Morten Dahl

27 May 2024

The Zama fhEVM:

Confidential smart contracts using FHE + ZK + MPC

ZAMA

Onchain data is public by design, making it hard to build dapps that require confidentiality

Zama's fhEVM

Transactions | Internal Txns | Erc20 Token Txns | Erc721 Token Txns | Erc1155 Token Txns | Analytics | Comments

Latest 25 ERC-20 Token Transfer Events — View All

Txn Hash	Age	From		To	Value	Token
0x61bac8ed64cf49ff537…	1 hr 8 mins ago	Uniswap V2: KCAL 2	IN	vitalik.eth	2,500	Step.app (KCAL)
0xd9f47a344e278579cb…	1 hr 15 mins ago	Justin Sun	IN	vitalik.eth	25,143,213.150843308745475521	Step.app (KCAL)
0xdea02c32d141997aaa…	12 hrs 57 mins ago	plamer.eth	IN	vitalik.eth	1	AssangeDAO (JUSTIC...)
0x74205c19a313ba8865…	1 day 11 hrs ago	Uniswap V2: SEGA 3	IN	vitalik.eth	227,158,544.808096280091774569	SEGA (SEGA)
0xad5c19e1af6de6508e…	2 days 20 hrs ago	0xad29c28a868c945caf9…	IN	vitalik.eth	21,420	ERC-20 (BASTAR...)
0x1014024546d2e94f39…	3 days 4 mins ago	Uniswap V2: ALIS 2	IN	vitalik.eth	153,473.76198500365822856	Acropolis DA... (ALIS)
0xbffdb2fcd52e96f136c7…	3 days 24 mins ago	vitalik.eth	OUT	OlympusDAO: DAO Funds	40,323.284453294043855726	Acropolis DA... (ALIS)
0x6ac57444413cd7bbef…	3 days 31 mins ago	Uniswap V2: ALIS 2	IN	vitalik.eth	40,323.284453294043855726	Acropolis DA... (ALIS)
0xb15136c85e15dd81b3…	3 days 1 hr ago	OlympusDAO: DAO Funds	IN	vitalik.eth	8,633.511805120159396357	Acropolis DA... (ALIS)
0xa9749c78f8ed9da996…	3 days 14 hrs ago	Uniswap V2: Bvlgari	IN	vitalik.eth	3,853,058,515,307.2989734036202684...	ERC-20 (Bvlgar...)
0x27fe35a36a42bbed75…	3 days 15 hrs ago	Uniswap V2: Bvlgari	IN	vitalik.eth	3,652,123,857,386.0501562459646840...	ERC-20 (Bvlgar...)

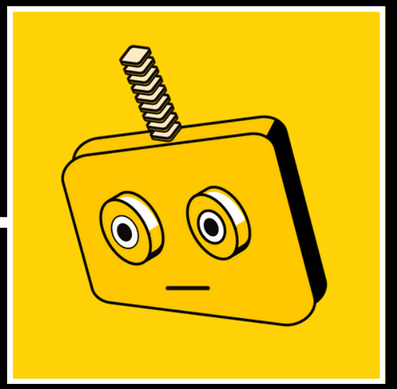
This leads to many privacy issues

Theft

Criminals know what you own, so they can easily target you and steal your crypto.

Surveillance

Governments can surveil you, even if you use multiple addresses.

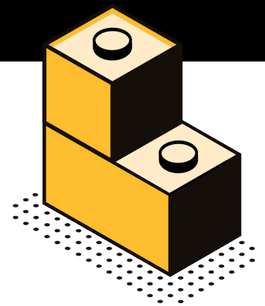


MEV

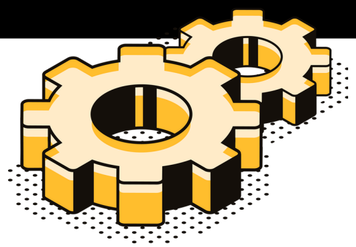
Bots can front-run you, creating a hidden tax on every transaction.

Zama's fhEVM enables confidential smart contracts using FHE

E2E encryption of transactions and state

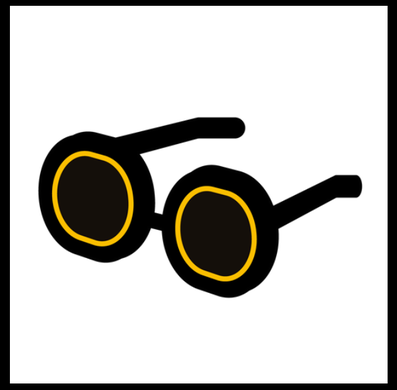


Onchain data availability



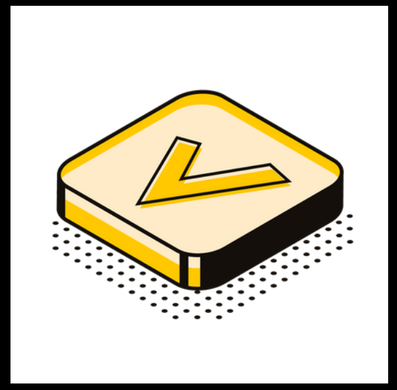
Doesn't break existing dapps and state

Without compromising transparency and usability



Computation

Users can still know what contracts are doing.



Access Control

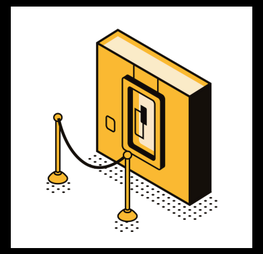
Contracts are free to implement their own access control logic.

Composability

It is easy to mix data from multiple users and compose smart contracts.

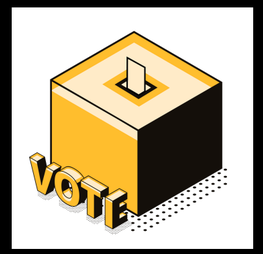
Zama's fhEVM unlocks new use cases

Tokenization
Manage and swap tokenized assets without other seeing it



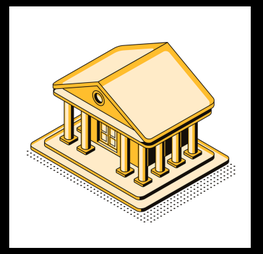
Blind Auctions
Bid on items without revealing the amount or the winner

Onchain Games
Hide cards and moves until reveal (e.g. poker, blackjack, ..)



Confidential Voting
Prevents bribery and blackmailing by keeping votes private

Encrypted DIDs
Store identities onchain and generate attestations without ZK



Private Transfers
Keep balances and amounts private, without using mixers

Technical overview

Zama's fhEVM combines state of the art cryptography in a provably secure way

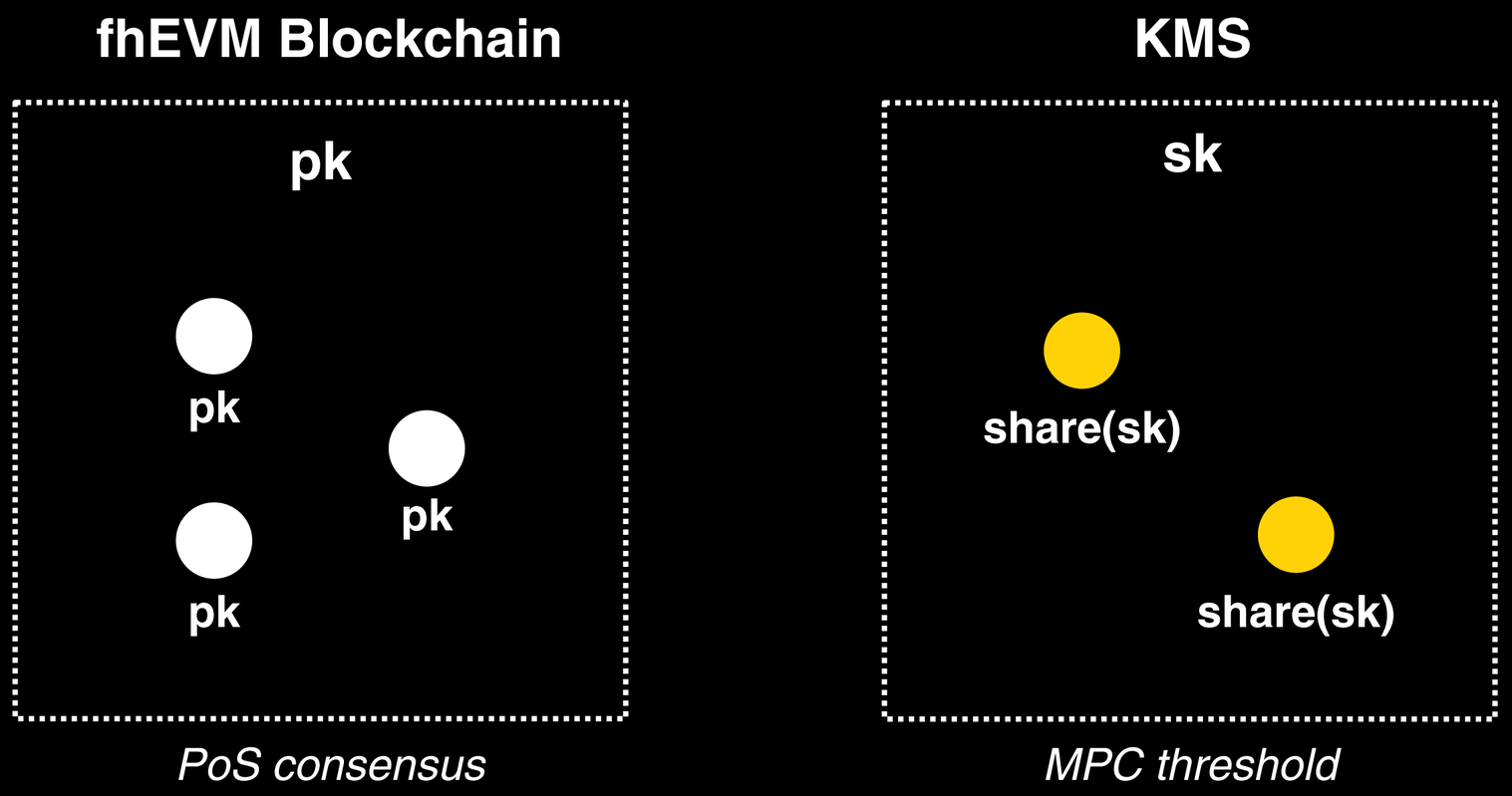
FHE + **MPC** + **ZK**

Homomorphic encryption is used to compute on private state, directly onchain

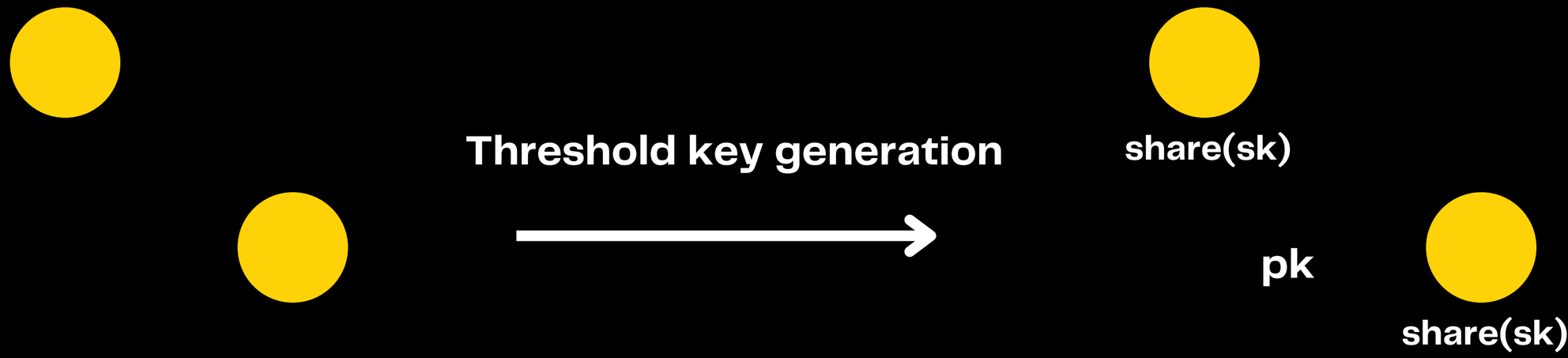
Multi-party computation is used for threshold decryption of FHE ciphertexts

Zero-Knowledge Proofs of Knowledge are used to ensure encryption and decryption integrity

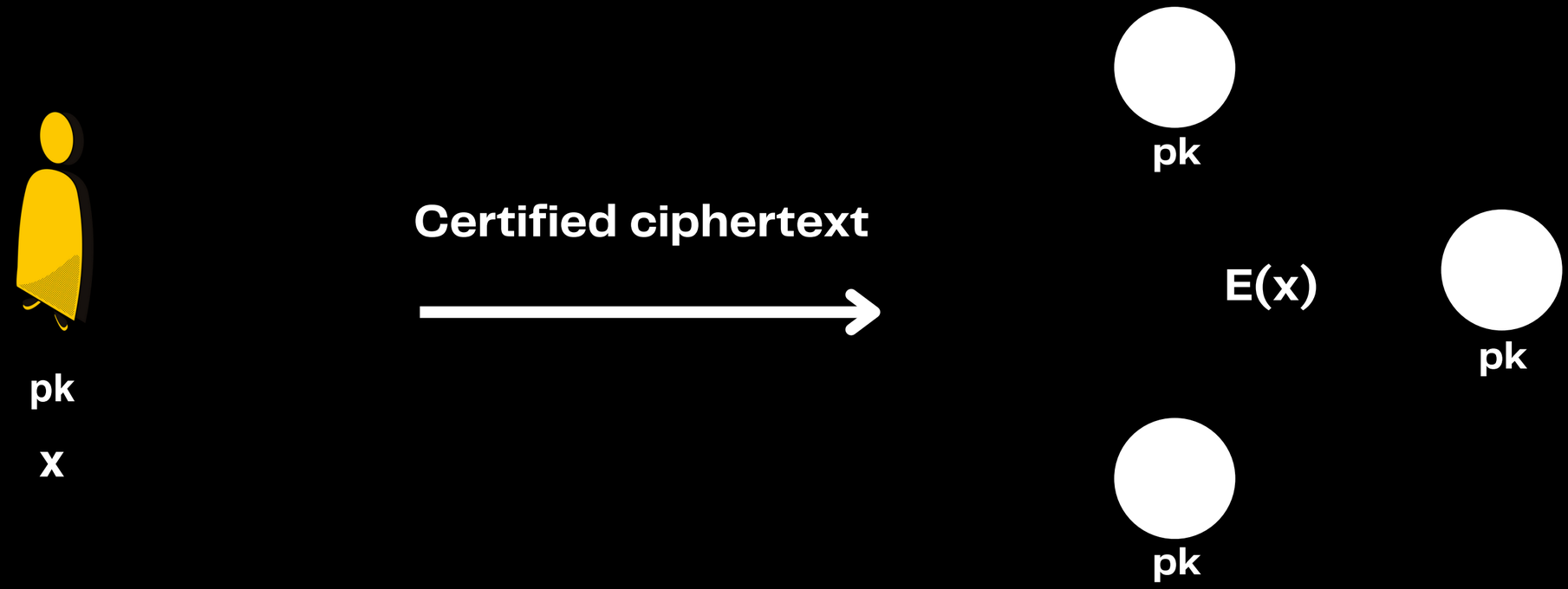
Everything is encrypted under single global FHE public key



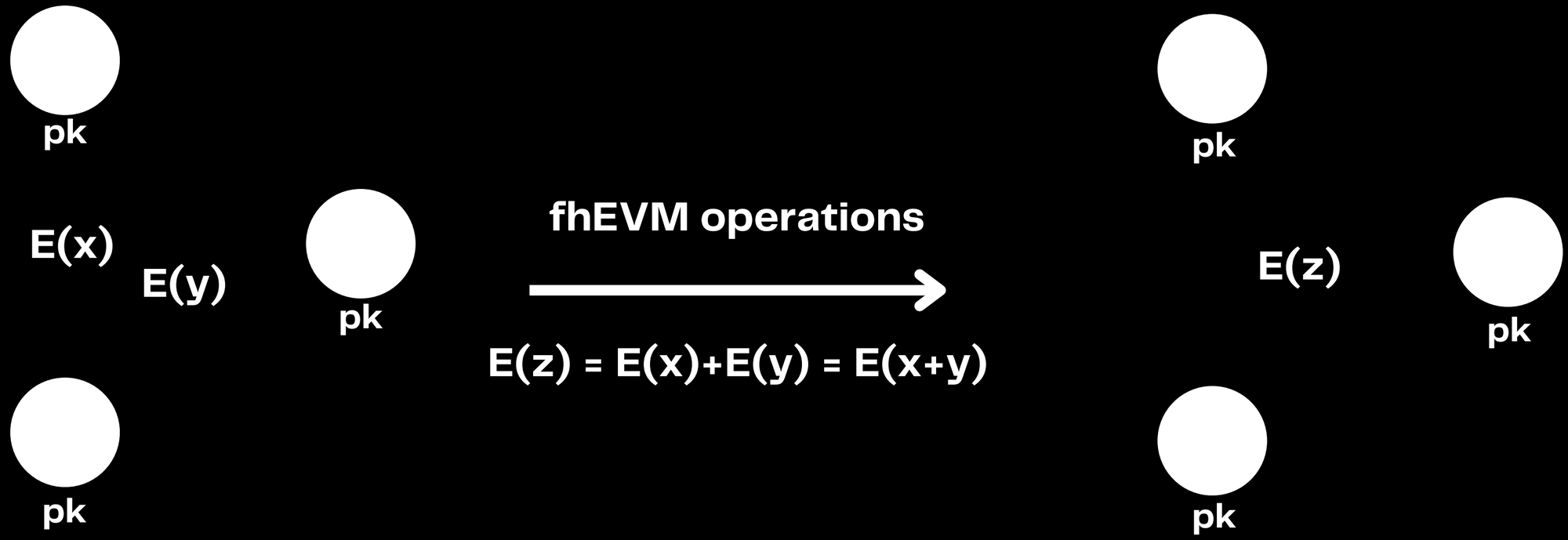
The global key is generated securely using a threshold protocol



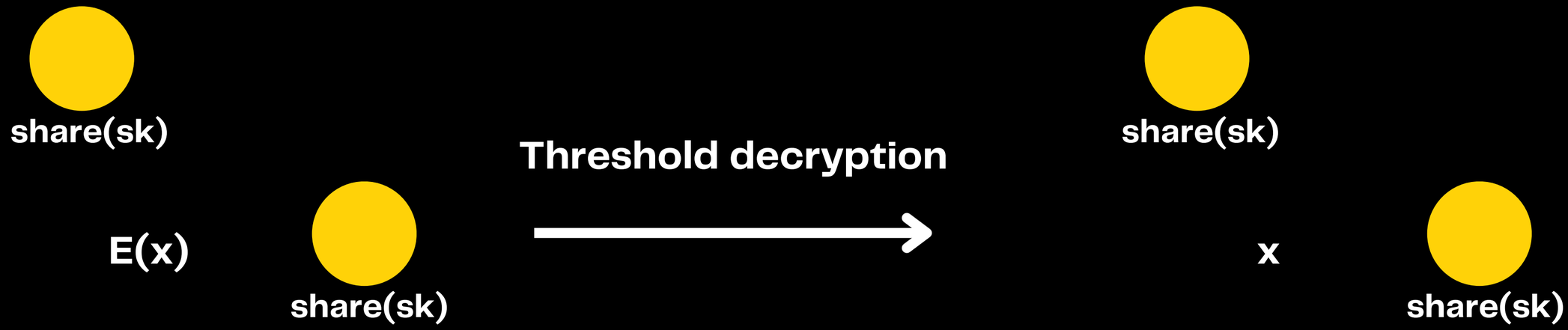
The inputs are simply encrypted using the global public FHE key



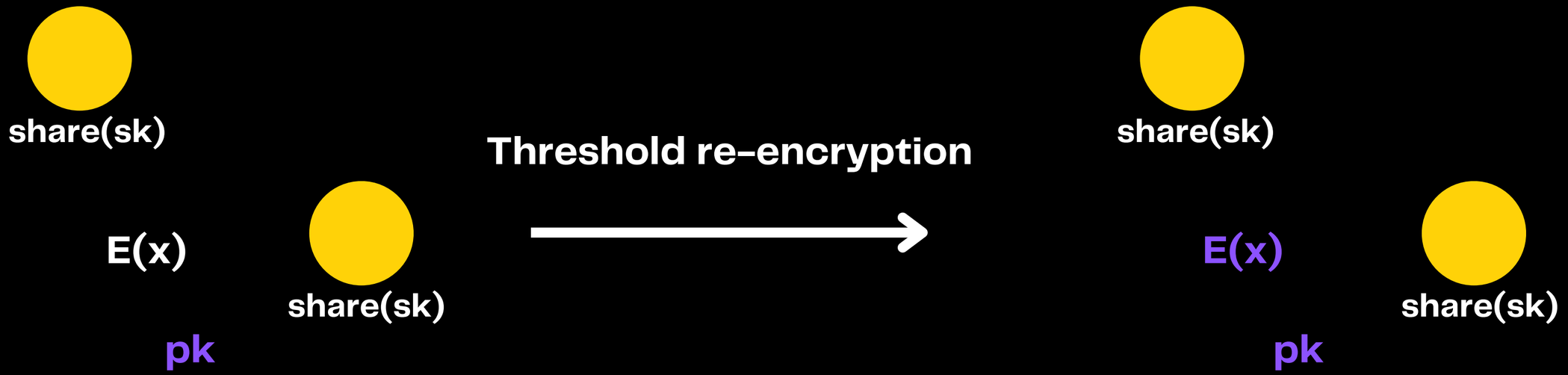
Computation is done locally by validators using homomorphic operations



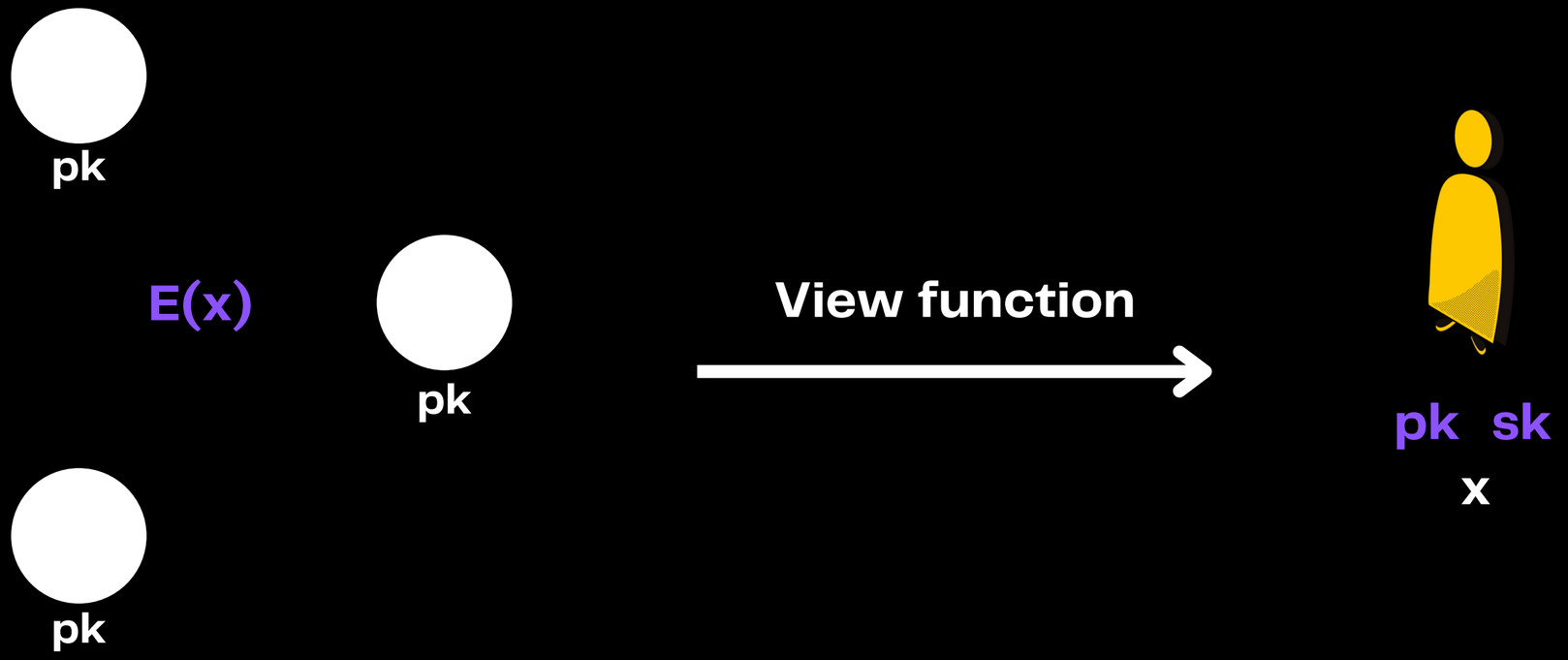
Values can be decrypted by validators using a threshold protocol



Values can also be re-encrypted to user public key using a threshold protocol



Re-encrypted values can be read and decrypted by the user owning the key



Solidity API

Solidity API

Developers can write confidential smart contracts without learning cryptography

```
contract EncryptedERC20 {
    // A mapping from address to an encrypted balance.
    mapping(address => euint64) internal balances;

    // Transfers an encrypted amount from the message sender address to the `to` address.
    function transfer(address to, bytes calldata encryptedAmount) public virtual returns (bool) {
        transfer(to, TFHE.asEuint64(encryptedAmount));
        return true;
    }

    // Returns the balance of the caller encrypted under the provided public key.
    function balanceOf(
        address wallet,
        bytes32 publicKey,
        bytes calldata signature
    ) public view virtual onlySignedPublicKey(publicKey, signature) returns (bytes memory) {
        if (wallet == msg.sender) {
            return TFHE.reencrypt(balances[wallet], publicKey, 0);
        }
        return TFHE.reencrypt(TFHE.asEuint64(0), publicKey, 0);
    }

    // Returns the encrypted balance of the caller.
    function balanceOfMe() public view virtual returns (euint64) {
        return balances[msg.sender];
    }
}
```

See an example contract

Solidity Integration
fhEVM contracts are simple solidity contracts that are built using traditional solidity toolchains.

Simple DevX
Developers can use the euint data types to mark which part of their contracts should be private.

Programmable Privacy
All the logic for access control of encrypted states is defined by developers in their smart contracts.

Solidity API

TFHE::euint32

```
mapping(address => euint32) balances;
```

- Represents an encrypted value.
- Can be used for computation, storage, composition, etc.
- Efficient since they are small (only handles to ciphertexts).
- euint8, euint16, euint32, euint64, ... add, sub, mul, eq, le, gt, ...

TFHE.asEuint

```
euint32 amount = TFHE.asEuint32(amountCiphertext);
```

- Well-formed to not leak anything about global FHE secret key.
- Prevent user from decrypting arbitrary ciphertexts.
- Ciphertexts include ZK proof of plaintext knowledge that must be checked.

TFHE.decrypt

```
require(TFHE.decrypt(TFHE.le(amount, balances[msg.sender])));
```

- Alternative is TFHE.cmux(eCondition, eTrueValue, eFalseValue).
- Decrypts leaks something, even when just used for checking requirements.

TFHE.reencrypt

```
return TFHE.reencrypt(balances[msg.sender], publicKey);
```

- Securely re-encrypt from global FHE public key to user NaCl public key.
- Optional authentication token to trust identity of sender (EIP-712).

Inside the fhEVM

Inside the fhEVM

Precompiled Smart Contract

- Calls out to Zama's FHE library (TFHE-rs)

Programmable Privacy

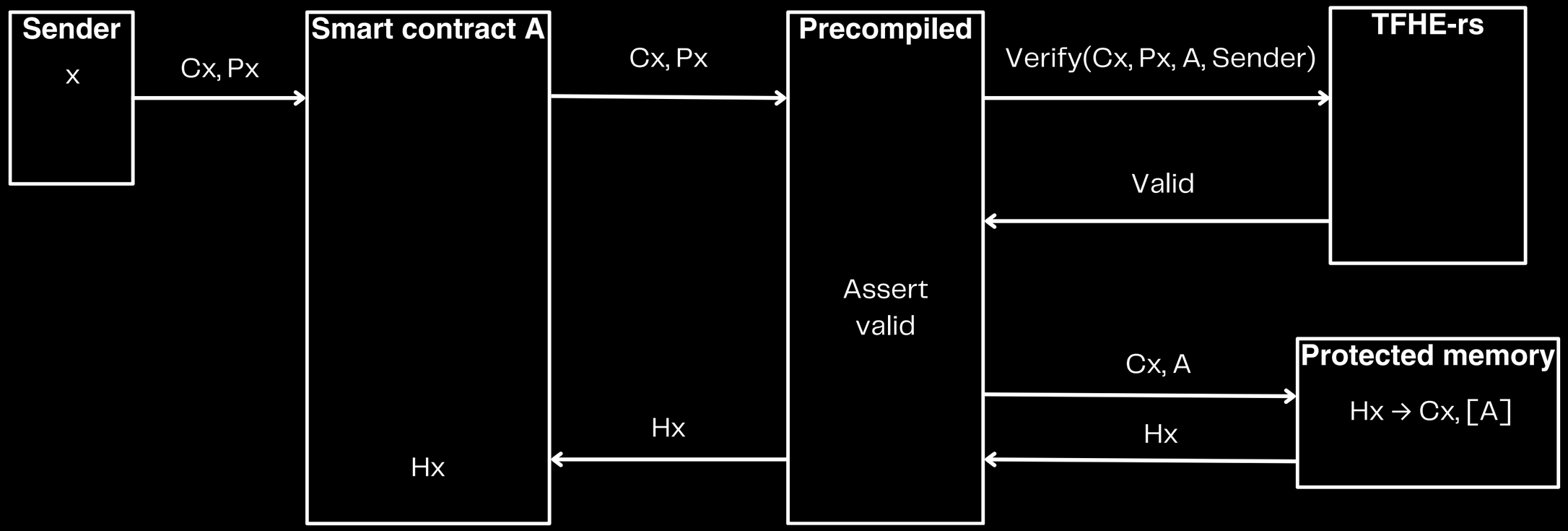
- Smart contracts decide what to decrypt
- Preventing misuse by limiting to ciphertexts that were honestly obtained

Smart contracts operate on handles to ciphertexts stored in protected memory

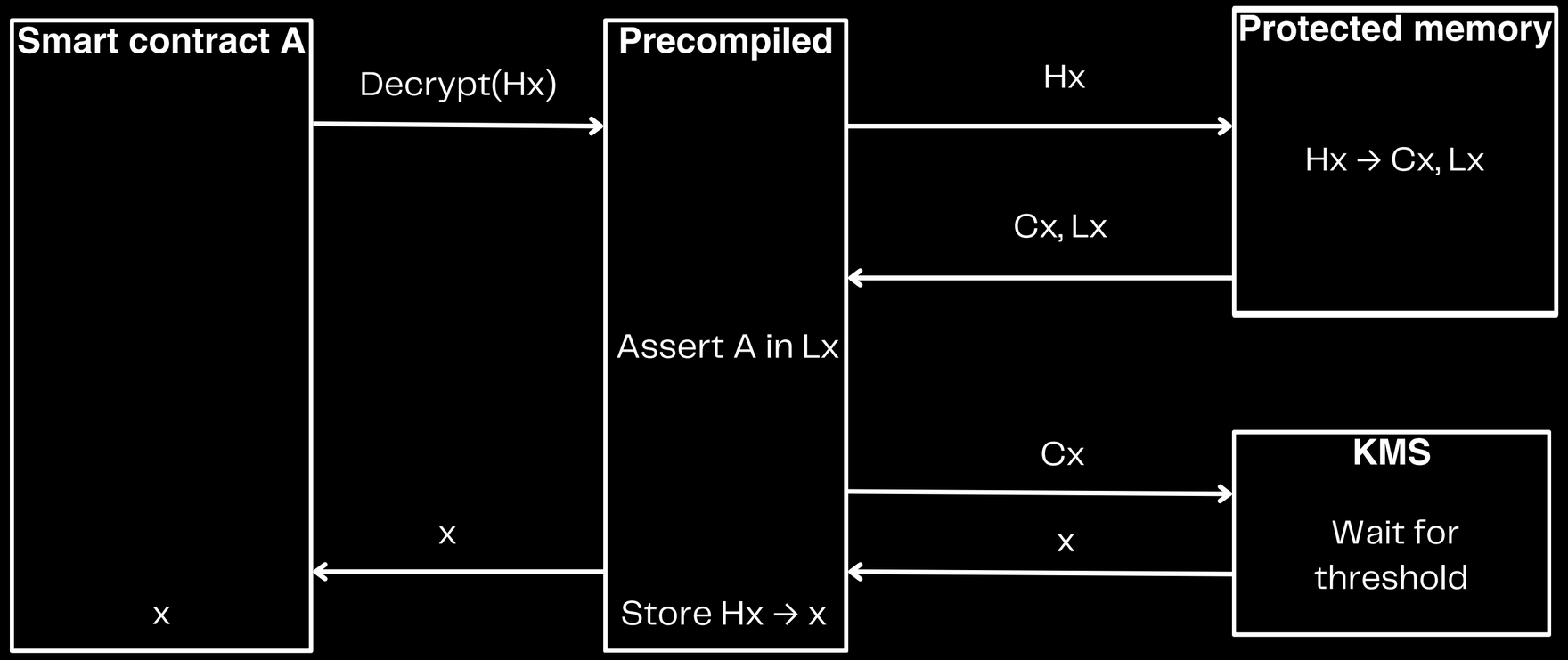
Hx = Hash(Cx)

Hx was honestly obtained by A if A in Lx

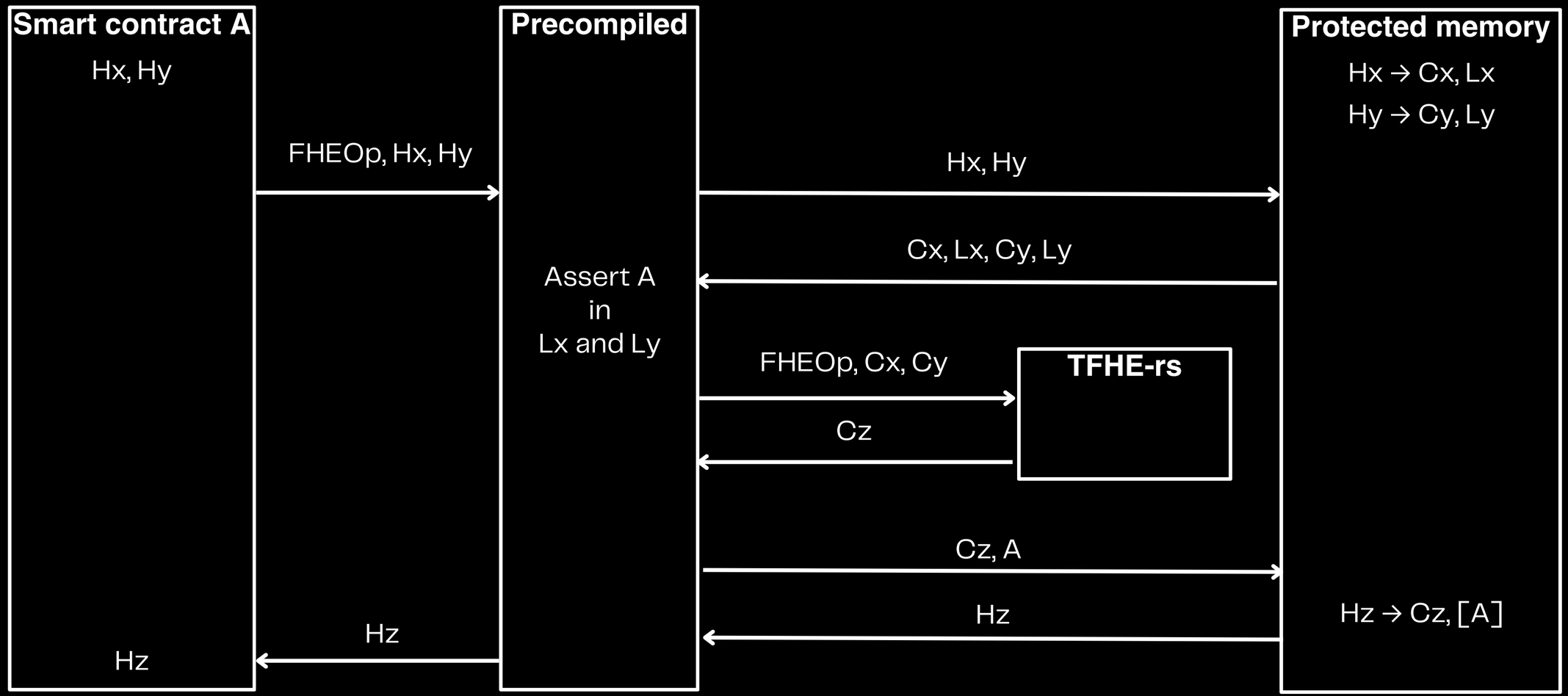
Verifying certified ciphertexts from users



Decryption only succeed if ciphertext was honestly obtained



Execution of FHE operation using TFHE-rs

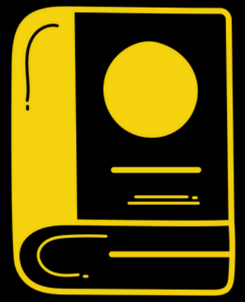


Wrapping up

Learn more

Try the fhEVM yourself today

Github



White Paper

Documentation

Zama's Bounty and Grant Program

Build FHE applications to tackle real-world privacy challenges using Zama's suite of libraries

All info: https://github.com/zama-ai/bounty-and-grant-program

Thank you!

morten.dahl@zama.ai

zama.ai

github.com/zama-ai

@zama_fhe

ZAMA