Design and Implementation of InBlock, A Distributed IP Address Registration System

Alberto García-Martínez, Stefano Angieri, Bingyang Liu, Fei Yang, Marcelo Bagnulo

Abstract—The current mechanism to secure BGP relies on the Resource Public Key Infrastructure (RPKI) for route origin authorization. The RPKI implements a hierarchical model that intrinsically makes lower layers in the hierarchy susceptible to errors and abuses from entities placed in higher layers. In this paper we present InBlock, a Distributed Autonomous Organization that provides de-centralized management of IP addresses based on blockchain, embedding an alternative trust model to the hierarchical one currently implemented by the RPKI. By leveraging on blockchain technology, InBlock requires consensus among the involved parties to change existent prefix allocation information. InBlock also fulfills the same objectives as the current IP address allocation system, i.e., uniqueness, fairness, conservation, aggregation, registration and minimized overhead. InBlock is implemented as a set of blockchain smart contracts in Ethereum, performing all the functions needed for the management of a global pool of addresses without human intervention. Any entity may request an allocation of addresses to the InBlock registry by solely performing a (crypto)currency transfer to the InBlock. We describe our InBlock implementation and we perform several experiments to show that it enables fast address registering and incurs in very low management costs.

Index Terms—IP address allocation, Blockchain, Distributed Autonomous Organization, Smart Contract, Ethereum.

1 Introduction

The Border Gateway Protocol (BGP) is the protocol of choice for exchanging routing information between different administrative domains in the Internet. As such, BGP has become part of the critical infrastructure required for the Internet and by extension of the modern society as whole. For the same reason, BGP has also become a preferred target of attacks directed to derange the normal operation of the Internet. By subverting BGP, attackers can hijack and/or eavesdrop communications as well as execute denial of service attacks. To prevent these attacks the Internet community has developed the Resource Public Key Infrastructure (RPKI [1]) and BGPsec [2]. These tools provide several cryptographic guarantees such as ensuring that the Autonomous System (AS) announcing a route towards an IP prefix is indeed the legitimate holder of the resource according to the allocation rules, preventing prefix hijacking and other vulnerabilities.

The RPKI is a public key infrastructure with a trust hierarchy tree that is aligned with the IP allocation hierarchy. Through the RPKI, the elements in the IP address delegation chain, i.e., IANA, Regional Internet Registries (RIRs), Local Internet Registries (LIRs) and end-sites, can obtain certificates that bind prefixes and public keys. The holder of an RPKI certificate can then sign a Route Origin Authorization (ROA) statement to indicate that a given AS number is entitled to originate a BGP route for that prefix, using the private key associated to a given prefix, and include it in the RPKI. Third parties can validate BGP routes by contrasting the received announcement with the corresponding ROA. In order to prevent routing attacks, routes for which the ROA validation fails are expected to be discarded.

The RPKI standard dates back to 2012 [1]. After eight years during which numerous successful attacks against BGP have been widely reported in the media (e.g., [3], [4]), the deployment of the RPKI is still far from universal. According to the global RPKI deployment monitor[1], more than 80% of prefixes announced in the global routing table cannot be validated using the RPKI. While there are multiple reasons that explain this lag on the RPKI deployment [5], different voices [6], [7] have raised concerns regarding the centralisation effects that the RPKI has on the global routing system. Indeed, under the RPKI model, the entities in the top of the hierarchy may now, through the capabilities provided by the RPKI mechanisms, change at will and in real time the validity of the allocations made by the inferior levels in the hierarchy. As it has been previously shown [8] this opens the door to a number of errors and abuses from higher entities in the hierarchy, with potentially catastrophic consequences for the rest. This centralisation of the enforcement power over the global routing system is a distinctive feature of the RPKI deployment.

Centralisation (a.k.a. consolidation) in the Internet has received a lot of attention lately. There are more and more parties that warn about the impact of centralisation in the Internet architecture, including the Internet Architecture Board (IAB) [9] and the Internet Society [10]. In particular,

Manuscript received XXXX; revised XXXX; accepted XXXX; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Argyraki. Date of publication XXXX; date of current version XXXX. The work of A. García-Martínez was supported by 5GROWTH project, H2020, Grant agreement ID: 856709. The work of S. Angieri and M. Bagnulo was partially funded by the InBlock project of the Huawei Innovation Research Program.
A. García-Martínez, S. Angieri and M. Bagnulo are with the Universidad Carlos III de Madrid, 28911 Madrid, Spain (e-mail: alberto@it.uc3m.es; sangieri@pa.uc3m.es; marcelo@it.uc3m.es).
B. Liu and C. Wang are with the Huawei, 100095 Beijing, China (e-mail: liubingyang@ huawei.com; wangchuang@huawei.com).
Digital Object Identifier KK.KKKK/XXX.XXXXXX

1. https://rpki-monitor.antd.nist.gov/

the IAB has identified four main areas of concerns regarding the centralisation of the Internet infrastructure [11], namely, *reliability* (given that the central entity becomes a *single point of failure*), *surveillance*, *concentration of information* and *extended scope* of the effects of the actions of a few entities. The RPKI brings all these concerns to the routing system. Indeed, the top of the hierarchy now becomes a *single point of failure*, and if compromised can jeopardise the integrity of the internet routing as whole. In the same vein, the top of the RPKI can be abused to modify BGP routes to divert global traffic towards specific eavesdropping points, having both *surveillance* and *concentration of information* implications. The *scope of the effects* of the actions of the top tiers of the RPKI becomes now *global*. Observe that we are not claiming that the entities at the top of the RPKI hierarchy are likely to abuse the power granted to them. We are merely challenging whether from an architectural viewpoint, the centralised approach of the RPKI is the most appropriate one for a global distributed infrastructure such as the Internet.

In this paper, we present InBlock, an alternative system to perform route origin authorization based on a decentralized trust model, aiming to limit the centralization of enforcement power in the higher levels of the allocation hierarchy. InBlock relies on the use of a blockchain to store address allocations and route origin authorizations. By storing prefix allocation information in the blockchain, InBlock embraces a different trust model in which blockchain consensus is required to revert or subvert an existing allocation. InBlock is defined as a set of smart contracts, i.e., code that runs autonomously in the blockchain, that manages IP addresses without any human intervention. While InBlock can support both IPv4 and IPv6, we focus on IPv6 as it has a large remaining pool of unassigned addresses.

1.1 Main Contributions.

The first contribution of this paper is the design of the InBlock, a system to securely manage a global pool of IPv6 addresses in a fully de-centralized manner. By relying on blockchain technology, the proposed InBlock design provides a distributed, automatic, irrevocable, tamper-free, publicly accessible, privacy-preserving resource allocation mechanism for the Internet. At the same time, the proposed design of the InBlock complies with the goals stated for the current Internet address assignment mechanism, i.e., uniqueness, fairness, conservation, aggregation, registration and minimized overhead [12]. We illustrate the proposed design by describing the overall operation of the proposed InBlock solution, including the different roles involved and the functions performed by each of them.

The second contribution of this paper is a Proof-Of-Concept implementation of InBlock. InBlock is implemented as a set of smart contracts in Ethereum [13]. We describe the different *transactions* and *calls* that are implemented to manage the IP address pool. The InBlock implementation is open source and publicly available.

The third contribution of this paper is the experimental validation of the proposed InBlock design and implementation. We deployed our InBlock implementation both in a private testebe and the public blockchain and assessed its functionality and performance. We systematically test all the *transactions* and *calls* implemented. We observe that InBlock is able to perform the IPv6 registry functions in an efficient manner, significantly reducing the operational costs compared to a traditional registry and also reducing in orders of magnitude the time required to perform an allocation. The scripts used to test the implementation are also publicly available to enable the reproducibility of the reported results.

2 Background on IP address management and BGP security

2.1 IP address management

The global pool of IP addresses is managed by ICANN, the Internet Corporation for Assigned Names and Numbers. ICANN has delegated the address management duties to the five Regional Internet Registries, RIRs, namely AFRINIC (Africa), APNIC (Asia Pacific region), ARIN (mainly US and Canada), LACNIC (Latin America and the Caribbean), RIPE (Europe, Middle East and Central Asia). RIRs are open membership-based bodies composed primarily of organizations that operate networks. The address resources received from ICANN are assigned according to policies developed regionally by each RIR, although coordinated with the rest. Then, the resources are allocated to their requesters according to the policies defined by each RIR (e.g. [12].

RIRs can allocate Provider Aggregatable (PA) blocks to Local Internet Registries (LIRs) and they also provide direct Provider Independent (PI) assignments directly to end-users. All RIRs define a minimum PA IPv6 allocation of /32 (e.g. see [12]).

2.2 BGP security

BGP is used to exchange prefix reachability information between the different networks in the Internet. The networks participating in the BGP protocol are identified by *AS numbers*, 32-bit unique identifiers. The lack of security features of the original BGP specification has enabled attackers to manipulate the routing information, or propagate legitimately held information in unintended ways. The effects of an attack to BGP include [14]:

- the traffic affected can experience a reduction of the quality of the traffic transmission, even resulting in the complete discard of the traffic (*blackholing*)
- the traffic affected can be manipulated or eavesdropped (*man-in-the-middle*)
- the services of the victim can be impersonated (*imposture*)

An example of one of such an attack occurred in December 2017 [3]. Prefixes usually originated by Google, Apple, Facebook, Microsoft, Twitch, NTT Communications and Riot Games were diverted for three minutes through networks that were not supposed to receive this traffic. Many other attacks against BGP occurred before and after this particular incident. In order to ameliorate the effect of such incidents, the BGP ecosystem has been enhanced with origin and path validation capabilities.

Origin validation is provided by the RPKI [1] architecture. This architecture defines a distributed repository of X.509 certificates used to assert that an entity is the legitimate holder of a set of IP addresses or a set of AS numbers. At the leafs of the RPKI certificate chain we find Route Origin Authorizations (ROAs), by which the holder of the prefix authorizes a network with an AS number to originate advertisements for it. A ROA is only valid if the whole certificate chain from which it is derived is valid.

The RPKI chain of trust follows the structure defined by the hierarchy of authorities involved in the allocation process for the IP address space and AS numbers. Thus, in order for an End User network to be able to issue a ROA for its prefixes, it needs a certificate issued by the LIR from which it received the addresses; in turn this LIR should have received a certificate from its corresponding RIR.

The chain of trust defined for the RPKI also implies that the RIRs are the natural *Trust Anchors* to configure in order to bootstrap the process of certificate validation.[2]

A router performing origin validation checks if the information included in a BGP route regarding to the AS that originated the announcement of a prefix matches with the content of a valid ROA. The certificate path, and eventually the ROAs, are discarded if the validation fails.

2.3 Adverse actions against the RPKI

We now describe the set of adverse actions that can be carried out against the different components of the RPKI. In the context of the RPKI, an adverse action is defined as a modification of the information published by the RPKI regarding a set of prefixes that is against the wishes of the holder of the resources. There are different forms of adverse actions [15]:

Deletion is the removal of an object from the publication point, against the wishes of the resource holder.

Suppression is failing to delete or publish an object according to the resource holder wishes.

Corruption is the modification of an object without including a valid signature. The object will not pass the signature validation checks.

Modification is the publication of a new valid object that is different to (and replaces) the current version approved by the resource holder.

Revocation of a certificate, so that the object will not pass the signature validation checks. This is achieved by including the certificate in the the appropriate Certificate Revocation List, without permission of the resource holder.

Injection is the creation of a new valid object into a publication point, without permission of the resource holder.

These actions can be taken against the CA certificates, the ROAs, the manifest and the revocation list. The three first actions can be performed by the publication point of the objects of the resource holders. The latter three actions require compromising the private key of the resource holder. Also, all of them can be executed by a parent CA. For a more detailed description of the possible actions, the reader is referred to [15].

2. Although Trust Anchor selection remains local to the validating network, who may add/remove Trust Anchors at will (for example, to support internal routing policies), in practical terms the anchors of the five RIRs must be configured, or the security information for large regions of the Internet will remain unknown, defeating the purpose of the mechanism.

3 Blockchains and Smart Contracts

A blockchain[16] is an immutable distributed ledger that records validated transactions permanently without the need of a trusted third party. The blockchain is a distributed ledger since all the information is stored in all the nodes composing the blockchain peer-to-peer network.

The blockchain is composed of a growing list of blocks, securely linked between each other through cryptography. Every block contains a hash pointer to a parent block, a timestamp and transactions' data. The addition of new valid blocks is determined through a distributed consensus mechanism. The consensus is an emerging artefact representing the agreement reached by the nodes of the blockchain network regarding the blocks to be added. The most popular consensus mechanism is Proof-of-Work (PoW).

In PoW, nodes try to solve a complex mathematical problem in order to gain the right to append a block to the existent chain (and make some profit). New block signers are chosen through a *mining* race. Every time a block is added, a new mining race starts and every miner tries to find the solution to gain the next mining block contest and receive the related fee.

The hash structure of the blockchain makes computationally unfeasible to alter the data of one block without the manipulation of all subsequent blocks. Tampering the ledger then requires both the collusion of the majority of the network and an enormous amount of computational power to rebuild the chain from the replaced block. This is the sense in which we interpret the immutability of the blockchain.

The blockchain paradigm can be extended to the automation of complex resource manipulation and transference procedures in a transparent and trustable manner, by means of the specification of *smart contracts*. A Smart Contract is a program stored in the blockchain that is executed by the nodes of the blockchain network. Smart Contracts can store state in the blockchain, and retrieve it securely (e.g., ensuring that the value retrieved corresponds to the last modification stored in the blockchain).

Blockchain and smart contracts technology is being adopted in a vast number of scenarios, including finances, Internet of Things, health care, energy, education and more [17]. There are multiple public blockchains currently available some of them with a large number of users.

Ethereum [13] is a popular public blockchain platform created to facilitate the development of Smart Contracts. Ethereum implements a Proof of Work (PoW) based consensus mechanism. Miners are rewarded in Ether, the Ethereum cryptocurrency, for the storage and processing power they contribute with [18]. Processing operations are defined through a built-in Turing-complete programming language, Solidity, that is executed in blockchain nodes by an Ethereum Virtual Machine. Every operation in the Ethereum network is triggered by transactions between accounts, either *Externally Owned Accounts* (EOAs), controlled by users, or *Contract Accounts*, associated to a Smart Contract with code and state stored with the account itself. Being a public blockchain, any party can create one or more EOAs and deploy (and run) smart contracts in Ethereum.

Smart Contracts are deployed in Ethereum through a blockchain transaction. Once deployed, the blockchain nodes will execute the code corresponding to the Smart Contract and modify its state whenever a monetary transaction directed to the Contract Account is received. Every Ethereum transaction includes a transaction-fee payable to the *miners*. Miners execute the smart contract, verify the validity of the transactions, i.e., that the transactions complies with the rules expressed in the smart contract, and, if valid, compute the resulting state. This state is eventually appended to the blockchain. Every miner executes the code of the contract, reaching the same final state.

The value (in Ether) of the transaction fee is set by the user issuing the transaction and should reflect the execution and storage cost of an operation, as well as the priority that the user wants to get from the blockchain miners. Higher transaction fees imply that the transaction will be processed earlier by the miners. The fee of a transaction is represented in Ethereum through the concept of *gas*. A fixed amount of gas is assigned to each operation and each transaction sets its own *gas price*. The amount of Ether that the issuer of the transaction transfers to the miner is the amount of gas required by the transaction multiplied by the gas price offered. Also, every transaction defines the gas limit field to set the maximum amount of gas that the transaction may consume. If during the runtime of the code associated to a transaction the gas used exceeds the gas-limit defined for the transaction, the processing is stopped with an *out-of-gas* error. Finally, Ethereum defines a global upper bound to the gas limit value for any single transaction.

Ethereum Smart Contracts can also include *calls*, code that does not generate blockchain annotations and does not incur in a transaction cost. A call operates over the blockchain data to retrieve information, perform validations, etc. Calls can be used as a mean to the smart contract designer to provide a canonical way of retrieving, elaborating or validating blockchain information. The execution of this code provides a higher compliance with the intentions of the smart contract designer, as opposed to an equivalent specification in pseudo-code, or a third-party implementation. Ethereum also imposes gas restrictions on the execution of calls. The reader is referred to [18] for further information regarding blockchains in general and Ethereum technology in particular.

4 InBlock Roles and Operation

InBlock is implemented as a set of Smart Contracts in Ethereum. InBlock runs autonomously, i.e., without humans involved in its daily operation. In this section we present an overview of how the InBlock works and then we describe the different roles involved in its operation.

4.1 Overview of InBlock's operation

InBlock is configured with a block of globally routable IPv6 addresses to allocate. This is done at the moment of the deployment of the InBlock smart contract and cannot be modified afterwards. Once deployed, InBlocks runs autonomously, processing address allocation requests received and managing the available address pool. Additional InBlock instances can be deployed with different address blocks to allocate, so that each instance independently manages a mutually disjoint block of addresses.

When an entity (called an *InBlock LIR*) wants to obtain an address allocation from the InBlock, it uses its Ethereum account to perform a request. The request is in the form of a blockchain transaction that transfers an allocation fee (paid in Ether) to InBlock. InBlock verifies that the transaction is valid and that the fee has been correctly transferred.

All the prefixes allocated by an InBlock instance have the same size. However, InBlock supports *aggregatable allocations*, i.e., a requesting party may ask for multiple prefix allocations that can be aggregated in a larger prefix.

The allocation fee is significantly larger than the actual costs of the InBlock operation. As we present later on, the operational cost (i.e., the miner's fees to perform the transactions required to annotate the allocation in the blockchain) is in the order of a few US$, while the allocation fee is expected to be similar to the cost of an allocation in the RIR system, around several hundreds of US$. The reason for this is that the fee serves as a mechanism to deter address stockpiling and other wasteful practices. The reader is referred to [19] for a thorough discussion on prefix allocation sizes and fees.

Upon reception of the transaction, the InBlock code goes through its associated state (stored in the blockchain) and finds an address prefix that is not currently allocated. Once an available prefix is found, InBlock associates the prefix with the identity of the requester, its Ethereum identity. This allocation information is recorded in the blockchain.

Allocations have a predefined lifetime. The holder of the resources can renew the allocation by making a new transaction in which it transfers the yearly fee to the InBlock before the expiration date. If this happens, InBlock extends the lifetime of the allocation for another one-year period. Each IPv6 allocation record stored in the blockchain contains the information about the allocated prefix, the holder's Ethereum Identity, the expiration date and a pointer to additional information, such as the holder's contact or routing policy information.

Once an InBlock LIR has obtained an address allocation from the InBlock, it can suballocate part of this prefix to other entities (called *InBlock End Users*). In this way, the InBlock mimics the RIR allocation chain, from InBlock to InBlock LIRs and to InBlock End Users. All entities that have obtained an InBlock prefix can also include ROA information for it, enabling route origin validation by third parties based on the information stored in the InBlock.

4.2 InBlock roles

We next describe the different roles defined for InBlock and their associated operations.

InBlock Manager. The InBlock manager inserts the InBlock code into the Ethereum blockchain by means of a transaction. Then, it issues another transaction to activate the InBlock instance, in which it defines the address pool available for the InBlock to allocate, the size of the allocations and the allocation fee in dollars. The InBlock manager also manages the Ethereum account that is used to receive the allocation fees, so that it can transfer the funds received through the InBlock account to any destination account.

InBlock does not provide functions to allow the manager to create or modify prefix allocations once the contract has been activated. The InBlock manager cannot revoke existent allocations nor it can prevent any party to issue a transaction to obtain a new allocation or to renew an existent one. The InBlock manager cannot modify the allocation fees neither. The motivation for this is to prevent the manager from arbitrarily raising the prefix allocation fees in order to impede particular entities to obtain an address allocation. The allocation fee is automatically updated by the InBlock contract according to the world's gross domestic product, to account for global inflation. All these restrictions serve the purpose of limiting the power of the InBlock manager over the system, drastically reducing the centralisation of power in the allocation system.

InBlock LIR. An InBlock LIR is an entity that obtains a prefix allocation directly from the InBlock. To do so, the InBlock LIR issues a transaction that transfers the allocation fee to the InBlock. While the prefix allocation fee is defined in a fiat currency (e.g., US$), the Ethereum transactions transfer Ether. Historically, the exchange rate between these two has varied greatly in time. So, the InBlock LIR first determines the amount of Ether that corresponds to the prefix allocation fee. This is done through the use of oracles.[3] The InBlock LIR issues a first transaction by which the InBlock code writes in the blockchain the currency exchange rate information retrieved from the oracles. Then a second transaction is issued by the LIR through which the InBlock code computes the allocation fee (expressed in the fiat currency) and stores it in the blockchain. This fee value is valid only for 24 hours for the requester account. Once the value in Ether of the allocation fee has been determined, the LIR issues the transaction that transfers the said amount of Ether to the InBlock, requesting the allocation of a prefix.

Upon the reception of the transaction, the InBlock code selects a free IPv6 prefix from the InBlock address pool, and associates the prefix to the account of the LIR. If the LIR already has a prefix allocated, it can request that the additional address space allocated is contiguous to previously owned allocations, if such address space is available. The prefix allocation is valid for a year and the LIR has to pay a yearly fee to renew it. If the prefix is not renewed, it returns to the InBlock free address pool for future allocations.

The LIR is entitled to manage its prefix allocations, so that it can include ROAs, contact and routing information associated to its prefixes (or subprefixes) in the blockchain. The InBlock provides functions to enable the LIR to assign sub-prefixes to other accounts (InBlock End Users). These users can autonomously update the contact and routing information for these delegated prefixes. The InBlock LIR also has the capability to revoke a prefix assignment to an InBlock End User.

InBlock End User. We refer to the entities which have received a prefix assignment from InBlock LIRs as *Inblock End Users*. We expect that a common case will be that InBlock LIRs are Internet Service Providers (ISPs) and that the InBlock End Users will be the ISP's customers. In this case, the address assignment from the InBlock LIR to the InBlock End User will be part of the provisioning of the Internet access service from the ISP to the customer. InBlock does provide the means to annotate information about the InBlock End Users in the blockchain though. InBlock End Users are authorized to update the InBlock registries associated with their assigned prefixes with contact and routing information. In particular, InBlock End Users can include Route Origin Authorizations (ROAs) that indicate the numbers of the ASes authorized to originate a BGP route advertisement for the prefix.

InBlock Third Parties. InBlock Third Parties access to the InBlock information stored in the blockchain to validate routes. A Third Party accesses the Ethereum blockchain and configures the InBlock identifier contract as a Trust Anchor for the validation of the information related with the prefixes managed by InBlock, meaning that the Third Party trusts in InBlock for the management of these prefixes.

InBlock Third Party validation is performed through Ethereum calls. The calls encode and enforce the validation rules defined in the InBlock. For example, the InBlock defines a call to verify if a given AS number is allowed to originate a route for a given prefix (i.e., if there is a valid ROA for a *'prefix, AS number'* couple). This call executes the following verifications. First, it verifies the allocation chain to ensure that the prefix belongs to the InBlock address pool and that it has been allocated to a LIR and that the LIR has assigned it to an End User. Once the allocation chain is verified, it obtains the ROA information from the blockchain and verifies that it contains the requested AS number.

Although these checks could be left for implementation to interested parties, we believe that the provision in the InBlock itself of the code that performs all the intended checks provides an additional level of security and compliance. The validation of the AS information according to the assignment, lifespan and delegation rules encoded in the InBlock extends the 'code-is-law' motto from the state creation to its validation. Besides, users of these calls do not need to be aware of the inner InBlock machinery to make use of its information.

4.3 InBlock security analysis

In this section, we analyse different attacks that can be performed against InBlock. InBlock does not propose a new security method. Instead, InBlock relies on existent security methods, i.e., the Ethereum blockchain, with well-known security properties, to provide functions currently supplied by the RPKI. So, in order to analyze InBlock from a security perspective, we follow the attack analysis framework presented in Section 2.3 for the RPKI and compare it to the security properties of Ethereum established in the literature. As described earlier, there are six types of adverse actions that can be executed against a secure IP address attestation, namely, *Deletion*, *Suppression*, *Corruption*, *Modification*, *Revocation* and *Injection*.

In Table 1 we compare the means required to execute the different types of adverse actions against the RPKI and the InBlock. The fundamental difference is that in the case of the RPKI all these actions can be performed by a parent CA, while this is not possible in the InBlock, as InBlock does not rely on a hierarchical structure. Avoiding the potential errors, abuses and misuses incurred by a parent CA is the main design goal of InBlock. In addition, regarding the first three types of actions, the difference is that blockchain

3. Oracles are third party services that write in the blockchain data from the external world.

information publication is not limited to a reduced set of publication points, but all blockchain nodes propagate the blockchain information. We next present in detail the different attacks against the InBlock.

Deletion: In the context of InBlock, we define the *Deletion adverse action* as the removal of information contained in a confirmed block.[4] Removal of information that was included in an unconfirmed block is considered a *Suppression adverse action* and discussed in the next paragraph. In InBlock, because of the very nature of the underlying blockchain technology, the Deletion attack is essentially unfeasible without affecting the whole blockchain. Once the data is stored in a confirmed block in the blockchain, it is immutable.[5] In order for the data to be actually removed, all Ethereum full nodes (from which the blockchain can be retrieved) must be compromised. At the time of this writing, there are 7,530 active full nodes in the Ethereum blockchain.[6] An InBlock third party can connect to any of these to obtain a copy of the InBlock data. In the case of the RPKI, there are 14 repositories to retrieve the RPKI data from.[7] While each RPKI item is associated to a single authoritative publication point, other repositories can store copies [1].

Suppression: This action can be achieved if the attacker controls at least 51% of the mining power of Ethereum [13]. If this the case, the attacker can prevent new transactions to be included in the blockchain by using its hashing power to mine new blocks on top of blocks that do not contain the victim's transactions and discard those that do. Currently, the hash rate of the Ethereum network is 176TH/s.[8] The cost of acquiring the computational power to perform a 51% attack against the Ethereum network is estimated as US$ 136,982 per hour[9], and the effort should be kept for the time during which the information is to be suppressed.

Corruption: Because every Ethereum full node validates all the transactions included in all the blocks before propagating the new block, performing this action would require to compromise all full nodes in the blockchain to publish invalid information, so the analysis is similar to the one for the Deletion attack presented earlier. Note that the corrupted transaction is stored in the blockchain, and this will become apparent to any third party auditing it.

Modification: To add a new transaction that modifies the information of an attestation contained in the InBlock, the attacker needs to compromise the private key of the affected LIR. The effort to compromise the private key of the holder of the resources is equal in the InBlock and the RPKI, the difference being that in the case of the RPKI, it is also subject to the vulnerabilities involving the parent CA.

Revocation: To revoke an existing attestation in the InBlock, the attacker must compromise the LIR's private key. The vulnerabilities of the parent CA also affect the RPKI in this case.

Injection: Similarly to the two previous attacks, this is feasible only by compromising the LIR's private key. The vulnerabilities of the parent CA also affect the RPKI in this case.

4. In Ethereum, a block is considered confirmed after 12 additional blocks have been added to the blockchain

5. The modification the data stored in the blockchain requires a *fork* of the blockchain. Forking is a public event that forces all Ethereum nodes to actively position themselves regarding the adoption of the fork and third parties would act accordingly.

6. https://www.ethernodes.org/sync

7. To obtain the number of different RPKI repositories, we configured an FRR router, https://frrouting.org, to gather the existing RPKI information and we parsed the logs resulting from this operation. The RPKI validator connects to the repositories associated to the five trust anchors (AFRINIC, APNIC, ARIN, LACNIC, RIPE), and retrieves the information stored and pointers to other repositories. The total number of repositories observed by January 2020 is 14.

8. https://www.crypto51.app/

9. https://www.crypto51.app/

Adverse action	RPKI	InBlock LIR
Deletion	Pub. Point, Parent CA	Ethereum nodes
Suppression	Pub. Point, Parent CA	51% attack
Corruption	Pub. Point, Parent CA	Ethereum nodes
Modification	Priv. Key, Parent CA	Priv. Key
Revocation	Priv. Key, Parent CA	Priv. Key
Injection	Priv. Key, Parent CA	Priv. Key

TABLE 1: Comparison of adverse actions against the RPKI and the InBlock

5 InBlock Smart Contract Implementation

We have implemented InBlock as a set of smart contracts (a main smart contract and several utilities implemented in different contracts for convenience) written in Solidity, and deployed it in Ethereum. We next describe different relevant aspects of our implementation.

5.1 Activating InBlock

To activate an InBlock Smart Contract previously inserted in the Ethereum blockchain, the InBlock Manager issues an `activateInBlock` transaction. This transaction defines the InBlock pool of prefixes, in the form of a (large) prefix, the length of the prefixes to allocate, and the prefix allocation fee. The fee is set in a fiat currency and its value is updated yearly by the InBlock according to the variation of the world gross domestic product, GDP, measured over the last 10-year period, i.e., 3.3% for 2008/2018.

5.2 Allocating prefixes

Any entity can request a prefix from InBlock, thus becoming an InBlock LIR. The prefix allocation process is performed in two steps: a first phase which comprises two transactions and one call to determine the value in Ether of the prefix allocation fee, and a second phase to actually allocate the prefix to the LIR, accomplished in one transaction.

InBlock relies on several third parties, *oracles*, for obtaining the conversion rate between the fiat currency and Ether. An oracle is a service provided by a third party that annotates in the blockchain a particular external world value. Once ensured that the information was inserted by the trusted oracle, this value can be used by a smart contract. *Provable* [20] is a solution to provide a secure connection between Ethereum smart contracts and the external world. In order to avoid dependency with a single oracle, InBlock uses several (three) different oracles providing currency conversion. The risk of individual oracles being compromised is reduced by removing outlier values: if the three oracles are available, the median value is returned; with two oracles, the mean is used. If the all the oracles go offline, InBlock uses a default value set in deployment time.

The `getOracleCurrencyConversion` transaction is issued by the LIR and triggers the InBlock's query to the oracles to obtain the exchange rate. Once the exchange rate is retrieved, the LIR issues the `computeAllocationCost` transaction to determine the allocation fee in Ether. The calculated fee is valid for 24 hours.

With the fee value, the LIR can select one of two different types of transactions to obtain a prefix allocation, namely `getAllocation` and `getSequentialAllocation`. The `getAllocation` transaction allocates a new prefix without considering previous allocations made by the InBlock to the requesting LIR. This is used normally when the requesting LIR has not previously received any prefix from the InBlock. The `getSequentialAllocation` function is used to request the allocation of a new prefix that is aggregatable[10] to an InBlock prefix previously allocated to the requesting LIR.

The `getAllocation` transaction implements the *sparse allocation* algorithm [21] to perform initial prefix allocations as separated to each other as possible. This algorithm maximizes the chances of obtaining aggregatable prefix allocations for all requesting LIRs. The sparse allocation algorithm starts by dividing the InBlock pool in halves, and allocates the first prefix at the beginning of the first half. The next request receives the prefix at the beginning of the second half. Then, the two halves are further divided equally, and the algorithm is applied recursively to the resulting blocks. InBlock checks that the prefixes selected by the algorithm have not been previously allocated before by a sequential request. If this is the case, the next prefix according to the sparse allocation algorithm is selected.

The request for sequential address allocation contiguous to a previously allocated prefix is implemented through the `getSequentialAllocation` transaction, that succeeds if the requester already holds a prefix, and the aggregatable prefix is available.

When a LIR issues a transaction requesting a prefix, the InBlock smart contract allocates storage for the prefix information record, which includes the Ethereum account identifier of the LIR, the time at which the prefix has been allocated (or renewed), the ROA(s), and the additional routing and policy information.

Any allocated prefix needs to be renewed before the expiration date. For this purpose, InBlock provides the `renewAllocation` transaction.

5.3 Prefix Delegation

A LIR can assign more-specific prefixes of the allocation it holds to InBlock End Users, by means of the `delegatePrefix` transaction. This transaction is only valid if the prefix to assign (or any part of it) has not been already assigned. A LIR can revoke a prefix assignment through the `revokeDelegatedPrefix` transaction. A LIR delegating the prefixes can revoke any assignment, as opposed to the InBlock which cannot revoke the allocations made to the LIRs.

10. Two prefixes are aggregatable if they can be expressed as a single (larger) prefix. This can only be done if the two prefixes are subsequent one another. Aggregation is considered beneficial for the global routing system scalability.

5.4 Recovering prefixes from expired allocations

The InBlock Manager discovers expired allocations with the `getIDsPrefixExpired` call. Then, through the `recoverExpiredAllocation` transaction, it inserts in the blockchain the list of prefixes discovered, so that they are eligible for next allocations.

The InBlock Manager is expected to asynchronously run this process for recovering expired allocations and return the resulting address space to the available address pool.

Note that the expiration of allocations also serves to fix a well-known issue with storing tokens in blockchains: if the holder of the resource loses the private key that secures the allocation, the resource is lost. It is estimated that 20% of existing Bitcoins are lost because of this [22]. In the case of InBlock, prefixes for which the holder's private key is lost are not renewed and can be returned to the InBlock pool.

5.5 ROAs and additional information

The current holder of a prefix, an InBlock LIR or an InBlock End User, can register in the InBlock the AS allowed to originate a BGP advertisement for the prefix by means of a `setROA` transaction. The `setROA` function can also be used to update or delete an existing ROA.

In addition, the holder of a prefix can perform a `setPolicyURI` transaction to include a link to an external URI where contact information or routing policy for a prefix is registered. This information is expected to be defined using RPSL [23], the standard routing policy language used in the Internet Routing Registries. The motivation for using an URI instead of including the whole policy information itself directly in the blockchain is to reduce the cost of the transaction to update the routing policy, which depends on the amount of information stored. The information stored by the `setPolicyURI` transaction in the blockchain includes a hash of the routing policy description, proving that the policy information accessible through the URI is authentic (generated by the holder of the prefix in InBlock), integral (unmodified), and up-to-date.

5.6 Third-party information retrieval

Third parties such as a router can access the blockchain to use the information stored in the InBlock to perform BGP route origin validation. To do so, the validating router configures the InBlock contract identifier as a Trust Anchor for the validation of the prefixes managed by the InBlock.

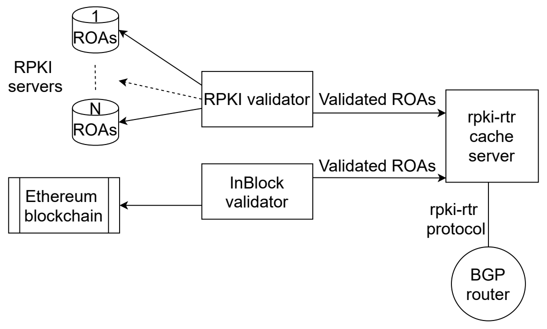

Fig. 1: Combined validation.

InBlock information retrieval can be implemented as *calls*. The `isPrefixInUse` call is used to determine if a prefix has been allocated or assigned to an End User. The get `getROA` call returns the ROA for a prefix and the `getPolicyURI` call gets the URI in which the routing policy corresponding to a prefix is stored.

The ROA information retrieved from the InBlock can be used to configure filter in a validating router. This can be implemented by proper modification of the RPKI-to-router (`rpki-rtr`) architecture [24]. In the `rpki-rtr` architecture routers access through a standard protocol to a trusted cache server to acquire prefix origin data. The integration of InBlock into this architecture requires the modification of the software at the `rpki-rtr` server to retrieve, validate and integrate InBlock's information into the prefix-to-origin AS state conveyed to the routers.

InBlock and RPKI route origin validation can coexist. A validating router can use the RPKI information to validate the origin of the routes of a set of prefixes and use the InBlock to validate another (disjoint) set of prefixes. This can be achieved by feeding both streams of information to the `rpki-rtr` server as shown in Figure 1.

To ensure that the validation information is fresh, we need to be able to update the InBlock state frequently. In order to efficiently support periodic updates of the InBlock information for all the allocated prefixes, we leverage on the blockchain monitoring capabilities, the ability of blockchain nodes to be configured to track the events associated to a particular contract and return those annotated after a given block identifier. In the case of the RPKI, RFC8182 [25] indicates that the information received by a repository must be published within a minute, but limits the polling frequency of the end users to a minute. As a result, information could be available to the routers in the order of few minutes (subject to the ability of the repositories to support requests from all validating entities at this rate). As we will show in Section 6, the blockchain monitoring capabilities can easily achieve refresh rates in the order of few minutes.

5.7 Key rollover

To enable the update of the cryptographic information granting access to InBlock resources, InBlock provides functions to transfer some of the resources to different Ethereum accounts in a way which resembles key rollover procedures common in network resource management.

The InBlock Manager can perform a `transferInBlockControl` transaction to transfer the ownership of the main InBlock contract to another Ethereum account. After executing this transaction, the holder of the new account will receive the fees resulting from allocations and renewals and it will be able to execute all the InBlock manager operations.

A LIR holding a prefix can also transfer its control to another account by means of the `transferAllocatedPrefixControl` transaction. InBlock also provides tools for the transfer of delegated prefixes. In this case, it is the LIR who indicates to which account the prefix is transferred to, through the `transferDelegatedPrefixControl` function.

5.8 Emergency stop

The provision of code implementing an emergency stop is a security mechanism that allows the InBlock manager to block the execution of some selected functions. This provides a countermeasure to limit the damages caused by a bug or a security issue [26], [27].

In the case of InBlock, the emergency stop function `prefixAllocationStop` suspends temporarily the `getAllocation` and `getSequentialAllocation` functions, so that InBlock cannot perform further allocations. The suspended functions can be reactivated by executing `prefixAllocationResume`.

This stop function does not affect the operation with prefixes already allocated. This means that previously allocated prefixes can still be renewed, delegated, ROAs can be set, etc., even if the aforementioned stop functions are executed, so that the InBlock manager cannot disrupt the system.

5.9 Execution in light nodes

Ethereum smart contracts are executed by Ethereum *nodes*. While full nodes download and verify every block in the blockchain, it is also possible to operate with the blockchain through a *light client*, with similar security as full nodes, but much lower memory and computing requirements [28]. A light client connects to full nodes to retrieve the most recent block headers. The light client leverages in the *Merkle tree* structure to identify the blocks containing the transactions related with the InBlock smart contract. Then, it requests these particular blocks and validates them. Note that the Merkle tree information of the last block can be used to determine the integrity of the whole blockchain or just a stream of transactions associated to a single account. In this way, the light client can securely obtain all the information related with InBlock, without the need to retrieve and validate the whole blockchain.

6 InBlock Experiments

We next present the results of several experiments targeted to evaluate the computation/storage and monetary costs involved in InBlock operation as well as the delays of the different operations involved.

The implementation of InBlock as well as the scripts used in the experiments are available at https://github.com/steang91/InBlock_Code/tree/IPv6.

6.1 Local experiments

Goal of the experiments: experimentally measure the computational cost of different functions implemented in InBlock. The computational cost of executing a function in Ethereum is expressed in gas. Because the coast in Ether associated to a transaction depends both in the gas and the gas prize set for that transaction, and Ether can be converted into a fiat currency such as US$, we also express the measured cost of the different functions in US$.

Experimental setup: The cost in gas of executing a function solely depends on the operations included in the function and it is constant for all the nodes in Ethereum. For this reason, it is enough to measure the gas incurred for the different functions on a local node and the result

	/20				/8			
Operation	**Min Gas**	**Max Gas**	**Mean Gas**	**Mean US$**	**Min Gas**	**Max Gas**	**Mean Gas**	**Mean US$**
getAllocation	209223	286518	218126	1.09	253875	272723	262512	1.31
getSequentialAllocation	237869	246119	242925	1.21	288196	296446	292181	1.46
renewAllocation	37739	37803	37786	0.19	37803	37867	37867	0.19
recoverExpiredAllocation	53451	60951	53482	0.27	53483	61015	53522	0.27
getOracleCurrencyConversion	120421	141568	130994	0.65	120421	141568	130994	0.65
computeAllocationCost	34703	34703	34703	0.17	34703	34703	34703	0.17
delegatePrefix	436466	1383448	904203	4.52	486670	1433652	954407	4.77
setRoa	45208	45272	45255	0.23	45272	45336	45336	0.23
setPolicyURI	88939	89003	88986	0.44	89003	89067	89067	0.44

TABLE 2: Gas and US$ transaction cost for different InBlock transactions (gas price 19 GWei, 1 Ether = US$ 263.43)

will concur with the gas in any node in the real Ethereum network. Thus, to estimate the cost in gas required by InBlock functions, we use the blockchain emulator Ganache CLI[11] within its default configuration. Ganache CLI is part of the Truffle suite of Ethereum development tools. It offers a fast and customizable blockchain emulator. It uses `ethereumjs` to simulate full client behaviour and allows to make transactions and calls to the blockchain without the overheads of running an actual Ethereum node. The use of Ganache CLI offers the following advantages: *i)* transactions are instantly mined, *ii)* no transaction costs, *iii)* accounts can be re-cycled, reset and instantiated with a fixed amount of Ether (no need for faucets or mining), *iv)* mining speed and gas price and can be modified. The specific details of the node hardware are irrelevant, since the experiment results does not depend on it.

Experimental methodology: We deploy InBlock in the local testnet and we execute the different functions described in 5. The amount of gas required to complete a transaction depends on the associated computation and storage needs. For some functions, these needs vary with the parameters involved on each particular execution, e.g., slight differences may arise from the number of bits of the prefix that is allocated next by a `getAllocation` function. For those functions, we execute the different InBlock functions varying the parameters and we calculate the mean, maximum and minimum gas used.

To provide a rough estimation of the cost in US$ for each transaction, we consider the daily mean gas price offered at the Ethereum network in the three-month period from December 2018 to February 2019 [29]. The mean gas value of this series is 19 GWei[12]. For the calculation of the cost in US$ we use an exchange rate of 1 Ether = US$ 263.43.

Experiments and results: We perform two rounds of experiments using two different InBlock root prefix lengths, /20 and /8. In both cases, the length of the allocatable prefix is a /32. The first case, /20, contains 4,096 allocatable prefixes, which we deem a reasonable size for an InBlock instance. The second case represents an upper bound for the size of an InBlock root prefix, as it covers the total amount of IPv6 addresses currently reserved for global unicast addresses (a /8 prefix). We next describe the results for each of the tested functions.

`deploy`: This function is executed by the InBlock manager once, to deploy the InBlock in the blockchain. The cost in gas to perform it is constant, and does not depend on InBlock root prefix size. We verify this by executing the function a reduced number of times, to obtain a constant result. It is not possible to deploy the whole InBlock contract in Ethereum through a single transaction because it would exceed the maximum gas limit per transaction, set to 6.7 MGAS. So, the code is divided into modules that are deployed separately. The total gas required for deploying InBlock is 12.18 MGAS, with an estimated cost of US$ 60.96. This is the most expensive operation, as it implies the storage of a large amount of bytes.

`activateInBlock`: This function is executed by the InBlock manager to activate the InBlock once deployed. The cost in gas of this function is constant, 154 KGAS (US$ 0.77).

`getSequentialAllocation` and **`getAllocation`**: These functions are used by an InBlock LIR to request new prefixes, either aggregatable to a previous allocation or sparse. As different executions of these functions can require different amounts gas, we define the following sequence of transactions to evaluate their cost: execute 100 times the `getAllocation` transaction (sparse allocation algorithm), and then perform one `getSequentialAllocation`. This sequence is repeated until 4,096 prefixes are allocated. Results are presented in Table 2.

`renewAllocation`, **`delegatePrefix`**, **`setRoa`** and **`setPolicyURI`**: After running the previous test involving the prefix allocations operations, for each round of tests, the InBlock has 4,096 prefixes allocated. We execute the four aforementioned operations, in this case once per prefix and measure the required gas for them. Results are presented in Table 2. The highest gas cost results from the delegation of prefixes, due to the costs of managing the tree that stores the prefix delegation information. This operation requires variable computation effort depending on the position of the prefix to allocate within the tree. The costs for managing ROAs and policies, which are expected to be the most frequent operations, are very low, well below US$ 1.

`recoverExpiredAllocation`. Using the state from the previous test, we artificially force to expire 100 prefix allocations and then we run `recoverExpiredAllocation`. We repeat the procedure 10 times. The results are shown in Table 2.

6.2 Mainet measurements

Goals of the experiment: The primary goal of the experiment is to measure the time required for executing the different InBlock transactions in the Ethereum blockchain. More precisely, we measure two delays: the time to write

11. https://www.trufflesuite.com/ganache
12. The maximum value observed is 370 GWei, reported for Feb 19th 2019; the second highest daily cost is more than ten times lower

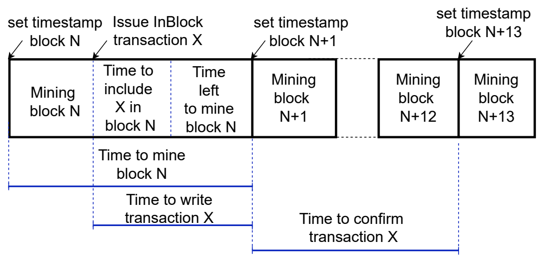

Fig. 2: Blockchain transaction times.

a transaction in the blockchain and the time to confirm a transaction. The time to write a transaction is the time between the transaction is issued and the time that the transaction appears in a block in the blockchain. The time to confirm a transaction is the time to write a transaction plus the time it takes to mine additional 12 new blocks on top of the block containing the transaction [30]. The confirmation time represents the recommended time to wait to have reasonable guarantees that the transaction cannot be reverted.

A secondary goal of the experiment is to validate the results obtained in the local tests and to measure the cost of `getOracleCurencyConversion` and `computeAllocationCost` that, because they involve external third parties, cannot be measured in a local testbed.

Experimental setup: We deploy the InBlock contracts into the live Ethereum blockchain (Mainnet). To this end, we use `geth`, an implementation of an Ethereum node in the Go programming language. `geth` also serves as a console for typing commands and executing specific functions. Using `geth`, we locally setup an Ethereum light node and connect it to the Mainnet. Then, we create and fund an EOA account, to be able to deploy InBlock and interact with it to run transactions. This setup also allows us to measure both the time it takes to execute transactions in the real Ethereum blockchain, and the cost (in gas, Ether and US$) of doing so.

Experimental methodology: To measure the transaction write time, we issue a transaction and observe the timestamps of the new block published in the blockchain containing our transaction. However, simply computing the difference between the timestamp of the block and the time we requested the transaction poses some issues. As illustrated in Figure 2, the timestamp included in a given block reflects the time when the miner started mining the block. Miners can include transactions in the block that were received after they started mining it. Thus, it is possible that our transaction appears in a block with a timestamp showing a value prior to the time the transaction was generated. In order to obtain an upper bound for the time when the block was actually mined, we use the timestamp of the subsequent block in the blockchain. This is an upper bound, under the assumption that miners start mining the next block as soon as the previous block was mined. So, our time measure is obtained by subtracting the local time when the transaction was issued from the timestamp of the block after the one where the transaction was registered.

To measure the time to confirm a transaction, the usual rule is to wait until 12 blocks are build upon the one to trust. We then measure the time elapsed until 12 additional blocks are mined and we present it along the results.

Operation	Mean time to write	Max time to write	Mean time to confirm	Max time to confirm
getAllocation	30	170	173	316
getSequential Allocation	35	149	213	433
renewAllocation	33	139	204	355
recoverExpired Allocation	37	182	187	306
getOracle CurrencyConversion	31	147	202	360
computeAllocation-Cost	49	99	262	387
delegatePrefix	28	153	183	365
setRoA	29	97	189	292
setPolicyURI	25	84	186	293

TABLE 3: Time in seconds to write and to confirm different InBlock transactions (gas price 19 GWei)

During these experiments, we also measured the gas actually used by the different transactions.

Experiments and results: We deploy InBlock on the Ethereum Mainnet blockchain, and activate it with a /20 InBlock root prefix, being able to allocate 4,096 prefixes. We perform 50 transactions for each of the InBlock functions tested and we measure their transaction write and confirmation times, as well as the gas used. Experiments were performed in March 2019. The experiment spanned over 10 hours, and each transaction is mined in a different block. We set the gas price to 19 GWei for all the experiments. In Table 3 we show the measured time for different transactions.

As the gas price is the same for all operations, the variations depend on the particular conditions of the blockchain when a transaction is issued. In general, we observe that the time to write a transaction is always below 3 minutes and the time to confirm an operation is below 440 seconds.

The overall time required to request a block can be estimated as the sum of `getOracleCurrencyConversion`, `computeAllocationCost` and the allocation itself (either `getAllocation`, `getSequentialAllocation`), so it is in the order of 10 minutes, as each operation should not be initiated until the previous one has been confirmed. However, the most frequent operations are expected to be policy changes, which result from the execution of individual functions, `setRoA` and `setPolicyURI`, with mean confirmation times around 3 minutes.

Additionally, using these experiments we measured the gas used by the `getOracleCurencyConversion` and the `computeAllocationCost`. These functions rely on external third parties, which is why we were unable to measure them in the local tests. Results are presented in Table 2. The mean of the sum of the fees the Oracles request for providing their services is 0.012 Ether (US$ 3.16 according to the change rates considered). The total cost for obtaining the price of the allocation fee is the sum of the Oracle fees plus the cost of the two transactions `getOracleCurencyConversion` and `computeAllocationCost`.

Mechanism name	Blockchain technology	Resource bootstrap	Initial allocation for LIR	Managed resources	Implementation
Internet Blockchain [31]	New blockchain, Bitcoin-like	Genesis block includes existing allocations, RIRs register new allocations	Centralized (through RIR)	IPv4 and IPv6 addresses, AS numbers, DNS and BGP routes	NA, only general architecture is defined
IPchain [32]	New blockchain, Proof of Stake	RIRs allocate blocks to themselves in genesis block, then register futher updates	Centralized (through RIR)	IP addresses, ROA	Open source
BGPcoin [33]	Ethereum	IANA, RIRs, NIRs use their accounts to include their resources	Centralized (through RIR)	IP addresses, AS numbers, ROA, adjacent ASes	Open source (smart contracts)
Sfyrakis, Kotronis [34]	New blockchain, Proof of Work	Genesis block includes existing allocations, RIRs register new allocations	Centralized (through RIR)	IP address, AS numbers, ROA, adjacent ASes	Open source
InBlock	Ethereum	Genesis block includes IPv6 address block to use in InBlock	Decentralized	Specific subset of IPv6 addresses, ROA	Open source (smart contracts)

TABLE 4: Comparison among blockchain proposals for management of Internet resources

7 Related Work

Several recent proposals (Internet Blockchain [31], IPchain [32], BGPcoin [33], [34]) specifically address the application of blockchain technology to the management of Internet addresses and routing resources. Table 4 describes the main characteristics of each of these proposals, along with its comparison with the InBlock implementation.

Regarding to the underlying blockchain used, some works (including InBlock) rely on an existing blockchain (Ethereum), while others propose to create a new one for this purpose. Internet Blockchain suggests the use of a Bitcoin-like blockchain, but no details are provided about the specifics of its design, implementation and deployment. IPchain is based on a PoS consensus mechanism with the addresses being the asset defining the stake balance. The main concern with such approach is that PoS results in major stakeholders having more chances to mine new blocks. While this may work well for IPv4, for IPv6, it means that the current authority of the global IPv6 address pool (IANA) would certainly control the majority of the stakes and probably the blockchain, defeating the goal of preventing centralized control of the IP address allocation system. Sfirakis and Kotronis [34] propose using a new PoW blockchain, but they do not discuss the risks of managing valuable resources as IP addresses in a small mining network.

In all cases, except for InBlock, the resources (IP addresses, AS numbers, etc.) are initially allocated to LIRs by IANA/RIRs. We name this approach as centralized, as it depends on the central authorities currently defined for Internet resource management. For the blockchains specifically created for this purpose, existing allocations (e.g., obtained from current registries or the RPKI) can be written in the genesis block. For all other proposals except for InBlock, initial allocations to LIRs have to be performed by IANA/RIRs. For InBlock, IANA/RIRs delegate the IPv6 pool to InBlock, but the assignment to LIRs is decentralized. Because all proposals other than InBlock rely on IANA/RIRs to annotate initial allocations to LIRs, the mechanisms can be used for both IPv4 and IPv6. The decentralized mechanism defined for InBlock requires abundance of resources to allocate, so that it is currently defined for IPv6; it could also be easily extended to AS numbers, but managing IPv4 addresses would need a different approach.

All the proposals facilitate the transference of the managed resources (or a subset of them) to other LIRs, without the participation of the RIRs or other higher-level entities. They also enable secure route origin validation by registering the association of the managed prefixes and their corresponding AS numbers (ROA). In fact, all other proposals but InBlock aim to replace the RPKI for this purpose. As InBlock only manages a subset of the (IPv6) address space, it does not aim to replace the RPKI, but to complement it for cases where decentralisation is desired (besides, InBlock could also coexist with any of this technologies to replace the RPKI for RIR-assigned prefixes).

Additional information such as the list of ASes that are adjacent to a given one can be included in the blockchain to enable limited path validation, such as checking that the AS pairs observed in a BGP route match with the topology annotated in the blockchain ([33] for the last hop, [31], [34] for the whole the path), although concerns are raised about the ability to perform this in practice [31].

Many papers [35], [36] show the design, implementation and measurement of the cost of systems based on Ethereum smart contracts. The main parameter measured in all these system papers is the gas spent in the transactions involved, as it is a key metric to prove its effectively usability in terms of economic cost. In Section 6 we use the same measurement methodology to provide the amount of gas consumed by each transaction and, after selecting a gas price, their actual cost in dollars. In addition, we include the measurement of mining and confirmation times of the InBlock operations that require writing into the blockchain.

8 Conclusions

In this paper we have presented the design and implementation of InBlock, a decentralized autonomous organization that is capable of performing the IPv6 global registry functions without human intervention. InBlock implements an alternative trust model to the hierarchical one currently provided by the RPKI. In particular, InBlock is not under control of any single entity, so no single entity can prevent another to obtain IP address allocations. InBlock provides quite strong privacy guarantees and censorship resistance.

Blockchain technology represents an opportunity to improve the management of sensible Internet resources such as IP addresses. Either as an autonomous decentralized organization, as we discuss in this paper, or as a system supervised by the current top-level resource holders, blockchain has the potential to operate at low cost, fast update times and high ledger visibility without sacrificing security.

References

[1] M. Lepinski and S. Kent. An Infrastructure to Support Secure Internet Routing. IETF RFC 6480, Feb 2012.
[2] M. Lepinski and K. Sriram. BGPsec Protocol Specification. RFC 8205, September 2017.
[3] A. Toonk. Today's BGP leak in Brazil. https://bgpmon.net/todays-bgp-leak-in-brazil/, Oct 2017. Accessed: 2020-02-18.
[4] Cisco. All Events for BGP Stream. https://bgpstream.com/. Accessed: 2020-02-18.
[5] Y. Gilad, A. Cohen, A. Herzberg, M. Schapira, and H. Shulman. Are We There Yet? On RPKI's Deployment and Security. In *NDSS 2017*. The Internet Society, 2017.
[6] L. Reyzin E. Heilman, D. Cooper and S. Goldberg. From the Consent of the Routed: Improving the Transparency of the RPKI. *Proceedings of the 2014 ACM conference on SIGCOMM*, Aug 2014.
[7] B. Kuerbis and M. Mueller. Negotiating a new governance hierarchy: An analysis of the conflicting incentives to secure internet routing. *Communications and Strategies*, 81, 03 2011.
[8] D. Cooper, E. Heilman, K. Brogle, L. Reyzin, and S. Goldberg. On the risk of misbehaving RPKI authorities. In *HotNets-XII*, 2013.
[9] J. Arkko, B. Trammell, M. Nottingham, C. Huitema, M. Thomson, J. Tantsura, and N. ten Oever. Considerations on Internet Consolidation and the Internet Architecture. Internet-Draft draft-arkko-iab-internet-consolidation-02, IETF, July 2019.
[10] Internet Society. Consolidation in the Internet Economy. ISOC Global Internet Report, ISOC, 2019.
[11] J. Arkko. Centralised Architectures in Internet Infrastructure. Internet-Draft draft-arkko-arch-infrastructure-centralisation-00, IETF Secretariat, November 2019.
[12] RIPE. IPv6 Address Allocation and Assignment Policy. https://www.ripe.net/publications/docs/ripe-699#minimum_allocation, May 2018.
[13] V. Buterin. Ethereum White Paper. https://github.com/ethereum/wiki/wiki/White-Paper, May 2018.
[14] P. Sermpezis, V. Kotronis, P. Gigis, X. Dimitropoulos, D. Cicalese, A. King, and A. Dainotti. ARTEMIS: Neutralizing BGP Hijacking within a Minute. *IEEE/ACM Transactions on Networking*, Oct 2018.
[15] S. Kent and D. Ma. Adverse Actions by a Certification Authority (CA) or Repository Manager in the Resource Public Key Infrastructure (RPKI). RFC 8211, September 2017.
[16] R. Bhadoria and V. Agasti. The paradigms of blockchain technology: Myths, facts & future. *International Journal of Information Systems and Social Change*, 10:1–14, 04 2019.
[17] R. Bhadoria, A. Nimbalkar, and N. Saxena. *On the Role of Blockchain Technology in the Internet of Things*, pages 129–140. Springer, 2020.
[18] R. Bhadoria, Y. Arora, and K. Gautam. *Blockchain Hands on for Developing Genesis Block*, pages 269–278. Springer, 2020.
[19] S. Angieri, A. García-Martínez, B. Liu, Z. Yan, C. Wang, and M. Bagnulo. A Distributed Autonomous Organization for Internet Address Management. *IEEE Transactions on Engineering Management*, 2019.
[20] Provable Documentation. http://docs.provable.xyz/#home. Accessed: 2020-02-6.
[21] P. Wilson, R. Plzak and A. Pawli. IPv6 Address Space Management. *RIPE-343*, Jun 2018.
[22] J. Roberts and N. Rapp. Exclusive: Nearly 4 Million Bitcoins Lost Forever, New Study Says. Fortune. http://fortune.com/2017/11/25/lost-bitcoins/, Jun 2018.
[23] D. Kessens, T. J. Bates, C. Alaettinoglu, D. Meyer, C. Villamizar, M. Terpstra, D. Karrenberg, and E. P. Gerich. Routing Policy Specification Language (RPSL). RFC 2622, June 1999.
[24] R. Bush and R. Austein. The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1. IETF RFC 8210, 2017.
[25] T. Bruijnzeels, O. Muravskiy, B. Weber, Cobenian and R. Austein. The RPKI Repository Delta Protocol (RRDP). IETF RFC 8182, 2017.
[26] A. Bahga and V. Madisetti. *Blockchain Applications: A Hands-On Approach*. VPT, 2017.
[27] M. Wohrer and U. Zdun. Smart contracts: security patterns in the ethereum ecosystem and solidity. In *2018 International Workshop on Blockchain Oriented Software Engineering*, pages 2–8, March 2018.
[28] IPv6 Address Space Management. https://www.parity.io/what-is-a-light-client/. Accessed: 2020-02-4.
[29] Etherscan. Ethereum GasPrice History. https://etherscan.io/chart/gasprice, March 2019.
[30] V. Buterin. On slow and Fast Block Times. https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/, Sep. 2015.
[31] A. Hari and T.V. Lakshman. The Internet Blockchain: A Distributed, Tamper-Resistant Transaction Framework for the Internet. *HotNets-XV*, 2016.
[32] J. Paillisse, A. Rodriguez-Natal, V. Ermagan, F. Maino, L. Vegoda, and A. Cabellos-Aparicio. An analysis of the applicability of blockchain to secure IP addresses allocation, delegation and bindings. Internet-Draft draft-paillisse-sidrops-blockchain-02, IETF, June 2018. Work in Progress.
[33] Q. Xing, B. Wang, and X. Wang. BGPcoin: Blockchain-based internet number resource authority and BGP security solution. *Symmetry*, 10(9):408, 2018.
[34] I. Sfirakis and V. Kotronis. Validating IP prefixes and AS-paths with blockchains. *arXiv preprint arXiv:1906.03172*, 2019.
[35] P. McCorry, S. F. Shahandashti, and F. Hao. A smart contract for boardroom voting with maximum voter privacy. In *Financial Cryptography and Data Security*, pages 357–375. Springer, 2017.
[36] M. Al-Bassam. SCPKI: A Smart Contract-based PKI and Identity System. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pages 35–40, 2017.

Alberto García-Martínez received a Ph.D. in telecommunications in 1999. In 1998 he joined Universidad Carlos III of Madrid (UC3M), where he has been an associate professor since 2001. His main interest areas are interdomain routing, transport protocols, network security and blockchain technology. He has published more than 50 papers in technical journals, magazines, and conferences, and has also co-authored three RFCs.

Stefano Angieri received a computer science bachelor degree in 2014 and a master degree in 2018 at Università degli studi Federico II di Napoli with a master thesis on blockchain technology. In 2018 he joined Universidad Carlos III de Madrid (UC3M) as Ph.D student. His main interest area is blockchain technology.

Liu Bingyang received his Ph.D. degree in computer science from Tsinghua University, Beijing, China in 2014. From 2014 to 2016, he was a postdoctoral research associate in the institute for network science and cyberspace in Tsinghua University. He is now a principal research engineer in Huawei network technologies lab. His research interest lies on network architecture, security, protocols and quality of services.

Fei Yang received his Ph.D. degree from Beijing University of Posts and Telecommunication, China, in 2001. He worked on mobile system software development in Samsung from 2001 to 2015. He is currently a principal research engineer in Huawei, working on network technology research.

Marcelo Bagnulo holds a tenured associate professor position at UC3M since 2008. His research interests include Internet architecture and protocols, interdomain routing and security. He has published more than 60 papers in journals and congresses and is the author of 18 RFCs in the IETF. Dr. Bagnulo was a member of the Internet Architecture Board between 2009 and 2011.