Blast

Blockchain-Assisted Key Transparency for Device Authentication

Alessandro Gattolin, Cristina Rottondi, Giacomo Verticale

Outline

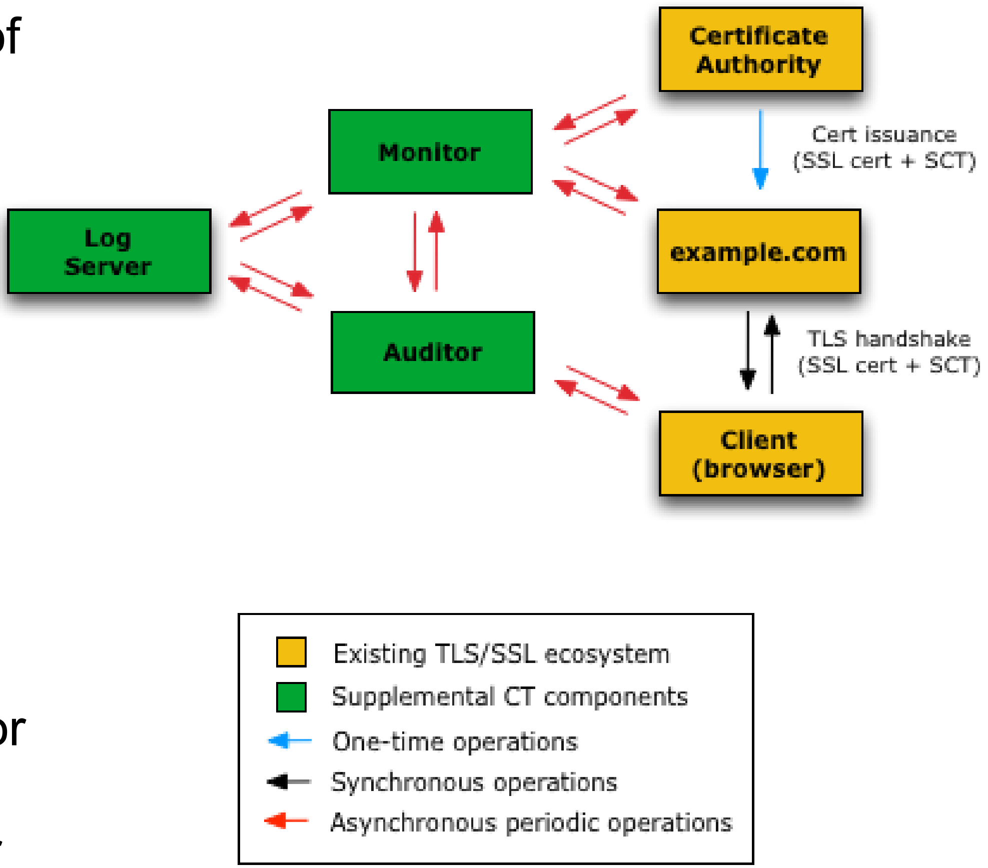
- Key Transparency
- Architecture of BlAsT
- Implementation on the Bitcoin chain
- Implementation on the Ethereum chain
- Techno-Economic Evaluation

Certificate Transparency (RFC 6962, 2013)

Problem with SSL certificates: lack of auditing

Certificate Transparency introduces

- public, append-only, logs
- public monitoring of certificates in the wild
- public auditing of certificate logs



Goal:

- impossible to issue a certificate for a domain without the certificate being visible to the domain owner

CONIKS Key Transparency (Melara et al, 2014)

Generalization of SSL certificate logs to dictionaries of user keys (email addresses, user authentication, etc.)

Main issue: certificate logs can be exploited as directories of URLs / domains

- unorthodox, but not a privacy concern

Solution: use Verifiable Random Functions (VRF)

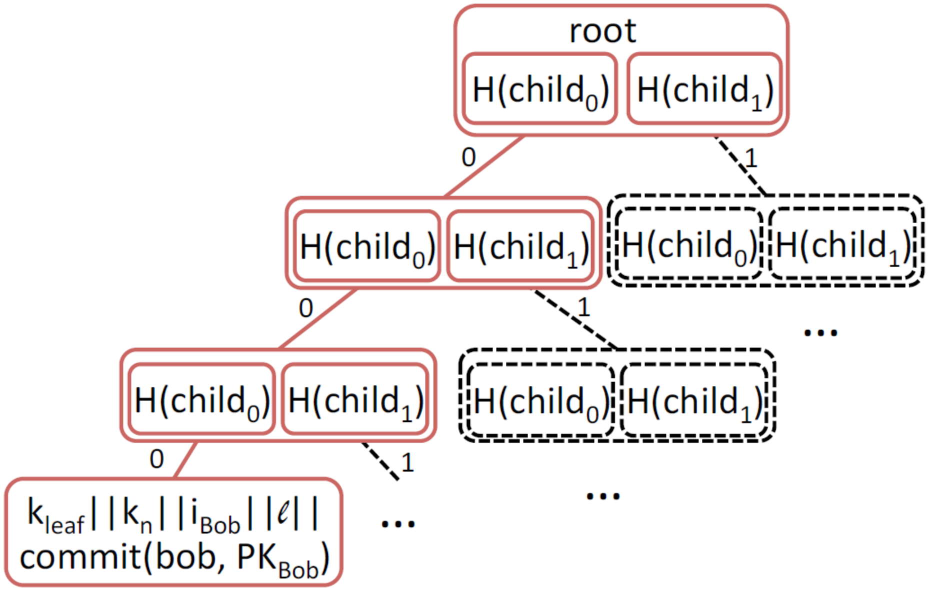
- monitors can query the identity provider for presence (or absence) of key, cannot get list of keys

CONIKS proofs

Positive and Negative proofs using Merkle Trees.

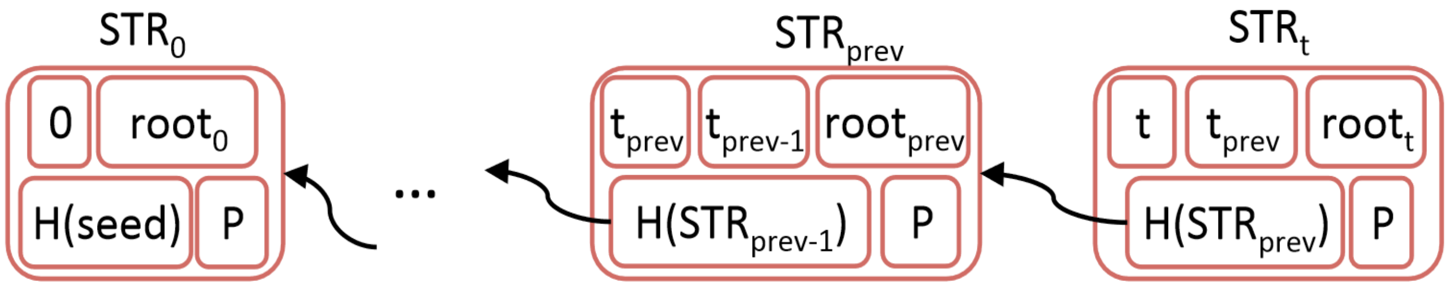
To prevent equivocation, the identity provider must sign the root of the tree

(STR = signed tree root)

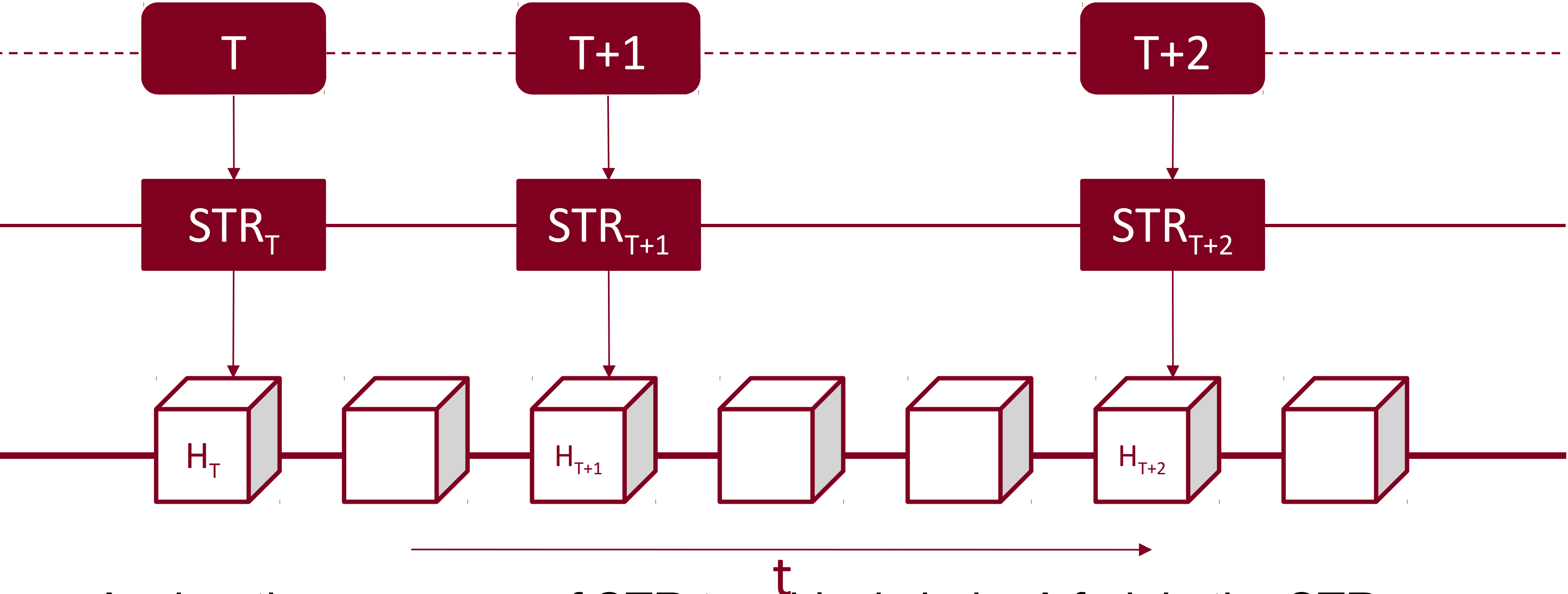


CONIKS auditing

An auditor verifies that the sequence of STRs does not fork over time.



The BLAST concept



Anchor the sequence of STR to a blockchain. A fork in the STR history implies a fork in the blockchain.

see also Catena (2017) and EthIKS (2016)

Blast Goals

Flexibility

- manages various kinds of data

Transparency

- key inclusion or non-inclusion is publicly verifiable

Non-equivocation

- itentity provider shows the same view to all the users

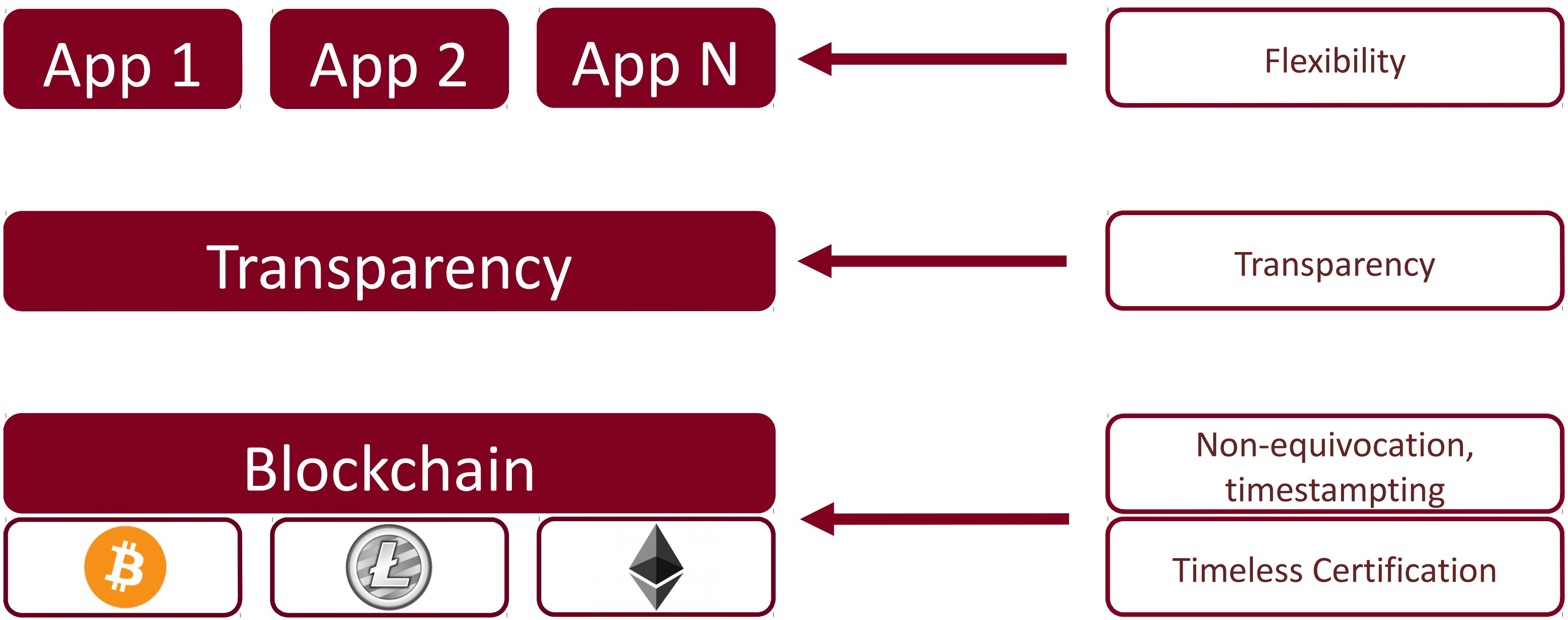
Efficient time-stamping

- older views of the directory are identified as such

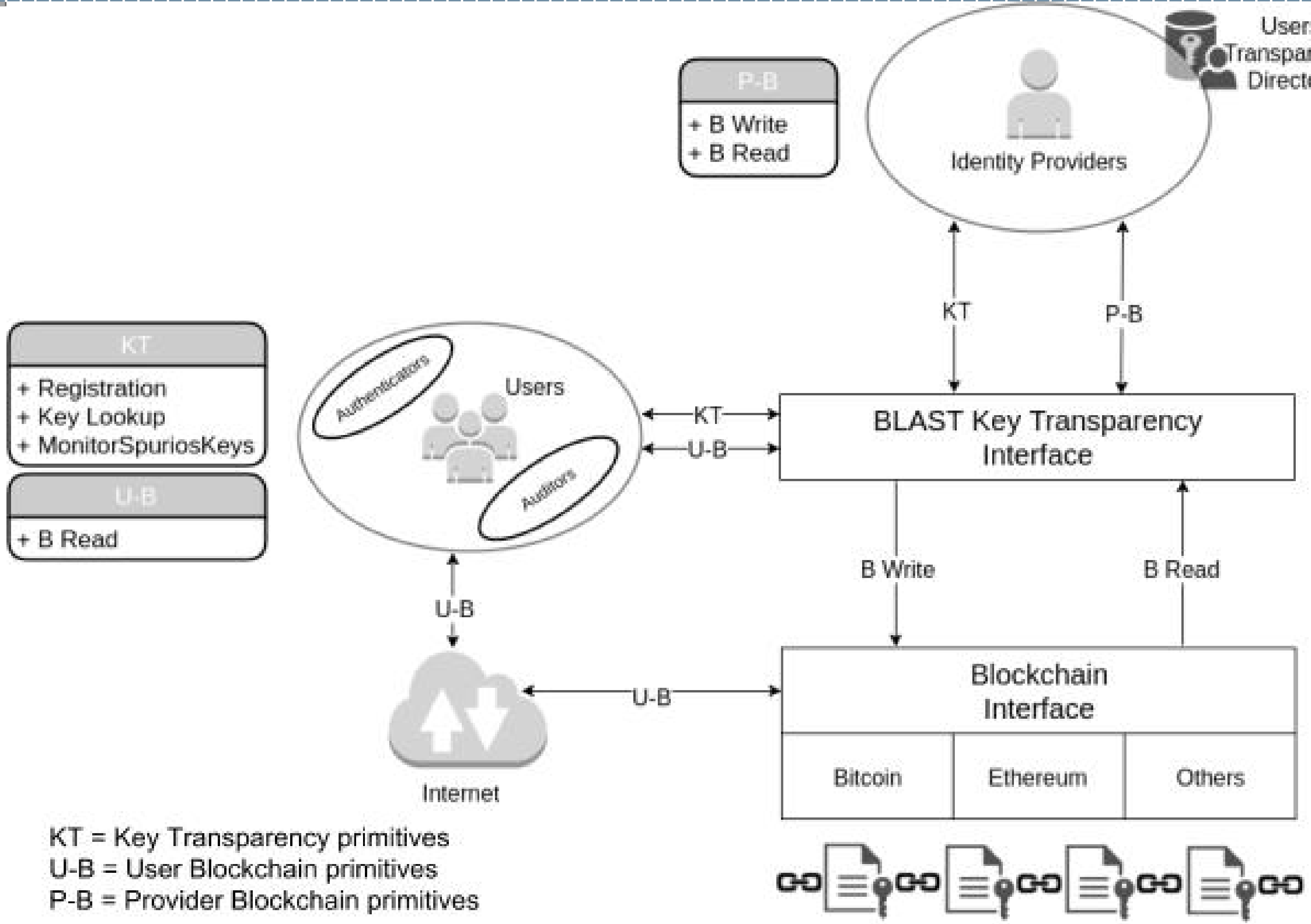
Timeless certification

- works also if the directory provider goes out of business

Protocol Layers & Division of Labor



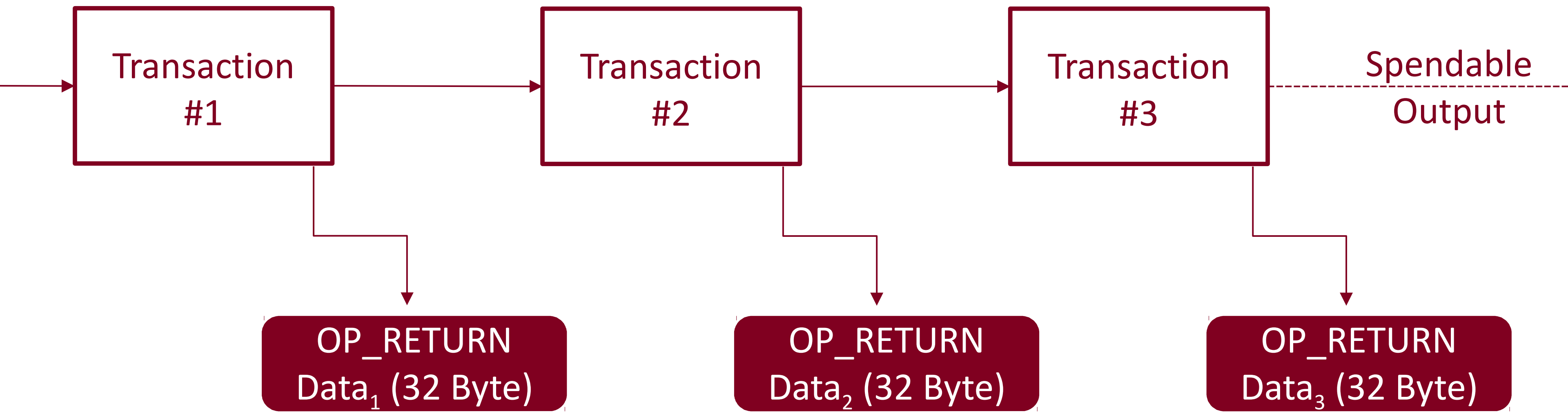
BLAST Architecture



KT = Key Transparency primitives
U-B = User Blockchain primitives
P-B = Provider Blockchain primitives

BLAST over Bitcoin

Output chaining technique



No need for auditors: the end-user / monitoring node can audit.

BLAST node can be a light node (less bandwidth, but requires cooperation of a possibly non-BLAST full node)

BLAST over Ethereum

CONIKS Signed Root Tree Stored within Ethereum and Updated with Smart Contract

- non-equivocation enforced by the smart contract
- audit by end-nodes is much less expensive

```
struct STR {
  uint8 P; //CONIKS security policy (default, strict)
  uint t; // current epoch number of CONIKS history
  uint tPrev; //previous t
  bytes32 tRoot; //current Hash root of Merkle Tree for data
  bytes32 tPrevSTR; //previous Hash root of Merkle Tree for data
  bytes32 rSTR; //first 32 bytes of the signature
  bytes32 sSTR; //second 32 bytes of the signature
  uint8 vSTR; //last byte of the signature
}
```

BLAST node can be a light node (less bandwidth, but requires cooperation of a possibly non-BLAST full node)

BLAST over Ethereum

```
function updateSTR(bytes32 _rSTR, bytes32 _sSTR, uint8 _vSTR, bytes32 _root) {
  bytes32 new_tPrevSTR = sha3(listSTR[msg.sender].tRoot);
  bytes32 check_hash = sha3(listSTR[msg.sender].tRoot, _root);
  if(_vSTR < 27){
    _vSTR += 27;
  }
  address signer = ecrecover(check_hash, _vSTR, _rSTR, _sSTR);
  // if done online, msg.sender must be set to some address
  if (signer == msg.sender) {
    listSTR[msg.sender] = STR(listSTR[msg.sender].P,
                             listSTR[msg.sender].t + 1,
                             listSTR[msg.sender].t,
                             _root,
                             listSTR[msg.sender].tRoot,
                             _rSTR,
                             _sSTR,
                             _vSTR);
  }
}
```

BLAST smart contract for STR update

Properties

Flexibility

- three layer structure

Transparency

- proofs of inclusion or non-inclusion can be verified with on-chain information

Non-equivocation

- equivocation requires forking of the blockchain

Efficient time-stamping

- blockchain provides ordering and (coarse) timestamping

Timeless certification

- proofs of inclusion or non-inclusion can be verified as long as the blockchain is available

Techno-Economic Analysis (as of today)

Blast over Bitcoin

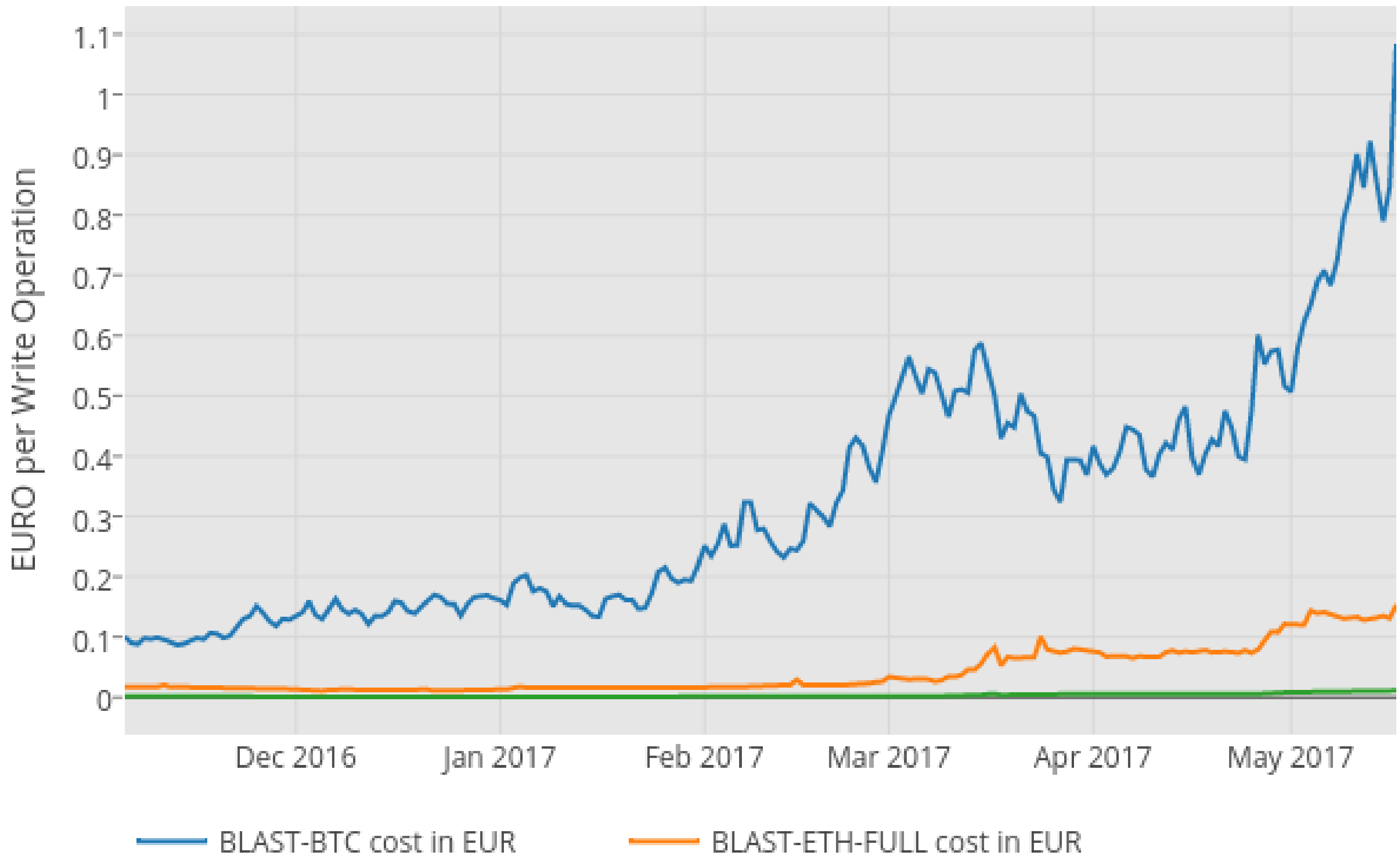
- 267 bytes per epoch
- about 35,000 satoshis per epoch
- about 2 EUR per epoch

Bast over Ethereum with on chain validation

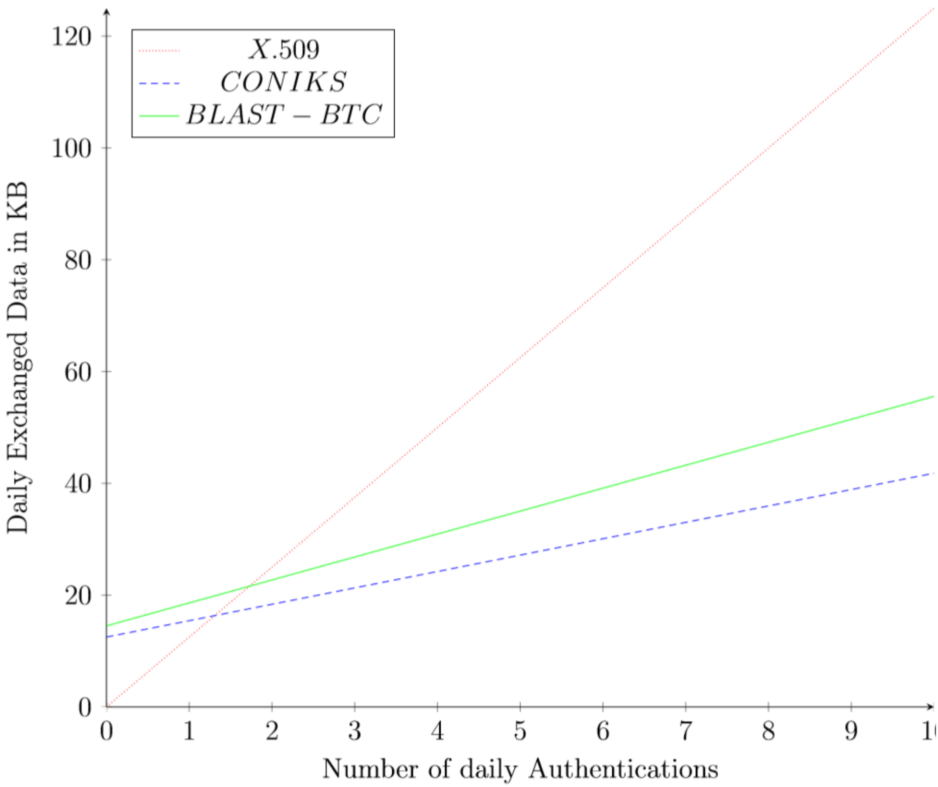
- 73,500 gas per epoch
- about 0.0003 ETH per epoch (@ 20 Gwei)
- 0.25 EUR per epoch

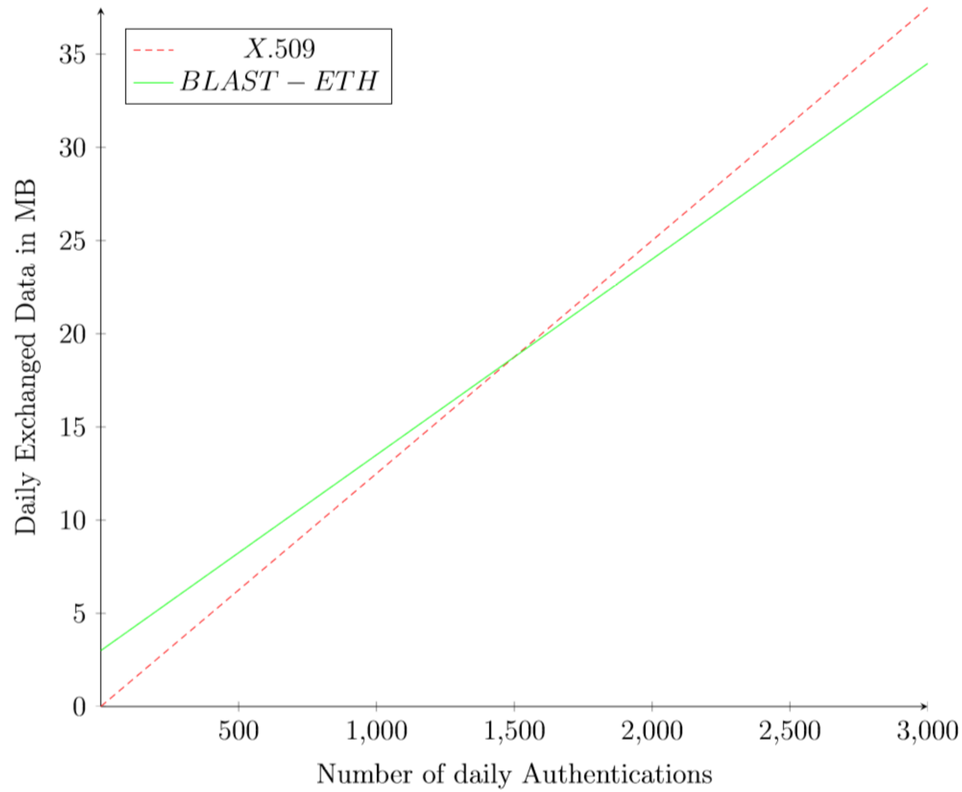
Only the identity provider pays

Techno-Economic Analysis (trend)



Using BLAST for DTLS authentication (instead of X.509)





Using BLAST for DTLS authentication (instead of X.509)

	Fixed	EDGE	3G (HSPA) 4G (LTE)	LP-WAN (Sigfox, LORA)	NB-IoT
DownRate	∞	236.8 Kbps	14.4 - 300 Mbps	Too small	250 Kbps
UpRate	∞	177.6 Kbps	5.8 - 75 Mbps	Too small	20-250 Kbps
Full Client	✓	✗	✓	✗	✗
BTC Light	✓	✓	✓	✗	✓
ETH Light	✓	✓	✓	✗	✓

Conclusion

Generalization of a 3-layer architecture with no technology lock-in

Blockchain applied to gain security properties (Transparency, Non-Equivocation, Timestamping, Timeless Certification)

Cost model for Blockchain layer and a techno-economic analysis and comparison with other solutions

Awards


Bitcoin
building layer2 applications
that matter
oraclize
11-12 May - webinar
13 May - hackathon