This is a test document

Matthias Valvekens

2022-11-29

Foreword

The foreword goes here.

Introduction

This document presents identity system-based secure overlay for the Internet. This system is based on a Key Event Receipt Infrastructure (KERI) or the KERI protocol {{KERI}}{{KERI-ID}}{{RFC0791}}. The primary key management operation is key Rotation (transference) via a novel key pre-rotation scheme as the background for the acronym KERI {{DAD}}{{KERI}}.

The main motivation for this work is to provide a secure decentralized foundation of attributional trust for the Internet as a trustable spanning layer in the form of an identifier system security overlay. This identifier system security overlay provides verifiable authorship (authenticity) of any message or data item via secure (cryptographically verifiable) attribution to a cryptonymous (cryptographic strength pseudonymous self-certifying identifier (SCID){{KERI}}{{UIT}}{{SCPK}}{{SFS}}{{SCPN}}{{SCURL}}{{PKI}}.

Thus, KERI addresses a major flaw in the original design of the Internet Protocol (IP) which is that the Internet has no security layer(s) (i.e., Session or Presentation layers) to provide interoperable verifiable authenticity {{RFC0791}}. There was no built-in mechanism for secure attribution to the source of a packet. Specifically, the IP packet header includes a source address field that indicates the IP address of the device that sent the packet. Anyone (including any intermediary) can forge an IP packet. Because the source address of such a packet can be undetectably forged, a recipient may not be able to ascertain when or if the packet was sent by an imposter. This means that secure attribution mechanisms for the Internet must be overlaid (bolted-on). KERI provides such a security overlay, specifically an identifier system security overlay.

# Scope

The identity system-based secure overlay for the Internet, based on KERI, includes a primary root-of-trust in Self-certifying identifiers (SCIDs) {{UIT}}{{SCPK}}{{SFS}}{{SCPN}}{{SCURL}}. This root-of-trust presents a formalism for Autonomic identifiers (AIDs) and Autonomic namespaces (ANs), which are part of an Autonomic identity system (AIS). This system uses the design principle of minimally sufficient means to provide a candidate trust spanning layer for the Internet. Associated with this system is a Decentralized key management infrastructure (DKMI).

The primary root-of-trust are SCIDs that are strongly bound at issuance to a cryptographic signing (public, private) keypair, which is self-contained until/unless control needs to be transferred to a new keypair. In that event, an append-only chained key-event log of signed transfer statements provides end-verifiable control provenance. This makes intervening operational infrastructure replaceable because the event logs may be served up by any infrastructure including ambient infrastructure. End verifiable logs on ambient infrastructure enable ambient verifiability (verifiable by anyone, anywhere, at any time).

The primary key management operation is key Rotation (transference) via a novel key pre-rotation scheme {{DAD}}{{KERI}}. Two primary trust modalities motivated the design, a direct (one-to-one) mode and an indirect (one-to-any) mode. The indirect mode depends on witnessed Key event receipt logs (KERL) as a secondary root-of-trust for validating events., which is the background for the acronym KERI, Key event receipt infrastructure. In the direct mode, the identity Controller establishes control via verified signatures of the controlling keypair. The indirect mode extends that trust basis with witnessed KERL for validating events. The security and accountability guarantees of indirect mode are provided by KERI’s Agreement Algorithm for Control Establishment (KA2CE) among a set of Witnesses.

The KA2CE approach may be much more performant and scalable than more complex approaches that depend on a total ordering distributed consensus ledger. Nevertheless, KERI may employ a distributed consensus ledger when other considerations make it the best choice. The KERI approach to DKMI allows for a more granular composition. Moreover, because KERI is event streamed it enables DKMI that operates in-stride with data events streaming applications such as web 3.0, IoT, and others where performance and scalability are more important. The core KERI engine is identifier namespace independent. This makes KERI a candidate for a universal portable DKMI {{KERI}}{{KERI-ID}}{{UIT}}. # Normative references

There are no normative references in this document. # Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

* ISO Online browsing platform: available at <https://www.iso.org/obp>
* IEC Electropedia: available at <http://www.electropedia.org/>

Primitive

a serialization of a unitary value. A cryptographic Primitive is the serialization of a value associated with a cryptographic operation including but not limited to a digest (hash), a salt, a seed, a private key, a public

Qualified

when Qualified, a cryptographic Primitive includes a prepended derivation code (as a proem) that indicates the cryptographic algorithm or suite used for that derivation. This simplifies and compactifies the essential information needed to use that cryptographic Primitive. All cryptographic Primitives expressed in either text or binary CESR are qualified by definition {{CESR-ID}}. Qualification is an essential property of CESR {{CESR-ID}}.

Cryptonym

a cryptographic pseudonymous identifier represented by a string of characters derived from a random or pseudo-random secret seed or salt via a one-way cryptographic function with a sufficiently high degree of cryptographic strength (e.g. 128 bits, see appendix on cryptographic strength) {{OWF}}{{COWF}}{{TMCrypto}}{{QCHC}}. A Cryptonym is a type of Primitive. Due the enctropy in its derivation, a Cryptonym is a universally unique identifier and only the Controller of the secret salt or seed from which the Cryptonym is derived may prove control over the Cryptonym. Therefore, the derivation function must be associated with the Cryptonym and may be encoded as part of the Cryptonym itself.

Self-certifying identifier (SCID)

a type of Cryptonym that is uniquely cryptographically derived from the public key of an asymmetric non-repudiable signing keypair, (public, private).

Autonomic identifier (AID)

a self-managing cryptonymous identifier that must be self-certifying (self-authenticating) and must be encoded in CESR as a qualified cryptographic Primitive.

Key state

includes the set of currently authoritative keypairs for an AID and any other information necessary to secure or establish control authority over an AID.

Key event

concretely, the serialized data structure of an entry in the Key event log (KEL) for an AID. Abstractly, the data structure itself. Key events come in different types and are used primarily to establish or change the authoritative set of keypairs and/or anchor other data to the authoritative set of keypairs at the point in the KEL actualized by a particular entry.

Establishment event

a key event that establishes or changes the Key state which includes the current set of authoritative keypairs (Key state) for an AID.

Non-establishment event

a key event that does not change the current Key state for an AID. Typically, the purpose of a non-establishment event is to anchor external data to a given Key state as established by the most recent prior Establishment event for an AID.

Inception event

an Establishment Event that provides the incepting information needed to derive an AID and establish its initial Key state.

Inception

the operation of creating an AID by binding it to the initial set of authoritative keypairs and any other associated information. This operation is made verifiable and Duplicity evident upon acceptance as the Inception event that begins the AID’s KEL.

Rotation event

an Establishment event that provides the information needed to change the Key state which includes a change to the set of authoritative keypairs for an AID.

Rotation

the operation of revoking and replacing the set of authoritative keypairs for an AID. This operation is made verifiable and Duplicity evident upon acceptance as a Rotation event that is appended to the AID’s KEL.

Interaction event

a Non-establishment Event that anchors external data to the Key state as established by the most recent prior Establishment event.

Key event log (KEL)

a Verifiable data structure that is a backward and forward chained, signed, append-only log of key events for an AID. The first entry in a KEL must be the one and only Inception event of that AID.

Version

more than one version of a KEL for an AID exists when for any two instances of a KEL at least one event is unique between the two instances.

Verifiable

a KEL is verifiable means all content in a KEL including the digests and the signatures on that content is verifiably compliant with respect to the KERI protocol. In other words, the KEL is internally consistent and has integrity vis-a-vis its backward and forward chaining digests and authenticity vis-a-vis its non-repudiable signatures. As a Verifiable data structure, the KEL satisfies the KERI protocol-defined rules for that verifiability. This includes the cryptographic verification of any digests or signatures on the contents so digested or signed.

Duplicity

the existence of more than one version of a verifiable KEL for a given AID. Because every event in a KEL must be signed with non-repudiable signatures any inconsistency between any two instances of the KEL for a given AID is provable evidence of Duplicity on the part of the signers with respect to either or both the Key state of that AID and/or any anchored data at a given Key state. A shorter KEL that does not differ in any of its events with respect to another, but a longer KEL is not duplicitous but merely incomplete. To clarify, Duplicity evident means that Duplicity is provable via the presentation of a set of two or more mutually inconsistent but independently verifiable instances of a KEL.

Verifier

any entity or agent that cryptographically verifies the signature(s) and/or digests on an event message. In order to verify a signature, a Verifier must first determine which set of keys are or were the controlling set for an identifier when an event was issued. In other words, a Verifier first must establish control authority for an identifier. For identifiers that are declared as non-transferable at inception, this control establishment merely requires a copy of the Inception event for the identifier. For identifiers that are declared transferable at Inception, this control establishment requires a complete copy of the sequence of Establishment events (Inception and all Rotations) for the identifier up to the time at which the statement was issued.

Validator

any entity or agent that evaluates whether or not a given signed statement as attributed to an identifier is valid at the time of its issuance. A valid statement must be verifiable, that is, has a verifiable signature from the current controlling keypair(s) at the time of its issuance. Therefore, a Validator first must act as a Verifier in order to establish the root authoritative set of keys. Once verified, the Validator may apply other criteria or constraints to the statement in order to determine its validity for a given use case. When that statement is part of a Verifiable data structure then the cryptographic verification includes verifying digests and any other structural commitments or constraints. To elaborate, with respect to an AID, for example, a Validator first evaluates one or more KELs in order to determine if it can rely on (trust) the Key state (control authority) provided by any given KEL. A necessary but insufficient condition for a valid KEL is it is verifiable i.e., is internally inconsistent with respect to compliance with the KERI protocol. An invalid KEL from the perspective of a Validator may be either unverifiable or may be verifiable but duplicitous with respect to some other verifiable version of that KEL. Detected Duplicity by a given Validator means that the Validator has seen more than one verifiable version of a KEL for a given AID. Reconcilable Duplicity means that one and only one version of a KEL as seen by a Validator is accepted as the authoritative version for that validator. Irreconcilable Duplicity means that none of the versions of a KEL as seen by a Validator are accepted as the authoritative one for that Validator. The conditions for reconcilable duplicity are described later.

Message

consists of a serialized data structure that comprises its body and a set of serialized data structures that are its attachments. Attachments may include but are not limited to signatures on the body.

Key event message

a Message whose body is a Key event and whose attachments may include signatures on its body.

Key event receipt

a Message whose body references a Key event and whose attachments must include one or more signatures on that Key event. # KERI foundational overview {#sec:content}

TODO

# KERI’s identifier system security overlay

## Overcoming existing security overlay flaws

The function of KERI’s identifier system security overlay is to establish the authenticity (or authorship) of the message payload in an IP Packet by verifiably attributing it to a cryptonymous Self-certifying identifier (an AID) via an attached set of one or more asymmetric keypair-based non-repudiable digital signatures. The current valid set of associated asymmetric keypair(s) is proven via the KEL {{KERI}}{{VDS}}{{ESMT}}{{RT}}. The identifier system provides a mapping between the identifier and the keypair(s) that control the identifier, namely, the public key(s) from those keypairs. The private key(s) is secret and is not shared.

An authenticatable (verifiable) internet message (packet) or data item includes the identifier and data in its payload. Attached to the payload is a digital signature(s) made with the private key(s) from the controlling keypair(s). Given the identifier in a message, any verifier of a message (data item) can use the identifier system mapping to look up the public key(s) belonging to the controlling keypair(s). The Verifier can then verify the attached signature(s) using that public key(s). Because the payload includes the identifier, the signature makes a non-repudiable cryptographic commitment to both the source identifier and the data in the payload.

KERI overcomes two major security overlay flaws. The first major flaw is that the mapping between the identifier (domain name) and the controlling keypair(s) is merely asserted by a trusted entity e.g., certificate authority (CA) via a certificate. Because the mapping is merely asserted, a Verifier cannot cryptographically verify the mapping between the identifier and the controlling keypair(s) but must trust the operational processes of the trusted entity making that assertion, i.e., the CA who issued and signed the certificate. As is well known, a successful attack upon those operational processes may fool a Verifier into trusting an invalid mapping i.e., the certificate is issued to the wrong keypair(s) albeit with a verifiable signature from a valid CA. {{CEDS}}{{KDDH}}{{DNSH}}{{SFTCA}}{{DNSP}}{{BGPC}}{{BBGP}}. Noteworthy is that the signature on the certificate is not made with the controlling keypairs of the identifier but made with keypairs controlled by the issuer i.e., the CA. The fact that the certificate is signed by the CA means that the mapping itself is not verifiable but merely that the CA asserted the mapping between keypair(s) and identifier. The certificate merely provides evidence of the authenticity of the assignment of the mapping but not evidence of the veracity of the mapping. The second major flaw is that when rotating the valid signing keys, there is no cryptographically verifiable way to link the new (rotated in) controlling/signing key(s) to the prior (rotated out) controlling/signing key(s). Key Rotation is merely implicitly asserted by a trusted entity (CA) by issuing a new certificate with new controlling/signing keys. Key Rotation is necessary because over time the controlling keypair(s) of an identifier becomes weak due to exposure when used to sign messages and must be replaced. An explicit rotation mechanism first revokes the old keys and then replaces them with new keys. Even a Certificate revocation list (CRL) as per RFC5280, with an online status protocol (OCSP) registration as per RFC6960, does not provide a cryptographically verifiable connection between the old and new keys, it is merely asserted {{RFC5280}}{{RFC6960}}{{OCSPW}}. The lack of a single universal CRL or registry means that multiple potential replacements may be valid. From a cryptographic verifiability perspective, Rotation by assertion with a new certificate that either implicitly or explicitly provides revocation and replacement is essentially the same as starting over by creating a brand-new independent mapping between a given identifier and the controlling keypair(s). This start-over style of key Rotation may well be one of the main reasons that other encryption methods, such as Pretty Good Privacy’s (PGP’s) web-of-trust failed {{WOT}}. Without a universally verifiable revocation mechanism, then any Rotation (revocation and replacement) assertions either explicit or implicit are mutually independent of each other. This lack of universal cryptographic verifiability of a Rotation fosters ambiguity at any point in time as to the actual valid mapping between the identifier and its controlling keypair(s). In other words, for a given identifier, any or all assertions made by some set of CAs may be potentially valid.

The KERI protocol fixes both of these flaws using a combination of AIDs, key pre-rotation, a Verifiable data structure, the KEL, as verifiable proof of Key state, and Duplicity-evident mechanisms for evaluating and reconciling Key state by Validators {{KERI}}. Unlike certificate transparency, KERI enables the detection of Duplicity in the Key state via non-repudiable cryptographic proofs of Duplicity, not merely the detection of inconsistency in the Key state that may or may not be duplicitous {{KERI}}{{CTAOL}}.

## Self-certifying identifiers (SCIDs)

The KERI identifier system overlay leverages the properties of cryptonymous SCIDs which are based on asymmetric public-key cryptography (PKI) to provide end-verifiable secure attribution of any Message or data item without needing to trust in any intermediary {{PKI}}{{KERI}}{{UIT}}{{SCPK}}{{SFS}}{{SCPN}}{{SCURL}}. A SCID is uniquely cryptographically derived from the public key of an asymmetric keypair, (public, private), which is self-certifying in the sense that does not rely on a trusted entity. Any non-repudiable signature made with the private key may be verified by extracting the public key from either the identifier itself or incepting information uniquely associated with the cryptographic derivation process for the identifier. In a basic SCID, the mapping between an identifier and its controlling public key is self-contained in the identifier itself. A basic SCID is ephemeral i.e., it does not support Rotation of its keypairs in the event of key weakness or compromise, and therefore must be abandoned once the controlling private key becomes weakened or compromised from exposure. {{PKI}}{{KERI}}{{UIT}}{{SCPK}}{{SFS}}{{SCPN}}{{SCURL}}{{PKI}}. The class of identifiers that generalize SCIDs with enhanced properties such as persistence is called Autonomic identifiers (AIDs).

## Autonomic identifiers (AIDs)

Use of a KEL gives rise to an enhanced class of SCIDs that are persistent, not ephemeral, because SCIDs’ keys may be refreshed or updated via Rotation allowing secure control over the identifier in spite of key weakness or even compromise. This family of generalized enhanced SCIDs is AIDs. Autonomic means self-governing, self-regulating, or self-managing and is evocative of the self-certifying, self-managing properties of this class of identifier. An AID may exhibit other self-managing properties such as transferable control using key pre-rotation which enables control over such an AID to persist in spite of key weakness or compromise due to exposure. Authoritative control over the identifier persists in spite of the evolution of the Key state.

## KERI’s basic fractionally weighted threshold

This partial Rotation feature for either reserve or Custodial rotation authority is best employed with thresholds that are fractionally weighted. The exact syntax for fractionally weighted thresholds is provided in the partial pre-rotation and Custodial rotation sections and a summary is provided here. A fractionally weighted threshold consists of a list of one or more clauses where each clause is itself a list of legal rational fractions (i.e., ratios of non-negative integers expressed as fractions, where zero is not allowed in the denominator). Each entry in each clause in the fractional weight list corresponds one-to-one to a public key appearing in a key list in an Establishment event. Key lists order a key set. A weight list of clauses orders a set of rational fraction weights. Satisfaction of a fractionally weighted threshold requires satisfaction of each and every clause in the list. In other words, the clauses are logically ANDed together. Satisfaction of any clause requires that the sum of the weights in that clause that correspond to verified signatures on that event must sum to at least a weight of one. Using rational fractions and rational fraction summation avoids the problem of floating-point rounding errors and ensures exactness and universality of threshold satisfaction computations.

For example, consider the following simple single clause fractionally weighted threshold, [1/2, 1/2, 1/2]. Three weights mean there must be exactly three corresponding key pairs. Let the three keypairs in one-to-one order be denoted by the list of indexed public keys, [A0, A1, A2]. The threshold is satisfied if any two of the public keys sign because 1/2 + 1/2 = 1. This is exactly equivalent to an integer-valued ‘2 of 3’ threshold.

The order of appearance of the public key in a given key list and its associated threshold weight list must be the same.

Fractionally weighted thresholds become more interesting when the weights are not all equal or include multiple clauses. Consider the following five-element single clause fractionally weighted threshold list, [1/2, 1/2, 1/2, 1/4, 1/4] and its corresponding public key list, [A0, A1, A2, A3, A4]. Satisfaction would be met given signatures from any two or more of A0, A1, or A2 because each of these keys has a weight of 1/2 and the combination of any two or more sums to 1 or more. Alternatively, satisfaction would be met with signatures from any one or more of A0, A1, or A2 and both of A3, and A4 because any of those combinations would sum to 1 or more. Because participation of A3 and A4 is not required as long as at least two of A0, A1, and A2 are available then A3 and A4 may be treated as reserve members of the controlling set of keys. These reserve members only need to participate in the event that only one of the other three is available. The flexibility of a fractionally weighted threshold enables redundancy in the combinations of keys needed to satisfice for both day-to-day and reserve contingency use cases.

## KERI’s secure bindings

In simple form, an identifier system security overlay binds together a triad consisting of the identifier, keypairs, and Controllers, the set of entities whose members each control a private key from the given set of keypairs. The set of Controllers is bound to the set of keypairs, the set of keypairs is bound to the identifier, and the identifier is bound to the set of Controllers. This binding triad can be diagrammed as a triangle where the sides are the bindings and the vertices are the identifier, the set of Controllers, and the set of keypairs. This triad provides verifiable control authority for the identifier. When these bindings are strong, then the overlay is highly invulnerable to attack. In contrast, when these bindings are weak, then the overlay is highly vulnerable to attack. With KERI, all the bindings of the triad are strong because they are cryptographically verifiable with a minimum cryptographic strength or level of approximately 128 bits. See the Appendix on cryptographic strength for more detail.

### Triad bindings

Each Controller in the set of Controllers creates an asymmetric (public, private) keypair. The public key is derived from the private key or seed using a one-way derivation that must have a minimum cryptographic strength of approximately 128 bits {{OWF}}{{COWF}}. Depending on the crypto-suite used to derive a keypair, the private key or seed may itself have a length larger than 128 bits. A Controller may use a cryptographic strength pseudo-random number generator (CSPRNG) {{CSPRNG}} to create the private key or seed material. Because the private key material must be kept secret, typically in a secure data store, the management of those secrets may be an important consideration. One approach to minimize the size of secrets is to create private keys or seeds from a secret salt. The salt must have an entropy of approximately 128 bits. The salt may then be stretched to meet the length requirements for the crypto suite’s private key size {{Salt}}{{Stretch}}. In addition, a hierarchical deterministic derivation function may be used to further minimize storage requirements by leveraging a single salt for a set or sequence of private keys {{HDKC}}. Because each Controller is the only entity in control (custody) of the private key, and the public key is universally uniquely derived from the private key using a cryptographic strength one-way function, then the binding between each Controller and their keypair is as strong as the ability of the controller to keep that key private {{OWF}}{{COWF}}. The degree of protection is up to each Controller to determine. For example, a Controller could choose to store their private key in a safe, at the bottom of a coal mine, air-gapped from any network, with an ex-special forces team of guards. Or the Controller could choose to store it in an encrypted data store (key chain) on a secure boot mobile device with a biometric lock, or simply write it on a piece of paper and store it in a safe place. The important point is that the strength of the binding between Controller and keypair does not need to be dependent on any trusted entity.

The identifier is universally uniquely derived from the set of public keys using a one-way derivation function {{OWF}}{{COWF}}. It is therefore an AID (qualified SCID). Associated with each identifier (AID) is incepting information that must include a list of the set of Qualified public keys from the controlling keypairs. In the usual case, the identifier is a Qualified cryptographic digest of the serialization of all the incepting information for the identifier. Any change to even one bit of the incepting information changes the digest and hence changes the derived identifier. This includes any change to any one of the Qualified public keys including its qualifying derivation code. To clarify, a Qualified digest as identifier includes a derivation code as proem that indicates the cryptographic algorithm used for the digest. Thus, a different digest algorithm results in a different identifier. In this usual case, the identifier is strongly cryptographically bound to not only the public keys but also any other incepting information from which the digest was generated.

A special case may arise when the set of public keys has only one member, i.e., there is only one controlling keypair. In this case, the Controller of the identifier may choose to use only the Qualified public key as the identifier instead of a Qualified digest of the incepting information. The identifier still is strongly bound to the public key but is not so strongly bound to any other incepting information. A variant of this single keypair special case is an identifier that cannot be rotated. Another way of describing an identifier that cannot be rotated is that it is a non-transferable identifier because control over the identifier cannot be transferred to a different set of controlling keypairs. In contrast, a rotatable keypair is transferable because control may be transferred via rotation to a new set of keypairs. Essentially, when non-transferable, the identifier’s lifespan is ephemeral, not persistent, because any weakening or compromise of the controlling keypair means that the identifier must be abandoned. Nonetheless, there are important use cases for an ephemeral SCID. In all cases, the derivation code in the identifier indicates the type of identifier, whether it be a digest of the incepting information (multiple or single keypair) or a single member special case derived from only the public key (both ephemeral or persistent). Each Controller in a set of Controllers may prove its contribution to the control authority over the identifier in either an interactive or non-interactive fashion. One form of interactive proof is to satisfy a challenge of that control. The Challenger creates a unique challenge message. The Controller responds by non-repudiably signing that challenge with the private key from the keypair under its control. The Challenger can then cryptographically verify the signature using the public key from the Controller’s keypair. One form of non-interactive proof is to periodically contribute to a monotonically increasing sequence of non-repudiably signed updates of some data item. Each update includes a monotonically increasing sequence number or date-time stamp. Any Verifier can then cryptographically verify the signature using the public key from the Controller’s keypair and verify that the update was made by the Controller. In general, only members of the set of Controllers can create verifiable non-repudiable signatures using their keypairs. Consequently, the identifier is strongly bound to the set of Controllers via provable control over the keypairs.

### Tetrad bindings

At Inception, the triad of identifier, keypairs, and Controllers are strongly bound together. But in order for those bindings to persist after a key Rotation, another mechanism is required. That mechanism is the KEL, a Verifiable data structure {{KERI}}{{VDS}}. The KEL is not necessary for identifiers that are non-transferable and do not need to persist control via key Rotation in spite of key weakness or compromise. To reiterate, transferable (persistent) identifiers each need a KEL, non-transferable (ephemeral) identifiers do not. For persistent (transferable) identifiers, this additional mechanism may be bound to the triad to form a tetrad consisting of the KEL, the identifier, the set of keypairs, and the set of Controllers. The first entry in the KEL is called the Inception event which is a serialization of the incepting information associated with the identifier mentioned previously. The Inception event must include the list of controlling public keys, must also include a signature threshold and must be signed by a set of private keys from the controlling keypairs that satisfy that threshold. Additionally, for transferability (persistence across rotation) the Inception event must also include a list of digests of the set of pre-rotated public keys and a pre-rotated signature threshold that will become the controlling (signing) set of keypairs and threshold after a Rotation. A non-transferable identifier may have a trivial KEL that only includes an Inception event but with a null set (empty list) of pre-rotated public keys.

A Rotation is performed by appending a Rotation event to the KEL. A Rotation event must include a list of the set of pre-rotated public keys (not their digests) thereby exposing them and must be signed by a set of private keys from these newly exposed newly controlling but pre-rotated keypairs that satisfy the pre-rotated threshold. The Rotation event also must include a list of the digests of a new set of pre-rotated keys as well as the signature threshold for the set of pre-rotated keypairs. At any point in time, the transferability of an identifier can be removed via a Rotation event that rotates to a null set (empty list) of pre-rotated public keys.

Each event in a KEL must include an integer sequence number that is one greater than the previous event. Each event after the Inception event also must include a cryptographic digest of the previous event. This digest means that a given event is cryptographically bound to the previous event in the sequence. The list of digests or pre-rotated keys in the Inception event cryptographically binds the Inception event to a subsequent Rotation event, essentially making a forward commitment that forward chains together the events. The only valid Rotation event that may follow the Inception event must include the pre-rotated keys. But only the Controller who created those keys and created the digests may verifiably expose them. Each Rotation event in turn makes a forward commitment (chain) to the following Rotation event via its list of pre-rotated key digests. This makes the KEL a doubly (backward and forward) hash (digest) chained non-repudiably signed append-only Verifiable data structure. Because the signatures on each event are non-repudiable, the existence of an alternate but verifiable KEL for an identifier is provable evidence of duplicity. In KERI, there may be at most one valid KEL for any identifier or none at all. Any Validator of a KEL may enforce this one valid KEL rule before relying on the KEL as proof of the current key state for the identifier which protects the Validator. Any unreconcilable evidence of Duplicity means the Validator does not trust (rely on) any KEL to provide the key state for the identifier. Rules for handling reconciliable Duplicity will be discussed later. From a Validator’s perspective, either there is one-and-only-one valid KEL or none at all which also protects the Validator by removing any potential ambiguity about Key state. The combination of a Verifiable KEL made from non-repudiably signed backward and forward hash chained events together with the only-one-valid KEL rule strongly binds the identifier to its current Key state as given by that one valid KEL or not at all. This in turn binds the identifier to the Controllers of the current keypairs given by the KEL thus completing the tetrad. At Inception, the KEL may be even more strongly bound to its tetrad by deriving the identifier from a digest of the Inception event so that even one change in not only the original controlling keys pairs but also the pre-rotated keypairs or any other incepting information included in the Inception event will result in a different identifier.

The essense of the KERI protocol is a strongly bound tetrad of identifier, keypairs, Controllers, and the KEL that forms the basis of its identifier system security overlay. The KERI protocol introduces the concept of Duplicity evident programming via Duplicity evident Verifiable data structures. The full detailed exposition of the protocol is provided in the following sections.

# KERI data structures and labels

## KERI data structures

A KERI data structure such as a Key event message body may be abstractly modelled as a nested key: value mapping. To avoid confusion with the cryptographic use of the term key, the term field is used instead to refer to a mapping pair and the terms field label and field value for each member of a pair. These pairs can be represented by two tuples e.g., (label, value). When necessary, this terminology is qualified by using the term field map to reference such a mapping. Field maps may be nested where a given field value is itself a reference to another field map. This nested set of fields are referred to as a nested field map or simply a nested map for short.

A field may be represented by a framing code or block delimited serialization. In a block delimited serialization, such as JSON, each field map is represented by an object block with block delimiters such as ‘{}’ {{RFC8259}}{{JSOND}}{{RFC4627}}. Given this equivalence, the term block or nested block can be used as synonymous with field map or nested field map. In many programming languages, a field map is implemented as a dictionary or hash table in order to enable performant asynchronous lookup of a field value from its field label. Reproducible serialization of field maps requires a canonical ordering of those fields. One such canonical ordering is called insertion or field creation order. A list of (field, value) pairs provides an ordered representation of any field map.

Most programming languages now support ordered dictionaries or hash tables that provide reproducible iteration over a list of ordered field (label, value) pairs where the ordering is the insertion or field creation order. This enables reproducible roundtrip serialization/deserialization of field maps. Serialized KERI data structures depend on insertion-ordered field maps for their canonical serialization/deserialization. KERI data structures support multiple serialization types, namely JSON, CBOR, MGPK, and CESR but for the sake of simplicity, we will only use JSON herein for examples {{RFC8259}}{{JSOND}}{{CBORC}}{{RFC8949}}{{MGPK}}{{CESR-ID}}. The basic set of normative field labels in KERI field maps is defined in the table in the following section.

## KERI field labels for data structures

KERI field labels for data structures {#tbl:field-lables}

| **Label** | **Title** | **Description** |
| --- | --- | --- |
| **v** | Version string |  |
| **i** | Identifier prefix (AID) |  |
| **s** | Sequence number |  |
| **t** | Message type |  |
| **te** | Last received event Message type in a Key state notice |  |
| **d** | Event SAID | SAID of the enclosing block or map |
| **p** | Prior event SAID |  |
| **kt** | Key signing threshold |  |
| **k** | List of signing keys | Ordered key set |
| **nt** | Next Key signing threshold | Next threshold for the next Establishement event |
| **n** | List of next Key digests | Ordered key digest set |
| **bt** | Backer threshold |  |
| **b** | List of Backers | Ordered Backer set of AIDs |
| **br** | List of Backers to remove | Ordered Backer set of AIDs |
| **ba** | List of Backers to add | Ordered Backer set of AIDs |
| **c** | List of Configuration Traits, Modes |  |
| **a** | List of Anchors | Seals; data attributes or data anchors depending on the message type |
| **di** | Delegator identifier prefix | AID |
| **rd** | Merkel Tree Root Digest | SAID |
| **ee** | Last Establishment event map |  |

A field label may have different values in different contexts but must not have a different field value or type. This requirement makes it easier to implement in strongly typed languages with rigid data structures. Notwithstanding the former, some field value types may be a union of elemental value types. Because the order of appearance of fields is enforced in all KERI data structures, whenever a field appears in a given Message or block in a Message, the Message in which a label appears must provide the necessary context to fully determine the meaning of that field and hence the field value type and associated semantics.

### Compact KERI field labels

The primary field labels are compact in that they use only one or two characters. KERI is meant to support resource-constrained applications such as supply chain or IoT applications. Compact labels better support resource-constrained applications in general. With compact labels, the over-the-wire verifiable signed serialization consumes a minimum amount of bandwidth. Nevertheless, without loss of generality, a one-to-one normative semantic overlay using more verbose expressive field labels may be applied to the normative compact labels after verification of the over-the-wire serialization. This approach better supports bandwidth and storage constraints on transmission while not precluding any later semantic post-processing. This is a well-known design pattern for resource-constrained applications.

### Special label ordering requirements

#### Version string field

The version string, ‘v’ field must be the first field in any top-level KERI field map in which it appears. Typically, the version string, ‘v’, field appears as the first top-level field in a KERI Message body. This enables a RegEx stream parser to consistently find the version string in any of the supported serialization formats for KERI Messages. The ‘v’ field provides a regular expression target for determining the serialization format and size (character count) of a serialized KERI Message body. A stream parser may use the version string to extract and deserialize (deterministically) any serialized KERI Message body in a stream of serialized KERI Messages. Each KERI Message in a stream may use a different serialization type.

The format of the version string is ‘KERIvvSSSShhhhhh\_’. The first four characters ‘KERI’ indicate the enclosing field map serialization. The next two characters, vv provide the lowercase hexadecimal notation for the major and minor version numbers of the version of the KERI specification used for the serialization. The first ‘v’ provides the major version number and the second ’v’provides the minor version number. For example, ‘01’ indicates major version 0 and minor version 1 or in dotted-decimal notation ‘0.1’. Likewise, ‘1c’ indicates major version 1 and minor version decimal 12 or in dotted-decimal notation ‘1.1’.

The next four characters ‘SSSS’ indicate the serialization type in uppercase. The four supported serialization types are ‘JSON’, ‘CBOR’, ‘MGPK’, and ‘CESR’ for the JSON, CBOR, MessagePack, and Composable Event Streaming Representation (CESR) serialization standards respectively {{JSOND}}{{RFC4627}}{{CBORC}}{{RFC8949}}{{MGPK}}{{CESR-ID}}. The next six characters provide in lowercase hexadecimal notation the total number of characters in the serialization of the KERI Message body. The maximum length of a given KERI Message body is thereby constrained to be \*224 = 16,777,216 characters in length. The final character ’-‘ is the version string terminator. This enables later versions of Authentic Chained Data Containers (ACDCs) to change the total version string size and thereby enable versioned changes to the composition of the fields in the version string while preserving deterministic regular expression extractability of the version string.

Although a given KERI serialization type may use field map delimiters or framing code characters that appear before (i.e., prefix) the version string field in a serialization, the set of possible prefixes is sufficiently constrained by the allowed serialization protocols to guarantee that a regular expression can determine unambiguously the start of any ordered field map serialization that includes the version string as the first field value. Given the version string, a parser then may determine the end of the serialization so that it can extract the full serialization (KERI Message body) from the stream without first deserializing it or parsing it field-by-field. This enables performant stream parsing and off-loading of KERI Message streams that include any or all of the supported serialization types interleaved in a single stream.

#### SAID fields

Some fields in KERI data structures may have a SAID (self-referential content addressable), as a field value. In this context, ‘d’ is short for digest, which is short for SAID. A SAID is a special type of cryptographic digest of its encapsulating field map (block). The encapsulating block of a SAID is called a SAD (Self-Addressed Data). Using a SAID as a field value enables a more compact but secure representation of the associated block (SAD) from which the SAID is derived. Any nested field map that includes a SAID field (i.e., is, therefore, a SAD) may be compacted into its SAID. The uncompacted blocks for each associated SAID may be attached or cached to optimize bandwidth and availability without decreasing security.

Each SAID provides a stable universal cryptographically verifiable and agile reference to its encapsulating block (serialized field map).

A cryptographic commitment (such as a digital signature or cryptographic digest) on a given digest with sufficient cryptographic strength including collision resistance {{HCR}}{{QCHC}} is equivalent to a commitment to the block from which the given digest was derived. Specifically, a digital signature on a SAID makes a verifiable cryptographic non-repudiable commitment that is equivalent to a commitment on the full serialization of the associated block from which the SAID was derived. This enables reasoning about KERI data structures in whole or in part via their SAIDS in a fully interoperable, verifiable, compact, and secure manner. This also supports the well-known bow-tie model of Ricardian Contracts (RC). This includes reasoning about the whole KERI data structure given by its top-level SAID, ‘d’ field as well as reasoning about any nested or attached data structures using their SAIDS.

#### AID fields

Some fields, such as the ‘i’ and ‘di’ fields, each must have an AID as a field value. An AID is a fully qualified SCID as described above {{KERI}}{{KERI-ID}}. An AID must be self-certifying.

In this context, ‘i’ is short for ‘ai’, which is short for AID. The AID given by the ‘i’ field may also be thought of as a securely attributable identifier, authoritative identifier, authenticatable identifier, authorizing identifier, or authoring identifier. Another way of thinking about an ‘i’ field is that it is the identifier of the authoritative entity to which a statement may be securely attributed, thereby making the statement verifiably authentic via a non-repudiable signature made by that authoritative entity as the Controller of the private key(s).

#### Namespaced AIDs

Because KERI is agnostic about the namespace for any particular AID, different namespace standards may be used to express KERI AIDs within AID fields in an ACDC. The examples below use the W3C DID namespace specification with the did:keri method {{DIDK-ID}}. But the examples would have the same validity from a KERI perspective if some other supported namespace was used or no namespace was used at all. The latter case consists of a bare KERI AID (identifier prefix).

ToDo Explain agnosticism vis a vis namespaces

Because AIDs may be namespaced, the essential component of an AID is the cryptographically derived Controller identifier prefix. An AID MUST be the Controller identifier prefix. part of a W3C Decentralized IDentifier (DID) {{W3C\_DID}} or other namespace convention.

Version string namespaces the AIDs as KERI so do not need any namespacing on a per identifier basis.

#### Version string field

The Version string field is obtained from the ACDC.

## Seals

### Digest seal

{  
 "d": "Eabcde..."  
}

### Merkle Tree root digest seal

{  
 "rd": "Eabcde8JZAoTNZH3ULvaU6JR2nmwyYAfSVPzhzS6b5CM"  
}

### Backer seal

{  
 "bi": "BACDEFG8JZAoTNZH3ULvaU6JR2nmwyYAfSVPzhzS6b5CM",  
 "d" : "EFGKDDA8JZAoTNZH3ULvaU6JR2nmwyYAfSVPzhzS6b5CM"  
}

### Event seal

{   
 "i": "Ebietyi8JZAoTNZH3ULvaU6JR2nmwyYAfSVPzhzS6b5CM.",  
 "s": "3",  
 "d": "Eabcde8JZAoTNZH3ULvaU6JR2nmwyYAfSVPzhzS6b5CM"  
}

### Last Establishment event seal

{  
 "i": "BACDEFG8JZAoTNZH3ULvaU6JR2nmwyYAfSVPzhzS6b5CM",  
}

# KERI key management

## Key pair labelling convention

In order to make Key event expressions both clearer and more concise, a keypair labelling convention is used. When an AID’s Key state is dynamic, i.e., the set of controlling keypairs is transferable, then the keypair labels are indexed in order to represent the successive sets of keypairs that constitute the Key state at any position in the KEL. Specifically, indexes are used on the labels for AIDs that are transferable to indicate which set of keypairs is associated with the AID at any given point in its Key state or KEL. In contrast, when the Key state is static, i.e., the set of controlling keypairs is non-transferable, then no indexes are needed because the Key state never changes.

Recall that, a keypair is a two tuple, (public, private), of the respective public and private keys in the keypair. For a given AID, the labelling convention uses an uppercase letter label to represent that AID. When the Key state is dynamic, a superscripted index on that letter is used to indicate which keypair is used at a given Key state. Alternatively, the index may be omitted when the context defines which keypair and which Key state, such as, for example, the latest or current Key state.

In general, without loss of specificity, an uppercase letter is used to label to represent both an AID and when indexed to represent its keypair or keypairs that are authoritative at a given Key state for that AID. In addition, when expressed in tuple form the uppercase letter also represents the public key and the lowercase letter represents the private key for a given keypair. For example, let ‘A’ denote and AID, then let ‘A’ also denote a keypair which may be also expressed in tuple form as (A, a). Therefore, when referring to the keypair itself as a pair and not the individual keys within the pair, either the uppercase label, ‘A’, or the tuple, (A, a), may be used to refer to the keypair itself. To refer to the individual keys within a tuple, the uppercase letter, ‘A’, refers to the public key, and the lowercase letter, ‘a’, refers to the private key.

The sequence of keypairs that are authoritative (i.e., establish control authority) for an AID should be indexed by the zero-based integer-valued, strictly increasing by one, variable ‘i’. Furthermore, as described above, an Establishment event may change the Key state. The sequence of Establishment events should be indexed by the zero-based integer-valued, strictly increasing by one, variable ‘j’. When the set of controlling keypairs that are authoritative for a given Key state includes only one member, then i = j for every keypair, and only one index is needed. But when the set of keypairs used at any time for a given Key state includes more than one member, then i ! = j for every keypair, and both indices are needed.

### Case in which only one index is needed

Because i = j, the indexed keypair for AID, A, is denoted by Ai or in tuple form by (Ai, ai) where the keypair that is indexed uses the ith keypair from the sequence of all keypairs.  
Example of the keypair sequence – one index: Expressed as the list, [A0, A1, A2, …] Where: A0 is the zero element in this sequence; (A0, a0) is the tuple form.

### Case in which both indexes are needed

Because i!= j, the indexed keypair for AID, ‘A’, is denoted by Ai or in tuple form by (Ai, ai) where the keypair that is indexed is authoritative or potentially authoritative for ith keypair from the sequence of all keypairs that is authoritative in the jth key state.

Example of the keypair sequence – using three keypairs to control authority:

Expressed as the list, [A0, A1, A2, A2,0, A3,1, A4,1, A5,1].

Where: the first two key states will consume the first six keypairs of the list.

Furthermore, with pre-rotation, each public key from the set of pre-rotated keypairs may be hidden as a Qualified cryptographic digest of that public key. The digest of the public key labelled A is represented using the functional notation H(A) for hash (digest). When singly indexed, the digest of Ai is denoted by H(Ai) and when doubly indexed the digest of Ai,j is denoted by H(Ai,j</sup]. A pre-rotated keypair is potentially authoritative for the next or subsequent Establishment event after the Establishment event when the digest of the pre-rotated keypair first appears. Therefore, its jth index value is one greater than the jth index value of the Establishment event in which its digest first appears. As explained in more detail below, for partial Rotation of a pre-rotated set, a pre-rotated keypair from a set of two or more pre-rotated keypairs is only potentially authoritative so that its actual authoritative jth index may change when it is actually rotated in, if ever.

Finally, each Key event in a KEL must have a zero-based integer-valued, strictly increasing by one, sequence number. Abstractly, the variable ‘k’ can be used as an index on any keypair label to denote the sequence number of an event for which that keypair is authoritative. Usually, this appears as a subscript. Thus any given keypair label could have three indices, namely, i,j,k that appear as follows, Ai,jk where ‘i’ denotes the ith keypair from the sequence of all keypairs, ‘j’ denotes the jth Establishment event in which the keypair is authoritative, and ‘k’ represents the kth Key event in which the keypair is authoritative. When a KEL has only Establishment event, then j = k.

## Key rotation/pre-rotation

An important innovation of KERI is that it solves the key Rotation problem of PKI (including that of simple SCIDs) via a novel but elegant mechanism called key pre-rotation {{DAD}}{{KERI}}. This pre-rotation mechanism enables an entity to maintain persistently or to regain control over an identifier despite exposure-related weakening over time or even compromise of the current set of controlling (signing) keypairs. With key pre-rotation, control over the identifier can be re-established by rotating to a one-time use set of unexposed but pre-committed Rotation keypairs that then become the current signing keypairs. Each Rotation in turn cryptographically commits to a new set of Rotation keys but without exposing them. Because the pre-rotated keypairs need never be exposed prior to their one-time use, their attack surface may be optimally minimized. The current Key state, the state of the controlling keys for an identifier at any time, is maintained via the KEL, an append-only Verifiable data structure. Cryptographic verifiability of the Key state over time is essential to remove this ambiguity over the mapping between the identifier (domain name) and the controlling keypair(s). Without this verifiability, the detection of potential ambiguity requires yet another bolt-on security overlay such as the certificate transparency system {{CTE}}{{CTAOL}}{{RFC6962}}{{RT}}{{VDS}}{{ESMT}}.

### Pre-rotation

Each Establishment event involves two sets of keys that each play a role that together establishes complete control authority over the AID associated at the location of that event in the KEL. To clarify, control authority is split between keypairs that hold signing authority and keypairs that hold Rotation authority. A Rotation revokes and replaces the keypairs that hold signing authority as well as replacing the keypairs that hold Rotation authority. The two set sets of keys are labelled current and next. Each Establishment event designates both sets of keypairs. The first (current) set consists of the authoritative signing keypairs bound to the AID at the location in the KEL where the Establishment event occurs. The second (next) set consists of the pre-rotated authoritative Rotation keypairs that will be actualized in the next (ensuing) Establishment event. Each public key in the set of next (ensuing) pre-rotated public keys is hidden in or blinded by a digest of that key. When the Establishment event is the Inception event then the current set also is the initial set. The pre-rotated next set of Rotation keypairs are one-time use only Rotation keypairs,but may be repurposed as signing keypairs after their one time use to rotate.

In addition, each Establishment event designates two threshold expressions, one for each set of keypairs (current and next). The current threshold determines the needed satisficing subset of signatures from the associated current set of keypairs for signing authority to be considered valid. The next threshold determines the needed satisficing subset of signatures from the associated next set of hidden keypairs for Rotation authority to be considered valid. The simplest type of threshold expression for either threshold is an integer that is no greater than nor no less than the number of members in the set. An integer threshold acts as an ‘M of N’ threshold where ‘M’ is the threshold and ‘N’ is the total number of keypairs represented by the public keys in the key list. If any set of ‘M’ of the ‘N’ private keys belonging to the public keys in the key list verifiably signs the event then the threshold is satisfied for the control authority role (signing or Rotation) associated with the given key list and threshold.

To clarify, each Establishment event must include a list (ordered) of the Qualified public keys from each of the current (initial) set of keypairs, a threshold for the current set, a list (ordered) of the Qualified cryptographic digests of the qualified public keys from the next set of keypairs, and a threshold for the next set. Each event must also include the AID itself as either a Qualified public key or a Qualified digest of the Inception Establishment event.

Each non-Establishment event must be signed by a threshold-satisficing subset of private keys from the current set of keypairs from the most recent Establishment event. The following sections detail the requirements for a valid set of signatures for each type of Establishment event.

#### Inception event pre-rotation

The creator of the Inception event MUST create two sets of keypairs, the current (initial) set, and the next set. The private keys from the current set are kept as secrets. The public keys from the current set are exposed via inclusion in the Inception event. Both the public and private keys from the next set are kept as secrets and only the cryptographic digests of the public keys from the next set are exposed via inclusion in the event. The public keys from the next set are only exposed in a subsequent Establishment event, if any. Both thresholds are exposed via inclusion in the event.

Upon emittance of the Inception event, the current (initial) set of keypairs becomes the current set of verifiable authoritative signing keypairs for the identifier. Emittance of the Inception event also issues the identifier. Moreover, to be verifiably authoritative, the Inception event must be signed by a threshold satisficing subset of the current (initial) set of private keys. The Inception event may be verified against the attached signatures using the included current (initial) list of public keys. When self-addressing, a digest of the serialization of the Inception event provides the AID itself as derived by the SAID protocol {{SAID-ID}}.

There must be only one Inception Establishment event. All subsequent Establishment events must be Rotation events.

#### Rotation using pre-rotation

Unlike Inception, the creator of a Rotation event needs to create only one set of keypairs, the newly next set. Both the public and private keys from the newly next set are kept as secrets and only the cryptographic digests of the public keys from the newly next set are exposed via inclusion in the event. The list of newly current public keys MUST include the old next threshold satisficing subset of old next public keys from the most recent prior establishment event. For short, the next threshold is denoted from the most recent prior Establishment event as the prior next threshold, and the list of unblinded public keys taken from the blinded key digest list from the most recent prior Establishment event as the prior next key list. The subset of old prior next keys that are included in the newly current set of public keys must be unhidden or unblinded because they appear as the public keys themselves and no longer appear as digests of the public keys. Both thresholds are exposed via inclusion in the event.

Upon emittance of the Rotation event, the newly current keypairs become the current set of verifiable authoritative signing keypairs for the identifier. The old current set of keypairs from the previous Establishment event has been revoked and replaced by the newly current set. Moreover, to be verifiably authoritative, the Rotation event must be signed by a dual threshold satisficing subset of the newly current set of private keys. To elaborate, the set of signatures on a Rotation event must satisfy two thresholds, the newly current threshold and the old prior next threshold from the most recent prior Establishment event. Therefore, the newly current set of public keys must include a satisfiable subset with respect to the old prior next threshold of public keys from the old prior next key list. The included newly current list of public keys enables verification of the Rotation event against the attached signatures.

The act of inclusion in each Establishment event of the digests of the new next set of public keys performs a pre-rotation operation on that set by making a verifiable forward blinded commitment to that set. Consequently, no other set may be used to satisfy the threshold for the next Rotation operation. Because the next set of pre-rotated keys is blinded (i.e., has not been exposed, used to sign or even published), an attacker cannot forge and sign a verifiable Rotation operation without first unblinding the pre-rotated keys. Therefore, given sufficient cryptographic strength of the digests, the only attack surface available to the adversary is a side-channel attack on the private key store itself and not on signing infrastructure. But the creator of the pre-rotated private keys is free to make that key store as arbitrarily secure as needed because the pre-rotated keys are not used for signing until the next Rotation. As long as the Controller keeps secret the pre-rotated public keys themselves, an attacker must attack the key storage infrastructure because side-channel attacks on signing infrastructure are obviated.

As explained later, for a Validator, the first seen rule applies, that is, the first seen version of an event is the authoritative one for that Validator. The first seen wins. The first published version becomes the first seen version. Upon Rotation, the old prior next keys are exposed but only after a new next set has been created and stored. Thus, the creator is always able to stay one step ahead of an attacker. By the time a new Rotation event is published, it is too late for an attacker to create a verifiable Rotation event to supplant it because the original version already has been published and may be the version first seen by the Validator. The window for an attacker is the network latency for the first published event to be first seen by the network of Validators. Any later, key compromise is too late.

In essence, each key set follows a Rotation lifecycle in which it changes its role with each Rotation event. A pre-rotated keypair set starts as the member of the next key set holding one-time Rotation control authority. On the ensuing Rotation, that keypair becomes part of the current key set holding signing control authority. Finally, on the following Rotation that keypair is discarded. The lifecycle for the initial key set in an Inception event is slightly different. The initial key set starts as the current set holding signing authority and is discarded on the ensuing Rotation event, if any.

#### Partial pre-rotation

Pre-rotation is a Rotation operation on a set of pre-rotated keys that may keep some keys in reserve (i.e., unexposed) while exposing others as needed.

As described above, a valid Rotation operation requires the satisfaction of two different thresholds. These are the current threshold of the given Rotation (Establishment) event with respect to its associated current public key list and the next threshold from the given Rotation event’s most recent prior Establishment event with respect to its associated blinded next key digest list. For short, the next threshold from the most recent prior Establishment event is denoted as the prior next threshold, and the list of unblinded public keys taken from the blinded key digest list from the most recent prior Establishment event as the prior next key list. Explication of the elements of the prior next key list requires exposing or unblinding the underlying public keys committed to by their corresponding digests that appear in the next key digest list of the most recent prior Establishment event. The unexposed (blinded) public keys may be held in reserve.

More precisely, any Rotation event’s current public key list must include a satisfiable subset of the prior next key list with respect to the prior next threshold. In addition, any Rotation event’s current public key list must include a satisfiable set of public keys with respect to its current threshold. In other words, the current public key list must be satisfiable with respect to both the current and prior next thresholds.

To reiterate, in order to make verifiable the maintenance of the integrity of the forward commitment to the pre-rotated list of keys made by the prior next event, i.e., provide verifiable Rotation control authority, the current key list must include a satisfiable subset of exposed (unblinded) pre-rotated next keys from the most recent prior Establishment event where satisfiable is with respect to the prior next threshold. Moreover, in order to establish verifiable signing control authority, the current key list must also include a satisfiable subset of public keys where satisfiable is with respect to the current threshold.

These two conditions are trivially satisfied whenever the current and prior next key lists and thresholds are equivalent. When both the current and the prior next key lists and thresholds are identical then the validation can be simplified by comparing the two lists and thresholds to confirm that they are identical and then confirming that the signatures satisfy the one threshold with respect to the one key list. When not identical, the Validator must perform the appropriate set math to confirm compliance.

Recall, that the order of appearance of the public key in a given key list and its associated threshold weight list must be the same. The order of appearance, however, of any public keys that appear in both the current and prior next key lists may be different between the two key lists and hence the two associated threshold weight lists. A Validator must therefore confirm that the set of keys in the current key list truly includes a satisfiable subset of the prior next key list and that the current key list is satisfiable with respect to both the current and prior next thresholds. Actual satisfaction means that the set of attached signatures must satisfy both the current and prior next thresholds as applied to their respective key lists.

Suppose that the current public key list does not include a proper subset of the prior next key list. This means that no keys were held in reserve. This also means that the current key list is either identical to the prior next key list or is a superset of the prior next key list. Nonetheless, such a Rotation may change the current key list and or threshold with respect to the prior next key list and/or threshold as long as it meets the satisfiability constraints defined above.

If the current key list includes the full set of keys from the prior next key list, then a full Rotation has occurred, not a partial Rotation because no keys were held in reserve or omitted. A full Rotation may add new keys to the current key list and/or change the current threshold with respect to the prior next key list and threshold.

### Reserve rotation

The pre-rotation mechanism supports partial pre-rotation or more exactly partial rotation of pre-rotated keypairs. One important use case for partial rotation is to enable pre-rotated keypairs designated in one Establishment event to be held in reserve and not exposed at the next (immediately subsequent) Establishment event. This reserve feature enables keypairs held by controllers as members of a set of pre-rotated keypairs to be used for the purpose of fault tolerance in the case of non-availability by other Controllers while at the same time minimizing the burden of participation by the reserve members. In other words, a reserved pre-rotated keypair contributes to the potential availability and fault tolerance of control authority over the AID without necessarily requiring the participation of the reserve keypair in a Rotation until and unless it is needed to provide continuity of control authority in the event of a fault (non-availability of a non-reserved member). This reserve feature enables different classes of key Controllers to contribute to the control authority over an AID. This enables provisional key control authority. For example, a key custodial service or key escrow service could hold a keypair in reserve to be used only upon satisfaction of the terms of the escrow agreement. This could be used to provide continuity of service in the case of some failure event. Provisional control authority may be used to prevent types of common-mode failures without burdening the provisional participants in the normal non-failure use cases.

### Custodial rotation

Partial pre-rotation supports another important use case that of custodial key Rotation. Because control authority is split between two key sets, the first for signing authority and the second (pre-rotated) for Rotation authority, the associated thresholds and key list can be structured in such a way that a designated custodial agent can hold signing authority while the original Controller can hold exclusive Rotation authority. The holder of the Rotation authority can then at any time without the cooperation of the custodial agent, if need be, revoke the agent’s signing authority and assign it so some other agent or return that authority to itself.

# Secure signing in KERI using CESR encoding

## Security considerations

TODO security # This is an annex {#sec:annexA .normative}

With some text

# This is another annex

With some more text