Account Model meets UTXO

a new paradigm for secure dApps & DeFi

Cheng Wang - Alephium

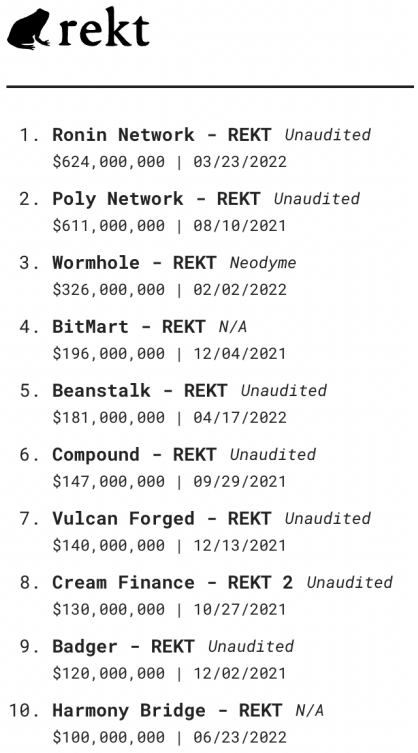
https://github.com/alephium

About Me

- Currently
  - Founder, core R&D of Alephium
  - Alephium is a sharded blockchain combining Account model and UTXO model
- Background
  - Proposed the first linear-time async Byzantine consensus algorithm in 2015
  - Researched number theory and consensus algorithm in universities

Contract Security

- Contract errors are a large part of all hacks
- Solidity auditing is time-consuming and expensive



How To Improve

Toolchains and best practices have come a long way

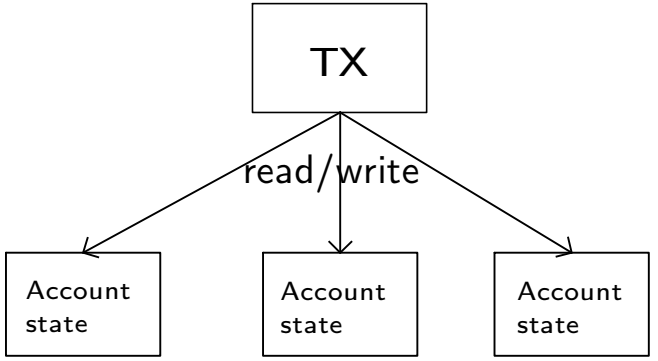
- Steeper learning curve for writing secure contracts
- Hard to scale dApp development

More thorough, introducing new TX model, VM, contract language

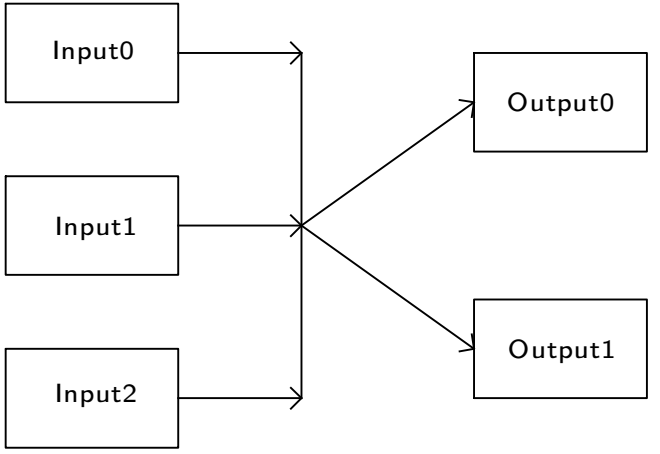
- Bootstrapping takes time, but good for the long run
- leveraging the UTXO model is one of the most promising approaches

Account vs UTXO

Account Model



UTXO model



Account vs UTXO

	Account Model	Classic UTXO model
Pros	• The programming model is dev friendly • Expressive and flexible with direct state access • Smaller TX size	• UTXO directly owned by users • Explicit inputs and outputs, good for verification • Better parallelism, scalability
Cons	• Mutable state creates more room for bugs • Harder to execute in parallel	• Concurrency issue if two TXs consume the same UTXO • Learn a new programming model

Stateful UTXO Model

Account model is convenient for computation, and UTXO model is good at securing digital assets.

Alephium combines the strengths of both models together: stateful UTXO model

- We decompose Account model into assets and contract states
- Tokens are first-class citizens and UTXO-based
- No concurrency issues due to mutable contract states
- Still dev-friendly programming model

Stateful UTXO Model: Execution

A brand new VM with many security features

- Stack-based VM (like JVM/WASM) for safety, readability, performance
- Input/output paradigm greatly mitigates the risk of unsafe external calls
  - Flash loan is not available by design (controversial)
- Built-in asset permission system for contract calls
- Runtime type checking, built-in overflow/underflow checking, etc.
- No low-level support for map data structure, use sub-contracts instead
  - mitigate risks of compromised contract states

MEV on Alephium

- Fine-grained transaction execution due to input/output paradigm
  - sandwich attack, on-chain oracle manipulation, etc. impossible in a single TX
  - arbitrage often requires multiple TXs, more competitions
  - flash loan is not available
- Plus pseudo-random execution of TXs
  - front-running and MEV are harder on Alephium
- We don't solve it. MEV remains an active research topic

Ralph Language

A domain-specific language is introduced

- work seamlessly with our TX model and VM
- hide low-level VM details
- enforce good coding practices by default
- built-in syntax for assets permissions
- simple and dev-friendly

```
// Demo only
Contract DexFun(
  mut xRsv: U256, mut yRsv: U256
) {
  @using(preapprovedAssets=true,
         assetsInContract=true)
  pub fn swapX(buyer: Address, x#: U256) {
    let y# = yRsv-xRsv*yRsv/(xRev+x#)
    transferTokenToSelf!(buyer, x#)
    transferTokenFromSelf!(buyer, y#)
    xRsv = xRsv + x#
    yRsv = yRsv - y#
  }
}
```

Asset Permission System

Dev specifies if a method accepts assets

```
@using(preApprovedAssets=true/false,
       assetsInContract=true/false)
pub fn assetCall(x: Address) { ... }
```

Braces syntax for explicit assets approval

```
contract.call{
    caller0 -> 12 alph, token0:34
    caller1 -> 56 alph, token1:78
}(param0, param1)
```

Manual asset approval for complex use cases

```
approveAlph!(123)
approveToken!(token, 456)
```

No reentrancy for assets call

```
pub fn invalidCall() {
    contract.assetCallA()
    contract.assetCallB()
}
```

Attack Examples

Top solidity issues by Security Boulevard:

- Reentrancy: solved by input&output paradigm
- Unchecked external call: VM terminates on exception
- Overpowered owner: assets permission system, users owned UTXO
- Arithmetic precision: logic errors cannot be solved by VM
- Relying on tx.origin: tx.origin is only available in scripts, not in contracts
- Overflow/underflow: VM checks integer bounds
- Unsafe type inference: the type system is built-in

Security Summary

A new paradigm combining Account model with UTXO model

- Input/output paradigm creates more room for verification, defense
- High default asset security due to UTXO model
- Contract state is mutable, but with security considerations
- Built-in asset permission system
- Built-in good security practices

Side note: our VM is efficient and lightweight too

- Wormhole bridge core contract is $\sim$1KB (vs $\sim$10KB on EVM, $\sim$1MB on Solana)
- Less computation and less IO cost

About Alephium

How it started

- 2017: let's build a sharded blockchain based on BlockFlow
- Before the DeFi boom

How it's going

- Mainnet launched in 2021, sharding in production
- Combined account&UTXO for security and scalability
- Focusing on hardening the protocol and improving the toolchain
- Absorbing new ideas

Let's BUIDL

Website:	https://alephium.org
Github:	https://github.com/alephium
Twitter:	@alephium
Blog:	https://medium.com/@alephium
Discord:	https://discord.gg/JErgRBfRSB
Telegram:	https://t.me/alephiumgroup