# Structuring Computation for Privacy-Preserving Apps


Wei Dai

@_weidai

Bain Capital Crypto

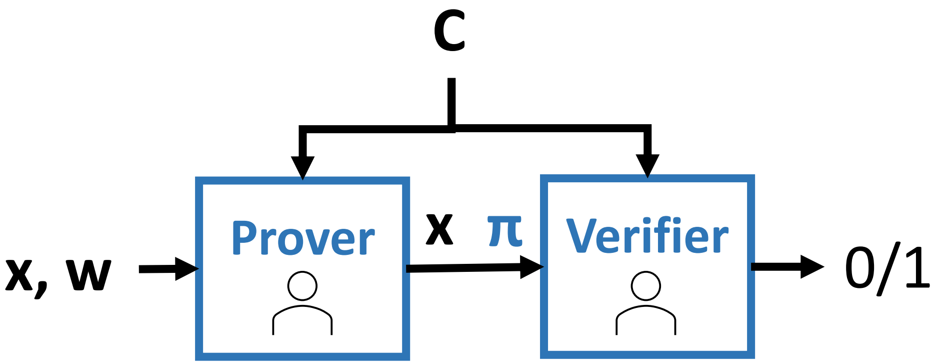

April 21, 2022

Do you have prior knowledge of zero-knowledge proofs?

# Zero-Knowledge Proofs (ZKPs)

**C** - arithmetic circuit, “program execution”
x – public input, w – secret witness

$\pi$ for x ~ “I know w such that C(x, w) = 1”




Typical provers: User wallets, proving services

Typical verifiers: Chains, EVM contracts

**Properties**

- Succinct: $\pi$ is short, verifier runtime is “small”
- Non-interactive: Only one message from P to V
- Transparent: No trusted setup
- Universal: No per-circuit trusted setup

**Security**

- Completeness: It works!
- Zero knowledge: Verifier learns nothing about w
- Knowledge soundness: Prover knows w

**History**

- Studied since the late 1980s
- Recent explosion, due to Z{ero}cash , Groth16, Sonic, Marlin, Plonk, …

Do ZKPs solve all **privacy** problems for blockchain apps?

(Think Uniswap, Aave, NFT auction)

No.

# Agenda of this talk

1. ZK is in contention with **on-chain composability** and **shared states.**

2. ZK for private states, **transparent compute for shared states.**

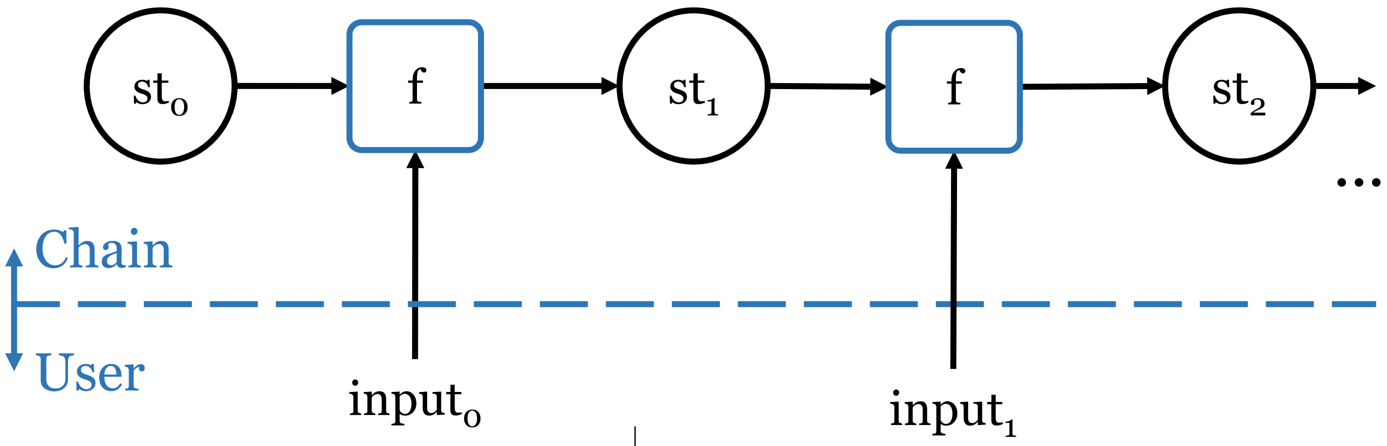
3. **Threshold FHE** for **on-chain confidential compute on shared state.**

4. Framework to program **transparent**, **ZK**, **FHE** computation.

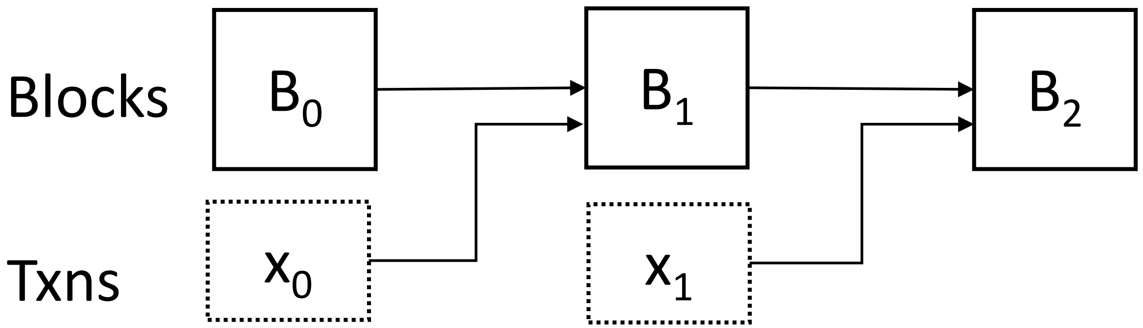
# (Public) State Machines

**State Machines**

Transition function computes $(st_{i+1}, output_i) = f(st_i, input_i)$




**Blockchains**

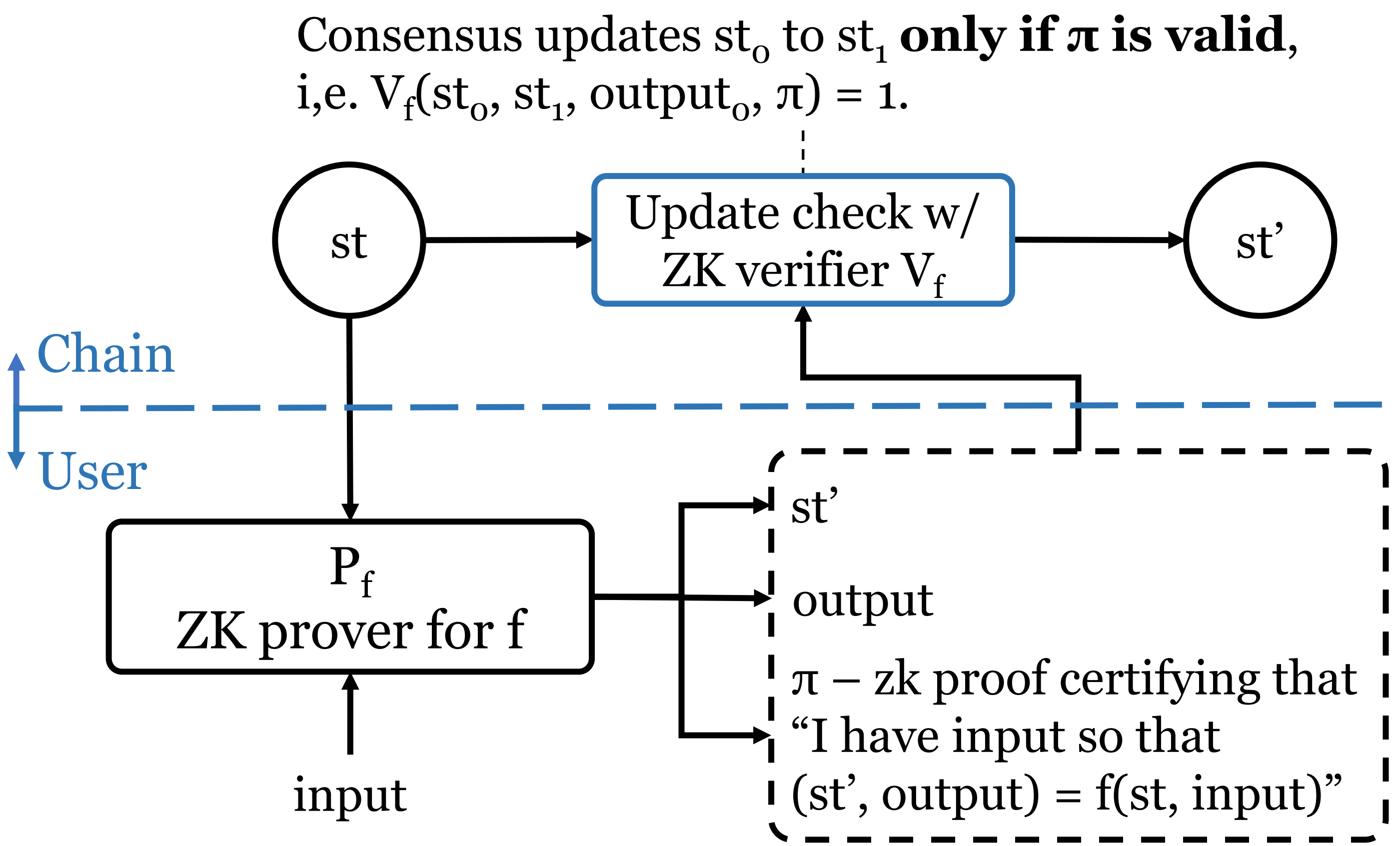



Blockchain State: $st_i = (B_0, ..., B_i)$

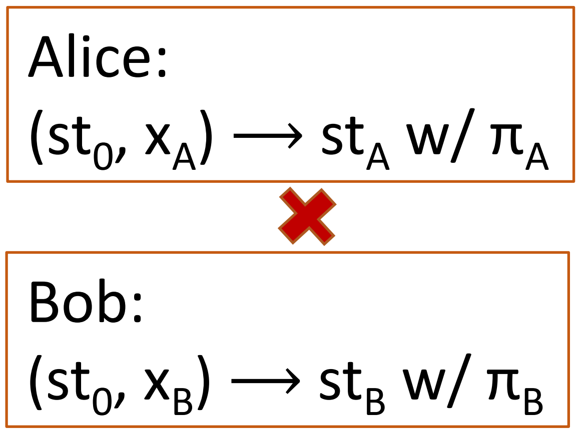
**Smart Contracts**

```
Contract LiquidityPool {
    uint public reserveX, reserveY;
    function swapXtoY( … ) public {
        …
    }
}
```

# ZK State Machines Execution (Zexe / Aleo / Mina Snaps)




Problem: **shared state** give rise to **race conditions**.

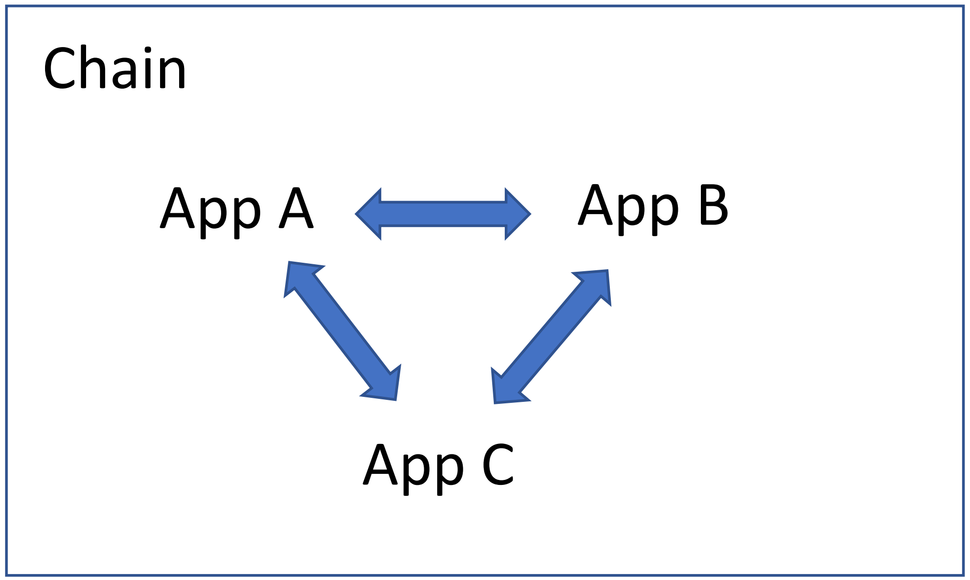



**Only one** state update can be performed.

ZKP smart contracts do not support **shared application state due to race conditions**

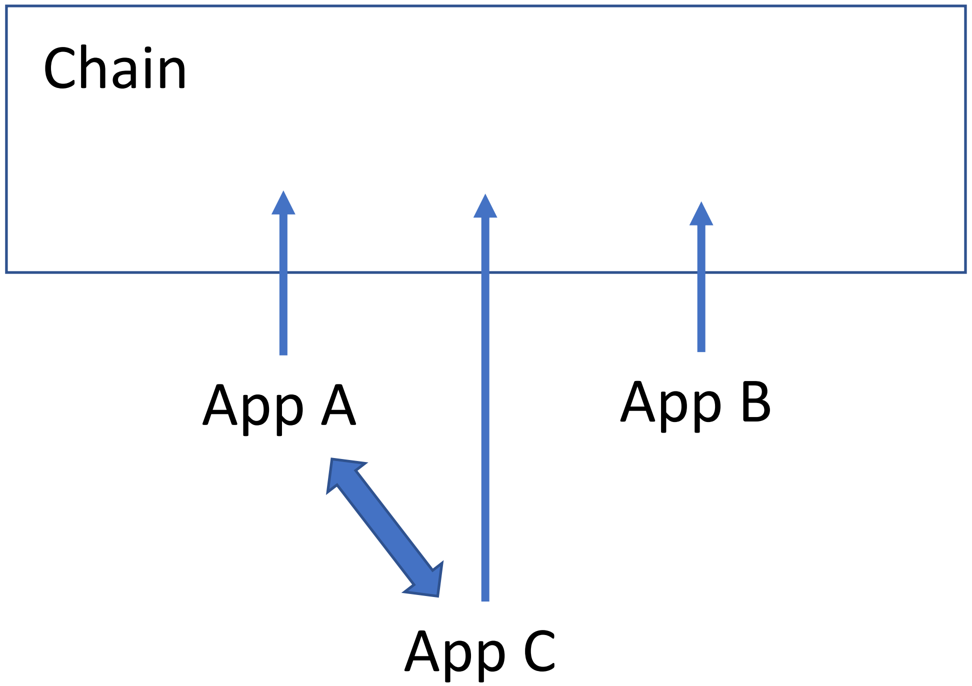
# On-chain vs off-chain apps

**On-chain apps**




**Scalability**
**Privacy**
**Default composability**

**"Full-ZK" Apps**




**Scalability**
**Privacy**
**Opt-in off-chain composability**

# Structuring computation: Transparent vs ZK

```
Contract MyContract:
  public st

  DoStuff(cm, π):
    RangeCheck.verify(…)
```

On-chain

Off-chain

```
ZKCirc RangeCheck(cm; x, r):
  Assert (cm = Commit(x; r)
  Assert (x < k)
```

```
Contract ZCashOrchard:
  public MT
  // Insert-only Merkle tree
  public NS // nullifiers
  Process(tx, π):
    Action.verify(MT.rt, tx, nf; π)
    Assert(nf ∉ NS)
    Ins(tx, MT); Ins(nf, NS)
```

Can be made “composable”:
Aztec Connect, FLAX, …

```
ZKCirc Action(rt, tx, nf; sk, …):
  “tx is valid spend against rt”
  “tx declare correct value change”
  “tx declare correct nf”
```

```
Contract AleoApp:
  public st // record

  Update(st, st’, π):
    Update.verify(…)
```

```
ZKCirc Update(st, st’; x):
  Assert (st’ = f(st, x))
```

ZKP touches no contract state → New state does not invalid old proofs → ZKP re-write contract state

# Agenda of this talk

1. **ZK** is in contention with **on-chain composability** and **shared states.**

2. **ZK** for private states, **transparent compute for** **shared states.**

3. **Threshold FHE** for **on-chain confidential compute on** **shared state.**

4. Framework to program **transparent**, **ZK**, **FHE** computation.

# Third type of computation?

**Replicated on-chain**
**No privacy**
**Shared state**

**ZK off-chain**
**Supports private state and inputs**
**No shared state**

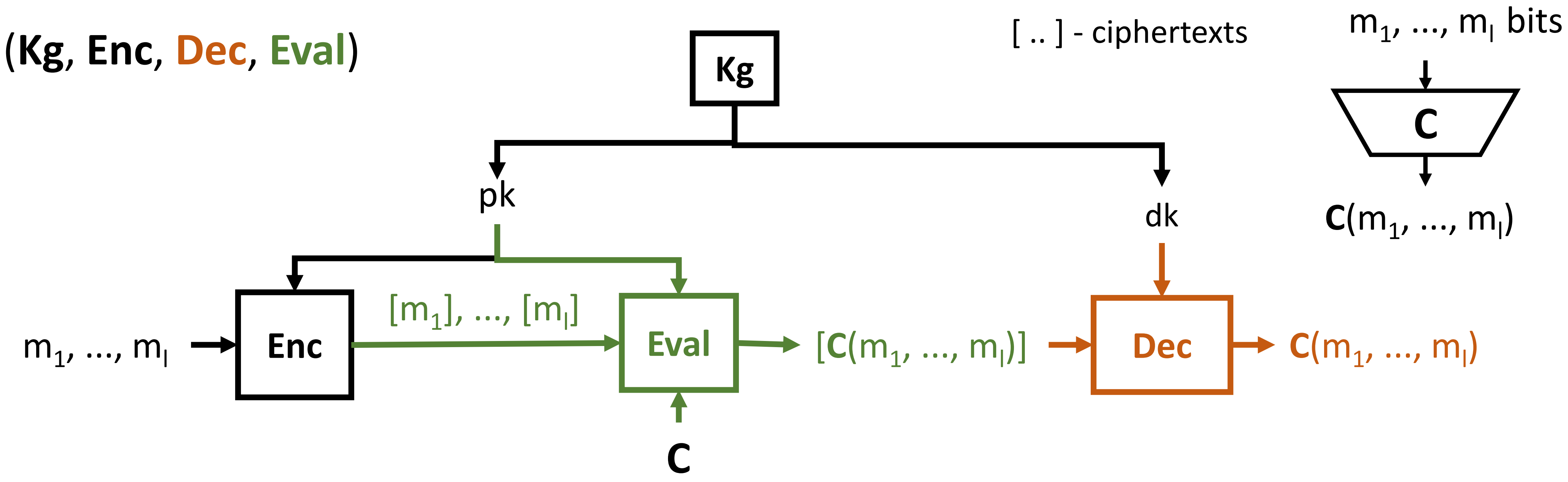
**Private input to confidential shared state?**
***Same trust assumption as consensus?***

**A: YES! w/ Multi-party computation (MPC) or Threshold Fully Homomorphic Encryption (FHE)**

# Fully Homomorphic Encryption

FHE: Computation over encrypted data

Problem: decryption key dk is a master secret!

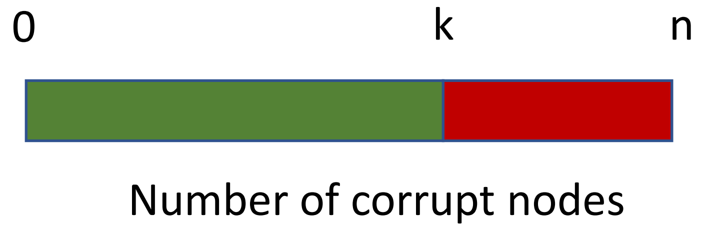



- FHE [Gentry09]: **C** is **any** circuit
  - Active area of R&D in academia and industry. Efficiency improving.
  - Many variants: leveled [GSW, FV, BGV], per-gate bootstrapping [FHEW, TFHE]
  - "Current" state-of-the-art for binary FHE $2^{\sim 12}$ binary gates (xnor, mux) per second on GPU [cuFHE, nuFHE].

# Threshold Cryptography

Liveness holds if k out of n servers cooperate

No security broken even if k – 1 servers collude




Threshold cryptography **particularly applicable to blockchains / BFT protocols** w/ k ~ 2n/3.

Threshold signatures

Dfinity: "Chain key cryptography"

Biconomy, Webb, Lit, …

Threshold encryption / decryption

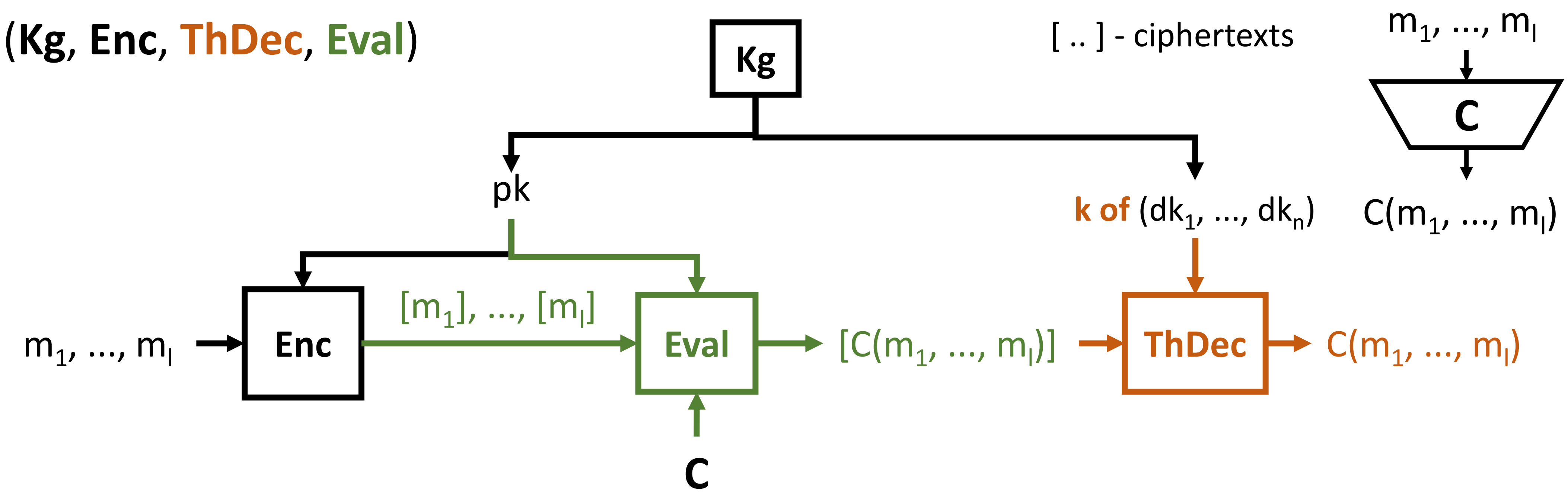
Anoma/Ferveo

Penumbra

We know of protocols to maintain "Shamir threshold secret shares" among a dynamic set of nodes.

- Distributed key generation [DYXMK21, Groth21]
- Dynamic proactive secret-sharing [MZWLZJS19, GKMPS21, Groth21]

# FHE with **Threshold Decryption**




- Achievable with Shamir secret shares
  - Generic lattice-based construction [BGGJKRS17] (ePrint:2017/956), “inefficient”
- Why? **Consensus-based, programmable selective information disclosure**
  - AMM spot price
  - Trade validity

# State Machines with Threshold Decryption


Decrypt part of the encrypted state est that is **explicitly marked for decryption**.

Can be replicated by any BFT-type consensus algorithm.

- Decryption available with a delay
- For privacy and safety, decryption => finalization

Rest of the talk: Assume a BFT-type blockchain system with **fixed FHE public key pk** that can replicate state machine with threshold decryption.

Q1: How to program this state machine?
Q2: Why is this useful?

# Types of Computation

## Transparent On-chain

EVM — Solidity

Wasm — Rust, …

Substrate
ABCI

## ZK Off-chain

Groth16 — Bellman, Circom, ZoKrates

Sonic, Marlin — Arkworks, ZoKrates

STARK / Plonk — Aztec, ZK-Garage, Halo2, Plonky{2}, Jellyfish, Risc0

## Confidential On-Chain

MPC

FHE — SEAL, Palisade, Concrete

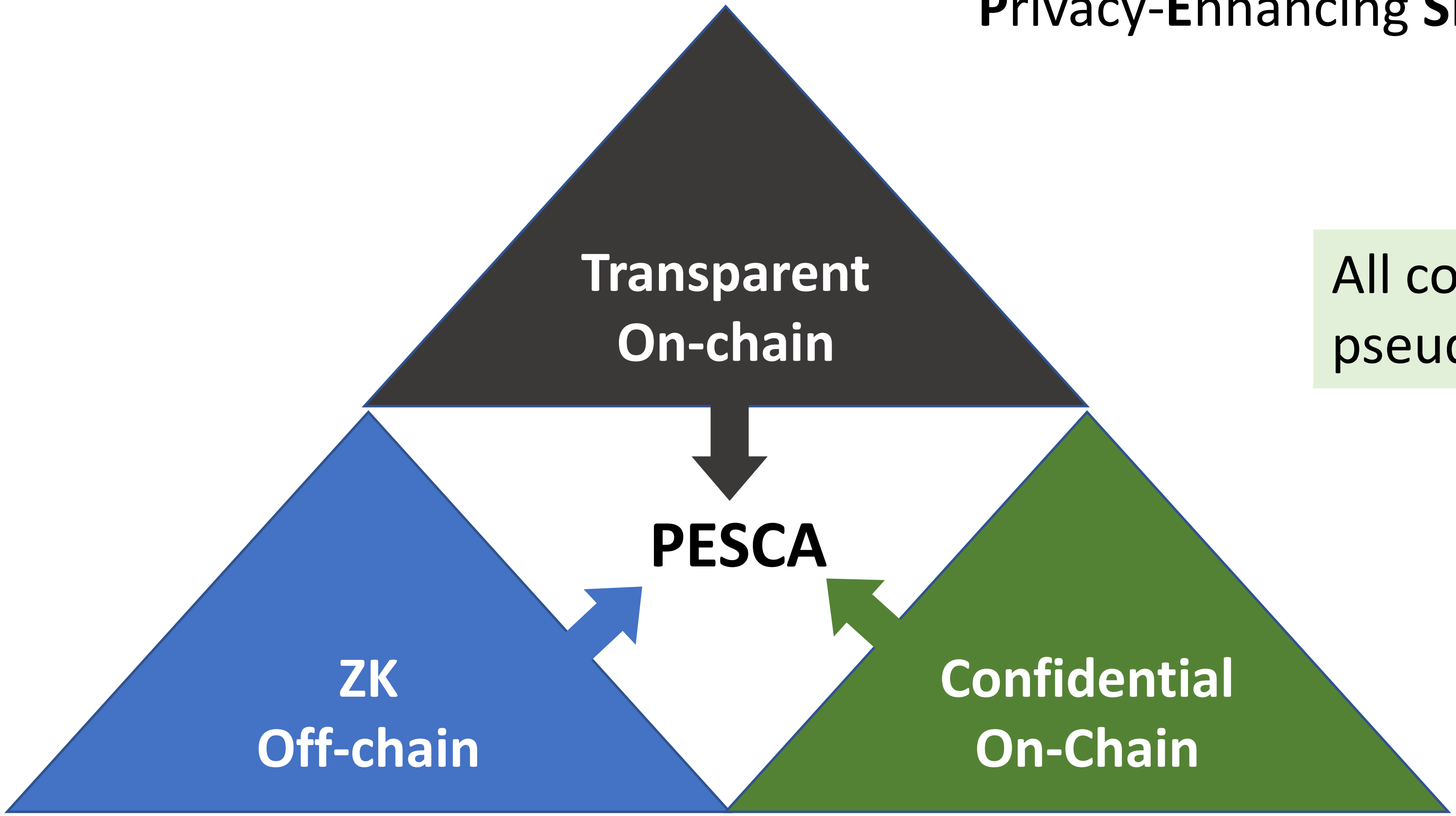
Supporting Shamir keys

FHEW
GSW
…

Implementation?

# Towards a Unified Framework: PESCA

**P**rivacy-**E**nhancing **S**mart-**C**ontract **A**rchitecture




All computation written in pseudocode!

# Expressive Programming Framework

```
Contract ExampeContract:
        Public Func ProcessA(input): // executed on-chain
                ValidateA.verify( input, π )
                state' = ComputeOverA( enc_state, input )
                Async d = ThDec():
                        …

        User Func GenerateA(): // executed off-chain
                input = …
                π = ValidataA.prove(input; …)

        ZK Circuit ValidateA(): // proved off-chain, verified on-chain

        FHE Circuit ComputeOverA(): // executed on-chain
```

# Rest of the Talk: Privacy-preserving CFMM and Auctions

ZCash-like ZK Circuits for token accounting

Confidential inputs

FHE circuits for application logic on confidential states

Merkle tree and nullifier set
Transparent Application logic

Threshold decryption
Information release

# Token with composable private usage

Idea: modify existing ZCash orchard design: value commitment => value encryption.

**Contract** ShieldedToken:

public MT, NS // Merkle tree of notes and nullifier set

**ZK Circuit** Action(tx; …):

v = …

Assert (tx.ev == FHE.$Enc_{pk}$(v, r))

**Private Func** Process (tx, π):

Action.verify( tx; π )

“Add spent notes nullifiers to NS”

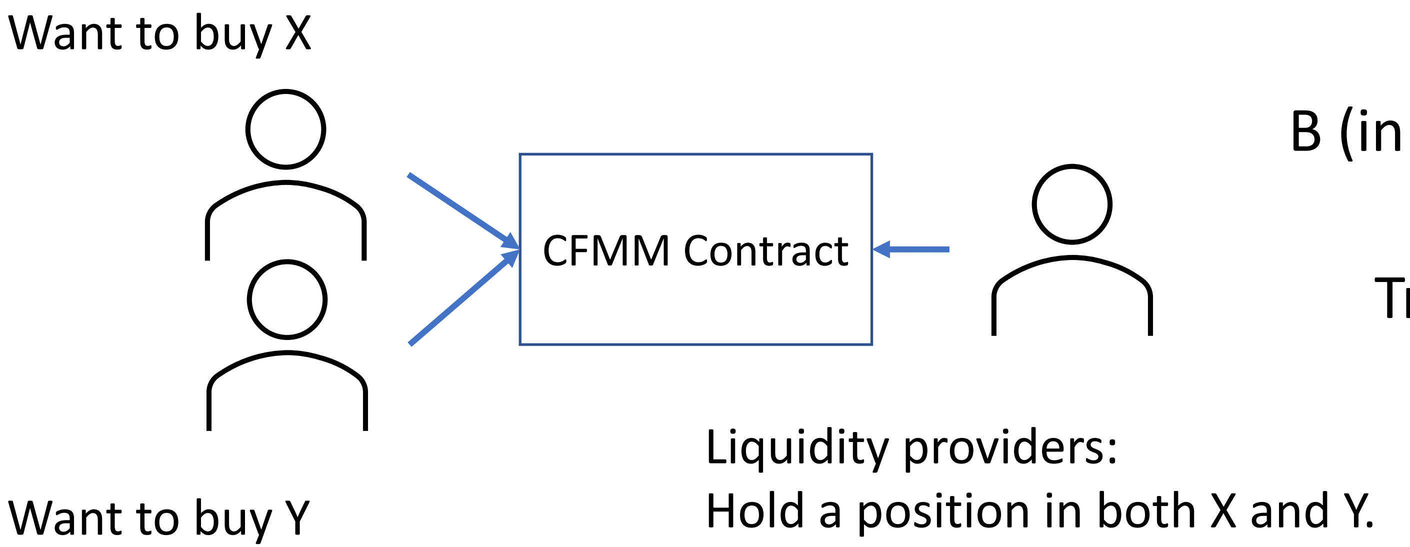
“Add new notes commitment to MT”

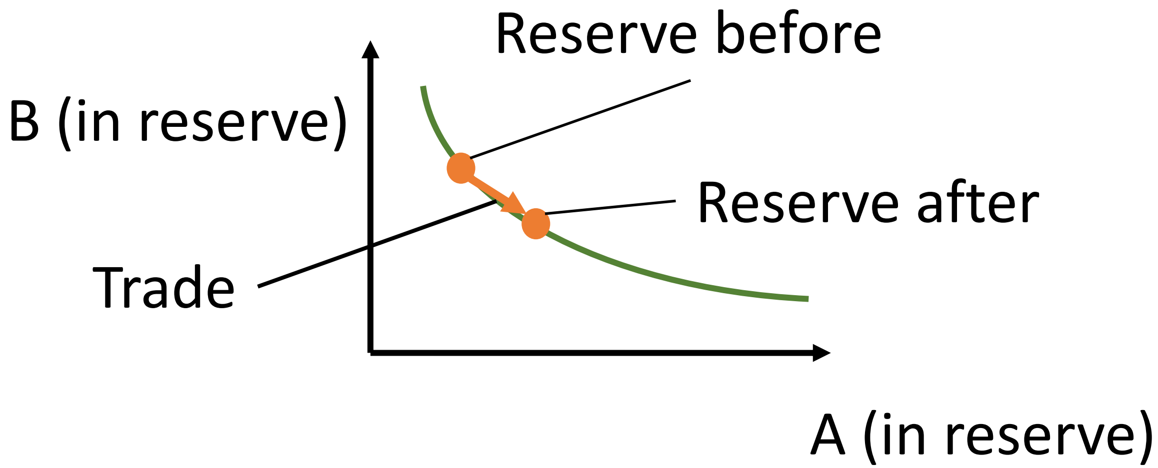
**User Func** GenerateAction():

tx = …

π = ValidataA.prove(tx; …)

# Constant Function Market Makers




Privacy-preserving: trade origins and amounts are not revealed.

Information leakage:

- # of trade requests executed / dropped
- Spot price that is released **programmatically**

# Privacy-preserving CFMM

**Contract** CFMM *extends ShieldedToken*:

private est // FHE encrypted state encrypting reserves (x, y)

**FHE Circuit** Trade( (x, y), (dx, dy) ):

If (x + dx)(y + dy) >= xy then Return ((x + dx, y + dy), 1)

Else Return ((x, y), 0)

**Pub Func** Trade(fund, refund, out):

Process(fund)

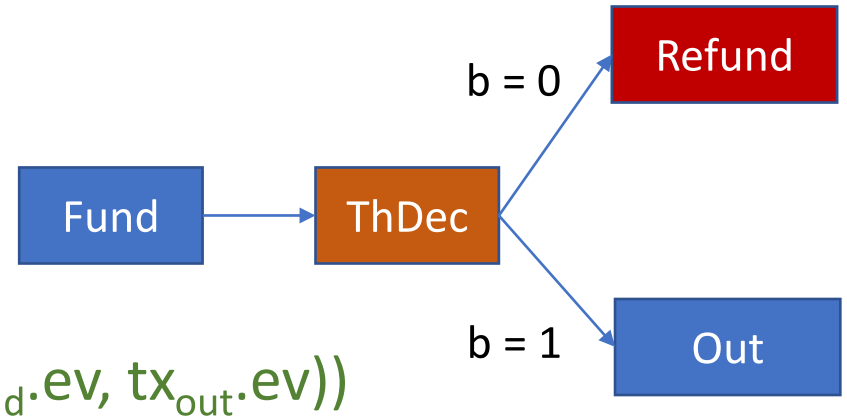
Balance.verify(fund, refund)

(est, eb) ⟵ FHE.Eval(Trade, est, ($tx_{fund}$.ev, $tx_{out}$.ev))

Async b ⟵ ThDec(eb):

If b = 1 then Process(out)

Else Proess(refund)



# Preventing malicious decryptions

**Contract** TargetContract
    public est → 

**Contract** AttackContract
    public est
    **Func** DecryptAll: …

Attack: want to decrypt est, make new contract C and program C to release est.

Mitigation: FHE initial states and all FHE input needs **accompanying ZKPs** particular to each contract.

**Contract** FHEBase:

InitFHEState(est, $\pi$s): // FHE states must be initialized via this method

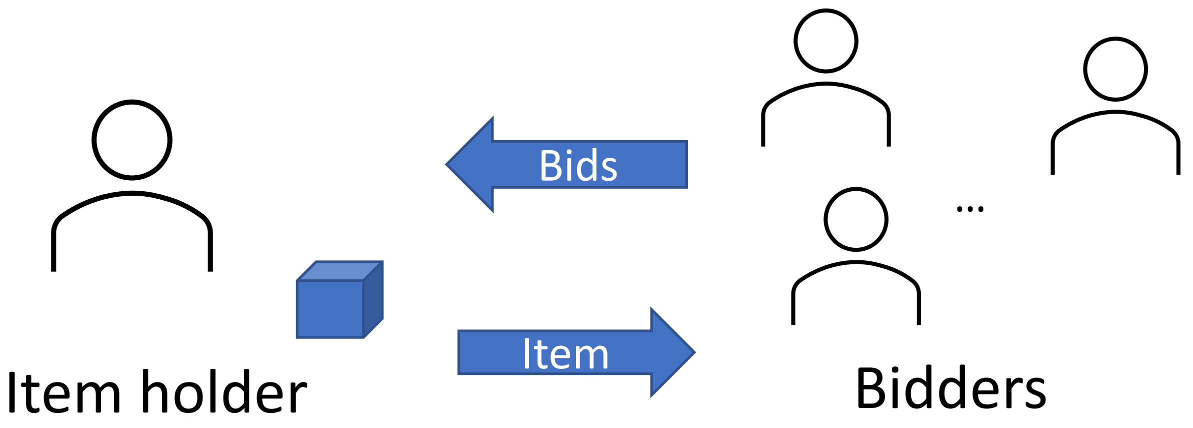
for each (eb, $\pi$) in zip(est, $\pi$s):

InitCheck.verify((this, eb), $\pi$)

ZK Circuit InitCheck(ContractID, eb; b, r):

Assert (eb = FHE.$Enc_{pk}$(b; r))

# Privacy-preserving Sealed-bid Auctions




Sealed-bid: Bids not revealed to other bidders

Privacy-preserving: bids not revealed, to anyone, even after the auction is over.

Information leakage:

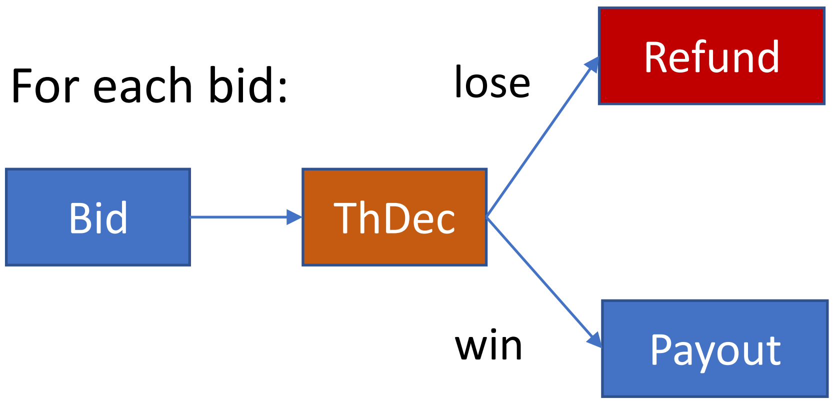
- Item seller learns settling price.
- Auction winner obtains item.
- All other bidders only learn that they did not win.

# Privacy-preserving Sealed-bid Auctions

```
Contract FPSBA extends ShieldedToken:
    private emax, ej // FHE encrypted state encrypting max_bid and winner index

    FHE Circuit Bid[j]( (max_bid, index), bid ):
        If (bid > max_bid) then Return (bid, j)
        Else Return (max_bid, index)

    Pub Func Setup( emax, ej ):
        j = 0; “state initiation checks”
    Pub Func Bid( bid, refund, payout ):
        j += 1; “balance checks”; Process(bid)
        (emax, ej) = FHE.Eval(Bid[j], (emax, ej), bid.ev)
    Pub Func Finalize():
        Async j = ThDec(ej):
            Process(payout_j)
            ∀ i ≠ j: Process(refund_i)
```



# Closing Remarks

- Paper on PESCA to appear.
- We are **hiring**! If you are interested in benchmarking and implementation of ZK, FHE, or threshold cryptography, contact me!