Verifiable Computation with Multiple Clients
Many current results on verifying outsourced computation

- **This Eurocrypt**
  - Streaming Authenticated Data Structures
  - Quadratic Span Programs and Succinct NIZKs without PCPs
  - Dynamic Proofs of Retrievability via Oblivious RAM

- **Recent**
  - Pinocchio: Nearly practical verifiable computation (IEEE S&P '13)
  - A Hybrid Architecture for Interactive Verifiable Computation (IEEE S&P '13)
  - Resolving the conflict between generality and plausibility in verified computation (Eurosys '13)
  - Taking proof-based verified computation a few steps closer to practicality (Usenix Sec '12)
  - Non-interactive verifiable computing: Outsourcing computation to untrusted workers (CRYPTO '10)
Typical model for verifying outsourced computation

- Server S
  - Normally correct
  - Sometimes faulty (untrusted, potentially malicious... Byzantine)

- One client C
System model

- **Server S**
  - Normally correct
  - Sometimes faulty (untrusted, potentially malicious ... Byzantine)

- **Many clients: \( C_1 \ldots C_n \)**
  - Correct, may crash
  - Invoke operations on server
  - Disconnected
  - Small trusted memory

- Asynchronous

- No client-client communication
Integrity violation from replay attack

write(1, x)
write(1, u)
write(1, t)
write(2, v)
read(1) → x
write(2, w)
read(1) → u
read(2) → w
From replay attacks to fork-linearizability

- In replay attack, server may present different views to clients
  - Cannot be detected by clients
  - Server “forks” the views of different clients

- Run a protocol to impose fork linearizability [MS02]
  - Ensures that if server forks the views of two clients once, then
    → their views are forked ever after
    → they never again see each others updates

- Every consistency or integrity violation results in a fork
  - Best achievable guarantee for interaction with untrusted server

- Forks can be detected on a cheap external channel with low security
  - Synchronized clocks
  - Periodic operations
  - Gossip
Fork-linearizability graphically

write(1,x) write(1,u) write(1,t)

write(2,v) read(1)→x write(2,w)

read(1)→u read(2)→w

write(1,x) write(2,v) write(2,w)

r(1)→x w(2,w)

w(1,u) r(1)→u w(1,t)

View of C₁
View of C₂
View of C₃
Fork-linearizable storage and services

- **Goal**
  - If server is correct, then clients see linearizable operations
  - In any case (= even when server corrupted and violates spec), the clients respect **fork-linearizability**

- **SUNDR** [Mazieres, Shasha, '02]
  - Secure untrusted data repository (storage system)

- **FAUST - Fail-aware untrusted storage** [CKS11]
  - Never blocks, uses sporadic client-to-client messages

- **Blind Stone Tablet** [Williams, Sion, Shasha, '09]
  - Never blocks, but may abort operations (databases)

- **Untrusted Services** [C11]
  - Generic protocol using ideas from authenticated data types
  - Blocking

- **Non-blocking Commutative-Operation Verification** [CO13]
  - Generic services, but operations verified by re-execution
  - Non-blocking for commuting operations
Conclusion

- **Existing work**
  - Storage-integrity verification protocols, simple functionality
  - Integrity and consistency verification protocols, but without efficient cryptographic verification
  - Cryptographic verification protocols, only for single-client model

- **Challenge**
  
  Build cryptographic tools for integrity and consistency verification
  - Stateful remote services
  - Preserve "forking" consistency notions
  - Non-blocking client operations
Literature

