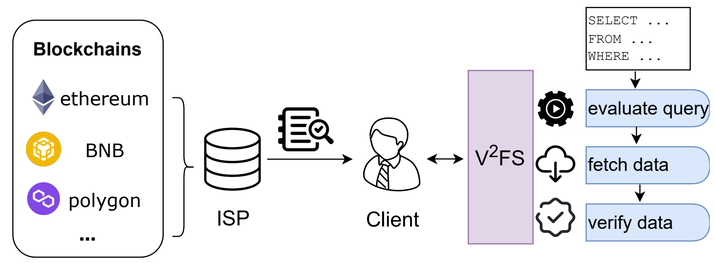
V²FS: A Verifiable Virtual Filesystem for Multi-Chain Query Authentication

Haixin Wang[1], Cheng Xu[1], Xiaojie Chen[2], Ce Zhang[1],
Haibo Hu[3], Shikun Tian[2], Ying Yan[2], Jianliang Xu[1]
[1] Hong Kong Baptist University [2] Ant Group [3] Hong Kong Polytechnic University [1]

{hxwang, chengxu, cezhang, xujl}@comp.hkbu.edu.hk
{tianxun.cxj, shikun.tiansk, fuying.yy}@antgroup.com haibo.hu@polyu.edu.hk,

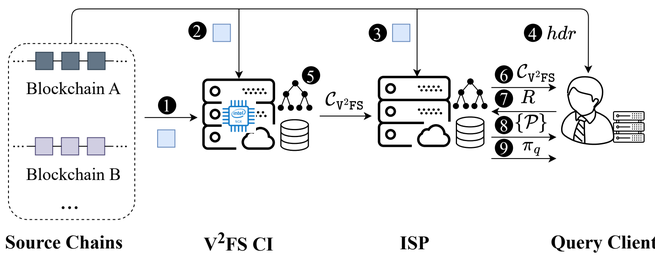
Background

- **Background**: Increasing demand to query multi-chain data
- **Requirements**:
  - Compatible with existing blockchains
  - Support various query types
  - Ensure query integrity
- **Existing Solution**: The Graph
  - A decentralized protocol for blockchain data indexing
  - Rely on a set of *indexers* to index blockchain data
  - Use a *dispute mechanism* to incentive parties behave honestly
  - Issue: Weak integrity guarantee and long dispute period
- **Strong Integrity Guarantee**: verifiable computation (VC)
  - Use authenticated data structure (ADS)
    - Support specific query type with efficiency
  - Use general VC schemes
    - Support general query types
    - Suffer from high computation overhead
    - Challenging to implement with various database engines
- **Our Solution**: verifiable virtual filesystem (V²FS)
  - Verifying computation → verifying data
  - Move computation to a trusted zone (Client)
  - Client evaluates query locally using an off-the-shelf query engine
    - Fetch data on-demand from a remote Indexing Service Provider (ISP)
    - Verify data integrity by cryptographic proof



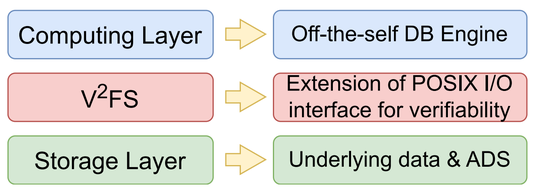
System Overview

- **Design Goals**:
  - *Blockchain compatibility*: compatible with existing blockchains
  - *Database compatibility*: accommodate diverse database engines
  - *Strong integrity guarantee*: provide efficient query verification
- **System Model**

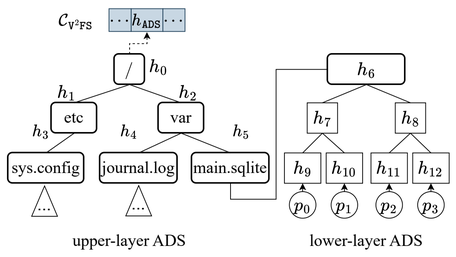


Verifiable Virtual Filesystem (V²FS)

- A middleware between the computing layer and storage layer
- Enable the utilization of off-the-shelf database engines for verifiable queries
  - Offer widely-used POSIX operations essential for any filesystems to work
  - `open`, `seek`, `read`, `write`, `close`
  - These operations are extended with integrity features

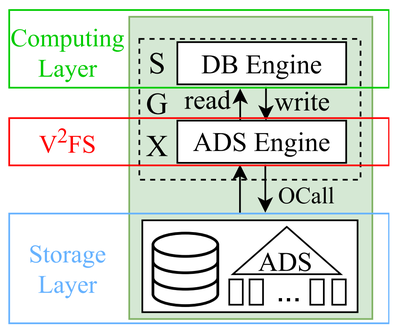


- **V²FS ADS**
  - Database is stored as *files*, which are organized in *pages* with a fixed size (e.g., 4096 bytes)
  - A variant of MHT consists of two layers to authenticate page retrieval
    - lower-layer: a complete binary tree built on pages in a single file
    - upper-layer: a trie structure indexing the path for each file



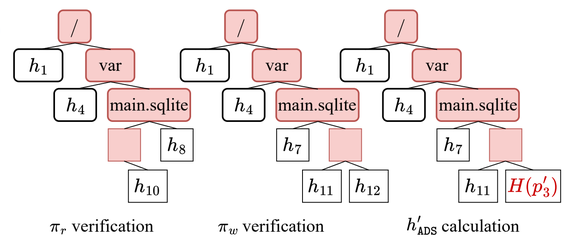
V²FS Maintenance

- V²FS CI: updates storage layer & generates trusted new ADS root
- Employ SGX for secure update
- SGX consists of a DB Engine and ADS Engine
- Storage layer is located outside of SGX
- OCalls are invoked when SGX reads/writes from/to outside-SGX storage
- Problem: Outside-SGX storage is distrusted
- Solution: Verifiable computation
  - Verify all pages to be read
  - Verify the neighboring nodes related to pages to be written



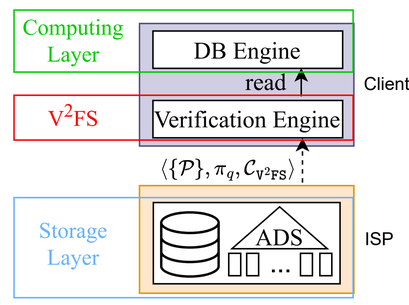
V²FS Maintenance–Example

- Update by a new block involves reading p_0 and writing p_3 to p_3'
- Verify $\pi_r \rightarrow p_0$ is correct.
- Verify $\pi_w \rightarrow$ all nodes in π_w are correct.
- Calculate new ADS root using π_w and p_3'



Query Processing

- **Query Workflow**
  - Client fetches the certificate generated by V²FS CI
  - Evaluates the query locally and requests data pages on-demand
  - ISP returns acquired data pages with a proof
  - Client computes the query results & verifies data integrity using the proof



- **Optimizations**
  - Intra-query Cache: Cache frequently visited pages within a single query
  - Inter-query Cache: Cache frequently visited pages among queries
  - Bloom Filter Integrated Validation: Utilize the page update information to validate pages in inter-query cache

Performance Evaluation