BlAsT: Blockchain-Assisted Key Transparency for Device Authentication

Alessandro Gattolin*, Cristina Rottondi◇, and Giacomo Verticale*

*Dipartimento di Elettronica Informazione e Bioingegneria, Politecnico di Milano, Italy
Email: alessandro.gattolin@mail.polimi.it, giacomo.verticale@polimi.it
†Dalle Molle Institute for Artificial Intelligence – University of Lugano (USI)
University of Applied Science and Arts of Southern Switzerland (SUPSI) – Switzerland
Email: cristina.rottondi@supsi.ch

***Abstract*—BlAsT is an efficient scheme for achieving certification of data continuity through a history attached to a public blockchain. The scheme guarantees the properties of linearity, non-equivocation, time-stamping and transparency. We discuss the implementation of BlAsT over the Bitcoin and the Ethereum blokchains, provide a techno-economic analysis to evaluate the costs related to blockchain adoption, and numerically assess the performance of the proposed architecture in terms of storage and bandwidth requirements. Results show that the above properties can be guaranteed with a small cryptocurrency payment. The proposed scheme can be used for IoT devices under the assumption that the device is able to either perform blockchain validation or delegate it to a trusted node.**

***Index Terms*—Blockchain, key transparency, IoT authentication**

I. Introduction

The Internet of Things (IoT) is gaining increasing attention from both enterprise and consumer sides. Even the simplest products are expected to be connected to the Internet in the near future, thus triggering the development of new value-added IT services impacting, among others, supply chain management, logistics, and post-sale services. A basic requirement for most of such services is the management of cryptographic keys. Current approaches are far from perfect and suffer of inadequate trust models, lack of transparency, and difficulties in auditing the behavior of trusted third parties. So far, certificate transparency [1] has been successful at fixing some issues of the TLS certificate system and there have been attempts to extend it to generic records of data associated to an ID (e.g. *CONIKS* [2] and Google Key Transparency [3]). These approaches require, however, a dedicated infrastructure of trusted nodes, which limits scalability. Usage of a public blockchain is therefore an appealing approach for reducing deployment costs by leveraging a shared infrastructure capable of providing a trusted environment.

This paper defines BlAsT, a key transparency infrastructure suitable for implementation over the two most popular public blockchains, i.e. Bitcoin and Ethereum. Then, it evaluates its suitability for deployment in the IoT context. Finally, it evaluates the expected cost of the operation of the BlAsT scheme. To achive this result, we implemented a simple proof-of-concept of the BlAsT solution and run them on the Bitcoin and Ethereum test networks.

The remainder of the paper is organized as follows. Section II overviews related works and highlights their differences with respect to BlAsT. Section III describes the BlAsT architecture, goals and protocols. Section IV discusses the implementation of BlAsT over Bitcoin and Ethereum and compares their requirements and costs. Section V concludes the paper.

II. Related Work

Some recent works have addressed the problem of key authentication. *Certificate Transparency* (CT) [1] is a standard proposed by IETF aimed at solving some SSL structural flaws. CT uses append-only certificates logs that can be queried by any party and defines the roles of monitors and auditors which interact with logs servers to search for dangerous certificates and check their consistency. This open framework enables an early detection of forged certificates and faster mitigation based on certificate revocation mechanisms.

CONIKS [2] introduces privacy-preserving features in order to avoid leaks of other data when querying for specific users. *CONIKS* defines a model for the management of public keys related to email addresses, which is based on the concept of *continuous identity*. The inclusion of a binding *ID - key* does not ensure by itself the correctness of the key, which must be guaranteed by the identity provider, but the binding remains valid over time and the identity owner can verify what keys are associated to his/her identity at the various providers, making it simple to detect spurious keys in the case of a compromised identity provider. The *CONIKS* framework guarantees two properties: non-equivocation and validity. Our proposed framework strengthens such properties by using the blockchain as a single source of truth, which reduces the need for additional gossiping protocols and for continuous auditing processes performed by trusted nodes. Indeed, BlAsT entities must reach consensus only about the data history antecedent to the blockchain join, then they will be able to verify non-equivocation and validity by themselves.

Google Key Transparency [3] is a recent open-source project that builds upon *CONIKS* and *Certificate Transparency*. Its basic principle is that the relationship between an entity and its cryptographic keys must be automatically verifiable and publicly auditable. This is achieved by defining a Merkle Tree that makes it possible to verify the inclusion of any given information. A gossiping protocol between monitors, key servers and users must guarantee that all the key servers share the same state. BlAsT aims at achieving the same objectives by adopting the blockchain technology to reach consensus on the state of the key logs built by a key server.

Among blockchain-based solutions, *EthIKS* [4] relies on the Ethereum [5] protocol, a decentralized platform in which it is possible to reach consensus also about the state of some code using the smart contract paradigm. Since the storage model of Ethereum is based on a Merkle Patricia Tree, *EthIKS* implements the *CONIKS* structure as a smart contract with the aim of guaranteeing the non-equivocation property (i.e., avoiding forks within the network). The drawback of the *EthIKS* solution is that, for every change of information, the identity provider has to update the smart contract state, which has a cost both in term of bandwidth and execution. As the number of updates per day is expected to be relevant, this introduces scalability issues. In comparison to *EthIKS*, BlAsT introduces much smaller costs.

Chronicled [6] proposes to equip smart things with Blouetooth Low Energy (BLE) or Near-Field Communications (NFC) chips, which securely store cryptographic keys that are also paired with an identity on an Ethereum smart contract. In this case, each digital identity can have different values depending on the kind of object it is attached to. *Chronicled*'s solution operates at hardware level, whereas BlAsT focuses on digital *ID - keys* bindings.

Certcoin [7] adopts a blockchain that is specifically designed to replace the current Public Key Infrastructure. It is built over Namecoin, the first fork of Bitcoin. *Certcoin* allows entities to register a new binding *name - cryptographic keys* and, later, to update or revoke the binding, without need of third parties. Differently from BlAsT, *Certcoin* is focused on the verification of the data stored on the blockchain and does not tackle the issuance problem.

III. BLAST: Blockchain-Assisted Key Transparency

A. Protocol Entities

The BlAsT framework comprises the following entities, shown in Figure 1.

a) Identity Provider: each provider is responsible for a different namespace with its own bindings. It receives clients' registration requests, adds the client's data to a Merkle binary prefix tree data structure, and signs the tree root at each epoch. Each Provider holds a public/private key pair for digital signatures and a blockchain address, which is used to identify where the provider writes the data on the blockchain. We assume that the binding among an identity provider, its public key and its blockchain address is known through an external authentication mechanism.

b) User: the end users of the system must use a software client for every interaction with the other actors. A client sends registration requests to the corresponding identity provider, monitors consistency of its bindings and saves results of prior checks. The client may or may not have read access to the blockchain, depending on the capabilities of the devices over which it is installed.

c) Authenticator: when a user needs to contact another user, its client starts a lookup procedure for the authentication data associated to the recipient's ID. The sender client retrieves the recipient's data from the identity provider and then accesses the blockchain in order to verify their consistency and validity.

d) Auditor: auditing by external nodes is required only when bootstrapping new clients. Since the non-equivocation is guaranteed by the uniqueness of the blockchain, once a client knows the blockchain address of an Identity Provider, it will be able to autonomously audit its behavior. The information that must be audited is therefore limited to the data history antecedent to the join of the blockchain.

e) Blockchain: a network of nodes used for storing relevant information that can be accessed in any moment by every entity. It grows in discrete time instants when a block of information is added and is secured thanks to the computational power dedicated by the miners.

B. Framework and Protocol Description

The BlAsT framework adopts a three-layer infrastructure. The *application layer* is in charge of the data management and interacts with the the *transparency layer* by communicating messages in the form of key-value pairs, so that each piece of information can be uniquely identified. The transparency layer is in charge of managing the Merkle tree used to authenticate the dataset. It builds the tree on behalf of the Identity Provider and verifies that a given key-value pair belongs to the tree on behalf of the Authenticator. The root of the Merkle tree represents the commitment of the Provider to a specific dataset at a specific time and is passed to the underlying blockchain layer. The *blockchain layer* enforces the linear history of the dataset, making the commitment publicly available, auditable, immutable, and timestamped. Once the commitment is stored in the blockchain, it is possible to prove that a given key-value pair is, or is not, included in the dataset at a specific time within a certified data history.

To this aim, the BlAsT protocol comprises the following phases:

a) Registration: a client contacts the correct identity provider to register or update a binding between a user ID and a public key in the form (u, PK_u), where u is the user ID and PK_u is the user's public key. The client receives a *proof* of inclusion in the system. Since the dataset is updated only at discrete epochs, the operation is considered as concluded only at the beginning of the next epoch, when the client can check if the binding has been correctly inserted using the received *proof* of inclusion.

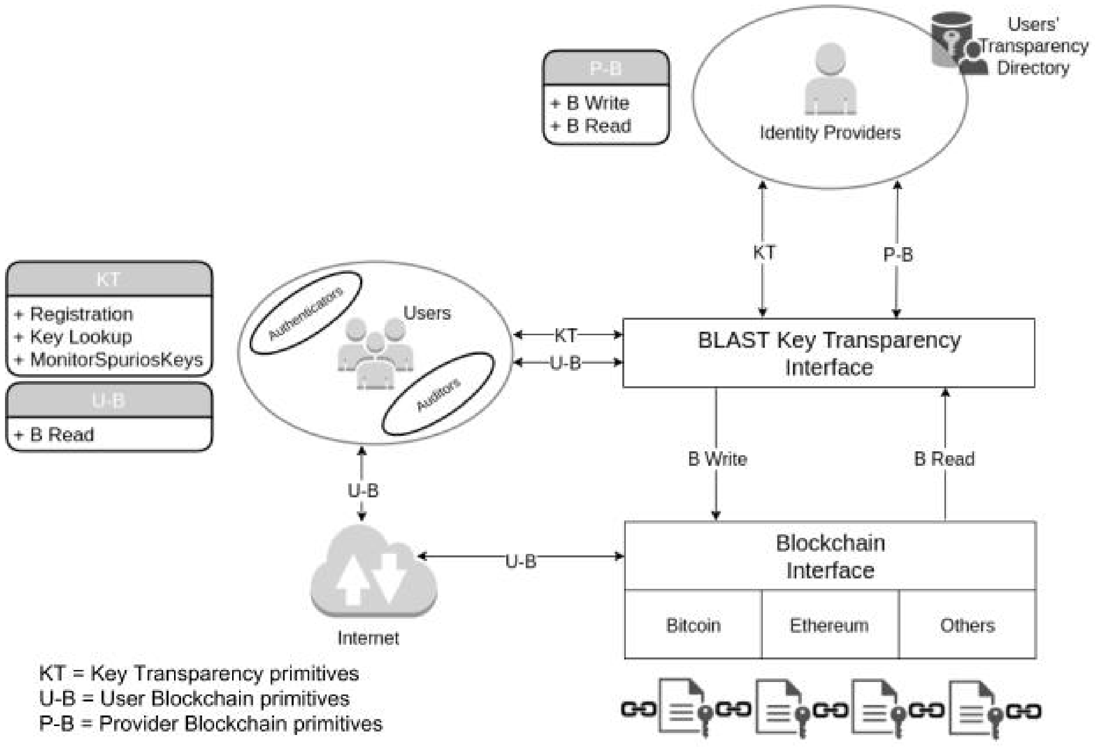


Fig. 1. The architecture and three-layer structure of BlAsT

b) STR write: to guarantee the non-equivocation property, identity providers write on the blockchain the Signed Tree Root (STR). The growth of the STR-chain enforced on the blockchain is discrete and the frequency can coincide with the epoch frequency or can be lowered for the sake of cost reduction. Each identity provider is associated to a blockchain address and holds the corresponding writing rights. This is enforced by the blockchain logic.

c) Key Lookup: clients should always be able to contact an Identity Provider to retrieve the information related to a specific user. If the user is part of its namespace and is currently registered, the Identity Provider returns the user's public key PK_u and the *proof of inclusion* composed by:

- the index j of the user identity in the Merkle binary prefix tree. Indexes are calculated with a Verifiable Random Function to prevent leaking information about the users in the dataset;
- a cryptographic commit to the pair (u, PK_u);
- the authentication path in the Merkle tree from the above commit to the root of the tree;
- the signature of the root of the Merkle tree.

Otherwise, the identity provider returns a *proof of absence* for the index j corresponding to the requested user identity. The proof is calculated in the same way as [2]. The proof of absence is used by the client to detect possibly spurious keys.

d) STR retrieval: to verify the consistency and validity of the data managed by the identity providers, clients retrieve from the blockchain the most recent STR. A client with direct read access to the blockchain can perform these checks autonomously. Otherwise, the client can rely on some trusted nodes to perform part of the validation.

e) Monitoring of spurious keys: periodically, each client performs a key lookup upon its identity providers for its own identity to verify the correctness of its binding. Each client also performs a key lookup upon various others identity providers to verify that there are no spurious keys associated to its identifier.

f) Auditing for non-equivocation: Since the blockchain guarantees that, once the address of an identity provider is known, the clients are able to follow the growth of the blockchain to get the new STRs, the auditing must be performed only at bootstrapping time for every client, which then could be completely independent from external auditors. A client can decide to query the blockchain at each epoch or to get updated only when it has to verify the consistency of new data.

C. Device Capabilities

A *read* operation from a public blockchain does not cost any money. However validating the read data requires some computational effort, which impacts the cost and complexity of the client node and the suitability of the solution for IoT devices. Depending on the capabilities of a device, different clients can be installed:

a) Full node: it stores locally the whole blockchain with complete blocks. In this way it is possible to locally validate both transactions and blocks. A full node does not trust any other single node of the network. Since everything is stored locally, a read operation can be completed locally. The main drawback is the cost of the device, both in terms of storage and bandwidth.

b) Light node: differently from a full node, a light node only downloads the block headers, the transactions it is interested in, and a proof of inclusion of these transactions in the blockchain. In the bitcoin network this technique is known as *SPV - Simple Payment Verification* and requires less bandwidth and storage than running a full node. A light node does not trust any other single node of the network, but requires the cooperation of a full node for carrying on read operations.

c) Non-blockchain enabled: a device that has not enough capabilities to run a full or light client, must contact a trusted remote node. This configuration is the lightest in term of hardware and bandwidth requirements, but introduces a trust problem towards the remote node.

D. Properties

BlAsT aims at satisfying the following properties:

- *Transparency*: thanks to the use of a Merkle Tree, a small proof of inclusion or absence at the transparency layer is able to guarantee that the client receives all the data associated to a user ID at a given epoch at a given Identity Provider. The use of permission-less blockchains makes it possible for anyone to verify that the answer is correct without further involvement of the Identity Provider.
- *Non-equivocation*: the blockchain acts as a single source of truth. After a user has solved the bootstrapping issue, he/she is able to move backward and forward through the linear history of data directly on the blockchain, in order to verify the uniqueness of the history. The only possibility for a compromised Identity Provider to maintain multiple instances of the key database, is to anchor each instance to a different fork of the blockchain. This is considered unfeasible except for very short periods.
- *Efficient time-stamping*: when data are written to the blockchain, they are also timestamped, since every block has a timestamp header field. Thanks to this time information, if an attacker obtains the Identity Provider's secret key, he/she cannot forge a new history and date it back to the past since it is considered not computationally feasible to rewrite the blockchain history.
- *Timeless certification*: this property is a direct consequence of the previous three. If the Identity Provider that certified some data becomes inactive, it is always possible to certify the validity of its data if we have such data and the related proof. This is possible because the blockchain history will still be accessible by anyone and this property remains valid as long as the blockchain remains alive.

IV. Example Implementations

In this Section, we discuss how our proposed framework can be integrated with existing blockchain technologies such as Bitcoin and Ethereum. Moreover, we evaluate the storage and bandwidth requirements necessary for the validation of the information retrieved by the blockchain.

A. BlAsT over the Bitcoin Blockchain

BlAsT implementation on the Bitcoin blockchain is very similar to Tomescu's Catena [8]. At each epoch, the Identity Providers write the signature of the Merkle tree root in a Blockchain transaction using the `OP_RETURN` opcode, which makes it possible to store arbitrary data. To save the data and update their value over time, BlAsT uses an approach similar to the transaction chain defined in [9]. The identity provider is identified by one blockchain address that represents the entry point for its data on the blockchain. All the entities in the framework must reach consensus about such address. Ideally, the address should be related to only two transactions: a *charge transaction*, which is motivated later, and the first *STR transaction*. An *STR transaction* has one input and two outputs. One output is an `OP_RETURN` and includes the STR, while the other is a classic `P2PKH` output toward a new blockchain address that is also controlled by the identity provider. When the provider wants to update the STR at a later epoch, he creates a new *STR transaction* that spends the output of the previous *STR transaction*. In this way, all the STRs are chained together by Bitcoin's transactions and an identity provider cannot decide to fork this history, thus being prevented from double-updating the STR-chain. Note that, even if the *STR chain* is enforced on the blockchain, the miners do not care about the data stored through the `OP_RETURN` script. Therefore, when a client retrieves a new *STR*, it has to check its correctness with respect to the previous one.

a) Blockchain storage: At the current market rates, the cost of storage depends on the size of the transaction, the acceptable delay before the transaction is included in a block, and the size of the queue of other users' transactions waiting to be included in a block. We used 21.co (now earn.com) tool that predicts Bitcoin transaction fees [10]. It uses a Montecarlo simulation in order to predict miners and queue sizes. The simulation tool outputs an estimate of the fees to be paid in Satoshi/Byte to get a 90% confidence interval that a transaction will be included in a block after a certain delay. A BlAsT transaction (with one input and two outputs) accounts for about 267 bytes. Table I shows the estimated fees as of May 2017 with an exchange rate of 1100 EUR/BTC. If the epoch is set to 1 hour, then it is advised to consider only the first column with a 160 Satoshis per byte fee. If instead the epoch is set to 1 day, all the columns can be considered.

b) Client requirements: In case of a *full node*, considering the current block size limit of 1MB and an average block time of 10 minutes, the local blockchain grows 144MB per day. The bandwidth requirements are harder to evaluate since they depend on the client configuration. The official Bitcoin Core implementation requires a download of 500MB/day and an upload of 5GB/day for a full node which actively contributes to the whole network. The bare minimum requirements can be around 150MB/day for download and 10MB/day for

TABLE I
TRANSACTION COST EXPRESSED IN SATOSHIS AND EURO WITH DELAY INFORMATION

	Transaction cost (satoshi/byte)		
	160	140	120
Delay (Blocks)	0	0 - 1	0 - 23
Delay (Minutes)	0 - 25	0 - 40	0 - 300
Tx. cost (Satoshis)	42720	37380	32040
Tx. cost (EUR)	0.47	0.41	0.35

TABLE II
SOLIDITY SMART CONTRACT EXECUTION COSTS FOR DIFFERENT FUNCTIONS

	gas	Ether	EUR
Deploy	841135	0.0168227	0.76
Init	159510	0.0031902	0.15
1st Update	103760	0.0020752	0.10
n-th Update	73760	0.0014752	0.07

upload. The bootstrapping requires at least a download of 100GB for recovering the blockchain starting from the genesis block.

Conversely, a *light node* is only required to download around 12KB/day since each block header accounts for 80 Bytes, plus some overhead for communication and keep-alive messages (again in the order of some KB). The retrieval of one block from a peer node requires at most 1MB.

Finally, for *non-blockchain enabled nodes*, assuming a scenario in which it is possible to establish a trust relationship with another peer node, a query that already returns the desired STR requires an exchange of only 7.5KB. This value has been evaluated using the Blocktrail platform, which accepts interactions only over SSL, and is a lower bound that does not consider possible overhead due to errors during transmission.

B. BlAsT over the Ethereum Blockchain

Ethereum is a public blockchain with an underlying cryptocurrency called *Ether*. An address can be an *externally owned* address and being associated with a wallet, or a *smart contract* address, which is associated not only to a wallet but also to a code and state. Using a high level language such as Solidity [11], it is possible to write the logic of a decentralized application that can be deployed on the public Ethereum blockchain and accessed through its smart contract address. To change the state of a smart contract it is sufficient to send a transaction to the smart contract address and the change of state happens when this transaction is included inside a block. Each operation has a cost expressed in gas units, which is fixed by the Ethereum protocol. A user that wants to execute a function of a contract must decide how much *ether* he/she is willing to spend for every unit of gas, and this is expressed through the *gasPrice* attribute of a transaction. The higher the *gasPrice*, the higher the probability that a miner will choose the transaction for creating the next block, because it will get a higher reward in case it manages to solve the hash puzzle. With the smart contract paradigm, we can include some BlAsT logic on the blockchain.

a) Blockchain storage and execution: We have developed a Solidity smart contract which is able to update the STR of a specific provider only if it is cryptographically chained to the previous STR, something that, with Bitcoin, must be verified by the client that retrieves the STR. The costs to be evaluated are the *execution cost*, which is determined by the operations that must be executed on-contract, and the *transaction cost*, that includes both the execution cost and the size of the transaction that is sent to the peer-to-peer network. The following results consider the *transaction costs* derived with the online Solidity compiler for different functions:

- *Deploy:* the operation that makes the contract logic available to anyone and that returns an Ethereum address. The transaction cost is 841135 gas units. It is performed only once, namely when BlAsT is used for the first time.
- *Init:* each identity provider is assigned an *STR struct* which contains 8 fields (security policy, current epoch, previous epoch, current Merkle Root, previous Merkle Root, and the ECDSA signature parameters r, s, and v). The first STR write operation costs 159510 gas units.
- *Update:* the updates to the contract are cheaper than the init operation. The first update consumes 103760 gas units, while the following ones consume 73760 gas units.

In Table II we show the current cost of execution with a standard *gasPrice* of 0.00000002 Ether per unit of gas and a conversion approximated to 1 ETH = 45 USD. If we consider the update operation in Ethereum and STR-write in Bitcoin, the former has consistently been from 5 to 7 times cheaper than the latter throughout the first half of 2017 even if Ethereum is also in charge of the continuity check of the STR-chain.

b) Client requirements: Ethereum does not impose a block size limit: each block has a *gasLimit* attribute and the number of transactions that can be included depends on the nature of the code being triggered for execution. Two transactions that occupy the same data volume might therefore consume different amounts of gas. For our analysis we consider the average values of the most important parameters of Ethereum [12]:

- *Gas limit:* 4,170,000
- *Block size:* 2.5MB
- *Block time:* 14,5 seconds

In the case of a full node, a day sees around 6000 blocks, which accounts for 15GB new data per day. The bandwidth requirements are then very high, but in term of storage it is possible to adopt State Tree Pruning [13] techniques that reduce the required storage requirements (around 12GB with Geth [14] and 6GB with Parity [15], the most used client).

Conversely, a light node adopting the Light Ethereum Subprotocol (which has been included for the first time at the end of 2016 in Geth 1.5) operates only on block headers, which are composed by different fields reaching about 500 Bytes, resulting in a bandwidth consumption of around 3MB per day. With pruning techniques the required storage can be reduced to 10 MB, but current implementations use around 200 MB.

Finally, a non-blockchain enabled node can rely on the INFURA service to get access to a remote node [16]. INFURA exposes both mainnet and testnet nodes. Alternatively it is possible to contact a known Ethereum node.

C. Techno-Economic Comparison

We have monitored Bitcoin exchange rates, Bitcoin fees, Ether exchange rates and gasPrice average values over the period from November 6th, 2016 to May 18th, 2017. In Figure 2 we show the cost of a write operation for the three implemented scenarios: BlAsT over Bitcoin (blue line), BlAsT over an Ethereum full node (orange line), and BlAsT over an Ethereum light node (green line).

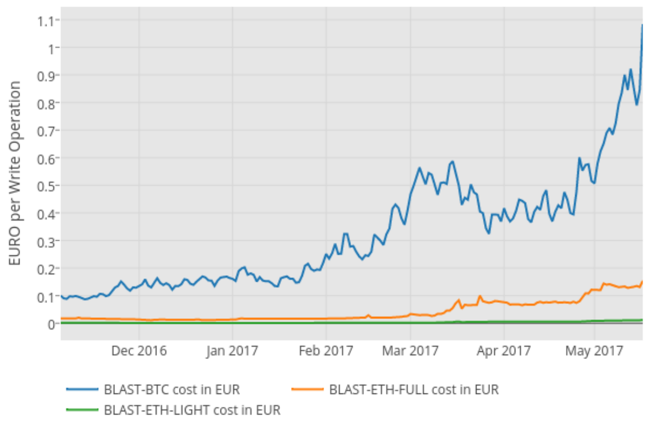


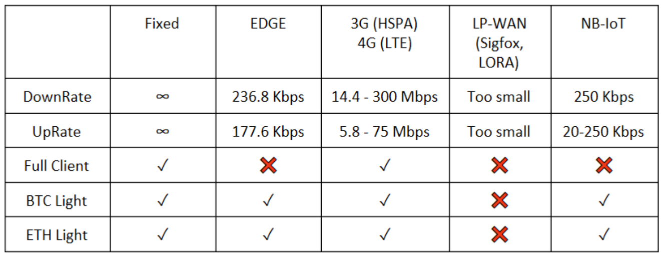
Fig. 2. Cost trend (in EUR) of BlAsT write operations

Considering different client configurations, in Table III we show the bandwidth requirements with some wireless access technologies in order to understand which devices can run a full or light node. We consider the fixed broadband connections as having unlimited speed, as well as popular mobile technologies suitable for long range communication with devices. In particular we consider the 2G mobile standard EDGE, which has a too small bandwidth for acting as a full client, but could provide enough bandwidth for a light node, both in the BTC and ETH implementations. The 3G/4G mobile standards have a large bandwidth that can support full nodes. Low Power WAN technologies such as SigFox or LoRa have a too small bandwith and can support neither a full node nor a light node. They might support a non-blockchain-enabled node. The 4G mobile standard NB-IoT is expected to provide enough bandwidth to support light BTC and ETH nodes, while it cannot support full nodes.

V. Conclusion

In this paper we proposed a blockchain-based approach to achieve key management transparency thanks to a three-layer architecture named BlAsT, which allows for the creation of an immutable and publicly certifiable linear history. The proposed framework can be built over different blockchain technologies, such as Bitcoin and Ethereum. To identify the most promising candidate, we performed a techno-economic analysis of costs related to blockchain write operations and bandwidth consumption. Results show that BlAsT is a cost-effective technique that makes it possible to end users to monitor what keys are associated to their identities and identify spurious keys.

TABLE III
Client configurations and access technologies



	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	✓	✓	✓	✗	✓

References

[1] B. Laurie, A. Langley, and E. Kasper, “Certificate transparency,” Internet Requests for Comments, RFC Editor, RFC 6962, June 2013.
[2] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, “Coniks: Bringing key transparency to end users.” in *Usenix Security*, 2015, pp. 383–398.
[3] Google, “Google key transparency,” 2017. [Online]. Available: https://github.com/google/keytransparency
[4] J. Bonneau, “Ethiks: Using ethereum to audit a coniks key transparency log,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 95–105.
[5] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.
[6] “Chronicled – smart supply chains solutions.” [Online]. Available: https://chronicled.com/
[7] C. Fromknecht, D. Velicanu, and S. Yakoubov, “A decentralized public key infrastructure with identity retention.” *IACR Cryptology ePrint Archive*, vol. 2014, p. 803, 2014.
[8] A. Tomescu and S. Devadas, “Catena: Efficient non-equivocation via bitcoin,” in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 393–409.
[9] C. Jämthagen and M. Hell, “Blockchain-based publishing layer for the keyless signing infrastructure,” in *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences*, 2016.
[10] “Predicting bitcoin fees for transactions.” [Online]. Available: https://bitcoinfees.earn.com/
[11] “Solidity.” [Online]. Available: https://solidity.readthedocs.io/en/v0.4.23/
[12] “Ethereum charts and statistics,” 2017. [Online]. Available: https://etherscan.io/charts
[13] V. Buterin, “State tree pruning,” in *Ethereum Blog*, 2015. [Online]. Available: https://blog.ethereum.org/2015/06/26/state-tree-pruning/
[14] “Geth.” [Online]. Available: https://github.com/ethereum/go-ethereum/wiki/geth
[15] “parity.” [Online]. Available: https://www.parity.io/
[16] “Infura – scalable blockchain infrastructure.” [Online]. Available: https://infura.io/