Zero Knowledge Proofs

ZKP Applications Overview & zkBridge, Trustless Bridge Made Practical

Instructors: Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, Yupeng Zhang
zkBridge: Trustless Bridge Made Practical
Cross-chain Bridges

- Multi-chain Universe
- Bridge: generic and efficient communication cross blockchains
- Desirable properties
  - Generality (support many applications)
  - Efficient
  - Secure with trust minimization (particularly crucial)
Current Common Bridge Approach: Trust Intermediary

Existing Approach: intermediary
- Side chain (PolyNetwork, Axelar)
- Committee (Wormhole, Ronin)
- Oracles (LayerZero)

Pros: Simple & efficient on-chain verification (e.g., multisig)
Cons: Need to rely on external trust on intermediaries

Trust Assumptions
- 2/3 honest nodes
- 2/3 honest committee
- independence between Oracle and Relayer
## Over $2B Lost in Cross-chain Bridge Attacks in last 18 months

<table>
<thead>
<tr>
<th>Bridge Protocol</th>
<th>Hacked Time</th>
<th>Total Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSC Bridge</td>
<td>2022-10</td>
<td>$568M</td>
</tr>
<tr>
<td>Nomad</td>
<td>2022-08</td>
<td>$200M</td>
</tr>
<tr>
<td>Harmony</td>
<td>2022-06</td>
<td>$100M</td>
</tr>
<tr>
<td>Ronin</td>
<td>2022-03</td>
<td>$600M</td>
</tr>
<tr>
<td>Wormhole (Solana)</td>
<td>2022-02</td>
<td>$325M</td>
</tr>
<tr>
<td>PolyNetwork</td>
<td>2021-08</td>
<td>$600M</td>
</tr>
</tbody>
</table>

Cause: Private Key Leakage
Light client verification:
- Verifying certain correctness properties of state transition in consensus protocol
- E.g., for BFT-based consensus, a light client needs to verify validator signatures and keeps track of validator rotation

Cosmos IBC
- Validators verifies block header information of another chain, performing light client verification
- Cons: require each chain to implement IBC client to perform the verification

NEAR Rainbow bridge
- Implement light client verification as a smart contract in Ethereum
- Cons: on-chain verification is very expensive
zkBridge—Trustless Bridge Made Practical

- With ZKP, we replace **honesty assumptions** with **Cryptographic assurance**

  ![Zero-knowledge proofs](image)

- Efficient on-chain verification using ZKP

---

zkBridge—Trustless Bridge Made Practical

- $\pi$: proving $h_{t+1}$ is correct given $h_t$ (and other info) (consensus-specific light client verification) with SNARKs
- E.g., “$\exists$ sigs by a majority of $C_1$ committee on $h_{t+1}$”

Advantages of zkBridge (zkbridge.org)

- Minimized trust
  - Cryptographic soundness instead of honest assumptions
- Efficient on-chain verification
  - Purpose-built zkSNARK enables efficient on-chain verification
- Permissionless and Decentralized
  - Provers are not trusted so anyone can join
- Extensible and Universal
  - Developers can develop their own application on top
Challenges

- SNARKs are expensive
- Blockchains are not designed to be “ZK friendly”
  - EdDSA digital signature is expensive to express as an arithmetic circuit (~2M gates)
- Each state transition can involve hundreds of sig v
- => Computing $\pi$ naively can be prohibitively expen...
Making zkBridge practical

- deVirgo: a distributed version of Virgo (IEEE S&P 2020)
  - Exploits “data parallelism”
  - Optimal parallelization ---- 100x speedup with 128 machines
  - Practical communication ---- less than 20% of proving time
- Reducing proof size by recursion
  - run deVirgo verifier in Groth16
- Batching

deVirgo: fast proof generation, relatively big proof
Groth16: slower proof generation, constant proof & verification.

Constant size proofs & verification with only a slight increase in prover time
Approach: deVirgo & 2-layer Proof Composition

Layer 1: Distributed Proof Generation on Distributed Machines

Large-scale Arithmetic Circuit C

deVirgo Prover (distributed machines)

proof $\pi_1$

devirgo Verifier

translate the verification process into a circuit C2 (much smaller than C1)

Groth16 Prover

proof $\pi_2$

Groth16 Verifier (on-chain smart contract)

Final Verification Result: True/False

Layer 2: Proof Composition

Efficient On-chain Proof Verification
Performance of zkBridge proofs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>w/o RV</td>
<td>w/ RV</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>w/o RV</td>
<td>w/ RV</td>
</tr>
<tr>
<td>8</td>
<td>12.52</td>
<td>7.34</td>
<td>1946476</td>
<td>78M</td>
</tr>
<tr>
<td>32</td>
<td>12.80</td>
<td>32.24</td>
<td>1952492</td>
<td>78M</td>
</tr>
<tr>
<td>128</td>
<td>13.28</td>
<td>131.89</td>
<td>1958508</td>
<td>79M</td>
</tr>
</tbody>
</table>

Table 2: Evaluation results. RV is the shorthand for recursive verification.

More results in paper: [https://zkbridge.org](https://zkbridge.org).
Extensibility of zkBridge

**Application Layer**
(user-specified cross-chain applications)

**Application Contracts**
(can be both embedded on C1 or C2)

**Base Layer**
(for block header synchronization)

Sender chain $C_1$

Block Header Relay Nodes

Generate proofs for block headers & relay the headers with proofs

Receiver chain $C_2$

The updater contract exposes an API for applications to learn the latest state of the other blockchain.

**Updater Contract**
(deployed on C2)
zkBridge has great extensibility

Developers can build application contracts to achieve more advanced functionalities such as:

1. Message Passing
2. Cross-chain Assets Transfer/Swap
3. cross-chain NFT Interoperations
The application deploys smart contracts using zkBridge and interact with them based on users’ requests.
Application Layer Use Case 1: Message Passing

The user wants to use the application for passing message from sender chain $C_1$ to receiver chain $C_2$.

The send contract is deployed on $C_1$. Both the updater contract and the verify contract are deployed on $C_2$.

$S\text{C}_\text{send}$ and $S\text{C}_\text{verify}$ provides a message-passing service that other applications can use.
Application Layer Use Case 1: Message Passing

1. *Send message m*

2. *Write message m on sender chain C₁*

3. *Relay header with proofs*

4. *Verify and accept*

5. *Call the verify contract with message m and its Merkle proof*

6. *Retrieve block header DAG from updater contract and verify message m.*

Upon verification, message m is made available to relevant applications on C₂.

Verification details:

1. Identify the block containing m and verify the block is on the main chain of DAG;
2. Verify submitted Merkle proof of message m in the corresponding block.

Step 3 and 4 are done in base layer.
Defense-in-Depth

- Base layer of zkBridge presents a unified interface for syncing block header from another chain
- Improving security with defense-in-depth
  - Combining multiple implementations: proof-diversity, n-version programming, combining with other approaches such as optimistic solutions
  - Design different policies for combining different implementations
    - E.g., Hashi (https://github.com/gnosis/hashi): an EVM Header Oracle Aggregator
zkBridge: trustless bridge made practical

- Minimized trust
- Efficient on-chain verification
- Efficient proof generation
- Permissionless & decentralized by design
- Extensible and universal

To learn more: [https://zkbridge.org](https://zkbridge.org), [https://rdi.berkeley.edu/research](https://rdi.berkeley.edu/research)

zkBridge Technology Enables Other Capabilities

- **State proof**
  - A cryptographic proof of state changes that occur in a given set of blocks (e.g., Algorand State Proof)

- **zk-based light client verification**
  - Support efficient light client verification, including mobile use case (e.g., Celo Plumo)

- **zkBridge can be extended to privacy chains with privacy protection**
zkBridge Track in ZKP/Web3 Hackathon

ZKP / Web3 Hackathon

March 1 - May 2, 2023

Hosted By Berkeley Center for Responsible, Decentralized Intelligence

zk-hacking.org

Virtual & UC Berkeley