Smart contracts are trusted only for correctness and availability but not for privacy. In Ethereum each entry point is a function. Three main pitfalls while development SC:

1. Errors in encoding state machines (problem with interpretation of input values, solution: validation of all fields, refund money in bad cases),
2. Failing to use cryptography (if you send your choice before another, he can find it in the blockchain and win you every time, solution: hash generation, commit and reveal phase. Two core properties here: binding – no changes and hiding – does not show until reveal),
3. Misaligned incentives (does not show that you have lost, solution: reveal deadline)

Ethereum specific errors:

1. Call-Stack bug: call stack limit is 1024
2. Blockhash bug: only 256 blocks
3. Incentives Bug in Ethereum mining model: exploit Ethereum design

Power-law distributions were observed in both transaction count and Ether balances across contracts.
Two types of accounts: private and contract.

Emputical Analysis:
1) Contract Name Usage. From 10174 only 6205 unique. This shows limited use of semantic naming and weak architectural practices

2) Compiler Version Usage. Solidity using pragma. Adoption of new compiler is fast.

3) Balances: 80% of balance was held by 10 contract

4) Transactions according power law distribution, BUT no strong correlation between balance and txs.

Most contracts 100-200lines code

Most fies 1 contract declaration

Bytecode 6k-7k

TABLE 4. Matrix of the cross-correlation coefficients between metrics and indicators computed among 10174 contracts.

	Line Total	Bytecode	M1: LoC	M2: CpL	M4: NDF	Tx Count
Line Total	1	0,52	0,95	0,22	0,91	0,02
Bytecode	0,52	1	0,56	0,03	0,52	0,02
LoC	0,95	0,56	1	0,02	0,91	0,02
CpL	0,22	0,03	0,02	1	0,12	-0,01
NDF	0,91	0,52	0,91	0,12	1	0,02
Tx Count	0,02	0,02	0,02	-0,01	0,02	1

Size and complex does not mean how it will be used

Five types of smart contracts: wallets, financial, games, libraries and notaries.

1. EtherDelta – exchange contract. Bitcoinereum – ERC-20 token mimicking Bitcoin, Cryptokitties – collectible game. 6.2% of transactions

2. Replay Safe Split. Contracts to prevent replay attacks ensure safe separation
3. Reistrar – register your smart contract. DSTOKEN, EOS
4. Contoller- Bitrix wallet creation, TRX
5. Golem Network Token- decentralized computing power

Balance of contract is determined by its goal. Key indicators for users: tx number, tx per day, for devs: number of deployments, versionds, code reuse. Golden standard: just one in use. The analysis shows that many smart contracts undergo significant evolution through re-deployment and versioning. In parallel, popular templates are often reused, especially in token projects.

Metrics of top 20 contracts:
Declared contracts – 5.9 (4.39) use inherutance and modular design.

Declared functions -26.7 (5.3) – more functionality per contract.

Max complexity < 4 max 7 (due to functionality)

Key results of second study:

1. Evolution and follow updates
2. Different purposes
3. Code reuse is common
4. Contract name no contract purpose
5. Transaction and Balance follow power law (security relevance)

Why Many Smart Contract Use Cases Are Simply Impossible

Expectations are overhyped

Smart contracts are not magical ai

Not everyone understands the limits of decentralized code computation => wasted time and resources for development

Three main misconceptions:
1. Contacting with external services. If we try to pull data from external resources, nodes can get different values, and consensus will not be confirmed. Infeasible usecases: fetch external data, trigger external event. Workaround – Trusted Intermediate (Oracle). It pushes data in blockchain through transaction and external service can monitore blockchain and trigger external services. **Trusted parties are always required working blockchain and real-world data.**

2. Using contract as an automate onchain payments requires assets to store in contract addresses. Pay rate. The smart contract cannot at the same time pay the rate and give money to use for building. In order for the agreement to be reached, he must already have the money in the account in the building. That is, you, as the issuer, must immediately put money up.

3. No privacy

Smart contract add flexibility but are slower and more complex than transactions. Most of dApps do not need them. Smart contract only control on-chain behavior and cant communicate with outside

Ekiden: A Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts

Ekiden is designed to overcome major limitations – lack of confidentiality and poor performance by combining blockchains and TEE:

1. Performance compared to ethereum 600x more throughput, 400x lower latency and 1000x lower cost
2. Hybrid Architecture: Compute nodes and consensus nodes

TEE provides proof that the program runs correctly, that proof is signed. Problem is availability is not guaranteed, state persistence is not reliable.

Technical challenges:

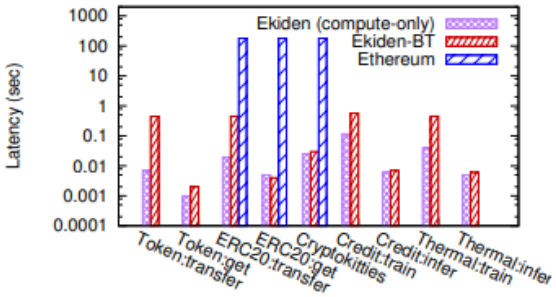
1. TEE Failures: Availability Failures, Side-channel attacks (Tee can leak sensetive data), Timer failures
2. Proof of Publication for PoW blockchains: verify the block inclusion without trusted time
3. TEE state requires storing keys, key replication gives tradeoff security vs availabilit
4. Atomic delivery of execution results

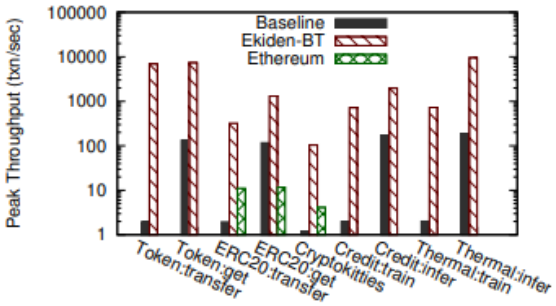
After computation -> output goes to client, new state and proof are accepted- If either fails, nothing is delivered.
Proof of Publication (PoP). Ensure that message(contract output or state update) is published on the blockchain. Honest provers can be rejected if block is delayed.

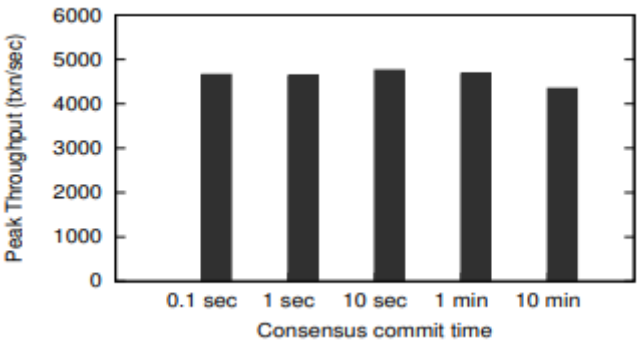
Key managment - Distributed Key Generation (DKG), Distributed Pseudorandom Function (PRF)

Atomic delivery. TEE encypts output fpr client with fresh key and after client confirm TEE sends output for blockchain.

PROTEkiden







Ekiden's performance remains high even if blockchain confirmation takes up to 10 minutes. State caching and batching allow the system to continue working without waiting for each commit to complete.