Universal Identifier Theory[1]

Samuel M. Smith Ph.D.

v1.32 2020/10/23

Abstract—A universal theory for identifiers is presented. This theory is based on a unified model of identifiers that include cryptographic autonomic identifiers (AIDs) and legitimized (authorized) human meaningful identifiers (LIDs). This model provides truly decentralized trust bases each derived from the cryptographic root-of-trust of a given AID. An AID is based on a self-certifying identifier (SCID) prefix. Self certifying identifiers are not human meaningful but have strong cryptographic properties. The associated self-certifying trust basis gives rise to a trust domain for associated cryptographically verifiable non-repudiable statements. Every other type of identifier including human meaningful identifiers may then be secured in this resultant trust domain via an end-verifiable authorization. This authorization legitimizes that human meaningful identifier as an LID though its association with an AID. The result is a secured trust domain specific identifier couplet of $aid|lid$. AIDs are provided by the open standard key event receipt infrastructure (KERI) [42]. This unified model provides a systematic methodology for the design and implementation of secure decentralized identifier systems that underpin decentralized trust bases and their associated ecosystems of interactions.

Index Terms—Autonomic Identifier, AID, Autonomic Identity System, AIS, Autonomic Namespace. Decentralized, Key Management Infrastructure, DKMI, Key, Event, Receipts, Pre-rotation, Rotation, Event, Streaming, KERI, KERL, KEL, KA2CE, Post-Quantum Security, Self-Sovereign Identity, SSI.

1 Introduction

A design goal for an identifier system is that it be secure, trustable, and universal. A system that is insecure may not be trusted, and a system that is neither secure nor trusted is not a viable candidate for universal use. Therefore security is of paramount importance. It is the foundation for any other useful property of the identifier system.

1.1 Unified Model of Identifiers

We use the term identifier system instead of identity system to emphasize that it is the properties of the identifier that are first and foremost. The identifier properties imbue the associated identity system with its operative properties. Indeed, we use a narrow definition of identity to crystallize our understanding of the associated system. We formally define a (digital) identity to be a (digital) identifier plus attributes. The attributes themselves may include identifiers thereby enabling complex identities which may be represented by an identity graph. A subset of a graph is a tree or a forest.

In general we want to have a theory or model of how to design identifier systems with desirable properties. As mentioned above, some desirable properties are security, trustability, and universality. Historically identifier systems has been designed without any unifying theory to guide the designers. This means that many identifier systems are largely bespoke to a given application. This may be problematic especially when the identifier system does not have a well defined approach to security. More recently, explicitly characterizing identifier properties has gained some recognition as a prerequisite to identifier system design. For example, the well known Zooko's triangle or *trilemma* is based on recognizing a conflict between three different desirable

1. https://keri.one

properties for an identifier [61]. The three properties are *human meaningful*, *secure*, and *decentralized*. The *trilemma* states that an identifier may have any two of the three properties by not all three. Some have claimed to have solved the trilemma by using a hybrid identifier system where human meaningful but insecure identifiers may be registered on a distributed consensus ledger. The registration process itself uses identifiers that are secure and decentralized but not human meaningful [50]. The fact that not one identifier but a combination of identifiers is needed to solve the trilemma hints that the identifier model therein employed is incomplete. A notional diagram of Zooko's triangle is shown below:

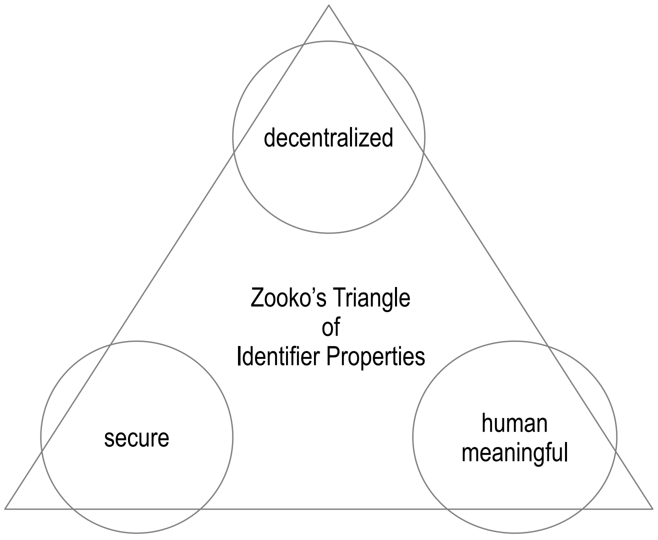

Figure 1.1. Zooko's Triangle

In our work we have developed a unified universal model of identifiers (digital) called the *autonomic identifier model* (AID) [42]. This model takes a security first approach. The model is based on the recognition that all other properties of an identifier are of little value if the identifier is not securely controlled. The AID model is based on a formal theory of identifiers that we have been developing for several years [10; 12; 35; 41–46; 49]. All identifiers may now be classified and described within the framework of this theory. This model provides a principled approach to identifier system design. This may be used to inform and guide the design and development of any concrete implementation.

1.2 Identifier Control and Trust Basis

As mentioned above the property that is of paramount importance to an identifier system is security. The dictionary definition of *secure* is *free from or not exposed to danger or harm; safe*. For identifiers security typically means secure from exploit or compromise. More specifically an identifier is *secure* with respect to an *entity* if there is a mechanism by which that *entity* may prove it has control over the identifier. The *entity* with such provable control we call the *controller* of the identifier. We may use secure identifiers in an identity system security overlay to secure other systems that do not have secure identifiers [42]. An important use case may be the internet because the Internet Protocol (IP) has no security mechanism for its IP addresses as identifiers. Likewise any existing insecure identifier system maybe secured by using a secure identifier system as a security overlay [42].

The problem we face is how best to prove control over an identifier (digital). There are many such mechanisms that have been used over the years. This begs the question: Is there one mechanism that is sufficiently superior that it may be used as the universal basis for identifier security? AID theory answers that question with an unequivocal yes! That mechanism is rooted in a property called self-certification. A self-certifying identifier (SCID) exhibits this property. Self-certification is based on well established asymmetric cryptographic operations for digital signatures where the identifier is uniquely derived using cryptographic one way functions from a source of information entropy captured by the controller [20; 34; 36; 37; 42]. This property cryptographically binds the identifier to the controlling asymmetric key-pair or key-pair(s). This forms a secure cryptographic root-of-trust for the identifier.

A *root-of-trust* is some component of a system that is secure by design and it security characteristics may be inherently trusted or relied upon by other components of the system. In a more general sense, the security of an identifier system is a function of its trust basis. A root-of-trust may form that trust basis. A self-certifying identifier is derived using cryptographic one-way functions with known cryptographic strength. For example, consider a basic SCID that is prefixed with a public key where that public key is derived from a private key. Suppose that the public key derivation algorithm (one-way function) has 128 bits of cryptographic strength. This means that as long as the private key remains private, an attacker has to perform something on the order of 2^{128} operations to brute force invert the derivation algorithm. This is practically infeasible. Security is then becomes a function of the protection of the secrecy of the private key. High security may be ensured through best practices key management.

The root-of-trust and hence the trust basis for a SCID is inherently decentralizable. Only the entity that captures sufficient random or pseudo-random entropy to generate the private key may derive and prove control over the resulting public key and therefore the resulting SCID. Thus any entity with a cryptographic strength pseudo-random number generator (e.g. a computer) may be the sole controller over the resulting SCID. The controller may choose to manage its keys in a non-decentralized way. But centralized key management is not necessary and is typically less secure. The governance mechanism for the identifier is at the discretion of the controller. Thus any position on a spectrum of centralization/decentralization is possible. Moreover, because the SCID includes the public key or some cryptographically derived material (a large pseudo-random string of characters), the identifier is universally unique [3]. This SCID may then be used as a prefix for a namespace using a URL like approach [54]. All identifiers in that name space then share the same uniquely provable cryptographic root-of-trust. So a SCID comes with three very important properties:

- Self-contained secure cryptographic root-of-trust
- Decentralized control via private key management
- Universally unique identifiers

But a SCID that does not support rotation of the underlying key-pairs is not sustainable as a persistently secure identifier. Eventually through exposure due to use, the key-pairs may become weakened. Advances in cryptographic exploits may also make them weak over time. Thus a truly secure system needs the ability to rotate the key-pairs to new ones. SCIDs with this capability we call self-managing self-certifying identifiers or autonomic identifiers (AIDs). The meaning of the word autonomic is self-managing so an AID is a non-trivial SCID. This is explained in more detail in the next section. But relevant here is that we may use an AID as the root-of-trust to form what we call an autonomic trust basis. This is diagrammed as follows:

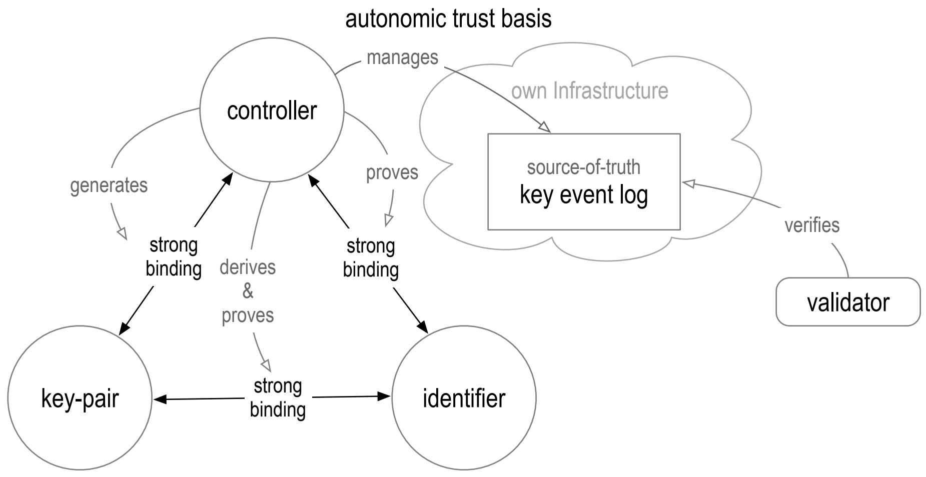

Figure 1.2. Autonomic trust basis

Two other trust bases are in common use for identifier systems. One we call algorithmic. An algorithmic trust basis relies on some network of nodes running some type of Byzantine fault tolerant totally ordering distributed consensus algorithm for its root-of-trust. These networks are more commonly known as a shared ledger or block chain such as BitCoin, Ethereum, or Sovrin [7–9; 18; 47]. The primary exploit of a pool of nodes running such an algorithm is to suborn or gain control of a majority of the nodes. Depending on the algorithm, such an exploit may not necessarily require inverting any cryptographic one-way functions. In a permission-less proof-of-work block chain, for example, it may require capturing 51% of the hashing power of the nodes on the network. An algorithmic (blockchain) root-of-trust, however, may be very secure. Distributed consensus algorithms have well know security properties [51]. The difference is that a network algorithmic root-of-trust is critically dependent on the network infrastructure (the ledger and nodes that write to the ledger). This locks the identifiers to that specific instance of infrastructure. In other words the associated identifiers are not portable. Should the ledger go away, so does the root-of-trust. One may characterize a ledger as a centralized logical construct that is governed in a decentralized manner. Thus the degree of decentralization is less than a self-certifying cryptographic root-of-trust (autonomic). In addition, ledgers tend to be relatively expensive, slow, and with low throughput. This makes using them as a root-of-trust much more constrained than a cryptographic root-of-trust. Thus from a property point of view a cryptographic root-of-trust (autonomic) is generally superior to a network algorithmic (block chain) root-of-trust. An algorithmic trust basis is diagrammed below:

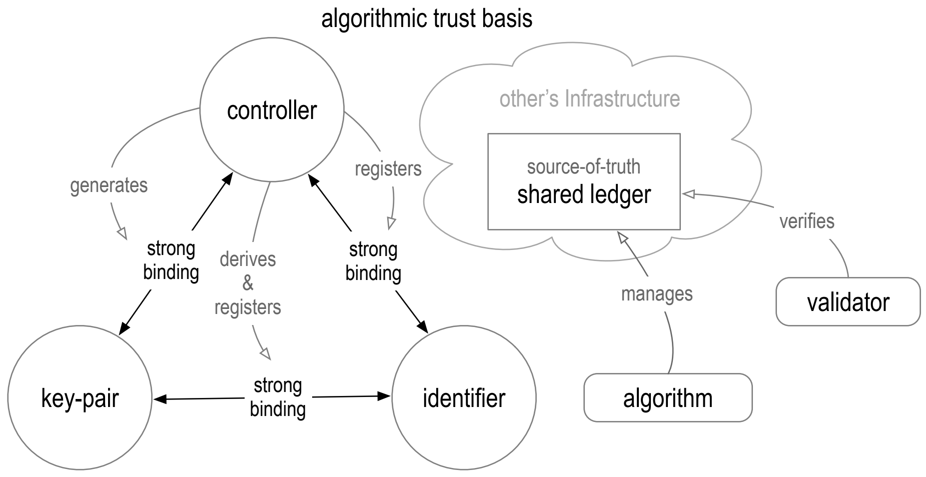

Figure 1.3. Algorithmic trust basis

The other commonly used trust basis in identifier systems is an administrative or organizational trust basis, i.e. a trusted entity. It is difficult if not pathologically so to characterize the roots-of-trust of a trusted entity. Security may be dependent on something as nebulous as employee morale. At its root, the Internet Protocol (IP) was not designed with security as a first class property. Instead Internet security is usually provided via a bolted on identity system security overlay. The security overlay in most widespread use is the open hierarchical Domain Name System plus Certificate Authorities (DNS+CA) [14; 15]. The DNS Certificate Authority system trust basis is administrative derived from the assumed "trusted" entities that are the certificate authorities. It uses the transport layer security (TLS) protocol and X.509 certificates [11; 53; 60]. As is well known, internet infrastructure based on DNS+CA continues to exhibit security vulnerabilities even when organizations follow best practices for conventional security system design and exhibit high levels of operational integrity [2; 4–6; 19; 23; 24; 29; 40; 48]. Given the history of continuing exploits of such a trust basis, it is decidedly inferior from a security perspective to either cryptographic or network algorithm derived trust bases. By comparison, identifiers issued in such a system with an administrative trust basis are neither secure nor decentralized. An administrative trust basis is diagrammed below:

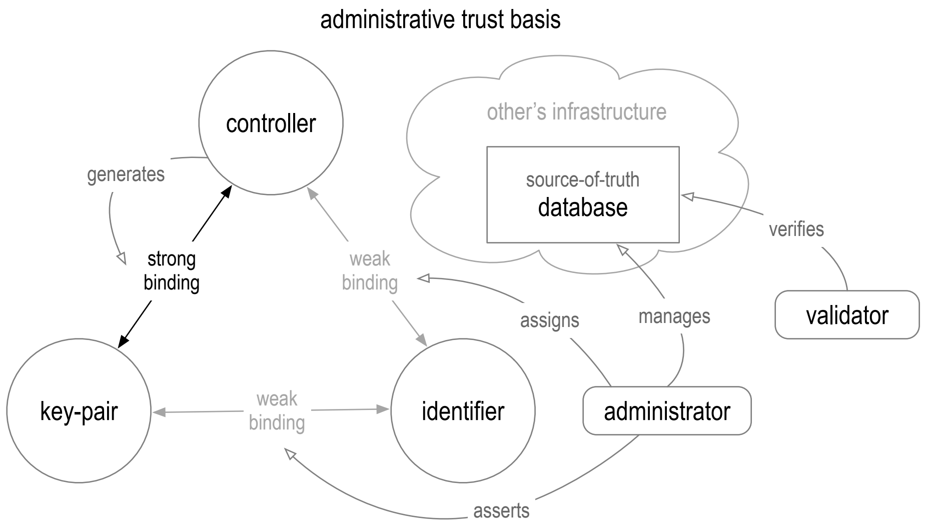

Figure 1.4. Administrative trust basis

1.3 Self-Managing Self-Certifying Identifiers (AIDs)

Although a self-certifying identifier (SCID) employs a cryptographic root-of-trust, it by itself does not provide control over an identifier that may persist in spite of a compromise of the key-pair(s) used to derive the identifier. Persistent control in spite of key compromise is the hard problem of key management. The typical solution to this problem is a mechanism for key rotation. Key rotation involves the revocation and replacement of the current controlling set of key-pair(s) with a new set. We call identifiers in a self-certifying identifier system that includes a secure key rotation mechanism self-managing self-certifying identifiers. There are other key management mechanisms such as delegation that such as system may also exhibit. As a short hand we call such a system an autonomic identifier system (AIS) and the associated identifiers are autonomic identifiers (AIDs). The meaning of the work *autonomic* is self-governing or self-managing and comes form the Greek *auto-nomos* or literally translated *self-rule*. Thus autonomic identifiers (AIDs) are self-certifying identifiers (SCIDs) that may include other self-managing properties such as key rotation. The Key Event Receipt Infrastructure (KERI) is an open standard mechanism (protocol) for an autonomic identifier system (AIS) [12; 35; 42]. KERI provides support for fully decentralized portable secure key management operations on self-certifying identifiers. KERI relies on key event logs (KELs) of key management events such as rotation and delegation. These logs are cryptographically verifiable hash chained and signed data structures. A KEL provides proof of control authority of a given set of keys over an AID. This proof of control authority is end-verifiable by any verifier to the cryptographic root-of-trust of the AID. Its verifiability is not dependent on any intervening infrastructure or trusted entity. The details of KERI of beyond the scope of this paper but may be found here [12; 35; 42].

The function of KERI is to establish control authority over an identifier (AID) in a cryptographically secure and verifiable way. Once that control authority has been established, a verifier may then securely verify any associated authorizations issued by the controller of the AID. These authorizations have the form of a signed authorization statement where the statement typically includes the AID under which the authorization is issued. A verifier may then verify the authorization by verifying the attached signature using the keys that were authoritative at the time the authorization was issued. These authorizations are secure to the extent that the established control authority is secure. The authorizations inherit their security from their associated AID. The trust domain of an AID is the set of statements that may be securely verified as being issued by the controller of that AID (along with any associated transactions or interactions). Authorizations then inhabit and are protect by this trust domain. In other words the authorizations are under the aegis of the AID (or equivalently the controller of the AID) that issued them.

Authorizations may take many forms. One form of particular interest is the W3C Verifiable Credential (VC) standard [55]. Verifiable credentials use the W3C Decentralized Identifier (DID) standard [57]. AIDs may be used as a type of DID.

1.4 Legitimized Human Meaningful Identifiers (LIDs)

Given an AID we may now unify the other desirable property of an identifier, that is human meaningfulness. In contrast the cryptographic material in an AID makes it decidedly non-human meaningful. The AID is represented succinctly by its cryptographic prefix. The prefix is a cryptographically derived pseudo-random sequence of characters. For example a 32 character AID prefix may be as follows:

`EXq5YqaL6L48pf0fu7IUhL0JRaU2_RxFP0AL43wYn148` (1.1)

Such a long pseudo-random string of characters is the antithesis of human meaningful. In contrast, examples of human meaningfulness might be easily recognized words or names or some efficiently hierarchically organized composition of characters. Human meaningfulness has two limiting characteristics. The first is scarcity, this exhibits itself in various undesirable ways such as name squatting, or race conditions to register or otherwise assert control. More importantly, there is no inherent security property of a human meaningful identifier. This makes them insecure by default. Nevertheless, given the trust domain of an AID any human meaningful identifier may be uniquely authorized, sanctioned, or legitimized within that trust domain. Because only the controller of the AID may issue verifiable authorizations associated with that AID, that controller alone may authorize the use of any human meaningful identifier under the aegis of its trust domain. The associated authorization statement legitimizes the use of that human meaningful identifier within the trust domain. This gives rise to a new class of identifier, a legitimized or authorized human meaningful identifier identifier and use the acronym LID. The important property of an LID is that it is verifiable with respect to a given AID. The pair forms a new identifier couplet that we may represent as follows:

$$aid|lid \tag{1.2}$$

where *aid* represents the cryptographic identifier prefix of an AID, the vertical bar, |, represents the legitimizing authorization, and *lid* represents the legitimized human meaningful identifier. This couplet may be diagrammed as follows:

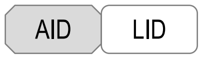

Figure 1.5. AID|LID couplet

This coupling is a special type of name spacing. For example suppose the LID is a library Dewey Decimal code for a book such as:

`625.127C125r` (1.3)

Using the AID prefix example from above, the full *aid|lid* couplet may be expressed as follows:

`EXq5YqaL6L48pf0fu7IUhL0JRaU2_RxFP0AL43wYn148|625.127C125r` (1.4)

The trust domain of an AID provides a context in which to interpret the appearance of any LID. The AID is implied by the context. This means that the AID may not need to be prepended or appear with the LID. This allows the human meaningfulness of the LID to exhibit itself without being encumbered by the AID. Any verifier of the LID, however, knows from the given context how to cryptographically verify the legitimacy of the LID. This model of an *aid|lid* couplet unifies all desirable identifier properties into one identifier system model. The AID part provides the security infrastructure while the LID part provides the application specific human meaningfulness. The connection between the two is provided by a legitimizing authorization represented by the |. With this model, there is no longer any global scarcity associated with a given LID because each AID may have its own copy of the LID. Scarcity only occurs within the trust domain of each AID but is completely managed by the controller of that trust domain (AID).

To further explicate this concept, we may characterize any *aid|lid* couplet as consisting of two classes of identifiers, *primary* and *secondary*. *Primary* identifiers are AIDs and are self-certifying to a cryptographic root-of-trust. *Secondary* identifiers are LIDs and are not self-certifying but are secured under the aegis of an associated primary identifier's trust domain. A *tertiary* class of identifier has no external security properties but may be useful within the confines of an application user interface or other local content. In other words, *tertiary* identifiers are unsecured and may only be used internally or where security is unimportant whereas *primary* and *secondary* identifier may be used *externally*. A *primary* may appear alone but a *secondary* must either appear within the known context of a *primary* or as a couplet with the *primary* itself. This trust domain contextualization is shown in the following diagram:

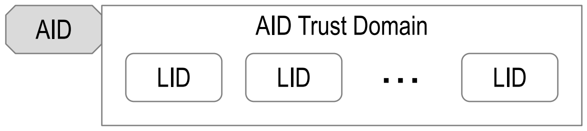

Figure 1.6. LIDs within the context of an AID trust domain

This unified model of *primary*, *secondary*, and *tertiary* identifiers may now be used to guide the design of any identifier system. The design of the primary AID may be tuned to the specific security, performance, and governance properties of the application. The design of the secondary LID may be tuned to provide any other desirable human meaningful properties of the identifier. And all other identifiers are tertiary.

1.5 Reputational Trust

In the context of an autonomic identifier system, trust has two distinct meanings or types. The first is trust is the cryptographic security of the root-of-trust of the AID and associated verifiable statements. The second is trust in the controlling entity of the private keys. The first type of trust is limited to the ability to cryptographically verify non-repudiable signed statements. These provide consistent attribution of any statement to the controller of the private keys. In other words, we may trust in *who* said something and we may prove that *what* they say is consistent (non-duplicitous) or not. But we may not trust in the veracity of *what* was said. For this we need the second type of trust. This second type of trust may be called *reputational* trust. It is trust that an entity will make accurate statements, that is, the entity is reputable. The dictionary definition of *reputable* is *held in good repute; honorable; respectable; estimable, considered to be good or acceptable usage; standard*. The related word *reputation* is a noun meaning, *the estimation in which a person or thing is held, especially by the community or the public generally*. Reputation is a measure of accumulated consistent behavior. It can be evaluated at different levels of granularity. The root of reputation comes from the Latin word *reputāre*, which is equivalent to *re+putāre*, that is, to re-think or re-consider.

Identifier systems need both attributional trust and reputational trust. The first derived from consistent attribution, enables us to trust that a statement was made by an entity (*who*) and the second, reputation, enables us to trust in *what* that entity said. This forms a trust balance. For example, a verifiable credential issued by an entity is only of value if we can trust both that the entity indeed issued it and that the contents or attributes of the credential are accurate. Cryptography allows us to verify that the issuer truly created the credential but we must trust the institutional, organizational, behavioral integrity or reputation of the issuer in order to trust in what the credential asserts. The later requires reputational trust in the issuer of the credential. Another way of understanding this trust balance is that the primary value in any issuance of information is dependent on the veracity of that information and that veracity is dependent on the credibility (reputation) of the issuer but we may only securely associate that reputation to that information if the authenticity of the issuer is established. So first authenticity (AID) and then veracity (LID). This dependency is shown in the following diagram.

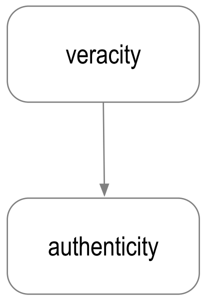

Figure 1.7. Veracity dependency on authenticity.

It is this reputational form of trust that makes a governing organization that authorizes LIDs within the context of their AID so valuable to the associated ecosystem. The integrity, consistency and repeatability of the organizational procedures used to issue authorizations for LIDs are essential to reputational trust.

The *aid|lid* coupling mirrors this trust balance. We rely on the cryptography for the trustworthy establishment of the control authority of the controller of the AID. And we rely on the reputational integrity of the controller to trust that a given LID has been authorized correctly.

A common usage of reputation is as a considered evaluation of how someone has behaved in the past that provides an estimation of how they might behave in the future. In other words, reputation is a measure of past behavior that may be used to predict future behavior. The context of a reputation may also be important, that is, past behavior may be a better predictor of future behavior when applied to similar contexts. Reputation is based on proof of consistent behavior which requires provably consistent attribution which brings us full circle. Provable trust in the AID mechanism enables provable trust in the entity behind the LID mechanism.

1.6 Root AID Establishment via *Well Known* Reputational Association.

The *aid|lid* couplet is an example of a very short *chain-of-trus*t. The anchor or root link to that chain is the AID. But in order for such an AID to usable it must be universally recognized as the authoritative identifier for a given entity. Because an AID is self-certifying the identifier itself includes a universally unique pseudo-random string of characters. So there may be only one. But the problem is knowing which identifier is the one and only one authoritative identifier. As described previously such strings are not human meaningful. Therefore they are not easily recognized as the authoritative identifier. The process that establishes an AID as the root identifier with an organization may be called a *well known* reputational association. Once this connection between root AID and organization becomes common knowledge then it is secure.

To elaborate, an AID prefix is cryptographic material. Best practice is to derive it via cryptographic one-way functions that maintain 128 bits of information theoretic entropy. Given this recognition, such a self-certifying identifier may practically be used as the root-of-trust for all other activity associated with the entity that controls the identifier. Control is established by virtue of possession of the private keys from which the identifier was derived. This enables a secure bootstrap mechanism whereby the controller may securely issue other identifiers or authorizations.

Because the identifier is not human meaningful, it may not be easily recognized as the authoritative root identifier (AID). Consequently, the identifier must first become *well known* as authoritative for that entity. This association becomes secure once it has become sufficiently well known that an attacker may not fool any verifier into recognizing a different identifier as the authoritative root identifier. In other words, the association between the AID and the entity must become *common knowledge*.

The process of elevating an identifier to *well known* (i.e. *common knowledge*) status may leverage several other already *well known* URLS or registration mechanisms that verifiably associate the AID to the entity [38]. These *well known* mechanisms may employ some vetting process to accept a registration. Or they may consist of already well known resources under the control of the entity wherein the entity provides a statement that links the entity to the new AID. The strength of the *well known* establishment is multiplied by the number of such associations, i.e. provides a reputational source of trust.

Any verifier may thereby check multiple redundant well known resources or registries to confirm that indeed the entity has declared the identifier as its authoritative AID. Once the authoritative status of the root AID is established via this *well known* process then any other identifier's authoritative status may be established merely via signed statements using the authoritative controlling key-pairs of the root AID. Because the root AID is self-certifying and self-managing, the public keys from the set of authoritative keys-pairs may be extracted from the identifier itself or if rotatable from its unique, verifiable key event log (KEL).

When the AID is expressed as a W3C DID (decentralized identifier), the associated DID:Doc may use the following draft specification to provide references to verifiable credentials (VCs) that link the identifier to well-known URLs (domains) [38; 58]. The process of establishing the authoritative nature of an identifier using well known locations (URLs) is standardized in IETF RFC-8615 [38]. The RFC-8615 approach reserves a specific path component `.well-known/`

of a URL in each DNS domain that provides well known meta-data about some other resource. Consequently, meta-data about an AID may provide by the entity that controls the DNS domain an authoritative association between the AID and the controlling entity or some other entity.

This well known path prefix, `/.well-known`, is reserved in HTTP, HTTPS, WebSocket (WS), and Secure WebSocket (WSS) URIs for these well-known locations. For example, the URL prefix `myentity.com/root-identifer/.well-known/` could return a document that specifies the actual AID as being the root self-certifying identifier for the entity that controls the domain name `myentity.com`.

The concept of well known or common knowledge associations is similar to the concept of a rich presentation of verifiable credentials [13]. A rich presentation makes the likelihood of the intersection of all the credentials by any but the true controller vanishingly small. Likewise, associations provided by multiple well known locations may establish with high likelihood the authoritative nature of a given AID. Consistent common knowledge is the root-of-reputational-trust by which the credibility of an association may be established. One need merely establish with sufficient redundancy a given association to be trustworthy.

In summary, because AIDs are self-certifying, they have a secure primary cryptographic root-of-trust that is end-verifiable. The well known establishment process may secure the first link in this chain. Given a sufficient degree of common knowledge that a given AID is the first link in the verifiable chain-of-trust for an entity, then any subsequent links in the chain may be established by cryptographically verifiable attestation via the preceding link. Given this chain-of-trust, other items of data such as verifiable credentials or LIDs may be securely associated with the chain. Any subsequent attestations may extend the chain-of-trust in a secure manner. Any subsequent links are not vulnerable because they may be cryptographically verifiably authorized by a chain-of-trust back to the the root-of-trust which is the AID itself.

1.7 Examples of LIDs

Various standards bodies or industry organizations act as the reputable body that authorizes or legitimizes the issuance of LIDs for a given industry. These LIDs are human meaningful in the sense that they follow a structured syntax that is not cryptographic. Although they may not be human friendly they are still human meaningful from their structure. Usually the issuance of an LID uses an administrative trust basis.

One example of an industry LID would be the GLEIF legal entity identifiers (LEI) [31–33]. Another example are the GS1 GTIN or SGTIN electronic product identifiers (ePI) [16; 17; 25; 26; 28; 30; 56]. By making the issuance a verifiable authorization via a VC from an AID with a cryptographic root of trust, these identifiers systems may inherit the security of the AID in the form of a couplet.

For example GLEIF's Legal Entity Identifier (LEI) may be viewed as a type of LID (legitimized human meaningful identifier) [31–33]. An LEI code specifically is an hierarchically structured 20 character string. An example LEI may be expressed as follows:

`254900OPPU84GM83MG36` (1.5)

GLEIF may create a trust domain based on GLEIF's sole control over a root AID . Lets call this root AID the GLEIF AID or GID for short. The GID is uniquely represented by its cryptographic prefix. Suppose this is given by:

`EXq5YqaL6L48pf0fu7IUhL0JRaU2_RxFP0AL43wYn148` (1.6)

The the resultant *gid*|*lei* couplet may be represented as follows:

`EXq5YqaL6L48pf0fu7IUhL0JRaU2_RxFP0AL43wYn148|254900OPPU84GM83MG36` (1.7)

Within the GLEIF GID trust domain, GLEIF may issue authorizations that legitimize any number of LIDs in the form of LEIs. The authorizations makes the LEIs verifiable to the root-of-trust of the GID. Thus these may be called verifiable LEIs. The entities that participate in the GLEIF ecosystem may have their own trust domains protected by their own AIDs. They may receive authority to issue LEIs against their verifiable AID. But this enables the ecosystem security to be based on secure cryptographic roots-of-trust not merely administrative roots-of-trust.

As a further example, a GS1 SGTIN is a hierarchically structured 23 character string [16; 17; 25; 26; 28; 30; 56]. An example SGTIN may be expressed as follows:

`48150614141000734314 59` (1.8)

As described previously, a trust domain is based on some entity's sole control over a root AID. Lets call this root AID the GS1 AID or GID for short. The GID is uniquely represented by its cryptographic prefix. Suppose this is given by:

`EXq5YqaL6L48pf0fu7IUhL0JRaU2_RxFP0AL43wYn148` (1.9)

The resultant *gid|sgtin* couplet may be represented as follows:

`EXq5YqaL6L48pf0fu7IUhL0JRaU2_RxFP0AL43wYn148|48150614141000734314 59` (1.10)

Within the GS1 trust domain of its AID, GS1 may issue authorizations in the form of VCs that legitimize any number of LIDs expressed as SGTINs. The authorization makes the associated SGTIN verifiable to the root-of-trust of the GS1 AID. Thus these may be called verifiable SGTINs. The entities that participate in the GS1 ecosystem may have their own trust domains protected by their own AIDs. They may receive authority to issue SGTINS against their verifiable AID. But this enables the ecosystem security to be based on secure cryptographic roots-of-trust not merely organizational or administrative roots-of-trust.

Another LID example is the GS1 GLN for global location number. A GLN is a hierarchically structured 13 character string. GLNs may be used to identify physical locations, operational locations and legal entities [1; 21; 22; 26; 27]. An example GLN may be expressed as follows:

`4567123489098` (1.11)

Given the same GID and associated trust domain for GS1 as above, the resultant *gid|gln* couplet may be represented as follows:

`EXq5YqaL6L48pf0fu7IUhL0JRaU2_RxFP0AL43wYn148|4567123489098` (1.12)

An ecosystem participant might use a combination of LIDs of different types from multiple trust domains. But any other participant will be able to end-verify the authorization status of that combination to the associated cryptographic roots-of-trust of each of the LIDs.

1.8 Identifier System Security Overlay Properties

Any identifier system security overly may have some degree of any combination of the following three properties but not all three completely, Or in other words one can can get variable degrees of all three but not completely all three at the same time. This is a trilemma for identifier systems.

The three properties are authenticity, privacy, and confidentiality.

KERI is design to provide the highest level of authenticity. Because KERI provides complete authenticity, by the trilemma KERM may not provide as well complete privacy and complete confidentiality. Likewise a that system that provides both complete privacy and complete confidential may not provide complete authenticity.

The reason a system may not provide all three completely is that with cryptography one has to layer the different properties. No single cryptographic operation provides all three. But layering exposes the weaknesses due to the separation between the layers. Because no layer can have all three, one must pick one or two properties for the bottom layer and then bolt on the remaining property or properties in the remaining layer or layers.

KERI employs non-repudiable signatures which provides the highest level of authenticity in a cryptographic operation. But verifiable non-repudiable signatures are correlation points for the associated identifier. This makes privacy incomplete. Alternatively a Diffie-Hellman key exchange provides for confidential encryption but does not provide non-repudiation so does not provide complete authenticity. Without going into detail here, careful examination of any cryptographic operation exposes a similar tradeoff. This triangle is diagrammed below:

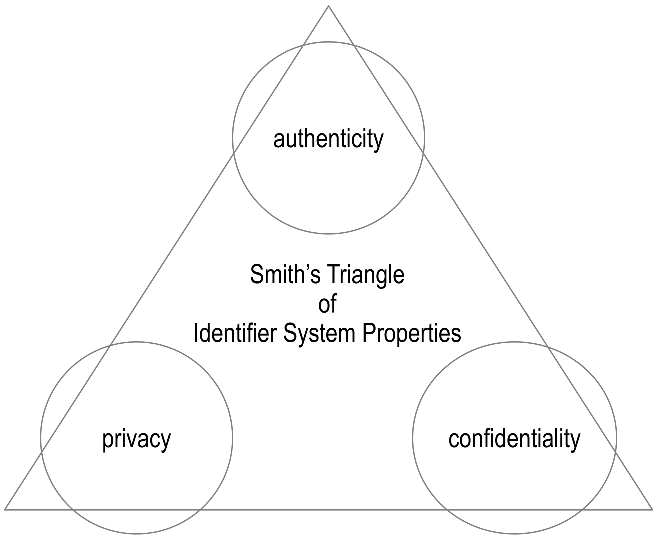

Figure 1.8. Triangle/Trilemma of Identifier System Properties

2 Separable Roots-of-Trust

KERI supports separable control over shared data. This separation of control is between the controller and the validator. The controller controls the generation and promulgation of key events for the controller's AID and the validator controls the confirmation network that verifies the key events.

2.1 Root-of-Trust for Controllers

The KERI white paper goes to some length to describe the trust basis for the controller of an identifier including specifically the cryptographic root-of-trust that AIDs derive from SCIDs (self-certifying identifiers) [42; 59]. This root-of-trust is protected by the controller's key management infrastructure. In KERI parlance the authoritative cryptographic material is called the identifier prefix. This is the a cryptographically strong string of characters that is derived via one-way functions from a pseudo-random string of characters that represents the information entropy captured by the controller. In addition to the key and event generation and signing infrastructure, when the associated AID is public, the controller also declares an event promulgation network of witnesses. A network of witnesses provide high availability of the associated key event log (KEL) for the AID. The controller may protect its key generation, key storage, and event signing infrastructure by running it inside a trusted execution environment (TEE) such as SGX, TrustZone, an HSM, a TPM, or other similarly protected hardware/software/firmware security system. This protects the cryptographic strength of the root-of-trust. The controller can then make the claim, that, with KERI, as long as its secrets remain secret then the rest of the controller's key management and event signing infrastructure may be as secure as the cryptographic strength of the one-way functions used to derive the identifier prefix. How the controller protects its secrets is application dependent.

Similarly the controller may protect its witnesses by running them inside some TEE like protection system. However, the witness construct allows the controller to use a *threshold structure* to protect the witnesses even when not using some form of a TEE. Threshold structures are defined below [51]. The following diagram shows the promulgation network for a controller of the identifier prefix of a KERI AID.

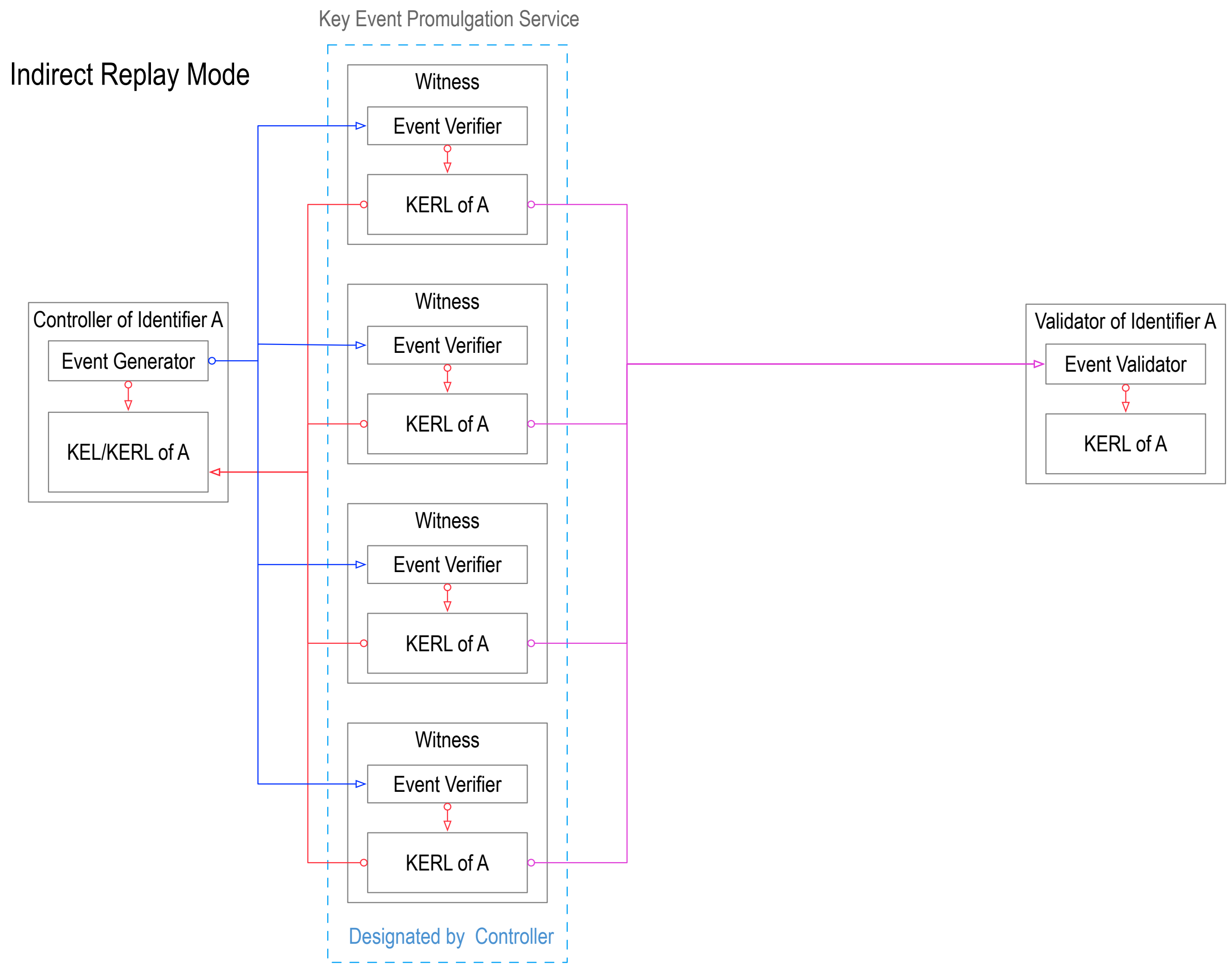

Figure 2.1. Controller's promulgation network

2.2 Root-of-Trust for Validators

The KERI white paper, however, does not go into length on the trust basis for the validator. The validator's essential task is to perform cryptographic signature verification on the events it receives either directly from the controller or indirectly from the controller's promulgation network. Should the software/firmware that runs the validator's signature verification become compromised then the validator may not know that the key events sourced by the controller are themselves compromised. The validator needs to protect its verification software.

One approach to protection is to run signature verification inside a TEE or special purpose HSM. This is the dual operation to the controller running signature generation inside a TEE or HSM. Signature verification is a simple process that may leverage well known software libraries. Because the verifier may use any host or device to run the verification software it may be difficult for an attacker to know which hosts or devices to attack. Nonetheless protecting the verifier(s) is important. The root-of-trust for the validator is the security of its verifying devices. It is not cryptographic per se, but depends on the protection of the cryptographic operations that perform the verification. From an end-to-end system's perspective the signature verification software/firmware needs to be as secure with respect of the validator as is the root-of-trust the signature generation software/firmware is with respect to the controller.

In addition to or instead of TEEs, a validator may also use the threshold structure of KERI to enable secure verification. Similar to a controller, a validator may employ a confirmation network of watcher (verifier) nodes. Indeed, KERI was designed specifically so that a pool of watchers may be employed for the validator function. When a pool of watchers is employed then that pool may be designed to be as secure as needed using a threshold structure (with or without using TEEs). Moreover, because KERI splits the system into two sides, each side has two distinct roots-of-trust, each of a different kind. Because the validator side is the easiest to manage, the KERI WP does not spend much time on it. The security considerations of both hardware and software on both sides should be addressed. The following diagram shows both a promulgation network of witnesses and a confirmation network of watchers.

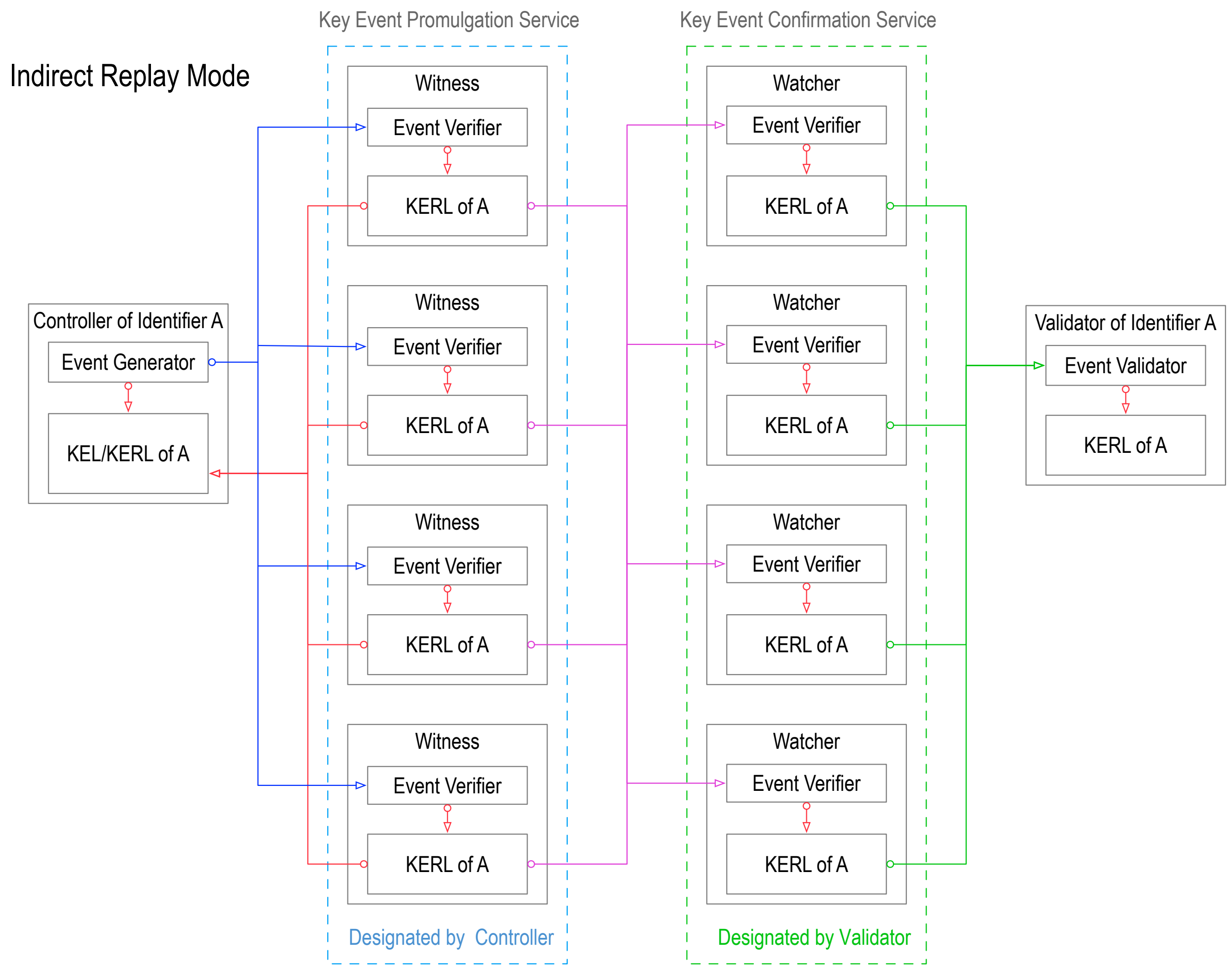

Figure 2.2. Controller’s promulgation network with validator’s confirmation network

To elaborate, the promulgation side (the controller) uses a root-of-trust that is a SCID and the security is rooted in the cryptographic strength of the controller's private keys and the degree of protection of its key management infrastructure. The confirmation side (the validator) has a different root-of-trust which is based on the degree of protection of its event signature verification software/firmware. The validator's root-of-trust is independent of the controller's. It is not a function of the key management of the controller.

2.3 Threshold Structure Security vs. TEE Security

This early influential paper by Nick Szabo describes the concept of a "threshold structure", which underlies the ideas in the security guarantees of multi-signatures, distributed consensus, and multi-factor authorization (MFA) to name a few [51]. KERI itself employs threshold structures namely, multi-sig and distributed consensus, and assumes that the key management is using best practices such as MFA. Threshold structures may be employed in a complementary manner to trusted execution environments (TEE) for security. Szabo's paper does a good job of explaining why the two types of security are complementary and that the whole may be greater than the sum of its parts.

What a threshold structure for security does for us is that we can have weaker key management or execution environment infrastructure individually but use a threshold structure to achieve greater overall security by multiplying the number of attack surfaces that an attacker must overcome to compromise our system. In other words, with threshold structures, overall security may be greater than the security of any of the individual parts. For example, in MFA the combination of two factors, something you have and something you know, may be much more secure than either of the factors by themselves.

As is well known, a distributed consensus algorithm is a provable algorithm using a set of nodes where the algorithm provides a security guarantee that is greater than the security guarantee of any of its constituent nodes. It makes a guarantee that in spite of a compromise of F of the $3F+1$ nodes, that the ledger written by the pool as a whole is still secure [7–9]. So it's improving security by multiplying the number of attack surfaces that must be simultaneously overcome. This may be more or less complex depending on how one defines complexity. To restate, the value of Byzantine Fault Tolerant algorithms is that one can arbitrarily increase the degree of difficulty to an attacker by multiplying the attack surfaces using a distributed network of nodes with sufficient non-common mode failure vulnerabilities [7–9]..

This applies to KERI as well. The witnesses and watchers independently multiply the attack surfaces of the promulgation and the confirmation networks such that each witness or watcher respectively may be relatively insecure but the system as a whole may be highly secure. Numerous papers discuss how secure a distributed consensus pool may be [9]. But when comparing *apples* (key management and trusted execution environment approach to security) to *oranges* (distributed consensus approach to security) its hard to say that the security of a distributed consensus algorithm is necessarily less secure than the key management infra-structure root-of-trust of any of its nodes. Although as a general rule, in an *apples* to *apples* comparison, more complex is less secure.

With KERI's design we can make the following claim:

> *Through threshold structures, KERI enables the promulgator (controller) to to make its system no less secure than the security of its secrets and enables the confirmer (validator) to make its security at least as secure as the security of the promulgator.*

Obviously in resource constrained applications there are limits to the application of distributed consensus for security. Nonetheless, the point of KERI is to capture the goodness of distributed consensus without the cost, latency, and throughput limitations of conventional blockchain distributed consensus algorithms used for cryptocurrencies. The most important feature of a cryptocurrency is that it must be double spend proof. Because KERI's key event operations are idempotent they do not need to be double spend proofed, so we can greatly simplify the distributed consensus algorithm in KERI. Which makes KERI relatively more attractive for many applications including IoT applications by comparison.

2.4 Separated Control

As a result of the relaxation of double spend proofing, KERI is able to break the distributed consensus algorithm into two halves and simplify it in the process. As mentioned above the two halves are the promulgation half and the confirmation half.

The promulgation half is under the control of the controller of the private keys. The controller is solely responsible for the security guarantees of its signed key events as well as the promulgation of those events. The controller is not dependent on any other entity for its security. The controller is free to make its promulgation system (witness set) arbitrarily as secure as it desires. Although there is no point in making it more secure than its root-of-trust in its key management. The promulgation infrastructure verifies signature on events already signed by the controller and then generates signatures on those events as receipts. The receipted events are then broadcast. No attacker can forge any signed event or receipt without compromising the signing infrastructure of the associated component. When using multi-sig this means compromising K of the N signatories. Moreover, no attacker can compromise the receipting infrastructure without compromising more than F of the witnesses. The worst an attacker can do without compromising the receipting infrastructure of F witnesses, is to DDOS or delete the broadcast. This is a well delimited and well understood type of attack with well understood mitigation.

The confirmation half is under the control of the validator. The validator is solely responsible for the security guarantees of its confirmation network. This network is verifying signatures on events promulgated by the controller and/or its witnesses. The validator is free to design and implement its confirmation network of watchers to be as secure as it needs and is not dependent on any external entity for that. Because verification happens in the edge, an attacker must attack and overcome a multitude of edge verifiers (watchers) with little net gain per individual exploit. This multiplies the attack surfaces that the attacker must overcome. Mitigation mechanisms for attacks on verifier firmware are also well known. Because verification is largely a fixed operation in software, the verification software may be deployed by the validator in a manner that may not be predictable by any attacker. Once deployed the attacker must individually find, target, and exploit each edge verifier (watcher). By multiplying the attack surfaces with multiple watchers, each validator may arbitrarily increase the difficulty of attack without having to use TEEs for its watchers.

KERI's design enables the validator to make a trade-off. A given watcher under the aegis of the validator may be secured by using a verifiable attestable hardware security module (HSM) (a type of TEE) to perform the end verification operations on the KERI events. This approach may provide assurance to the validator that the cryptographic signatures are verified correctly and that the verification software/firmware on the watcher has not been tampered with. Or the validator may choose to use a threshold structure to secure the signature verifications of its watchers through a distributed consensus pool of confirming nodes that each perform the signature verification and then compare results. Or the validator may choose to use a combination of both where each watcher uses an HSM/TEE to protect its signature verification so that fewer watchers are needed because each watcher is correspondingly more secure.

Likewise, the promulgation (controller) can make similar trade-offs for its witnesses. More witnesses without HSM/TEE for signature verification and receipt generation or fewer with HSM/TEE. But most importantly the separation of control via the splitting up of promulgation and confirmation enables each to design and implement according to their desired guarantees of security.

In summary, by splitting the attack surfaces into two independent sides. The controller (promulgation side) and the validator (confirmation side) each may independently set their security levels by multiplying their own attack surfaces separably.

Nonetheless in the special case when the signature verification is performed on a single device (a single watcher) then that watcher/verifier may very well benefit from using a TEE in order to have the same degree of security as the controller when the controller is using a TEE. But if the validator is using a pool of watchers then the watchers as a pool may be as secure as a controller's TEE without any of the watchers using a TEE.

KERI by design puts verification at the edge. The weakest case is an application where there is only one verifier/watcher. One example of this is a mobile device under the physical control of the validator. But even in this case verification at the edge means the attacker is fighting an asymmetric war where the asymmetry is in the favor of the validator. In spite of a lot of effort, an attacker may only compromise one device at a time. This assumes non-common mode exploits such as exploiting the crypto library on GitHub that sources the verifier's software.

In other use cases where the device is not under the physical control of the validator such as a remote payment card reader, or some other like application where the sole verifier is a known high value target, then the verifying device may benefit greatly from using a TEE.

As a use case, the end verifiability of KERI opens up an opportunity for HSMs for signature verification. Typically mobile phone devices could be used as a type of general purpose HSM for KERI event signature verification but a dedicated device that only does verification may provide a better security for cost trade-off. Currently there are many low cost hardware devices (HSMs) such as a Ledger Nano, UbiKey, Trezor etc. that provide signature generation. But not so for signature verification. With KERI there may be applications where verifiable attestable HSMs may be used for event signature verification. The trusted computing group recently released a standard for low cost HSM chips that would be applicable to HSM for signature verification for end verifiers [52]. This may be used in combined with the IETF standard for remote attestation [39]. Together a validator could implement an HSM for end verification where it could query the HSM and be assured via a signed attestation from the HSM that its signature verification firmware has not been tampered with.

2.5 Assumptions about Verifiers

KERI makes some assumptions about a verifier.

- The software/firmware that is doing the signature verification is under the control of the validator.
- The verification hardware/software is not dependent on any external infrastructure to do the verification at the time of verification other than to receive a copy of the events.
- The software/hardware that is verifying signatures has not been compromised. Singly when only one watcher or multiply above a threshold when using multiple watchers.

If the verification software/firmware has been compromised then no cryptographic operations run by the verifier may be trusted by the validator. But given the three assumptions above, the worst that an external attacker can do when attacking a verifier is a DDOS or delete attack to prevent the verifier from seeing the events. An attacker may not forge events to deceive a verifier.

Duplicity detection, which protects, not against an external attacker, but against a malicious controller does require access to watchers that are also recording duplicitous events. So this may be viewed as a dependence on external infrastructure, but its dependence is generic. Furthermore, the goal is to make duplicity detection ambient such that a malicious controller may be easily detected.

2.6 Zero Trust Computing

KERI's separable control design allows the controller and each validator to independently design their infrastructure. Because KERI is not dependent on a shared public ledger, each may implement their KERI components using off-the-shelf cloud computing infrastructure. Best practices for implementation should follow zero trust computing principles. These principles are described at more length elsewhere but may be summarized as follows [10]:

1. *Network Hostility.*
   The network is always hostile, internally & externally; Locality is not trustworthy. Solutions must provide means to mitigate network layer security vulnerabilities (man-in-the-middle, DNS hijacking, BGP attacks).
2. *E2E Security.*
   Inter-host communication must be end-to-end signed/encrypted and data must be stored signed/encrypted. Data is signed/encrypted in motion and at rest.
3. *E2E Provenance.*
   Data flow transformations must be end-to-end provenanced using verifiable data items (verifiable data chains or VCs). Every change shall be provenanced.
4. Verify every-time for every-thing.
   Every network interaction or data flow must be authenticated and authorized using best practice cryptography.
5. *Authorization is behavioral.*
   Policies for authentication and authorization must be dynamically modified based on behavior (reputation).
6. *No single point of trust.*
   Policies for authentication and authorization must be governed by end-verified diffuse-trust distributed consensus. Policy is protected by diffuse trust.
7. *Hosts locked down.*
   Hosts or host components executing any of the logic mentioned above must be locked down. Any changes to the host execution logic or behavior must be fully security tested and validated over the respective possible combinations of hardware and software platform. This means locking down key management and cryptographic operations on the devices. This includes key generation and storage, as well as signature generation and signature verification. These may benefit from the use of some form of trusted execution environment (TEE) either generally or specially as in a trusted platform module (TPM) or a hardware security module (HSM). In addition to key management and cryptographic operations, special security measures must be implemented regarding secure execution of the application logic (e.g. code injection, insecure object references, cross-site/service request forgery, cross-service scripting, etc.).

3 Security Summary

One useful way of describing KERI is as a decentralized key management infrastructure (DKMI) based on key change events that supports attestable key events and consensus-based verification of key events.

With KERI, security becomes largely a function of a given participant's infrastructure and not another entity's “trusted” internet infrastructure. Each controller of an identifier gets to pick their own infrastructure, and each validator gets to pick their own infrastructure. This is a big advantage as it does away with the whole house of cards that are the “trusted entities” in DNS/CA.

It's much easier to secure one's own keys well than to secure everyone else's internet computing infrastructure well.

4 Key Rotation and Statement Revocation

The term *revocation* has two completely different meanings in the identity space. In key management one may speak of revoking keys. With statement issuance, authorization issuance, or credential issuance, one may speak of revoking an authorization statement, a token, or a credential.

This becomes confusing when the act of revoking keys also implicitly revokes the authorization of statements signed with those keys. Any statement may be effectively authorized by virtue of the attached signature(s) made with a set of authoritative keys. The statement itself may be authorizing some other function in the system. So, the verification of the signature on an authorizing statement is essential to determining the authoritativeness of the associated authorized function. To clarify when an authorization is conveyed via a signed statement, the signature acts to authorize the statement. The meaning (semantics) of the statement may merely be data that thereby becomes authoritative. Or the meaning (semantics) of the statement may be to convey an authorization for some other function. In either case revoking the authorization of the statement revokes the authoritativeness of the semantics of the statement.

KERI terminology usually avoids that confusion between rotation and revocation because a key rotation operation is the equivalent of a key revocation operation followed by a key replacement operation. So one operation, *rotate*, is implemented instead of two operations *(revoke and replace*). A bare key revocation is indicated by replacement with a null key. So only one operation is needed, that is, rotate where a special case of rotation is to rotate to a null key.

Given the KERI definition of rotate as applied to keys, revocation is usually unambiguously applied to mean revocation of the authorization of a signed statement. Otherwise when not clear from the context, one may simply use the modifier *key* revocation vs. *statement* revocation.

To clarify, the authority of a signed statement is imbued by the attached signature(s) created with a set of authoritative keys. This begs the question: is a signed statement authoritative/authorized after the keys used to sign it have been rotated? If not, then the statement is effectively revoked as no longer being an authoritative/authorized statement. If the statement is still authoritative/authorized after the keys used to sign it have been rotated, then the statement is not effectively revoked by the rotation itself but requires a separate signed revocation statement that rescinds/revokes its authoritative/authorized status. This revocation statement may be signed by a different set of authoritative keys than the keys used to sign the statement being revoked.

4.1 Authorization Models

Given that key rotation and statement authorization are two distinct operations, a given identifier system needs to define the model of how to verify the authoritative state of a given authorized statement when the keys used to sign that statement have been rotated. There are several models for determining the status of an authorized statement after a rotation of the keys used to sign that statement. Each authorization model has an associated verification rule.

- *Rule 1:* Persistent Authorization Model

Any signed statement that is signed with the set of authoritative keys at the time of its signing, is valid until revoked (rescinded) with another signed statement that is signed with the set of authoritative keys at the time of the signing of the revocation statement.

This means that merely rotating keys does not revoke or rescind the validity of prior signed statements. Otherwise every time one rotates keys one would have to reaffirm (reissue) every prior signed statement that used the now obsolete keys. This is the model most compatible with the *issuer-holder-verifier model* of VCs. A VC is a type of signed statement. In this case, the verifier does not need to communicate with the issuer, but need only communicate with the holder and a verifiable data registry (VDR) to determine the revocation state of the VC. This tripartite model enables more privacy for the holder because the issuer is not able to track each presentation of a VC by a holder to a verifier. KERI may be used to support this model because the key event log (KEL) may be used as a VDR to anchor both issuance and revocation statements.

- *Rule 2:* Ephemeral Authorization Model

All statements signed with the set of authoritative keys at the time of their signing are automatically revoked when that set of authoritative keys are rotated.

This model of mandatory revocation of all signed statements is most commonly used in token-based security systems where all tokens issued under a given set of keys are automatically revoked when those keys are rotated. A given signed statement is authoritative only if it verifies against the current set of keys. This simplifies the system because no VDR is required. However, this means that whenever the keys are rotated, any continuing authorizations require reissuance of new signed statements using the new keys. This may be problematic when using the tripartite VC model of issuer-holder-verifier. The issuer must push out new VCs to all the holders each time it rotates keys.

Typically, the ephemeral authorization model works best in a bipartite VC model where the verifier requests verification from the issuer directly. In this case, the holder does not actually hold a VC but merely a reference to a VC held on its behalf by the issuer. Alternatively, the identifiers in the VC are dynamic references that resolve to service endpoints. In either case the holder presents the references to the verifier who contacts the issuer directly to determine authorization status. Therefore, when rotating keys, the issuer does not have to re-issue a VC but merely tracks the authorized state of each VC internally. This bipartite approach is less privacy preserving than the tripartite approach because the issuer may now track all presentation requests by the holder. KERI also supports this model in the sense that an issuer may use the KEL for only tracking key rotations and not signed statements.

- *Rule 3*: Some Authorizations are Persistent, and Some are Ephemeral (Hybrid) Model

Any presentation of a signed statement with persistent authorization must include a reference to its location in the VDR (log) to determine the authoritative keys at the time (location) in the VDR when the statement was signed. The signed statement is authorized until revoked (rescinded) with another signed statement that is signed with the set of authoritative keys at the time of the signing of the revocation statement. If the VDR location reference is absent in the presentation, then the authorization is assumed to be ephemeral and the verifier uses the current authoritative keys to verify the authorization status of the signed statement.

This is a hybrid of Rule 1 and Rule 2. In this model, only logged signed statements use rule 1) and all other signed statements use rule 2). Because Rule 3) requires a log for the persistently authorized signed statements, it may still be privacy preserving relative to a Rule 2 implementation without a log. In this case the verifier need only reference the log (VDR) to get the authoritative keys instead of making a presentation directly to the verifier. KERI supports Rule 3 because it provides both the key state and may also provide any anchored signed statement state.

To elaborate, for Rule 1 or 3 to be practically implemented, one must maintain a verifiable data registry (VDR), such as, either a log of signed statements or a cryptographic commitment (digest) to those statements (via a Merkle tree or hash chained data structure). With a VDR one can verify that a persistently authorized statement was signed with the authoritative set of keys at the time of signing. Verification is always with respect to the set of signing keys used to create the signature at the time the statement was signed. This means the verifier must have a way of determining the history or lineage of control authority via a log or ledger of the key rotation history to know that a persistently authorized statement was signed with the authoritative set of keys at the time of signing. This means that the log or ledger must not only log the lineage of keys (key rotation history), but also the statements (or at least a digest of the statements) signed by those keys. Otherwise, a compromise of the current signing keys (which rotation protects from) would allow an exploit to create forged but verifiable supposedly authorized statements after the keys have been rotated.

The log itself allows the controller to anchor cryptographic digests of any other data to the log at a specific place in the log. One may use the KERI KEL as a hash chain clock. The log provides a point in hash chain time for any anchored data item. That anchor is a cryptographic commitment to that data. The ordering of all anchored statement may then be verifiable like a ledger but without needing a "ledger", as the KEL is effectively a "micro ledger". This means that, with KERI, one may use the KEL as a clock for determining both the issuance and later revocation time for any anchored statements associated with VCs.

If the use of a VDR (log or ledger) is not practical, then only Rule 2 is feasible. This is one reason why token-based authorization systems such as OIDC and JWT only use Rule 2. This means that when the keys used to sign the token have been rotated the token's authorization is automatically revoked; effectively, the token is always verified by the current set of signing keys so it will fail verification after key rotation.

When leveraging authorization tooling to implement VCs, one should carefully consider how much is gained or lost by using tools built for different authorization models.

- *Rule 4:* Rotate the Identifier not the Keys Model

  A signed statement's authorization is persistent until the ephemeral identifier associated with the authoritative keys has been rotated. Rotation of the identifier effectively rotates the authoritative keys.

This model addresses the case where the keys controlling an identifier are not rotatable. This is more common than one might expect (for example, an Ethereum or Bitcoin address). In this case, any signed statement may ***not*** be revoked by merely rotating keys because the keys cannot be rotated. Instead, one might be tempted to use a revocation registry to determine if a given signed statement has been revoked by a later revocation statement. This approach is problematic, from a security perspective, because signing keys may eventually become compromised due to exposure at which point an exploiter may then corrupt the revocation registry, rendering the registry unrecoverable.

In KERI, an AID or SCID prefix may be declared as non-transferable at inception (which means one may not rotate the AID's or SCID's controlling keys). This means no key event log (KEL) is ever needed. Furthermore, in KERI an identifier that is declared as transferable at inception (i.e. supports key rotation) may later be retired by rotating to a null key, after which no further events are accepted into its log.

By definition, a new SCID has a new unique set of controlling keys. Therefore a practical solution for signed statement revocation of statements issued by SCIDs with non-rotatable keys is to rotate the SCID itself to a new SCID. In this case, one may then use Rule 2) to verify any signed statements made with the old identifier's (SCID) keys vis-a-vis the new identifier's (SCID) keys.

The complication to this solution is keeping track of the fact that the identifier has indeed been rotated. Therefore one still needs a VDR (log or ledger) to track which identifier is authoritative for a given function that relies on signed statements for authorization. So, Rule 4 is yet another hybrid. The hybrid is that a VDR (log or ledger) used to track key rotation and signed statements for one SCID with rotatable keys may also be used to track the state of other ephemeral identifiers with non-rotatable keys where those ephemeral identifiers are authorized by the one rotatable identifier. The advantage of Rule 4 is that one VDR (log, ledger) that is in the trust domain of one AID (SCID) may be used to track ephemeral identifiers (AIDs), not simply ephemeral keys. The advantage of ephemeral identifiers is that they are discoverable whereas bare ephemeral keys are not.

4.1.1 Ephemeral Identifier Rotation using Rule 4

KERI employs the model of Rule 4 for witnesses. Witnesses use ephemeral (non-transferable) identifiers where non-transferable means non-rotatable controlling keys. This greatly simplifies the establishment of control authority for the witnesses. As mentioned above, an ephemeral (non-transferable) identifier needs no event log to establish its control authority. Therefore, verifiers do not need to retrieve key event logs to establish the control authority of a given witness in order to verify witness receipts on events. Should a witness become compromised then the controller of the identifier to which the witness is acting as a witness (not the witness' own identifier) rotates out the compromised witness in a key rotation event and rotates in a new witness with a new identifier.

This same strategy may be applied to any management domain where one transferable identifier uses its KEL to track the verifiable authorizing declarations of other ephemeral identifiers. These "authorized" identifiers may be ephemeral because the authorizing declaration statements anchored in its KEL allow the non-ephemeral identifier to track the rotations of the authorized ephemeral identifiers themselves instead of tracking the rotation of the keys for the authorized ephemeral identifiers.

Another example is an IoT application where a marshal maintains a template of the network configuration with a mapping between IoT device identifiers and network addresses. The Marshal may benefit from a transferable identifier but the IoT devices need only use ephemeral (non-transferable) identifiers because should they ever be compromised the marshal may replace their entry in the template with a new identifier (and associated keys).

5 Multi-Valent Key Management Infrastructure for an Entity

The root-of-trust for an AID based trust domain in KERI is the self-certifying identifier prefix for that AID. The security of that root-of-trust is essentially the security of the private key(s) that control that identifier prefix. More generally this is the security of the key management of the associated private keys. Recall that in KERI the keys being managed for control authority are cryptographic signing keys. The essential management functions we call the three R's of key management. These are key reproduction (key pair generation and storage), key recovery, and key rotation. Key rotation depends on the reproduction function for new key generation and storage and key recovery provides redundancy in the key storage or recreation. In addition to these management functions the other important relevant operations are event generation and event signing with those private keys. A useful way of viewing key management is that there are two infrastructures. One is the key-pair generation and storage infrastructure and the other is the key event generation and signing infrastructure. This is shown in the figure below.

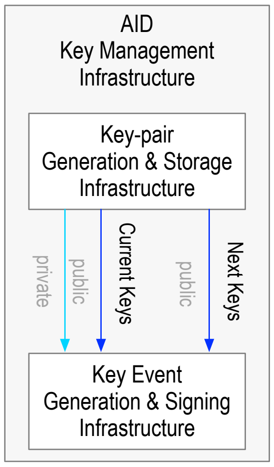

Figure 5.1. Univalent key management infrastructure

The fundamental security principle is that secrets (random seeds or private keys) must remain secret. There are various mechanisms and best practices to protect both the key generation and storage and the event signing infrastructures. Key pairs (public, private) are created in the key-pair generation and storage infrastructure and then may be moved to the key event generation and signing infrastructure in order to sign events. The movement of keys, especially private keys poses a security risk. The signing operation potentially exposes the private key to a side-channel attack exposed by either the movement of private keys from the generation infrastructure to the signing infrastructure or in the signing computing infrastructure itself. Consequently, a given protection mechanism may co-locate both infrastructures. In this case we may refer to this combined infrastructure as a univalent key management infrastructure. A more secure albeit less convenient or performant univalent key management infrastructure may use special computing devices or components to store private keys and/or create signatures. These may include one or more of a secure enclave, a hardware security module (HSM), a trusted platform module (TPM) or some other type of trusted execution environment (TEE). The degree of protection usually forces a trade-off between security, cost, and performance. Typically key generation happens relatively infrequently compared to event signing. But highly secure key generation may not support highly performant signing. This creates an architecture trade-off problem.

The way KERI addresses this security-cost-performance architecture trade-off is via delegation of identifier prefixes. Delegation includes a delegator and a delegate. For this reason we may call this a *cooperative delegation*. This is a somewhat novel form of delegation. A major advantage of cooperative delegation is the delegator's key management protects the delegate's via recovery by the delegator. With cooperative delegation, any exploiter that compromises only the delegate's authoritative keys may not capture control authority of the delegate. Any exploit of the delegate only is recoverable by the delegator. A successful exploiter must also compromise the delegator's authoritative keys. A nested set of layered delegations in a delegation tree, wraps each layer with compromise recovery protection of the next higher layer. This maintains the security of the root layer for compromise recovery all the way out to the leaves in spite of the leaves using less secure key management methods.

To elaborate, in a cooperative delegation, the key generation and storage functions of the delegator and delegate, in terms of the controlling private keys, may be completely isolated from each other. This means that each may use its own independent key management infrastructure with no movement of private keys between the two infrastructures. We call this a bivalent key management infrastructure. This is shown in the following diagram.

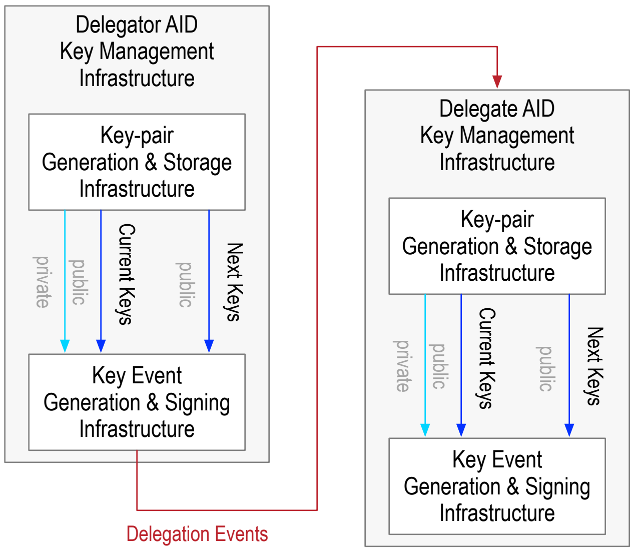
Delegator AID Key Management Infrastructure
Key-pair Generation & Storage Infrastructure
private
public
Current Keys
public
Next Keys
Key Event Generation & Signing Infrastructure
Delegation Events
Delegate AID Key Management Infrastructure
Key-pair Generation & Storage Infrastructure
private
public
Current Keys
public
Next Keys
Key Event Generation & Signing Infrastructure

Figure 5.2. Bivalent delegated key management infrastructure

The only data that needs to move between the two infrastructures (delegator and delegate) comes from the associated key events in the delegations. The movement of signed event data does not pose a security risk in any meaningful sense. A signed event is a verifiable statement that contains no secrets and given sufficient cryptographic strength of the underlying one way functions may not be forged by any practical means. The important feature of a delegated identifier prefix is that recovery from a compromise of the private keys of the delegate may be performed via a rotation event authorized by the delegator. In the worst case the delegator may revoke the delegates identifier prefix and replace it with a new one. This allows a beneficial separation of concerns where the delegator's key management infrastructure may be much more secure but less performant while the delegate's key management infrastructure may be less secure but much more performant. The delegate's infrastructure is still protected by recovery via the more secure delegator's infrastructure. In a bivalent setup, the delegator need only authorize delegation events which happens relatively infrequently so that high performance is not needed. Conversely the delegate may need to sign at scale other statements for various application interactions. The following diagram shows the key event streams for a delegation.

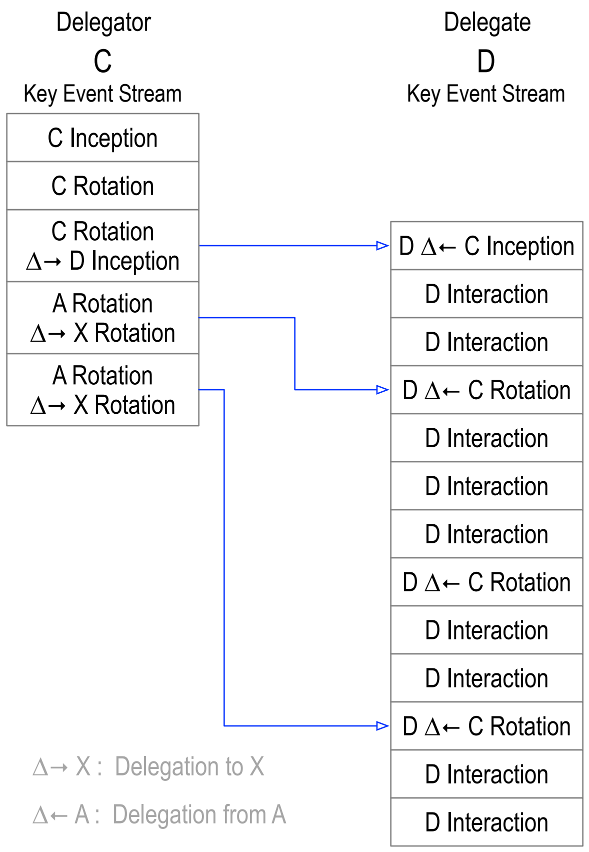

Figure 5.3. Delegation key event streams

This delegation structure may be extended. A delegator may have multiple delegates thereby enabling elastic horizontal scalability. The following diagram shows multiple delegates from a single delegator.

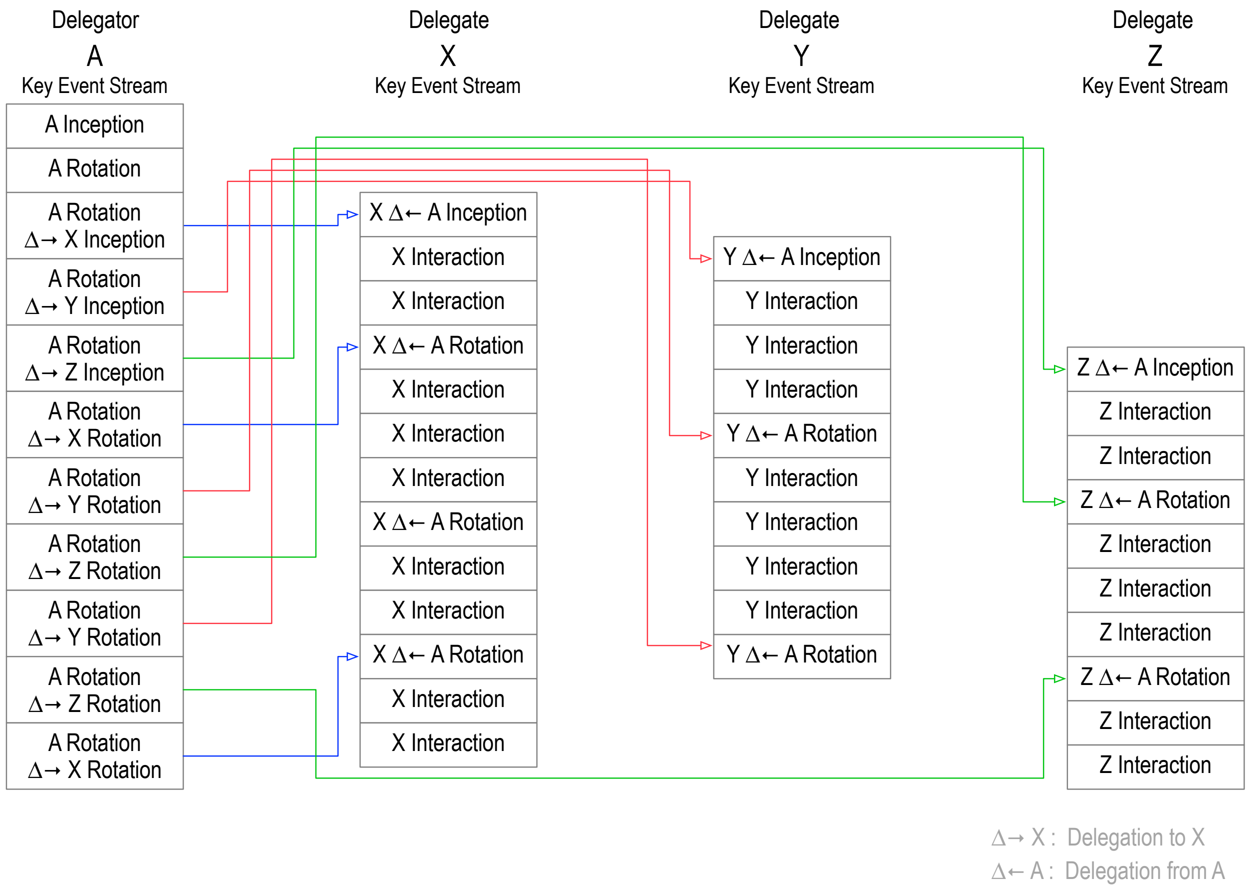

Figure 5.4. Horizontally scaled delegated key event streams

Furthermore, each delegate may act as a delegator for its own delegates to form a nested delegation tree. This allows mapping key management infrastructures to any hierarchically structured organization's computing infrastructure. With this construction, both security and performance trade-offs may be made as appropriate. Such an extended delegation setup we call a multivalent key management infrastructure.

The important feature of such a nested delegation (multivalent) infrastructure is that the delegator, at the next higher level, may always recover, via a key rotation, from a compromised private signing key of any of its delegates in the next lower level. This nested recovery capability extends all the way up to the root identifier prefix. Thus an extremely well protected but less performant root may be used to protect a tree of less protected but more performant branches. This branching structure may be tuned to varying degrees of security-cost-performance trade-offs.

5.1 Root AID for Trust Domain

An entity may provide the root-of-trust for some ecosystem via its root AID. Lets call this the RID for root AID. The RID must be protected using the highest level of security in its key management. Although through the use of a multivalent key management infrastructure, the entity can employ extreme protection of the RID while still enabling more performant key management infrastructure for its operations. The RID should employ a multi-signature self-addressing self-certifying identifier. This should be a minimum of 2 of 3, but 3 of 5 might provide more resistance to signatory unavailability. Suggested for each RID signatory is an air-gapped key generation and storage mechanism. This may include a TPM, TEE, or some other dedicated device.

The event signing infrastructure may also be air-gapped and co-located with the the key generation and storage. However with multi-signature protection other component arrangement are viable.

The RID key events should only be rotation events. This is the most secure use of KERI. This means that each delegation is performed via an extended rotation event. Each delegation event employs a set of first time, one time rotation keys. A set of dedicated witnesses provide the the RID event stream KERL.

5.2 Delegations

Delegated identifiers are used to control the other components at the top level of the ecosystem. These include a delegate to control internally facing infrastructure and multiple delegates to control different externally facing infrastructures. A notional diagram of those delegations is shown in the following diagram.

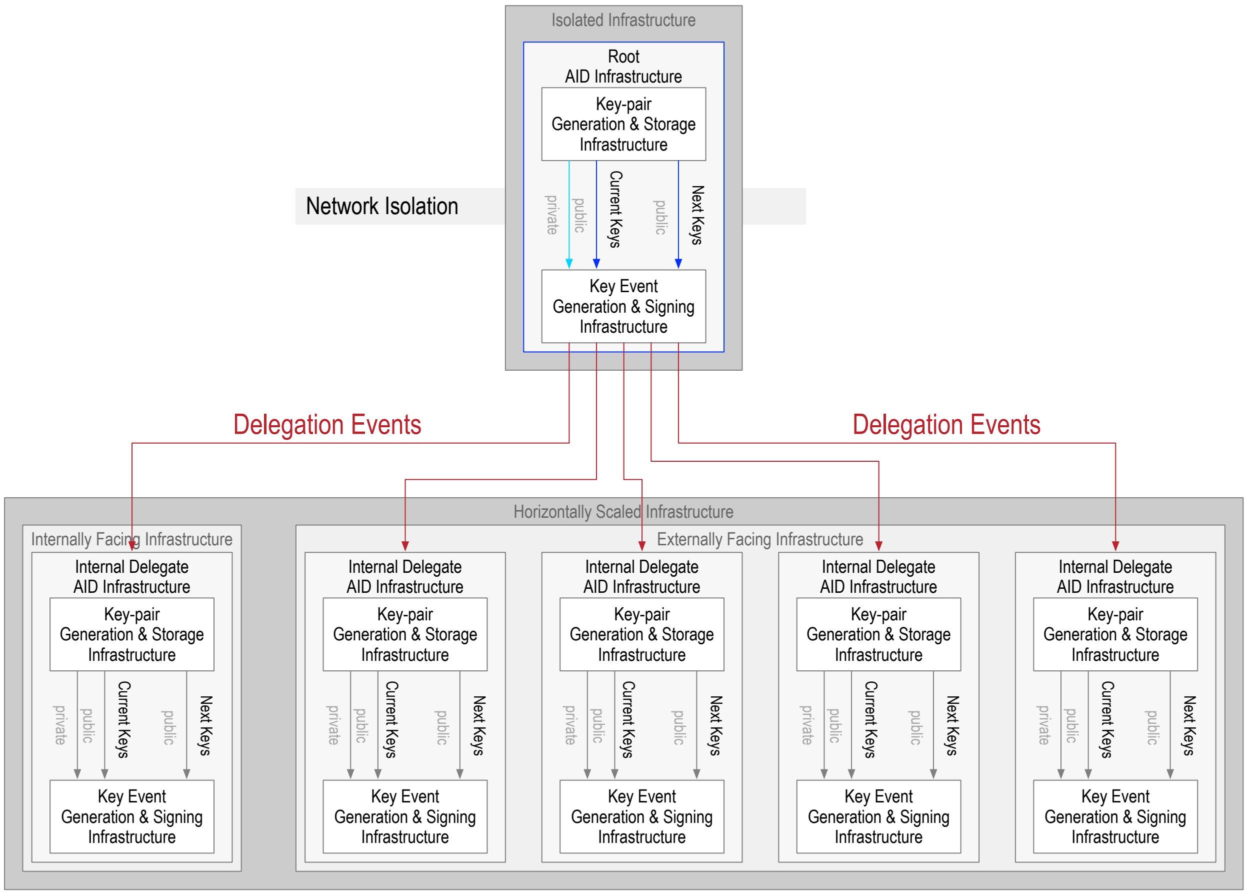
Isolated Infrastructure
Root AID Infrastructure
Key-pair Generation & Storage Infrastructure
private
public
Current Keys
public
Next Keys
Key Event Generation & Signing Infrastructure
Network Isolation
Delegation Events
Delegation Events
Horizontally Scaled Infrastructure
Internally Facing Infrastructure
Externally Facing Infrastructure
Internal Delegate AID Infrastructure

Figure 5.5. Multivalent delegated controller key management infrastructure

This key infrastructure must be online in order to issue verifiable credential VC authorization of LIDs at scale. Each key management infrastructure instance may employ a TPM or other TEE for enhanced security. This is a trade-off.

The next layer down in the delegation hierarchy may employ one of two control mechanisms depending on how tightly the root entity chooses to control that layer. The two control approaches are as follows: The first is a combination of delegated identifiers for each of the next layer down entities coupled with authorizing verifiable credentials (VCs). The second is merely authorizing verifiable credentials with each entity creating their own trust domains via their own root

AIDs. The former case allows the root entity to both recover or revoke delegated AID identifiers not merely the authorizing VCs. The latter case only allows revocation of authorizing VCs and does not enable the root entity to recover or revoke the associated lower lever entity delegated AID. Revocation of the delegated AIDs provides a more comprehensive control mechanism but imposes more of a burden on the root entity for management of those AIDs. The two approaches are shown in the following two diagrams. The first is the delegated AID plus VC approach and the second is the VC only approach.

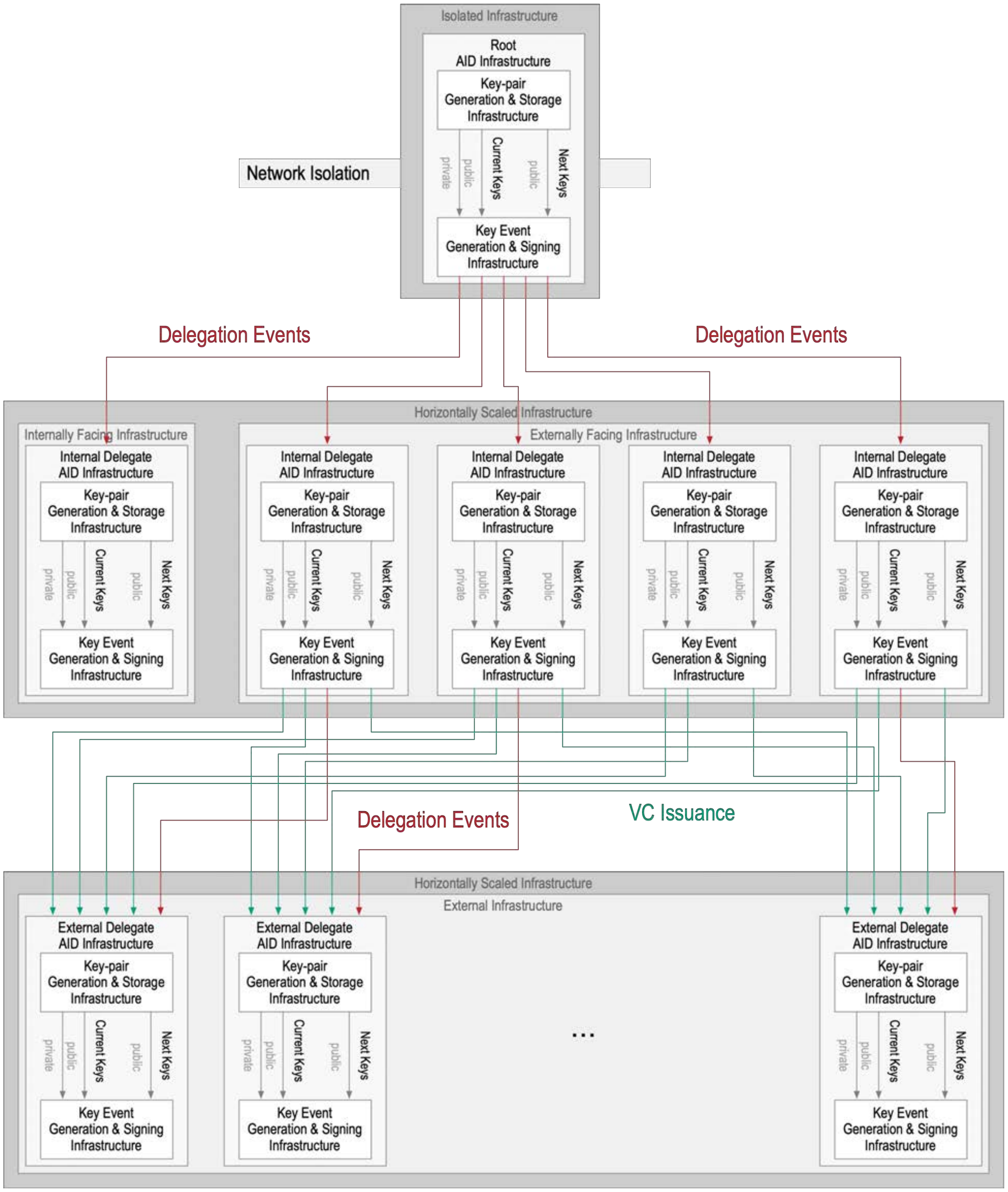
Isolated Infrastructure
Root AID Infrastructure
Key-pair Generation & Storage Infrastructure
private
public
Current Keys
public
Next Keys
Key Event Generation & Signing Infrastructure
Network Isolation
Delegation Events
Delegation Events
Horizontally Scaled Infrastructure
Internally Facing Infrastructure
Externally Facing Infrastructure
Internal Delegate AID Infrastructure
Delegation Events
VC Issuance
Horizontally Scaled Infrastructure
External Infrastructure
External Delegate AID Infrastructure
...

Figure 5.6. Multilevel delegated key management and VC issuance infrastructure

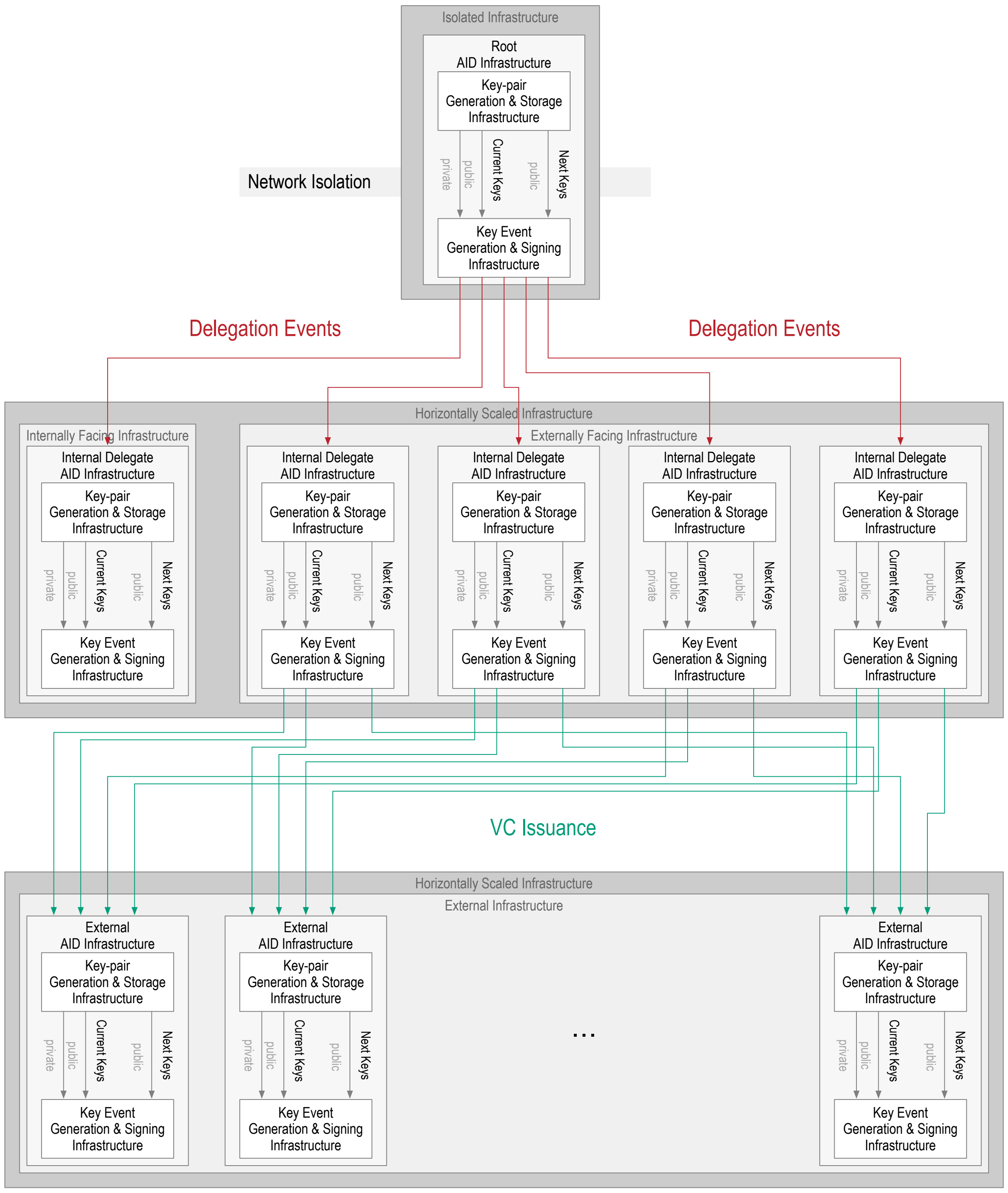
Isolated Infrastructure
Root AID Infrastructure
Key-pair Generation & Storage Infrastructure
private
public
Current Keys
public
Next Keys
Key Event Generation & Signing Infrastructure
Network Isolation
Delegation Events
Delegation Events
Horizontally Scaled Infrastructure
Internally Facing Infrastructure
Externally Facing Infrastructure
Internal Delegate AID Infrastructure
VC Issuance
Horizontally Scaled Infrastructure
External Infrastructure
External AID Infrastructure
...

Figure 5.7. Delegated key management and VC issuance infrastructure

6 Promulgation Infrastructure

As described above KERI enables a separation of control between promulgation and confirmation infrastructure. A given promulgation infrastructure is under the aegis of its controller and a given confirmation infrastructure is under the aegis of its validator. In a multivalent key management infrastructure there may be multiple instances of promulgation infrastructure, one for each controller where each delegator or delegate is a controller in its own right at each layer of the infrastructure. Likewise validators of controllers at each layer may have have multiple instances of confirmation infrastructure.

One of the issues facing a root entity that provides the root-of-trust for an ecosystem occurs when the potential ecosystem participants are diverse and each may want to use different promulgation and/or confirmation infrastructure to best meet their application or industry specific needs. Because KERI is agnostic about the specific promulgation or confirmation infrastructure used in each instance, the root entity may choose to support multiple types of promulgation and confirmation infrastructure as per the needs of its ecosystem participants. Moreover, these choices are not fixed but may change over time as technology and ecosystem's needs evolve.

Specifically, one important class of infrastructure for both promulgation and confirmation are shared distributed ledgers (blockchains) such as Sovrin or Ethereum. In KERI a special case for both a witness and a watcher is a shared ledger oracle. This is shown in the following diagram.

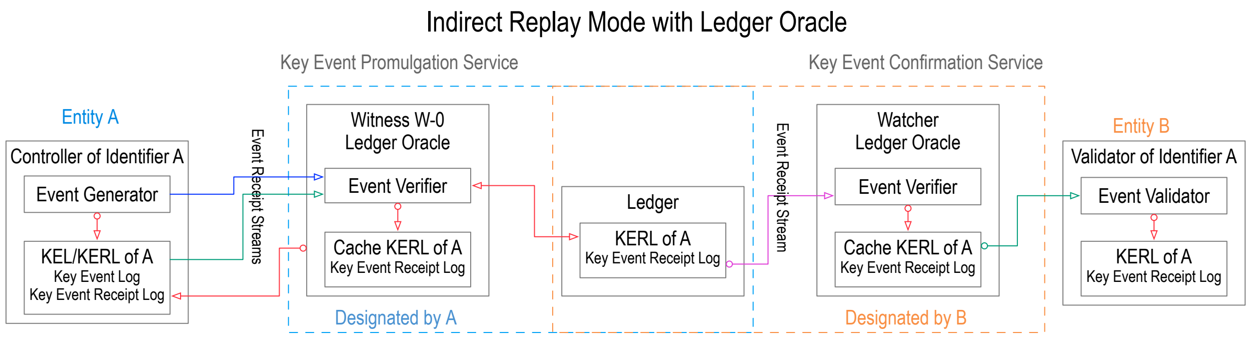

Figure 6.1. Shared ledger oracles for witness and watcher

Given ledger oracles as witnesses and in combination with delegated identifiers controlling those oracles, the root entity may implement an heterogeneous promulgation infrastructure using a combination of cloud hosted witnesses and ledger oracle witnesses for its externally facing infrastructure. Each ledger provides the basis for a network of participants associated with that ledger. The ledger itself is also a network of hosts. Thus it is appropriate to call this a network of networks approach to the witness (promulgation) infrastructure. A notional diagram of this architecture is shown below.

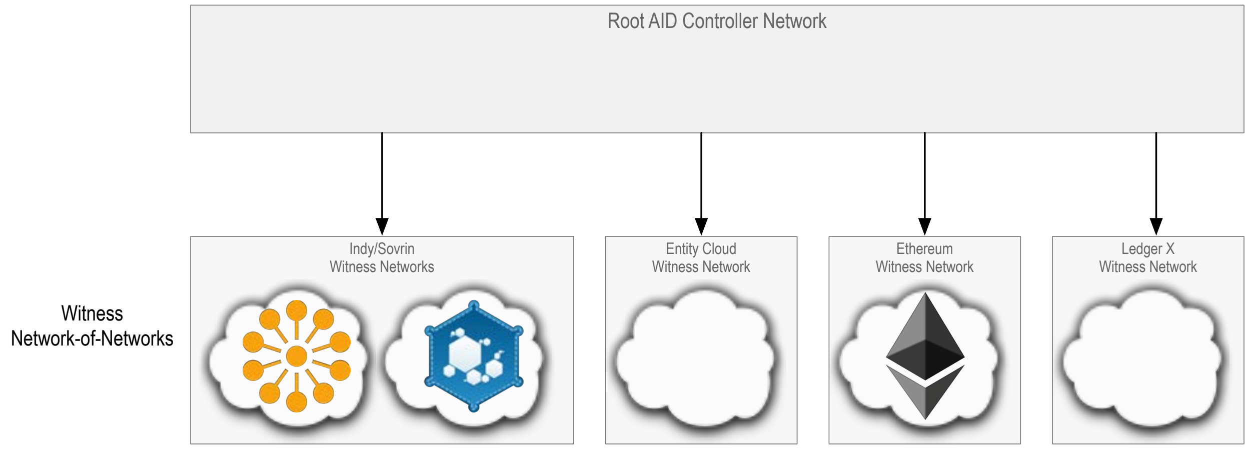

Figure 6.2. Ecosystem witness network-of-networks

In the diagram above, the entity cloud box represents a high performant globally distributed and synchronized cluster of KERI witnesses. The Indy/Sovrin box represents at the very least the Sovrin Foundation MainNet and may eventually include any interoperable Indy networks in the Sovrin ecosystem's own network-of-networks. The Ethereum box represents use of the the

Ethereum Ledger as a witness. Each of these non-native KERI witness networks will need a KERI interoperability graft. This is already anticipated for inclusion in the new Indy DID method. The following diagram shows in more detail a notional architecture for the controller and witness network of networks.

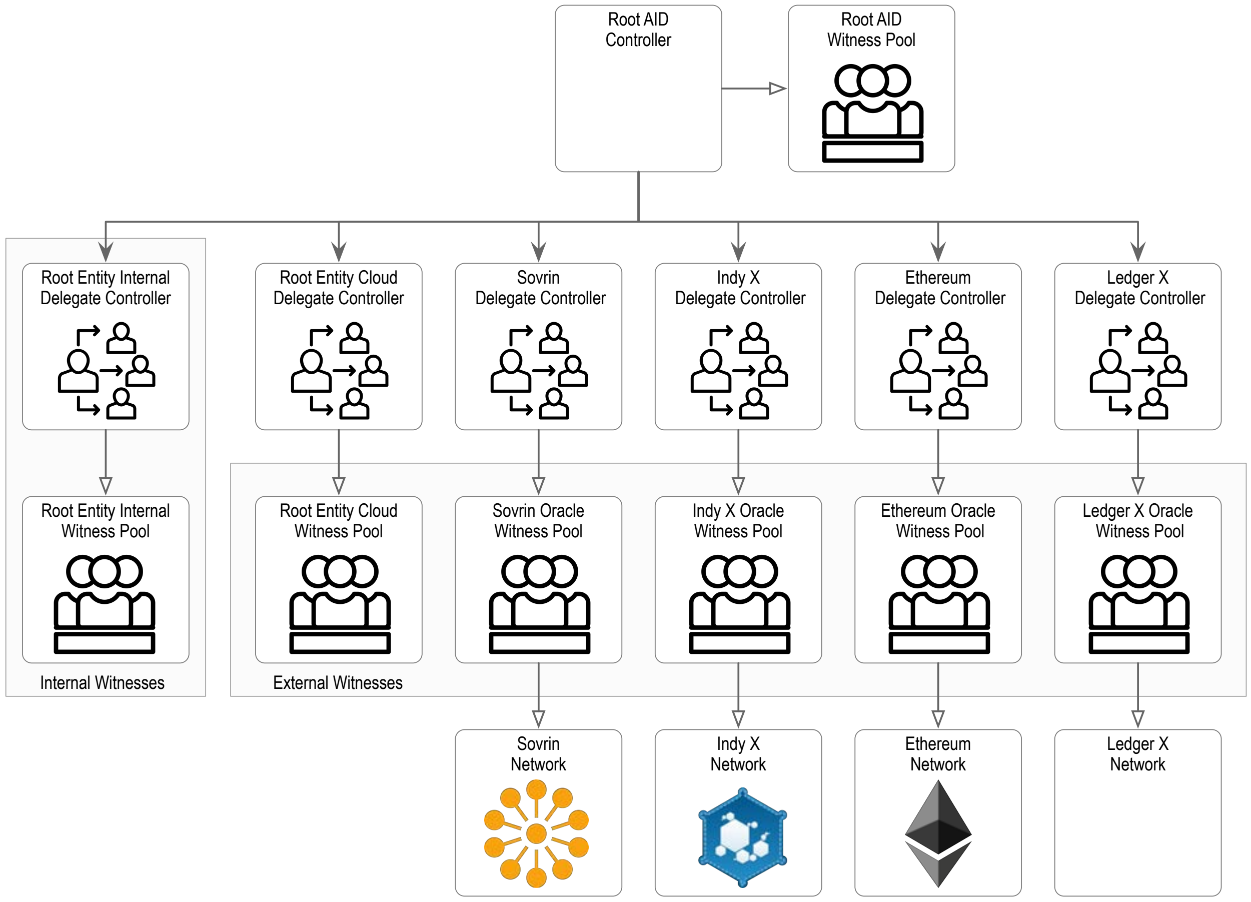
Root AID Controller
Root AID Witness Pool
Root Entity Internal Delegate Controller
Root Entity Cloud Delegate Controller
Sovrin Delegate Controller
Indy X Delegate Controller
Ethereum Delegate Controller
Ledger X Delegate Controller
Root Entity Internal Witness Pool
Root Entity Cloud Witness Pool
Sovrin Oracle Witness Pool
Indy X Oracle Witness Pool
Ethereum Oracle Witness Pool
Ledger X Oracle Witness Pool
Internal Witnesses
External Witnesses
Sovrin Network
Indy X Network
Ethereum Network
Ledger X Network

Figure 6.3. Ecosystem controller and witness network-of-networks

We can expand the diagram above to include the watchers and witnesses of the next layer in the delgation hierarchy. This is shown in the following diagram:

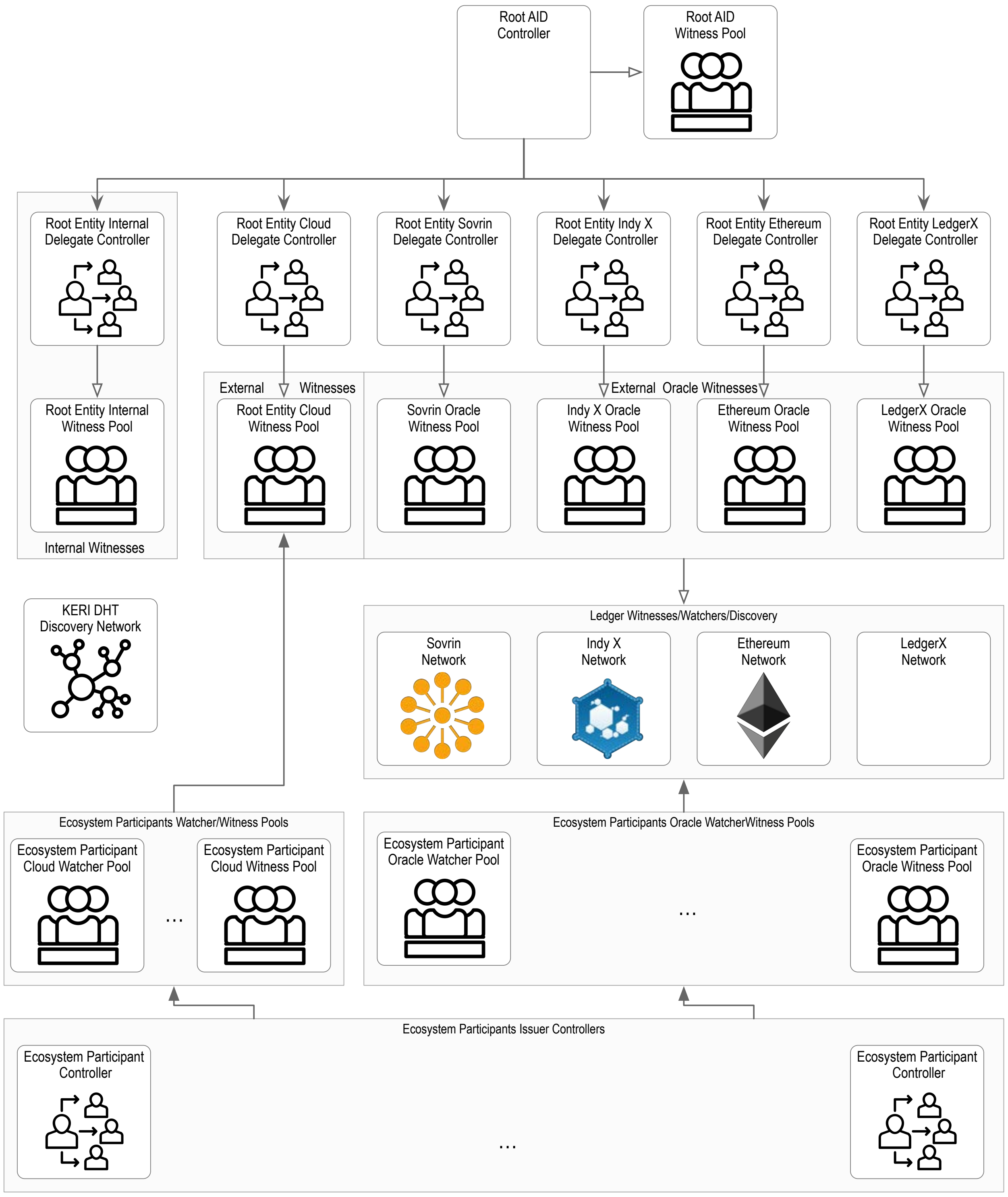
Root AID Controller
Root AID Witness Pool
Root Entity Internal Delegate Controller
Root Entity Cloud Delegate Controller
Root Entity Sovrin Delegate Controller
Root Entity Indy X Delegate Controller
Root Entity Ethereum Delegate Controller
Root Entity LedgerX Delegate Controller
External Witnesses
External Oracle Witnesses
Root Entity Internal Witness Pool
Root Entity Cloud Witness Pool
Sovrin Oracle Witness Pool
Indy X Oracle Witness Pool
Ethereum Oracle Witness Pool
LedgerX Oracle Witness Pool
Internal Witnesses
KERI DHT Discovery Network
Ledger Witnesses/Watchers/Discovery
Sovrin Network
Indy X Network
Ethereum Network
LedgerX Network
Ecosystem Participants Watcher/Witness Pools
Ecosystem Participant Cloud Watcher Pool
...
Ecosystem Participant Cloud Witness Pool
Ecosystem Participants Oracle WatcherWitness Pools
Ecosystem Participant Oracle Watcher Pool
...
Ecosystem Participant Oracle Witness Pool
Ecosystem Participants Issuer Controllers
Ecosystem Participant Controller
...
Ecosystem Participant Controller

Figure 6.4. Delegated Ecosystem participants in network-of-networks

The portability of KERI identifier prefixes is provided via the key event receipt log (KERL) for each prefix. A witness provides a copy of a KERL. The control authority over that prefix may be established by verifying the KERL. The KERL provides a cryptographic proof of control authority that is not dependent on the source of the copy of the KERL. Moreover, each prefix merely contains a universally unique string of cryptographic material. A given prefix may be used to control any and all name spaces that share that same prefix. Because a KERL is only concerned with the identifier prefix not any associated name space. The proof of control authority via the KERL is the same for all the name spaces that share the same prefix and therefore share the same KERL. To clarify, the same KERL may be used to prove control authority over any name space that shares its associated prefix. This includes the disparate name spaces for each of the witness networks shown above. This makes AIDs based on KERI truly portable.

7 Verifiable Credential Issuance Architecture for Ecosystem

As previously discussed in the unified identifier theory section. A given AID forms a trust domain within which LIDs may be verifiable authorized. The root AID, that is, the RID forms the ecosystem trust domain within which the root entity may authorize the issuance of verifiable LIDs. Verifiable issuance of LIDs may be performed using verifiable credentials (VCs). VCs may also be issued as a delegation where a VC issued by one entity authorizes another entity to issue a delegated VCs. The verification logic may then traverse back up the delegation chain to the the root VC issuance. Each VC issuer must have and AID and each VC holder (issued to) must have an AID. These may be either independently generated or delegated AIDs. When the associated entities act under the auspices of the same parent organization then it may make sense to use delegated AIDs for the entities. This has the advantage that the delegated AIDs are published in the delegator's KEL and therefore may not need a separate publication mechanism for verification of authorization to participate in under the auspices of the organization. The delegation events provide that proof of authorization. This delegation chain forms a nested set of trust-domains. This is shown in the following diagram.

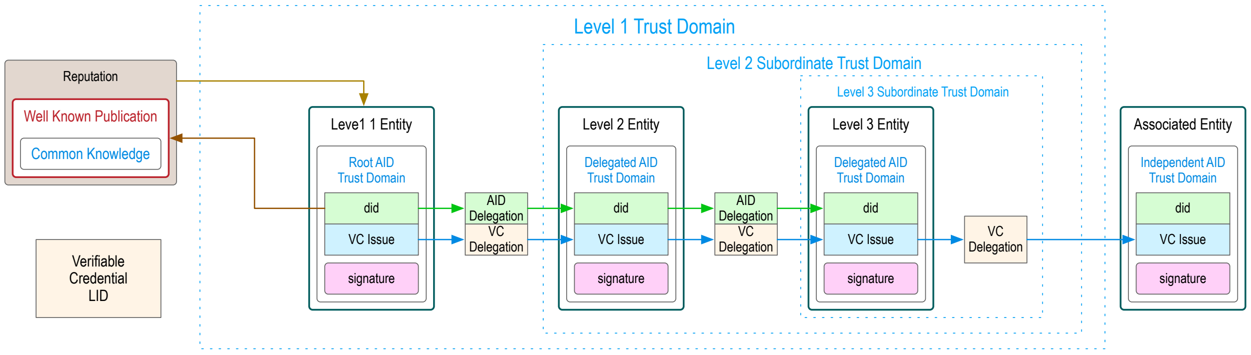

Figure 7.1. Delegated entity chain-of-trust

An important detail to understand about VC issuance is that an organization does not actually issue a VC. Some entity that works for the organization actually issues the VC. This entity is usually a natural person but may be a software agent. Ultimately a software agent is managed by some person. Best practices for securing issuance recognizes that persons are the ultimate source of issuance authorization actions. This means that fraudulent issuance is a vulnerability that is a function of the loyalty and accountability of the issuing person or agent to the issuing organization. One mechanism for increasing issuing accountability is to explicitly identify the issuing persons or agents. There are two mechanisms within KERI for doing that. One is to explicitly track the holders of key pairs. Thus any issuance action may be tracked via the signing keys. This has the drawback that a key rotation is required to remove signing authority. This may not be a problem for senior or trusted company officials but may be problematic for front line workers or software agents. The second method for tracking accountability is to issue authorized issuance capability credentials to front-line workers and agents. This adds a more flexibly revocable issuance accountability. The two methods are best employed in concert with both key holder tracking of the organizational VC and tracking of the issuing AID of a delegated VC issuer. This provides two non-common mode accountability mechanisms that have to both be suborned in order to successfully improperly issue a VC. The following diagram extends the previous nested trust domain issuance diagram to include individual employee delegated AID as tracked delegated VC issuers. This provides a highly fraud resistant approach to VC issuance within and across organizations in an ecosystem.

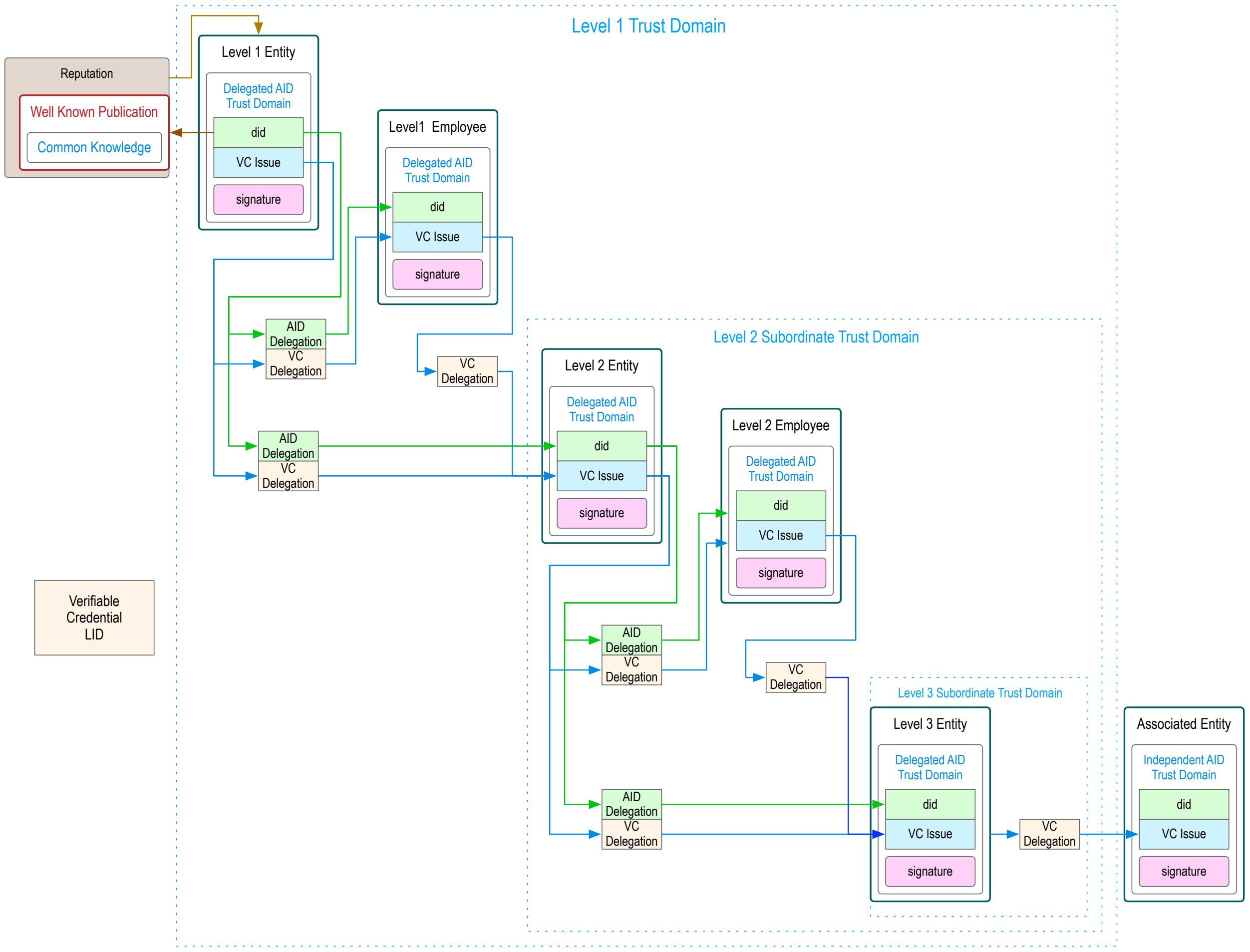

Figure 7.2. Delegated entity with employee chain-of-trust

In the diagram above, every entity labeled Employee could just as well be labeled software agent. The diagram shows a gross level of granularity in terms of employee/agent accountability. For more highly automated VC issuance processes a verifiable data supply chain may be used.

8 Verifiable Algorithms for LID Issuance

As the VC issuance process becomes more highly automated the accountability architecture needs to become more granular. The end result is what we call a verifiable algorithm that uses a provenanced data supply chain. This is described in some detail here [10; 46]. A verifiable algorithm is both an explainable algorithm and a transparent algorithm which achieves fine grained GDPR compliance for transparency and fairness verifiability. Furthermore a verifiable algorithm for VC issuance provides point accountability of the individual software agents that contribute to the issuance of each LID VC. An example of a provenanced data supply chain for verifiable algorithms is shown below.

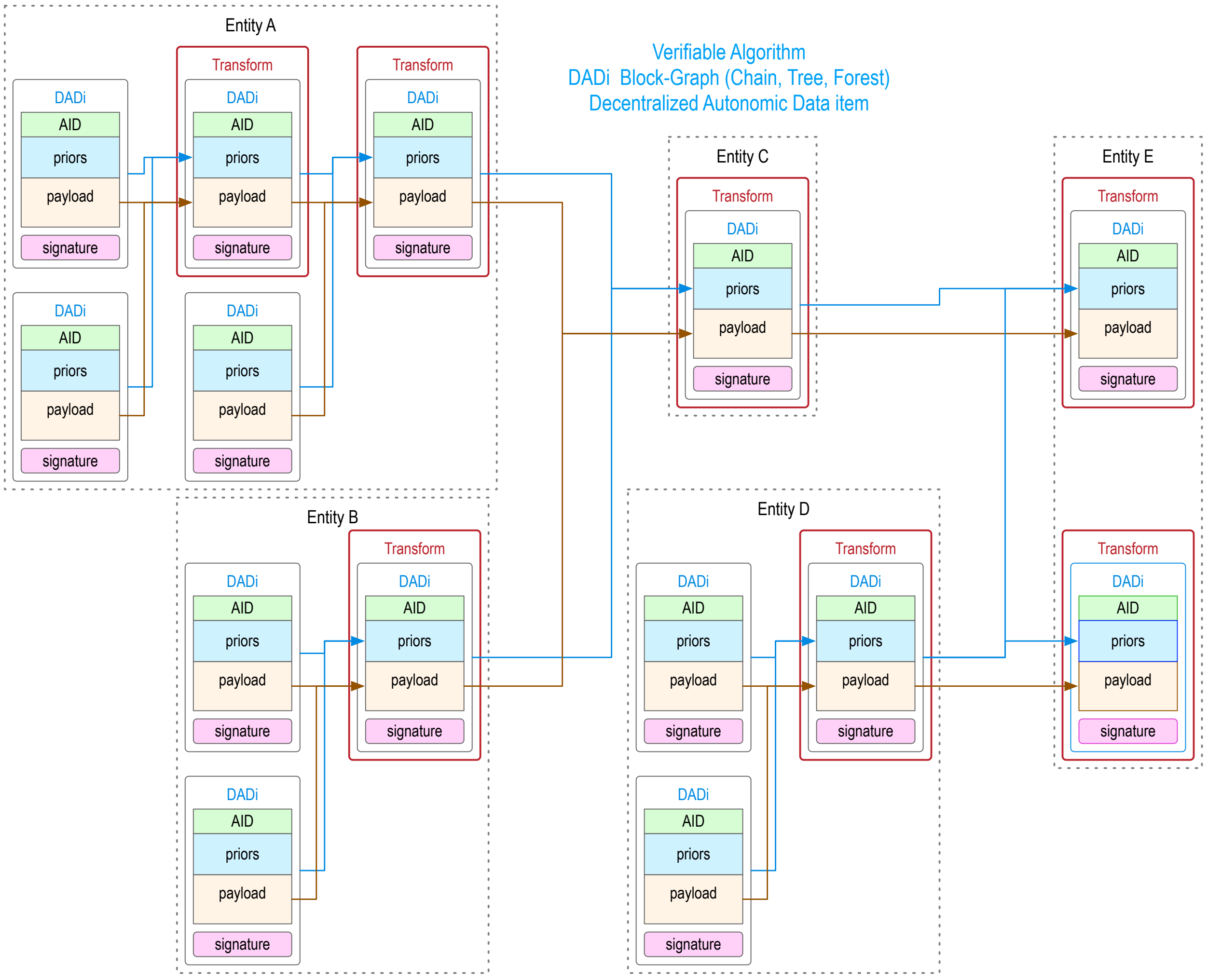

Figure 8.1. Provenanced data supply chain for verifiable algorithmic VC issuance

Each transformation block (with an AID) in the diagram may be a software agent or processing unit that transforms the data provided to it. The data flow shows how each block's transformed data contributes to a final aggregated data product. Because each transform block includes a digest of its priors and each block is signed, then the algorithm itself verifiable. In this case the data product is a VC that authorizes an LID.

9 Conclusion

A unified model for the design of universal identifier systems is presented. The core concept is that an autonomic identifier (AID) may provide a trust domain for authorized human meaningful identifiers issued within the context of that AID. This authorized human meaningful identifiers are thereby legitimized by association with that AID. We call these legitimized identifiers LIDs. The fundamental construct is the couplet, $aid|lid$, where the vertical bar represents the authorization or legitimization of the LID with respect to the trust domain of the AID. Given this model, a verifiable LID issuance infrastructure based on KERI is provided. The model includes the decentralized key management infrastructure as well as the verifiable credential (VC) issuance infrastructure. The design trade-offs are explained. This explains the methodology for design and implementation using an autonomic (end-verifiable cryptographic) root-of-trust. This enables true portability of the associated identifiers across different support infrastructures both in time and space. This is an innovative approach that mitigates the primary security vulnerabilities of an administrative (trusted entity) root-of-trust and the primary portability limitations of an algorithmic (shared ledger) root-of-trust. This provides a security model for truly decentralized internet based ecosystems where each ecosystem may control is own trust domain but inter-operably interact with any other trust domain.

Acknowledgments

The author wishes to thank all those that provided helpful feedback.

Author

Samuel M. Smith Ph.D. has a deep interest in decentralized identity and reputation systems. Samuel received a Ph.D. in Electrical and Computer Engineering from Brigham Young University in 1991. He then spent 10 years at Florida Atlantic University, eventually reaching full professor status. In addition to decentralized identity and reputation, he has performed pioneering research in automated reasoning, machine learning, and autonomous vehicle systems. He has over 100 refereed publications in these areas and was principal investigator on numerous federally funded research projects. Dr. Smith has been an active participant in open standards development for networking protocols, and decentralized identity. He is also a serial entrepreneur.

References

[1] "Global Location Numbers (GLN): A key enabler for improving efficiency and visibility of the supply and demand chains,"

https://www.gs1.org/docs/idkeys/GS1_Global_Location_Numbers.pdf

[2] "A Deep Dive on the Recent Widespread DNS Hijacking Attacks," KrebsonSecurity, 2019/02/19

https://krebsonsecurity.com/2019/02/a-deep-dive-on-the-recent-widespread-dns-hijacking-attacks/

[3] "A Universally Unique Identifier (UUID) URN Namespace," IETF RFC-4122, 2005/07/01

https://tools.ietf.org/html/rfc4122

[4] Birge-Lee, H., Sun, Y., Edmundson, A., Rexford, J. and Mittal, P., "Using BGP to acquire bogus TLS certificates," vol. Workshop on Hot Topics in Privacy Enhancing Technologies, no. HotPETs 2017, 2017

[5] Birge-Lee, H., Sun, Y., Edmundson, A., Rexford, J. and Mittal, P., "Bamboozling certificate authorities with {BGP}," vol. 27th {USENIX} Security Symposium, no. {USENIX} Security 18, pp. 833-849, 2018

https://www.usenix.org/conference/usenixsecurity18/presentation/birge-lee

[6] Birge-Lee, H., Wang, L., Rexford, J. and Mittal, P., "Sico: Surgical interception attacks by manipulating bgp communities," vol. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 431-448, 2019

https://dl.acm.org/doi/abs/10.1145/3319535.3363197

[7] "Byzantine fault tolerance," Wikipedia,

https://en.wikipedia.org/wiki/Byzantine_fault_tolerance

[8] Castro, M. and Liskov, B., "Practical Byzantine fault tolerance," vol. OSDI 99, pp. 173-186, 1999

http://www.pmg.lcs.mit.edu/papers/osdi99.pdf

[9] Clement, A., Wong, E. L., Alvisi, L., Dahlin, M. and Marchetti, M., "Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults.," vol. NSDI 9, pp. 153-168, 2009

http://static.usenix.org/events/nsdi09/tech/full_papers/clement/clement.pdf

[10] Conway, S., Hughes, A., Ma, M. et al., "A DID for Everything," Rebooting the Web of Trust RWOT 7, 2018/09/26

https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/A_DID_for_everything.pdf

[11] Cooper, D., Santesson, S., Farrell, S. et al., "RFC 5280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," IETF, 2008/05/01
https://tools.ietf.org/html/rfc5280

[12] "DIF KERI Python Implementation Repository (keripy)," Decentralized Identity Foundation,
https://github.com/decentralized-identity/keripy

[13] Dingle, P., Hammann, S., Hardman, D., Winczewski, C. and Smith, S., "Alice Attempts to Abuse a Verifiable Credential," RWOT 9, 2020/01/24
https://github.com/WebOfTrustInfo/rwot9-prague/blob/master/final-documents/alice-attempts-abuse-verifiable-credential.pdf

[14] "DNS Certification Authority Authorization,"
https://en.wikipedia.org/wiki/DNS_Certification_Authority_Authorization

[15] "Domain Name System," Wikipedia,
https://en.wikipedia.org/wiki/Domain_Name_System

[16] "EJC-RFID Info: Item Level Identification," epc-rfid.info,
https://www.epc-rfid.info/sgtin

[17] "EPC Tag Data Standard 1.11," GS1, 2017/09/01
https://www.gs1.org/sites/default/files/docs/epc/GS1_EPC_TDS_i1_11.pdf

[18] "Ethereum," Ethereum Foundation,
http://www.ethdocs.org/en/latest/introduction/index.html

[19] Gavrichenkov, A., "Breaking HTTPS with BGP Hijacking," BlackHat, 2015
https://www.blackhat.com/docs/us-15/materials/us-15-Gavrichenkov-Breaking-HTTPS-With-BGP-Hijacking-wp.pdf

[20] Girault, M., "Self-certified public keys," EUROCRYPT 1991: Advances in Cryptology, pp. 490-497, 1991
https://link.springer.com/content/pdf/10.1007%2F3-540-46416-6_42.pdf

[21] "GLN Allocation Rules," GS1,
https://www.gs1.org/glnrules

[22] "Global Location Number (GLN)," GS1,
https://www.gs1.org/standards/id-keys/gln

[23] Goodin, D., "A DNS hijacking wave is targeting companies at an almost unprecedented scale," Ars Technica, 2019/01/10
https://arstechnica.com/information-technology/2019/01/a-dns-hijacking-wave-is-targeting-companies-at-an-almost-unprecedented-scale/

[24] Grant, A. C., "Search for Trust: An Analysis and Comparison of CA System Alternatives and Enhancements," Dartmouth Computer Science Technical Report TR2012-716, 2012
https://pdfs.semanticscholar.org/7876/380d71dd718a22546664b7fcc5b413c1fa49.pdf

[25] "GS1 DataMatrix Guideline 2.5.1: Overview and technical introduction to the use of GS1 DataMatrix," GS1, 2018/01/01
https://www.gs1.org/docs/barcodes/GS1_DataMatrix_Guideline.pdf

[26] "GS1 General Specifications 20.0," GS1, 2020/01/01
https://www.gs1.org/docs/barcodes/GS1_General_Specifications.pdf

[27] "GS1 GLN Allocation Rules Standard 2.0.1," GS1, 2017/01/01
https://www.gs1.org/docs/barcodes/GS1_GLN_Allocation_Rules.pdf

[28] "GS1 Lightweight Messaging Standard for Verification of Product Identifiers 1.1," GS1, 2019/07/01
https://www.gs1.org/sites/default/files/docs/standards/gs1_lightweight_verification_messaging_standard_v1-1.pdf

[29] "How Cybercrime Exploits Digital Certificates," InfoSecInstitute, 2014/07/28
https://resources.infosecinstitute.com/cybercrime-exploits-digital-certificates/#gref

[30] "How to Translate a U.P.C. to a GTIN to an SGTIN to an EPC," GS1,
https://www.gs1us.org/DesktopModules/Bring2mind/DMX/Download.aspx?Command=Core_Download&EntryId=389&language=en-US&PortalId=0&TabId=134

[31] "Introducing the Legal Entity Identifier (LEI)," GLEIF,
https://www.gleif.org/en/about-lei/introducing-the-legal-entity-identifier-lei#

[32] "ISO 17442-1:2020
Financial services — Legal entity identifier (LEI) — Part 1: Assignment," ISO, 2020/08/01
https://www.iso.org/standard/78829.html

[33] "ISO 17442-2:2020
Financial services — Legal entity identifier (LEI) — Part 2: Application in digital certificates," ISO, 2020/08/01
https://www.iso.org/standard/79917.html

[34] Kaminsky, M. and Banks, E., "SFS-HTTP: Securing the Web with Self-Certifying URLs," MIT, 1999
https://pdos.csail.mit.edu/~kaminsky/sfs-http.ps

[35] "KERI Project DIF," Decentralized Identity Foundation,
https://github.com/decentralized-identity/keri

[36] Mazieres, D. and Kaashoek, M. F., "Escaping the Evils of Centralized Control with self-certifying pathnames," MIT Laboratory for Computer Science, 2000
http://www.sigops.org/ew-history/1998/papers/mazieres.ps

[37] Mazieres, D., "Self-certifying File System," MIT Ph.D. Dissertation, 2000/06/01
https://pdos.csail.mit.edu/~ericp/doc/sfs-thesis.ps

[38] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)," IETF 8615, 2019/05/01
https://tools.ietf.org/html/rfc8615

[39] "Remote ATtestation proceduresS (RATS) WG," IETF,
https://datatracker.ietf.org/wg/rats/about/

[40] Serrano, N., Hadan, H. and Camp, L. J., "A complete study of PKI (PKI's Known Incidents)," Available at SSRN 3425554, 2019
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3425554

[41] Shae, M., Smith, S. M. and Stocker, C., "Decentralized Identity as a Meta-platform: How Cooperation Beats Aggregation," Rebooting the Web of Trust, vol. RWOT 9, 2019/11/19
https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/CooperationBeatsAggregation.pdf

[42] Smith, S. M., "Kery Event Receipt Infrastructure (KERI) Design," Github, 2020/04/22
https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/KERI_WP_2.x.web.pdf

[43] Smith, S. M., "Open Reputation Framework," vol. Version 1.2, 2015/05/13
https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/open-reputation-low-level-whitepaper.pdf

[44] Smith, S. M. and Khovratovich, D., "Identity System Essentials," 2016/03/29
https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/Identity-System-Essentials.pdf

[45] Smith, S. M., "Meta-Platforms and Cooperative Network-of-Networks Effects: Why Decentralized Platforms Will Eat Centralized Platforms," SelfRule, 2018/04/25
https://medium.com/selfrule/meta-platforms-and-cooperative-network-of-networks-effects-6e61eb15c586

[46] Smith, S. M., "Decentralized Autonomic Data (DAD) and the three R's of Key Management," Rebooting the Web of Trust RWOT 6, Spring 2018
https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/DecentralizedAutonomicData.pdf

[47] "Sovrin: A Protocol and Token for Self- Sovereign Identity and Decentralized Trust," Sovrin.org, 2018/01/01

https://sovrin.org/wp-content/uploads/2018/03/Sovrin-Protocol-and-Token-White-Paper.pdf

[48] Stevens, G., "DNS Poisoning Attacks: A Guide for Website Admins," HashedOut, 2020/01/21
https://www.thesslstore.com/blog/dns-poisoning-attacks-a-guide-for-website-admins/

[49] Stocker, C., Smith, S. and Caballero, J., "Quantum Secure DIDs," RWOT10, 2020/07/09
https://github.com/WebOfTrustInfo/rwot10-buenosaires/blob/master/final-documents/quantum-secure-dids.pdf

[50] Szabo, N., "Secure Property Titles with Owner Authority," 1998
https://nakamotoinstitute.org/secure-property-titles/

[51] Szabo, N., "Advances in Distributed Security," 2003
https://nakamotoinstitute.org/advances-in-distributed-security/

[52] TCG, "Implicit Identity Based Device Attestation," Trusted Computing Group, vol. Version 1.0, 2018/03/05
https://trustedcomputinggroup.org/wp-content/uploads/TCG-DICE-Arch-Implicit-Identity-Based-Device-Attestation-v1-rev93.pdf

[53] "Transport Layer Security," Wikipedia,
https://en.wikipedia.org/wiki/Transport_Layer_Security

[54] "Uniform Resource Identifier (URI): Generic Syntax," IETF RFC-3986, 2005/01/01
https://tools.ietf.org/html/rfc3986

[55] "Verifiable Credentials Data Model," W3C Candidate Recommendation,
https://w3c.github.io/vc-data-model/

[56] "Verification Messaging: GS1 Lightweight Messaging Standard for Verification of Product Identifiers," GS1,
https://www.gs1.org/verification-messaging

[57] W3C, "Decentralized Identifiers (DIDs)," W3C Draft Community Group Report,
https://www.w3.org/TR/did-core/

[58] "Well Known DID Configuration: DIF Working Group Approved Draft," DIF,
https://identity.foundation/.well-known/resources/did-configuration/

[59] Windley, P. J., "The Architecture of Identity Systems," Technometria, 2020/09/28
https://www.windley.com/archives/2020/09/the_architecture_of_identity_systems.shtml

[60] "X.509," Wikipedia,
https://en.wikipedia.org/wiki/X.509

[61] "Zooko's Triange," Wikipedia,
https://en.wikipedia.org/wiki/Zooko%27s_triangle