Cryptoeconomic Incentivization & the Crosschain Protocol Stack

Peter Robinson May 6, 2022
Peter Robinson
Technical Director & Applied Cryptographer

Peter works in ConsenSys’s R&D group on the bridges and crosschain communications protocols. He has twenty-three granted patents in fields ranging from distributed computing, cryptography, blockchain, and virtualisation to graphical password design. He holds a Bachelor of Computer Engineering and a Masters of Business Administration from Queensland University of Technology, a Masters of Telecommunications Engineering from University of Wollongong, is a graduate of the International Space University, and has submitted his PhD at University of Queensland on the thesis topic, Crosschain Communications. Peter is the co-chair of the Enterprise Ethereum Alliance’s Crosschain Interoperability Working Group.

Peter is the organiser for the Ethereum Engineering Group Meet-up (meet-up, youtube) and is the Founding Program Chair for IEEE ICBC Crosschain Workshop.
Abstract

The GPACT protocol allows arbitrary call execution trees to be executed across blockchains. The protocol sits in the Function Call Layer of the Crosschain Protocol Stack. This talk analyses economic models that could exist across the Crosschain Protocol Stack, and in particular in conjunction with the GPACT protocol.
Crosschain Protocol Stack
## Crosschain Protocol Stack

<table>
<thead>
<tr>
<th>Crosschain Applications</th>
<th>Application code across blockchains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crosschain Function Calls</td>
<td>Enables functions to be executed across blockchains</td>
</tr>
<tr>
<td>Crosschain Messaging</td>
<td>Enables events generated on one blockchain to be trusted on another blockchain</td>
</tr>
</tbody>
</table>
## Crosschain Protocol Stack

<table>
<thead>
<tr>
<th>Crosschain Applications</th>
<th>Application code across blockchains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crosschain Function Calls</td>
<td>Enables functions across blockchains</td>
</tr>
<tr>
<td>Crosschain Messaging</td>
<td>Enables events generated on one blockchain to be trusted on another blockchain</td>
</tr>
</tbody>
</table>

**GPACT**
GPACT Capabilities
Atomicity

- Updates on all chains are either all committed, or are all discarded.
Crosschain Application

Source Blockchain

```solidity
contract ConS {
    func crossSwap(addr to, int val) {
        from = msg.sender
        if (balance[from] < val) {
            revert("Insufficient balance")
        }
        balance[to] = balance[to] + val
        balance[from] = balance[from] - val
        CrossCall(DestBC, ConD, trans, from, to, val)
    }
}
```

Destination Blockchain

```solidity
contract ConD {
    func trans(addr to, from, int val) {
        if (balance[from] < val) {
            revert("Insufficient balance")
        }
        balance[to] = balance[to] + val
        balance[from] = balance[from] - val
    }
}
```
Failure Modes of Non-Atomic Transfers

Source Blockchain state **updated** but Destination Blockchain state **not updated:**

- Some problem on the Destination Blockchain causes a Revert.
- Software failure causes message to never be forwarded / transaction to never be submitted to the Destination Blockchain.
- Relayer has permission problems / configuration problems / not enough Eth to pay for gas that prevents the message transaction from being submitted to the Destination Blockchain.

Imagine:

- Alice wants to swap 2 Eth on Source Blockchain for 2 Wrapped Eth on Destination Blockchain.
- On Source Blockchain: Alice escrows 2 Eth to Bridge Contract.
- On Destination Blockchain: Nothing happens.
- Alice has to complain the Bridge Operator.
- The admin of the Bridge Contract has to submit a transaction to do a *magic* transfer (any token to anyone, any amount) operation to give Alice her 2 Eth back - if this capability exists.
Multichain

Other technologies are between two blockchains only.
Return Values

Other technologies can not return values across blockchains.

Source Blockchain

```solidity
contract ConS {
    func doSome(int aaa) {
        val = CrossCall(DestBC, ConD, trans, aaa)
        now do something with val
    }
}
```

Destination Blockchain

```solidity
contract ConD {
    func trans(int a) returns int {
        do stuff here
        return bb + 5
    }
}
```
Conditional Logic

Source Blockchain

```solidity
contract ConS {
    func doSome(int aaa) {
        purchased = CrossCall(
            DestBC1,ConD, trans, aaa)
        if (!purchased)
            purchased = CrossCall(
                DestBC2,ConF, trans, aaa, bb)
    }
}
```

Destination Blockchain 1

```solidity
contract ConD {
    func trans(int a) returns bool {
    }
}
```

Destination Blockchain 2

```solidity
contract ConF {
    func trans(int a, int b) returns bool {
    }
}
```

Other technologies can not do this type of complex logic.
Application Authentication

Other technologies do have a methodology for limiting which contract from a which remote blockchain called.

Source Blockchain

```solidity
contract ConS {
    func doSome(int aaa) {
        purchased = CrossCall(DestBC, ConD, trans, aaa)
        now do something based on purchased
    }
}
```

Destination Blockchain

```solidity
contract ConD {
    func trans(int a) returns bool {
        require(source == authorised)
        if (purchase is possible)
            execute purchase
            return true
        else
            return false
    }
}
```

This is similar to `require(msg.sender == authorised)`
Developer Experience

- GPACT offers a similar programming model to what developers are accustomed to with Solidity and the EVM.
- Function calls across contracts and across blockchains.
- Composable programming.
- Synchronous execution (from the perspective of application).

Other technologies rely on users learning new paradigms.
How GPACT Works
The General Purpose Atomic Crosschain Transaction (GPACT) protocol is a call tree commitment scheme that provides atomic crosschain function calls across an arbitrarily deep call tree.

Github: https://github.com/ConsenSys/gpact
Trade Wallet Contract

function executeTrade(address _seller, uint256 _quantity)

Logic Contract

function stockShipment(address _seller, address _buyer, uint256 _quantity)

Oracle Contract

function getPrice() view returns (uint256)

Balances Contract

function transfer(address _from, address _to, uint256 _amount)

Stock Contract

function delivery(address _from, address _to, uint256 _quantity)
Trade Wallet Contract

function executeTrade(_seller 0x25d.., _quantity 7)

Logic Contract

function stockShipment(_seller 0x25d.., _buyer 0xa91.., _quantity 7)

Oracle Contract

function getPrice() view returns (uint256)

Balances Contract

function transfer(_from 0x25d.., _to 0xa91.., _amount 56)

Stock Contract

function delivery(_from 0xa91.., _to 0x25d.., _quantity 7)
Sequence

Wallet

Terms

Price Oracle

Finance

Stock

Start
Segment (getPrice)
Segment (transfer)
Segment (delivery)
Segment (shipment)
Root (executeTrade)
Signalling (transfer)
Signalling (delivery)
Execution Process

1. Start, Segment, Root, Signalling
2. Check signatures / proofs
3. Application entry point function call
4. Application function call
5. Indicate storage locked
6. Crosschain function call

- **Blockchain**
  - Business Logic
  - LockableStorage
  - Crosschain Control
- **Crosschain Function Call Layer**
  - Crosschain Application Layer
  - Business Logic
  - LockableStorage
  - Verifier
  - Messaging Layer
Crosschain Messaging Layer

The crosschain protocol stack relies on the Crosschain Messaging Layer:
- Allowing Ethereum Events to be trusted on remote blockchains / sidechains / roll-ups.

https://entethalliance.github.io/crosschain-interoperability/draft_crosschain_techspec_messaging.html
interface CrosschainVerifier {
    function decodeAndVerifyEvent(uint256 _blockchainId, bytes32 _eventSig,
        bytes calldata _encodedInfo, bytes calldata _signatureOrProof)
        external view;
}

Function decodeAndVerifyEvent: Decode and verify event information. Use require to fail the transaction if
any of the information is invalid. For Ethereum Virtual Machine (EVM) based source blockchains, the event
information will be an Ethereum event.

Parameters:

- `_blockchainId`: The blockchain that emitted the event. This could be used to determine which sets of
  signing keys should be used to verify the signature parameter. The `_blockchainId` must be in EIP 3220
  format.

- `_eventSig`: The event function-signature hash. This value is emitted as part of an event. It identifies which
  event was emitted.

- `_encodedInfo`: The abi.encodePacked of the blockchain identifier (_blockchainId), the Crosschain Control
  contract's address, the event function signature (_eventSig), and the event data.

- `_signatureOrProof`: Signatures or proof information that an implementation can use to check that
  _encodedInfo is valid.

The sections below detail the format of this value.
Crosschain Messaging Layer

- Only act on Ethereum Events that were emitted by transactions that are in blocks that are final.
interface CrosschainFunctionCallReturnInterface is CrosschainFunctionCallInterface
{
  /**
   * Call a function on another blockchain that returns a uint256 value. Function call implementations
   * may implement this function. Implementations that do not support this functionality should revert
   * with the message, "NOT SUPPORTED: crossBlockchainCallReturnsUint256".
   *
   * @param _bcId Blockchain identifier of blockchain to be called.
   * @param _contract The address of the contract to be called.
   * @param _functionCallData The function selector and parameter data encoded using ABI encoding rules.
   */
  function crossBlockchainCallReturnsUint256(
      uint256 _bcId,
      address _contract,
      bytes calldata _functionCallData
  ) external returns (uint256);
Function Call with Return Value: Example

```solidity
uint256 currentPrice = gpact.crossBlockchainCallReturnsUint256(
    priceBcId,
    address(priceOracleContract),
    abi.encodeWithSelector(priceOracleContract.getPrice.selector)
);
```
Application

SDK

Blockchain / Rollup A
- Crosschain Application Contracts
- GPACT
- Messaging Verifier
- Attestor / Relayer

Blockchain / Rollup B
- Crosschain Application Contracts
- GPACT
- Messaging Verifier
- Attestor / Relayer
Deployment Options
One App using one GPACT using one set of Relayers

Off Chain

App1

Relayer / Attestor Bc1

Relayer / Attestor Bc2

Relayer / Attestor Bc3, Bc4

On Chain

App1

GPACT

Verifier Bc1

Verifier Bc2

Verifier Bc3 & 4

Relayer / Attestor Bc1, Bc2, Bc3, Bc4

Relayer / Attestor Bc3, Bc4
Each App using a separate GPACT and separate set of Relayers

Off Chain

App1

- Relayer / Attestor Bc1
- Relayer / Attestor Bc2
- Relayer / Attestor Bc3, Bc4

App2

- Relayer / Attestor Bc1
- Relayer / Attestor Bc3, Bc4

On Chain

App1

- GPACT
- Verifier Bc1
- Verifier Bc3&4

App2

- GPACT
- Verifier Bc1
- Verifier Bc3&4

Verifier Bc1

Verifier Bc2

Verifier Bc3&4

Relayer / Attestor Bc1

Relayer / Attestor Bc2

Relayer / Attestor Bc3, Bc4

Relayer / Attestor Bc1

Relayer / Attestor Bc2

Relayer / Attestor Bc3, Bc4
Each App using a separate GPACT with a shared set of Relayers
Apps using a shared GPACT with a shared set of Relayers

Off Chain

- App1
- App2
- App3

- Relayer / Attestor Bc1
- Relayer / Attestor Bc1
- Relayer / Attestor Bc2
- Relayer / Attestor Bc3, Bc4

On Chain

- App1
- App2
- App3

- GPACT
- Verifier Bc1
- Verifier Bc2
- Verifier Bc3&4
Relayer / Attestor / Messaging Layer Incentivization Options
Threshold Signing

- M of N Relayer / Attestor nodes agree that something happened on the source chain.
- Options:
  - Sign: a message (could be an event), or a Merkle root (block header).
  - Signing: Separate signing, cooperative multi-sign, MPC signing.
  - Slashing: Nodes lose stakes bond if the sign invalid messages or Merkle Roots.
  - Mechanism:
    - Relayer transfers signed messages*.
    - Relayer transfers signed Merkle Roots, user relays messages.
    - Attestor provides sign messages, user relays messages.
Operator of GPACT contract on a source chain could pay a monthly fee (off-chain) to have messages or Merkle Roots attested / relayed.

Advantages:
- No gas paid to account for each item relayed.
- Simple.

Disadvantages:
- How do you ensure all N (in the M of N scheme) are active participants in the threshold signing?
Threshold Signing: Incentivization: On-Chain

Verifier contract on each chain contract pays ERC 20 tokens to the Relayers / Attestors that signed the message / Merkle Root.

Advantages:
- Direct attribution of work of each signer.

Disadvantages:
- Costs 20,000 gas per validator + 20,000 for the user / payer.
- In an M of N scheme, only the fastest (most favoured?) validators get paid.
- For an MPC scheme, where there is only one signature, payment must be for “all signers as a group” rather than each signer.
Threshold Signing: Incentivization: Off-Chain / Micropayments

Each Relayers / Attestors for a blockchain that signs (eventually) is paid via a payment channel (countersigned current balances).

Advantages:
- Direct attribution of work of each signer.
- Off-chain: except for set-up and settlement fees, no gas costs.

Disadvantages:
- More complicated.
Threshold Signing: Incentivization: Staking & Bribing

An attacker could bribe M of the N Relayers / Attestors to sign invalid information OR an attacker could obtain M of N Relayers / Attestors private keys

Considerations:

- If signed information can be cryptographically proven to be fraudulent, then only one watcher (who doesn’t need to be a Relayer / Attestor) is needed to prove fraudulent behaviour.
  - However, they can be front run - more on this later.
- If M is less than half N, then maybe M fraudulent signers could be slashed by M other signers than indicate fraud.
Optimistic Approaches

- 1 of N Relayer / Attestor nodes says that something happened on the source chain. Watchers are used to slash in case of back behaviour.
- Options:
  - Slashing: Cryptographic proof of bad behaviour OR M of N votes.
  - Permissioned set of watchers or watchers are permissionless.
  - Proof options: open (MEVable) or commit-reveal (possibly still attackable).
Optimistic Approaches: Incentivization

Considerations:

● How is the 1 of N Attestor that incurs cost to submit the Merkle Root transaction to the source chain selected?
● How are the Relayers that incur cost to submit signed Merkle Root transactions to target chains selected and compensated?
● If the watchers are permissioned, then the problem is back to bribing M of N signers.
● If commit-reveal proof systems are used, then this delays the time between finding a problem and being able to report it (as there is one transaction for the commitment to the proof, and one transaction for the revealing of the proof).
● Attackers could front run fraud proofs, to claim the slashed stake themselves.
Light Client

- Leverage consensus protocol of source chain to prove events occurred.
- Options:
  - Consensus protocol proof implementation: on-chain or off-chain with zero knowledge proof on-chain.
Light Client: Incentivization

Relayers could be paid per month, or per message that is validated.

Advantages:

● Trustless! (assuming you trust the source chain)

Disadvantages:

● Each block header will need to be transferred.
  ○ For target chains that verify information from a source chain intermittently, this could be a large overhead.
GPACT / Function Call Layer Incentivization Options
GPACT Fee Charging: Ether on Start

Methodology:
- GPACT start() function is payable.
- User sends a fee with the start() function call.
- Revert if fee is not correct.

Advantages:
- User pays fee only once.
- User only needs Ether for fee payment on the root blockchain.

Disadvantages:
- No payment is made on blockchains where segments execute.
  - Charging would have to account for the blockchains the transaction will execute on.
  - A process of distributing funds would be required.
- Won’t work on rollups and other chains that don’t have Ether.
GPACT Fee Charging: ERC20 on Start

Methodology:
- GPACT start() function transfers a number of XYZ ERC 20 tokens from msg.sender’s account to the GPACT operator account.
- Revert if fee is not correct.

Advantages:
- User pays fee only once.
- User only needs ERC 20 tokens for fee payment on the root blockchain.

Disadvantages:
- No payment is made on blockchains where segments execute.
  - Charging would have to account for the blockchains the transaction will execute on.
  - A process of distributing funds would be required.
- Costs approx 40,000 gas extra on root chain to pay for the transaction.
GPACT Fee Charging: Ether on Start, Segment, and Root

Methodology:
- GPACT start(), segment(), and root() functions are payable.
- User sends a fee with the function calls.
- Revert if fee is not correct.

Advantages:
- User pays fees each time Relayers / Attestors do work.

Disadvantages:
- User pays fees on all chains.
- User needs Ether for fee payment on all blockchains.
- User might find it difficult to estimate overall fees.
- Won’t work on rollups and other chains that don’t have Ether.
Methodology:
- GPACT start(), segment(), and root() functions transfer a number of XYZ ERC 20 tokens from msg.sender’s account to the GPACT operator account.
- Revert if fee is not correct.

Advantages:
- User pays fees each time Relayers / Attestors do work.

Disadvantages:
- User pays fees on all chains.
- User needs ERC 20 tokens for fee payment on all blockchains.
- User might find it difficult to estimate overall fees.
- Costs approx 40,000 gas extra per chain to pay for the transaction.
Application Level Incentivization
Application Level Incentivization

Request do an ERC 20 transfer / swap
(I can’t pay for gas on Target Chain)
Application Level Incentivization

ERC 20 Approve

Source Chain

Target Chain
Application Level Incentivization

GPACT start()
(commit to call tree)

Source Chain

Target Chain
Application Level Incentivization

GPACT segment()
(provisional: transfer from Bob to Alice)
Application Level Incentivization

GPACT root()
(finalise transfer from Alice to Bob)

Source Chain

Target Chain
GPACT signalling()
(finalise: transfer from Bob to Alice)
Application Level Incentivization

GPACT signalling()
(finalise: transfer from Bob to Alice)
Application Level Incentivization

Ouch!
Application Level Incentivization

Request do an ERC 20 transfer / swap
(I can’t pay for gas on Target Chain)
Application Level Incentivization

ERC 20 Approve

Source Chain

Target Chain
Application Level Incentivization

GPACT start()
(commit to call tree)

Source Chain

Target Chain
Application Level Incentivization

Source Chain

GPACT segment() (provisional: transfer from Alice to Bob)

Target Chain
Application Level Incentivization

GPACT root()
(finalise transfer from Bob to Alice)

Source Chain

Target Chain
Application Level Incentivization

GPACT signalling()
(finalise: transfer from Alice to Bob)

Source Chain

Target Chain
Final Thoughts
Final Thoughts

GPACT contract’s trust relationship with Relayer / Attestors is important.
- For message signing: Relayers / Attestors only sign messages from approved GPACT contracts.
- For Merkle Root signing: Relayer verification contract on target chain only verifies proofs from approved GPACT contracts.

Operating Relayers / Attestors securely is complicated:
- This should be valuable.