Received: 19 February 2021 | Revised: 22 July 2021 | Accepted: 26 July 2021

DOI: 10.1002/nem.2183

SPECIAL ISSUE PAPER

WILEY

Veritaa: A distributed public key infrastructure with signature store

Jakob Schaerer[1,2] | **Severin Zumbrunn**[2] | **Torsten Braun**[1]

[1]Institute of Computer Science, University of Bern, Bern, Switzerland

[2]Abilium GmbH, Bern, Switzerland

Correspondence
Jakob Schaerer, Institute of Computer Science, University of Bern, Bern, Switzerland.,
Email: jakob.schaerer@inf.unibe.ch

Summary

Today, the integrity and authenticity of digital documents and data are often hard to verify. Existing public key infrastructures (PKIs) are capable of certifying digital identities but do not provide solutions to store signatures immutably, and the process of certification is often not transparent. We propose Veritaa, a distributed public key infrastructure with an integrated signature store (DPKISS). The central part of Veritaa is the Graph of Trust that manages identity claims and singed declarations between identity claims and document identifiers. An application-specific distributed ledger is used to store the transactions that form the Graph of Trust immutably. For the distributed certification of identity claims, a reputation system based on signed trust declarations and domain vetting is used. In this work, we have designed and implemented the proposed architecture of Veritaa, created a testbed, and performed several experiments. The experiments show the benefits and the high performance of Veritaa.

1 | INTRODUCTION

In the wave of digitalization and with the fast growth of the Internet of Things (IoT), digital documents and data become an essential and valuable resource. The high value of data mainly results from their high availability, easy accessibility, and the possibility to move them effortlessly all over the world. Nevertheless, when data are moved over different channels, it is almost impossible to verify their integrity and authenticity. For many real-world applications, it is of great importance that the integrity and authenticity of data can be verified. Hashes are usually used to check the integrity of data and digital signatures to check its authenticity. Asymmetric encryption with private and public keys is used for digital signatures. Nevertheless, as long as public keys are not mapped to entities, their signatures are just unknown entities' signatures. An identity claim maps a public key to an identifier of an entity. However, as long as the identity claim is not authenticated, it is just a claim that everyone could have created. It would be possible to authenticate all public keys manually in a small world, but this is neither applicable nor scalable in most cases.

Public key infrastructures (PKI) are required for the authentication of a large number of identity claims. Public key infrastructures can be distinguished by their certification model. Public key infrastructures use certificate authorities (CA) who sign certificates, and distributed public key infrastructures (DPKI) often rely on the decentralized Web of Trust.

The X.509[1] standard is a popular PKI. X.509 certificates are defined as data structures that bind public-key values to subjects, and trusted certificate authorities (CAs) assert these bindings.[2] The CAs are trusted in tree-shaped hierarchies. A root CA is asserting subordinate CAs by validating their identity. With over one thousand CAs all over the world,[3] misfortune or bad intentions cannot be excluded, and the security of some CAs might be compromised. In the past, several incidents showed flaws of CA-based PKIs.[4] For example, hackers could get access to servers of the certificate authority DigiNotar and used this CA's infrastructure to issue false certificates of high-profile domains like Google.[5] Incidents like this show the limitations of hierarchical and intransparent systems regarding security and reliability.

Pretty good privacy (PGP),[6] together with the Web of Trust (WoT),[7,8] is a popular DPKI. In the Web of Trust, the public keys are certified by other entities.[7,8] While the WoT provides transparency about the trust relations between entities, it lacks scalability and usability. Additionally, public keys can not easily be revoked with the WoT.[9]

PKIs are responsible for the management of a key during its whole life-cycle. The life-cycle of a public key contains its creation, authentication, certification, utilization, and deprecation. To validate a signature it is crucial to know in what state a public key was when the signature was created. If this state is not known, all signatures lose their validity when the public key enters its deprecation state. Each signature and the state of the public key must be stored non-repudiably and immutably to preserve a signature's validity.

In this work, we propose Veritaa, a decentralized public key infrastructure with signature store (DPKISS). Veritaa consists of the Graph of Trust (GoT), which stores identity claims of entities and signatures created with these keys. On the Graph of Trust, entities can sign trust declarations towards other identity claims they have authenticated. In this work, we propose using these trust declarations as reputation votes and domain vetting for the distributed certification of identity claims. Domain vetting is the process of performing the check that the creator of an identity claim is also the owner of a specific domain. To ensure the Graph of Trust's immutability, we have created a distributed ledger called the acyclic block confirmation graph (ABCG) optimized for the storage of graph transactions. Furthermore, we propose a distributed block discovery service to map the blocks to timestamps. This work is an extended version of Veritaa—the Graph of Trust.[10]

The main contributions of this paper are as follows: the design and implementation of Veritaa, a system to manage digital identities and their signatures; the design and implementation of the acyclic block confirmation graph (ABCG), a BlockDAG-based DLT optimized to store append-only graph transactions; the design of a reputation-based certification model based on the Graph of Trust (GoT); evaluation of the proposed architecture.

2 | RELATED WORK

2.1 | Distributed ledger technology (DLT)

A distributed ledger is a distributed database used to share, replicate, and synchronize data across multiple locations and between various entities. To achieve a replica of the database on all participating nodes, they must consent on what transactions are added to the distributed ledger.

Blockchains are the most prominent form of distributed ledgers. In blockchains, transactions are collected in blocks, and each block confirms its predecessor by adding the block hash of the previous block in its own header. Since each block can only have one successor, the network requires a consensus algorithm like Proof of Work (PoW)[11] or Proof of Stack (PoS)[12] to decide which node is allowed to create the next block. If multiple blocks confirm the same block, the longest blockchain is considered valid. The consensus algorithm makes it almost impossible for a single entity to mine multiple consecutive blocks, and therefore the transactions can be considered immutable. In blockchains, transactions are absolutely ordered. The cryptocurrency Bitcoin is built on a blockchain with PoW as a consensus algorithm.[11] The success of Bitcoin revealed the drawbacks of PoW. The arms race between the miners led to an enormous energy consumption,[13–16] and the throughput limited to seven transactions per second limits its scalability.[17] Ethereum is a blockchain popular for its generalized technology on which all transaction-based state machine concepts can be built.[18] In Ethereum, smart contracts written in a Turing complete programming language can be executed.[18] Ethereum was initially using PoW as consensus algorithm. However, due to scalability and efficiency issues of PoW, Ethereum is in progress to migrate to PoS at the time of writing.

BlockDAGs are DLTs where blocks confirm multiple other blocks[19] and therefore build a directed acyclic graph instead of a single blockchain. Another approach is the TDAG, where the DAG is built on transactions instead of blocks.[19] An advantage of DAG-based DLTs is that it is easier to append new blocks to a DAG than to the end of a blockchain. This makes DAG-based DLTs more suitable for nodes with constrained resources, enables higher throughput, and improves energy efficiency. However, in contrast to blockchains, the DAG data structures do not inherently contain a total order of transactions. Therefore, in applications where the node state depends on the execution order of the transactions, the network requires to build consensus about the transaction's order. For example, the cryptocurrency IOTA is built on a TDAG, the Tangle.[20] Since the DAG data structure does not provide a total order of the transactions, IOTA initially used a centralized coordinator to resolve conflicting transactions. At the time of writing, IOTA is migrating to a new decentralized consensus algorithm.[21]

In this work, we propose to use the ABCG, an application-specific BlockDAG, to store a distributed PKI and signatures immutably. The ABCG is optimized for append-only transactions that build a graph. Furthermore, the append-only transactions are designed so that the order of execution does not influence the system state.

2.2 | Public key infrastructure (PKI)

Several public key infrastructures based on distributed ledgers have been proposed in the literature. The decentralization of the single point of failure in CA-based PKI and the certificate transparency is stated as the main advantage of DLT-based PKI. DLT-based PKIs can be categorized by how the certificate requests are certified. Like in conventional PKIs, the certification is mainly achieved through trusted CAs or the Web of Trust.

In blockchain-based PKI Solutions for IoT, the authors implement and evaluate three different PKI on Emercoin, and Ethereum.[22] The first proposed PKI uses the Emercoin name value store to store certificates on the blockchain. The other proposed solutions rely on Ethereum smart contracts. To meet the requirements of IoT applications, they evaluated two Ethereum architectures: one with a trusted remote node and one with an Ethereum light sync node. However, the authors focus on storing self-signed certificates and are not providing a solution to authenticate and certify public keys. Therefore, it is difficult for third parties to verify the authenticity of the stored certificates.

Kubilay et al. proposed CertLedger, a PKI architecture with certificate transparency based on a blockchain.[23] In their work, they aim to eliminate the split-world attacks and provide certificate and revocation transparency. CertLedger is based on an Ethereum smart contract. CAs perform the tasks to check the identity of a domain owner and the certification of certificate requests. All certification and revocation actions are transparently stored on the blockchain. The CertLedger board observes all actions and can untrust malicious CAs. The authors show a complete architecture for CA-based PKI built on a smart contract and propose a sound governance model.

In Papageorgiou et al,[24] the authors design and implement a blockchain-based DPKI on top of hyperledger fabric. To reduce the risk of a single point of failure, they propose a scheme where multiple trusted CAs certify certificate requests. For the distributed certification, they use multiple signatures of different network operators (CAs) to sign public keys and then store them on the blockchain.

PBCert is a privacy-reserving blockchain-based public key infrastructure with a focus on scalability.[25] To reach scalability, PBCert is separated in a control and storage plane. In the control plane, all certificate operations are stored on a blockchain. Certification requests are sent to trusted CAs, and the certificates are then stored in an Ethereum blockchain. The certificate revocations are stored on OCSP servers' in the storage plane, and the OCSP servers' root hashes are secured on the blockchain of the control plane. In PBCert, obscure responses to client's requests are used to preserve privacy. Therefore, the client only sends the first n bits of the certificate hash to an OCSP server, and the server responds with a BloomFilter of matching hashes.

A complete blockchain-based PKI system that supports the X.509 standard[1] has been implemented in a smart contract on top of the Ethereum blockchain.[2] All certificates, including those of the CA's, are stored on the blockchain. Therefore, misbehavior of a CA is tracked on the blockchain and can be noticed by all participating entities. In the smart contracts of each CA, lists with all issued and revoked certificates are contained.[9] This improves the security of the tree-based PKI systems.

Bitcoin has been used to enhance pretty good privacy (PGP) and the related WoT. A new certificate format based on Bitcoin has been introduced that allows a user to verify PGP certificates using Bitcoin identity-verification transactions.[26]

The smart contract-based PKI and identity system (SCPKI) is an alternative PKI system based on a decentralized and transparent design using a WoT model and a smart contract on the Ethereum blockchain. In SCPKI, entities can verify fine-grained attributes (such as phone number, company, or domain name) of another entity's identity.[27] The verification of the attributes results in a WoT like data structure. However, this work rather focuses on asserting specific attributes than on the derivation of public-key certificates. Additionally, the authors mention the operational costs for running the smart contract on Ethereum as a challenge and, therefore, publish a full implementation and a cheaper light implementation.

Several DLT-based PKIs have been proposed in the literature. These PKI mostly follow the traditional CA or WoT model and use the DLT as a distributed database to increase certificate transparency and enable fast revocation. Additionally, PKIs that follow the CA model often use smart contracts to control and govern CAs. With all certificate actions transparently stored on the DLT, governing boards can observe the CAs' behavior and revoke their permissions. PKIs that follow the WoT model mostly use DLTs as distributed databases and for fast revocation. In traditional WoT the revocation lists were difficult to realize. Most of the investigated projects rely on Ethereum smart contracts or existing general-purpose DLTs. In this work, we propose the Graph of Trust, a WoT-based certification model. Besides the trust relations of the WoT also signed declarations, revocations, and domain validation information are stored immutably on the GoT. To overcome the problem that it is difficult for new entities to get certified in a WoT, we propose to use domain validation. With the proof of domain ownership, well-known organizations and companies can bootstrap their reputation.

Veritaa uses the acyclic block confirmation graph, an append-only BlockDAG optimized for graph transactions, to store the transactions that build the GoT. These transactions describe how the GoT is built, and all nodes that apply all transactions of the ABCG on their GoT will end in the same state. The Graph of Trust is stored in an optimized graph database that enables big data algorithms known from social media analytics.

3 | VERITAA

Veritaa is a high performance and scalable distributed public key infrastructure with an integrated signature store (DPKISS). This means that it also integrates an immutable database to store public declarations that have been signed by key pairs managed by Veritaa. The Veritaa framework comprises the Graph of Trust (GoT), the acyclic block confirmation graph (ABCG) to store the Graph of Trust immutably, and a peer-to-peer network to connect multiple Veritaa nodes and exchange data.

3.1 | The Graph of Trust (GoT)

The Graph of Trust is used to store and publish identity claims, document identifiers, and signed declarations in Veritaa. Each Veritaa node maintains a copy of the GoT.

3.1.1 | Elements

The GoT consists of identity claims, document identifiers, and signed relations. These elements are used to represent real-world relations between entities and digital documents. Figure 1 shows the elements of the GoT. In the GoT, each

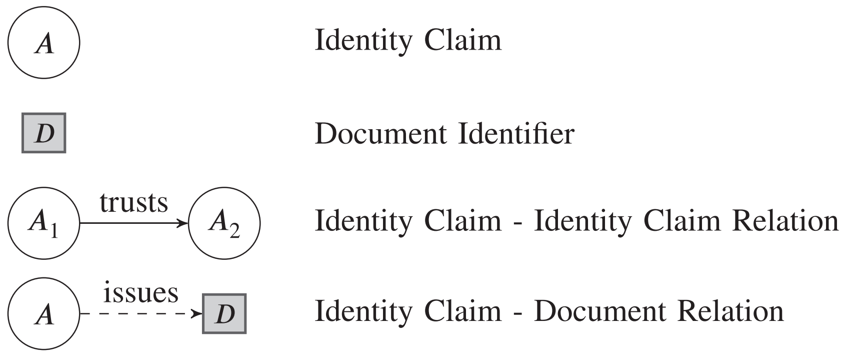
A
Identity Claim
D
Document Identifier
A1
trusts
A2
Identity Claim - Identity Claim Relation
A
issues
D
Identity Claim - Document Relation

FIGURE 1 Elements of the Graph of Trust

relationship starts from an identity claim and is signed by the key pair of the starting identity claim. Therefore, an identity claim can only create relations on its behalf.

3.1.2 | Authentication

An identity claim is a public key associated with the name of an entity to which it allegedly belongs. As each participant of the Veritaa network can create identity claims for all entities, the identity claims must be authenticated. To authenticate an identity claim domain vetting can be used. To enable domain vetting, the creator can add a validation domain to an identity claim. To prove that the same entity owns the domain and the identity claim, the creator has to solve a challenge that only the domain owner can solve. As a challenge, the creator can either add the hash of the public key to the well-known folder[28] in the HTTP URL of the domain, or the creator can add this hash to a DNS TXT record. Each participant of the network can also authenticate its peers manually (for example, by phone or passport validation). When an agent has authenticated another agent's identity claim, it can publicly declare this authentication on the GoT with a signed trust link.

3.1.3 | Certification

The mapping of digital entities to real-world organizations, devices, and people is a big challenge. In Veritaa, all entities with access to a Veritaa node can create identity claims. However, these identity claims are not yet certified, and the PKI requires a model to validate and certify their identity. In Veritaa, we propose using trust, reputation, and domain validation information to certify identity claims. Domain validation proves that the creator of an identity claim owns a particular domain. When the used domain has a good reputation, for example, a company's domain, a third party can use this information to authenticate an identity claim.

Trust

To authenticate public keys, an agent either has to trust a third party or to authenticate each public key independently. Due to the huge number of identity claims, it is not feasible to authenticate all public keys manually. Therefore, it is necessary to trust other entities that they have performed the authentication properly. In a centralized public key infrastructure, all have to trust the central entity that it performs the authentication of identity claims correctly. Any event that destroys trust in this central trusted entity causes the centralized public key infrastructure to collapse. In contrast, in the decentralized public key infrastructure, trust is distributed over all participating entities. Consequently, if some entities are not trusted anymore, this should only affect the identity claims certified by the entities that lost the network's trust. Therefore, a distributed public key infrastructure is more resilient to the failure of a certifying entity.

Trust is an important concept in the process of certifying the authenticity of third parties' public keys. Merriam-Webster defines trust as assured reliance on the character, ability, strength, or truth of someone or something.[29] A little more specific Gambeta defines: trust is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before it can monitor such action (or independently of its capacity ever to be able to monitor it) and in a context in which it affects its own action.[30] Merriam-Webster and Gambeta define trust as a binary relation. An agent either trusts another agent or not. Gambeta's definition highlights that trust depends on a subjective probability threshold, defined by the trusting agent. If a potentially trusting agent A thinks that the probability that a trusted agent B performs a particular action is above this threshold, then A trusts B and otherwise not. It is up to A's discretion to define this threshold, and its level depends on the risk when this trust was wrong. Different agents have varying priorities, and consequently, the definition of this threshold is subjective. In the real world, the definition of this threshold and the estimation of the probability that a certain agent performs a particular action is also subjective. In literature, different trust measures are used to estimate this probability. Some authors propose to classify the trustworthiness of entities in distrust, ignorance, minimal, average, good, and complete, where the evaluated entity is compared with the rest of the entities.[31] Furthermore, others propose to use the mode of authentication to define a confidence level. For example, authentication over a telephone line has a lower confidence level than a passport verification.[7]

In this work, we currently limit the trust relations to a simple declaration of trust from entity A towards an entity B and use an additional domain validation scoring to weight the trust relationships. Nevertheless, a confidence level for trust could be added by simply weighing the trust links. Additionally, we define that the (benevolent) signatory should have authenticated the identity claim out of the band for each signed trust declaration. A trust declaration must never be signed on information derived from the GoT.

Gambeta's definition also mentions that trust is context-sensitive.[30] For example, trusting the mailman to deliver a letter does not imply trusting him to fly a plane. Some certainly can, but most can not. In the process of authenticating identity claims in the GoT, two quite close but still very different contexts of trust can be found. The first context occurs when A signs a trust relationship towards B, then it assures that it has authenticated that the public key really belongs to the entity it claims, and, therefore, it trusts this identity claim is valid. The second context of trust can be found when A trusts B that its trust statements are valid. Trust that an identity claim belongs to a certain entity is not the same as trusting an entity to authenticate identity claims properly. The WoT uses recommendation links to assess how much one trusts that another agent correctly authenticates third parties.[7,8] However, even if this recommendation links in the WoT have a depth property that can be larger than one hop, the subjectivity of trust limits this scheme's applicability to one hop. Consequently, trust can be used for the certification of close identity claims, but it is difficult to draw conclusions about identity claims where the trust chain is long.

To overcome trust subjectivity and the short range of trust chains, we propose to use the public trust declarations in the GoT as reputation voting and the domain validation information to derive certification information.

Reputation

Trust describes what one agent thinks of another and reputation describes how the public sees an agent. Merriam-Webster defines reputation as overall quality or character as seen or judged by people in general.[32] The public declarations of trust can be considered recommendations. From these recommendations the reputation can be derived.

While trust is a subjective relationship that is difficult to evaluate, reputation is what the public thinks about an entity, and it can be measured. Algorithms like HITS,[33] closeness centrality,[34] Eigenvector centrality,[35] EigenTrust,[36] and PageRank[37] have been around for years and gained importance together with the fast growth of the Internet and Social Media. All these algorithms measure the centrality of nodes in a network. The centrality tells how well a node is connected in a network and how much influence it has. In a graph with trust links, this influence can be considered as reputation.

If it is possible to build a reputation system that prevents attackers from gaining an unjustified reputation, this reputation system can be used for distributed certification. The advantage of reputation is that it can be used to reason about the authenticity of distant identity claims.

While reputation can certify the authenticity of distant identity claims, trust can be used to certify the authenticity of identity claims with short certification paths. For most applications, a combination of reputation and trust will be meaningful. For example, let us assume that an agent wants to authenticate the signature of a local weather station. Since this local weather station is only a device, it is not domain validated and it does not have many incoming trust links. However, the countrywide meteorology service is well reputed and has a domain validated identity claim. The agent is now able to authenticate the meteorology service over reputation. From the countrywide meteorology service's identity claim, the agent can now follow the trust link to the regional department's identity claim and, therefore, authenticate this regional department identity claim. Finally, the agent finds a trust link from the regional identity claim to the weather station in question. Since all the trust links followed in this example are within the same organization, the subjectivity remains limited.

Distributed certification with trust and reputation

In the previous sections, we discussed what trust and reputation are in the context of Veritaa. In this section, we show how these concepts are used in Veritaa to certify identity claims.

For the certification, attackers must not be able to manipulate their own reputation. In Veritaa, each full node has an immutable copy of the GoT and is able to calculate each identity claim's reputation. Since its creator must sign each trust relationship, an attacker cannot forge trust relationships of other identity claims. However, attackers could form malicious collectives to receive more reputation. In order to prevent this attack, we use domain validation information. The domain validation information is used to initialize random walks and weight trust links. With this weighting, it is difficult for attackers to gain reputation.

To score the domains, we use the function shown in Equation (1) where $v \in V$ are all identity claims, m is the number of domain validated identity claims, $n = |V|$, and $rank(v) \in \{0, 1, \ldots, m\}$ is the ranking function that assigns a rank to each identity claim. The best-ranked domain is mapped to the lowest value. Multiple methods can be used to rank the domains. For example, the traffic of domains or an arbitrary list of the validator could be used.

$$score(v) = \begin{cases} \frac{1}{m}(m - rank(v)) \text{ if v is domain validated} \\ \frac{1}{n} \quad \text{else} \end{cases} \tag{1}$$

The *score* function is used to define the trust relations' weight and initialize the random walks of the different Eigenvector based algorithms. Therefore, we define C as the weighted adjacency matrix. Let A be the adjacency matrix of all identity claims $v \in V$ and trust links from the GoT. Equation (2) shows how C is defined.

$$c_{ij} = a_{ij} \cdot score(v_i) \tag{2}$$

With this weighted adjacency matrix C, the reputation is calculated. A reputation function is a function that maps the weighted adjacency matrix together with domain scoring to a vector that contains the reputation of all identity claims.

$$reputation : C \times \mathbb{R}^n \rightarrow \mathbb{R}^n \tag{3}$$

In this work, we apply the Eigenvector Centrality,[35] EigenTrust,[36] and PageRank[37] on the weighted adjacency matrix C and initialize the random walks with the score function.

The reputation distribution that results from the reputation function can be used for distributed certification, where the more influential nodes, according to the algorithm, have higher reputation. In the evaluation, we show that the algorithms can keep the reputation of attackers low (Section 5.5.2). Since all identity claims will get some reputation, a threshold is required to decide which identity claims are certified. For example, a reputation threshold could be defined, or it could be defined that only those identity claims are certified that have a higher reputation than a specific domain validated identity claim. The choice for the right threshold is at the users' discretion and depends on the risk of a wrong certification.

3.1.4 | Certificate revocation

Since the private key that belongs to an identity claim is not in the scope of Veritaa's security, it must be assumed that some keys are lost or compromised. Therefore, the GoT must be able to revoke broken certificates.

In non-DLT-based PKIs, broken certificates are usually revoked by revocation lists. These lists are difficult to maintain, and the time to disseminate the revocation information is long.[38] DLT-based PKIs have the advantage that certificate revocation can directly be stored on the ledger. In Veritaa, an owner of a private key can directly revoke the corresponding certificate on the DLT. Figure 2 shows entity A on the GoT that has posted a self-revocation to declare that the certificate is revoked. Each reader of the graph can directly recognize that this public key is not valid anymore.

3.1.5 | Signed declarations

Besides the public key management, Veritaa can also be used to immutably store signatures. By storing the signatures on the GoT, it is possible to determine the signing key's life-cycle state when the signature has been created. This has the advantage that the signature can still be validated when the key is deprecated.

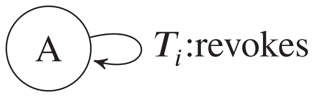

FIGURE 2 Self-revocation

On the GoT, different declarations can be signed. A declaration is a relation between two identity claims or an identity claim and a document identifier. The relation always starts from the signing identity claim and ends in another identity claim or a document identifier. The declarations have a type of a finite set. For example, the type of relations between two identity claims is in the set {*trusts*, *audits*, *validates*}, and the type of relations between an identity claim and a document identifier is in the set {*issues*, *approves*, *signs*}.

To enable signatures on objects that are not part of the GoT, a hash function is used to generate document identifiers. Since the hash of a document changes when the document is changed, the declarations are always for a specific version. Therefore, it is easy to verify if a document has been changed after a signed declaration.

Figure 3 shows an example where A_1 has issued three documents. A_2 declares by the approval of D_3 that it agrees with the contents of this document. Since A_3 has authenticated the identity claim of A_2 it can verify that A_2 has signed an approval declaration D_3.

3.2 | Acyclic block confirmation graph (ABCG)

To ensure the integrity of the GoT, it is of highest importance that all transactions can neither be reversed nor altered. Erroneous transactions must be transparently revoked so that the validator can comprehend all changes. Therefore, the database that stores the GoT must immutably store all transactions that build the GoT. Veritaa uses the acyclic block confirmation graph (ABCG) as DLT to immutably store the GoT in a peer-to-peer network. The ABCG is an application-specific BlockDAG based DLT optimized to store the append-only transactions to that build the GoT. Figure 4 shows an example ABCG.

3.2.1 | Transactions

The GoT is built out of append-only transactions that contain either an identity claim or a relationship between two identity claims or an identity claim and a document identifier. Each transaction that contains an identity claim is self-signed by the contained identity claim. The self-signature proves that a matching private key for the identity claim exists. Each transaction that contains a relationship is signed by the identity claim where the relationship starts. This

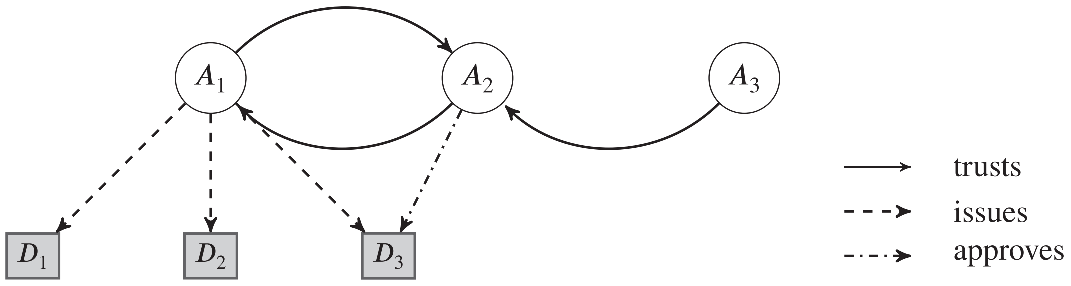

FIGURE 3 Document signing

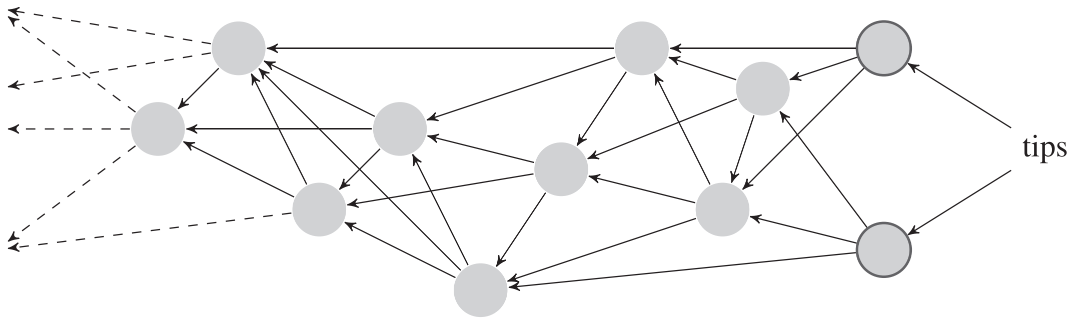

FIGURE 4 Acyclic block confirmation graph (ABCG)

ensures that only owners of the private key can create declarations for an identity claim. Since all network participants should be able to create new identity claims and each private key owner of an identity claim can declare and sign whatever it wants.

3.2.2 | Blocks

The ABCG is a BlockDAG where each block confirms three blocks. The advantage of the ABCG over a single blockchain is that new blocks do not necessarily need to be appended at the end of the ABCG. Consequently, the nodes can always commit new blocks, and the creators must not compete. When the honest nodes agree on always confirming the three blocks with locally the least confirmations, all blocks should be confirmed over time. A block with no confirmation at all is called a tip.[20] In a peer-to-peer network, the information does not propagate instantaneously, and therefore, the nodes are not always synchronized perfectly. Some nodes might have tips in their local copy of the DLT that are already confirmed on other nodes. Consequently, not all nodes share the same tips. The total number of tips in the network depends on the time required for a new block to disseminate through the whole network and the number of blocks confirmed by each new block.[20] Nevertheless, as long as the blocks are confirmed, the number of tips does not affect the operation of Veritaa.

Confirmation is of fundamental importance to ensure the non-repudiability and immutability of the transactions. The confirmation of a block is done by adding the hash of the confirmed block in the confirming block's header. As the hashes of the confirmed blocks are contained in the block's hash, it is secured that the previous blocks can not be changed. If a block in the ABCG was changed, its hash would change, and, therefore, the consecutive blocks would point to a non-existing block. A hash tree secures the immutability of the transactions in a block. Figure 5 shows an example of some blocks. Similar to the blocks, each transaction contains the hash of the previous transaction. The hash of the last transaction T_{Xn} is contained in the block hash and, therefore, the block hash would change if a single transaction was changed. This hash tree ensures that all transactions can not be altered after a block is confirmed.

3.2.3 | Consensus

In Veritaa, consensus is used to achieve the same state of the ABCG, and the GoT on all synchronized nodes in the presence of malicious or faulty nodes. Benevolent nodes only append and forward valid blocks to reach consensus. A block is valid if all of the following conditions are meet: all transactions and the hash tree in the block are valid; the block confirms at least three valid and no invalid blocks; the block is signed by an identity claim that exists in the local copy of the ABCG or the block(s) to add. For public Veritaa networks, a difficulty requirement (PoW) can be set to increase the computational cost to create blocks and protect the network from spam. Note that, since the nodes do not

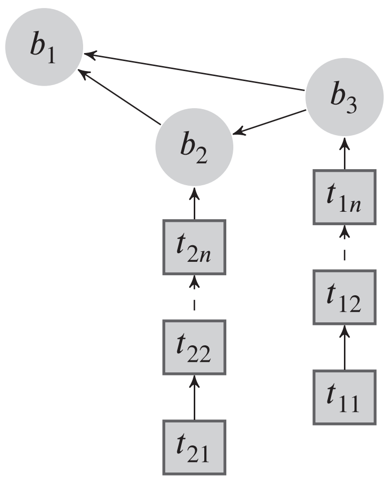

FIGURE 5 Hash tree

need to compete for the next block, this difficulty does not lead to an arms race. All transactions that either add an identity claim, a timestamp, or a declaration are valid.

Two nodes are synchronized when they have the same tips. When two nodes have the same tips, they have the same set of transactions in their local copy of the ABCG. The transactions contain the instructions to build the Graph of Trust. Since only transactions that either add a vertex or an edge to the Graph of Trust exist, the order of execution does not matter (For simplicity, add first all vertices and then the edges). Consequently, two synchronized nodes that apply all transactions result in the same state of the GoT.

3.3 | Time considerations

Veritaa with the Graph of Trust is responsible for the management and authentication of identity claims. Each identity claim has a life cycle that follows the following states: inception, authentication, certification, utilization, and deprecation. Several reasons can lead to the deprecation of a key pair. A key pair could be replaced due to its age, a lost private key, or when the private key has been compromised. Signatures that have been created with an identity claim are only valid if the identity claim was certified and not deprecated at the time of signing. It is essential to know the identity claim's state at the time of a signature's creation to verify its validity. Additionally, assigning a timestamp to each signature should be possible to map digital signatures with real-world events.

In the context of Veritaa, we can distinguish three different times: first, when an event happened in the real world; second, when the transaction related to this event is committed in a block to the ABCG; and third, the time when the block reached its finality (i.e., it was immutably written to the distributed ledger).

When a new block is committed to the Veritaa network, a gossip-based protocol is used to disseminate it. With gossip-based dissemination, data is spread exponentially fast through the network.[39] With this quick dissemination of new blocks, and all valid blocks accepted by benevolent nodes, the time between a new block has been committed to the network, and its finality is short. However, note that the time between a real-world event and its first occurrence on the ledger is hardly controllable by a distributed ledger.

In the ABCG, new blocks can be appended to all valid blocks. Therefore, new blocks can also be added to old blocks, and not only to tips. This property makes the protocol fault-tolerant and enables applications with no permanent or direct Internet connection, for example, behind firewalls, in low-power applications, or at very remote sites. However, the DAG structure of the ABCG does neither provide total order nor time information.

Block discovery services are used to enhance Veritaa with time information. A discovery service creates a timestamp in a configurable interval. Each discovery service can use its own interval, and each full node can run a discovery service.

A valid timestamp transaction is always signed by a valid identity claim found on the GoT, and its time is always greater than the time of its previous timestamp transaction. Additionally, all blocks with a timestamp transaction of a benevolent operator will always confirm all tips. By confirming all tips, the ABCG is divided into partitions, and each block can uniquely be assigned to the partition that belongs to a specific interval.

Each timestamp block can be considered as a snapshot. Since a benevolent discovery service confirms all tips, it is possible starting from a given timestamp to reach all valid blocks that have been in the local ABCG copy of the given discovery service at the given time. The breadth first search (BFS) algorithm can be used[40] to discover all blocks that can be reached from a given timestamp. The time of the first timestamp that discovered a block is the block's discovery time. The discovery time is used to sort the blocks.

Figure 6 shows how two example timestamp blocks partition the GoT. The two timestamps belong to the same discovery service. To make this example graph more readable, the blocks confirm only two previous blocks instead of three like in our implementation. The blocks with a black border are the tips at the time when the timestamp was created. The assignment of blocks to timestamps is simple. All blocks that can be reached from the timestamp i of the discovery service s are assigned to timestamp t_i^s if they are not already assigned to another timestamp t_j^s with $t_j^s < t_i^s$. With this algorithm, the green blocks belong to timestamp t_i^s, the blue to t_{i+1}^s and the red to t_{i+2}^s.

The blue interval also shows an example of a newly discovered block d that has been appended to two much older blocks. The reasons for this delay can be manifold. For example, the block was delivered with an opportunistic routing protocol, the creating node did not have a permanent Internet connection, or the block was intentionally appended to older blocks. However, this is no problem with the proposed algorithm, and block d will be assigned to the first timestamp after it has been discovered on the ABCG.

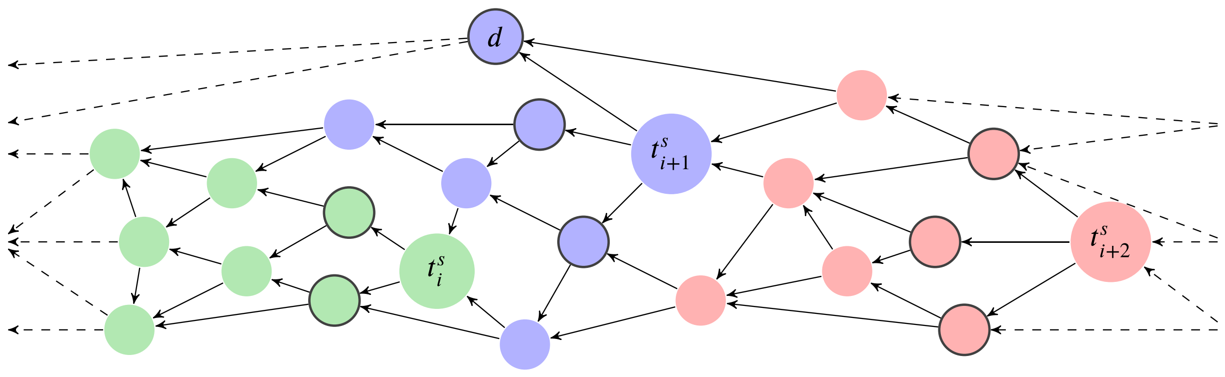

FIGURE 6 Fragmentation of the ABCG by a discovery service

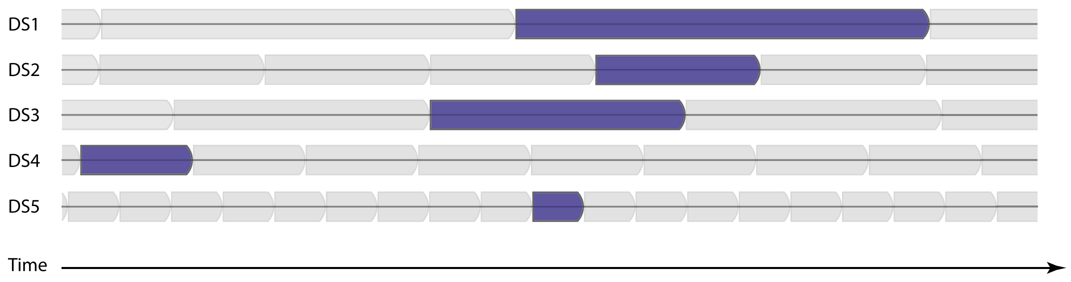

FIGURE 7 Time assignment

All full nodes of the Veritaa network can start a discovery service. With multiple discovery services in the network and all timestamps publicly available on the ABCG, each reader of the ABCG can assign each block uniquely to one interval of each discovery service. The different timestamps assigned to one block can be considered as a vote for when the network discovered the block.

Figure 7 shows an example of five discovery services. Each service uses a different discovery interval. For each discovery service, the interval when a certain block b has been discovered is highlighted. Due to inaccurate clocks, propagation delays, and even manipulation, the intervals might not always overlap. However, it is still possible to extract valuable time information.

To evaluate a block's discovery time, a reader calculates the median of the timestamps of the n most reputed discovery services. For example, let us assume $n = 3$ and DS1, DS2, DS4 are the most trusted discovery services in the example of Figure 7 then the timestamp of DS2 would be assigned to the block in question.

With the proposed approach, the blocks can be ordered and timestamped up to a specific resolution. The resolution depends on the interval length and the number of trusted discovery services. There is a tradeoff between the time resolution and the overhead generated by the timestamp required to achieve this resolution. By using different discovery intervals, the requirements for many applications can be met. For example, for the signature on a working contract, a daily timestamp might be sufficient, and for an auditable sensor infrastructure, a resolution below one minute might be useful. If absolute ordering of blocks is required, the GoT could be implemented on a general-purpose DLT at the cost of energy efficiency and flexibility.

4 | IMPLEMENTATION

Veritaa is a distributed system where several nodes exchange information over a peer-to-peer network. Figure 8 shows the architecture of a Veritaa node. Applications can access the core functionalities of Veritaa over the northbound (NB) consumer API. In the reference implementation, a web-application is used to operate the Veritaa node.

The northbound API is a REST API. In the core layer, the actions received by the NB API are executed on the GoT or the user management component. The user data and the private keys are not stored in the ledger and are not exchanged over the peer-to-peer network. Neo4j is used to store the data locally. Neo4j is a graph database optimized for storing and accessing data with graph structure.[41] The ABCG consists of a block scheme that references three previous blocks. These confirmations ensure the immutability of the distributed ledger.

We use the graph data science (GDS)[42] library of Neo4j and the efficient Java matrix library (EJML)[43] to calculate the reputation. For encryption and signatures, we have already tested and implemented a version with RSA and a version with ECDSA. As hash function we use SHA3-256.

The peer-to-peer layer is used to exchange blocks with peers. The peer-to-peer layer consists of peer discovery, peer management, and data exchange components. The peer-to-peer network is unstructured and was implemented with Java IO and UDP datagrams. Each node has a predefined list of known hosts. This list is used to connect to the peer-to-peer network. At startup, a node selects one host from the list of known hosts and requests new peers. The node then connects to some of these received nodes. By continuously discovering the neighborhood, each node maintains a set of peers. The peers are selected by their availability and round trip time.

5 | EVALUATION

5.1 | Testbed

To evaluate Veritaa and all its components, a testbed was created. If not other mentioned, the testbed was deployed to a single-station Kubernetes server with 128GB RAM and an Intel Xeon E5-1650V2. It was possible to deploy up to

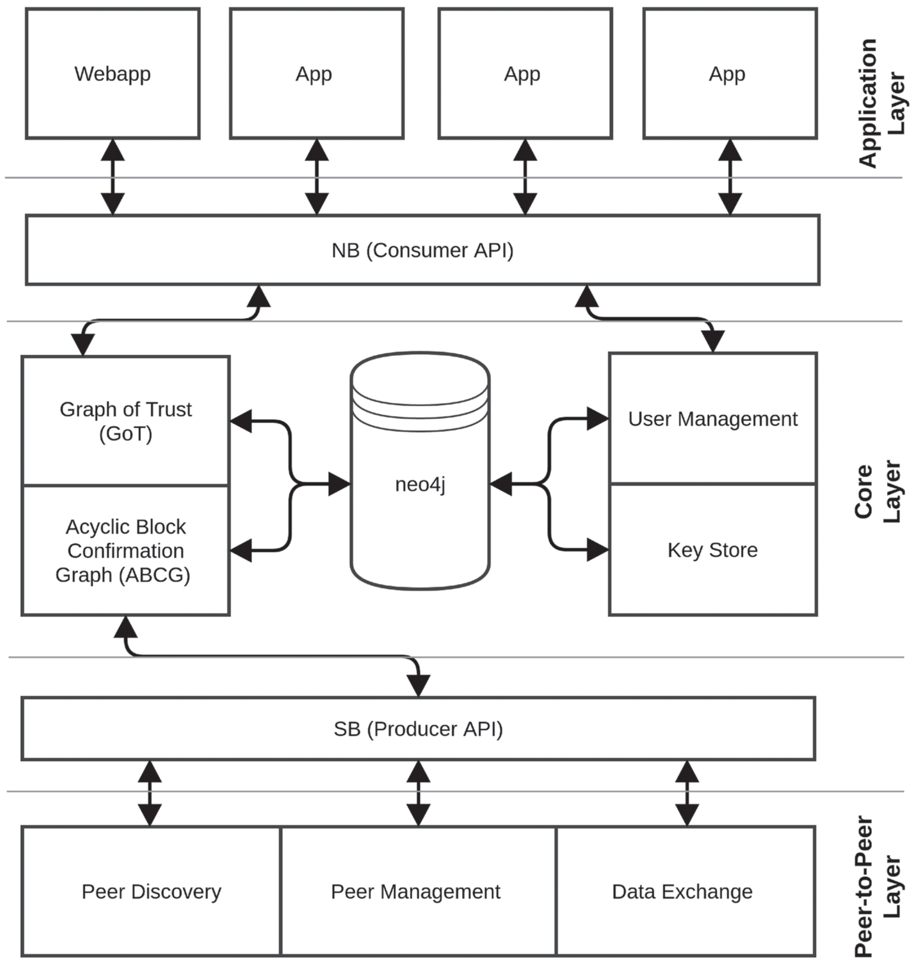

FIGURE 8 Architecture

70 Veritaa nodes on this setup, but bigger networks are possible by vertically scaling the Kubernetes cluster. Figure 9 shows the testbed consisting of a Kubernetes cluster that runs multiple Veritaa nodes as pods. A specialized root pod is used to initialize the ABCG and acts as the known host. Multiple other nodes are created by replication.

When the testbed is started, the root pod initializes the ABCG, and afterwards, the other pods are started. To connect to the network, pods are requesting an initial set of peers from the root pod, and subsequently, more peers are requested from existing peers.

Each pod contains an activity simulation component. This component is used to generate traffic in Veritaa. To generate traffic, nodes create and sign random transactions and post them to the Veritaa network.

5.2 | Stability

To measure the network's stability, the number of tips, and the clustering coefficient[44] of the peer-to-peer network can be analyzed. Tips are blocks that are not confirmed yet. The clustering coefficient provides information about how well the peer-to-peer network is connected. The number of tips in the network depends on the network's connectivity, the propagation delay of new blocks in the network, the number of confirmations per block, and the block creation rate. In a congested network, the number of tips will increase as other nodes do not know about the existence of new blocks and, therefore, confirm blocks that are already confirmed on other nodes.

To evaluate the stability of a distributed system, the simulation of joining and leaving (churn) nodes is an interesting experiment. In the following experiment, 25 Veritaa nodes were deployed in the testbed. Each node maintains a set of six peers and creates a block every 100 ms. After 500 s, 25 additional nodes were started, and after 1000 s, they were removed again. Figure 10 shows the average clustering coefficient[44] in the network. When the 25 additional nodes are added, the clustering coefficient drops because the nodes connect to the well-known node, which forms a cluster around this node. As soon as the additional nodes have received their peers from the well-known node, they connect to the peers, and the cluster dissolves. The impact of the removal (churn) of 25 nodes after 1000 s can be seen but has only a little impact on the average cluster coefficient as nodes are randomly removed, and all nodes are well connected with six peers out of 50.

The response of the number of tips to the joining and leaving nodes shown in Figure 10 is also notable. The blue line represents the average, and the grey represents the maximum and the minimum number of tips in the network (due to propagation delays, the nodes might not see the same tips). Initially, the root pod is started, and it initializes the network. Then, the initial 25 nodes join the network. Since the global ABCG is empty at this point, no blocks need to be synchronized. When the 25 additional nodes join after 500 s, they first need to create their set of peers and synchronize with the network. To synchronize with the network, the nodes need to load all blocks of the ABCG from their peers. During the synchronization process, blocks are added to the local ABCG of a node. These blocks actually already have been confirmed on other nodes, but the confirming blocks are not loaded yet. These locally unconfirmed blocks result in a high number of tips during the synchronization process. After synchronizing the newly connected nodes, the average number of tips goes down and remains slightly higher than before the nodes were joining. The higher tip count is the result of a higher propagation delay due to the increased network size. After we removed the 25 nodes, the average number of tips falls back to the level before the nodes' addition. This experiment shows that Veritaa is robust and resilient against heavy network change.

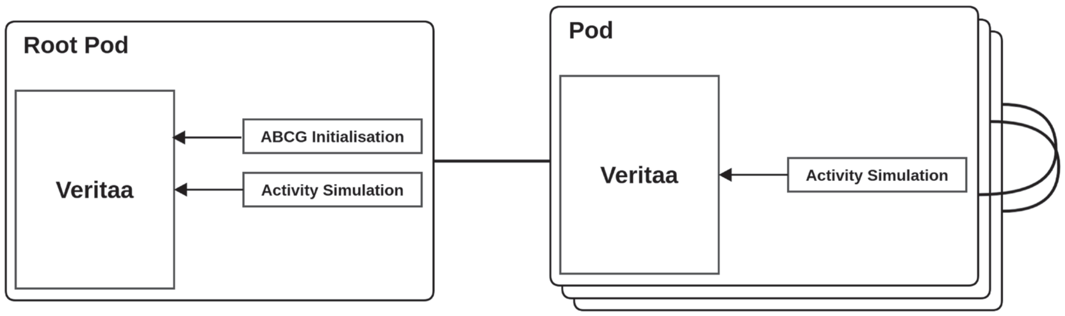

FIGURE 9 Testbed

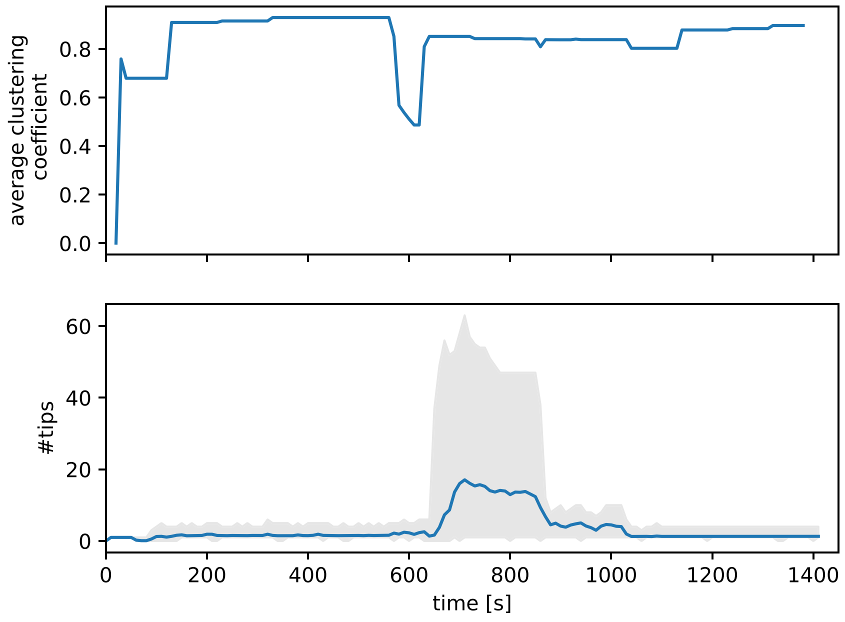

FIGURE 10 Churn—average cluster coefficient when nodes are joining and leaving

5.3 | Throughput

The number of transactions that can be handled by a distributed ledger defines the limiting factor of its scalability. In peer-to-peer applications like Veritaa, all nodes can create blocks at any time, and every node needs to be able to process and store all new blocks. Therefore, the potential computational capability for producing new blocks is much bigger than the consuming one. Since creating a new block in Veritaa is computationally cheap, the computational costs to create a new block are not essentially more expensive than validating a block. Consequently, the maximum number of incoming transactions that can be processed by a node forms an upper bound for the throughput. In order to measure this upper bound, a consumer and a producer node with four CPU cores each were deployed to the testbed. The producer node generates new blocks that are sent to the consumer node. The throughput was measured on the consumer node.

Figure 11 shows how much time was spent in the different processing states when a packet is received. Most time is required for processing the block content. When the block content is processed, the GoT is updated according to the transactions in the block. To update the GoT, existing entities need to be read from the database. These reads are computationally relatively expensive, but the read time can be optimized if multiple document identifiers are searched, and, therefore, the database read overhead per transaction becomes smaller when the block contains more transactions. The same accounts for writing the changes into the database. The validation process of the transactions only requires a small amount of time. As all transactions are validated separately, the validation time per transaction does not decrease much, when more transactions are contained in a block. The time required to parse a block is minimal and can be neglected when the block size is increased.

Figure 12 shows the throughput of Veritaa in relation to the block size. The blue line represents the average and the grey line indicates the standard error. For a block size smaller than 1000 transactions, the throughput is decreasing due to overhead. However, above 1000 transactions per block, the throughput is around 7000 tps. The fluctuations mainly result from the database access speed that depends on the data to store.

5.4 | Block discovery services

In this section, we evaluate the block discovery services. For this evaluation, we have deployed Veritaa nodes with block discovery services enabled to a Kubernetes cluster distributed over four data centers in Finland, Germany, and Switzerland. In this experiment, we evaluate the discovery delay with different settings. The discovery delay is the time

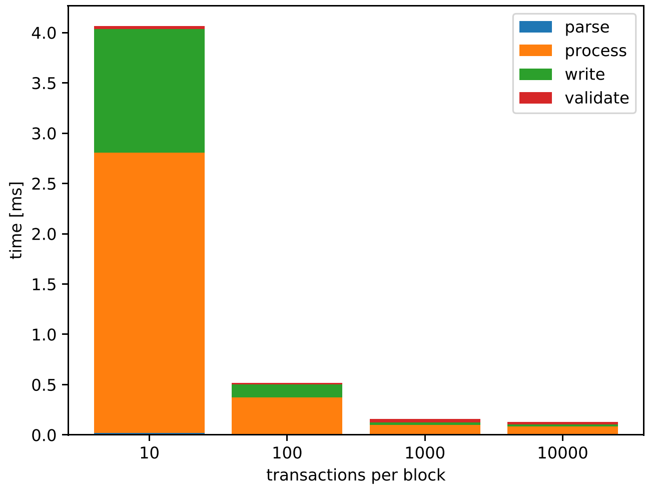

FIGURE 11 Average time to process a transaction

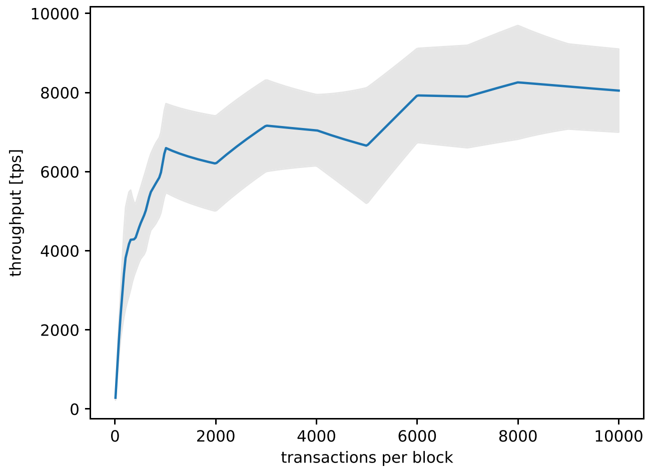

FIGURE 12 Throughput [tps] in relation to the block size

between the creation of a block and its discovery in the network by block discovery services. Each block discovery service has a unique timestamp that first discovers a given block, and from all these timestamps, the discovery time is calculated. This experiment compares the minimum, median, and maximum functions to derive the discovery time from all timestamps that first discovered a given block. In this experiment, all block discovery services have the same timestamp interval. The timestamp interval is the time between the creation of two timestamps by the same block discovery service.

Figure 13A shows the discovery delay with a 60 s timestamp interval, and Figure 13B shows the discovery delay with a 600 s timestamp interval. The median function is the most robust against attacks and has an average discovery delay of approximately the half interval length. The maximum function is the timestamp of the block discovery service that has discovered a given block the latest. Consequently, with the maximum function, the mean of all discovery times is close to the interval length. Since the minimum function returns the first timestamp that discovered a block, the discovery delay is close to zero.

5.5 | Security analysis

Adversaries might attempt to attack the peer-to-peer network, the ABCG, or the GoT of Veritaa. The goal of an adversary attacking Veritaa might be: manipulate the GoT in order to gain a higher reputation; abuse a compromised

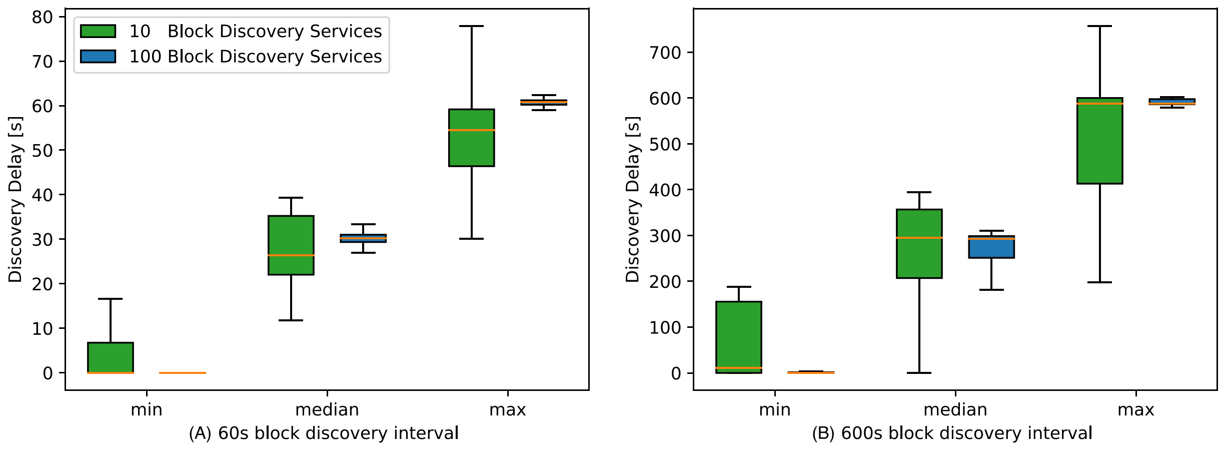

FIGURE 13 Delays until blocks are discovered

key; and interfere with the operation of the DPKI. We assume that the used cryptographic functions are secure (hash functions and signatures). We also assume that each benevolent node can connect to the peer-to-peer network containing other benevolent nodes. Some of the benevolent nodes operate a block discovery service. Most but not all nodes are honest but curious. An adversary can eavesdrop, modify, and forge transactions. Note that an attacked node can quickly check if a block is valid (Section 3.2.3).

5.5.1 | DDoS

An attacker might launch a DDoS attack against some nodes in the peer-to-peer network to interfere with the operation of the PKI. However, while the adversary might successfully take some nodes out of operation, it is unlikely to successfully attack the whole network. Since each node maintains a replica of the DLT, the PKI remains operational.

An adversary could also try to interfere with the operation of the PKI on the ABCG layer. In an unpermissioned network, an adversary might create a vast amount of valid but nonsense transactions and send them to the network. Since all Veritaa nodes maintain a replica of the ABCG, all full nodes would require adding these transactions. The resources for adding and storing those nonsense transactions would increase the operational costs. In order to prevent spam in unpermissioned networks, it is possible to add a proof-of-work requirement to blocks. With this PoW requirement, blocks are only confirmed by benevolent nodes if they fulfill a particular difficulty. Thus, the PoW can increase the cost of creating spam to a level where the cost of creating spam is higher than its benefits. Note that this PoW requirement does not lead to an arms race since there exists no competition for mining the next block. In a permissioned network, it is possible to revoke the authorization of suspicious nodes.

5.5.2 | Reputation

In this work, we propose to use domain validation information together with trust declarations to build a reputation system for the certification of identity claims. A reputation algorithm assigns each identity claim a value so that the most reputed identity claim has the highest and the least reputed has the lowest value.

To calculate all identity claims' reputation, we use the well-known Eigenvector Centrality,[35] EigenTrust,[36] and PageRank[37] algorithms. These algorithms basically represent a random surfer that traverses the graph and returns the probabilities in which state the surfer most likely is. In a directed graph, the random surfer might be locked in a cluster with no way out, and, therefore, PageRank and EigenTrust use a probability to start over at a random node at each step.

Attackers might try to build malicious collectives to manipulate and improve their reputation and certify illicit identity claims. This section evaluates if such a reputation function can be used to prevent these kinds of attacks.

Figure 14 shows an example with a collective of five attackers (red) that try to manipulate the reputation by voting for each other, the cluster with the benevolent identity claims contains two domain validated identity claims (green)

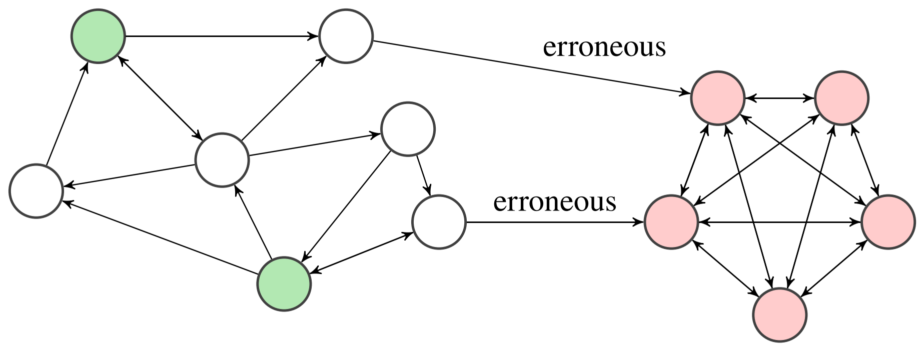

FIGURE 14 An example Graph of Trust with a group of attackers

and two identity claims of the benevolent cluster have signed an erroneous trust link towards the attackers. Since they want to trap the random walker to get more reputation, the attackers do not create trust links towards the benevolent cluster.

To improve the resilience against attacks, we initialize the centrality algorithms with the domain scoring vector, weight the edges according to the domain scoring, and use the domain scoring for a probability where a node starts over. Consequently, the random walker is more likely to start at domain validated identity claims, is more likely to remain around them, and by chance also more likely starts over at domain validated identity claims.

Figure 15 shows example reputation distributions for Eigenvector, PageRank, and EigenTrust algorithm executed on a GoT with 100 identity claims of which 10 are domain validated (blue) and 10 are attackers (red). In this example, five erroneous trust links have been created towards the attacking cluster. The example in Figure 15 shows that the proposed reputation function based on the trust declaration and the domain validation information can keep the reputation of the attackers low while the domain validated nodes have a higher reputation. It also shows that they do not perform equally. An identity claim can have a large Eigenvector centrality if it has important or many neighbors. Therefore, the domain validated identity claims might not be found in the top ranks. With the PageRank and the EigenTrust algorithm, the domain validated identity claims are ranked higher, but some attacking identity claims could also gain higher ranks. To measure and compare each centrality algorithm's quality, we define a rank error and a certification quantity function.

The *rank_error* function (4) calculates the rank error, where U is the set of attacking identity claims, $m = |U|$ is the number of the attacking identity claims, $n = |V|$ is the total number of identity claims, and $rank(v)$ is the function that returns the rank of each identity claim according to its reputation. The best rank is 0, and the least is $n-1$. The *rank_error* results in a value between zero and one where one is returned when the attackers occupy the m top ranks and zero when they occupy the m lowest ranks.

$$rank\_error(U,m,n) = \frac{1}{m(n-m)} \sum_{u \in U} (n - rank(u)) - \frac{1}{2} m(m+1) \tag{4}$$

The *certification_quantity* function (5) measures how many identity claims are certified by one domain validated identity claim. U is the set of attacking identity claims, W is the set of domain validated identity claims, and $rank(v)$ is the function that returns the rank of each identity claim according to its reputation. Note that the criteria for the certification might differ by application or user requirements. In our experiments, we evaluate all identity claims with a better reputation than the best-ranked attacker as certified. Other certification strategies might be the worst-ranked domain validated identity claim or a reputation threshold.

$$certification\_quantity(U,W) = \frac{1}{|W|} min rank(u) \tag{5}$$

We conducted two experiments with the python NetworkX[45] library to test the reputation functions' quality. We generated graphs with the powerlaw_cluster_graph generator of NetworkX. For each measurement, we have generated 50 random graphs and show the average of the results. In the first experiment, we vary the number of attacking identity

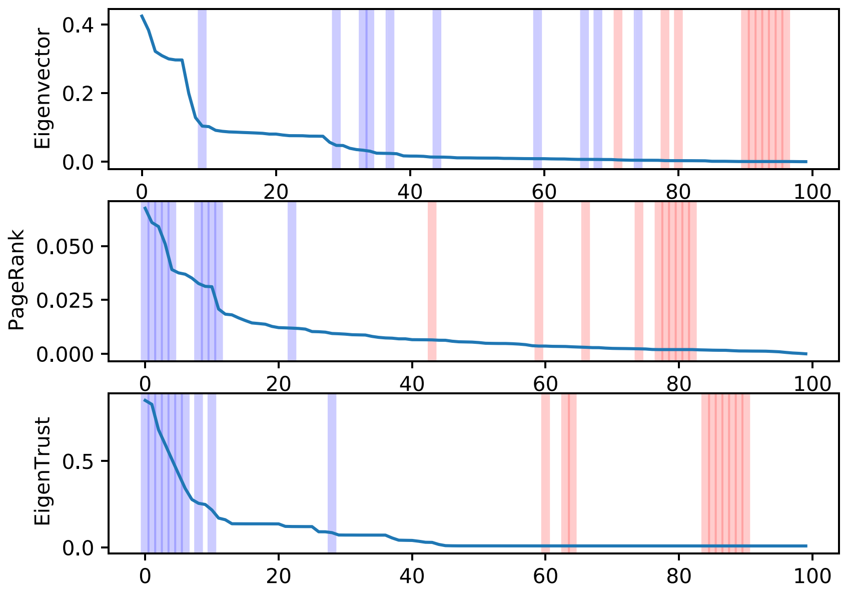

FIGURE 15 Reputation distribution

claims while the number of domain validated identity claims is fixed to 10, and the normal identity claims to 100. The number of erroneous trust links is $0.1 \cdot |U|$.

Figure 16A shows the rank error and the certification quantity for the different attacker cluster sizes. These results show a low and stable rank error with a high certification quantity for the PageRank algorithm. The number of attacking identity claims does not show a relevant impact on the quality of the PageRank. EigenTrust and Eigenvector do not perform as well as the PageRank algorithm, but up to one-third of the identity claims as attackers, these algorithms can still certify around 50% of the identity claims.

In the second experiment, we vary the number of erroneous trust links from the benevolent cluster towards the attackers while the number of attackers is fixed to 10, the number of domain validated identity claims are set to 10, and the normal identity claims are 100. Figure 16B shows the result of this second experiment. Here, the PageRank performs a bit worse than EigenTrust and Eigenvector Centrality, and for the certification quantity, they perform almost the same. With 20 erroneous trust links, each attacker has got, on average, two reputation votes from the benevolent cluster. The relatively low rank error and the decreasing certification quantity show that most attackers are still in the lower half of the reputation ranking. This shows that the cluster with the domain validated identity claims remains more influential even if many members of the benevolent cluster have erroneously created trust links to attackers. Note that a creator of an erroneous trust link is always able to revoke this again, and, therefore, it should not happen that an unauthenticated attacking cluster gains so many trust votes. At some point, when a cluster has a lot of incoming trust links, it might also just be trusted.

The results show that all evaluated reputation algorithms perform pretty well in avoiding a malicious collective gaining reputation. Therefore, reputation can be used to certify the well-reputed identity claims. Nevertheless, the certification threshold must be defined by the user and depends on the application. Depending on the certification threshold, some legitimate identity claims will not have enough reputation to be certified. However, if they are closely connected over trust-declarations with a certified and well-reputed identity claim, they might be certified over this certification chain.

5.5.3 | Block discovery service

In order to claim that a signature was created at another time, an attacker might try to manipulate the block discovery times created by the Block Discovery Services. However, when the median is used to calculate the block discovery time, the attacker needs to control more than 50% of the n most reputed discovery services. Since it is likely that the central nodes with the highest reputation also operate timestamp services, the attacker would require to compete for reputation against half of the most reputed nodes in the network.

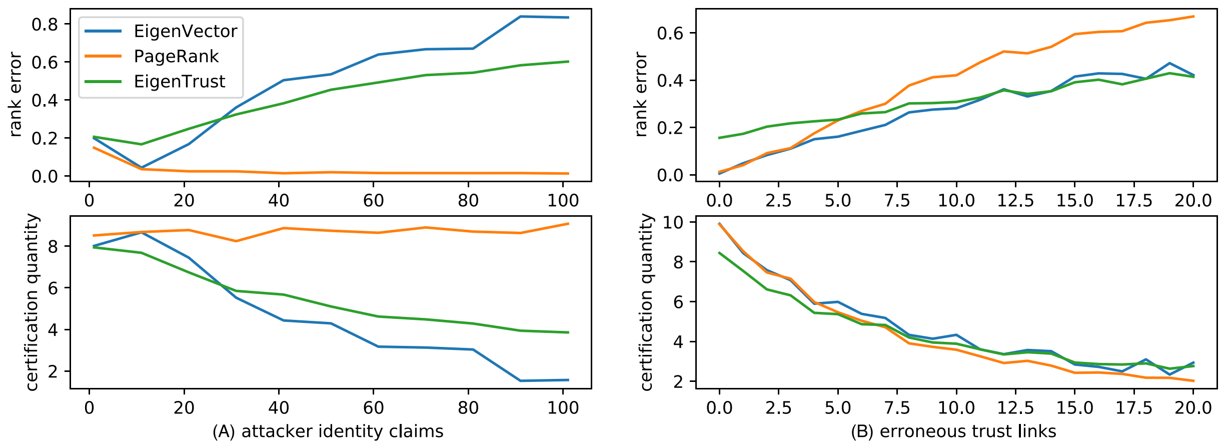

FIGURE 16 Quality of reputation in relation to number erroneous trust links and attacker cluster size

5.5.4 | Split-world

A split-world attack is an attack where an adversary tries to show only a fraction of the GoT to a node. For example, an adversary could block a revocation transaction from reaching a node to abuse a compromised key. In order to do so, the adversary would require to create malicious nodes that do not forward the block with the revocation transaction and ensure that the attacked node only has malicious nodes as peers. Nevertheless, the attacked node can continually update its peers, and since some benevolent nodes exist in the network, it eventually will synchronize with a benevolent node and receive the block.

5.5.5 | Manipulation of the graph of trust

An adversary might try to manipulate the Graph of Trust. For example, to delete an immutably stored signature. While it is always possible for a private key holder to transparently revoke its signed transactions, it must not be possible to delete or change them. Each node builds the GoT out of the transactions stored in the ABCG. Therefore, an adversary would require to change a block in the ABCG to change or remove a transaction. Let us assume that an adversary has altered or removed a transaction contained in block B. Then the hash of the block changes. A block with a new hash is like a new block B_m, and the blocks that have confirmed this manipulated block would still refer to B. At least the nodes that have confirmed the original block would still have B in their replica of the ABCG, and other nodes can restore it from there. Consequently, it is not possible to manipulate or delete blocks from the ABCG.

6 | CONCLUSION AND FUTURE WORK

In this work, we discussed the architecture of Veritaa. The Graph of Trust is used to store and represent identity claims and signed declarations. The acyclic block confirmation graph is an application-specific BlockDAG optimized for immutably storing and replicating graph transactions. The ABCG asserts that the local GoT copies of all Veritaa members are in the same state.

We showed how signed trust relations and domain validation information of the GoT can build a reputation system and how this reputation can be used to derive certification. Additionally, we showed how Block Discovery Services are used to create distributed timestamps of blocks. For some applications, the missing total order of transactions in the ABCG might be a limitation. However, for applications that require a total order of transactions, the GoT could be implemented on a general-purpose DLT with the drawback of higher complexity and lower efficiency. In WoT-based PKI, the participants are required to reveal their trust relations. The public declaration of trust relations might be a limiting factor in privacy-sensitive environments. Nevertheless, well-reputed organizations could operate as CAs in such environments, and the privacy-preserving agents must not publish any trust declarations.

In this work, we motivated that reputation algorithms can derive certification information from the GoT. However, the GoT also contains information about how reputation evolves. Future work could evaluate if this evolution of reputation can be used to reason about the validity of identity claims.

DATA AVAILABILITY STATEMENT

Research data are not shared.

ORCID

Jakob Schaerer https://orcid.org/0000-0002-7166-1303
Torsten Braun https://orcid.org/0000-0001-5968-7108

REFERENCES

1. Cooper M, Dzambasow Y, Hesse P, Joseph S, Nicholas R. Internet X.509 Public Key Infrastructure: Certification Path Building. RFC 4158, RFC Editor; 2005
2. Yakubov A, Shbair WM, Wallbom A, Sanda D, State R. A blockchain-based PKI management framework. In: *2018 IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*. Taipei, Taiwan: IEEE; 2018:1-6
3. Eckersley P, Burns J. The EFF SSL Observatory. 2010. https://www.eff.org/observatory. Accessed Dezember 09, 2019.
4. Cert Spotter - Timeline of PKI Security Failures. https://sslmate.com/certspotter/failures. Accessed December 7, 2019.
5. Prins JR. DigiNotar certificate authority breach 'Operation black tulip'. tech. rep., 2011.
6. Zimmermann P. *The official PGP user's guide*. Cambridge, Mass: MIT Press; 1995.
7. Maurer U. Modelling a public-key infrastructure. In: Bertino E, Kurth H, Martella G, Montolivo E, eds. *Computer Security - ESORICS 96 Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer; 1996:325-350.
8. Caronni G. Walking the Web of trust. In: *9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2000)*. Gaithersburg, MD, USA: IEEE Computer Society; 2000:153-158
9. Hepp T, Spaeh F, Schoenhals A, Ehret P, Gipp B. Exploring Potentials and Challenges of Blockchain-based Public Key Infrastructures. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Paris, France: IEEE; 2019: 847-852.
10. Schaerer J, Zumbrunn S, Braun T. Veritaa - The Graph of Trust. In: *2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. Paris, France: IEEE; 2020:168-175.
11. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System. tech. rep., 2009. https://bitcoin.org/bitcoin.pdf. Accessed November 24, 2019.
12. Nguyen CT, Hoang DT, Nguyen DN, Niyato D, Nguyen HT, Dutkiewicz E. Proof-of-stake consensus mechanisms for future blockchain networks: Fundamentals, applications and opportunities. *IEEE Access*. 2019;7:85727-85745. Conference Name: IEEE Access. https://doi.org/10.1109/ACCESS.2019.2925010
13. Vries DA. Renewable energy will not solve Bitcoin's sustainability problem. *ET J*. 2019;3(4):893-898. https://doi.org/10.1016/j.joule.2019.02.007
14. Vranken H. Sustainability of bitcoin and blockchains. *Curr Opin Environ Sustain*. 2017;28:1-9. https://doi.org/10.1016/j.cosust.2017.04.011
15. Malone D, O'Dwyer K. Bitcoin Mining and its Energy Footprint. In: *Institution of Engineering and Technology*. Ireland: Limerick; 2014: 280-285.
16. Stoll C, Klaaßen L, Gallersdörfer U. The Carbon Footprint of Bitcoin. *Joule*. 2019;3(7):1647-1661. https://doi.org/10.1016/j.joule.2019.05.012
17. Croman K, Decker C, Eyal I, et al. On Scaling Decentralized Blockchains. In: Clark J, Meiklejohn S, Ryan PY, Wallach D, Brenner M, Rohloff K, eds. *Financial Cryptography and Data Security Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer; 2016:106-125.
18. Wood DG. Ethereum: A secure decentralised generalised transaction ledger. tech. rep., 2014. http://gavwood.com/Paper.pdf. Accessed November 25, 2019.
19. Ali MS, Vecchio M, Pincheira M, Dolui K, Antonelli F, Rehmani MH. Applications of Blockchains in the Internet of Things: A Comprehensive Survey. *IEEE Commun Surv Tutorials*. 2019;21(2):1676-1717. https://doi.org/10.1109/COMST.2018.2886932
20. Popov S. The Tangle. tech. rep., 2019. https://iota.org/IOTAWhitepaper.pdf. Accessed November 18, 2019.
21. Popov S, Moog H, Camargo D, et al. The coordicide. tech. rep., 2020. https://iota.org/IOTAWhitepaper.pdf. Accessed July 03, 2021.
22. Singla A, Bertino E. Blockchain-Based PKI Solutions for IoT. In: *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. Philadelphia, PA: IEEE; 2018:9-15.
23. Kubilay MY, Kiraz MS, Mantar HA. CertLedger: A new PKI model with Certificate Transparency based on blockchain. *Comput Secur*. 2019;85:333-352. https://doi.org/10.1016/j.cose.2019.05.013
24. Papageorgiou A, Mygiakis A, Loupos K, Krousarlis T. DPKI: A Blockchain-Based Decentralized Public Key Infrastructure System. In: *2020 IEEE Global Internet of Things Summit (GIoTS)*. Dublin, Ireland: IEEE; 2020:1-5.
25. Yao S, Chen J, He K, Du R, Zhu T, Chen X. PBCert: Privacy-preserving blockchain-based certificate status validation toward mass storage management. *IEEE Access*. 2019;7:6117-6128. https://doi.org/10.1109/ACCESS.2018.2889898

26. Wilson D, Ateniese G. From Pretty Good to Great: Enhancing PGP Using Bitcoin and the Blockchain. In: Qiu M, Xu S, Yung M, Zhang H, eds. *Network and System Security Lecture Notes in Computer Science*. Cham: Springer International Publishing; 2015:368-375.
27. Al-Bassam M. SCPKI: A Smart Contract-based PKI and Identity System. In: *ACM Asia Conference on Computer and Communications Security (ASIA CCS 2017)*. Abu Dhabi United Arab Emirates: ACM; 2017:35-40.
28. Nottingham M. Well-Known Uniform Resource Identifiers (URIs). Tech. Rep. RFC8615, RFC Editor; 2019
29. trust. In: Merriam-Webster.com. 2020. https://www.merriam-webster.com. Accessed December 15, 2020.
30. Gambetta D. Can we trust trust. *Trust*. 2000;13:213-237.
31. Abdul-Rahman A, Hailes S. A distributed trust model. In: *New Security Paradigms Workshop (NSPW 1997)*. Langdale, Cumbria, United Kingdom: ACM Press; 1997:48-60.
32. reputation. In: Merriam-Webster.com; 2020. https://www.merriam-webster.com. Accessed December 15, 2020.
33. Kleinberg JM. Authoritative sources in a hyperlinked environment. *J ACM (JACM)*. 1999;46(5):604-632. Publisher: ACM New York, NY, USA
34. Freeman LC. Centrality in social networks conceptual clarification. *Soc Networks*. 1978;1(3):215-239. https://doi.org/10.1016/0378-8733(78)90021-7
35. Bonacich P. Power and centrality: A family of measures. *Am J Sociol*. 1987;92(5):1170-1182. Publisher: University of Chicago Press
36. Kamvar SD, Schlosser MT, Garcia-Molina H. The Eigentrust algorithm for reputation management in P2P networks. In: *Proceedings of the twelfth international conference on World Wide Web (WWW 2003)*. Budapest, Hungary: ACM Press; 2003:640.
37. Page L, Brin S, Motwani R, Winograd T. The PageRank citation ranking: Bringing order to the web. tech. rep., Stanford InfoLab; 1999.
38. Liu Y, Tome W, Zhang L, et al. An End-to-End Measurement of Certificate Revocation in the Web's PKI. In: *Proceedings of the 2015 Internet Measurement Conference (IMC 2015)*. Tokyo Japan: ACM; 2015:183-196
39. Kermarrec AM. Steen vM. Gossiping in distributed systems. *ACM SIGOPS Operat Syst Rev*. 2007;41(5):2-7. https://doi.org/10.1145/1317379.1317381
40. Bundy A, Wallen L. Breadth-First Search. In: Bundy A, Wallen L, eds. *Catalogue of Artificial Intelligence Tools Symbolic Computation*. Berlin, Heidelberg: Springer; 1984:13-13.
41. Robinson I, Webber J, Eifrem E. *Graph databases - new opportunities for connected data*. Sebastopol, CA: O'Reilly; 2015.
42. Needham M, Hodler AE. *Graph algorithms: practical examples in Apache Spark and Neo4j*. 1st ed. Sebastopol, California, O'Reilly Media; 2019 OCLC: on1066191517.
43. Peter Abeles. Efficient Java Matrix Library; 2020. http://ejml.org. Accessed January 02, 2021.
44. Latapy M, Magnien C, Vecchio ND. Basic notions for the analysis of large two-mode networks. *Soc Networks*. 2008;30(1):31-48. https://doi.org/10.1016/j.socnet.2007.04.006
45. Hagberg AA, Schult DA, Swart PJ. Exploring Network Structure, Dynamics, and Function using NetworkX. In: Varoquaux G, Vaught T, Millman J, eds. *Proceedings of the 7th Python in Science Conference*. Pasadena, CA USA; 2008:11-15.

AUTHOR BIOGRAPHIES

Jakob Schaerer is currently a PhD student at the University of Bern, Switzerland. He is a co-founder of the technology company Abilium GmbH. He received his MSc in Computer Science in 2018 from the University of Bern, Switzerland.

Severin Zumbrunn is a co-founder of the technology company Abilium GmbH. He received his MSc in Computer Science in 2019 from the University of Bern, Switzerland.

Torsten Braun received the PhD degree from the University of Karlsruhe, Germany, in 1993. Since 1998, he has been a Full Professor with the Department of Computer Science, University of Bern. He has also been the Vice President of the SWITCH (Swiss Research and Education Network Provider) Foundation since 2011. He was a recipient of the best paper awards from the LCN 2001, the WWIC 2007, the EE-LSDS 2013, the WMNC 2014, the ARMSCC 2014 Workshop, and the GI-KuVS Communications Software Award in 2009.

How to cite this article: Schaerer J, Zumbrunn S, Braun T. Veritaa: A distributed public key infrastructure with signature store. *Int J Network Mgmt*. 2021;e2183. doi:10.1002/nem.2183