SIDROPS
Internet-Draft
Intended status: Standards Track
Expires: 19 March 2026

J. Snijders
T. Bruijnzeels
RIPE NCC
T. Harrison
APNIC
W. Ohgai
JPNIC

15 September 2025

The Erik Synchronization Protocol for use with the Resource Public Key Infrastructure (RPKI)
draft-spaghetti-sidrops-rpki-erik-protocol-02

Abstract

This document specifies the Erik Synchronization Protocol for use with the Resource Public Key Infrastructure (RPKI). Erik Synchronization can be characterized as a data replication system using Merkle trees, a content-addressable naming scheme, concurrency control using monotonically increasing sequence numbers, and HTTP transport. Relying Parties can combine information retrieved via Erik Synchronization with other RPKI transport protocols. The protocol’s design is intended to be efficient, fast, and easy to implement.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 19 March 2026.

Copyright Notice

Table of Contents

1. Introduction

This document specifies the Erik Synchronization Protocol for use with the Resource Public Key Infrastructure (RPKI) [RFC6480]. Erik Synchronization can be characterized as a data replication system using Merkle Trees [M1987], a content-addressable naming scheme [RFC6920], concurrency control using monotonically increasing sequence numbers [RFC0677], and HTTP transport [RFC9110]. Relying Parties can combine information retrieved via Erik Synchronization with other RPKI transport protocols ([RFC5781] and [RFC8182]). The protocol's design is intended to be efficient, fast, and easy to implement [RFC1925].

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Related Work

The reader is assumed to be familiar with the terms and concepts described in "Maintenance of duplicate databases" [RFC0677], "An Infrastructure to Support Secure Internet Routing" [RFC6480], "The RPKI Repository Delta Protocol (RRDP)" [RFC8182], "Manifests for the Resource Public Key Infrastructure (RPKI)" [RFC9286], and "A Digital Signature Based on a Conventional Encryption Function" [M1987].

1.3. Glossary

This section describes the terminology and abbreviations used in this document. Though the definitions might not be clear on a first read, later on the terms will be introduce with more detail.

Erik relay An intermediate between CA publication repositories and Relying Parties.

FQDN The fully qualified domain name of a RPKI repository instance referenced in an end-entity certificate's Subject Information Access (SIA) extension's id-ad-signedObject accessDescription.

Hash A message digest calculated for an object using the SHA-256

algorithm.

Tree head Also known as "root hash", this is the hash of the current ErikIndex associated with a given FQDN.

ErikIndex An ordered listing of Partition identifiers and associated ErikPartition objects' hashes.

ErikPartition An ordered listing of the manifest objects' hashes, manifestNumber values, thisUpdate values, and their certificates' SIA extension values.

Snapshot A compressed tarball archive containing all RPKI objects for a given FQDN.

2. Informal Overview

In this synchronization protocol Merkle trees are used to determine whether differences exist between client and relay. Merkle trees are hierarchical data structures: the hash value of each node is computed recursively by hashing the concatenated hash values of the node's children. The tree head hash represents the entire dataset. If the tree head hash is not the same between two replicas, the relay provides the client with hashes of smaller and smaller portions of the to-be-replicated dataset until the exact list of out-of-sync or missing objects is identified. Sequence numbers are then used to determine whether these differences are relevant enough for the client to fetch. All data is fetched using addresses derived from hashes (except for the hash tree heads). This approach reduces unnecessary data transfer between caches which contain mostly similar data.

The client starts by querying an Erik relay for its current tree head for a given FQDN. If the tree head is different compared to the previous run (or compared to the tree head calculated from the locally cached objects), the client can fetch the ErikIndex object using the tree head hash. With the ErikIndex in hand, the client can determine which ErikParitions changed or are missing and fetch accordingly. The client then can compare the _manifestNumber_ sequence number for each manifest listed in the ErikPartition, and proceed to fetch (purportedly) newer versions of manifests of interest. Whenever a relay offers a manifest with a lower sequence number the client can ignore it. With the information contained within manifests, clients can fetch addressed by content and store by name. The client now has sufficient information to proceed to fetch any missing Certificates, Signed objects, and CRLs.

3. Erik Synchronization Data Structure Definitions

In this synchronization protocol the _signal layer_ makes use of DER-encoded messages [X.690].

Design note: DER encoding was selected for its canonical properities and because RPKI cache implementations already support ASN.1 encoding.

```
RpkiErikSynchronization-2025
  { iso(1) member-body(2) us(840) rsadsi(113549)
    pkcs(1) pkcs9(9) smime(16) mod(0)
    id-mod-rpkiErikSynchronization-2025(TBD) }

DEFINITIONS EXPLICIT TAGS ::=
BEGIN

IMPORTS
  CONTENT-TYPE, Digest, DigestAlgorithmIdentifier
  FROM CryptographicMessageSyntax-2010 -- in [RFC6268]
  { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
    pkcs-9(9) smime(16) modules(0) id-mod-cms-2009(58) }

  AccessDescription, KeyIdentifier
  FROM PKIX1Implicit88 -- in [RFC5280]
  { iso(1) identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) pkix(7) id-mod(0) id-pkix1-implicit(19) }
;

ct-rpkiErikIndex CONTENT-TYPE ::=
  { TYPE ErikIndex IDENTIFIED BY id-ct-rpkiErikIndex }

id-ct-rpkiErikIndex OBJECT IDENTIFIER ::=
  { iso(1) identified-organization(3) dod(6) internet(1) private(4)
    enterprise(1) snijders(41948) erikindex(826) }

ErikIndex ::= SEQUENCE {
  version [0]      INTEGER DEFAULT 0,
  indexScope       IA5String,
  indexTime        GeneralizedTime,
  hashAlg          DigestAlgorithmIdentifier,
  partitionList    SEQUENCE SIZE (1..ub-Partitions) OF PartitionRef }

ub-Partitions INTEGER ::= 1024

PartitionRef ::= SEQUENCE {
  identifier       INTEGER (1..ub-Partitions),
  hash             Digest,
```

```
    size             INTEGER (100..MAX) }

  ct-rpkiErikPartition CONTENT-TYPE ::=
    { TYPE ErikPartition IDENTIFIED BY id-ct-rpkiErikPartition }

  id-ct-rpkiErikPartition OBJECT IDENTIFIER ::=
    { iso(1) identified-organization(3) dod(6) internet(1) private(4)
      enterprise(1) snijders(41948) erikpartition(827) }

  ErikPartition ::= SEQUENCE {
    version [0]      INTEGER DEFAULT 0,
    partitionTime    GeneralizedTime,
    hashAlg          DigestAlgorithmIdentifier,
    manifestList     SEQUENCE SIZE (1..MAX) OF ManifestRef }

  ManifestRef ::= SEQUENCE {
    hash             Digest,
    size             INTEGER (1000..MAX),
    aki              KeyIdentifier,
    manifestNumber   INTEGER (0..MAX),
    thisUpdate       GeneralizedTime,
    location         SEQUENCE SIZE (1..MAX) OF AccessDescription }
  END
```

3.1. ErikIndex

An ErikIndex represents all current manifest objects available under a given FQDN and thus the complete state of the repository as it is known to the relay.

3.1.1. The version field

The version number of the ErikIndex object MUST be 0.

3.1.2. The indexScope field

The indexScope field contains the fully qualified domain name of the Signed Object location of the manifests referenced through this particular ErikIndex. The FQDN MUST be in the "preferred name syntax", as specified by Section 3.5 of [RFC1034] and modified by Section 2.1 of [RFC1123].

3.1.3. The indexTime field

The indexTime is the most recent partitionTime value among the ErikPartitions referenced from this ErikIndex. The field’s value roughly indicates when the ErikIndex was generated and can be used for troubleshooting and measurement purposes.

For the purposes of this profile, GeneralizedTime values MUST be expressed UTC (Zulu) and MUST include seconds (i.e., times are YYYYMMDDHHMMSSZ), even where the number of seconds is zero. GeneralizedTime values MUST NOT include fractional seconds. See Section 4.1.2.5.2 of [RFC5280].

Design note: using the most recent partitionTime, rather than the local system's notion of "now", helps reduce churn in distributed systems.

3.1.4. The hashAlg field

This field contains the OID of the hash algorithm used to hash the ErikPartitions. The hash algorithm used MUST conform to the RPKI Algorithms and Key Size Profile specification [RFC7935].

3.1.5. The partitionList field

This field is a sequence of PartitionRef instances. There is one PartitionRef for each current ErikPartition. Each PartitionRef is a 3-tuple consisting of the partition identifier, the hash of the partition object, and the size of the partition object.

Information elements are unique with respect to one another and sorted in ascending order of the partition identifier.

3.2. ErikPartition

An ErikPartition represents a subset of manifest objects available under a given FQDN. Each ErikPartition is an ordered listing of the manifest objects' hashes, manifestNumber values, thisUpdate values, and their end-entity certificates' SIA extension values.

3.2.1. The version field

The version number of the ErikPartition object MUST be 0.

3.2.2. The partitionTime field

The partitionTime is the most recent thisUpdate value among the manifests contained within this ErikPartition. The field's value roughly indicates when the ErikPartition was generated and can be used for troubleshooting and measurement purposes.

For the purposes of this profile, GeneralizedTime values MUST be expressed UTC (Zulu) and MUST include seconds (i.e., times are YYYYMMDDHHMMSSZ), even where the number of seconds is zero. GeneralizedTime values MUST NOT include fractional seconds. See Section 4.1.2.5.2 of [RFC5280].

Design note: using the most recent manifest thisUpdate value, rather than the local system’s notion of "now", helps reduce churn in distributed systems.

3.2.3. The hashAlg field

This field contains the OID of the hash algorithm used to hash the manifest objects referenced in this ErikPartition. The hash algorithm used MUST conform to the RPKI Algorithms and Key Size Profile specification [RFC7935].

3.2.4. The manifestList field

This field is a sequence of ManifestRef instances. There is one ManifestRef for each current manifest. A manifest is nominally current until the time specified in nextUpdate or until a manifest is issued with a greater manifestNumber, whichever comes first (see Section 4.2.1 of [RFC9286]).

A ManifestRef is a 4-tuple consisting of the hash of the manifest object, the size of the manifest object, the manifest issuer’s key identifier, the manifestNumber, and the thisUpdate contained within the object, and a sequence of AccessDescription instances from the manifest’s End-Entity certificate’s Subject Information Access extension.

Information elements are unique with respect to one another and sorted in ascending order of the hash.

4. Snapshots and Differentials

Clients should be able to bootstrap initial state by fetching a snapshot. A snapshot contains all RPKI objects a relay associated with a given FQDN as of the time of the snapshot’s production. Snapshots do not contain ErikIndex or ErikPartition objects. Because snapshots are not produced in lockstep with updates to the merkle trees, clients MUST perform a tree comparison after fetching a snapshot.

Note: Snapshots encoding is TBD.

A compressed form ([RFC1951], [RFC1952]) should be used because opportunistic compression in the transport layer (e.g. HTTP compressed content-encoding) is not universally available.

Fetching differences between snapshots ('deltas') might be specified at a later time, or might be concluded to be unnecessary.

Design notes: While the [ustar] format is 'streamable', widely available, and an open standard, a notable downside is that .tar.gz objects likely wont be produced in a deterministic fashion across different implementations, warranting a choice for a bespoke format.

5. Client-side Processing

A client can decide whether or not to fetch ErikIndex and ErikPartition objects, by comparing the hash to previous fetches.

A client can decide whether or not to fetch a given manifest object, by comparing the manifestNumber and thisUpdate with what's locally cached and what's offered by the remote relay.

A client can compute which products listed in the manifest's fileList need to be fetched from the relay.

As there is no concept of 'sessions' (like in RRDP), clients can interchangably use different Erik relay. When one Erik relay generates a HTTP error, the client can try fetching the requested object from another Erik relay.

6. Querying an Erik Relay

6.1. Fetching objects by hash

This specification uses "Named Information" identifiers mapped to .well-known HTTP/HTTPS URLs for object retrieval, as described in [RFC6920].

For example, issuance #54 of ripe-ncc-ta.mft has the following SHA256 digest:
c2d0427bc5a32c42eea1ab5663d592b1fc29c7d4ef16ab0b5e1d631d039dcc21.

To fetch the aforementioned object from an relay hosted at relay.example.net, a client would access the following HTTP URL:
https://relay.example.net/.well-known/ni/sha-256/wtBCe8WjLELuoatWY9WSsfwpx9TvFqsLXh1jHQOdzCE

6.2. Fetching ErikIndex objects

The URIs to fetch ErikIndex objects can be constructed using the following Well-Known URI template with the erik keyword as suffix and the IndexScope as parameter: https://{relay_host}/.well-known/erik/{indexScope}.

For example, the URI to fetch an ErikIndex for the rpki.ripe.net indexScope from a relay at relay.example.net would be: https://relay.example.net/.well-known/erik/rpki.ripe.net.

7. Transport Error Detection and Handling

The client MUST calculate the hash of fetched objects and verify it is the same as the expected hash (which is embedded in the name through which the object was retrieved). If there is a hash mismatch, the client may try fetching the object from a different Erik relay or treat this as a _failed fetch_ (see Section 6.6 of [RFC9286]).

8. Setting Up an Erik Relay

Erik relays can be operated by third parties, without permission from or coordination with publication point operators or CAs.

Relays generate ErikIndexes and ErikPartitions derived from their current validation state, the client then cherry-picks which objects (if any) it wishes to fetch. Relays fetch fresh data from other relays or from CA-designated publication points.

Design notes: a decision must be made a deterministic "manifest-to-partition" assignment scheme. Job’s proof-of-concept relay uses the first few octets of the SHA-256 hash of the manifest for produce as a stable assignment strategy. Other strategies could be to assign manifests to ErikPartitions based on the "hour-of-day" of the CMS signing timestamp or the Authority Key Identifier.

9. Comparison with other RPKI transport protocols

Ignoring obvious "on the wire format" differences between Erik, Rsync, and RRDP; there are a number of key design differences between the protocols. Rsync and RRDP can be described as "general purpose" synchronisation protocols, while the Erik protocol design is RPKI-specific. In the Erik protocol, manifest objects (which RPs require for validation anyway) are an integral part of the signaling layer.

9.1. Comparison with Rsync

In Rsync, the server and the client construct and transfer a full listing of all available objects, and then transfer objects as necessary. In effect, this allows clients to ’jump’ to the latest repository state, regardless of the state of the local cache.

A major downside of Rsync is that the list of files itself can become a burden to transfer. As of June 2025, in order to merely establish whether a client is synchronized or not with the RIPE NCC repository at rpki.ripe.net, as much as 5.8 megabytes of data are exchanged without exchanging any RPKI data.

When synchronizing once an hour, Rsync generally consumes less network traffic than RRDP.

9.2. Comparison with RRDP

The key concept in RRDP is that the client downloads a "journal", containing all add/update/delete operations and replays this journal to arrive at the current repository state.

A major downside of RRDP is that (depending on the RRDP polling interval) clients end up downloading data which has become outdated. Imagine a hypothetical CA which issues and revokes a ROA every 10 minutes and a client that synchronizes every 60 minutes; in effect the client must fetch 5 outdated states, wasting bandwidth.

When synchronizing every 15 minutes, RRDP generally consumes less network traffic than Rsync.

9.2.1. Garbage Collection

In contrast to RRDP, the Erik protocol has no concept of server-specific "stateful" sessions that persist across polling attempts. This obviates the need for withdraw instructions as part of the protocol exchange: clients can simply delete objects that are no longer referenced from their current validation state and refetch them later on if needed.

10. Open Questions

This section is to be removed before publishing as an RFC.

* Should the hash tree heads contain a previous hash field so clients can observe gaps in chains?

* Should ErikPartitions also list the filesize of the referenced manifests? Perhaps there are optimisations possible when the filesizes are communicated to clients?

* Which of the possible deterministic manifest-to-partition assignment strategies yield the best results?

* What is the best archive format for Snapshots? The ustar/gzip combo might not easily yield deterministic results across different implementations.

* Is the concept of Differentials/Deltas needed in Erik Synchronization?

* What will be the upperbound for the number of partitions? (ub-Partitions)

11. Operational Considerations

As of July 2025, the global Internet's RPKI churn rate appears to be 2 new objects per second. The ecosystem is estimated to be composed of ~ 5000 RPKI cache instances and ~ 50 repository servers. Assuming 10 minute fetching intervals and 150 metadata requests per synchronization run (for exchange of Merkle tree data), an Erik relay serving all the Internet's RPKI cache instances would probably need to be able to sustain serving an average of at least 11,000 HTTP requests per second. This order of magnitude in terms of scaling requirements can easily be handled by a single commodity server.

12. Security Considerations

This document makes no changes to RPKI certificate validation procedures.

See Section 5 of [RFC8182] for applicable security considerations.

13. IANA Considerations

13.1. S/MIME Module Identifier

The IANA is requested to add an item to the "SMI Security for S/MIME Module Identifier" registry as follows:

Decimal	Description	References
TDB	id-mod-rpkiErikSynchronization-2025	[this-draft]

13.2. OIDs for Content Types

For the current proof-of-concept phase the id-ct-rpkiErikIndex and id-ct-rpkiErikPartition OIDs were assigned from a PEN arc. This section will be updated once it is clear which IANA-managed registries would be best suited for OID assignment for these object identifiers.

13.3. Well-Known URI

An URI Suffix in the Well-Known URIs registry specific to Erik synchronization will be requested. See https://github.com/protocol-registries/well-known-uris/issues/67 for the request.

The proposed suffix is erik.

14. References

14.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <https://www.rfc-editor.org/info/rfc1951>.

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, DOI 10.17487/RFC1952, May 1996, <https://www.rfc-editor.org/info/rfc1952>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[RFC7935] Huston, G. and G. Michaelson, Ed., "The Profile for Algorithms and Key Sizes for Use in the Resource Public Key Infrastructure", RFC 7935, DOI 10.17487/RFC7935, August 2016, <https://www.rfc-editor.org/info/rfc7935>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9286] Austein, R., Huston, G., Kent, S., and M. Lepinski, "Manifests for the Resource Public Key Infrastructure (RPKI)", RFC 9286, DOI 10.17487/RFC9286, June 2022, <https://www.rfc-editor.org/info/rfc9286>.

[X.690] ITU-T, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.690-202102-I/en>.

14.2. Informative References

[M1987] Merkle, R., "A Digital Signature Based on a Conventional Encryption Function", Advances in Cryptology -- CRYPTO ’87 Proceedings, Lecture Notes in Computer Science, Vol. 293, DOI 10.1007/3-540-48184-2_32, 1988, <https://doi.org/10.1007/3-540-48184-2_32>.

[RFC0677] Johnson, P. and R. Thomas, "Maintenance of duplicate databases", RFC 677, DOI 10.17487/RFC0677, January 1975, <https://www.rfc-editor.org/info/rfc677>.

[RFC1925] Callon, R., "The Twelve Networking Truths", RFC 1925, DOI 10.17487/RFC1925, April 1996, <https://www.rfc-editor.org/info/rfc1925>.

[RFC5781] Weiler, S., Ward, D., and R. Housley, "The rsync URI Scheme", RFC 5781, DOI 10.17487/RFC5781, February 2010, <https://www.rfc-editor.org/info/rfc5781>.

[RFC6480] Lepinski, M. and S. Kent, "An Infrastructure to Support Secure Internet Routing", RFC 6480, DOI 10.17487/RFC6480, February 2012, <https://www.rfc-editor.org/info/rfc6480>.

[RFC8182] Bruijnzeels, T., Muravskiy, O., Weber, B., and R. Austein, "The RPKI Repository Delta Protocol (RRDP)", RFC 8182, DOI 10.17487/RFC8182, July 2017, <https://www.rfc-editor.org/info/rfc8182>.

[ustar] IEEE/Open Group, "ustar Interchange Format", IEEE Std 1003.1-2024, DOI 10.1109/IEEESTD.2018.8423794, June 2024, <https://pubs.opengroup.org/onlinepubs/9799919799/utilities/pax.html#tag_20_94_13_06>.

Acknowledgements

The authors wish to thank George Michaelson, Theo de Raadt, Bob Beck, and Theo Buehler for the lovely conversations that led to this proposal.

This protocol is named after Erik Bais, who passed away in 2024, as a small token of appreciation for his friendship.

Authors' Addresses

Job Snijders
The Netherlands
Email: job@sobornost.net

Tim Bruijnzeels
RIPE NCC
The Netherlands
Email: tim@ripe.net

Tom Harrison
APNIC
Australia
Email: tomh@apnic.net

Wataru Ohgai
JPNIC
Japan
Email: alt@nic.ad.jp