Private Smart Contracts Using Homomorphic Encryption

Morten Dahl, Levent Demir, Louis Tremblay Thibault

Summary:

- Encrypted blockchain state and transaction inputs
- Smart contracts compute on encrypted data
- Programmatic privacy via smart contract functions
- Contract composition and data sharing straight forward
- Private decryption key kept secret by threshold MPC
- Smart contracts expressed in e.g. unmodified Solidity

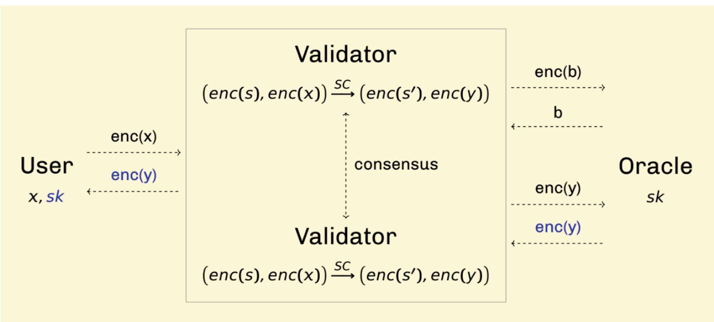
Programmatic Privacy:

Smart contracts decide what users can decrypt using normal functions, with full flexibility to implement custom access control.

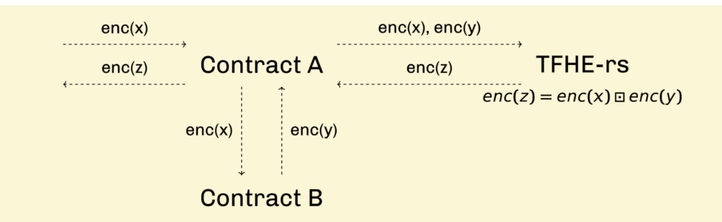
Misuse is prevented since contracts can only approve de-cryption of ciphertexts they obtained honestly:

- Given as input from user along with PoK
- Received from another contract
- Computed using TFHE-rs

Encrypted Blockchain:



Validators:



Why TFHE?

- Exact operations → exact results
- Deterministic operations → support consensus
- Programmable bootstrapping → apply any function
- Fast bootstrapping → unlimited computation

ERC20 Tokens With Encrypted Balance:

```
function _transfer(
    address from,
    address to,
    FHEUInt amount
) internal {
    // Make sure the sender has enough tokens.
    Common.requireCt(FHEOps.lte(amount, balances[from]));

    // Add to the balance of `to` and subract from the balance of `from`.
    balances[to] = FHEOps.add(balances[to], amount);
    balances[from] = FHEOps.sub(balances[from], amount);
}
```

```
function mint(bytes calldata encryptedAmount) public onlyContractOwner {
    FHEUInt amount = Ciphertext.verify(encryptedAmount);
    balances[contractOwner] = amount;
    totalSupply = FHEOps.add(totalSupply, amount);
}
```

Blind Auctions:

```
function bid(bytes calldata encryptedValue) public {
    require(!auctionStopped);
    FHEUInt value = Ciphertext.verify(encryptedValue);
    FHEUInt existingBid = bids[msg.sender];
    if (FHEUInt.unwrap(existingBid) != 0) {
        FHEUInt isHigher = FHEOps.lt(existingBid, value);
        bids[msg.sender] = FHEOps.cmux(isHigher, value, existingBid);
        tokenContract.transferFrom(msg.sender, address(this), FHEOps.mul(isHigher, value));
    } else {
        bids[msg.sender] = value;
        tokenContract.transferFrom(msg.sender, address(this), value);
    }
    FHEUInt currentBid = bids[msg.sender];
    if (FHEUInt.unwrap(highestBid) == 0) {
        highestBid = currentBid;
    } else {
        highestBid = FHEOps.cmux(FHEOps.lt(highestBid, currentBid), currentBid, highestBid);
    }
}
```

Threshold Oracle:

Threshold MPC protocols optimized for TFHE:

- Key generation
- Decryption
- Re-encryption

ZAMA

RESOURCES
zama.ai/blog
github.com/zama-ai/tfhe-rs