Conifer: centrally-managed PKI with blockchain-rooted trust

Yuhao Dong
University of Waterloo
yd2dong@uwaterloo.ca

Woojung Kim
University of Waterloo
w3kim@uwaterloo.ca

Raouf Boutaba
University of Waterloo
rboutaba@uwaterloo.ca

***Abstract*—Secure naming systems, or more narrowly public key infrastructures (PKIs), form the basis of secure communications over insecure networks. All security guarantees against active attackers come from a trustworthy binding between user-facing names, such as domain names, to cryptographic identities, such as public keys.**

By offering a secure, distributed ledger with highly decentralized trust, blockchains such as Bitcoin show promise as the root of trust for naming systems with no central trusted parties. PKIs based upon blockchains, such as Namecoin and Blockstack, have greatly improved security and resilience compared to traditional centralized PKIs. Yet blockchain PKIs tend to significantly sacrifice scalability and flexibility in pursuit of decentralization, hindering large-scale deployability on the Internet.

We propose Conifer, a novel PKI with a architecture based upon CONIKS, a centralized transparency-based PKI, and Catena, a blockchain-agnostic way of embedding a permissioned log, but with a different lookup strategy. In doing so, Conifer achieves decentralized trust with security at least as strong as existing blockchain-based naming systems, yet without sacrificing the flexibility and performance typically found in centralized PKIs. We also present our reference implementation of Conifer, demonstrating how it can easily be integrated into applications. Finally, we use experiments to evaluate the performance of Conifer compared with other naming systems, both centralized and blockchain-based, demonstrating that it incurs only a modest overhead compared to traditional centralized-trust systems, while being far more scalable and performant than purely blockchain-based solutions.

I. Introduction

Nowadays, cryptographic protocols such as TLS, which provide secure communications over insecure networks such as the Internet, have gained extremely pervasive deployment. It would not be an understatement to say that without these secure channels, use of the Internet could not have expanded into fields such as private communication and e-commerce. Yet the security of these protocols rely ultimately on one thing: a secure *public key infrastructure*. Application-level names must be somehow bound to cryptographic identities, such as public keys, in a trustworthy way; otherwise, security against active man-in-the-middle attacks cannot be achieved.

Unfortunately, building a secure PKI has proved to be quite difficult. Naming systems were once conjectured to be subject to a tradeoff known as *Zooko's triangle* [9], which suggests that a naming system cannot simultaneously achieve the three properties of *human-meaningful* (arbitrary names chosen by users), *decentralized* (no single party can subvert the system), and *secure* (does not trust the network). Traditionally, PKIs such as those used in TLS and S/MIME achieve security and human-meaningfulness by introducing trusted third parties — certificate authorities (CAs) or key servers. Though compromised or incompetent trusted parties have repeatedly damaged security [6], [13], Zooko's triangle seemed to imply that there could be no better alternative.

Blockchains, public append-only ledgers that are unforgeable yet fully decentralized, largely demonstrated that Zooko's triangle was after all not true. The first blockchain, Bitcoin [12], was initially conceived only as a cryptocurrency for financial purposes, but soon afterwards a derivative, Namecoin [2], pioneered the idea of building a secure naming system by encoding DNS records inside a blockchain. Namecoin's dedicated blockchain, in additioning to providing a Bitcoin-like cryptocurrency ("namecoins"), also stores a global log of state transitions to provide consensus on the mapping of names to cryptographic identities without needing any trusted third party. Several newer blockchain naming system designs, such as Certcoin [7], follow the same general design of embedding name records directly in a special-purpose cryptocurrency blockchain.

However, Namecoin-like blockchain PKIs bring significant improvements in the area of decentralizing trust, they also face many new challenges. All nodes in the network must synchronize and validate a local copy of the blockchain, so anybody wishing to look up names in a secure fashion must face large, linearly-increasing storage costs. Additionally, without widespread adoption, proof-of-work blockchains such as those of Bitcoin and Namecoin are vulnerable to attacks which nullify their security guarantees. Namecoin, in fact, is known to be subject to a 51% attack due to its small amount of miners [4], an almost complete breakdown in the trust decentralization of a blockchain. Finally, deploying new features to system using a dedicated blockchain like Namecoin is difficult, as blockchain nodes must all agree to run a newer version of the protocol within a short period of time to maintain the distributed consensus.

Newer blockchain-based PKIs do attempt to mitigate these issues, most recently and successfully Blockstack [4]. However, although Blockstack makes it easier to deploy new features and reduces the amount of data that needs to be replicated to all participants by moving most of the data away from the underlying blockchain, it still fails to eliminate the requirement for verifying large amounts of blockchain data, and continues to be much less flexible in enforcing rules for namespaces compared to centralized solutions.

These issues with existing blockchain-based distributed PKIs motivate us to create a new system, Conifer, which builds upon the existing work of CONIKS [11], a centralized transparency-based PKI, and Catena [14], a data structure embedding a permissioned, efficiently queriable log into a public blockchain. Unlike the direct combination of Catena and CONIKS as proposed by the authors of Catena, though, Conifer uses a different lookup algorithm — with changes to CONIKS's data structures to support it — to provide fully proactive security similar to usual blockchain-based naming system, eliminating the need for monitoring a trusted central entity and fully decentralizing trust. This allows us to ultimately anchor trust on a decentralized blockchain, while retaining much of the properties of traditional PKIs including low overhead, policy flexibility, and performance. We believe that compared to existing systems, Conifer makes it significantly more practical to deploy a PKI with fully blockchain-backed security.

II. Background and related work

In this section, we discuss the background to the development of Conifer — in particular, our definition of a secure naming system and previous work on improving upon traditional centralized-trust PKIs.

A. Secure naming systems

It is important to define what exactly we mean by a "secure naming system". Unfortunately, the security goals of different PKIs are often vaguely described, causing confusion on exactly what sort of guarantees a system attempts to give. Thus, instead of referring simply to "security", we discuss two quite separate concepts in this paper: *policy enforcement* and *identity retention*.

Policy enforcement refers to the enforcement of *external* guarantees on bindings in the namespace. For example, in the traditional PKI used in TLS, CAs checking for domain ownership before issuing domain-validated certificates is an example of policy enforcement: "domain ownership" is a concept defined by DNS, a system external to the PKI. Another common example is real-world identity certification, as in TLS's extended validation certificates, where having a name in the naming system certifies claims about real-world facts like business registration.

Identity retention, on the other hand, is a purely *internal*, self-consistency property. We use this term to refer to the following two features:

- Everybody sees the same value bound to a given name (also known as *non-equivocation* [11])
- Values bound to a name cannot be changed without authorization from the name's owner

Violations of identity retention tend to be far more devastating to secure communications than violations of policy enforcement. For example, man-in-the-middle attacks inherently require an adversary to make unauthorized changes to the public key bindings of a name, thus breaking identity retention. On the other hand, failure of policy enforcement is rarely a disaster, unless a system relies solely on policy enforcement to provide its security guarantees, such as in TLS's PKI.

B. Failure of centralized trust

In conventional PKIs, centralized trusted entities, such as CAs, are in charge of enforcing the abovementioned security properties. For example, in the CA-based PKI used in TLS, CAs must check every request for a certificate to execute the policy enforcement that CAs are required to do — if a CA makes a mistake or is compromised, then the PKI's security guarantees immediately break down.

Unfortunately, such collapses of centralized PKI security are not very uncommon. The most high-profile incident arguably happened in 2011 when the DigiNotar certificate authority was compromised [15]. This lead to vast amounts of malicious certificates being issued, some of which were subsequently used in man-in-the-middle attacks on Google services [3]. Other breaches of CAs and other centrally-trusted authorities, such as software update signing authorities, have continued to undermine PKI security [13], and even honest, uncompromised central authorities often prove to be very lax in actually enforcing their claimed security policies [6].

It seems clear that eliminating or reducing central points of trust in a PKI would greatly help it achieve stronger security, and doing so is indeed a key goal of most newer PKIs.

C. Transparency-based approaches

Conifer's design is derived from centralized *transparency*-based PKIs. This class of newer PKI design, instead of attempting to completely remove any centralized trusted third parties, aims to increase the public visibility of their operations, generally by using some sort of auditable, hard-to-forge log of everything the trusted party does. By making changes in the naming system public, anybody can check for evidence of malicious behavior by the trusted third party; the hope is that the very high probability of eventual discovery would deter potentially malicious trusted parties from carrying out attacks. Certificate Transparency (CT) [10], an experimental IETF open standard and extension to the traditional CA-based PKI, is probably the most well-known transparency-based system; it provides logs where CAs can transparently deposit certificates for examination for suspiciousness.

The direct ancestor of Conifer, though, is CONIKS [11], an entirely new secure naming system based on pairing transparency with a trusted service rather than an extension to an existing PKI. Unlike CT, which relies on third-party monitors that analyze every change in the PKI for suspiciousness, CONIKS introduces the concept of *self-monitoring*, where each owner of a name securely monitors log entries related only to his or her own name for suspicious behavior, using novel data structures that eliminate the need to monitor the entire log. In addition, CONIKS supports optional *strict* users that sign their own name binding changes, giving a measure of identity retention, making it possible for monitors to produce definitive rather than heuristic proof that the trusted service is either honest or misbehaving. Third-party auditors communicating with each other, though, are still needed to check that logs do not violate constraints such as being append-only and globally consistent.

Transparency-based schemes do significantly enhance PKI security, reducing blind trust in central authorities by making it easy to quickly produce evidence that a certain trusted authority is misbehaving and distrust it, while keeping the authority in full control of policy enforcement. However, in general the following problems are apparent:

1) Reliance on altruistic third parties: Additional third parties, such as monitors and auditors, must be introduced to keep watch on the trusted authority, but there is little incentive for third parties to contribute.

2) Need for secure "whistleblowing": Malicious behavior by the trusted service provider is not detected uniformly across all participants: for example, in CONIKS, only the owner of a name can detect suspicious changes by the service provider, while other parties looking up the name would not see any evidence of problems. Transparency-based systems thus typically assume that evidence of malicious behavior can be globally broadcast [11], [10] quickly enough that the reputation of the service immediately drops and usage discontinues.

3) Reactive rather than proactive: Finally, transparency fundamentally facilitates *damage control* after a PKI's security has already collapsed, by severely tarnishing the reputations of central providers who behave maliciously. Although this may deter intentionally malicious providers who wish to covertly tamper with name bindings, in practice centralized providers often become the target of compromise by unrelated attackers, who may simply wish to, for example, steal as much confidential data as possible through man-in-the-middle attacks before discovery.

D. Blockchain-based PKI

Unlike PKIs that attempt to simply keep watch on centralized trusted parties, naming systems based on embedding name information in a blockchain are fundamentally decentralized, with no trusted parties at all. All participants in a blockchain network form a peer-to-peer network that maintains an append-only, growing log of transactions; without changing the consensus of a majority of the network, past blocks cannot be rewritten. Importantly, blockchains typically use economic incentives to encourage selfish participants, or "miners", to honestly contribute large amounts of resources to maintaining and verifying the append-only log, making it very difficult for attackers to amass enough resources to hijack the network consensus.

Namecoin [2] is the first blockchain-based naming system, and it is implemented as a separate cryptocurrency based on the Bitcoin code [12], designed to function as an alternative to the Domain Name System (DNS). As the first fork of the Bitcoin software, Namecoin adds to the blockchain a name-value store for mapping domain names to DNS records [8]. In a blockchain-based PKI, identity retention of names is protected as long as the honest nodes in the network reach a consensus, as there is a single canonical log of events in the blockchain, and all name updates are signed by the name owner.

Unfortunately, blockchain-based systems do have significant problems with scalability and deployability — having every user of the PKI replicate an ever-growing blockchain is clearly impractical. Although state-of-the-art blockchain PKIs such as Blockstack [4] do mitigate this overhead somewhat, usage of these systems on devices such as smartphones generally require delegating the work of storing and querying the blockchain to a trusted gateway, defeating the robustly decentralized trust of blockchains.

E. Towards a better system

Both transparency-based and blockchain-based naming systems make large strides towards stronger security compared to traditional, centralized-trust PKIs, yet important unsolved problems remain. In particular, transparency-based systems suffer from lingering security issues due to the still-trusted central authority, while blockchain-based systems have severe limitations in scalability and flexibility due to the need to synchronize an unwieldy, permissionless blockchain across a peer-to-peer network.

Catena [14] pioneered the idea of *combining* a transparency-based system with a blockchain: in particular, the authors show that by embedding part of the data structure of CONIKS into the Bitcoin blockchain, the need for third-party auditors is eliminated. Unfortunately, Catena's variant of CONIKS still partially retains the reactive security model of CONIKS, failing to fully decentralize trust — self-monitoring and whistleblowing is still required to detect malicious behavior by the provider.

It is clear that a newer strategy is needed. We want a PKI that provides robust identity retention backed by a blockchain, like Namecoin and Blockstack, but on the other hand the flexible policy enforcement and high performance of transparency-based systems. By basing its architecture on that of Catena and CONIKS, but making some crucial changes to CONIKS's data structure and verification algorithm, Conifer aims to achieve exactly that.

III. Overall architecture

Conifer uses a client-server architecture, where *providers* administer *namespaces*; *clients* then look up names in a certain namespace by querying a particular provider. Instead of imposing a global infrastructure of providers, Conifer allows developers to set up application-specific providers, similar to how traditional PKIs may have application-specific root CAs — thus, there is no such thing as a canonical "Conifer name".

The general design of Conifer is based on that of CONIKS, a state-of-the-art transparency-based naming system, combined with that of Catena; this combination has already been suggested by the Catena authors. However, Conifer aims to completely prevent violations of identity retention by the provider, unlike the straightforward use of Catena and CONIKS, which though eliminating the need for third-party auditors, still uses reactive security based on whistleblowing by self-monitoring name owners.

In order to achieve that goal, in addition to adopting Catena's use of a *transaction chain* [14] in a Bitcoin blockchain to efficiently eliminate the need for third-party auditors, several important changes are made to the centralized portion of CONIKS.

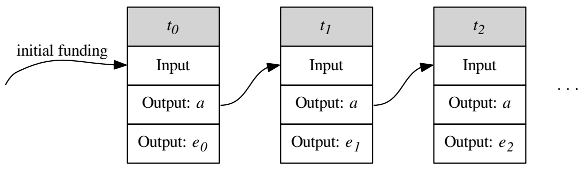

Fig. 1. Transaction chain

The CONIKS radix trees are replaced with an *operation forest* consisting of chronologically-ordered *operation trees*, each containing updates and name registrations within a certain time period, rather than a full snapshot as in CONIKS. This allows clients to more efficiently scan through the entire history of a name on every lookup — a change that eliminates monitoring and gives Conifer fully decentralized trust and proactive security. Furthermore, instead of a dichotomy between "strict" mode or "nonstrict" mode, Conifer allows name owners to flexibly specify permissions on a name binding, ranging from fully trusting the provider to complex hierarchical key quorums, making it much easier to adopt provider-distrusting key management policies. As in Blockstack [4] and Catena [14], we choose the Bitcoin blockchain for its much higher level of security compared to other public blockchains, even though Conifer would be quite easy to port to almost all cryptocurrency blockchains.

A. *Transaction chain*

The transaction chain of a provider is simply a Catena log [14] stored in the Bitcoin blockchain. A Catena log provides the abstraction of a public append-only list with the following properties:

- Only the provider can append to the log
- Two clients' views of the log are always identical, even assuming a malicious provider
- Entries in the log can be quickly obtained and verified using Bitcoin's *simple payment verification* (SPV) [12] algorithm, without compromising the above two properties

We can leverage the properties of Bitcoin (or in fact, any cryptocurrency blockchain) to implement this abstraction. The provider controls the private key corresponding to a single Bitcoin address, a. Before the first transaction in the chain, the *genesis transaction* t_0 is broadcast, a should control a significant amount of bitcoins. t_0 would then take all this money as its input, and have two outputs: the first one sending all the money (less the transaction fee) back to a, and the second one sending the zero bitcoins to an `OP_RETURN` metadata entry encoding an arbitrary value e_0.

After that, each t_i would spend the first output of t_{i-1}, sending most of it back to a in its first output and recording e_i as its second output.

Fig. 1 illustrates the structure of the transaction chain. It is clear that we have the desired properties listed previously:

- Only the provider, who controls the private key corresponding to a, can append to the chain
- Bitcoin's double-spending prevention means that given the location of the genesis transaction in the blockchain, there is unambiguously a single transaction chain that follows
- All the relevant entries in the log can easily be obtained by querying for transactions involving a; SPV ensures that the transactions indeed form the unambiguous chain accepted in the blockchain.

B. *Operation forest*

The operation forest, on the other hand, is a data structure that exists in the provider's database, and is not stored in the blockchain. It is a chronologically-ordered collection of key-value dictionaries called *operation trees*; each tree corresponds to a fixed period of time known as an *epoch*, and is a dictionary mapping names in the namespace to changes that happened to the names. Thus, the operation forest as a whole stores the entire history of every name in the namespace; it is analogous to the snapshot trees of CONIKS, though instead of storing a snapshot of the namespace at each epoch, it stores only the changes to the bindings.

1) Operation tree structure: An operation tree is a data structure used by the provider to store changes to the namespace happening over a given epoch. Unlike the binary radix tree used in CONIKS, it is implemented using a perfectly balanced *Merkle binary search tree*. Each node is either empty, or stores:

- An *index* derived from name in the namespace
- All the changes done to the name over the time period covered by the tree
- A hash of the left child of the node
- A hash of the right child of the node

As in CONIKS, a verifiable unpredictable function (VUF) that only the provider can compute is used to map names to their respective indexes to increase the difficulty of name enumeration, which can be a privacy risk [11]. Fig. 2 is an illustration of the structure with 5 names, omitting the values associated with the names or the VUF.

We choose a perfectly balanced BST instead of a Merkle binary radix tree as in CONIKS in order to guarantee that the height of the tree is logarithmic in the number of changes in an epoch; as we will later see minimizing the height of the operation trees is crucial to reducing lookup overhead.

2) Proving existence and nonexistence: An important property of operation trees that it shares with CONIKS's Merkle radix trees is that given the hash of the root node, we can efficiently produce a *proof* that a certain name exists in the tree with a certain value associated with it, or a proof that it does not exist. We illustrate this by using the tree in Fig. 2.

For example, the collection of nodes $n_{\text{root}}, n_1, n_{11}$ is a proof that "fred" exists in the tree, with a certain value associated with it (not drawn in the tree). As long as we have a trustworthy hash of n_{root}, this proof cannot be forged. We can also prove, for instance, that "aaron" does not exist in the tree, by $n_{\text{root}}, n_0, n_{00}$, since if "aaron" does exist, n_{00} would not lack children.

In general, the structure of the tree guarantees that there can be only one valid entry for each name in the tree — there cannot be two different bindings $N : V$ and $N : V'$, both with valid

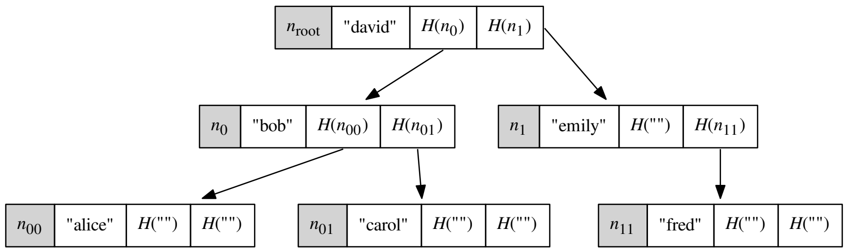

Fig. 2. Example of an operation tree

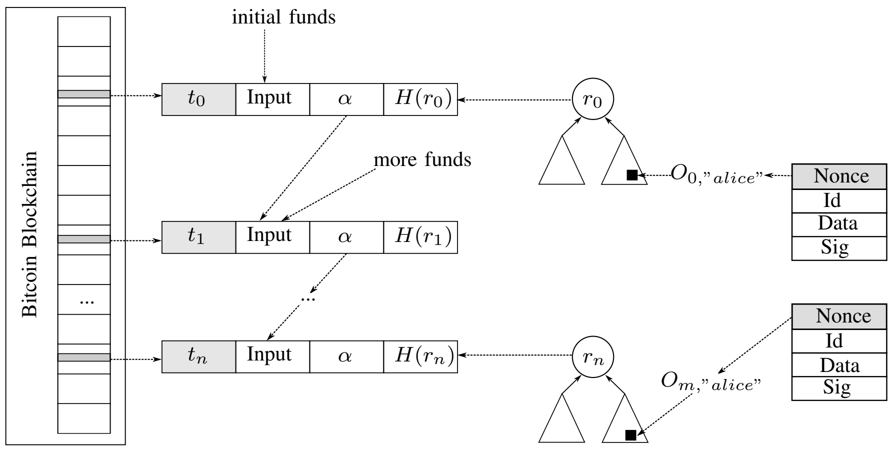

Fig. 3. Conifer overview

proofs of existence. This result can be shown easily. Consider validating the proof consisting of the nodes $p_1, p_2, \ldots, p_n$, given as a result for a query for the name N. For every p_i,

- If $N = p_i.\text{name}$, then the proof should stop at p_i.
- If $N < p_i.\text{name}$, and if p_i has a left child, then $H(p_{i+1})$ must be $p_i.\text{lefthash}$.
    - In case p_i has no left child, this is a proof of non-existence, and should stop at p_i.
- (The case of $N > p_i.\text{name}$ is the same, except with left and right reversed)

It is clear that given any node in the proof, we can perfectly predict either the next node in the proof, or that the proof ends. Thus, given that the root of the tree p_1 and the name queried N are both known, there can only be one valid proof of (non)-existence $p_1, \ldots, p_n$ and one binding $V = p_n.\text{value}$. Nobody with knowledge of the tree root could be fooled into accepting any other value V'.

Finally, since the tree is perfectly balanced, the length of a proof is bounded by the height of the tree, which is proportional to $\log \delta$, where δ is the number of names in the operation tree (i.e. the amount of name bindings that have changed since the last operation tree).

C. *Securing the operation forest with a transaction chain*

Using the properties of the operation forest, we see that we can securely verify the history of any name, given that we have:

- Trusted hashes of the roots of all the operation trees in the forest
- For each tree in the forest, a valid proof that the operations done to the name within the tree interval exist in the tree or, if no operations were created for the name, a valid proof that no binding to the name exists in the tree (the proofs do not need to come from a trusted source)

To store the trusted hashes, we use the “transaction chain” mentioned earlier the same way the Catena authors applied it to plain CONIKS — every e_i stored in the chain is a cryptographic hash of the ith tree’s root. By the properties we already know hold for a transaction chain, we then know that assuming the security of the underlying blockchain, everybody will obtain the same, unforgeably append-only list of tree root hashes, and thus, also the same unforgeably append-only history of a name.

Of course, simply having an append-only history of a name only gives us a measure of transparency on the behavior of the provider, and not the full identity retention desired. To achieve that, cryptographically secured *operation logs* are needed.

D. Operation logs and identity retention

The operation forest, combined with the transaction chain, gives us a way of securely obtaining an append-only history of any name, which only the provider can append to — the operation log, consisting of many individual *operations*. Similar to CONIKS's "strict mode", each operation in the operation log must be signed by a cryptographic identity declared in the previous operation, preventing any changes to a name's binding unauthorized by the owner of the name.

1) Structure of an operation: Each operation contains the following fields:

- An *identity script* representing the cryptographic identity authorized to make changes to the data bound to the name
- A collection of cryptographic *signatures*, valid with respect to the identity script declared in the previous operation
- Data associated with name

2) Identity scripts and signatures: Identity scripts are the entities representing cryptographic identities in Conifer. The ith operation o_i in an operation log is only valid if the signatures on o_i are valid with respect to the owner identity script declared in o_{i-1}. Each identity script represents a tree structure of key quorums; every identity represented by a script is either:

- An Ed25519 [5] public key, or
- n out of m identities

Fig. 4 depicts a hypothetical identity of a user in a chat application, where the PKI manages important public profile details, such as name and application-level public key. Our example user has enabled two-factor authentication for his or her account, so changing important data must require authorization using both the password and a device; the Conifer identity itself elegantly encodes this requirement.

We use a simple stack-based scripting language, inspired by payment scripts in Bitcoin [12], to represent these trees.

E. Looking up and registering names

We have now described a way of organizing a name database in the provider, with identity retention protected by the Bitcoin blockchain and an operation signing system. A client can then securely look up names in the PKI following the steps below:

Obtain transaction chain: The client starts with the hash of the genesis transaction in the transaction chain hardcoded into the software. A protocol like Stratum is used to obtain every transaction in the chain from an untrusted Bitcoin node, and SPV is used to verify that all the transactions are indeed in the real Bitcoin blockchain. We now have $e_1, \ldots, e_n$, all the entries in the transaction chain.

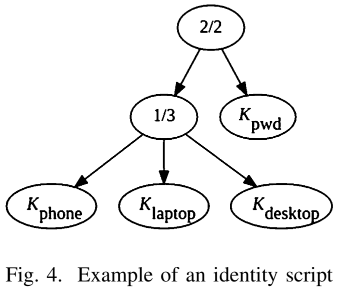

Fig. 4. Example of an identity script

Query name history: We now query the provider for the history of the name N. For each time period $t_i \in t_1, \ldots, t_n$ and its corresponding operation tree:

- If any operations were done during t_i, the provider sends us the operations, together with a proof that the operations exist in the operation tree.
- If no operations were done during t_i, the provider sends us a proof that nothing is bound to N in the tree.

Finally, we validate the ith proof to make sure it indeed is a valid proof starting from a tree root that hashes to e_i from the transaction chain.

Registering names is actually not defined by the Conifer system itself: individual provider operators may define their own ways in which new operations are submitted to and vetted by the provider before publication. For example, an provider operated by a secure messaging service may require the user to complete a registration form in an app before any operations are published.

Validate operation log: All the operations from the previous step are organized in chronological order, and the signatures on each operation are checked with respect to the identity script declared in the previous operation.

F. Summary

In this section we presented an architectural overview of Conifer. Our basic design is similar to the blockchain-enhanced CONIKS proposed by the Catena authors, with two data structures, the transaction chain, which roots the consistency of name bindings to the Bitcoin blockchain, and the operation forest, which enables efficient and secure lookups against an untrusted database, underpinning Conifer's design. Unlike CONIKS, Conifer eliminates the need for monitoring by moving the responsibility of validating a name's history from name owners to everybody who looks up the name; the use of a different data structure prevents this from being prohibitively expensive. This allows Conifer to achieve strong, proactive identity retention comparable to traditional blockchain PKIs, eliminating the lingering centralized trust of CONIKS. Fig. 3 is an overview of the data structures involved in Conifer.

IV. Evaluation

In this section, we evaluate Conifer against existing naming systems by qualitatively measuring their performance using experiments.

A. Lookup performance

In our first quantitative experiment, we evaluate the performance of doing lookups in Conifer. As a comparison, we also evaluate the performance of secure client-server lookups in Blockstack, the current state-of-the-art in blockchain-based PKIs. Unlike other blockchain-based naming systems, Blockstack has a secure thin-client lookup system — SNV — with at least some decentralized trust, allowing a contest between two systems with comparable secure.

Both a Blockstack full node and a Conifer provider are installed on a server, and a client with around 50 ms of network

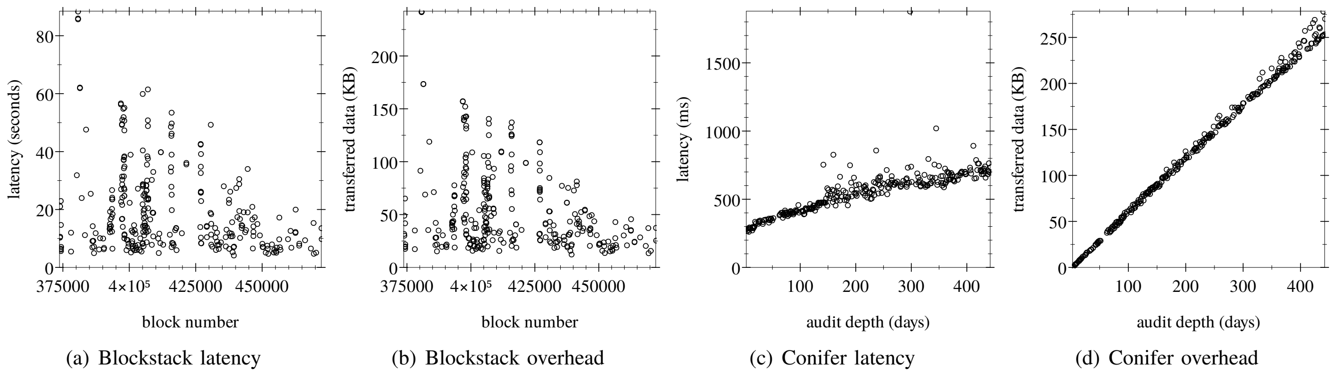

Fig. 5. Lookup performance of Blockstack SNV against Conifer

latency to the server is used to benchmark the two systems. We query our Conifer provider with 300 random dummy names previously placed in the namespace, with various cutoffs to the tree audit depth to simulate names of differing cache staleness or expiry times (see **??**), while for Blockstack we use SNV to verify 300 random existing name records in the operational Blockstack network. Total latency and bytes transferred are then measured by tracing network packets using Wireshark, avoiding inaccurate measurements due to application startup latency.

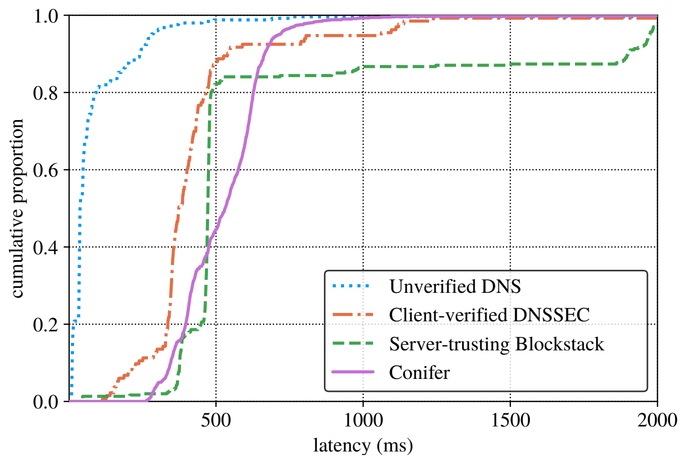

Fig. 6. Conifer lookup latency, compared with centralized-trust systems

We can plainly see the drastically different performance characteristics of the two systems in Fig. 5 summarizing the results of this experiment. Blockstack's transfer overhead grows significantly as we query names registered in older and older blockchain blocks, while latency is extremely high — ranging from 5 to 80 seconds — and proportional to bytes transferred. This is expected, as SNV needs to verify more blocks the further back the name is in history; it is also implemented as an interactive protocol where every bit of data must be separately requested.

Conifer, on the other hand, shows obvious linear growth of lookup overhead and slow, roughly linear growth of lookup latency in proportion to audit depth. In particular, latency is bound by the network round-trip time for short audit depths and is extremely low in comparison to Blockstack — around 200 ms for short audit depths and 700 ms for year-long audit depths. This is due to the lookup protocol, which was carefully designed so that the client can obtain all the (non)existence proofs for a name within a single network round-trip. On the other hand, for very long audit depths, the amount of data transferred can reach several hundred KB, exceeding that of a typical Blockstack query, though in practice most lookups would be for partially cached names with short audit depths and involve far less data transfer.

Not only is Conifer's lookup procedure much more performant than the state-of-the-art blockchain-based secure thin client, it offers acceptable speed even compared to systems with centralized trust. Fig. 6 compares lookup latency between Conifer and three systems lacking distributed trust: unsecured DNS, DNSSEC-secured DNS with all signature validation done by the client, and Blockstack's default server-trusting mode. For DNS, we look up a list of US government DNSSEC-enabled domains [1] using the `dig` tool and its `+sigchase` option, while for Conifer and Blockstack we use the same set of random names from the first experiment. We see that though DNS is very fast due to its aggressive caching, compared with client-verified DNSSEC and server-trusting Blockstack, both of which are less amenable to caching by the ISP, Conifer offers quite usable performance, even though it has fully-verifying clients and distributed trust.

Thus, we conclude that the price Conifer pays in lookup performance to achieve much stronger security is quite small, and should not be problematic for the vast majority of applications.

B. *Bootstrapping data*

To achieve identity retention as strong as that of Conifer, previous blockchain-based systems must use full nodes rather than thin clients like Blockstack's SNV, but blockchain full nodes are notorious for requiring very large and linearly-growing amounts of *bootstrapping data*. Classically, all full nodes must download the entire blockchain on first connection and continually synchronize it to local storage, causing large delays in joining the network. Even if optimizations such as pruning [12] and "fastsync" , commonly deployed on Blockstack nodes [4], obviate the need to actually store all the blocks seen and speed up initial download, new blocks must still be constantly replicated across all nodes, using up significant amounts of bandwidth.

Conifer also has two pieces of bootstrapping data — the transaction chain and Bitcoin blockchain headers — which just like a blockchain grow linearly and must be synchronized by every Conifer client. How does the cost of keeping up with this data compare to downloading and and keeping up with a blockchain?

TABLE I
GROWTH RATES OF BOOTSTRAPPING DATA

System	Mean monthly growth	Cumulative size
Bitcoin (Blockstack)	1.20 GB	155.6 GB
Ethereum	3.37 GB	337.6 GB
Namecoin	67.6 MB	5.26 GB
Conifer	370.2 KB	40.7 MB †

To answer this question, we use Table I, which shows how fast bootstrapping data grows for various blockchains and Conifer — blockchain data is gathered from existing historical records, while Conifer data comes from measuring an actual Conifer client's disk usage over a period of time. Note that Blockstack full nodes also need to catch up with the Bitcoin network like a Bitcoin full node, and thus the Bitcoin numbers also apply to Blockstack.

It is clear that the rate at which blockchains typically grow is quite high, and even simply catching up continually could deplete the available bandwidth of, say, smartphones on limited data plans. In addition, even blockchains with very low activity, such as Namecoin, still eventually accumulate gigabytes of blocks, placing a barrier to newly-bootstrapping full nodes. On the other hand, although Conifer does have linearly-growing bootstrapping data, the growth rate is minuscule compared to blockchains and would not be a problem for all but the most tightly constrained embedded environments.

V. Future work

In this section, we discuss our future plans for deploying and further improving Conifer.

A. Real-world deployment in chat application

We are currently developing a open-source secure instant messaging / VoIP application, Aether, that uses Conifer as its underlying PKI. We hope to use it not only to evaluate how Conifer performs in a realistic scenario, but also as a demonstration of how Conifer allows blockchain-backed security to be easily integrated in a user-friendly application. In particular, we intend Aether to show how Conifer's flexible identity script system can be used to build a key management system with a user experience similar to existing password-based and two-factor authentication, enabling strong identity retention without reducing usability.

B. Reducing lookup overhead

Although tricks such as caching do significantly reduce average-case lookup overhead, in the worst case, a client looking up a name may need to download a proof of (non)existence for every day since the provider began operating. For some applications, this may cause unacceptable overhead, while in many cases name expiry and similar solutions that weaken identity retention may not be secure enough. Strategies that can significantly reduce the worst-case lookup overhead of Conifer without compromising security would be helpful in allowing Conifer to be used in extremely resource-constrained environments, such as embedded systems or IoT devices with very slow network links, and it is thus a potentially fruitful future research area.

VI. Conclusion

We presented Conifer, a scalable naming system with centralized control yet blockchain-backed trust. Conifer uses a new architecture informed by the successes and shortcomings of transparency-based PKIs such as CONIKS and blockchain-based systems such as Blockstack, ensuring that the central administrator cannot violate identity retention without breaking the security guarantees of the underlying cryptocurrency blockchain. Experimental results show that Conifer performs markedly better than existing blockchain-based systems, with faster lookup than thin clients and dramatically reduced storage overhead compared to full nodes, while demonstrating that its performance penalty compared to traditional centralized PKIs is fairly small.

References

[1] "Estimating usg ipv6 & dnssec external service deployment status." [Online]. Available: https://fedv6-deployment.antd.nist.gov/cgi-bin/generate-gov

[2] "Namecoin." [Online]. Available: http://namecoin.info

[3] H. Adkins, "An update on attempted man-in-the-middle attacks," *Google Online Security Blog*, 2011.

[4] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. USENIX Association, 2016, pp. 181–194.

[5] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, pp. 1–13, 2012.

[6] A. Delignat-Lavaud, M. Abadi, A. Birrell, I. Mironov, T. Wobber, and Y. Xie, "Web pki: Closing the gap between guidelines and practices." in *NDSS*, 2014.

[7] C. Fromknecht, D. Velicanu, and S. Yakoubov, "A decentralized public key infrastructure with identity retention," *Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep*, vol. 6, 2014.

[8] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design," in *WEIS*, 2015.

[9] D. Kaminsky, "Spelunking the triangle: Exploring aaron swartz's take on zooko's triangle," January 2011. [Online]. Available: http://dankaminsky.com/2011/01/13/spelunk-tri/

[10] B. Laurie, A. Langley, and E. Kasper, "Certificate transparency," Tech. Rep., 2013.

[11] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "Coniks: Bringing key transparency to end users." in *USENIX Security Symposium*, 2015, pp. 383–398.

[12] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[13] A. Niemann and J. Brendel, "A survey on ca compromises." [Online]. Available: https://www.cdc.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_CDC/Documents/Lehre/SS13/Seminar/CPS/cps2014_submission_8.pdf

[14] A. Tomescu and S. Devadas, "Catena: Efficient non-equivocation via bitcoin," in *IEEE Symp. on Security and Privacy*, 2017.

[15] K. Zetter, "Diginotar files for bankruptcy in wake of devastating hack," *Wired magazine, September*, 2011.