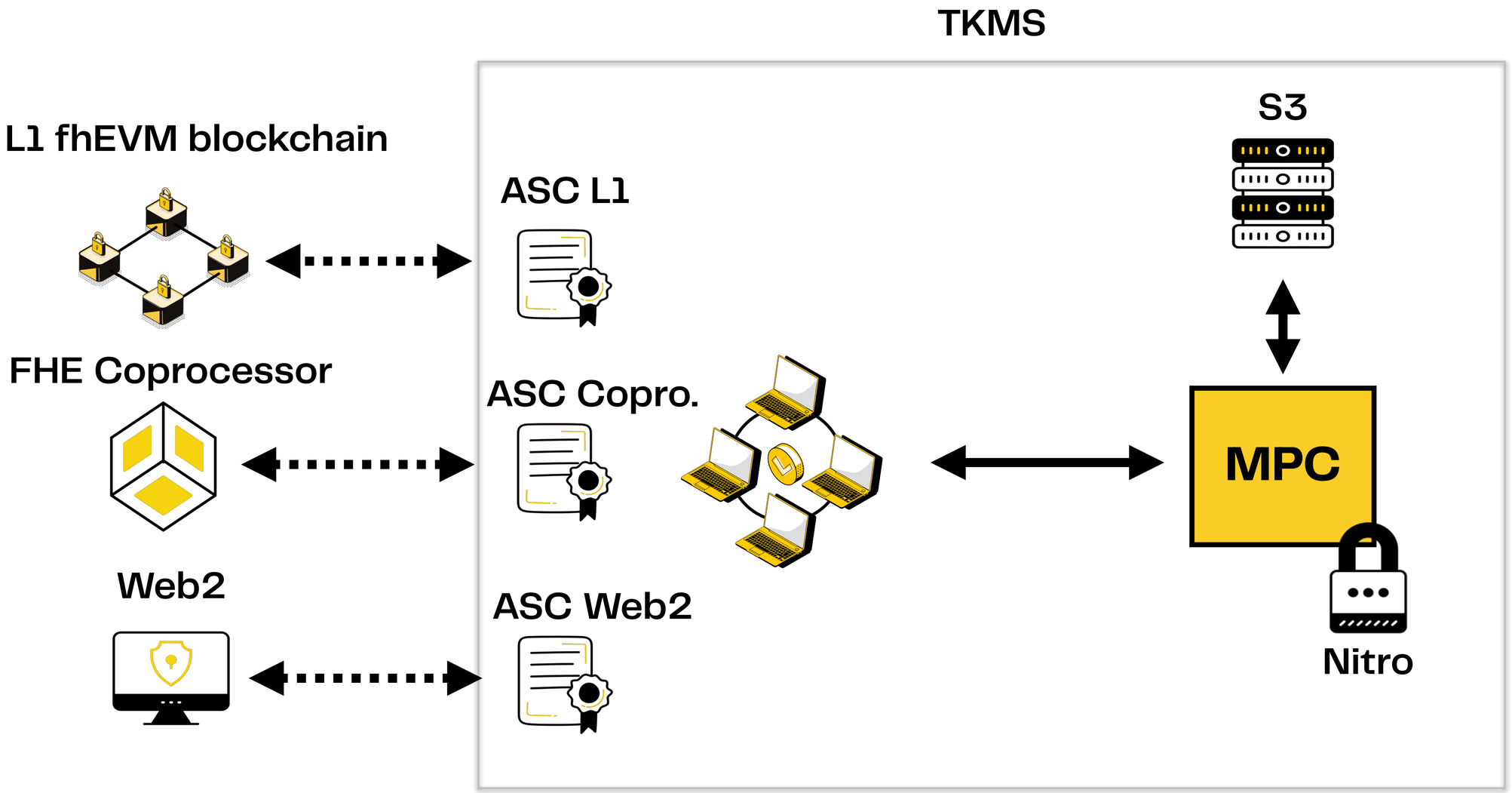
dApp, DAO

No, dApp

No, dApp



Zama's TKMS


TKMS
L1 fhEVM blockchain
FHE Coprocessor
Web2
ASC L1
ASC Copro.
ASC Web2
S3
MPC
Nitro

TKMS Blockchain

Requirements

- Low latency
- High throughput
- Fast finality

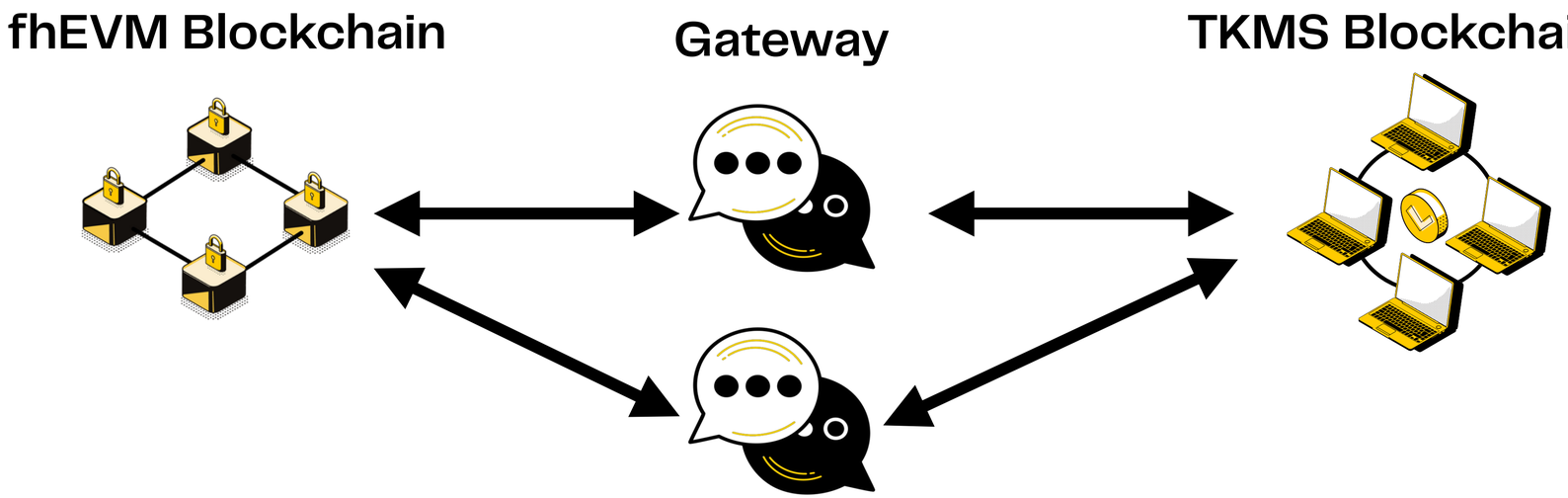
Alleviation

- Does not need to be Turing complete
- Can be permissioned
- Can consist of only a few servers (similar to MPC protocol)

TKMS Blockchain

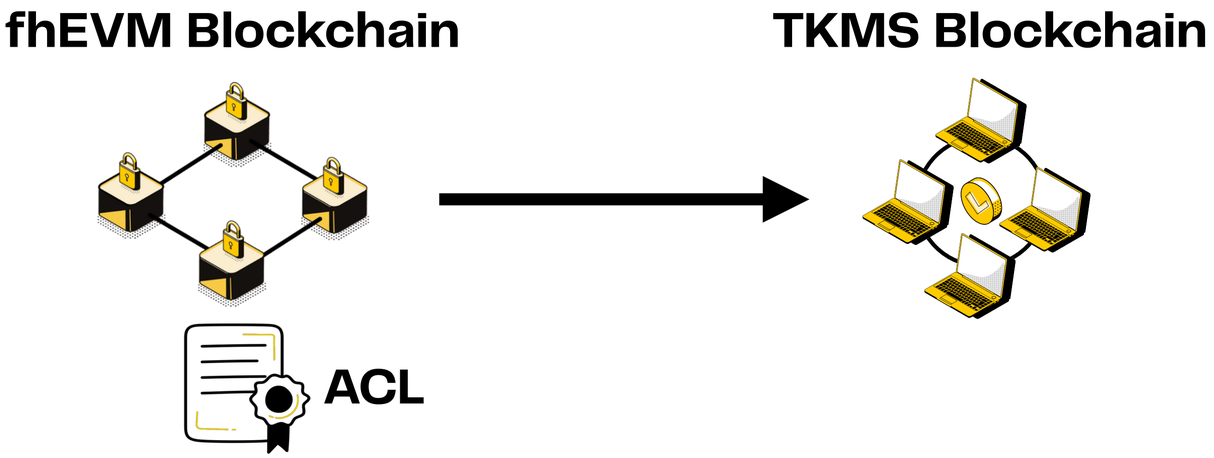
LATENCY SEC	**~ 1 SEC**
FINALISATION SEC	**~ 1 SEC**

fhEVM and TKMS blockchain connection

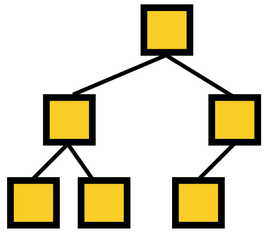


Requirement: Non-centralized trust for secrecy, correctness and reliability

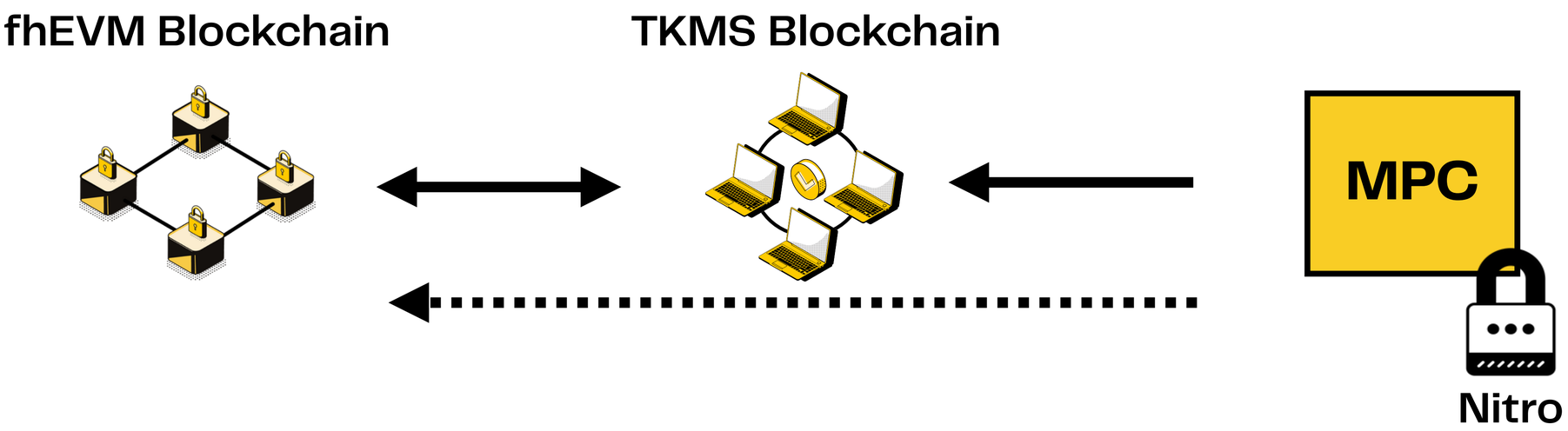
fhEVM and TKMS blockchain connection



- Keep permissible decryptions in an Access Control List (ACL)
  - Merkle tree
- Keep ACL root in block header
- Decryption requests contains signed block header with proof
- TKMS blockchain will validate proofs and signed headers (based on fhEVM validators)

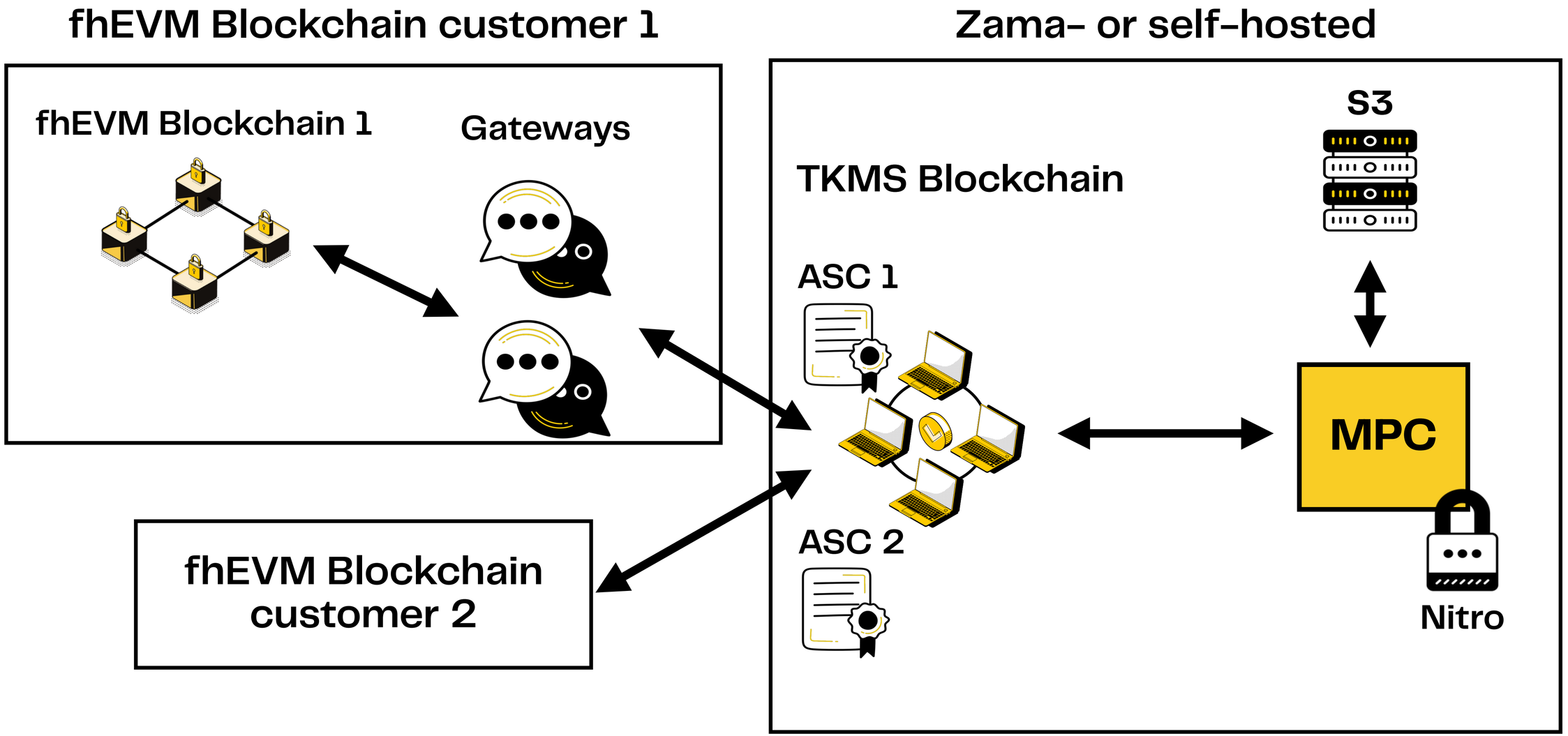


fhEVM and TKMS blockchain connection



- Decryptions and reencryptions signed by MPC servers
  - Validated on the fhEVM blockchain
- Updates of MPC servers will happen with consensus on the TKMS blockchain

E2E Flow


fhEVM Blockchain customer 1
fhEVM Blockchain 1
Gateways
fhEVM Blockchain
customer 2
Zama- or self-hosted
S3
TKMS Blockchain
ASC 1
ASC 2
MPC
Nitro

Benchmark

Decryption time in seconds

TYPE	ENGINE		TKMS E2E	TKMS E2E –> FHEVM
	T= 1	T= 4	T= 1	T= 1
EBOOL	0.002	0.54	2.3	5.1
EUINT4	0.006	0.64	3.5	5.4
EUINT8	0.007	0.58	3.7	5.6
EUINT16	0.013	0.59	3.8	5.7
EUINT32	0.019	0.58	3.9	5.8
EUINT64	0.029	1.1	4.0	6.0
EUINT2048	0.72	2.3	6.8	11

Security

Trust Model

No single point of failure

- **Secrecy**, **correctness** and **reliability** is distributed
- **Secrecy, correctness** can only be broken by
  - > t corrupt MPC and breaking Nitro
  - > t corrupt TKMS blockchain validators
- **Reliability** can only be broken by
  - > t corrupt MPC servers
  - > t corrupt TKMS blockchain validators
  - All gateways corrupt/unavailable

Assumption:

- Cryptography is secure
- Code is secure
- fhEVM is not broken
- t < n/3

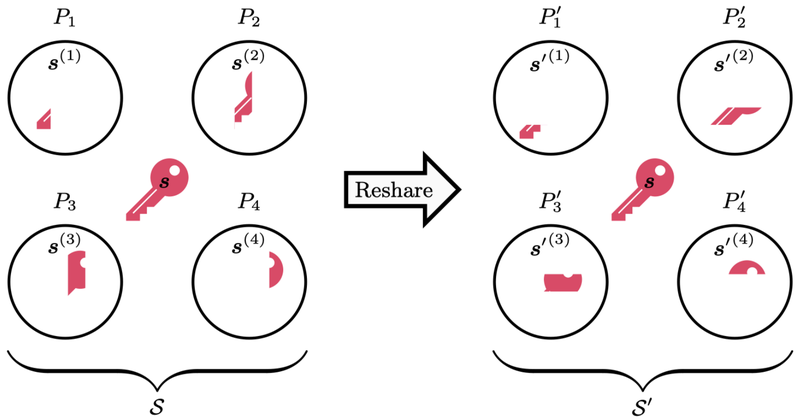
Further security considerations

Selective failures

- Decryption of malformed ciphertexts can leak some entropy about the secret key
- Prevention of this **must** be done by caller/key owner
  - fhEVM ASC contract
  - Zero-knowledge proof [Lib23]
    - Powers-of-tau

Refreshing

- In case of leak of a secret share by an MPC server, **all** shares can be refreshed efficiently **without** reencryption
- Highly efficient
- Allows for efficient and secure rolling of MPC parties



Deployment

Roadmap

Cryptography

• TLS	Done
• Signing encryption	Done
• FHE	Done
• MPC	Done
• ZKP	Done
• External audit	Coming soon...
• E2E validation	Coming soon...

Infrastructure

• Blockchain	Done
• Smart contracts	Done
• AWS deployment	Done
• Nitro	Done*
• S3 storage	Coming soon...
• Horizontal scaling	Coming soon...
• E2E communication	Done
• Testnet	Done

Advanced features

• DoS prevention	Coming soon...
• Updating MPC parties	Coming soon...
• Open sourcing	Coming soon...
• Centralized version	Done
• 4 MPC parties	Done*
• Up to 16 MPC parties	Coming soon...

Deployment

Open Source

Features

- Complete TKMS stack, including blockchain, MPC protocols, etc.

Support

- Community support via dedicated Discord channel on FHE.org

License

- Free for personal use, prototyping, evaluation & research

Hosted

Features

- Key generation, decryption and reencryption as a service
- High level APIs and developer console
- Fully transparent and auditable

Support

- 24/7 Premium support
- 99% uptime guarantee

License

- Transaction fees + Support

Self-Managed

Features

- Deploy a TKMS onto your own infrastructure
- Integrate with your own developer console and APIs
- Customize the number of PoA and MPC nodes
- Choose between multiple TEEs

Support

- 24/7 Premium support
- Integration support

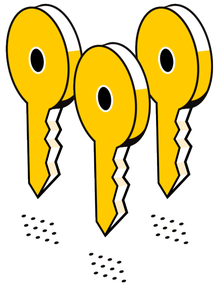
License

- Volume based license + support

Using Zama's TKMS is easy as 1–2–3

1. Configure the TKMS

The TKMS can be configured via Zama's developer console, or by uploading a configuration smart contract to the TKMS chain.



2. Generate your FHE keys

Threshold key generation is triggered via Zama's developer console, or by submitting a transaction to the TKMS chain.

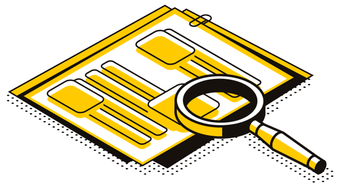
3. Decrypt Ciphertexts

Decrypting FHE ciphertexts is done via an API call, or by submitting a transaction directly to the TKMS chain.

Immerse yourself in the TKMS

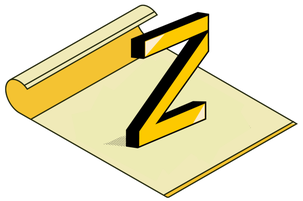
Github

Coming soon...



Documentation

Coming soon...



White Paper

Coming soon...

Thank you.

ZAMA

Contact and Links

tore.frederiksen@zama.ai

zama.ai

Github

Community links

ZAMA