

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 3, 2019

H. Birkholz
Fraunhofer SIT
M. Wiseman
GE Global Research
H. Tschofenig
ARM Ltd.
July 02, 2018

Reference Terminology for Remote Attestation Procedures
draft-birkholz-attestation-terminology-02

Abstract

This document is intended to illustrate and remediate the impedance mismatch of terms related to remote attestation procedures used in different domains today. New terms defined by this document provide a consolidated basis to support future work on attestation procedures in the IETF and beyond.

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 January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements notation	4
2. Basic Roles of RATS	4
3. Computing Context	4
3.1. Formal Semantic Relationships	5
3.2. Characteristics of a Computing Context	6
4. Computing Context Identity	7
5. Attestation Workflow	7
6. Reference Use Cases	8
6.1. The Lying Endpoint Problem	10
6.2. Who am I a talking to?	11
7. Trustworthiness	11
8. Remote Attestation	12
8.1. Building Block Terms	12
9. IANA considerations	13
10. Security Considerations	13
11. Acknowledgements	13
12. Change Log	13
13. References	14
13.1. Normative References	14
13.2. Informative References	14
Authors' Addresses	14

1. Introduction

During its evolution, the term Remote Attestation has been used in multiple contexts and multiple scopes and in consequence accumulated various connotations with slightly different semantic meaning. Correspondingly, Remote Attestation Procedures (RATS) are employed in various usage scenarios and different environments.

In order to better understand and grasp the intend and meaning of specific RATS in the scope of the security area - including the requirements that are addressed by them - this document provides an overview of existing work, its background, and common terminology. As the contribution, from that state-of-the-art a set of terms that provides a stable basis for future work on RATS in the IETF is derived.

The primary application of RATS is to increase the trust and confidence in the integrity of the object characteristics and properties of a system entity that is intended to interact and

exchange data with other system entities remotely. How an objects's characteristics are attested remotely and which characteristics are actually chosen to be attested varies with the requirements of the use cases, or -- in essence -- depends on the risk that is intended to be mitigated via RATS. Effectively, RATS are a vital tool to be used to increase the confidence in the level of trust of a system that is supposed to be a trusted system.

In the remainder of this document a system that is capable to provide an appropriate amount of information about its integrity is considered to be a trustworthy system - or simply trustworthy.

The primary characteristics of a trustworthy system are commonly based on information about the integrity of its intended composition, its enrolled and subsequently installed software components, and the scope of known valid states that a trustworthy system is supposed to operate in.

It is important to note that the activity of attestation itself in principle only provides the evidence that proves the integrity of a (subset) of a system's object characteristics. The provided evidence is used as a basis for further activities. Specific RATS define the higher semantic context about how the evidence is utilized and what RATS actually can accomplish; and what they cannot accomplish, correspondingly. Hence, this document is also intended to provide a map of terms, concepts and applications that illustrate the ecosystem of current applications of RATS.

In essence, a prerequisite for providing an adequate set of terms and definitions in the domain of RATS is a general understanding and a common definitions of "what" RATS can accomplish "how" RATS can to be used.

Please note that this document is still missing multiple reference and is considered "under construction". The majority of definitions is still only originating from IETF work. Future iterations will pull in more complementary definitions from other SDO (e.g. Global Platform, TCG, etc.) and a general structure template to highlight semantic relationships and capable of resolving potential discrepancies will be introduced. A section of context awareness will provide further insight on how attestation procedures are vital to ongoing work in the IETF (e.g. I2NSF & tokbind). The definitions in the section about RATS are still self-describing in this version. Additional explanatory text will be added to provide more context and coherence.

1.1. Requirements notation

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 RFC 2119, BCP 14 [RFC2119].

2. Basic Roles of RATS

The use of the term Remote Attestation Procedures always implies the involvement of at least two parties that each take on a specific role in corresponding RATS - the Attestor role and the Verifier role. Depending on the object characteristics attested and the nature of the parties, information is exchanged via specific types of Interconnects between them. The type of interconnect ranges from GIO pins, to a bus component, to the Internet, or from a direct physical connection, to a wireless association, to a world wide mesh of peers. In other words, virtually every kind communication path (Interconnect) can be used by system entities that take on the role of Attestor and Verifier (in fact, a single party can take on both roles at the same time, but there is only a limited use to this architecture).

Attestor: The role that designates the subject of the remote attestation. A system entity that is the provider of evidence takes on the role of an Attestor.

Verifier: The role that designates the system entity that is the appraiser of the evidence provided by the Attestor. A system entity that is the consumer of evidence takes on the role of a Verifier.

Interconnect: A channel of communication between Attestor and Verifier that enables the appraisal of evidence created by the Attestor by a remote Verifier.

3. Computing Context

This section introduces the term Computing Context in order to simplify the definition of RATS terminology.

The number of approaches and solutions to create things that provide the same capabilities as a "simple physical device" continuously increases. Examples include but are not limited to: the compartmentalization of physical resources, the separation of software instances with different dependencies in dedicated containers, and the nesting of virtual components via hardware-based and software-based solutions.

System entities are composed of system entities. In essence, every physical or logical device is a composite of system entities. In consequence, a composite device also constitutes a system entity. Every component in that composite is a potential Computing Context capable of taking on the roles of Attestor or Verifier. The scope and application of these roles can range from:

- o continuous mutual attestation procedures of every system entity inside a composite device, to
- o sporadic remote attestation of unknown parties via heterogeneous Interconnects.

Analogously, the increasing number of features and functions that constitute components of a device start to blur the lines that are required to categorize each solution and approach precisely. To address this increasingly challenging categorization, the term Computing Context defines the characteristics of the system entities that can take on the role of an Attestor and/or the role of a Verifier. This approach is intended to provide a stable basis of definitions for future solutions that continuous to remain viable long-term.

Computing Context : An umbrella term that combines the scope of the definitions of endpoint [ref NEA], device [ref 1ar], and thing [ref t2trg], including hardware-based and software-based sub-contexts that constitute independent, isolated and distinguishable slices of a Computing Context created by compartmentalization mechanisms, such as Trusted Execution Environments (TEE), Hardware Security Modules (HSM) or Virtual Network Function (VNF) contexts.

3.1. Formal Semantic Relationships

The formal semantic relationship of a Computing Context and the definitions provided by RFC 4949 is as follows.

The scope of the term computing context encompasses

- o an information system,
- o an object and in consequence a system component or a composite of system sub-components, and
- o a system entity or a composite of system entities.

Analogously, a sub-context is a subsystem and as with system components, computing contexts can be nested and therefore be

physical system components or logical ("virtual") system (sub-)components.

The formal semantic relationship is based on the following definitions from RFC 4949.

(Information) System: An organized assembly of computing and communication resources and procedures - i.e., equipment and services, together with their supporting infrastructure, facilities, and personnel - that create, collect, record, process, store, transport, retrieve, display, disseminate, control, or dispose of information to accomplish a specified set of functions.

Object: A system component that contains or receives information.

Subsystem: A collection of related system components that together perform a system function or deliver a system service.

System Component: A collection of system resources that (a) forms a physical or logical part of the system, (b) has specified functions and interfaces, and (c) is treated (e.g., by policies or specifications) as existing independently of other parts of the system. (See: subsystem.)

An identifiable and self-contained part of a Target of Evaluation.

System Entity: An active part of a system - [...] (see: subsystem) - that has a specific set of capabilities.

3.2. Characteristics of a Computing Context

While the semantic relationships highlighted above constitute the fundamental basis to provide a define Computing Context, the following list of object characteristics is intended to improve the application of the term and provide a better understanding of its meaning:

A computing context:

- o provides its own independent environment in regard to executing and running software,
- o provides its own isolated control plane state (by potentially interacting with other Computing
- o Contexts) and may provide a dedicated management interface by which control plane behavior can be effected,

- o can be identified uniquely and therefore reliably differentiated in a given scope, and
- o does not necessarily has to include a network interface with associated network addresses (as required, e.g. by the definition of an endpoint) - although it is very likely to have (access to) one.

In contrast, a docker [ref docker, find a more general term here] context is not a distinguishable isolated slice of an information system and therefore is not an independent Computing Context. [more feedback on this statement is required as the capabilities of docker-like functions evolve continuously]

Examples include: a smart phone, a nested virtual machine, a virtualized firewall function running distributed on a cluster of physical and virtual nodes, or a trust-zone.

4. Computing Context Identity

The identity of a Computing Context provides the basis for creating evidence about data origin authenticity. Confidence in the identity assurance level [NIST SP-800-63-3] or the assurance levels for identity authentication [RFC4949] impacts the confidence in the evidence an Attestor provides.

5. Attestation Workflow

This section introduces terms and definitions that are required to illustrate the scope and the granularity of RATS workflows in the domain of security automation. Terms defined in the following sections will be based on this workflow-related definitions.

In general, RATS are composed of iterative activities that can be conducted in intervals. It is neither a generic set of actions nor simply a task, because the actual actions to be conducted by RATS can vary significantly depending on the protocols employed and types of Computing Contexts involved.

Activity: A sequence of actions conducted by Computing Contexts that compose a remote attestation procedure. The actual composition of actions can vary, depending on the characteristics of the Computing Context they are conducted by/in and the protocols used to utilize an Interconnect. A single activity provides only a minimal amount of semantic context, e.g. defined by the activity's requirements imposed on the Computing Context, or via the set of actions it is composed of. Example: The conveyance of

cryptographic evidence or the appraisal of evidence via imperative guidance.

Task: "A piece of work to be done or undertaken."

In the scope of RATS, a task is a procedure to be conducted.

Example: A Verifier can be tasked with the appraisal of evidence originating from a specific type of Computing Contexts providing appropriate identities.

Action: "The accomplishment of a thing usually over a period of time, in stages, or with the possibility of repetition."

In the scope of RATS, an action is the execution of an operation or function in the scope of an activity conducted by a Computing Context. A single action provides no semantic context by itself, although it can limit potential semantic contexts of RATS to a specific scope. Example: Signing an existing public key via a specific openssl library, transmitting data, or receiving data are actions.

Procedure: "A series of actions that are done in a certain way or order."

In the scope of RATS, a procedure is a composition of activities (sequences of actions) that is intended to create a well specified result with a well established semantic context. Example: The activities of attestation, conveyance and verification compose a remote attestation procedure.

6. Reference Use Cases

This document provides NNN prominent examples of use cases attestation procedures are intended to address:

- o Verification of the source integrity of a computing context via data integrity proofing of installed software instances that are executed, and
- o Verification of the identity proofing of a computing context.

These use case summary highlighted above is based in the following terms defined in RFC4949 and complementary sources of terminology:

Assurance: An attribute of an information system that provides grounds for having confidence that the system operates such that the system's security policy is enforced [RFC4949] (see Trusted System below).

In common criteria, assurance is the basis for the metric level of assurance, which represents the "confidence that a system's principal security features are reliably implemented".

The NIST Handbook [get ref from 4949] notes that the levels of assurance defined in Common Criteria represent "a degree of confidence, not a true measure of how secure the system actually is. This distinction is necessary because it is extremely difficult-and in many cases, virtually impossible-to know exactly how secure a system is."

Historically, assurance was well-defined in the Orange Book [<http://csrc.nist.gov/publications/history/dod85.pdf>] as "guaranteeing or providing confidence that the security policy has been implemented correctly and that the protection-relevant elements of the system do, indeed, accurately mediate and enforce the intent of that policy. By extension, assurance must include a guarantee that the trusted portion of the system works only as intended."

Confidence: The definition of correctness integrity in [RFC4949] notes that "source integrity refers to confidence in data values". Hence, confidence in an attestation procedure is referring to the degree of trustworthiness of an attestation activity that produces evidence (Attestor), of an conveyance activity that transfers evidence (interconnect), and of a verification activity that appraises evidence (Verifier), in respect to correctness integrity.

Identity: Defined by [RFC4949] as the collective aspect of a set of attribute values (i.e., a set of characteristics) by which a system user or other system entity is recognizable or known. (See: authenticate, registration. Compare: identifier.)

There are different scopes an identity can apply to:

Singular identity: An identity that is registered for an entity that is one person or one process.

Shared identity: An identity that is registered for an entity that is a set of singular entities in which each member is authorized to assume the identity individually, and for which the registering system maintains a record of the singular entities that comprise the set. In this case, we would expect each member entity to be registered with a singular identity before becoming associated with the shared identity.

Group identity: An identity that is registered for an entity that is a set of entities (2) for which the registering system does not maintain a record of singular entities that comprise the set.

Identity Proofing: A process that vets and verifies the information that is used to establish the identity of a system entity.

Source Integrity: The property that data is trustworthy (i.e., worthy of reliance or trust), based on the trustworthiness of its sources and the trustworthiness of any procedures used for handling data in the system.

Data Integrity: (a) The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner. (See: data integrity service. Compare: correctness integrity, source integrity.)

(b) The property that information has not been modified or destroyed in an unauthorized manner.

Correctness: The property of a system that is guaranteed as the result of formal verification activities.

Correctness integrity: The property that the information represented by data is accurate and consistent.

Verification: (a) The process of examining information to establish the truth of a claimed fact or value.

(b) The process of comparing two levels of system specification for proper correspondence, such as comparing a security model with a top-level specification, a top-level specification with source code, or source code with object code.

Forward Authenticity (FA): A property of secure communication protocols, in which later compromise of the long-term keys of a data origin does not compromise past authentication of data from that origin. FA is achieved by timely recording of assessments of the authenticity from entities (via "audit logs" during "audit sessions") that are authorized for this purpose, in a time frame much shorter than that expected for the compromise of the long-term keys.

6.1. The Lying Endpoint Problem

A very prominent goal of attestation procedures - and therefore a suitable example used as reference in this document - is to address the "lying endpoint problem".

Information created, relayed, or, in essence, emitted by a computing context does not have to be correct. There can be multiple reasons why that is the case and the "lying endpoint problem" represents a scenario, in which the reason is the compromization of computing contexts with malicious intent. A compromised computing context could try to "pretend" to be integer, while actually feeding manipulated information into a security domain, therefore compromising the effectiveness of automated security functions. Attestation - and remote attestation procedures specifically - is an approach intended to identify compromised software instances in computing contexts.

Per definition, a "lying endpoint" cannot be "trusted system".

Trusted System: A system that operates as expected, according to design and policy, doing what is required - despite environmental disruption, human user and operator errors, and attacks by hostile parties - and not doing other things.

Remote attestation procedures are intended to enable the consumer of information emitted by an computing context to assess the validity and integrity of the information transferred. The approach is based on the assumption that if evidence can be provided in order to prove the integrity of every software instance installed involved in the activity of creating the emitted information in question, the emitted information can be considered valid and integer.

In contrast, such evidence has to be impossible to create if the software instances used in a computing context are compromised. Attestation activities that are intended to create this evidence therefore also to also provide guarantees about the validity of the evidence they can create.

6.2. Who am I a talking to?

[working title, write up use case here, ref teep requirements]

7. Trustworthiness

A "lying endpoint" is not trustworthy.

Trusted System: A system that operates as expected, according to design and policy, doing what is required - despite environmental disruption, human user and operator errors, and attacks by hostile parties - and not doing other things.

Trustworthy: pull in text here

8. Remote Attestation

Attestation: An object integrity authentication facilitated via the creation of a claim about the properties of an Attestor, such that the claim can be used as evidence.

Conveyance: The transfer of evidence from the Attestor to the Verifier.

Verification: The appraisal of evidence by evaluating it against declarative guidance.

Remote Attestation: A procedure composed of the activities attestation, conveyance and verification.

8.1. Building Block Terms

[working title, pulled from various sources, vital]

Attestation Identity Key (AIK): A special purpose signature (therefore asymmetric) key that supports identity related operations. The private portion of the key pair is maintained confidential to the computing context via appropriate measures (that have a direct impact on the level of confidence). The public portion of the key pair may be included in AIK credentials that provide a claim about the computing context.

Claim: A piece of information asserted about a subject. A claim is represented as a name/value pair consisting of a Claim Name and a Claim Value [RFC7519]

In the context of SACM, a claim is also specialized as an attribute/value pair that is intended to be related to a statement [I-D.ietf-sacm-terminology].

Computing Context Characteristics: The composition, configuration and state of a computing context.

Evidence: A trustworthy set of claims about an computing context's characteristics.

Identity: A set of claims that is intended to be related to an entity. [merge with RFC4949 definition above]

Integrity Measurements: Metrics of computing context characteristics (i.e. composition, configuration and state) that affect the confidence in the trustworthiness of a computing context. Digests

of integrity measurements can be stored in shielded locations (e.g. a PCR of a TPM).

Reference Integrity Measurements: Signed measurements about a computing context's characteristics that are provided by a vendor or manufacturer and are intended to be used as declarative guidance [I-D.ietf-sacm-terminology] (e.g. a signed CoSWID).

Trustworthiness: The qualities of computing context characteristics that guarantee a specific behavior specified by declarative guidance. Trustworthiness is not an absolute property but defined with respect to a computing context, corresponding declarative guidance, and has a scope of confidence. A trusted system is trustworthy. [refactor definition with RFC4949 terms]

Trustworthy Computing Context: a computing context that guarantees trustworthy behavior and/or composition (with respect to certain declarative guidance and a scope of confidence). A trustworthy computing context is a trusted system.

Trustworthy Statement: evidence that trustworthy conveyed by a computing context that is not necessarily trustworthy. [update with tamper related terms]

9. IANA considerations

This document will include requests to IANA:

- o first item
- o second item

10. Security Considerations

There are always some.

11. Acknowledgements

Maybe.

12. Change Log

No changes yet.

13. References

13.1. Normative References

- [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>>.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <<https://www.rfc-editor.org/info/rfc4949>>.

13.2. Informative References

- [I-D.ietf-sacm-terminology] Birkholz, H., Lu, J., Strassner, J., Cam-Winget, N., and A. Montville, "Security Automation and Continuous Monitoring (SACM) Terminology", draft-ietf-sacm-terminology-14 (work in progress), December 2017.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.

Authors' Addresses

Henk Birkholz
Fraunhofer SIT
Rheinstrasse 75
Darmstadt 64295
Germany

Email: henk.birkholz@sit.fraunhofer.de

Monty Wiseman
GE Global Research
USA

Email: monty.wiseman@ge.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
UK

Email: hannes.tschofenig@gmx.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 25, 2018

P. Hallam-Baker
Comodo Group Inc.
May 24, 2018

Data At Rest Encryption: DARE Container
draft-hallambaker-dare-container-00

Abstract

This document describes DARE Container, a message and file syntax that allows a sequence of data frames to be represented with cryptographic integrity, signature and encryption enhancements to be constructed in an append only format. The format supports data integrity checks using digest chains and Merkle trees. The simplest supports efficient append only write operations and efficient read operations in either the forward or reverse direction. Support for efficient random-access reads may be provided through the use of binary trees or index records appended to the end of the file.

This document is also available online at
<http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>
[1] .

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 November 25, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Definitions	5
2.1. Related Specifications	5
2.2. Requirements Language	5
3. Container Format	5
3.1. Container Profile	6
3.1.1. Index Profiles	7
3.1.2. Content Profiles	7
3.2. Payload Encryption	8
4. Index Mechanisms	8
4.1. Tree	8
4.2. Position Index	9
4.3. Metadata Index	9
5. Integrity Mechanisms	9
5.1. Digest Chain calculation	9
5.2. Binary Merkle tree calculation	10
6. Reference	10
6.1. Container Headers	10
6.1.1. Structure: ContainerHeaderFirst	10
6.1.2. Structure: ContainerHeader	11
6.2. Content Metadata Structure	12
6.2.1. Structure: ContentMeta	12
6.3. Index Structures	12
6.3.1. Structure: ContainerIndex	12
6.3.2. Structure: IndexPosition	13
6.3.3. Structure: KeyValue	13
6.3.4. Structure: IndexMeta	13
6.4. Signature	13
7. Further Work	14
7.1. Fast open with random access	14
7.2. Partitioning of very large data sets across hosts	15
7.3. Filtering and redaction	15
7.4. Encryption of large data blocks	15
7.5. Concurrent Writes	16
8. Security Considerations	16
9. IANA Considerations	16

10. Acknowledgements	16
11. Appendix A: Examples and Test Vectors	16
11.1. Simple container	16
11.2. Payload and chain digests	18
11.3. Merkle Tree	19
11.4. Signed container	21
11.5. Encrypted container	21
12. Appendix B	24
13. References	25
13.1. Normative References	25
13.2. Informative References	25
13.3. URIs	26
Author's Address	26

1. Introduction

DARE Container is a message and file syntax that allows a sequence of data frames to be represented with cryptographic integrity, signature, and encryption enhancements to be constructed in an append only format. DARE Container was developed in response to needs that arose out of the design of the Mathematical Mesh [draft-hallambaker-mesh-architecture] . It is built on the binary encodings of JSON data objects, JSON-B and JSON-C [draft-hallambaker-jsonbcd] and the DARE Message format [draft-hallambaker-dare-message] .

The high level requirements supported include:

- o Recording Mesh transactions in persistent storage.
- o Synchronizing transaction logs between hosts.
- o Representing message archives (aka mail spool)
- o Signing and encrypting single data items.

The features supported by DARE Container include:

- o The format is append only, thus providing for rapid write operations and enabling the use of technologies that provide atomic transactions.
- o All length and index values support the use of integers of at least 64 bits.
- o Data frames may be of variable length.

- o Data frames may be read in either direction. This allows the last n frames to be read as efficiently as the first n frames.
- o Appending a data frame to an existing file is efficient taking no more than $\log_2(n)$ operations.
- o A binary tree index MAY be constructed on an incremental basis, allowing random access to the n th record in the file in $\log_2(n)$ operations.
- o An index MAY be appended to an existing container to allow random access to the n th record in the file in $\log_2(n)$ operations
- o Permits the use of modern data encodings (e.g. JSON [RFC7159]).
- o Supports digital signature and public key operations on the payloads of individual data frames.
- o Data frame content (i.e. payload data) may be overwritten without invalidating the integrity of any other frame. This allows content to be expunged in exigent circumstances (court order, regulatory, confidentiality breach, etc.) without compromising the integrity of the rest of the data in the container.

Many file proprietary formats are in use that support some or all of these capabilities but only a handful have public, let alone open, standards. DARE Container is designed to provide a superset of the capabilities of existing message and file syntaxes, including:

- o Cryptographic Message Syntax [RFC5652] defines a syntax used to digitally sign, digest, authenticate, or encrypt arbitrary message content.
- o The.ZIP File Format specification [ZIPFILE] developed by Phil Katz.
- o The BitCoin Block chain [BLOCKCHAIN] .
- o JSON Web Encryption and JSON Web Signature

Attempting to make use of these specifications in a layered fashion would require at least three separate encoders and introduce unnecessary complexity.

Every data format represents a compromise between different concerns, in particular:

Data Storage The space required to record data in the encoding.

Memory Overhead The additional volatile storage (RAM) required to maintain indexes etc. to support efficient retrieval operations.

Number of Operations The number of operations required to retrieve data from or append data to an existing encoded sequence.

Number of Disk Seek Operations Optimizing the response time of magnetic storage media to random access read requests has traditionally been one of the central concerns of database design. The DARE Container format is designed to the assumption that this will cease to be a concern as solid state media replaces magnetic.

While the cost of storage of all types has declined rapidly over the past decades, so has the amount of data to be stored. DARE Container represents a pragmatic balance of these considerations for current technology. In particular, since payload volumes are likely to be very large, memory and operational efficiency are considered higher priorities than data volume.

2. Definitions

2.1. Related Specifications

DARE Container makes use of the following related standards and specifications.

Encoding Content frame headers are encoded using JavaScript Object Notation (JSON) [RFC7159] , JSON-B or JSON-C [draft-hallambaker-jsonbcd] .

Cryptography The encryption and signature schemes used are based on JSON Web Signature [RFC7515] and JSON Web Encryption [RFC7516] .

2.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] .

3. Container Format

A DARE Container consists of a sequence of JBCD Frames containing up to three ordered JBCD records as follows:

Header Record [required] Metadata of any type including container metadata, payload metadata and DARE message headers.

Payload Record [optional] The frame data. Payload records are either complete or incremental. A complete payload contains a complete unit of whatever data is being written to the log. An incremental payload contains a part of a unit that has been split across multiple records.

Guard Record [optional] An opaque data record with a known value written to the end of a frame to allow a writer to avoid corruption of the container framing data by detecting an incomplete append state.

A DARE frame consists of a forward length indicator, the framed data and a reverse length indicator. The reverse length indicator is written out backwards to allow the frame to be read in the reverse direction:

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html> [2].]]

JBCD Bidirectional Frame

When first reading an existing file, an application will typically read the first frame and the last frame (if the container has more than one frame). This allows the reader to quickly determine the format(s) used by the container, the number of frames in the container and the location of any index frames (if present).

The container format is designed to support creation of write-once and append-only file formats. Each frame SHOULD be written as an atomic operation.

The first frame in a container and the first record in a frame have special roles that are described in this document.

- o The first frame in a container describes the container format options and defaults. These include the range of encoding options for frame metadata supported and the container profiles to which the container conforms.
- o The first record in a frame MUST NOT contain payload data

3.1. Container Profile

A key objective of the DARE Container format is that the simplest possible reader be capable of reading any container file albeit with possibly reduced performance.

A Container MAY conform to one or more profiles. Conforming to a profile typically requires a writer to provide additional information when writing a file but does not require a reader to interpret it unless use of a feature (e.g. authentication) that depends on the additional information is required.

3.1.1. Index Profiles

The following profiles are currently defined:

Tree Frame headers contain `IndexPosition` entries that specify the start position of previous frames. This enables efficient random access to any frame in the file.

Digest Frame headers contain `PayloadDigest` entries that specify the digest value of the corresponding payload data in that frame.

Chain Frame headers contain `ChainDigest` entries that link each frame to the preceding frame.

Merkle Frame headers contain `TreeDigestPartial` and `TreeDigestFinal` entries linking all the frames in the container in a binary Merkle Tree.

The use of Chain and Merkle Trees for integrity checks is described below.

The use of Tree and Index frames is described below.

3.1.2. Content Profiles

The following profiles are currently defined:

Singleton A container with exactly one content frame. A container declared as a singleton frame cannot have additional content frames appended (but metadata frames may be)

Multi A container whose payload data is limited to content frames. A container declared as a multi container may contain 0, 1 or more content frames.

Archive A multi-container whose payload data is limited to content frames whose last frame contains a metadata index for the content frames in the container.

Unitary A multi-container in which each frame represents exactly one payload object.

Serial A multi-container in which payload objects MAY be split across multiple consecutive frames.

Interleaved A multi-container in which payload objects MAY be split across multiple frames which may in turn be interleaved with frames containing other payload objects in complete or partial form.

3.2. Payload Encryption

DARE container payloads MAY be encrypted as DARE Message Enhanced Data Sequences [draft-hallambaker-dare-message] . This specification builds on JSON Web Encryption (JWE) [RFC7516] to provide a flexible framework allowing a single key exchange to be applied to encrypt multiple data sequences.

The DARE Container and DARE Message format are designed to compliment each other:

- o DARE Message Format allows a Master Key established in a single key exchange to be applied to multiple DARE related messages.
- o DARE Container Format allows sets of DARE related messages to be organized so that individual messages and the keying material required to interpret them can be efficiently retrieved.

4. Index Mechanisms

An index may be appended to an existing file at any time. Since the use of bidirectional frames makes reading the last record is as efficient as reading the first, the last record in an indexed file is usually either the index itself or a pointer to the last index.

An index frame consists of a frame header

Use of index frames provides read access to any record in the file in $O(1)$ operations but attempting to compiling a complete index with every write incurs an $O(n)$ penalty on write for both operations and storage. Accordingly, random read access to a file while it is being written is better supported using an index tree.

4.1. Tree

Binary search is supported by means of the `TreePosition` parameter specified in the `FrameHeader`. This parameter specifies the value of the immediately preceding apex.

Calculation of the immediately preceding apex is most easily described by representing the array index in binary with base of 1 (rather than 0). An array index that is a power of 2 (2, 4, 8, 16, etc.) will be the apex of a complete tree. Every other array index has the value of the sum of a set of powers of 2 and the immediately preceding apex will be the value of the next smallest power of 2 in the sum.

For example, to find the immediately preceding apex for frame 5, we add 1 to get 6. $6 = 4 + 2$, so we ignore the 2 and the preceding frame is 4.

The values of Tree Position are shown for the first 8 frames in figure xx below:

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html> [3].]]

Merkle Tree Integrity check

An algorithm for efficiently calculating the immediately preceding apex is provided in Appendix C.

4.2. Position Index

Contains a table of index, position pairs pointing to prior locations in the file.

4.3. Metadata Index

Contains a list of IndexMeta entries. Each entry contains a metadata description and a list of frame indexes (not positions) of frames that match the description.

5. Integrity Mechanisms

Frame sequences in a DARE container MAY be protected against a frame insertion attack by means of a digest chain, a binary Merkle tree or both.

5.1. Digest Chain calculation

A digest chain is simple to implement but can only be verified if the full chain of values is known. Appending a frame to the chain has $O(1)$ complexity but verification has $O(n)$ complexity:

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html> [4].]]

Hash chain integrity check

The value of the chain digest for the the first frame (frame 0) is $H(IV+H(\text{Payload}_0))$, where IV is an initialization vector consisting of a string of zero bytes and payload_n is the sequence of payload data bytes for frame n

The value of the chain digest for frame n is $H(H(\text{Payload}_{n-1}+H(\text{Payload}_n)))$, where $A+B$ stands for concatenation of the byte sequences A and B.

5.2. Binary Merkle tree calculation

The tree index mechanism describe earlier may be used to implement a binary Merkle tree. The value `TreeDigest` specifies the apex value of the tree for that node.

Appending a frame to the chain has $O(\log_2 n)$ complexity provided that the container format supports at least the binary tree index. Verifying a chain has $O(\log_2 n)$ complexity, provided that the set of necessary digest inputs is known.

To calculate the value of the tree digest for a node, we first calculate the values of all the sub trees that have their apex at that node and then calculate the digest of that value and the immediately preceding local apex.

6. Reference

TBS stuff

6.1. Container Headers

TBS stuff

6.1.1. Structure: ContainerHeaderFirst

Inherits: ContainerHeader

Inherits: ContainerHeader

DataEncoding: String (Optional) Specifies the data encoding for the header section of for the following frames. This value is ONLY valid in Frame 0 which MUST have a header encoded in JSON.

6.1.2. Structure: ContainerHeader

Inherits: DAREHeader

Describes a container header. A container header MAY contain any DARE Message header.

Index: Integer (Optional) The record index within the file. This MUST be unique and satisfy any additional requirements determined by the ContainerType.

ContainerType: String (Optional) Specifies the container type for the following records.

IsMeta: Boolean (Optional) If true, the current frame is a meta frame and does not contain a payload.

Note: Meta frames MAY be present in any container. Applications MUST accept containers that contain meta frames at any position in the file. Applications MUST NOT interpret a meta frame as a data frame with an empty payload.

UniqueID: String (Optional) Unique object identifier

ContentMeta: ContentMeta (Optional) Content meta data.

TreePosition: Integer (Optional) Position of the frame containing the apex of the preceding sub-tree.

IndexPosition: Integer (Optional) Specifies the position in the file at which the last index entry is to be found

ExchangePosition: Integer (Optional) Specifies the position in the file at which the key exchange data is to be found

ContainerIndex: ContainerIndex (Optional) An index of records in the current container up to but not including this one.

PayloadDigest: Binary (Optional) If present, contains the digest of the Payload.

ChainDigest: Binary (Optional) If present, contains the digest of the PayloadDigest values of this frame and the frame immediately preceding.

TreeDigest: Binary (Optional) If present, contains the Binary Merkle Tree digest value.

Event: String (Optional) Unique object identifier

Labels: String [0..Many] List of labels that are applied to the payload of the frame.

KeyValues: KeyValue [0..Many] List of key/value pairs describing the payload of the frame.

First: Integer (Optional) Frame number of the first object instance value.

Previous: Integer (Optional) Frame number of the immediately prior object instance value

6.2. Content Metadata Structure

TBS stuff

6.2.1. Structure: ContentMeta

Information describing the object instance

ContentType: String (Optional) The content type field as specified in JWE

Paths: String [0..Many] List of filename paths for the payload of the frame.

UniqueID: String (Optional) Unique object identifier

Created: DateTime (Optional) Initial creation date.

Modified: DateTime (Optional) Date of last modification.

6.3. Index Structures

TBS stuff

6.3.1. Structure: ContainerIndex

A container index

Full: Boolean (Optional) If true, the index is complete and contains position entries for all the frames in the file. If absent or false, the index is incremental and only contains position entries

for records added since the last frame containing a ContainerIndex.

Positions: IndexPosition [0..Many] List of container position entries

Metas: IndexMeta [0..Many] List of container position entries

6.3.2. Structure: IndexPosition

Specifies the position in a file at which a specified record index is found

Index: Integer (Optional) The record index within the file.

Position: Integer (Optional) The record position within the file relative to the index base.

6.3.3. Structure: KeyValue

Specifies a key/value entry

Key: String (Optional) The key

Value: String (Optional) The value corresponding to the key

6.3.4. Structure: IndexMeta

Specifies the list of index entries at which a record with the specified metadata occurs.

Index: Integer [0..Many] List of record indices within the file where frames matching the specified criteria are found.

ContentType: String (Optional) Content type parameter

Paths: String [0..Many] List of filename paths for the current frame.

Labels: String [0..Many] List of labels that are applied to the current frame.

6.4. Signature

Payload data MAY be signed using a JWS [RFC7515] as applied in the DARE Message format [draft-hallambaker-dare-message] .

Signatures are specified by the Signatures parameter in the content header. The data that the signature is calculated over is defined by the typ parameter of the Signature as follows.

Payload The frame payload data.

PayloadDigest The value of the PayloadDigest parameter

ChainDigest The value of the ChainDigest parameter

TreeDigestFinal The value of the TreeDigestFinal parameter

If the typ parameter is absent, the value Payload is implied.

A frame MAY contain multiple signatures created with the same signing key and different typ values.

The use of signatures over chain and tree digest values permit multiple frames to be validated using a single signature verification operation.

7. Further Work

The container format is intended to be the basis of future work to support:

- o Very large container sizes (larger than the size of the host's memory).
- o Partitioning of very large data sets across multiple hosts with parallel append.
- o Fault tolerance

7.1. Fast open with random access

The container format is designed to be capable of supporting efficient random access to frames in containers considerably larger than the processing memory of the host computer without the need to pre-load indexes.

A combination of the following strategies is being considered:

- o Use memory mapped file views to container data to optimize random access times while controlling memory use and time taken to construct memory views.

- o When the container is first bound, use the binary tree index data in TreePosition parameters to support random access operations until index building is complete.
- o Perform Index building operations as a non-blocking background task.

7.2. Partitioning of very large data sets across hosts

While storage devices capable of storing tens of Tb of data with RAID redundancy are commonplace, it is generally desirable that there be at least as many CPU cores as disks. Thus, partitioning of data sets across multiple hosts becomes desirable for throughput even if a single host could handle the storage requirement.

7.3. Filtering and redaction

In the types of applications envisaged in the Mesh, almost every data set may be reduced to collections that are bound to a single account. While it is obviously desirable that a user's mail messages (for example) be replicated across multiple machines to provide fault tolerance, fragmenting the copies of this data set across multiple machines should be avoided unless the data volumes are so large as to require it.

7.4. Encryption of large data blocks

The encoding scheme is 64-bit clean throughout and thus supports containers and frames as large as 18 petabytes. Larger data volumes could be supported through use of 128-bit integer pointers but even if the technology to support such data volumes were developed, it is highly unlikely anyone would want to represent data sets anywhere near this size in a serial format.

Due to limitations in the design of the encryption schemes that may be used (e.g. AES-GCM), the maximum encrypted frame size is 64GB. While this is not currently a major concern for encryption of individual data files, it is easy to see situations in which an archive of encrypted files could exceed that amount. One possibility would be to define a modification to AES -GCM which caused the encryption key to be incremented by a fixed amount after encrypting a certain amount of data, though this might well present implementation challenges unless the maximum data block size was chosen to be deliberately small so as to force code paths to be exercised. Another possibility would be to limit the size of encrypted data frames by requiring the frame pointer to be no larger than 32 bits and require larger data items to be represented as a sequence of frames.

7.5. Concurrent Writes

The container format deliberately avoids support for concurrent write operations. Should this be desirable, some mechanism must be provided to cache write fragments to an intermediate file and then consolidate them for writing to the master log.

8. Security Considerations

9. IANA Considerations

10. Acknowledgements

11. Appendix A: Examples and Test Vectors

The data payloads in all the following examples are identical, only the authentication and/or encryption is different.

- o Frame 1..n consists of 300 bytes being the byte sequence 00, 01, 02, etc. repeating after 256 bytes.

For conciseness, the wire format is omitted for examples after the first, except where the data payload has been transformed, (i.e. encrypted).

11.1. Simple container

Here the simple container:

```

f4 5b
f0 59
  7b 0a 20 20 22 49 6e 64    65 78 22 3a 20 30 2c 0a
  20 20 22 43 6f 6e 74 61    69 6e 65 72
  ...
  7d 2c 0a 20 20 22 44 61    74 61 45 6e 63 6f 64 69
  6e 67 22 3a 20 22 4a 53    4f 4e 22 7d
5b f4
f5 01 40
f0 0f
  7b 0a 20 20 22 49 6e 64    65 78 22 3a 20 31 7d
f1 01 2c
  00 01 02 03 04 05 06 07    08 09 0a 0b 0c 0d 0e 0f
  10 11 12 13 14 15 16 17    18 19 1a 1b 1c 1d 1e 1f
  20 21 22 23 24 25 26 27    28 29 2a 2b 2c 2d 2e 2f
  30 31 32 33 34 35 36 37    38 39 3a 3b 3c 3d 3e 3f
  40 41 42 43 44 45 46 47    48 49 4a 4b 4c 4d 4e 4f
  50 51 52 53 54 55 56 57    58 59 5a 5b 5c 5d 5e 5f
  60 61 62 63 64 65 66 67    68 69 6a 6b 6c 6d 6e 6f
  70 71 72 73 74 75 76 77    78 79 7a 7b 7c 7d 7e 7f
  80 81 82 83 84 85 86 87    88 89 8a 8b 8c 8d 8e 8f
  90 91 92 93 94 95 96 97    98 99 9a 9b 9c 9d 9e 9f
  a0 a1 a2 a3 a4 a5 a6 a7    a8 a9 aa ab ac ad ae af
  b0 b1 b2 b3 b4 b5 b6 b7    b8 b9 ba bb bc bd be bf
  c0 c1 c2 c3 c4 c5 c6 c7    c8 c9 ca cb cc cd ce cf
  d0 d1 d2 d3 d4 d5 d6 d7    d8 d9 da db dc dd de df
  e0 e1 e2 e3 e4 e5 e6 e7    e8 e9 ea eb ec ed ee ef
  f0 f1 f2 f3 f4 f5 f6 f7    f8 f9 fa fb fc fd fe ff
  00 01 02 03 04 05 06 07    08 09 0a 0b 0c 0d 0e 0f
  10 11 12 13 14 15 16 17    18 19 1a 1b 1c 1d 1e 1f
  20 21 22 23 24 25 26 27    28 29 2a 2b
40 01 f5

```

Figure 1

The header values are:

Frame 0

```

{
  "Index": 0,
  "ContainerType": "List",
  "ContentMeta": {},
  "DataEncoding": "JSON"}

```

Figure 2

Frame 1

```
{  
  "Index": 1}
```

Figure 3

11.2. Payload and chain digests

Frame 0

```
{  
  "Index": 0,  
  "ContainerType": "Chain",  
  "ContentMeta": {},  
  "PayloadDigest": "z4PhNX7vuL3xVChQ1m2AB9Yg5AULVxXcg_SpIdNs6c5H0  
NE8XYXysP-DGNKHfuwvY7kxvUdBeoGlODJ6-SfaPg",  
  "ChainDigest": "FEHy24Y6cLModDXWH31kVc2a3TdhjXPooKHpLab2Jbs0lYQ  
nJolmowXAYHhkOGY0kg3jrKNTjds0myf4Dwlsg",  
  "DataEncoding": "JSON"}
```

Figure 4

Frame 1

```
{  
  "Index": 1,  
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZD  
lZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",  
  "ChainDigest": "7JaijhBvQUOjBiOl_Zt6NtJil8iB0rW9HeM_4iYooc_AaAf  
utlF0LLVY6PO7INB-eztypyEqVzgMil9JkjtRGQ"}
```

Figure 5

Frame 2

```
{  
  "Index": 2,  
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZD  
lZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",  
  "ChainDigest": "wJZFyD6lnntCJ0Bv80l6-Cn-sR2u3iD0zCRjOLxje8dsKIu  
UnP4XlmgeNenNDBdXysrFs3vVAqkC-hfSAPF0Aw"}
```

Figure 6

Frame 3

```
{
  "Index": 3,
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZD
lZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "ChainDigest": "RORNZxIcM23cZtXPh9vuHhkgiGa_O4a0ZiU0ku2OK4dB974
clvh5F0VZsX7IwVBayAG2nDTdqhyZ-qOnTRiumA"}
```

Figure 7

11.3. Merkle Tree

Frame 0

```
{
  "Index": 0,
  "ContainerType": "Merkle",
  "ContentMeta": {},
  "PayloadDigest": "z4PhNX7vuL3xVChQlm2AB9Yg5AULVxXcg_SpIdNs6c5H0
NE8XYXysP-DGNKHfuwvY7kxvUdBeoGlODJ6-SfaPg",
  "TreeDigest": "FEHy24Y6cLModDXWH31kVc2a3TdjhXPooKHpLAb2Jbs0lYQn
JolmowXAYHhkOGY0kg3jrKNTjds0myf4Dwlstdg",
  "DataEncoding": "JSON"}
```

Figure 8

Frame 1

```
{
  "Index": 1,
  "TreePosition": 0,
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZD
lZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest": "fPTYagAvSDP_755jpFUs-Wq6cgvtr5vrFwW-E12vsrbq1ReN
sGzp-V2XqzFPiWaUckACPjegD7ioelbGzxoWQQ"}
```

Figure 9

Frame 2

```
{
  "Index": 2,
  "TreePosition": 315,
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZD
lZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest": "7fyKKQNLGEeHXloCsV8NtOdPm615SkDnMlvkcexx2tOuVd5k
kZIdLdsWRCLic9luTSsUN6D6_-c-8ftbhL9dJg"}
```

Figure 10

Frame 3

```
{
  "Index": 3,
  "TreePosition": 315,
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZD
lZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest": "b9ca9Pv-6fxUg-V3ulOhhRngxebkZCxyDmWhQUYeADmSvvPb
jMcNTUJxdDpKlMPrDBInSWMChinsc5s9Tv4byw"}
```

Figure 11

Frame 4

```
{
  "Index": 4,
  "TreePosition": 1441,
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZD
lZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest": "glhQeWJgDlNoTSGfMb6NhQk5-p6iaAI2_GiAhBM-F2Cp3UvJ
7AR_bc2Drp5YElGXAzC2K5qZ30l7j2D-jqykFw"}
```

Figure 12

Frame 5

```
{
  "Index": 5,
  "TreePosition": 1441,
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZD
lZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest": "p89BhjJAgMMoSrOmot6oaBGa6Dgz-zogZjZ9mm1Iz4yLHxm9
7nWAIBaZFiClXkuCoP-tr3tag_rHoZhgQV8_PQ"}
```

Figure 13

Frame 6

```
{
  "Index": 6,
  "TreePosition": 2570,
  "PayloadDigest": "8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZD
lZeaWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest": "HEA7EeUGfSjZqjmN3PDp0FVbnixBBXfSQAYm_rNPHVWJVMDu
3SfmXKvN_yBTtMXk-Jad9cyXDKsecLNHLyoQWg"}
```

Figure 14

11.4. Signed container

11.5. Encrypted container

The following example shows a container in which all the frame payloads are encrypted under the same master secret established in a key agreement specified in the first frame.

Frame 0

```
{
  "enc": "A256CBC",
  "recipients": [{
    "kid": "MCRW5-YRWQK-M4HE3-D4JE6-IYQLM-22ITA-A",
    "epk": {
      "PublicKeyDH": {
        "kid": "MBSNR-T4GZ6-LBZTD-G2CP3-MGT4K-J4VGF-A",
        "Domain": "YE6bnq1MlX5ojaJto6PLP_PEWa",
        "Public": "Cmh_MVvRHND0H6GB2Gjke8n8z3UvqNXmNcYQRKPMqEvSsa_zagAw
ChVd3EX1KZ7D8mQYiCbh-F5n2vvydzJ6UOqa4Ur-VzrGCaorB_zdSWoy8DxlHIcSL
iCoE24lf_hHdv25lCu9-wPRXAlr2f3aAPMd9Mq_iqpkSoonXjMdN16uhUxkC-dX70
2YVq2z9S5_Q_sWeYAgdg-9nVVdMQcY3I9mWu5HvbIo_z-lnTUP5mmRD4Otii67EpG
K02QooxPJb0u8UEKFp665ZMu0Snd6nCfBwlxeuVoUgoXlB6vPac_65c7AdyY3obnF
ai6qZ8wiwV8NFyyyRL3jUuT8mGbalAA"}}},
    "wmk": "PkGqGm7GBjOlsTgQbBdpccPNqoVq76b0OVxLweTerdzYyx5iRdIXXA"}],
  "Index": 0,
  "ContainerType": "List",
  "ContentMeta": {},
  "DataEncoding": "JSON"}
```

Figure 15

Frame 1

```
{
  "Index": 1,
  "ExchangePosition": 0}
```

Figure 16

Frame 2

```
{
  "Index": 2,
  "ExchangePosition": 0}
```

Figure 17

Here are the container bytes. Note that the content is now encrypted and has expanded by 25 bytes. These are the salt (16 bytes), the AES padding (4 bytes) and the JSON-B framing (5 bytes).

```
f5 03 0c
f1 03 04
  7b 0a 20 20 22 65 6e 63    22 3a 20 22 41 32 35 36
  43 42 43 22 2c 0a 20 20    22 72 65 63
  ...
  7d 2c 0a 20 20 22 44 61    74 61 45 6e 63 6f 64 69
  6e 67 22 3a 20 22 4a 53    4f 4e 22 7d
f0 03
  88 01 00
0c 03 f5
f5 01 72
f0 28
  7b 0a 20 20 22 49 6e 64    65 78 22 3a 20 31 2c 0a
  20 20 22 45 78 63 68 61    6e 67 65 50 6f 73 69 74
  69 6f 6e 22 3a 20 30 7d
f1 01 45
  88 10 a8 9e 8c f3 f7 d6    fa c1 b7 e7 a0 30 71 bc
  ae 7a 89 01 30 04 90 ce    1d 1a 79 f3 92 31 ae 2b
  a5 b3 13 29 75 9c 48 07    b1 9b f1 46 d3 95 d2 81
  3e 60 4f 76 16 fe 6d 2f    23 99 40 13 84 91 ca 1f
  53 30 ec fe 66 b4 e1 38    95 6c 64 9f f7 59 30 5b
  b7 d0 b5 1d 7d 29 77 61    21 e2 2e 29 30 66 ed 7a
  92 4f 63 d4 7f 9d 3c 7f    12 59 8e 3f d9 f2 5a 63
  8e ab 0a 3b 95 fa fc 16    1f b7 43 86 c5 f1 ef 6b
  d8 fc 8d 43 44 2f 55 4c    06 e4 19 08 2e ba 28 21
  af 04 88 e8 0e 5c 5f 25    7e 5a af 7a d5 44 aa de
  2e ed e2 cd 9c df 58 95    b2 d2 05 d8 64 90 0c 65
  b6 0b be 7b 86 99 90 e3    4d 73 f9 91 74 bf 07 7f
  29 0b 0e 9d 5f bf ce 60    90 4a fb ef 37 9c 62 85
  e8 01 2d 87 c8 a4 5c 07    ea dc c0 ec e4 10 d7 48
  ee 57 e2 35 aa c0 e7 65    5f 5a 16 b3 cb e0 96 6b
  ee 85 86 ff 71 d1 37 7d    65 81 6d 67 88 97 81 58
  5d 30 1d 6c 8d 1a 2f 08    b4 d5 d4 80 17 fe 22 54
  cd 4c 02 34 fa 4a e4 9c    c0 c9 2f a2 e0 3e 58 f6
  14 ef a8 53 9b f2 69 f1    d0 10 26 93 45 13 d8 b5
  86 57 2e 6c b5 fb 74 eb    ee b7 b5 a5 87 f6 ec b8
  2d 55 48 31 ac
72 01 f5
f5 01 72
f0 28
  7b 0a 20 20 22 49 6e 64    65 78 22 3a 20 32 2c 0a
  20 20 22 45 78 63 68 61    6e 67 65 50 6f 73 69 74
  69 6f 6e 22 3a 20 30 7d
f1 01 45
```

```

88 10 2d 8f 69 b8 89 72    a5 f0 65 76 9d b6 7f 52
bd b2 89 01 30 11 1d 1b    a4 ee 8d bc d7 2a 02 08
82 d2 3a 96 a3 36 b3 4a    ad 67 d5 31 4c 5e 12 b0
ac 16 d3 ee 9d 34 d2 0a    f7 70 6b e7 63 d2 63 21
8d 65 4f 60 13 78 51 11    e9 53 f6 e2 70 79 5d 8f
a9 66 8c 7f 0c 55 d1 8f    9a 4b e7 2c dc b5 85 9e
9c aa a8 2f ba 79 1f b6    c5 3b e8 2d 72 04 74 d8
f6 a6 ca 8e 67 68 d4 64    59 1c be 65 a2 51 02 22
af 94 bd fe ac 01 bc 65    23 e8 fb 4f bb 62 21 f7
77 7d e2 8a 4a a7 86 09    9e b0 b3 3c 44 fa 91 d0
95 22 e0 78 46 ee ab 8b    7c 84 83 c9 e2 67 ae 2e
f6 24 42 2b 42 3e 70 c6    e1 91 ec 12 83 c2 95 ff
a5 ea 09 70 8d 93 35 e8    1d 11 d6 90 6c d8 be 9e
bc d7 98 ad 7f c7 58 cc    e3 0d ce 51 d7 5d 05 5b
23 e8 42 ba ea 79 81 98    2a 9b a9 5d e8 2e c9 7e
14 d4 7a 7f 0f f8 aa 5c    2c b2 8c 0a f4 c6 b3 89
52 e4 c8 16 69 aa c2 a3    f8 6e 05 ff c4 8d 04 92
9e 83 c5 b2 8f 8a 1f 19    72 e2 f4 bc 8c 48 4f fc
75 a2 17 73 d5 8a 6e 22    9a b7 f8 a9 86 98 33 a4
4b 57 9a 60 59 a2 19 7f    e2 35 f1 e0 63 5d bf 25
c5 8f c6 90 f6
72 01 f5

```

Figure 18

The following example shows a container in which all the frame payloads are encrypted under separate key agreements specified in the payload frames.

Frame 0

```

{
  "Index": 0,
  "ContainerType": "List",
  "ContentMeta": {},
  "DataEncoding": "JSON"}

```

Figure 19

Frame 1

```
{
  "enc": "A256CBC",
  "recipients": [{
    "kid": "MCRW5-YRWQK-M4HE3-D4JE6-IYQLM-22ITA-A",
    "epk": {
      "PublicKeyDH": {
        "kid": "MCHK5-FYGDU-J3LE2-H427C-OPD6G-HD4QY-A",
        "Domain": "YE6bnq1MlX5ojaJto6PLP_PEWa",
        "Public": "kRBM44hYJidVQYULAtk8SHrYUMVjmBxLnPVwEwmi7tVK4xBipCAH
rx06dlbO-gtd9f6V984yEtgXs1GJibo4pCvwXyWq70CoPosc9Rymingo2c6jjLFQP
NvPks0e9kYSVLVTSM34XaTzQKjX4S9AkG4ziI4IIrENDPQN6gfRe0Px-NCB-tr3pCW
xJPPxpK2cYopHJX1jYamN69aF7r4ALx2jQSTiwv1558bU-atJNRqYv6dXVseiLpTo
TUVpwG838VGptjPlDeL_Tuk-di7uxj_n-EUVARr1N0HJwVcjJUHtrOt37jKtM5-RQ
5XKZ3WRvX23N0ae50WjZV-aHmWjvIw"}},
      "wmk": "iU--JG-ERFbIiIlpm9WYlOCg9K2yUEefcTFXWnzgkIqsEt84dqP_FA"}],
  "Index": 1}

```

Figure 20

Frame 2

```
{
  "enc": "A256CBC",
  "recipients": [{
    "kid": "MCRW5-YRWQK-M4HE3-D4JE6-IYQLM-22ITA-A",
    "epk": {
      "PublicKeyDH": {
        "kid": "MAK46-P3CXG-TWHTU-KX7AV-EKKBKZ-JIZW4-A",
        "Domain": "YE6bnq1MlX5ojaJto6PLP_PEWa",
        "Public": "HvdTPFfrFM9x6lzaXC6VqCX_QKxglKRCirshm4lYs9dmZgvTJTep
QLejdDzuT4IUUpCcdDq_lZloCbQVPtl-lHxm60UWYzeJeoBHcxpffXhJVEjFU7Ieet
vBAZP4IHnT_uhyxHomf6oHAo-VvtUa_nPRUpA4PyKApEL6bZPwwwU0WjoCfX25ezs
M8ODZDA2BF4-TQlVHmfzPHIIEZ0oscpcHEcjYBAOjoYrYxdyORUa56E0aXifW5Hb1
R00ZqXPB9dg3j-TEG6Bt1lMk1GSPe3zw339j-9bF19VLTAlhX5WHvZfDSHQdJev1T
bFV14t_tCaIg0jfkGcZSLzVNN1GwhQA"}},
      "wmk": "G69T62RWbpRGzv1IDAKSAB_ETERF3ZGHNx5Jqv-zj-3EPpgJsJZizA"}],
  "Index": 2}

```

Figure 21

12. Appendix B

```
public long PreviousFrame (long Frame) {
    long x2 = Frame + 1;
    long d = 1;

    while (x2 > 0) {
        if ((x2 & 1) == 1) {
            return x2 == 1 ? (d / 2) - 1 : Frame - d;
        }
        d = d * 2;
        x2 = x2 / 2;
    }
    return 0;
}
```

Figure 22

13. References

13.1. Normative References

- [draft-hallambaker-dare-message]
"[Reference Not Found!]".
- [draft-hallambaker-jsonbcd]
Hallam-Baker, P., "Binary Encodings for JavaScript Object Notation: JSON-B, JSON-C, JSON-D", draft-hallambaker-jsonbcd-12 (work in progress), May 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015.

13.2. Informative References

- [BLOCKCHAIN]
Chain.com, "Blockchain Specification".

- [draft-hallambaker-mesh-architecture]
Hallam-Baker, P., "Mathematical Mesh: Architecture",
draft-hallambaker-mesh-architecture-04 (work in progress),
September 2017.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70,
RFC 5652, DOI 10.17487/RFC5652, September 2009.
- [ZIPFILE] PKWARE Inc, "APPNOTE.TXT - .ZIP File Format
Specification", October 2014.

13.3. URIs

- [1] <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>
- [2] <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>
- [3] <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>
- [4] <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 28, 2019

P. Hallam-Baker
Comodo Group Inc.
August 27, 2018

Data At Rest Encryption Part 2: DARE Container
draft-hallambaker-dare-container-02

Abstract

This document describes DARE Container, a message and file syntax that allows an append-only sequence of data frames to be represented with cryptographic integrity, signature and encryption enhancements. The format supports data integrity checks using digest chains and Merkle trees. The simplest supports efficient write operations and efficient read operations in either the forward or reverse direction. Support for efficient random-access reads may be provided through the use of binary trees or index records appended to the end of the file.

This document is also available online at
<http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>
[1] .

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 February 28, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Container Format	3
1.2. Write	4
1.3. Read Access	5
1.4. Encryption and Authentication	5
1.5. Integrity and Signature	5
1.6. Redaction	6
1.7. Alternative approaches	6
1.8. Efficiency	7
2. Definitions	7
2.1. Related Specifications	7
2.2. Requirements Language	8
3. Container Navigation	8
3.1. Tree	9
3.2. Position Index	9
3.3. Metadata Index	10
4. Integrity Mechanisms	10
4.1. Digest Chain calculation	10
4.2. Binary Merkle tree calculation	10
5. Reference	11
5.1. Container Headers	11
5.1.1. Structure: ContainerHeaderFirst	11
5.1.2. Structure: ContainerHeader	11
5.2. Content Metadata Structure	12
5.2.1. Structure: ContentMeta	12
5.3. Index Structures	13
5.3.1. Structure: ContainerIndex	13
5.3.2. Structure: IndexPosition	13
5.3.3. Structure: KeyValue	13
5.3.4. Structure: IndexMeta	13
5.4. Signature	14
6. Security Considerations	14
7. IANA Considerations	14
8. Acknowledgements	14
9. Appendix A: Examples and Test Vectors	14
9.1. Simple container	15
9.2. Payload and chain digests	16
9.3. Merkle Tree	17

9.4. Signed container	19
9.5. Encrypted container	21
10. Appendix B	25
11. References	25
11.1. Normative References	25
11.2. Informative References	26
11.3. URIs	26
Author's Address	26

1. Introduction

DARE Container is a message and file syntax that allows a sequence of data frames to be represented with cryptographic integrity, signature, and encryption enhancements to be constructed in an append only format. DARE Container was developed in response to needs that arose out of the design of the Mathematical Mesh [draft-hallambaker-mesh-architecture] . It is built on the binary encodings of JSON data objects, JSON-B and JSON-C [draft-hallambaker-jsonbcd] and the DARE Message format [draft-hallambaker-dare-message] .

The format is designed to meet the requirements of a wide range of use cases including:

- o Recording transactions in persistent storage.
- o Synchronizing transaction logs between hosts.
- o File archive.
- o Message spool.
- o Signing and encrypting single data items.
- o Incremental encryption and authentication of server logs.

1.1. Container Format

A DARE Container consists of a sequence of variable length frames. Each frame consists of a forward length indicator, the framed data and a reverse length indicator. The reverse length indicator is written out backwards allowing the length and thus the frame to be read in the reverse direction:

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html> [2].]]

JBCD Bidirectional Frame

Each frame contains a single DARE Message consisting of a Header, Payload and Trailer (if required). The first frame in a container describes the container format options and defaults. These include the range of encoding options for frame metadata supported and the container profiles to which the container conforms.

All internal data formats are 64 bit clean allowing for containers of up to 18 exabytes to be written.

Five container types are currently specified but this may be reduced if the Digest and Tree types are withdrawn. These are:

Simple The container does not provide any index or content integrity checks.

Tree Frame headers contain entries that specify the start position of previous frames at the apex of the immediately enclosing binary tree. This enables efficient random access to any frame in the file.

Digest Each frame trailer contains a PayloadDigest field. Modification of the payload will cause verification of the PayloadDigest value to fail on that frame.

Chain Each frame trailer contains PayloadDigest and ChainDigest fields allowing modifications to the payload data to be detected. Modification of the payload will cause verification of the PayloadDigest value to fail on that frame and verification of the ChainDigest value to fail on all subsequent frames.

Merkle Tree Frame headers contain entries that specify the start position of previous frames at the apex of the immediately enclosing binary tree. Frame Trailers contain TreeDigestPartial and TreeDigestFinal entries forming a Merkle digest tree.

1.2. Write

In normal circumstances, DARE Containers are written as an append only log. As with DARE Messages, integrity information (payload digest, signatures) is written to the message trailer. Thus, large

payloads may be written without the need to buffer the payload data provided that the content length is known in advance.

1.3. Read Access

The use of reverse length indicators allows DARE containers to support efficient sequential access in either the forward or reverse directions.

Random access to any part of a file MAY be supported by means of a binary tree index and/or an index record providing direct access to any part of the file.

1.4. Encryption and Authentication

Frame payloads and associated attributes MAY be encrypted and/or authenticated using the approach described in [draft-hallambaker-dare-message] .

Incremental encryption is supported allowing encryption parameters from a single public key exchange operation to be applied to encrypt multiple frames. The public key exchange information is specified in the first encrypted frame and subsequent frames encrypted under those parameters specify the location at which the key exchange information is to be found by means of the ExchangePosition field.

The only restriction on the use of incremental encryption is that the frame containing the key exchange information MUST precede the frames that reference the exchange parameters.

To avoid cryptographic vulnerabilities resulting from key re-use, the DARE key exchange requires that each encrypted sequence use an encryption key and initialization vector derived from the master key established in the public key exchange by means of a unique salt.

Each DARE Message and by extension, each DARE Container frame MUST specify a unique salt value of at least 128 bits. Since the encryption key is derived from the salt value by means of a Key Derivation Function, erasure of the salt MAY be used as a means of rendering the payload plaintext value inaccessible without changing the payload value.

1.5. Integrity and Signature

Signatures MAY be applied to a payload digest, the final digest in a chain or tree. The chain and tree digest modes allow a single signature to be used to authenticate all frame payloads in a container.

The tree signature mode is particularly suited to applications such as file archives as it allows files to be verified individually without requiring the signer to sign each individually. Furthermore, in applications such as code signing, it allows a single signature to be used to verify both the integrity of the code and its membership of the distribution.

As with DARE Message, the signature mechanism does not specify the interpretation of the signature semantics. The presence of a signature demonstrates that the holder of the private key applied it to the specified digest value but not their motive for doing so. Describing such semantics is beyond the scope of this document and is deferred to future work.

1.6. Redaction

The chief disadvantage of using an append-only format is that containers only increase in size. In many applications, much of the data in the container becomes redundant or obsolete and a process analogous to garbage collection is required. This process is called redaction.

The simplest method of redaction is to create a new container and sequentially copy each entry from the old container to the new, discarding redundant frames and obsolete header information.

For example, partial index records may be consolidated into a single index record placed in the last frame of the container. Unnecessary signature and integrity data may be discarded and so on.

It is also possible but not necessarily advisable to perform such a redaction in-place provided that the redaction process does not increase the length of any individual frame and that appropriate provision is made for file locking to prevent conflicts and to provide for safe resumption should an interruption occur during the process.

1.7. Alternative approaches

Many file proprietary formats are in use that support some or all of these capabilities but only a handful have public, let alone open, standards. DARE Container is designed to provide a superset of the capabilities of existing message and file syntaxes, including:

- o Cryptographic Message Syntax [RFC5652] defines a syntax used to digitally sign, digest, authenticate, or encrypt arbitrary message content.

- o The.ZIP File Format specification [ZIPFILE] developed by Phil Katz.
- o The BitCoin Block chain [BLOCKCHAIN] .
- o JSON Web Encryption and JSON Web Signature

Attempting to make use of these specifications in a layered fashion would require at least three separate encoders and introduce unnecessary complexity. Furthermore, there is considerable overlap between the specifications providing multiple means of achieving the same ends, all of which must be supported if decoders are to work reliably.

1.8. Efficiency

Every data format represents a compromise between different concerns, in particular:

Compactness The space required to record data in the encoding.

Memory Overhead The additional volatile storage (RAM) required to maintain indexes etc. to support efficient retrieval operations.

Number of Operations The number of operations required to retrieve data from or append data to an existing encoded sequence.

Number of Disk Seek Operations Optimizing the response time of magnetic storage media to random access read requests has traditionally been one of the central concerns of database design. The DARE Container format is designed to the assumption that this will cease to be a concern as solid state media replaces magnetic.

While the cost of storage of all types has declined rapidly over the past decades, so has the amount of data to be stored. DARE Container represents a pragmatic balance of these considerations for current technology. In particular, since payload volumes are likely to be very large, memory and operational efficiency are considered higher priorities than compactness.

2. Definitions

2.1. Related Specifications

DARE Container makes use of the following related standards and specifications.

Encoding Content frame headers are encoded using JavaScript Object Notation (JSON) [RFC7159] , JSON-B or JSON-C [draft-hallambaker-jsonbcd] .

Cryptography The encryption and signature schemes used are based on JSON Web Signature [RFC7515] and JSON Web Encryption [RFC7516] as applied in DARE Message [draft-hallambaker-dare-message] .

2.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119] .

3. Container Navigation

Three means of locating frames in a container are supported:

Sequential Access frames sequentially starting from the start or the end of the container.

Binary search Access any container frame by frame number in $O(\log_2(n))$ time by means of a binary tree constructed while the container is written.

Index Access and container frame by frame number or by key by means of an index record.

All DARE Containers support sequential access. Only tree and Merkle tree containers support binary search access. An index frame MAY be written appended to any container and provides $O(1)$ access to any frame listed in the index.

Two modes of compilation are considered:

Monolithic Frames are added to the container in a single operation, e.g. file archives,

Incremental Additional frames are written to the container at various intervals after it was originally created, e.g. server logs, message spools.

In the monolithic mode, navigation requirements are best met by writing an index frame to the end of the container when it is complete. It is not necessary to construct a binary search tree unless a Merkle tree integrity check is required.

In the incremental mode, Binary search provides an efficient means of locating frames by frame number but not by key. Writing a complete index to the container every m write operations provides $O(m)$ search access but requires $O(n^2)$ storage.

Use of partial indexes provides a better compromise between speed and efficiency. A partial index is written out every m frames where m is a power of two. A complete index is written every time a binary tree apex record is written. This approach provides for $O(\log_2(n))$ search with incremental compilation with approximately double the overhead of the monolithic case.

3.1. Tree

Binary search is supported by means of the `TreePosition` parameter specified in the `FrameHeader`. This parameter specifies the value of the immediately preceding apex.

Calculation of the immediately preceding apex is most easily described by representing the array index in binary with base of 1 (rather than 0). An array index that is a power of 2 (2, 4, 8, 16, etc.) will be the apex of a complete tree. Every other array index has the value of the sum of a set of powers of 2 and the immediately preceding apex will be the value of the next smallest power of 2 in the sum.

For example, to find the immediately preceding apex for frame 5, we add 1 to get 6. $6 = 4 + 2$, so we ignore the 2 and the preceding frame is 4.

The values of Tree Position are shown for the first 8 frames in figure xx below:

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html> [3].]]

Merkle Tree Integrity check

An algorithm for efficiently calculating the immediately preceding apex is provided in Appendix C.

3.2. Position Index

Contains a table of frame number, position pairs pointing to prior locations in the file.

3.3. Metadata Index

Contains a list of IndexMeta entries. Each entry contains a metadata description and a list of frame indexes (not positions) of frames that match the description.

4. Integrity Mechanisms

Frame sequences in a DARE container MAY be protected against a frame insertion attack by means of a digest chain, a binary Merkle tree or both.

4.1. Digest Chain calculation

A digest chain is simple to implement but can only be verified if the full chain of values is known. Appending a frame to the chain has $O(1)$ complexity but verification has $O(n)$ complexity:

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html> [4].]]

Hash chain integrity check

The value of the chain digest for the first frame (frame 0) is $H(H(\text{null})+H(\text{Payload}_0))$, where null is a zero length octet sequence and payload_n is the sequence of payload data bytes for frame n

The value of the chain digest for frame n is $H(H(\text{Payload}_{n-1})+H(\text{Payload}_n))$, where A+B stands for concatenation of the byte sequences A and B.

4.2. Binary Merkle tree calculation

The tree index mechanism describe earlier may be used to implement a binary Merkle tree. The value TreeDigest specifies the apex value of the tree for that node.

Appending a frame to the chain has $O(\log_2 n)$ complexity provided that the container format supports at least the binary tree index. Verifying a chain has $O(\log_2 n)$ complexity, provided that the set of necessary digest inputs is known.

To calculate the value of the tree digest for a node, we first calculate the values of all the sub trees that have their apex at that node and then calculate the digest of that value and the immediately preceding local apex.

5. Reference

TBS stuff

5.1. Container Headers

TBS stuff

5.1.1. Structure: ContainerHeaderFirst

Inherits: ContainerHeader

Inherits: ContainerHeader

DataEncoding: String (Optional) Specifies the data encoding for the header section of for the following frames. This value is ONLY valid in Frame 0 which MUST have a header encoded in JSON.

5.1.2. Structure: ContainerHeader

Inherits: DAREHeader

Describes a container header. A container header MAY contain any DARE Message header.

Index: Integer (Optional) The record index within the file. This MUST be unique and satisfy any additional requirements determined by the ContainerType.

ContainerType: String (Optional) Specifies the container type for the following records.

IsMeta: Boolean (Optional) If true, the current frame is a meta frame and does not contain a payload.

Note: Meta frames MAY be present in any container. Applications MUST accept containers that contain meta frames at any position in the file. Applications MUST NOT interpret a meta frame as a data frame with an empty payload.

UniqueID: String (Optional) Unique object identifier

ContentMeta: ContentMeta (Optional) Content meta data.

TreePosition: Integer (Optional) Position of the frame containing the apex of the preceding sub-tree.

IndexPosition: Integer (Optional) Specifies the position in the file at which the last index entry is to be found

ExchangePosition: Integer (Optional) Specifies the position in the file at which the key exchange data is to be found

ContainerIndex: ContainerIndex (Optional) An index of records in the current container up to but not including this one.

PayloadDigest: Binary (Optional) If present, contains the digest of the Payload.

ChainDigest: Binary (Optional) If present, contains the digest of the PayloadDigest values of this frame and the frame immediately preceding.

TreeDigest: Binary (Optional) If present, contains the Binary Merkle Tree digest value.

Event: String (Optional) Unique object identifier

Labels: String [0..Many] List of labels that are applied to the payload of the frame.

KeyValues: KeyValue [0..Many] List of key/value pairs describing the payload of the frame.

First: Integer (Optional) Frame number of the first object instance value.

Previous: Integer (Optional) Frame number of the immediately prior object instance value

5.2. Content Metadata Structure

TBS stuff

5.2.1. Structure: ContentMeta

Information describing the object instance

ContentType: String (Optional) The content type field as specified in JWE

Paths: String [0..Many] List of filename paths for the payload of the frame.

UniqueID: String (Optional) Unique object identifier

Created: DateTime (Optional) Initial creation date.

Modified: DateTime (Optional) Date of last modification.

5.3. Index Structures

TBS stuff

5.3.1. Structure: ContainerIndex

A container index

Full: Boolean (Optional) If true, the index is complete and contains position entries for all the frames in the file. If absent or false, the index is incremental and only contains position entries for records added since the last frame containing a ContainerIndex.

Positions: IndexPosition [0..Many] List of container position entries

Metas: IndexMeta [0..Many] List of container position entries

5.3.2. Structure: IndexPosition

Specifies the position in a file at which a specified record index is found

Index: Integer (Optional) The record index within the file.

Position: Integer (Optional) The record position within the file relative to the index base.

5.3.3. Structure: KeyValue

Specifies a key/value entry

Key: String (Optional) The key

Value: String (Optional) The value corresponding to the key

5.3.4. Structure: IndexMeta

Specifies the list of index entries at which a record with the specified metadata occurs.

Index: Integer [0..Many] List of record indices within the file where frames matching the specified criteria are found.

ContentType: String (Optional) Content type parameter

Paths: String [0..Many] List of filename paths for the current frame.

Labels: String [0..Many] List of labels that are applied to the current frame.

5.4. Signature

Payload data MAY be signed using a JWS [RFC7515] as applied in the DARE Message format [draft-hallambaker-dare-message] .

Signatures are specified by the Signatures parameter in the content header. The data that the signature is calculated over is defined by the typ parameter of the Signature as follows.

Payload The value of the PayloadDigest parameter

Chain The value of the ChainDigest parameter

Tree The value of the TreeDigestFinal parameter

If the typ parameter is absent, the value Payload is implied.

A frame MAY contain multiple signatures created with the same signing key and different typ values.

The use of signatures over chain and tree digest values permit multiple frames to be validated using a single signature verification operation.

6. Security Considerations

7. IANA Considerations

8. Acknowledgements

9. Appendix A: Examples and Test Vectors

The data payloads in all the following examples are identical, only the authentication and/or encryption is different.

- o Frame 1..n consists of 300 bytes being the byte sequence 00, 01, 02, etc. repeating after 256 bytes.

For conciseness, the raw data format is omitted for examples after the first, except where the data payload has been transformed, (i.e. encrypted).

9.1. Simple container

the following example shows a simple container with first frame and a single data frame:

```
f4 5d
f0 59
f0 00
5d f4
f5 01 40
f0 0f
f1 01 2c
40 01 f5
```

Figure 1

Since there is no integrity check, there is no need for trailer entries. The header values are:

Frame 0

```
{
  "Index":0,
  "ContainerType":"List",
  "ContentMeta":{},
  "DataEncoding":"JSON"}
```

[Empty trailer]

Figure 2

Frame 1

```
{
  "Index":1}
```

[Empty trailer]

Figure 3

9.2. Payload and chain digests

The following example shows a chain container with a first frame and three data frames. The headers of these frames is the same as before but the frames now have trailers specifying the PayloadDigest and ChainDigest values:

Frame 0

```
{
  "Index":0,
  "ContainerType":"Chain",
  "ContentMeta":{},
  "DataEncoding":"JSON"}
```

[Empty trailer]

Figure 4

Frame 1

```
{
  "Index":1}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "ChainDigest":"T7S1FcrgY3AaWD4L-t5WlK-3XYkPTcOdGEGyjglTD6yMYVRVz9
tn_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}
```

Figure 5

Frame 2

```
{
  "Index":2}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "ChainDigest":"T7S1FcrgY3AaWD4L-t5WlK-3XYkPTcOdGEGyjglTD6yMYVRVz9
tn_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}
```

Figure 6

Frame 3

```
{
  "Index":3}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "ChainDigest":"T7S1FcrgY3AaWD4L-t5W1K-3XYkPTcOdGEGyjjglTD6yMYVRVz9
tn_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}
```

Figure 7

9.3. Merkle Tree

The following example shows a chain container with a first frame and six data frames. The trailers now contain the TreePosition and TreeDigest values:

Frame 0

```
{
  "Index":0,
  "ContainerType":"Merkle",
  "ContentMeta":{},
  "DataEncoding":"JSON"}
```

[Empty trailer]

Figure 8

Frame 1

```
{
  "Index":1,
  "TreePosition":0}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest":"T7S1FcrgY3AaWD4L-t5W1K-3XYkPTcOdGEGyjjglTD6yMYVRVz9t
n_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}
```

Figure 9

Frame 2

```
{
  "Index":2,
  "TreePosition":319}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest":"7fHmkEIsPkN6sDYAOLvpIJn5Dg3PxDDAaq-1l2kh8722kokkFnZ
QcYtjuVC71aHNXI18q-lPnfRkmwryG-bhqQ"}
```

Figure 10

Frame 3

```
{
  "Index":3,
  "TreePosition":319}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest":"T7S1FcrgY3AaWD4L-t5WlK-3XYkPTcOdGEGyjglTD6yMYVRVz9t
n_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}
```

Figure 11

Frame 4

```
{
  "Index":4,
  "TreePosition":1451}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest":"vJ6ngNATvZcXSMALi5IUqz1lGBxBnTNVcC87VL_BhMRCbAvKSj8
gs0VFGxxLkZ2myrtaDIwhHoswiTiBMLNWug"}
```

Figure 12

Frame 5

```
{
  "Index":5,
  "TreePosition":1451}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest":"T7SlFcrgY3AaWD4L-t5WlK-3XYkPTcOdGEGyjglTD6yMYVRVz9t
n_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}
```

Figure 13

Frame 6

```
{
  "Index":6,
  "TreePosition":2586}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest":"WgHlz3EHczVPqgtpc39Arv7CFIsCbFVsk8wg0j2qLlEfur9SZ0m
dr65Ka-HF0Qx8gg_DAOiJwUrwADDXyxVJOg"}
```

Figure 14

9.4. Signed container

The following example shows a tree container with a signature in the final record. The signing key parameters are:

```
{
  "PrivateKeyRSA": {
    "kid": "MBWNO-2J43U-ESWKW-XQWL6-6YGEW-UOPWU-A",
    "n": "1NzbnmakMalVH1mRv7TEDEhwXDNojn5wZbqltv1gp5PgZwzX-klYXuFhj0-
MpO0zcwptsUaYJhwdvvgW_ludUpISQYluXOB3UMj2e_0yl64MvnqTL47SZQuAN3QQ
9cuCw_-_Eyj_jerspauqa6RpNzGcabZrtRl7J7DPVZ3SNlw-H_Wxd4HkrFVW_Yqup
htNL1JciQJYm2DSu9dbetqPZ80x6IBargY850mBYOzvNNE5S-dRJQH0JY4SG-ESYF
BuAhtBlOMgbIOXNiQ96EegA-vPW9XRF-SHdX5mkafefDGK4rT_RoE4grWhDM3jBz8
1-F2ZA_GpNVEvB-25_vF96lQ",
    "e": "AQAB",
    "d": "k_v_h7Jo-TvUt44X6jSax-pTdBHrljklzSYxGEE4yIBbmMVe-Gl2ECkTLe
nNbnafo4RGJ_Vgxkk7PEZO-p7Uz5OBtX-rf83tCgihEyg8aaFIZ-hl_xY9Pqr5uGA
MQGNJaoVMsLb99QNNZhE4JTquP56mVvDQaI3Zn6bhhA0ZqpxS_x6iRUV5KnHCRd47
DKGHcAsQR_caxGec7M_XNPqpl0tcoGQz46-I0SVVcqtjb_YysEvez4eJhb_ZTU4C7
pz4wXDj6B0ppFJVZEMaIhKo8FCnoQdXEJl4vSiBzUFRSlPc6gjQk2CBhxc_kb782w
VwW5wtgOVxhVlk2piQ2NrloQ",
    "p": "6osXfrJLiPfk0ky17vqzRs-M5mOxZU2LEFGyHTXxx6EYTWixEsx2Sdf6kx
UD5K_QdYnqgd3yobbGdDpMwwEuwgogWVCz90nc9QdCUy4MCpz8lpOQdli_tXMmtOg
GBu9mapMTEOwc7HlLlSDRezV_TzTAH4izv-CUEZ_M5EcwyFM",
    "q": "6FYD6NV4rKU1ACTGQyIvWGrkrS_F6phB-whx0nSVFkbbkwpJiPnC6XqjZ3
OYPCZylxaTHFnxCs0nntrraeNEcWfNPrpTN5XIZjbOiIaKA2iCQkWlDoi8oduEtTK
oIcuy32oBz6MpUeCWz10ZQ4EeTF3RyCc_jpf8oRvZ0e3ItHc",
    "dp": "hfjbe9BmWx-HqCaPSanEW-9UQYmym_X2OGUiA5N7vxcI5ZymgOFvs_B9v
iQj7C4NOgaEl3EjFgJsS5m9nSoAxm-4WKxDkD6NyxxzRYugLkshnc69otvNnlkKnWn
CqeK2o57mJC4KDZwRGczIKloTH6jtsfta8Lh8fFQ4doEuV7uc",
    "dq": "r6R_ViE0FoJalaLhflU09mmZMViBbkXm86nBqthZ97pmrLvJRdVTxgCh0
c6w0yBZluEJHBDeykSoZE6qVCWtE3Le1kI0MTx6ANQENXBInCUA_Kr8Ck3TFSYIYJ
fIRaxiMMZKUjfoQAji2WXGeKL_TcpLkt4hDWLXaNDOTgdOiSc",
    "qi": "DfhtLB1Ox1Kgp3E4jqy5Qxeb7-v7_uv8n_5-E1OQ3NLSRV2m_auojkR19
nY3gokHKNSXM41qKlJLU001ROjOO2KUq57s8GZkheVfbJLNCJ6KAw_aRT2IgyJm2b
e2v5OCHSkm88tgJWbtKj-OPKTFV5gOMVdeCzGX286ErjDHGCM" } }
```

Figure 15

The container headers and trailers are:

Frame 0

```
{
  "Index": 0,
  "ContainerType": "Merkle",
  "ContentMeta": { },
  "DataEncoding": "JSON" }
```

[Empty trailer]

Figure 16

Frame 1

```
{
  "Index":1,
  "TreePosition":0}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest":"T7S1FcrgY3AaWD4L-t5WlK-3XYkPTcOdGEGyjglTD6yMYVRVz9t
n_KQc6GdA-P4VSRigBygV65OEd2Vv3YDhww"}
```

Figure 17

Frame 2

```
{
  "dig":"S512",
  "Index":2,
  "TreePosition":319}

{
  "PayloadDigest":"8dyi62d7MDJlsLm6_w4GEgKBjzXBRwppu6qbtmAl6UjZDlZe
aWQlBsYhOu88-ekpNXpZ2iY96zTRI229zaJ5sw",
  "TreeDigest":"7fHmkEIsPkN6sDYAOLvpIJn5Dg3PxDDAaq-1l2kh8722kokkFnZ
QcYtjuVC71aHNXI18q-lPnfRkmwryG-bhqQ"}
```

Figure 18

9.5. Encrypted container

The following example shows a container in which all the frame payloads are encrypted under the same master secret established in a key agreement specified in the first frame.

Frame 0

```

{
  "enc": "A256CBC",
  "Salt": "Th7V_b2--KdV5rETQZbHsw",
  "recipients": [{
    "kid": "MDAD3-E4BYE-MK6CH-QA2HD-TKRS2-KIX5Y-A",
    "epk": {
      "PublicKeyDH": {
        "kid": "MANS7-VSJ2X-75NSK-4SEOM-FRITG-6Q7EO-A",
        "Domain": "YE6bnqlMlX5ojaJto6PLP_PEWa",
        "Public": "MYw__h5Ii4honcLOGFKY42KdvJed5T47CZI3QTRSBXUPiOD
0D0W49YIyf-m30K-BvMWVzNsh9qAQFpK53A5UJO_pQmfbmrZLHq3AjdYoRlc44YOn
usUokZN43xtGly4m7Rcf_42M3cofa80UWejL3mz3zQKTojxwXz3cykLA2tsTjHdKE
00Q6jWef952YkPQNKJmR-42aX7Ppz5kZt1QtgfuV4QYk_b0NFfHSNk9CAs6Klav5K
qGjxL4mfIxvmrksyIR5solCo2BMbh_AOdyZNuEa4nURvS9YtWrXZRjQtMZPKB8MXG
_mSaxUco3Ulem2hmdkgIkTveF7-da6otILw"}},
      "wmk": "0HfYk15gAvxngunLoHymHSunrb0KlGjGuTERlpSxB5JLpFJj6KJZZA"
    }
  ],
  "Index": 0,
  "ContainerType": "List",
  "ContentMeta": {},
  "DataEncoding": "JSON"
}

[Empty trailer]

```

Figure 19

Frame 1

```

{
  "enc": "A256CBC",
  "Salt": "BmY8ZYeD8KglSnIrBIAUYa",
  "Index": 1}

[Empty trailer]

```

Figure 20

Frame 2

```

{
  "enc": "A256CBC",
  "Salt": "pA-kigxfhhsGC5lM9xrAZQ",
  "Index": 2}

[Empty trailer]

```

Figure 21

Here are the container bytes. Note that the content is now encrypted and has expanded by 25 bytes. These are the salt (16 bytes), the AES padding (4 bytes) and the JSON-B framing (5 bytes).

```
f5 03 3c
f1 03 27
f0 10
3c 03 f5
f5 01 7c
f0 47
f1 01 30
7c 01 f5
f5 01 7c
f0 47
f1 01 30
7c 01 f5
```

Figure 22

The following example shows a container in which all the frame payloads are encrypted under separate key agreements specified in the payload frames.

Frame 0

```
{
  "Index":0,
  "ContainerType":"List",
  "ContentMeta":{},
  "DataEncoding":"JSON"}
```

[Empty trailer]

Figure 23

Frame 1


```
{
  "enc": "A256CBC",
  "Salt": "LUAkC3V-ztXJfF7OS-RNwQ",
  "recipients": [{
    "kid": "MDAD3-E4BYE-MK6CH-QA2HD-TKRS2-KIX5Y-A",
    "epk": {
      "PublicKeyDH": {
        "kid": "MCYIZ-PH7YZ-XAY6Q-2EMQR-GMRXG-66YPK-A",
        "Domain": "YE6bnq1MlX5ojaJto6PLP_PEWa",
        "Public": "4cDrUAlsZpyGZGhgVbLlunEyl709a8ku-6IGVZaSgyOmsb5
rCt_pjnN49fjy7hqsS364S4eUDRj7e3vfIBeav-zeynaiJB6lY6eJBMmlaEuomyyy
0lSrq5eaX6i0A8CRWuuRK8IJAHQ31Q89Bah5XayMxc3bqPxcioLlgqU75w65fKmY9
4mRdAIw2Yk-PTcQVztVDk2ooGJuvLEnuIk29g-leFbMmCAXND9hhDerVupBND3Ljh
BXylTvfIx-mrw0SMPO_IsVLbhrXQ6lTMygucttT3GPf25lgWBaEdiHVuPqmTBtagB
Vgq9-gHySYtuCNRk7RfB5rtz6CUmENEwx1QA" } },
    "wmk": "25yqAeqIG1S6BrCSQxU_M8WKB4t8J_lHWI-68DNYORTUoCpaitosig"
  ] },
  "Index": 1
}
```

[Empty trailer]

Figure 24

Frame 2

```
{
  "enc": "A256CBC",
  "Salt": "EVrEz8ai_v9s9hZ0ZXMB1Q",
  "recipients": [{
    "kid": "MDAD3-E4BYE-MK6CH-QA2HD-TKRS2-KIX5Y-A",
    "epk": {
      "PublicKeyDH": {
        "kid": "MB43P-C7BVX-SROQ6-Z5LW5-5DLRO-PS675-A",
        "Domain": "YE6bnq1MlX5ojaJto6PLP_PEWa",
        "Public": "mJRyK45fAf2ksxW7M8W961hCKIlg7dQqn9rA2K6vdShlo4v
gLqjHDzLYqr9yQKteGUfl0PLIuHAOFq3p_oJDhaioJKX6Akv-y-dcfDdi2WgOOr_k
R9NPtl-nmsbJ8fx6v1QdXLzdoGZjsj_t6wyIo5w287p7DylFdYkhriqVv7tgBELZI
53Pur4F5QbK-qJYeSwY3fAksQYrviMZbFVwic8v5QQNoAGcoGguBOOz-aljFuol9
0o7pNT-Soj3DAPfmXS1hmPR-hpqyzELAat-q_O22P4d_NtUOmaLwZXSZVYUBLE8Fj
1okt4_8zGfCsGpsHl9rwqlyoVIZ4VadldWw" } },
    "wmk": "P6UjESsvq5bATPHc5vPfEcs5ulpCkLVBCv5_aYdHdwQG3rpPihHj2g"
  ] },
  "Index": 2
}
```

[Empty trailer]

Figure 25

10. Appendix B

```
public long PreviousFrame (long Frame) {
    long x2 = Frame + 1;
    long d = 1;

    while (x2 > 0) {
        if ((x2 & 1) == 1) {
            return x2 == 1 ? (d / 2) - 1 : Frame - d;
        }
        d = d * 2;
        x2 = x2 / 2;
    }
    return 0;
}
```

Figure 26

11. References

11.1. Normative References

- [draft-hallambaker-dare-message]
Hallam-Baker, P., "Data At Rest Encryption Part 1: DARE Message", draft-hallambaker-dare-message-01 (work in progress), July 2018.
- [draft-hallambaker-jsonbcd]
Hallam-Baker, P., "Binary Encodings for JavaScript Object Notation: JSON-B, JSON-C, JSON-D", draft-hallambaker-jsonbcd-13 (work in progress), July 2018.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015.

11.2. Informative References

- [BLOCKCHAIN] Chain.com, "Blockchain Specification".
- [draft-hallambaker-mesh-architecture] Hallam-Baker, P., "Mathematical Mesh: Architecture", draft-hallambaker-mesh-architecture-04 (work in progress), September 2017.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009.
- [ZIPFILE] PKWARE Inc, "APPNOTE.TXT - .ZIP File Format Specification", October 2014.

11.3. URIs

- [1] <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>
- [2] <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>
- [3] <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>
- [4] <http://mathmesh.com/Documents/draft-hallambaker-dare-container.html>

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 25, 2018

P. Hallam-Baker
Comodo Group Inc.
May 24, 2018

Data At Rest Encryption Part 1: DARE Message
draft-hallambaker-dare-message-00

Abstract

This document describes the Data At Rest Encryption (DARE) message syntax. This syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary message content.

This document is also available online at
<http://mathmesh.com/Documents/draft-hallambaker-dare-message.html> [1]

.

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 November 25, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Existing approaches	4
1.2. The DARE Approach	4
2. Definitions	5
2.1. Related Specifications	5
2.2. Requirements Language	5
2.3. Defined terms	6
3. Applications	7
3.1. Message Types	7
3.1.1. Consolidated Messages	7
3.1.2. Related Messages	7
3.1.3. Detached Messages	7
3.1.4. Annotated Messages	8
3.1.5. Proxy Re-encryption	8
3.2. Additional Use Cases	8
3.2.1. Streaming data	8
3.2.2. Information Erasure by Key Deletion	8
3.2.3. Field level encryption	9
4. Message Format	9
4.1. Encodings	9
4.1.1. JSON	9
4.1.2. JSON-B	10
4.1.3. Application Directed Encoding	10
5. Processing Model	10
5.1. Consolidated Message Generation	11
5.1.1. Message Header	11
5.1.2. Enhanced Data Sequence	13
5.1.3. Trailer	15
5.2. Consolidated Message Recovery	15
5.2.1. Verify signers	15
5.2.2. Match recipient info	15
5.2.3. Recover Master Key	15
5.2.4. Process Enhanced Data Sequences	16
5.2.5. Signature validation	16
5.3. Detached Messages	16
5.4. Cloaked Messages	16
6. Algorithms	17
6.1. Field: kwd	17
7. Reference	17
7.1. Message Classes	17
7.1.1. Structure: DAREMessageSequence	17
7.2. Header and Trailer Classes	18
7.2.1. Structure: DARETrailer	18

7.2.2. Structure: DAREHeader	18
7.3. Cryptographic Data	19
7.3.1. Structure: DARESigner	19
7.3.2. Structure: X509Certificate	19
7.3.3. Structure: DARESignature	20
7.3.4. Structure: DARERecipient	20
8. Security Considerations	20
8.1. Encryption/Signature nesting	20
8.2. Side channel	20
8.3. Salt reuse	20
9. IANA Considerations	20
10. Acknowledgements	20
11. Test Examples	20
11.1. Plaintext Message	21
11.2. Plaintext Message with EDS	22
11.3. Encrypted Message	22
11.4. Signed Messages	24
12. References	24
12.1. Normative References	24
12.2. URIs	25
Author's Address	25

1. Introduction

This document describes the Data At Rest Encryption (DARE) message syntax. This syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary message content.

The DARE Message syntax is based on a subset of the JSON Web Signature [RFC7515] and JSON Web Encryption [RFC7516] standards and shares many fields and semantics. The processing model and data structures have been simplified to remove as many redundant features and alternative means of specifying the same content.

An important innovation in the DARE message format is the separation of key exchange and data encryption operations so that a Master Key (MK) established in a single exchange to be applied to multiple octet sequences. This means that a public key operation may be used to encrypt multiple parts of the same message or to multiple messages.

To maintain the security of the cryptographic algorithm, each octet sequence is encrypted under a different encryption key (and IV if required) derived from the Master Key by means of a salt. Depending on application needs, the salt may be explicit or implicit. An explicit salt is an opaque sequence of octets prepended to the data item. An implicit salt is an octet sequence constructed by application specific means such as the sequence number of a message or the byte position of a field in a file.

1.1. Existing approaches

Traditional cryptographic containers describe the application of a single key exchange to a single octet sequence. Examples include PKCS#7/CMS [RFC2315] , OpenPGP [RFC4880] and JSON Web Encryption [RFC7516] .

To encrypt a message using RSA, the creator first generates a random encryption key and initialization vector (IV). The encryption key is encrypted under the public key of each recipient to create a per-recipient decryption entry. The encryption key, plaintext and IV are used to generate the ciphertext (figure 1).

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html> [2].]]

Monolithic Key Exchange and Encrypt

This approach is adequate for the task of encrypting a single octet stream. It is less than satisfactory when encrypting multiple octet streams or very long streams for which a rekeying operation is desirable.

1.2. The DARE Approach

The DARE key exchange begins with the same key exchange used to produce the CEK in JWE but instead of using the CEK to encipher data directly, it is used as one of the inputs to a Key Derivation Function (KDF) that is used to derive parameters for each block of data to be encrypted. To avoid the need to introduce additional terminology, the term 'CEK' is still used to describe the output of the key agreement algorithm (including key unwrapping if required) but it is more appropriately described as a Master Key (figure 2).

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html> [3].]]

Exchange of Master Key

A Master Key may be used to encrypt any number of data items. Each data item is encrypted under a different encryption key and IV (if required). This data is derived from the Master Key using the HKDF function [RFC5869] using a different salt for each data item and separate info tags for each cryptographic function (figure 3).

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html> [4].]]

Data item encryption under Master Key and per-item salt.

This approach to encryption offers considerably greater flexibility allowing the same format for data item encryption to be applied at the transport, message or field level.

2. Definitions

2.1. Related Specifications

The DARE message format is based on the following existing standards and specifications.

Object serialization The JSON-B [draft-hallambaker-jsonbcd] encoding is used for object serialization. This encoding is an extension of the JavaScript Object Notation (JSON) [RFC7159] .

Message syntax The cryptographic processing model is based on JSON Web Signature (JWS) [RFC7515] , JSON Web Encryption (JWE) [RFC7516] and JSON Web Key (JWK) [RFC7517] .

Cryptographic primitives. The HMAC-based Extract-and-Expand Key Derivation Function [RFC5869] and Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm [RFC3394] are used.

The Uniform Data Fingerprint method of presenting data digests is used for key identifiers and other purposes [draft-hallambaker-udf] .

Cryptographic algorithms The cryptographic algorithms and identifiers described in JSON Web Algorithms (JWA) [RFC7518] are used together with additional algorithms as defined in the JSON Object Signing and Encryption IANA registry [IANAJOSE] .

2.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] .

2.3. Defined terms

The terms "Authentication Tag", "Content Encryption Key", "Key Management Mode", "Key Encryption", "Direct Key Agreement", "Key Agreement with Key Wrapping" and "Direct Encryption" are defined in the JWE specification [RFC7516] .

The terms "Authentication", "Ciphertext", "Digital Signature", "Encryption", "Initialization Vector (IV)", "Message Authentication Code (MAC)", "Plaintext" and "Salt" are defined by the Internet Security Glossary, Version 2 [RFC4949] .

Annotated Message A DARE Message that contains an edss field with at least one entry.

Authentication Data A Message Authentication Code or authentication tag.

Buffered Generation Mode A mode of generating a DARE message in which the data input is read completely before beginning output.

Consolidated Message A DARE message that contains the key exchange information necessary for the intended recipient(s) to decrypt it.

Detached Message A DARE message that does not contain the key exchange information necessary for the intended recipient(s) to decrypt it.

Encryption Context The master key, encryption algorithms and associated parameters used to generate a set of one or more enhanced data sequences.

Enhanced data sequence (EDS) A sequence consisting of a salt, content data and authentication data (if required by the encryption context).

Enhancement Applying a cryptographic operation to a data sequence. This includes encryption, authentication and both at the same time.

Generator The party that generates a DARE message.

Group Encryption Key A key used to encrypt data to be read by a group of users. This is typically achieved by means of some form of proxy re-encryption or distributed key generation.

Group Encryption Key Identifier A key identifier for a group encryption key.

Master Key (MK) The master secret from which keys are derived for authenticating enhanced data sequences.

Recipient Any party that receives and processes at least some part of a DARE message.

Related Message A set of DARE messages that share the same key exchange information and hence the same Master Key.

Unbuffered Streaming Mode. A mode of generating a DARE message in which data MAY be output before the input is read.

Uniform Data Fingerprint (UDF) The means of presenting the result of a cryptographic digest function over a data sequence and content type identifier specified in the Uniform Data Fingerprint specification [draft-hallambaker-udf]

3. Applications

3.1. Message Types

3.1.1. Consolidated Messages

A consolidated DARE message contains the ciphertext and authentication data for a data object together with the key exchange information necessary for the intended recipient(s) to process it.

Consolidated messages provide the same functionality as traditional PKJCS#7/CMS and OpenPGP message formats and may be used as a one-for-one replacement.

3.1.2. Related Messages

A set of DARE messages are related if they share the same key exchange information.

Related messages allow a single key exchange to be amortized over a collection of data items. This is particularly useful when a large collection of short data items is required such as in a server log or a chat-room transcript.

3.1.3. Detached Messages

A detached DARE message contains only the ciphertext and authentication data for a data object and does not provide any key exchange information.

The use of detached messages in a protocol allows key exchange information and message data to be passed to a recipient separately, possibly over different channels or even with entirely different partners.

For example, a service returning the last hundred entries from a log file encrypted as a series of DARE related messages need only provide the key exchange data once.

3.1.4. Annotated Messages

An annotated DARE message contains a message header with encrypted metadata in addition to an encrypted body. The use of annotated messages allows a receiver to process message data and metadata separately.

For example, a mail application typically provides users with a display showing a short summary of the messages received (sender, date, subject, etc.). Encrypting the metadata to be shown to the user in the summary display separately from the message body allows this data to be presented to the user without the need to download any part of the message bodies

3.1.5. Proxy Re-encryption

The ability to re-use the output of a key exchange is of particular importance when using proxy re-encryption or distributed key generation as completing each key agreement incurs an interaction with the key server.

3.2. Additional Use Cases

3.2.1. Streaming data

The DARE message format supports encryption and decryption in 'streaming mode' in which blocks of output data are emitted as the input is presented.

This allows synchronous communications (e.g. video, voice) to be supported and permits files of arbitrary size to be encrypted with finite state. It is not necessary to buffer the entire plaintext or ciphertext before generating a message.

3.2.2. Information Erasure by Key Deletion

Overwriting a DARE salt value prevents decryption of the corresponding data unless the salt can be recovered. Use of a suitably large random salt allows erasure of the salt to be

considered equivalent to erasure of the message data. For example, if a salt of 128 random bits and an encryption algorithm of at least 128 bits are used, the work factor for decryption will be $O(2^{128})$ even if the decryption key is compromised.

3.2.3. Field level encryption

The DARE key exchange and data item encoding may be applied to encrypt multiple fields in a single file under a Common Encryption Key. Field level encryption is particularly useful in database and spreadsheet applications.

4. Message Format

A consolidated DARE message is a sequence of four parts as follows.

Header Information a reader requires to being processing the message body.

By definition, the header of a consolidated message contains the key agreement information and the header of a detached message does not. The header may also be used to specify content metadata (such as the data type) and encrypted annotations.

Salt An opaque sequence of octets that is (statistically) unique to this combination of body and Master Key.

Body The data block

Trailer Information a reader requires to complete processing the message body that is not provided in the header.

A detached DARE message has the same sequence of four parts but the header part is empty. The header being communicated out-of-band with respect to the message to which it appears.

4.1. Encodings

A DARE message MAY be presented in JSON encoding or a compact encoding based on JSON-B.

4.1.1. JSON

The DARE message is encoded as JSON sequence with up to four entries. The position of the item in the sequence specifies its function. Thus the Header entry MAY be empty but MUST not be absent.

Header The header is encoded as a JSON object

Salt The salt is encoded as a base64url encoded string

Body The body is encoded as a base64url encoded string

Trailer The trailer is encoded as a JSON object

For example, the following sequence is a JSON encoded message with an empty header and trailer and a salt and body of zero length:

```
[ {}, "", "", {} ]
```

Figure 1

4.1.2. JSON-B

JSON-B Encoding provides a more compact representation and in particular, allows ciphertext to be presented in binary form as opposed to Base-64 encoding. Note that JSON-B encoding is a superset of JSON, a JSON-B decoder will be able to decode either format without additional tagging to specify which format is being used.

The square braces used to specify a JSON sequence MUST be present when a DARE Message is embedded in a JSON-B encoded object but MAY be omitted in situations where no ambiguity arises from doing so. For example, when presenting a DARE Message as a standalone file or in a DARE Container.

4.1.3. Application Directed Encoding

Applications MAY define their own encoding mechanisms to suit their needs.

5. Processing Model

The DARE processing model is based on the model in JWE [RFC7516] with the following extensions:

- o Support for multiple recipients
- o Support for multiple message bodies encrypted under keys derived from a single key exchange.
- o Signature and encryption are supported in a single format rather than separately.
- o Authentication tags are appended to the end of the message body.

- o Message Authentication Codes are supported as a means of authentication.

5.1. Consolidated Message Generation

Two generation modes are supported:

Buffered The data body is buffered in memory while the data header is completed.

Streaming The DARE message is generated in a single pass.

Use of buffered mode avoids the need for data chunking and allows messages to provide all the information required for processing in the message header.

Use of streamed mode avoids the need to buffer the data body while assembly of the header is completed. This allows messages of arbitrary size to be processed with fixed resources.

5.1.1. Message Header

The Message header contains the key exchange information which MAY be shared by multiple related messages,

5.1.1.1. Signatures and Signers

If the message is to be authenticated by means of a digital signature, the Header MUST contain either a Signatures field or a Signers field but not both.

The Signatures field contains the actual signature value which cannot usually be calculated until processing of the message body is complete. The Signers field contains all the information from a signature except for the signature itself. This allows a verifier to perform decryption and signer verification processing in parallel and to reject a message that is not signed by an accepted signer before completing processing of the message body.

5.1.1.2. Key Exchange

The DARE key exchange is based on the JWE key exchange except that encryption modes are intentionally limited and the output of the key exchange is the DARE Master Key rather than the Content Encryption Key.

A DARE Key Exchange MAY contain any number of Recipient entries, each providing a means of decrypting the Master Key using a different private key.

If the Key Exchange mechanism supports message recovery, Direct Key Agreement is used, in all other cases, Key Wrapping is used.

This approach allows messages with one intended recipient to be handled in the exact same fashion as messages with multiple recipients. While this does require an additional key wrapping operation, that could be avoided if a message has exactly one intended recipient, this is offset by the reduction in code complexity.

If the key exchange algorithm does not support message recovery (e.g. Diffie Hellman and Elliptic Curve Diffie-Hellman), the HKDF Extract-and-Expand Key Derivation Function is used to derive a master key using the following info tag:

```
"dare-master" [64 61 72 65 2d 6d 61 73 74 65 72] Key derivation info
  field used when deriving a master key from the output of a key
  exchange.
```

The master key length is the maximum of the key size of the encryption algorithm specified by the key exchange header, the key size of the MAC algorithm specified by the key exchange header (if used) and 256.

5.1.1.3. Key Identifier

The JWE/JWS specifications define a kid field for use as a key identifier but not how the identifier itself is constructed. All DARE key identifiers are either UDF key fingerprints [draft-hallambaker-udf] or Group Key Identifiers.

A UDF fingerprint is formed as the digest of an IANA content type and the digested data. A UDF key fingerprint is formed with the content type application/pkix-keyinfo and the digested data is the ASN.1 DER encoded PKIX certificate keyInfo sequence for the corresponding public key.

A Group Key Identifier has the form <fingerprint>@<domain>. Where <fingerprint> is a UDF key fingerprint and <domain> is the DNS address of a service that provides the encryption service to support decryption by group members.

5.1.2. Enhanced Data Sequence

A DARE Enhanced Data Segment is an atomic unit that contains a salt and the result(s) of applying the salt and Master Key to a plaintext under the enhancement mode specified in the key exchange. The following enhancement modes are supported:

Plaintext The EDS consists of a plaintext.

Authenticated The EDS consists of a plaintext with an authentication tag appended to the end.

Encrypted The EDS consists of a ciphertext only

Encrypted and Authenticated The EDS consists of a ciphertext with an authentication tag appended to the end.

In each case the encryption and/or authentication algorithms and all associated parameters (key size, output length) are specified in the associated key exchange header.

A DARE message MAY contain multiple Enhanced Data Sequences. The message body, cloaked headers, annotations and signature values are all presented as Enhanced Data Sequences.

The message body is distinct from all other Enhanced Data Sequences in that each message MUST have exactly one message body and only the message body can be signed. Additionally, when the compact encoding is used, it is the only Enhanced Data Sequence that can be of variable length.

5.1.2.1. Salt

A salt is a sequence of zero or more octets that is unique within the scope of a Master Key.

Generators SHOULD NOT generate salt values that exceed 1024 octets.

The salt value is opaque to the DARE encoding but MAY be used to encode application specific semantics including:

- o Frame number to allow reassembly of a data sequence split over a sequence of messages which may be delivered out of order.
- o Transmit the Master Key in the manner of a Kerberos ticket to allow some (but not necessarily all) to avoid the need to perform a key exchange.

- o Identify the Master Key under which the Enhanced Data Sequence was generated.
- o Enable erasure of the encrypted data plaintext by erasure of the encryption key.

For data erasure to be effective, the salt must be constructed so that the difficulty of recovering the key is sufficiently high that it is infeasible. For most purposes, a salt with 128 bits of appropriately random data will be sufficient.

5.1.2.2. Key Derivation

Encryption and/or authentication keys are derived from the Master Key using a Extract-and-Expand Key Derivation Function as follows:

1. The Master Key and salt value are used to extract the PRK (pseudorandom key)
2. The PRK is used to derive the algorithm keys using the application specific information input for that key type.

The application specific information inputs are:

"dare-encrypt" [64 61 72 65 2d 65 6e 63 72 79 70 74] To generate an encryption or encryption with authentication key.

"dare-iv" [64 61 72 65 2d 65 6e 63 72 79 70 74] To generate an initialization vector.

"dare-mac" [dare-mac] To generate a Message Authentication Code key.

5.1.2.3. Content

The content of an Enhanced Data Sequence is the plaintext or ciphertext with the appended authentication tag as directed by the enhancement mode.

Support for enhancement of unbuffered streaming data presents implementations with two cases of interest:

5.1.2.3.1. Known Length

If the length of the body is known in advance of enhancement the generator can calculate the final length of the Enhanced Data Segment before encryption begins. This allows encryption of (e.g.) data files as a single data-last production when the compact encoding is being used.

5.1.2.3.2. Unknown Length

If the final length of the body is not known in advance of enhancement the generator must either buffer the output data in memory before generation or use an encoding that permits indefinite length octet sequences to be represented. In the compact encoding, this means a (possibly zero length) sequence of data-chunk productions terminated by a single data-last production

5.1.3. Trailer

The trailer only contains information when the message is authenticated by means of a signature.

5.1.3.1. Signatures

A list of message signature values.

If the message header contains a Signer field, the trailer **MUST** contain a Signatures field giving the corresponding signature values.

5.2. Consolidated Message Recovery

Decryption and verification is the opposite of the generation process

5.2.1. Verify signers

Recipients **MAY** verify that the message signatures are adequate to meet the requirements of the security policy at any point in message recovery.

The recipient determines which signer or signature entries are appropriate to their needs and if so, which digest algorithms are to be applied to the message plaintext.

5.2.2. Match recipient info

The first step in recovering the master key is to determine which (if any) of the recipient entries can be used for decryption by examining the kid fields.

5.2.3. Recover Master Key

Having identified the key exchange to use, the Master key.

5.2.4. Process Enhanced Data Sequences

Having recovered the Master Key, the recipient can decrypt whichever Enhanced Data Sequences it requires.

5.2.5. Signature validation

If validation of one or more signature entries is required, the recipient recovers the signature value from the corresponding Enhanced Data Sequences and performs signature verification according to the specified algorithm.

5.3. Detached Messages

Processing of a detached message is the same as for a consolidated message except for the fact that at least some of the header information is passed out of band with respect to the message. There are thus two sets of headers:

Context Header The header that contains the key exchange (passed out of band with respect to the message).

Message Header Additional header data contained in the message itself.

Message headers take precedence over context headers. Thus if the encryption algorithm field 'enc' is specified in both places, the value specified in the message header takes precedence.

5.4. Cloaked Messages

A cloaked message is a DARE message that contains a Cloaked field in the header.

Implementations MAY support generation and parsing of cloaked messages. Implementations SHOULD NOT generate and MAY reject nested cloaked headers unless specifically directed by an application specification.

The use of cloaked headers allows a second layer of key agreement to be specified within the first. The message body is always encrypted under the Master Key corresponding to the innermost key exchange.

Enhanced Data Sequences that are contained in a cloaked header are encrypted under the master key contained in that header.

6. Algorithms

6.1. Field: kwd

The key wrapping and derivation algorithms.

Since the means of public key exchange is determined by the key identifier of the recipient key, it is only necessary to specify the algorithms used for key wrapping and derivation.

The default (and so far only) algorithm is kwd-aes-sha2-256-256.

Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm [RFC3394] is used to wrap the Master Exchange Key. AES 256 is used.

HMAC-based Extract-and-Expand Key Derivation Function [RFC5869] is used for key derivation. SHA-2-256 is used for the hash function.

7. Reference

A DARE Message consists of a Header, an Enhanced Data Sequence (EDS) and an optional trailer. This section describes the JSON data fields used to construct headers, trailers and complete messages.

Wherever possible, fields from JWE, JWS and JWK have been used. In these cases, the fields have the exact same semantics. Note however that the classes in which these fields are presented have different structure and nesting.

7.1. Message Classes

A DARE Message contains a single DAREMessageSequence in either the JSON or Compact serialization as directed by the protocol in which it is applied.

7.1.1. Structure: DAREMessageSequence

A DARE Message containing Header, EDS and Trailer in JSON object encoding. Since a DAREMessage is almost invariably presented in JSON sequence or compact encoding, use of the DAREMessage subclass is preferred.

Although a DARE Message is functionally an object, it is serialized as an ordered sequence. This ensures that the message header field will always precede the body in a serialization, this allowing processing of the header information to be performed before the entire body has been received.

Header: DAREHeader (Optional) The message header. May specify the key exchange data, pre-signature or signature data, cloaked headers and/or encrypted data sequences.

Body: Binary (Optional) The message body

Trailer: DARETrailer (Optional) The message trailer. If present, this contains the signature.

7.2. Header and Trailer Classes

A DARE Message sequence MUST contain a (possibly empty) DAREHeader and MAY contain a DARETrailer.

7.2.1. Structure: DARETrailer

A DARE Message Trailer

Signatures: DARESignature [0..Many] A list of signatures. A message trailer MUST NOT contain a signatures field if the header contains a signatures field.

7.2.2. Structure: DAREHeader

Inherits: DARETrailer

A DARE Message Header. Since any field that is present in a trailer MAY be placed in a header instead, the message header inherits from the trailer.

EncryptionAlgorithm: String (Optional) The encryption algorithm as specified in JWE

AuthenticationAlgorithm: String (Optional) Message Authentication Code algorithm

Cloaked: Binary (Optional) If present in a header or trailer, specifies an encrypted data block containing additional header fields whose values override those specified in the message and context headers.

When specified in a header, a cloaked field MAY be used to conceal metadata (content type, compression) and/or to specify an additional layer of key exchange. That applies to both the Message body and to headers specified within the cloaked header.

Processing of cloaked data is described in?

ContentType: String (Optional) The content type field as specified in JWE

EDSS: Binary [0..Many] If present, the Encrypted Data Segments field contains a sequence of Encrypted Data Segments encrypted under the message Master Key. The interpretation of these fields is application specific.

Signers: DARESigner [0..Many] A list of 'presignature'

Recipients: DARERecipient [0..Many] A list of recipient key exchange information blocks.

7.3. Cryptographic Data

DARE Message uses the same fields as JWE and JWS but with different structure. In particular, DARE messages MAY have multiple recipients and multiple signers.

7.3.1. Structure: DARESigner

The signature value

Dig: String (Optional) Digest algorithm hint. Specifying the digest algorithm to be applied to the message body allows the body to be processed in streaming mode.

Alg: String (Optional) Key exchange algorithm

KeyIdentifier: String (Optional) Key identifier of the signature key.

Certificate: X509Certificate (Optional) PKIX certificate of signer.

Path: X509Certificate (Optional) PKIX certificates that establish a trust path for the signer.

7.3.2. Structure: X509Certificate

X5u: String (Optional) URL identifying an X.509 public key certificate

X5: Binary (Optional) An X.509 public key certificate

7.3.3. Structure: DARESignature

Inherits: DARESigner

The signature value

SignatureValue: Binary (Optional) The signature value as an Enhanced Data Sequence under the message Master Key.

7.3.4. Structure: DARERecipient

Recipient information

KeyIdentifier: String (Optional) Key identifier for the encryption key.

The Key identifier MUST be either a UDF fingerprint of a key or a Group Key Identifier

KeyWrapDerivation: String (Optional) The key wrapping and derivation algorithms.

WrappedMasterKey: Binary (Optional) The wrapped master key. The master key is encrypted under the result of the key exchange.

RecipientKeyData: String (Optional) The per-recipient key exchange data.

8. Security Considerations

8.1. Encryption/Signature nesting

8.2. Side channel

8.3. Salt reuse

9. IANA Considerations

10. Acknowledgements

11. Test Examples

In the following examples, Alice's public key parameters are:

```
{
  "PrivateKeyDH": {
    "kid": "MB6GU-TKTNH-QOH3W-UXMLL-336UD-R52NE-A",
    "Domain": "YE6bnq1MlX5ojaJto6PLP_PEWa",
    "Public": "rJIvzTjRmFa6SBtwnPe_ZkBGfxxa_W0yS5K0cnVI2x6rtT2eSkpp
ovzCtTagsAlnAwI426HgQcinyF4AxJMNqOqqG9KvoGnbmMclnMdvRYY7nGFecoT4R
6BmvpJtjih3M6odHslh60niJMQGuYdhF9-8nFPi9uC4WJl jplq9FCReHKG_0RkBp
dsa2CGQNQLG3yXCvFxj99cvkZknmUY_VSq8X3tMVA0-3MbN-fIPTr5aCuJE5jPcrK
38taOwekH5XGyc-Mls4-rUd5zK0nr2uCiO2RIVKJQf jQ1h62rTVRf3bdP477Wk1ZN
iiWuHi juF8uJVlT2RDLMRJ6NM3exca",
    "Private": "8PreevvNyy8MEutpHO_bqh2_N_9zH9R5mE5N76dLXIX0VZsob0f
z9za_lQ-nP4aRAFUPwJpqli0FNFS5a3BWW2AnRV_hCxsVFA12UX6v4eCxnPyx2sU
is9rlbgptfj_wu2FtEIKZTFh1AZ7ncgypK0N6OdWkolhmHUDVrWBU1PFG34jVgvY9
-MFWX8Ee3Uq6foMyGQo6wGRqMlvUWcfiFrETH-Xdi-DqKPfabeaKB3zGCA-YRH3L8
SlzQZ-qzE0G7tJvA8yMlZJ84m-GT8R_uXvgMoPWr1Je5oR16ZyIKru2enfH2jwOzV
tTiDfwQ3RVgA-TzrsIO-pCLyQ-ztYOQ"}}
```

Figure 2

The body of the test message is the UTF8 representation of the following string:

```
"This is a test long enough to require multiple blocks"
```

Figure 3

The EDS sequences, are the UTF8 representation of the following strings:

```
"Subject: Message metadata should be encrypted"
"2018-02-01"
```

Figure 4

11.1. Plaintext Message

A plaintext message without associated EDS sequences is an empty header followed by the message body:

```
{
  "DAREMessage": [{},
    "VGhpcyBpcyBhIHRlc3QgbG9uZyBlbm91Z2ggdG8gcmVxdWlyZ
SBtdWx0aXBsZSBibG9ja3M"}}
```

Figure 5

11.2. Plaintext Message with EDS

If a plaintext message contains EDS sequences, these are also in plaintext:

```
{
  "DAREMessage": [{
    "edss": ["U3ViamVjdDogTWVzc2FnZSBtZXRhZGF0YSBzaG91bGQgYmUgZW5jcn
lwdGVk",
    "MjAxOC0wMi0wMQ" ]}],
  "VGhpcyBpcyBhIHRlc3QgbG9uZyBlbm91Z2ggdG8gcmVxdWlyZSBtdW
x0aXBsZSBibG9ja3M" ]}]
```

Figure 6

11.3. Encrypted Message

The creator generates a master session key:

```
1E 20 22 8F B7 5E 47 59 06 DA D9 39 A1 00 7E C5
63 15 A0 42 18 40 1D AD 17 EB 5B 87 E3 F3 0F B0
```

Figure 7

The creator generates an ephemeral key:

```
{
  "PrivateKeyDH": {
    "kid": "MB7ZZ-3W33A-F2VNY-ADWAT-6X65X-HHHPG-A",
    "Domain": "YE6bnq1MlX5ojaJto6PLP_PEWa",
    "Public": "bhlMqhBTwLSjpphlieEbehhDLmJzlNa6T_KgryqJk6CpgosOGMnj
nvkVx4SflKwMDZgdmlHMmKxuAF5AFcvC5iVaLKcN8GFKiUMFnl-u5HCSkZ4p07VtD
WIEw22eQ6kpne6lPROLGenizs8NEC3lQwDsbt7BlT3yvNyys7Cq1Zi7zEyticxRP
JT-7ayFb0c2SkkeSgcaAnOnADIjtOSUNP6oyJfdAGCric3L9q2sYQH0A3zofCDkMH
PTz9B2juBTIkF2ZP-E4FL3qskm8rFhWXYDaBOoz0L8Ri0H4yL5lIR_NYeU7NmEsmN
soPnMzFub5-vDk05hKzSYsZSapxsNw",
    "Private": "H9NDxDPDczPI4VfaADhpktKGNnbbFEXU32LrN6HWUoQc-jjgEjq
JQPSSsh5OqOGI-qH3MRIWQptnZBoZZw8AZrst2hMSUgavR5pjdOUfI3qFDJ7X2BoJ
9tWIXderQRY68yZKQ2GRIsYWuJX-G4rMBne3OJMpYvSgOdMVAULfhIqQxQ4Bg36nh
5aiAFhHbAhFT_U8G9uJVp-MziOpFJMDJuuVpaf19BMwEroAbXbnqcCPZxm52MwrUa
15bUsRPiad4Rs3nUy3vwdBC005jJQDw_F7ood6-uhpTx7t_R5vntf5Tjq7hEU1QLq
EZskmJ6uK4Lya6-WoAPpsmTKgXI_HXg" ]}]
```

Figure 8

The key agreement value is calculated:

```

22811486391394414590492811341153777463986902783003089366556548209526762765730219
68381370738679940675029069199175392312837914680796793807880753926876415377038564
68986656850082146272303282349425733892929597554045646160247703300321053908130787
23890794285774439949542234967770376637613771818877350239294601632726372687979478
37026548247760601082258162277489985459491560857702094183250665013248801559362919
53094453368421262877531847317120913452105430381959521014059560632481379056251189
65725721384090610579010991196609345058434675430657696472302230895509628359510168
125995303027050657800721605305012305489054529556556071287

```

Figure 9

The key agreement value is used as the input to a HKDF key derivation function with the info parameter "master".

```

E6 2B 66 B3 03 F1 23 62 50 1D 88 5E A3 1C DB 86
16 80 A1 1A 62 E9 9F 2B D1 5B 35 66 84 FB 08 ED

```

Figure 10

To create the first EDS, a salt value is assigned. In this case a single octet with the value '01'. The salt value is then used to create the encryption key and IV as follows:

Salt:

```
Example.DareEDSSalt.ToStringBasel6FormatHex()
```

Encryption Key:

```
Example.DareMessageKeyEncrypt.ToStringBasel6FormatHex()
```

IV:

```
Example.DareMessageKeyIV.ToStringBasel6FormatHex()
```

Figure 11

The output sequence is the salt followed by the ciphertext:

```

88 01 01 88 40 15 3A AA 2A 62 BA 09 F2 C5 7D EF
40 F1 DF 4F 8E 62 9B 75 0B C1 CB 6D AE 87 BE 8C
C3 BE 92 F9 42 64 8D 37 DB 1D 69 7C D9 60 21 05
94 C5 1C 8B F0 AC 17 5F 55 D6 D1 1F A5 EA 1C D7
42 39 E7 8C 61

```

Figure 12

The completed message is:

```
{
  "DAREMessage": [{
    "recipients": [{
      "kid": "MB6GU-TKTNH-QOH3W-UXMLL-336UD-R52NE-A",
      "epk": {
        "PublicKeyDH": {
          "kid": "MB7ZZ-3W33A-F2VNY-ADWAT-6X65X-HHHPG-A",
          "Domain": "YE6bnqlMlX5ojaJto6PLP_PEWa",
          "Public": "bhlMqhBTwLSjpplhieEbehDLmJzlNa6T_KgryqJk6CpgosOGMnj
nvkVx4SflKwMDZgdmlHMmKxuAF5AFcvC5iVaLKcN8GFKiUMFn1-u5HCSkZ4pO7VtD
WIEw22eQ6kpne6lPRoLGenizs8NEC3lQwDsbt7BlT3yvNyys7Cq1Zi7zEytcikxRP
JT-7ayFb0c2SkkeSgcaAnOnADIjtOSUNP6oyJfdAGCric3L9q2sYQH0A3zofCDkMH
PTz9B2juBTIkF2ZP-E4FL3qskm8rFhWXYDaBOoz0L8Ri0H4yL5lIR_NYeU7NmEsmN
soPnMzFub5-vDk05hKzSYsZSapxsNw"}},
      "wmk": "BbAZbgVlhI35WYKXJ-JgYDr0XyaFwa30KNJaoWmQhItN2Fb2WYfesA"}]],
  "iAEBiEAVOqoqYroJ8sV970Dx30-OYptlC8HLba6HvozDvpL5QmSNN9s
daXzZYCEfLMUci_CsF19V1tEfpeoc10I554xh"]}]}
```

Figure 13

11.4. Signed Messages

This is not yet implemented.

12. References

12.1. Normative References

- [draft-hallambaker-jsonbcd]
 - Hallam-Baker, P., "Binary Encodings for JavaScript Object Notation: JSON-B, JSON-C, JSON-D", draft-hallambaker-jsonbcd-11 (work in progress), April 2018.
- [draft-hallambaker-udf]
 - Hallam-Baker, P., "Uniform Data Fingerprint (UDF)", draft-hallambaker-udf-10 (work in progress), April 2018.
- [IANAJOSE]
 - "[Reference Not Found!]".
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
- [RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998.

- [RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002.
- [RFC4880] Callas, J., Donnerhackle, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015.

12.2. URIs

- [1] <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html>
- [2] <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html>
- [3] <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html>
- [4] <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html>

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 28, 2019

P. Hallam-Baker
Comodo Group Inc.
August 27, 2018

Data At Rest Encryption Part 1: DARE Message Syntax
draft-hallambaker-dare-message-02

Abstract

This document describes the Data At Rest Encryption (DARE) message syntax. This syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary message content.

This document is also available online at
<http://mathmesh.com/Documents/draft-hallambaker-dare-message.html> [1]

.

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 February 28, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Encryption and Integrity	3
1.1.1. Key Exchange	4
1.1.2. Data Erasure	5
1.2. Signature	5
1.2.1. Signing Individual Plaintext Messages	6
1.2.2. Signing Individual Encrypted Messages	6
1.2.3. Signing Sequences of Messages	6
2. Definitions	7
2.1. Related Specifications	7
2.2. Requirements Language	7
2.3. Defined terms	7
3. Architecture	9
3.1. Processing Considerations	9
3.2. Content Metadata and Annotations	10
3.3. Encoded Data Sequence	11
3.4. Encryption and Integrity	12
3.4.1. Key Exchange	13
3.4.2. Key Identifiers	14
3.4.3. Salt Derivation	14
3.4.4. Key Derivation	15
3.5. Signature	16
4. Algorithms	16
4.1. Field: kwd	16
5. Reference	16
5.1. Message Classes	17
5.1.1. Structure: DAREMessageSequence	17
5.2. Header and Trailer Classes	17
5.2.1. Structure: DARETrailer	17
5.2.2. Structure: DAREHeader	18
5.3. Cryptographic Data	18
5.3.1. Structure: DARESigner	19
5.3.2. Structure: X509Certificate	19
5.3.3. Structure: DARESignature	19
5.3.4. Structure: DARERecipient	19
6. Security Considerations	20
6.1. Encryption/Signature nesting	20
6.2. Side channel	20
6.3. Salt reuse	20
7. IANA Considerations	20
8. Acknowledgements	20
9. Test Examples	20
9.1. Plaintext Message	22

9.2. Plaintext Message with EDS	22
9.3. Encrypted Message	22
9.4. Signed Message	25
9.5. Signed and Encrypted Message	26
10. References	27
10.1. Normative References	27
10.2. Informative References	28
10.3. URIs	29
Author's Address	29

1. Introduction

This document describes the Data At Rest Encryption (DARE) Message Syntax. This syntax is used to digitally sign, digest, authenticate, or encrypt arbitrary message content.

The DARE Message Syntax is based on a subset of the JSON Web Signature [RFC7515] and JSON Web Encryption [RFC7516] standards and shares many fields and semantics. The processing model and data structures have been streamlined to remove alternative means of specifying the same content.

A DARE Message consists of a Header, Payload and an optional Trailer. To enable single pass encoding and decoding, the Header contains all the information required to perform cryptographic processing of the Payload and authentication data (digest, MAC, signature values) may be deferred to the Trailer section.

The DARE Message Syntax is designed to compliment the DARE Container syntax. A DARE Container is an append-only log format consisting of a sequence of frames. Cryptographic enhancements (signature, encryption) may be applied to individual frames or to sets of frames. Thus, a single key exchange may be used to provide a master key to encrypt multiple frames and a single signature may be used to authenticate all the frames in the container up to and including the frame in which the signature is presented.

The DARE Message syntax may be used either as a standalone cryptographic message syntax or as a means of presenting a single DARE Container frame together with the complete cryptographic context required to verify the contents and decrypt them.

1.1. Encryption and Integrity

An important innovation in the DARE Message Syntax is the separation of key exchange and data encryption operations so that a Master Key (MK) established in a single exchange to be applied to multiple octet sequences. This means that a public key operation may be used to

encrypt multiple parts of the same message or to multiple frames in a DARE Container.

To avoid reuse of the key and to avoid the need to communicate separate IVs, each octet sequence is encrypted under a different encryption key (and IV if required) derived from the Master Key by means of a salt that is unique for each octet sequence that is encrypted. The same approach is used to generate keys for calculating a MAC over the octet sequence if required. This approach allows encryption and integrity protections to be applied to the message payload, to header or trailer fields or to application defined Enhanced Data Sequences in the header or trailer.

1.1.1. Key Exchange

Traditional cryptographic containers describe the application of a single key exchange to encryption of a single octet sequence. Examples include PKCS#7/CMS [RFC2315] , OpenPGP [RFC4880] and JSON Web Encryption [RFC7516] .

To encrypt a message using RSA, the encoder first generates a random encryption key and initialization vector (IV). The encryption key is encrypted under the public key of each recipient to create a per-recipient decryption entry. The encryption key, plaintext and IV are used to generate the ciphertext (figure 1).

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html> [2].]]

Monolithic Key Exchange and Encrypt

This approach is adequate for the task of encrypting a single octet stream. It is less than satisfactory when encrypting multiple octet streams or very long streams for which a rekeying operation is desirable.

In the DARE approach, key exchange and key derivation are separate operations and keys MAY be derived for encryption or integrity purposes or both. A single key exchange MAY be used to derive keys to apply encryption and integrity enhancements to multiple data sequences.

The DARE key exchange begins with the same key exchange used to produce the CEK in JWE but instead of using the CEK to encipher data directly, it is used as one of the inputs to a Key Derivation Function (KDF) that is used to derive parameters for each block of

data to be encrypted. To avoid the need to introduce additional terminology, the term 'CEK' is still used to describe the output of the key agreement algorithm (including key unwrapping if required) but it is more appropriately described as a Master Key (figure 2).

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html> [3].]]

Exchange of Master Key

A Master Key may be used to encrypt any number of data items. Each data item is encrypted under a different encryption key and IV (if required). This data is derived from the Master Key using the HKDF function [RFC5869] using a different salt for each data item and separate info tags for each cryptographic function (figure 3).

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html> [4].]]

Data item encryption under Master Key and per-item salt.

This approach to encryption offers considerably greater flexibility allowing the same format for data item encryption to be applied at the transport, message or field level.

1.1.2. Data Erasure

Each encrypted DARE Message specifies a unique Master Salt value of at least 128 bits which is used to derive the salt values used to derive cryptographic keys for the message payload and annotations.

Erasure of the Master Salt value MAY be used to effectively render the message payload and annotations undecipherable without altering the message payload data. The work factor for decryption will be $O(2^{128})$ even if the decryption key is compromised.

1.2. Signature

As with encryption, DARE Message signatures MAY be applied to an individual message or a sequence of messages.

1.2.1. Signing Individual Plaintext Messages

When an individual plaintext message is signed, the digest value used to create the signature is calculated over the binary value of the payload data. That is, the value of the payload before the encoding (Base-64, JSON-B) is applied.

1.2.2. Signing Individual Encrypted Messages

When an individual plaintext message is signed, the digest value used to create the signature is calculated over the binary value of the payload data. That is, the value of the payload after encryption but before the encoding (Base-64, JSON-B) is applied.

Use of signing and encryption in combination presents the risk of subtle attacks depending on the order in which signing and encryption take place [Davis2001] .

Na?ve approaches in which a message is encrypted and then signed present the possibility of a surreptitious forwarding attack. For example, Alice signs a message and sends it to Mallet who then strips off Alice's signature and sends the message to Bob.

Na?ve approaches in which a message is signed and then encrypted present the possibility of an attacker claiming authorship of a ciphertext. For example, Alice encrypts a ciphertext for Bob and then signs it. Mallet then intercepts the message and sends it to Bob.

While neither attack is a concern in all applications, both attacks pose potential hazards for the unwary and require close inspection of application protocol design to avoid exploitation.

To prevent these attacks, each signature on a message that is signed and encrypted MUST include a witness value that is calculated by applying a MAC function to the signature value as described in section XXX.

1.2.3. Signing Sequences of Messages

To sign multiple messages with a single signature, we first construct a Merkle tree of the message payload digest values and then sign the root of the Merkle tree.

[This is not yet implemented but will be soon.]

2. Definitions

2.1. Related Specifications

The DARE message format is based on the following existing standards and specifications.

Object serialization The JSON-B [draft-hallambaker-jsonbcd] encoding is used for object serialization. This encoding is an extension of the JavaScript Object Notation (JSON) [RFC7159] .

Message syntax The cryptographic processing model is based on JSON Web Signature (JWS) [RFC7515] , JSON Web Encryption (JWE) [RFC7516] and JSON Web Key (JWK) [RFC7517] .

Cryptographic primitives. The HMAC-based Extract-and-Expand Key Derivation Function [RFC5869] and Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm [RFC3394] are used.

The Uniform Data Fingerprint method of presenting data digests is used for key identifiers and other purposes [draft-hallambaker-udf] .

Cryptographic algorithms The cryptographic algorithms and identifiers described in JSON Web Algorithms (JWA) [RFC7518] are used together with additional algorithms as defined in the JSON Object Signing and Encryption IANA registry [IANAJOSE] .

2.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] .

2.3. Defined terms

The terms "Authentication Tag", "Content Encryption Key", "Key Management Mode", "Key Encryption", "Direct Key Agreement", "Key Agreement with Key Wrapping" and "Direct Encryption" are defined in the JWE specification [RFC7516] .

The terms "Authentication", "Ciphertext", "Digital Signature", "Encryption", "Initialization Vector (IV)", "Message Authentication Code (MAC)", "Plaintext" and "Salt" are defined by the Internet Security Glossary, Version 2 [RFC4949] .

Annotated Message A DARE Message that contains an Annotations field with at least one entry.

Authentication Data A Message Authentication Code or authentication tag.

Complete Message A DARE message that contains the key exchange information necessary for the intended recipient(s) to decrypt it.

Detached Message A DARE message that does not contain the key exchange information necessary for the intended recipient(s) to decrypt it.

Encryption Context The master key, encryption algorithms and associated parameters used to generate a set of one or more enhanced data sequences.

Encoded data sequence (EDS) A sequence consisting of a salt, content data and authentication data (if required by the encryption context).

Enhancement Applying a cryptographic operation to a data sequence. This includes encryption, authentication and both at the same time.

Generator The party that generates a DARE message.

Group Encryption Key A key used to encrypt data to be read by a group of users. This is typically achieved by means of some form of proxy re-encryption or distributed key generation.

Group Encryption Key Identifier A key identifier for a group encryption key.

Master Key (MK) The master secret from which keys are derived for authenticating enhanced data sequences.

Recipient Any party that receives and processes at least some part of a DARE message.

Related Message A set of DARE messages that share the same key exchange information and hence the same Master Key.

Uniform Data Fingerprint (UDF) The means of presenting the result of a cryptographic digest function over a data sequence and content type identifier specified in the Uniform Data Fingerprint specification [draft-hallambaker-udf]

3. Architecture

A DARE message is a sequence of three parts as follows.

Header A JSON object containing information a reader requires to begin processing the message.

Payload An array of octets.

Trailer A JSON object containing information calculated from the message payload.

For example, the following sequence is a JSON encoded DARE Message with an empty header, a payload of zero length and an empty trailer:

```
[ {}, "", {} ]
```

Figure 1

DARE Messages MAY be encoded using JSON serialization or a binary serialization for greater efficiency.

JSON Offers compatibility with applications and libraries that support JSON. Payload data is encoded using Base64 incurring a 33% overhead.

JSON-B A superset of JSON encoding that permits binary data to be encoded as a sequence of length-data segments. This avoids the Base64 overhead incurred by JSON encoding.

JSON-C A superset of JSON-C which provides additional efficiency by allowing field tags and other repeated string data to be encoded by reference to a dictionary.

DARE Message processors MUST support JSON serialization and SHOULD support JSON-B serialization.

3.1. Processing Considerations

The DARE Message Syntax supports single pass encoding and decoding without buffering of data. All the information required to begin processing a DARE message (key agreement information, digest algorithms), is provided in the message header. All the information that is derived from message processing (authentication codes, digest values, signatures) is presented in the message trailer.

The choice of message encoding does not affect the semantics of message processing. A DARE Message MAY be reserialized under the

same serialization or converted from any of the specified serialization to any other serialization without changing the semantics or integrity properties of the message.

3.2. Content Metadata and Annotations

A header MAY contain header fields describing the payload content. These include:

ContentType Specifies the IANA Content Type.

Annotations A list of Encoded Data Sequences that provide application specific annotations to the message.

The format of the Encoded Data Sequences is described in the following section.

Consider the following mail message:

From: Alice@example.com
To: bob@example.com
Subject: TOP-SECRET Product Launch Today!

The CEO told me the product launch is today. Tell no-one!

Figure 2

Existing encryption approaches require that header fields such as the subject line be encrypted with the body of the message or not encrypted at all. Neither approach is satisfactory. In this example, the subject line gives away important information that the sender probably assumed would be encrypted. But if the subject line is encrypted together with the message body, a mail client must retrieve at least part of the message body to provide a 'folder' view.

The following is a plaintext DARE Message in which the header fields of the mail message are presented as annotations:

```
[{
  "cty": "application/example-mail",
  "Annotations": [ "iAEBiBdGcm9tOiBBbGljZUBleGFtcGx1LmNvbYgA",
    "iAECiBNUbzogYm9iQGV4YWlwbGUuY29tiAA",
    "iAEDiClTdWJqZWN0OiBUT1AtU0VUDUKVUIFByb2R1Y3QgTGF1bmNoIFRvZGF5
    IYgA"
  ] },
  "VGhlIENFTyB0b2xkIG1lIHRobzSBwcm9kdWN0IGxhdW5jaCBpcyB0b2RheS4gVGVs
  bCBubylvbmUh"
]
```

Figure 3

3.3. Encoded Data Sequence

An encoded data sequence (EDS) is a sequence of octets that encodes a data sequence according to cryptographic enhancements specified in the context in which it is presented. An EDS MAY be encrypted and MAY be authenticated by means of a MAC. The keys and other cryptographic parameters used to apply these enhancements are derived from the cryptographic context and a Salt prefix specified in the EDS itself.

An EDS sequence contains exactly three binary fields encoded in JSON-B serialization as follows:

Salt Prefix A sequence of octets used to derive the encryption key, Initialization Vector and MAC key as required.

Body The plaintext or encrypted content.

Authentication Tag The authentication code value in the case that the cryptographic context specifies use of authenticated encryption or a MAC, otherwise is a zero-length field.

Requiring all three fields to be present, even in cases where they are unnecessary simplifies processing at the cost of up to six additional data bytes.

The encoding of the 'From' header of the previous example as a plaintext EDS is as follows:

```
88 01
  01
88 17
  46 72 6f 6d 3a 20 41 6c   69 63 65 40 65 78 61 6d
  70 6c 65 2e 63 6f 6d
88 00

~~~~
```

Figure 4

3.4. Encryption and Integrity

Encryption and integrity protections MAY be applied to any DARE Message Payload and Annotations.

The following is an encrypted version of the message shown earlier. The payload and annotations have both increased in size as a result of the block cipher padding. The header now includes Recipients and Salt fields to enable the content to be decoded.


```
[{
  "enc": "A256CBC",
  "Salt": "cvbklMnkhWE1-3_t3Z-UQ",
  "cty": "application/example-mail",
  "Annotations": [ "iAEBiCDUKHYfjqs5SHQAzwezya-guHTj0SkhmeHtjbHSWjLa
Fg",
    "iAECiCCi5gIDQIVQLnQmnc3ZUHb4rtmq3U-ctF52eEXRdNapxw",
    "iAEDiDAsnYWB_5SpxqyX_GLKSW4ztdPbEfsxMAUQnRzr7moJoORhlSxzLLYt
NPovkFVqXBg"
  ],
  "recipients": [{
    "kid": "MDAD3-E4BYE-MK6CH-QA2HD-TKRS2-KIX5Y-A",
    "epk": {
      "PublicKeyDH": {
        "kid": "MDIGT-2AA2Q-HFVRN-WYCNy-F32NC-FB4NP-A",
        "Domain": "YE6bnqlMlX5ojaJto6PLP_PEWa",
        "Public": "a3Vld_zF5tPfv3GxjfsdkdjdJqT0isXW9iwbZeXWGiCBV
dwTiGXAzDF0SqBMLbqe3lp1ZEMFh0mas6evqNUCM-yb9HuVtjzaaDcn3_jH2kkgQw
S9_4cD8_qFCazvxwAgKCnLn_VzKOBduT0uvQz1FEuCSv4t0kLlbCwDIe_7TMaoQPN
RcsVvMPPW6lfWkC5CiHaGN78bNCzkmplAbHV2g10oC4_NdRZ6m8kBkXWz3bnnLFGI
KWzXKTgfQXrksSktKa9pkwA_6KWEMiLPPHQhtj7wud09o7TzWfdcz3lFaGeMmTBSH
dFyRl0LSbia9qsrgN6tLUCwOtbuKV3XoEYlvwA" } },
    "wmk": "mzXRWy5OjlsZFHT_teqx3E6wtL6eLVFekiUXO_txmUSt3ILj9xgJ
Lw" }
  ] },
  "vTg5MAhAmIKF7LCFeE8xFhqOFQ5znVtkIctqyyctDkTNL69Tf0KhsC5VoVty3JCq
RNth3hIfdHJZlU_BnMRzyA"
]
```

Figure 5

3.4.1. Key Exchange

The DARE key exchange is based on the JWE key exchange except that encryption modes are intentionally limited and the output of the key exchange is the DARE Master Key rather than the Content Encryption Key.

A DARE Key Exchange MAY contain any number of Recipient entries, each providing a means of decrypting the Master Key using a different private key.

If the Key Exchange mechanism supports message recovery, Direct Key Agreement is used, in all other cases, Key Wrapping is used.

This approach allows messages with one intended recipient to be handled in the exact same fashion as messages with multiple recipients. While this does require an additional key wrapping

operation, that could be avoided if a message has exactly one intended recipient, this is offset by the reduction in code complexity.

If the key exchange algorithm does not support message recovery (e.g. Diffie Hellman and Elliptic Curve Diffie-Hellman), the HKDF Extract-and-Expand Key Derivation Function is used to derive a master key using the following info tag:

```
"dare-master" [64 61 72 65 2d 6d 61 73 74 65 72] Key derivation info
  field used when deriving a master key from the output of a key
  exchange.
```

The master key length is the maximum of the key size of the encryption algorithm specified by the key exchange header, the key size of the MAC algorithm specified by the key exchange header (if used) and 256.

3.4.2. Key Identifiers

The JWE/JWS specifications define a kid field for use as a key identifier but not how the identifier itself is constructed. All DARE key identifiers are either UDF key fingerprints [draft-hallambaker-udf] or Mesh/Recrypt Group Key Identifiers.

A UDF fingerprint is formed as the digest of an IANA content type and the digested data. A UDF key fingerprint is formed with the content type application/pkix-keyinfo and the digested data is the ASN.1 DER encoded PKIX certificate keyInfo sequence for the corresponding public key.

A Group Key Identifier has the form <fingerprint>@<domain>. Where <fingerprint> is a UDF key fingerprint and <domain> is the DNS address of a service that provides the encryption service to support decryption by group members.

3.4.3. Salt Derivation

A Master Salt is a sequence of 16 or more octets that is specified in the Salt field of the header.

The Master Salt is used to derive salt values for the message payload and associated encoded data sequences as follows.

Payload

EDS

Encoders SHOULD NOT generate salt values that exceed 1024 octets.

The salt value is opaque to the DARE encoding but MAY be used to encode application specific semantics including:

- o Frame number to allow reassembly of a data sequence split over a sequence of messages which may be delivered out of order.
- o Transmit the Master Key in the manner of a Kerberos ticket to allow some (but not necessarily all) to avoid the need to perform a key exchange.
- o Identify the Master Key under which the Enhanced Data Sequence was generated.
- o Enable erasure of the encrypted data plaintext by erasure of the encryption key.

For data erasure to be effective, the salt MUST be constructed so that the difficulty of recovering the key is sufficiently high that it is infeasible. For most purposes, a salt with 128 bits of appropriately random data is sufficient.

3.4.4. Key Derivation

Encryption and/or authentication keys are derived from the Master Key using a Extract-and-Expand Key Derivation Function as follows:

1. The Master Key and salt value are used to extract the PRK (pseudorandom key)
2. The PRK is used to derive the algorithm keys using the application specific information input for that key type.

The application specific information inputs are:

"dare-encrypt" [64 61 72 65 2d 65 6e 63 72 79 70 74] To generate an encryption or encryption with authentication key.

"dare-iv" [64 61 72 65 2d 65 6e 63 72 79 70 74] To generate an initialization vector.

"dare-mac" [dare-mac] To generate a Message Authentication Code key.

3.5. Signature

While encryption and integrity enhancements can be applied to any part of a DARE message, signatures are only applied to payload digest values calculated over one or more message payloads.

The payload digest value for a message is calculated over the binary payload data. That is, after any encryption enhancement has been applied but before the message encoding is applied. This allows messages to be converted from one encoding to another without affecting signature verification.

Single Payload The signed value is the payload digest of the message payload.

Multiple Payload. The signed value is the root of a Merkle Tree in which the payload digest of the message is one of the leaves.

Verification of a multiple payload signature naturally requires the additional digest values required to construct the Merkle Tree. These are provided in the Trailer in a format that permits multiple signers to reference the same tree data.

4. Algorithms

4.1. Field: kwd

The key wrapping and derivation algorithms.

Since the means of public key exchange is determined by the key identifier of the recipient key, it is only necessary to specify the algorithms used for key wrapping and derivation.

The default (and so far only) algorithm is kwd-aes-sha2-256-256.

Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm [RFC3394] is used to wrap the Master Exchange Key. AES 256 is used.

HMAC-based Extract-and-Expand Key Derivation Function [RFC5869] is used for key derivation. SHA-2-256 is used for the hash function.

5. Reference

A DARE Message consists of a Header, an Enhanced Data Sequence (EDS) and an optional trailer. This section describes the JSON data fields used to construct headers, trailers and complete messages.

Wherever possible, fields from JWE, JWS and JWK have been used. In these cases, the fields have the exact same semantics. Note however that the classes in which these fields are presented have different structure and nesting.

5.1. Message Classes

A DARE Message contains a single DAREMessageSequence in either the JSON or Compact serialization as directed by the protocol in which it is applied.

5.1.1. Structure: DAREMessageSequence

A DARE Message containing Header, EDS and Trailer in JSON object encoding. Since a DAREMessage is almost invariably presented in JSON sequence or compact encoding, use of the DAREMessage subclass is preferred.

Although a DARE Message is functionally an object, it is serialized as an ordered sequence. This ensures that the message header field will always precede the body in a serialization, this allowing processing of the header information to be performed before the entire body has been received.

Header: DAREHeader (Optional) The message header. May specify the key exchange data, pre-signature or signature data, cloaked headers and/or encrypted data sequences.

Body: Binary (Optional) The message body

Trailer: DARETrailer (Optional) The message trailer. If present, this contains the signature.

5.2. Header and Trailer Classes

A DARE Message sequence MUST contain a (possibly empty) DAREHeader and MAY contain a DARETrailer.

5.2.1. Structure: DARETrailer

A DARE Message Trailer

Signatures: DARESignature [0..Many] A list of signatures. A message trailer MUST NOT contain a signatures field if the header contains a signatures field.

5.2.2. Structure: DAREHeader

Inherits: DARETrailer

A DARE Message Header. Since any field that is present in a trailer MAY be placed in a header instead, the message header inherits from the trailer.

EncryptionAlgorithm: String (Optional) The encryption algorithm as specified in JWE

AuthenticationAlgorithm: String (Optional) Message Authentication Code algorithm

Cloaked: Binary (Optional) If present in a header or trailer, specifies an encrypted data block containing additional header fields whose values override those specified in the message and context headers.

When specified in a header, a cloaked field MAY be used to conceal metadata (content type, compression) and/or to specify an additional layer of key exchange. That applies to both the Message body and to headers specified within the cloaked header.

Processing of cloaked data is described in?

ContentType: String (Optional) The content type field as specified in JWE

EDSS: Binary [0..Many] If present, the Encrypted Data Segments field contains a sequence of Encrypted Data Segments encrypted under the message Master Key. The interpretation of these fields is application specific.

Signers: DARESigner [0..Many] A list of 'presignature'

Recipients: DARERecipient [0..Many] A list of recipient key exchange information blocks.

5.3. Cryptographic Data

DARE Message uses the same fields as JWE and JWS but with different structure. In particular, DARE messages MAY have multiple recipients and multiple signers.

5.3.1. Structure: DARESigner

The signature value

Dig: String (Optional) Digest algorithm hint. Specifying the digest algorithm to be applied to the message body allows the body to be processed in streaming mode.

Alg: String (Optional) Key exchange algorithm

KeyIdentifier: String (Optional) Key identifier of the signature key.

Certificate: X509Certificate (Optional) PKIX certificate of signer.

Path: X509Certificate (Optional) PKIX certificates that establish a trust path for the signer.

5.3.2. Structure: X509Certificate

X5u: String (Optional) URL identifying an X.509 public key certificate

X5: Binary (Optional) An X.509 public key certificate

5.3.3. Structure: DARESignature

Inherits: DARESigner

The signature value

SignatureValue: Binary (Optional) The signature value as an Enhanced Data Sequence under the message Master Key.

5.3.4. Structure: DARERecipient

Recipient information

KeyIdentifier: String (Optional) Key identifier for the encryption key.

The Key identifier MUST be either a UDF fingerprint of a key or a Group Key Identifier

KeyWrapDerivation: String (Optional) The key wrapping and derivation algorithms.

WrappedMasterKey: Binary (Optional) The wrapped master key. The master key is encrypted under the result of the key exchange.

RecipientKeyData: String (Optional) The per-recipient key exchange data.

6. Security Considerations

6.1. Encryption/Signature nesting

6.2. Side channel

6.3. Salt reuse

7. IANA Considerations

8. Acknowledgements

9. Test Examples

In the following examples, Alice's encryption private key parameters are:

```
{
  "PrivateKeyDH": {
    "kid": "MDAD3-E4BYE-MK6CH-QA2HD-TKRS2-KIX5Y-A",
    "Domain": "YE6bnq1MlX5ojaJto6PLP_PEWa",
    "Public": "GaMqeF8I2s00AUMm1DldcXflA6lrrbUflBbpYHLtz3GrwkVR_JsGd
qlCVWQEX9McxsYTJUoFJzeHivLsyoULAY9H3Qtm_aClOeAR08yXboUTjKtwgH1qu0
s6kKl-p-tcaqu2PXReTVfOG9HtSR60iSuRZ9G9JHpKQNZCTdpoq4B2oBw6LvSly-p
18bDCB_Je00cqT8Aba08TjkhIG20HaZDEgdDOUlnIAT1hIhx48sLuPSdou-F76l47
8Jte_oJKTqGCUUeOm_397_d4sbaCiPO0RMllIFC7VEhi2TIDL6bRF7ujmCdxreYn9
DldJrlnF7hkcXEALL_5NyLVxwKdBw",
    "Private": "cokgZSxP_IdDAswgUZWDFu6qlKrEVX83uwDU2ji5LWiIoknyE6dY
wW2_mADDQ_2DRxm73J3fTH6C73KAOx-cJnJYDqr8kU3FhBEXBn8wxFL6M-SgVxyKa
jZ5MRL0-3EAYNCAE_HLNoeUqAqGachw-lnDgQO7e4F0hkEwa29JdwrBKTJxY93WAB
Xd3xZahbNws3sh2MU_FGU6AzfnTWPdvWTTxImySVF47pex-gRbYFVV-CmlZUuof-5
flM-RwUlnf28qJ8SJ_-zKmJLc6TSrMky8oxlu6v7WHoFRiweMCyBQ3x2Zc2ypXPfF
DvvAu5RYiVoUbrlES1UswPObiE2lOg"}}
```

Figure 6

Alice's signature private key parameters are:


```
{
  "PrivateKeyRSA": {
    "kid": "MBWNO-2J43U-ESWKW-XQWL6-6YGEW-UOPWU-A",
    "n": "1NzbnmakMalVH1mRv7TEDEhwXDNojn5wZbqltv1gp5PgZwzX-klYXuFhj0-
MpO0zcwptsUaYJhwdvvgW_ludUpISQYluXOB3UMj2e_0yl64MvnqTL47SZQuAN3QQ
9cuCw_-_Eyj_jerspauqa6RpNzGcabZrtRl7J7DPVZ3SNlw-H_Wxd4HkrFVW_Yqup
htNL1JciQJYm2DSu9dbetqPZ80x6IBargY850mBYOzvNNE5S-dRJQH0JY4SG-ESYF
BuAhtBlOMgbIOXNi96EegA-vPW9XRF-SHdX5mkafefDGK4rT_RoE4grWhDM3jBz8
1-F2ZA_GpNVEvB-25_vF961Q",
    "e": "AQAB",
    "d": "k_v_h7Jo-TvUt44X6jSax-pTdBHrljklzSYxGEE4yIBbmMVe-Gl2ECkTLe
nNbnafo4RGJ_Vgxkk7PEZO-p7Uz5OBtX-rf83tCgihEyg8aaFIZ-h1_xY9Pqr5uGA
MQGNJaoVMsLb99QNNZhE4JTquP56mVvDQaI3Zn6bhhA0ZqpxS_x6iRUV5KnHCRd47
DKGHcAsQR_caxGec7M_XNPqpl0tcoGQz46-I0SVVcqtjb_YysEvez4eJhb_ZTU4C7
pz4wXDj6B0ppFJVZEMaIhKo8FCnoQdXEJl4vSiBzUFRSlPc6gjQk2CBhxc_kb782w
VvW5wtgOVxhV1k2piQ2NrloQ",
    "p": "6osXfrJLiPfk0ky17vqzRs-M5mOxZU2LEFGyHTXxx6EYTWixEsx2Sdf6kx
UD5K_QdYnqgd3yobbGdDpMwwEuwgogWVCz90nc9QdCUy4MCpz8lpOQdli_tXMmtOg
GBu9mapMTEOwc7HlLlSDRezV_TzTAH4izv-CUEZ_M5EcwyFM",
    "q": "6FYD6NV4rKU1ACTGQyIvWGrkrS_F6phB-whx0nSVFkbbkwpJiPnC6XqjZ3
OYPCZylxaTHFnxCs0nntrraeNEcWfNPrpTN5XIZjbOiIaKA2iCQkWLdoi8oduEtTK
oIcuy32oBz6MpUeCWz10ZQ4EeTF3RyCc_jpf8oRvZ0e3ItHc",
    "dp": "hfjbe9BmWx-HqCaPSanEW-9UQYmym_X2OGUiA5N7vxci5ZymgOFvs_B9v
iQj7C4NOgaEl3EjFgJsS5m9nSoAxm-4WKxDkD6NyxxRYugLkshnc69otvNnlkKnWn
CqeK2o57mJC4KDZwRGczIKloTH6jtsfta8Lh8fFQ4doEuV7uc",
    "dq": "r6R_ViE0FoJalaLhflU09mmZMViBbkXm86nBqtHZ97pmrLvJRdVTxgCh0
c6w0yBZluEJHBDeYkSoZE6qVCWtE3Le1kI0MTx6ANQENXBInCUA_Kr8Ck3TFSYIYJ
fIRaxiMMZKUjfoQAji2WXGeKL_TcpLkt4hDWLXaNDOTgdOiSc",
    "qi": "DfhtLB1Ox1Kgp3E4jqy5Qxeb7-v7_uv8n_5-E1OQ3NLSRV2m_auojkR19
nY3gokHKNSXM41qKlJLU001ROjOO2KUq57s8GZkheVfbJLNCJ6KAw_aRT2IgyJm2b
e2v5OCHSkm88tgJWbtKj-OPKTFV5gOMVdeCzGX286ErjDHGCM"}}}
```

Figure 7

The body of the test message is the UTF8 representation of the following string:

"This is a test long enough to require multiple blocks"

Figure 8

The EDS sequences, are the UTF8 representation of the following strings:

"Subject: Message metadata should be encrypted"
 "2018-02-01"

Figure 9

9.1. Plaintext Message

A plaintext message without associated EDS sequences is an empty header followed by the message body:

```
{
  "DAREMessage": [{},
    "VGhpcyBpcyBhIHRlc3QgbG9uZyBlbm91Z2ggdG8gcmVxdWlyZSBtdWx0aXBsZS
    BibG9ja3M"
  ]}
```

Figure 10

9.2. Plaintext Message with EDS

If a plaintext message contains EDS sequences, these are also in plaintext:

```
{
  "DAREMessage": [{
    "Annotations": ["iAEBiC1TdWJqZWN0OiBNZXNzYWdlIG1ldGFkYXRhIHNo3
    VsZCBiZSB1bmNyeXB0ZWSIAA",
      "iAECiAoyMDE4LTAyLTAxIAA"
    ]},
    "VGhpcyBpcyBhIHRlc3QgbG9uZyBlbm91Z2ggdG8gcmVxdWlyZSBtdWx0aXBsZS
    BibG9ja3M"
  ]}
```

Figure 11

9.3. Encrypted Message

The creator generates a master session key:

```
94 D7 0B C4  8C 43 B2 0E  7C 2A F8 C2  37 9C 8F E5
CA A7 8F BC  4C B3 9F CB  14 AA 5F 4C  41 AF 52 4B
```

Figure 12

The creator generates an ephemeral key:

```
{
  "PrivateKeyDH":{
    "kid":"MAK3V-IOKWY-NFGD7-YBPY3-OWINU-3WRPC-A",
    "Domain":"YE6bnq1MlX5ojaJto6PLP_PEWa",
    "Public":"aeZpaolEpPIXrAFheMEHjjxJMucxpFJ5LvDXJunryv0xftpDseipYF
8jng0pBAE-P7CDPdw_WQgYDdW4NgF7BUbYq1EaXbcVluLYBB2Yw2MAj-Up-7p0lFc
5kjiDnRz0Sd0lJARCTkR5A2cuyMvuEg8cntCQWcoyZh4ep1PktlWDKoSKltov00cM
-9hj-uGFiBV6-cb15fkQ7pyKp-XiH_2YkMCiUYhkPx9ZObrlWNNMVHn9TyLgufPnLH
skaE2JDFckEBMljwXdI9z_DeUx4FNESRQQ68UGfDhfepPwR3_xbL9OFbFiJSjdd51
pxfzvi2SbJm8uKY5K3omsUMszyqjQA",
    "Private":"iXfaIGr7vRVqJyTZEj_5F-3n7DL5KZzQ0sL4wydy1-F27A6kHjBW
yOrV0qr9BLuLRGH4vvfYSael_ILWzgKfC22KqN1CjJdzChwZvFyQCCXynrkH06Kak
uDWpczlC0n8T8WvTboalzVNvOfVM__QcywomDoY6tUCbIn6fQ5xJyWaR8EGQHPFdF
7W-O0ezKIieSKC25fIkIOt3c2-dqQdzUl2Vlk6R_6wkFxxX2NNqpV7VktgrglQ6AUJ
b8gFt6H7gmsSvdks8lXv-1VHwwH_8HBRAOcqsRu9THuz8T4H30d30FZ-NXtCpUEwi
citGRXZlUzmChc_IuIkfdQfuAGvttAA"}}}
```

Figure 13

The key agreement value is calculated:

A2	CD	5A	08	24	70	27	5F	6F	28	A5	6D	B0	AA	47	31
50	D0	F3	DA	07	13	4A	72	F7	BB	19	8E	72	60	82	51
32	0B	CF	7B	A1	8A	FD	40	7E	D5	E1	79	87	20	8B	2A
73	F5	20	2D	1C	94	FB	2D	8D	4F	C0	DA	6D	D7	C0	7A
D1	C6	35	A8	AD	D2	DD	BF	F9	19	C3	BB	60	20	04	F7
D7	F0	81	F6	F3	94	68	DE	6F	B6	B4	67	CD	9B	F3	E4
A0	28	6E	59	5A	A2	FE	00	10	17	B3	34	2C	07	20	06
2C	B9	34	F0	4C	C8	E9	C1	CA	80	1D	02	15	B6	CE	D4
EC	BB	91	2B	FA	7D	9B	14	24	13	16	46	1C	D2	5B	12
4A	0E	60	28	D6	38	E1	35	79	D6	DF	66	C0	C6	7F	A7
E3	C1	9F	25	CD	01	A5	52	7A	C1	B5	ED	3C	24	0F	8F
DD	E6	27	60	F7	2D	CF	D5	7E	13	F8	29	53	7B	43	FC
13	F6	7B	55	8A	44	AD	16	A9	27	75	E0	9A	DF	95	F6
F6	2C	D5	26	88	1B	EB	20	EE	26	C6	4E	17	20	54	BA
DB	A5	37	A4	99	A2	FC	49	93	DE	F5	8F	F5	3B	D1	F3
FF	9D	71	38	B1	AE	94	E0	10	61	16	CA	8F	B9	16	1C

Figure 14

The key agreement value is used as the input to a HKDF key derivation function with the info parameter System.Byte[] to create the key used to wrap the master key:

```
40 39 4F 58 34 F4 0F B3 AA 0F 88 7D EE AC 98 1A
91 84 25 3C 74 F6 E4 25 35 DB D6 78 6A ED 6B F5
```

Figure 15

The wrapped master key is:

```
14 98 FB 8C 0A D0 C8 C6 65 D3 9F F5 6B 80 42 96
0A 71 E9 93 F7 14 D2 29 C9 DB 96 FE FA 9A DB 74
29 F4 35 36 BC CF BF 41
```

Figure 16

This information is used to calculate the Recipient information shown in the example below.

To encrypt a message, we first generate a unique salt value:

```
82 D1 8C BA A0 F0 26 5C 7A 35 5F 82 1F 88 35 CD
```

Figure 17

The salt value and master key are used to generate the payload encryption key:

```
D3 74 4A 8E 69 65 A4 71 8B 14 44 AA CC E6 7F 66
07 87 91 3E F3 41 DE 2D DE 5F 4A 7C 19 D5 75 79
```

Figure 18

Since AES is a block cipher, we also require an initialization vector:

```
0A 5A 94 71 54 7B C2 16 E8 BF D2 66 5D 8D E5 BF
```

Figure 19

The output sequence is the encrypted bytes:

```

30 33 D9 A1 12 19 EF 96 1C 43 45 98 50 7B D2 B1
A7 94 C2 C8 1B 52 00 3E DD A2 B4 9F 51 A5 BD 8C
F4 3D 81 15 95 D0 6D D7 19 5F 28 E3 A0 FF EA 26
29 27 78 B7 49 E7 B2 E3 AD A7 8C D1 C0 61 D5 68

```

Figure 20

Since the message is not signed, there is no need for a trailer. The completed message is:

```

{
  "DAREMessage": [{
    "enc": "A256CBC",
    "Salt": "gtGMuqDwJlx6NV-CH4glzQ",
    "recipients": [{
      "kid": "MDAD3-E4BYE-MK6CH-QA2HD-TKRS2-KIX5Y-A",
      "epk": {
        "PublicKeyDH": {
          "kid": "MAK3V-IOKWY-NFGD7-YBPY3-OWINU-3WRPC-A",
          "Domain": "YE6bnqlMlX5ojaJto6PLP_PEWa",
          "Public": "aeZpaolEpPIXrAFheMEHjjxJMucxpFJ5LvDXJunryv0
xfpDseipYF8jng0pBAE-P7CDPdw_WQgYDdW4NgF7BUbyq1EaXbcV1uLYBB2Yw2MAj
-Up-7pO1Fc5kjiDnRz0Sd0lJARCTkR5A2cuyMvuEg8cntCQWcoyZh4ep1PktlWDKo
SKltovO0cM-9hj-uGFiBV6-cb15fkQ7pyKp-XiH_2YkMCiUYhkPx9ZObr1WNMVHn9
TyLgufPnLHskaE2JDFckEBMl jwXdI9z_DeUx4FNESRQQ68UGfDhfepPwR3_xbL9OF
bFiJSjdd5lpxfzvi2SbJm8uKY5K3omsUMszyqjQA" } } },
      "wmk": "FJj7jArQyMZl05_1a4BClgpx6ZP3FNIpyduW_vqa23Qp9DU2vM
-_QQ" }
    ] },
  "MDPZoRIZ75YcQ0WYUHvSsaeUwsgbUgA-3aK0n1GlvYz0PYEVldBt1xlfK0Og_-
omKSd4t0nnsuOtp4zRwGHVaA"
}]

```

Figure 21

9.4. Signed Message

Signed messages specify the digest algorithm to be used in the header and the signature value in the trailer. Note that the digest algorithm is not optional since it serves as notice that a decoder should digest the payload value to enable signature verification.

```
{
  "DAREMessage": [{
    "dig": "S512"},
    "VGhpcyBpcyBhIHRlc3QgbG9uZyBlbm91Z2ggdG8gcmVxdWlyZSBtdWx0aXBsZS
    BibG9ja3M",
    {
      "signatures": [{
        "signature": "s4LxUqNOV1-0uryJCIyFqnKak3EHZuLBaaUrehaRCaZH
        2KqASBftk87fTl73XUeV-0TfvlfV8STP7l0brcWnPTKQgXSMXvuoxqF0n9qcpXubB
        xzdGHdFX-5GeQMFsr8NutBBng-LSZVe2eCb7n29dpCgZ84v5Y4JGzvKHOy8vaU" }
      ],
      "PayloadDigest": "raim8SV5adPbWWn8FMM4mrRAQCO9A2jZ0NZAnFXWlG0x
      F6sWGJbnKSdtIJMmMU_hjarlIPEoY3vy9UdVlH5KAg" }
    ]
  }
```

Figure 22

9.5. Signed and Encrypted Message

A signed and encrypted message is encrypted and then signed. The signer proves knowledge of the payload plaintext by providing the plaintext witness value.

```

{
  "DAREMessage": [{
    "enc": "A256CBC",
    "dig": "S512",
    "Salt": "AB4x8M6bLZjdSr6W9ntB2A",
    "recipients": [{
      "kid": "MDAD3-E4BYE-MK6CH-QA2HD-TKRS2-KIX5Y-A",
      "epk": {
        "PublicKeyDH": {
          "kid": "MDJLN-DFSIO-ZOL6P-2NOZD-YHBQH-3JQSK-A",
          "Domain": "YE6bnq1MlX5ojaJto6PLP_PEWa",
          "Public": "XBYmS2ui9AgETUvwK9d3eLWQiQ8240yddATCETYAAUH
H0m29aDiBpIT9TcDVs6dtNWH-mnZMTfwT_RQJvyMcfM1AYFvmMfu3GoFS6Z4nngor
hZGFNGgWEG7yX9eOvaiBWpTs2gw7AM4PvUh9r5G0IolfIvQcTRZHpKuU3GqV0E0xp
PhrEajKzLSrDNu6tZ2yDe7BQBSI7YNgZGeXqOTr3KmfdfYU7Hk_ak3M_LhtCCe2-m
bwPuDUD0PUWScZLrKbMWzLIzO0OrDYMwjSrJmirH2RxQKM2a_kJAQoBFtmXjlaFLC
yvOKK1libPnWn7_JZtUy_k4IvDex6b7VKVap6ySQ" } } },
      "wmk": "qSlrvoKhPj1Vy3B6uYkgRYYTJtOzDEeiqqezpTAoGh8Op2VV6x
cbUw" }
    ] },
  "MudZFUAZRBINTZO9szFJCr9NUrE1slmeuZfrftorYK5_gI8-NAjLud8Jx8LM_R
DVNXfi5GKGNFuvfr0MlqyFRQ",
  {
    "signatures": [{
      "signature": "SAzZxGhqUVUSQcta3vZ10L9AkdLD40sVSE9k2opol_FW
4aEyMEM4c5UZTnkf6Ndkz1047TrjHowzagSwkTdq-dV7r70xH-oQ9fTeG1GyT8xwo
Yw4KsSjD71X_lwgi2BI-WMvdVliOFk78MC52eF7SsA0mkbWnSSB0WHBjoJvEaI",
      "witness": "zvxEBQmdHJCin9brH4lpD-8CNJXWezsSMhWAT-CS1k0" }
    ],
    "PayloadDigest": "ScoVguz7ufoe547gh9LIC00ptI24ZzpwLplJFzeJQx9d
G7LfeBTi2oNPr2GBuLp29pRRuDqgdRCftuLeRB18kg" }
  ] }

```

Figure 23

10. References

10.1. Normative References

- [draft-hallambaker-jsonbcd]
Hallam-Baker, P., "Binary Encodings for JavaScript Object Notation: JSON-B, JSON-C, JSON-D", draft-hallambaker-jsonbcd-13 (work in progress), July 2018.
- [draft-hallambaker-udf]
Hallam-Baker, P., "Uniform Data Fingerprint (UDF)", draft-hallambaker-udf-10 (work in progress), April 2018.

- [IANAJOSE] "[Reference Not Found!]".
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
- [RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, DOI 10.17487/RFC2315, March 1998.
- [RFC3394] Schaad, J. and R. Housley, "Advanced Encryption Standard (AES) Key Wrap Algorithm", RFC 3394, DOI 10.17487/RFC3394, September 2002.
- [RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015.

10.2. Informative References

- [Davis2001] Davis, D., "Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML", May 2001.

10.3. URIs

[1] <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html>

[2] <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html>

[3] <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html>

[4] <http://mathmesh.com/Documents/draft-hallambaker-dare-message.html>

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 25, 2018

P. Hallam-Baker
Comodo Group Inc.
May 24, 2018

Binary Encodings for JavaScript Object Notation: JSON-B, JSON-C, JSON-D
draft-hallambaker-jsonbcd-12

Abstract

Three binary encodings for JavaScript Object Notation (JSON) are presented. JSON-B (Binary) is a strict superset of the JSON encoding that permits efficient binary encoding of intrinsic JavaScript data types. JSON-C (Compact) is a strict superset of JSON-B that supports compact representation of repeated data strings with short numeric codes. JSON-D (Data) supports additional binary data types for integer and floating-point representations for use in scientific applications where conversion between binary and decimal representations would cause a loss of precision.

This document is also available online at
<http://mathmesh.com/Documents/draft-hallambaker-jsonbcd.html> [1] .

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 November 25, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Objectives	3
2. Definitions	4
2.1. Requirements Language	4
2.2. Defined Terms	4
2.3. Related Specifications	4
2.4. Terminology	5
3. Extended JSON Grammar	5
4. JSON-B	7
4.1. JSON-B Examples	9
5. JSON-C	10
5.1. JSON-C Examples	11
6. JSON-D (Data)	12
7. JBCD Frames and Records	13
8. Acknowledgements	14
9. Security Considerations	14
10. IANA Considerations	15
11. References	15
11.1. Normative References	15
11.2. Informative References	15
11.3. URIs	15
Author's Address	15

1. Introduction

JavaScript Object Notation (JSON) is a simple text encoding for the JavaScript Data model that has found wide application beyond its original field of use. In particular JSON has rapidly become a preferred encoding for Web Services.

JSON encoding supports just four fundamental data types (integer, floating point, string and boolean), arrays and objects which consist of a list of tag-value pairs.

Although the JSON encoding is sufficient for many purposes it is not always efficient. In particular there is no efficient representation for blocks of binary data. Use of base64 encoding increases data volume by 33%. This overhead increases exponentially in applications

where nested binary encodings are required making use of JSON encoding unsatisfactory in cryptographic applications where nested binary structures are frequently required.

Another source of inefficiency in JSON encoding is the repeated occurrence of object tags. A JSON encoding containing an array of a hundred objects such as {"first":1,"second":2} will contain a hundred occurrences of the string "first" (seven bytes) and a hundred occurrences of the string "second" (eight bytes). Using two byte code sequences in place of strings allows a saving of 11 bytes per object without loss of information, a saving of 50%.

A third objection to the use of JSON encoding is that floating point numbers can only be represented in decimal form and this necessarily involves a loss of precision when converting between binary and decimal representations. While such issues are rarely important in network applications they can be critical in scientific applications. It is not acceptable for saving and restoring a data set to change the result of a calculation.

1.1. Objectives

The following were identified as core objectives for a binary JSON encoding:

- o Easy to convert existing encoders and decoders to add binary support
- o Efficient encoding of binary data
- o Ability to convert from JSON to binary encoding in a streaming mode (i.e. without reading the entire binary data block before beginning encoding.
- o Lossless encoding of JavaScript data types
- o The ability to support JSON tag compression and extended data types are considered desirable but not essential for typical network applications.

Three binary encodings are defined:

JSON-B (Binary) Encodes JSON data in binary. Only the JavaScript data model is supported (i.e. atomic types are integers, double or string). Integers may be 8, 16, 32 or 64 bits either signed or unsigned. Floating points are IEEE 754 binary64 format [IEEE754]. Supports chunked encoding for binary and UTF-8 string types.

JSON-C (Compact) As JSON-B but with support for representing JSON tags in numeric code form (16 bit code space). This is done for both compact encoding and to allow simplification of encoders/decoders in constrained environments. Codes may be defined inline or by reference to a known dictionary of codes referenced via a digest value.

JSON-D (Data) As JSON-C but with support for representing additional data types without loss of precision. In particular other IEEE 754 floating point formats, both binary and decimal and Intel's 80 bit floating point, plus 128 bit integers and bignum integers.

Each encoding is a proper superset of JSON, JSON-C is a proper superset of JSON-B and JSON-D is a proper superset of JSON-C. Thus a single decoder MAY be used for all three new encodings and for JSON. Figure 1 shows these relationships graphically:

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-jsonbcd.html> [2].]]

Encoding Relationships.

2. Definitions

This section presents the related specifications and standard, the terms that are used as terms of art within the documents and the terms used as requirements language.

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2.2. Defined Terms

The terms of art used in this document are described in the Mesh Architecture Guide [draft-hallambaker-mesh-architecture] .

2.3. Related Specifications

The JSON-B, JSON-C and JSON-D encodings are all based on the JSON grammar [RFC7159] . IEEE 754 Floating Point Standard is used for encoding floating point numbers [IEEE754] ,

2.4. Terminology

No new terms of art are defined

3. Extended JSON Grammar

The JSON-B, JSON-C and JSON-D encodings are all based on the JSON grammar [RFC7159] using the same syntactic structure but different lexical encodings.

JSON-B0 and JSON-C0 replace the JSON lexical encodings for strings and numbers with binary encodings. JSON-B1 and JSON-C1 allow either lexical encoding to be used. Thus any valid JSON encoding is a valid JSON-B1 or JSON-C1 encoding.

The grammar of JSON-B, JSON-C and JSON-D is a superset of the JSON grammar. The following productions are added to the grammar:

x-value Binary encodings for data values. As the binary value encodings are all self delimiting

x-member An object member where the value is specified as an X-value and thus does not require a value-separator.

b-value Binary data encodings defined in JSON-B.

b-string Defined length string encoding defined in JSON-B.

c-def Tag code definition defined in JSON-C. These may only appear before the beginning of an Object or Array and before any preceding white space.

c-tag Tag code value defined in JSON-C.

d-value Additional binary data encodings defined in JSON-D for use in scientific data applications.

The JSON grammar is modified to permit the use of x-value productions in place of (value value-separator) :

```

JSON-text = (object / array)

object = *cdef begin-object [
    *( member value-separator | x-member )
    (member | x-member) ] end-object

member = tag value
x-member = tag x-value

tag = string name-separator | b-string | c-tag

array = *cdef begin-array [ *( value value-separator | x-value )
    (value | x-value) ] end-array

x-value = b-value / d-value

value = false / null / true / object / array / number / string

name-separator = ws %x3A ws ; : colon
value-separator = ws %x2C ws ; , comma

```

Figure 1

The following lexical values are unchanged:

```

begin-array      = ws %x5B ws ; [ left square bracket
begin-object     = ws %x7B ws ; { left curly bracket
end-array        = ws %x5D ws ; ] right square bracket
end-object       = ws %x7D ws ; } right curly bracket

ws = *( %x20 %x09 %x0A %x0D )

false = %x66.61.6c.73.65 ; false
null  = %x6e.75.6c.6c    ; null
true  = %x74.72.75.65    ; true

```

Figure 2

The productions number and string are defined as before:

```

number = [ minus ] int [ frac ] [ exp ]
decimal-point = %x2E          ; .
digit1-9 = %x31-39           ; 1-9
e = %x65 / %x45              ; e E
exp = e [ minus / plus ] 1*DIGIT
frac = decimal-point 1*DIGIT
int = zero / ( digit1-9 *DIGIT )
minus = %x2D                  ; -
plus = %x2B                   ; +
zero = %x30                   ; 0

string = quotation-mark *char quotation-mark
char = unescaped /
escape ( %x22 / %x5C / %x2F / %x62 / %x66 /
%x6E / %x72 / %x74 / %x75 4HEXDIG )

escape = %x5C                  ; \
quotation-mark = %x22          ; "
unescaped = %x20-21 / %x23-5B / %x5D-10FFFF

```

Figure 3

4. JSON-B

The JSON-B encoding defines the b-value and b-string productions:

```

b-value = b-atom | b-string | b-data | b-integer |
b-float

b-string = *( string-chunk ) string-term
b-data = *( data-chunk ) data-last

b-integer = p-int8 | p-int16 | p-int32 | p-int64 | p-bignum16 |
n-int8 | n-int16 | n-int32 | n-int64 | n-bignum16

b-float = binary64

```

Figure 4

The lexical encodings of the productions are defined in the following tables where the column 'tag' specifies the byte code that begins the production, 'Fixed' specifies the number of data bytes that follow and 'Length' specifies the number of bytes used to define the length of a variable length field following the data bytes:

Production	Tag	Fixed	Length	Data Description
string-term	x80	-	1	Terminal String 8 bit length
string-term	x81	-	2	Terminal String 16 bit length
string-term	x82	-	4	Terminal String 32 bit length
string-term	x83	-	8	Terminal String 64 bit length
string-chunk	x84	-	1	Terminal String 8 bit length
string-chunk	x85	-	2	Terminal String 16 bit length
string-chunk	x86	-	4	Terminal String 32 bit length
string-chunk	x87	-	8	Terminal String 64 bit length
data-term	x88	-	1	Terminal String 8 bit length
data-term	x89	-	2	Terminal String 16 bit length
data-term	x8A	-	4	Terminal String 32 bit length
data-term	x8B	-	8	Terminal String 64 bit length
data-term	X8C	-	1	Terminal String 8 bit length
data-term	x8D	-	2	Terminal String 16 bit length
data-term	x8E	-	4	Terminal String 32 bit length
data-term	x8F	-	8	Terminal String 64 bit length

Table 1

Table 1: Codes for String and Data items

Production	Tag	Fixed	Length	Data Description
p-int8	xA0	1	-	Positive 8 bit Integer
p-int16	xA1	2	-	Positive 16 bit Integer
p-int32	xA2	4	-	Positive 32 bit Integer
p-int64	xA3	8	-	Positive 64 bit Integer
p-bignum16	xA7	-	2	Positive Bignum
n-int8	xA8	1	-	Negative 8 bit Integer
n-int16	xA9	2	-	Negative 16 bit Integer
n-int32	xAa	4	-	Negative 32 bit Integer
n-int64	xAAB	8	-	Negative 64 bit Integer
n-bignum16	xAf	-	2	Negative Bignum
binary64	x92	8	-	IEEE 754 Floating Point Binary 64 bit
b-value	xB0	-	-	True
b-value	xB1	-	-	False
b-value	xB2	-	-	Null

Table 2

Table 2: Codes for Integers, 64 Bit Floating Point, Boolean and Null items.

A data type commonly used in networking that is not defined in this scheme is a datetime representation. To define such a data type, a string containing a date-time value in Internet type format is typically used.

4.1. JSON-B Examples

The following examples show examples of using JSON-B encoding:

A0 2A	42 (as 8 bit integer)
A1 00 2A	42 (as 16 bit integer)
A2 00 00 00 2A	42 (as 32 bit integer)
A3 00 00 00 00 00 00 00 2A	42 (as 64 bit integer)
A5 00 01 42	42 (as Bignum)
80 05 48 65 6c 6c 6f	"Hello" (single chunk)
81 00 05 48 65 6c 6c 6f	"Hello" (single chunk)
84 05 48 65 6c 6c 6f 80 00	"Hello" (as two chunks)
92 3f f0 00 00 00 00 00 00	1.0
92 40 24 00 00 00 00 00 00	10.0
92 40 09 21 fb 54 44 2e ea	3.14159265359
92 bf f0 00 00 00 00 00 00	-1.0
B0	true
B1	false
B2	null

Figure 5

5. JSON-C

JSON-C (Compressed) permits numeric code values to be substituted for strings and binary data. Tag codes MAY be 8, 16 or 32 bits long encoded in network byte order.

Tag codes MUST be defined before they are referenced. A Tag code MAY be defined before the corresponding data or string value is used or at the same time that it is used.

A dictionary is a list of tag code definitions. An encoding MAY incorporate definitions from a dictionary using the dict-hash production. The dict hash production specifies a (positive) offset value to be added to the entries in the dictionary followed by the UDF fingerprint [draft-hallambaker-udf] of the dictionary to be used.

Production	Tag	Fixed	Length	Data Description
c-tag	xC0	1	-	8 bit tag code
c-tag	xC1	2	-	16 bit tag code
c-tag	xC2	4	-	32 bit tag code
c-def	xC4	1	-	8 bit tag definition
c-def	xC5	2	-	16 bit tag definition
c-def	xC6	4	-	32 bit tag definition
c-tag	xC8	1	-	8 bit tag code and definition
c-tag	xC9	2	-	16 bit tag code and definition
c-tag	xCA	4	-	32 bit tag code and definition
c-def	xCC	1	-	8 bit tag dictionary definition
c-def	xCD	2	-	16 bit tag dictionary definition
c-def	xCE	4	-	32 bit tag dictionary definition
dict-hash	xD0	4	1	UDF fingerprint of dictionary

Table 3

Table 3: Codes Used for Compression

All integer values are encoded in Network Byte Order (most significant byte first).

5.1. JSON-C Examples

The following examples show examples of using JSON-C encoding:

```

C8 20 80 05 48 65 6c 6c 6f      "Hello"      20 = "Hello"
C4 21 80 05 48 65 6c 6c 6f      21 = "Hello"
C0 20                          "Hello"
C1 00 20                        "Hello"

D0 00 00 01 00 20              Insert dictionary at code 256
e3 b0 c4 42 98 fc 1c 14
9a fb f4 c8 99 6f b9 24
27 ae 41 e4 64 9b 93 4c
a4 95 99 1b 78 52 b8 55        UDF (C4 21 80 05 48 65 6c 6c 6f)

```

Figure 6

6. JSON-D (Data)

JSON-B and JSON-C only support the two numeric types defined in the JavaScript data model: Integers and 64 bit floating point values. JSON-D (Data) defines binary encodings for additional data types that are commonly used in scientific applications. These comprise positive and negative 128 bit integers, six additional floating point representations defined by IEEE 754 [IEEE754] and the Intel extended precision 80 bit floating point representation [INTEL] .

Should the need arise, even bigger bignums could be defined with the length specified as a 32 bit value permitting bignums of up to 2^{35} bits to be represented.

d-value = d-integer | d-float

d-float = binary16 | binary32 | binary128 | binary80 |
decimal32 | decimal64 | decimal 128

Figure 7

The codes for these values are as follows:

Production	Tag	Fixed	Length	Data Description
p-int128	xA4	16	-	Positive 128 bit Integer
n-int128	xAC	16	-	Negative 128 bit Integer
binary16	x90	2	-	IEEE 754 Floating Point Binary 16 bit
binary32	x91	4	-	IEEE 754 Floating Point Binary 32 bit
binary128	x94	16	-	IEEE 754 Floating Point Binary 64 bit
Intel80	x95	10	-	Intel extended Floating Point 80 bit
decimal32	x96	4	-	IEEE 754 Floating Point Decimal 32
Decimal64	x97	8	-	IEEE 754 Floating Point Decimal 64
Decimal128	x98	16	-	IEEE 754 Floating Point Decimal 128

Table 4

Table 4: Additional Codes for Scientific Data

7. JBCD Frames and Records

Tag codes in the range xF0-xFF are reserved for specifying markers for frames and records. These tags are not used to encode JSON data, they are only used to encapsulate opaque binary data blobs as a unit.

A JBCD record consists of consist of the tag, a length and the data item. The length indication provided by the record format allows efficient traversal of a sequence of records in the forward direction only.

A JBCD Frames consists of consist of the tag, a length and the data item followed by the tag-length sequence repeated with the bytes written in the reverse order. The first length indication allows efficient traversal of a sequence of records in the forward direction and the second allows efficient traversal in the reverse direction.

[[This figure is not viewable in this format. The figure is available at <http://mathmesh.com/Documents/draft-hallambaker-jsonbcd.html> [3].]]

JBCD Records and Frames

The JBCD-Frame tags currently defined are:

Production	Tag	Fixed	Length	Data Description
uframe	xF0	-	1	Record, 8 bit length
uframe	xF1	-	2	Record, 16 bit length
uframe	xF2	-	4	Record, 32 bit length
uframe	xF3	-	8	Record, 64 bit length
bframe	xF4	-	1	Frame, 8 bit length
bframe	xF5	-	2	Frame, 16 bit length
bframe	xF6	-	4	Frame, 32 bit length
bframe	xF7	-	8	Frame, 64 bit length
	xF8-xFF	-	-	Reserved

Table 5

The author does not expect additional framing tags to be added but codes F8-FF are reserved in case this is desired.

It may prove convenient to represent message digest values as large integers rather than binary strings. While very few platforms or programming languages support mathematical operations on fixed size

integers larger than 64, this is not a major concern since message digests are rarely used for any purpose other than comparison for equality.

Production	Tag	Fixed	Length	Data Description
p-int128	Xa4	16	-	Positive 128 bit Integer
p-int256	Xa5	32	-	Positive 256 bit Integer
p-int512	Xa6	64	-	Positive 512 bit Integer

Table 6

8. Acknowledgements

This work was assisted by conversations with Nico Williams and other participants on the applications area mailing list.

9. Security Considerations

A correctly implemented data encoding mechanism should not introduce new security vulnerabilities. However, experience demonstrates that some data encoding approaches are more prone to introduce vulnerabilities when incorrectly implemented than others.

In particular, whenever variable length data formats are used, the possibility of a buffer overrun vulnerability is introduced. While best practice suggests that a coding language with native mechanisms for bounds checking is the best protection against such errors, such approaches are not always followed. While such vulnerabilities are most commonly seen in the design of decoders, it is possible for the same vulnerabilities to be exploited in encoders.

A common source of such errors is the case where nested length encodings are used. For example, a decoder relies on an outermost length encoding that specifies a length on 50 bytes to allocate memory for the entire result and then attempts to copy a string with a declared length of 1000 bytes within the sequence.

The extensions to the JSON encoding described in this document are designed to avoid such errors. Length encodings are only used to define the length of x-value constructions which are always terminal and cannot have nested data entries.

10. IANA Considerations

[TBS list out all the code points that require an IANA registration]

11. References

11.1. Normative References

- [draft-hallambaker-udf]
Hallam-Baker, P., "Uniform Data Fingerprint (UDF)", draft-hallambaker-udf-10 (work in progress), April 2018.
- [IEEE754] IEEE Computer Society, "IEEE Standard for Floating-Point Arithmetic", IEEE 754-2008, DOI 10.1109/IEEESTD.2008.4610935, August 2008.
- [INTEL] Intel Corp., "Unknown".
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014.

11.2. Informative References

- [draft-hallambaker-mesh-architecture]
Hallam-Baker, P., "Mathematical Mesh: Architecture", draft-hallambaker-mesh-architecture-04 (work in progress), September 2017.

11.3. URIs

- [1] <http://mathmesh.com/Documents/draft-hallambaker-jsonbcd.html>
- [2] <http://mathmesh.com/Documents/draft-hallambaker-jsonbcd.html>
- [3] <http://mathmesh.com/Documents/draft-hallambaker-jsonbcd.html>

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 22 October 2022

P. M. Hallam-Baker
20 April 2022

Binary Encodings for JavaScript Object Notation: JSON-B, JSON-C, JSON-D
draft-hallambaker-jsonbcd-22

Abstract

Three binary encodings for JavaScript Object Notation (JSON) are presented. JSON-B (Binary) is a strict superset of the JSON encoding that permits efficient binary encoding of intrinsic JavaScript data types. JSON-C (Compact) is a strict superset of JSON-B that supports compact representation of repeated data strings with short numeric codes. JSON-D (Data) supports additional binary data types for integer and floating-point representations for use in scientific applications where conversion between binary and decimal representations would cause a loss of precision.

This document is also available online at
<http://mathmesh.com/Documents/draft-hallambaker-jsonbcd.html>.

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 22 October 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction	2
1.1. Objectives	3
2. Definitions	4
2.1. Requirements Language	4
2.2. Defined Terms	4
2.3. Related Specifications	5
2.4. Terminology	5
3. Extended JSON Grammar	5
4. JSON-B	7
4.1. JSON-B Examples	9
5. JSON-C	10
5.1. JSON-C Examples	11
6. JSON-D (Data)	12
7. JBCD Frames and Records	13
8. JSON-BCD Application Binding	15
8.1. JSON Binding	15
8.2. JSON-B Binding	16
8.3. Examples	17
9. Acknowledgements	17
10. Security Considerations	17
11. IANA Considerations	18
12. Normative References	18
13. Informative References	19

1. Introduction

JavaScript Object Notation (JSON) is a simple text encoding for the JavaScript Data model that has found wide application beyond its original field of use. In particular JSON has rapidly become a preferred encoding for Web Services.

JSON encoding supports just four fundamental data types (integer, floating point, string and boolean), arrays and objects which consist of a list of tag-value pairs.

Although the JSON encoding is sufficient for many purposes it is not always efficient. In particular there is no efficient representation for blocks of binary data. Use of base64 encoding increases data volume by 33%. This overhead increases exponentially in applications where nested binary encodings are required making use of JSON encoding unsatisfactory in cryptographic applications where nested binary structures are frequently required.

Another source of inefficiency in JSON encoding is the repeated occurrence of object tags. A JSON encoding containing an array of a hundred objects such as {"first":1,"second":2} will contain a hundred occurrences of the string "first" (seven bytes) and a hundred occurrences of the string "second" (eight bytes). Using two byte code sequences in place of strings allows a saving of 11 bytes per object without loss of information, a saving of 50%.

A third objection to the use of JSON encoding is that floating point numbers can only be represented in decimal form and this necessarily involves a loss of precision when converting between binary and decimal representations. While such issues are rarely important in network applications they can be critical in scientific applications. It is not acceptable for saving and restoring a data set to change the result of a calculation.

1.1. Objectives

The following were identified as core objectives for a binary JSON encoding:

- * Easy to convert existing encoders and decoders to add binary support
- * Efficient encoding of binary data
- * Ability to convert from JSON to binary encoding in a streaming mode (i.e. without reading the entire binary data block before beginning encoding).
- * Lossless encoding of JavaScript data types
- * The ability to support JSON tag compression and extended data types are considered desirable but not essential for typical network applications.

Three binary encodings are defined:

JSON-B (Binary) Encodes JSON data in binary. Only the JavaScript

data model is supported (i.e. atomic types are integers, double or string). Integers may be 8, 16, 32 or 64 bits either signed or unsigned. Floating points are IEEE 754 binary64 format [IEEE754]. Supports chunked encoding for binary and UTF-8 string types.

JSON-C (Compact) As JSON-B but with support for representing JSON tags in numeric code form (16 bit code space). This is done for both compact encoding and to allow simplification of encoders/decoders in constrained environments. Codes may be defined inline or by reference to a known dictionary of codes referenced via a digest value.

JSON-D (Data) As JSON-C but with support for representing additional data types without loss of precision. In particular other IEEE 754 floating point formats, both binary and decimal and Intel's 80 bit floating point, plus 128 bit integers and bignum integers.

Each encoding is a proper superset of JSON, JSON-C is a proper superset of JSON-B and JSON-D is a proper superset of JSON-C. Thus a single decoder MAY be used for all three new encodings and for JSON. Figure 1 shows these relationships graphically:

(Artwork only available as svg: No external link available, see [draft-hallambaker-jsonbcd-22.html](#) for artwork.)

Figure 1: Encoding Relationships.

2. Definitions

This section presents the related specifications and standard, the terms that are used as terms of art within the documents and the terms used as requirements language.

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2.2. Defined Terms

The terms of art used in this document are described in the [_Mesh Architecture Guide_ \[draft-hallambaker-mesh-architecture\].](#)

2.3. Related Specifications

The JSON-B, JSON-C and JSON-D encodings are all based on the JSON grammar [RFC7159]. IEEE 754 Floating Point Standard is used for encoding floating point numbers [IEEE754],

2.4. Terminology

No new terms of art are defined

3. Extended JSON Grammar

The JSON-B, JSON-C and JSON-D encodings are all based on the JSON grammar [RFC7159] using the same syntactic structure but different lexical encodings.

JSON-B0 and JSON-C0 replace the JSON lexical encodings for strings and numbers with binary encodings. JSON-B1 and JSON-C1 allow either lexical encoding to be used. Thus, any valid JSON encoding is a valid JSON-B1 or JSON-C1 encoding.

The grammar of JSON-B, JSON-C and JSON-D is a superset of the JSON grammar. The following productions are added to the grammar:

x-value Binary encodings for data values. As the binary value encodings are all self-delimiting

x-member An object member where the value is specified as an X-value and thus does not require a value-separator.

b-value Binary data encodings defined in JSON-B.

b-string Defined length string encoding defined in JSON-B.

c-def Tag code definition defined in JSON-C. These may only appear before the beginning of an Object or Array and before any preceding white space.

c-tag Tag code value defined in JSON-C.

d-value Additional binary data encodings defined in JSON-D for use in scientific data applications.

The JSON grammar is modified to permit the use of x-value productions in place of (value value-separator) :

```
JSON-text = (object / array)

object = *cdef begin-object [
    *( member value-separator | x-member )
    (member | x-member) ] end-object

member = tag value
x-member = tag x-value

tag = string name-separator | b-string | c-tag

array = *cdef begin-array [ *( value value-separator | x-value )
    (value | x-value) ] end-array

x-value = b-value / d-value

value = false / null / true / object / array / number / string

name-separator = ws %x3A ws ; : colon
value-separator = ws %x2C ws ; , comma
```

The following lexical values are unchanged:

```
begin-array      = ws %x5B ws ; [ left square bracket
begin-object     = ws %x7B ws ; { left curly bracket
end-array        = ws %x5D ws ; ] right square bracket
end-object       = ws %x7D ws ; } right curly bracket
```

```
ws = *( %x20 %x09 %x0A %x0D )
```

```
false = %x66.61.6c.73.65 ; false
null  = %x6e.75.6c.6c    ; null
true  = %x74.72.75.65    ; true
```

The productions number and string are defined as before:

```

number = [ minus ] int [ frac ] [ exp ]
decimal-point = %x2E          ; .
digit1-9 = %x31-39           ; 1-9
e = %x65 / %x45              ; e E
exp = e [ minus / plus ] 1*DIGIT
frac = decimal-point 1*DIGIT
int = zero / ( digit1-9 *DIGIT )
minus = %x2D                  ; -
plus = %x2B                   ; +
zero = %x30                   ; 0

string = quotation-mark *char quotation-mark
char = unescaped /
escape ( %x22 / %x5C / %x2F / %x62 / %x66 /
%x6E / %x72 / %x74 / %x75 4HEXDIG )

escape = %x5C                ; \
quotation-mark = %x22        ; "
unescaped = %x20-21 / %x23-5B / %x5D-10FFFF

```

4. JSON-B

The JSON-B encoding defines the b-value and b-string productions:

```

b-value = b-atom | b-string | b-data | b-integer |
b-float

```

```

b-string = *( string-chunk ) string-term
b-data = *( data-chunk ) data-last

```

```

b-integer = p-int8 | p-int16 | p-int32 | p-int64 | p-bignum16 |
n-int8 | n-int16 | n-int32 | n-int64 | n-bignum16

```

```

b-float = binary64

```

The lexical encodings of the productions are defined in the following tables where the column 'tag' specifies the byte code that begins the production, 'Fixed' specifies the number of data bytes that follow and 'Length' specifies the number of bytes used to define the length of a variable length field following the data bytes:

Production	Tag	Fixed	Length	Data Description
string-term	x80	-	1	Terminal String 8 bit length
string-term	x81	-	2	Terminal String

				16 bit length
string-term	x82	-	4	Terminal String 32 bit length
string-term	x83	-	8	Terminal String 64 bit length
string-chunk	x84	-	1	Terminal String 8 bit length
string-chunk	x85	-	2	Terminal String 16 bit length
string-chunk	x86	-	4	Terminal String 32 bit length
string-chunk	x87	-	8	Terminal String 64 bit length
data-term	x88	-	1	Terminal String 8 bit length
data-term	x89	-	2	Terminal String 16 bit length
data-term	x8A	-	4	Terminal String 32 bit length
data-term	x8B	-	8	Terminal String 64 bit length
data-chunk	X8C	-	1	Terminal String 8 bit length
data-chunk	x8D	-	2	Terminal String 16 bit length
data-chunk	x8E	-	4	Terminal String 32 bit length
data-chunk	x8F	-	8	Terminal String 64 bit length

Table 1

Table 1: Codes for String and Data items

Production	Tag	Fixed	Length	Data Description
p-int8	xA0	1	-	Positive 8 bit Integer
p-int16	xA1	2	-	Positive 16 bit Integer
p-int32	xA2	4	-	Positive 32 bit Integer
p-int64	xA3	8	-	Positive 64 bit Integer
p-bignum16	Xa7	-	2	Positive Bignum
n-int8	xA8	1	-	Negative 8 bit Integer
n-int16	xA9	2	-	Negative 16 bit Integer
n-int32	xAA	4	-	Negative 32 bit Integer
n-int64	xAB	8	-	Negative 64 bit Integer
n-bignum16	xAF	-	2	Negative Bignum
binary64	x92	8	-	IEEE 754 Floating Point Binary 64 bit
b-value	xB0	-	-	True
b-value	xB1	-	-	False
b-value	xB2	-	-	Null

Table 2

Table 2: Codes for Integers, 64 Bit Floating Point, Boolean and Null items.

A data type commonly used in networking that is not defined in this scheme is a datetime representation. To define such a data type, a string containing a date-time value in Internet type format is typically used.

4.1. JSON-B Examples

The following examples show examples of using JSON-B encoding:

A0 2A	42 (as 8 bit integer)
A1 00 2A	42 (as 16 bit integer)
A2 00 00 00 2A	42 (as 32 bit integer)
A3 00 00 00 00 00 00 00 2A	42 (as 64 bit integer)
A5 00 01 42	42 (as Bignum)
80 05 48 65 6c 6c 6f	"Hello" (single chunk)
81 00 05 48 65 6c 6c 6f	"Hello" (single chunk)
84 05 48 65 6c 6c 6f 80 00	"Hello" (as two chunks)
92 3f f0 00 00 00 00 00 00	1.0
92 40 24 00 00 00 00 00 00	10.0
92 40 09 21 fb 54 44 2e ea	3.14159265359
92 bf f0 00 00 00 00 00 00	-1.0
B0	true
B1	false
B2	null

5. JSON-C

JSON-C (Compressed) permits numeric code values to be substituted for strings and binary data. Tag codes MAY be 8, 16 or 32 bits long encoded in network byte order.

Tag codes MUST be defined before they are referenced. A Tag code MAY be defined before the corresponding data or string value is used or at the same time that it is used.

A dictionary is a list of tag code definitions. An encoding MAY incorporate definitions from a dictionary using the dict-hash production. The dict hash production specifies a (positive) offset value to be added to the entries in the dictionary followed by the UDF fingerprint [draft-hallambaker-udf] of the dictionary to be used.

Production	Tag	Fixed	Length	Data Description
c-tag	xC0	1	-	8 bit tag code
c-tag	xC1	2	-	16 bit tag code
c-tag	xC2	4	-	32 bit tag code
c-def	xC4	1	-	8 bit tag definition
c-def	xC5	2	-	16 bit tag definition
c-def	xC6	4	-	32 bit tag definition
c-tag	xC8	1	-	8 bit tag code and definition
c-tag	xC9	2	-	16 bit tag code and definition
c-tag	xCA	4	-	32 bit tag code and definition
c-def	xCC	1	-	8 bit tag dictionary definition
c-def	xCD	2	-	16 bit tag dictionary definition
c-def	xCE	4	-	32 bit tag dictionary definition
dict-hash	xD0	4	1	UDF fingerprint of dictionary

Table 3

Table 3: Codes Used for Compression

All integer values are encoded in Network Byte Order (most significant byte first).

5.1. JSON-C Examples

The following examples show examples of using JSON-C encoding:

```

C8 20 80 05 48 65 6c 6c 6f      "Hello"      20 = "Hello"
C4 21 80 05 48 65 6c 6c 6f      21 = "Hello"
C0 20                             "Hello"
C1 00 20                         "Hello"

D0 00 00 01 00 20                Insert dictionary at code 256
e3 b0 c4 42 98 fc 1c 14
9a fb f4 c8 99 6f b9 24
27 ae 41 e4 64 9b 93 4c
a4 95 99 1b 78 52 b8 55          UDF (C4 21 80 05 48 65 6c 6c 6f)

```

6. JSON-D (Data)

JSON-B and JSON-C only support the two numeric types defined in the JavaScript data model: Integers and 64 bit floating point values. JSON-D (Data) defines binary encodings for additional data types that are commonly used in scientific applications. These comprise positive and negative 128 bit integers, six additional floating point representations defined by IEEE 754 [IEEE754] and the Intel extended precision 80 bit floating point representation [INTEL].

Should the need arise, even bigger bignums could be defined with the length specified as a 32 bit value permitting bignums of up to 2^{35} bits to be represented.

d-value = d-integer | d-float

d-float = binary16 | binary32 | binary128 | binary80 |
decimal32 | decimal64 | decimal 128

The codes for these values are as follows:

Production	Tag	Fixed	Length	Data Description
p-int128	xA4	16	-	Positive 128 bit Integer
n-int128	xAC	16	-	Negative 128 bit Integer
binary16	x90	2	-	IEEE 754 Floating Point Binary 16 bit
binary32	x91	4	-	IEEE 754 Floating Point Binary 32 bit
binary128	x94	16	-	IEEE 754 Floating Point Binary 64 bit
Intel80	x95	10	-	Intel extended Floating Point 80 bit
decimal32	x96	4	-	IEEE 754 Floating Point Decimal 32
Decimal64	x97	8	-	IEEE 754 Floating Point Decimal 64
Decimal128	x98	16	-	IEEE 754 Floating Point Decimal 128

Table 4

Table 4: Additional Codes for Scientific Data

7. JBCD Frames and Records

Tag codes in the range xF0-XFF are reserved for specifying markers for `_frames_` and `_records_`. These tags are not used to encode JSON data, they are only used to encapsulate opaque binary data blobs as a unit.

A JBCD record consists of consist of the tag, a length and the data item. The length indication provided by the record format allows efficient traversal of a sequence of records in the forward direction only.

A JBCD Frames consists of consist of the tag, a length and the data item followed by the tag-length sequence repeated with the bytes written in the reverse order. The first length indication allows efficient traversal of a sequence of records in the forward direction and the second allows efficient traversal in the reverse direction.

(Artwork only available as svg: No external link available, see draft-hallambaker-jsonbcd-22.html for artwork.)

Figure 2: JBCD Records and Frames

The JBCD-Frame tags currently defined are:

Production	Tag	Fixed	Length	Data Description
uframe	xF0	-	1	Record, 8 bit length
uframe	xF1	-	2	Record, 16 bit length
uframe	xF2	-	4	Record, 32 bit length
uframe	xF3	-	8	Record, 64 bit length
bframe	xF4	-	1	Frame, 8 bit length
bframe	xF5	-	2	Frame, 16 bit length
bframe	xF6	-	4	Frame, 32 bit length
bframe	xF7	-	8	Frame, 64 bit length
	xF8-xFF	-	-	Reserved

Table 5

The author does not expect additional framing tags to be added but codes F8-FF are reserved in case this is desired.

It may prove convenient to represent message digest values as large integers rather than binary strings. While very few platforms or programming languages support mathematical operations on fixed size integers larger than 64, this is not a major concern since message digests are rarely used for any purpose other than comparison for equality.

Production	Tag	Fixed	Length	Data Description
p-int128	Xa4	16	-	Positive 128 bit Integer
p-int256	Xa5	32	-	Positive 256 bit Integer
p-int512	Xa6	64	-	Positive 512 bit Integer

Table 6

8. JSON-BCD Application Binding

The JSON serialization format is intentionally limited in the range of data types it supports. To serialize data not supported by the JSON data model, it is necessary to specify a means of transforming the unsupported data type into a supported one.

In particular JSON does not provide a Date-Time object or a means of encoding binary data except as an array of bytes.

The JSON-BCD Application Binding specifies a data model designed to support the needs of specifying network protocols and associated static data. The data model is close to that of JSON with three additional types:

- * Binary data
- * DateTime
- * Object of specified type

8.1. JSON Binding

The following data field types are specified:

Integer

Integer values are encoded as JSON number values.

Float

Float values are encoded as JSON number values.

String

Test strings are encoded as JSON text strings.

Boolean

Boolean values are encoded as JSON 'false', 'true' or 'null' tokens according to value.

Sequence

Sequences of data items that are encoded as JSON arrays

Object of known type

Objects whose type is known to the receiver are encoded as JSON objects

Object of variable type

Objects whose type is not known to the receiver are encoded as JSON objects containing a single field whose name describes the type of the object value and whose value contains the value.

Binary Data

Byte sequences are converted to BASE64-url encoding [RFC4648] and encoded as JSON string values.

Date Time

Date Time values are converted to Internet time format as described in [RFC3339] and encoded as JSON string values.

8.2. JSON-B Binding

The JSON-B value allows a superset of the JSON encodings to be used.

Integer

Integer values are encoded as JSON number values or as JSON-B b-integer values.

Float

Float values are encoded as JSON number values or as JSON-B b-integer or b-float values.

String

Test strings are encoded as JSON text strings or as JSON-B b-string values.

Boolean

Boolean values are encoded as JSON 'false', 'true' or 'null' tokens or as JSON-B b-atom values according to value

Sequence

Sequences of data items that are encoded as JSON arrays

Object of known type

Objects whose type is known to the receiver are encoded as JSON objects

Object of variable type

Objects whose type is not known to the receiver are encoded as JSON objects containing a single field whose name describes the type of the object value and whose value contains the value.

Binary Data

Byte sequences are converted to BASE64-url encoding [RFC4648] and encoded as JSON string values or as JSON-B b-data values.

Date Time

Date Time values are converted to Internet time format as described in [RFC3339] and encoded as JSON string values or as JSON-B b-string values.

8.3. Examples

9. Acknowledgements

This work was assisted by conversations with Nico Williams and other participants on the applications area mailing list.

10. Security Considerations

A correctly implemented data encoding mechanism should not introduce new security vulnerabilities. However, experience demonstrates that some data encoding approaches are more prone to introduce vulnerabilities when incorrectly implemented than others.

In particular, whenever variable length data formats are used, the possibility of a buffer overrun vulnerability is introduced. While best practice suggests that a coding language with native mechanisms for bounds checking is the best protection against such errors, such approaches are not always followed. While such vulnerabilities are most commonly seen in the design of decoders, it is possible for the same vulnerabilities to be exploited in encoders.

A common source of such errors is the case where nested length encodings are used. For example, a decoder relies on an outermost length encoding that specifies a length on 50 bytes to allocate memory for the entire result and then attempts to copy a string with a declared length of 1000 bytes within the sequence.

The extensions to the JSON encoding described in this document are designed to avoid such errors. Length encodings are only used to define the length of x-value constructions which are always terminal and cannot have nested data entries.

11. IANA Considerations

[TBS list out all the code points that require an IANA registration]

12. Normative References

[draft-hallambaker-udf]

Hallam-Baker, P., "Uniform Data Fingerprint (UDF)", Work in Progress, Internet-Draft, draft-hallambaker-udf-12, 6 January 2019, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-udf-12>>.

[IEEE754] IEEE Computer Society, "IEEE Standard for Floating-Point Arithmetic", IEEE 754-2008, DOI 10.1109/IEEESTD.2008.4610935, 29 August 2008, <<https://doi.org/10.1109/IEEESTD.2008.4610935>>.

[INTEL] Intel Corp., "Unknown".

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<https://www.rfc-editor.org/rfc/rfc3339>>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/rfc/rfc7159>>.

13. Informative References

[draft-hallambaker-mesh-architecture]
Hallam-Baker, P., "Mathematical Mesh 3.0 Part I: Architecture Guide", Work in Progress, Internet-Draft, draft-hallambaker-mesh-architecture-19, 25 October 2021, <<https://datatracker.ietf.org/doc/html/draft-hallambaker-mesh-architecture-19>>.

Mboned
Internet-Draft
Intended status: Experimental
Expires: December 31, 2018

J. Holland
K. Rose
Akamai Technologies, Inc.
June 29, 2018

Asymmetric Manifest Based Integrity
draft-jholland-mboned-ambi-00

Abstract

This document introduces Asymmetric Manifest-Based Integrity (AMBI). AMBI allows each receiver of a stream of multicast packets to check the integrity of the contents of each packet in the data stream. AMBI operates by passing cryptographically verifiable manifests for the data packets, over out-of-band communication channels.

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 December 31, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Comparison with TESLA	4
1.2. Terminology	4
2. Protocol Specification	4
2.1. Packet Identifiers	4
2.1.1. Overview	4
2.1.2. RTP Sequence Number	5
2.1.3. SRTP Sequence Number	5
2.1.4. UDP Option	5
2.2. Anchor Message	6
2.2.1. Overview	6
2.2.2. DNS-based Anchor URI Bootstrap	6
2.2.3. Anchor Message YANG model	6
2.2.4. Example Anchor Message	13
2.3. Manifests	15
2.3.1. Overview	15
2.3.2. Manifest Layout	15
3. IANA Considerations	16
4. Security Considerations	17
4.1. Packet Identifiers	17
5. References	17
5.1. Normative References	17
5.2. Informative References	17
Authors' Addresses	18

1. Introduction

Multicast transport poses security problems that are not easily addressed by the same security mechanisms used for unicast transport.

The "Introduction" sections of the documents describing TESLA [RFC4082], and TESLA in SRTP [RFC4383], and TESLA with ALC and NORM [RFC5776] present excellent overviews of the challenges unique to multicast authentication, briefly summarized here:

- o A MAC based on a symmetric shared secret cannot be used because each packet has multiple receivers that do not trust each other.
- o Asymmetric per-packet signatures can handle only very low bit-rates because of the computational overhead.
- o An asymmetric signature of a larger message comprising multiple packets requires reliable receipt of all such packets, something that cannot be guaranteed in a timely manner even for protocols that do provide reliable delivery, and the retransmission of which

may anyway exceed the useful lifetime for data formats that can otherwise tolerate some degree of loss.

Asymmetric Manifest-Based Integrity (AMBI) specifies a method for receivers or middle boxes to cryptographically authenticate and verify the integrity of a stream of packets, by communicating packet "manifests" (described in Section 2.3) via an out-of-band communication channel that provides authentication and verifiable integrity.

Each manifest contains cryptographic hashes of packet payloads corresponding to specific packets in the authenticated data stream.

Three ways to authenticate a manifest are defined:

- o Asymmetric signature of a message containing the manifest.
- o Authenticated unicast stream providing a sequence of manifests.
- o Using one of the prior two constructions to bootstrap a root manifest containing authentication information for further manifests. This we term "recursive authentication".

When using asymmetric signatures, recursive authentication allows the sender to amortize the computational overhead for a single asymmetric signature across enough data packets to sustain high data rates. When using a secure unicast stream, the recursive verification allows for scaling the authenticated data stream to more receivers than can otherwise be sustained with a limited set of trusted servers.

Upon successful verification of the contents of a manifest and receipt of any subset of the corresponding data packets, the receiver has proof of the integrity of the contents of the data packets listed in the manifest.

An "anchor message", described in Section 2.2, provides the link between an authenticated data stream and the out-of-band channel of manifests that authenticates it.

Authenticating the integrity of the data packets depends on:

- o authentication of the anchor message that provides the linkage between the manifest channel and the data stream; and
- o the secrecy and cryptographic strength of private keys used for signing manifests, or the authentication of the secure unicast streams used for transmitting manifests; and

- o the difficulty of generating a collision for the packet hashes in the manifest.

1.1. Comparison with TESLA

AMBI and TESLA [RFC4082] and [RFC5776] attempt to achieve a similar goal of authenticating the integrity of streams of multicast packets. AMBI imposes a higher overhead, as measured in the amount of extra data required, than TESLA imposes. In exchange, AMBI provides non-repudiation (which TESLA does not), and relaxes the requirement for establishing an upper bound on clock synchronization between sender and receiver.

This tradeoff enables new capabilities for AMBI, relative to TESLA. In particular, when receiving multicast traffic from an untrusted transit network, AMBI can be used by a middle box to authenticate packets from a trusted source before forwarding traffic through the network, and the receiver also can separately authenticate the packets. (This use case is not possible with TESLA because the data packets can't be authenticated until a key is disclosed, so either the middlebox has to forward data packets without first authenticating so that the receiver has them prior to key disclosure, or the middlebox has to hold packets until the key is disclosed, at which point the receiver can no longer establish their authenticity.)

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Protocol Specification

2.1. Packet Identifiers

2.1.1. Overview

Packet identifiers are a sequence number contained within the authenticated payload of the packet. This sequence number is used inside a manifest to associate each packet hash with a specific packet. Each authenticated packet MUST contain a packet identifier. See Section 4.1 for a discussion of the security implications.

This document defines a new UDP option in Section 2.1.4 for use as a packet identifier.

Some multicast-capable transport protocols have a sequence number embedded in data packets in the protocol. The sequence numbers in

these protocols MAY be used instead of the new UDP option, to avoid introducing extra overhead in the authenticated data packets.

In Section 2.1.2, Section 2.1.3, and Section 2.1.4, this document defines some sample ways to specify packet identifiers based on such sequence numbers embedded in existing protocols.

Other appropriate sequence number systems may exist, such as the anti-replay Sequence Number field in Section 3.1 of [RFC6584], when NORM or FLUTE operates with an authentication profile that uses it (however, since that example already provides authentication, it is not added as an option in this document). The AMBI anchor message format can be extended in future documents to support those or other suitable schemes by adding values to the registry defined in Section 3.

In some deployments, in contrast to using the new UDP option, the approach of using an existing sequence number may carry a benefit because it requires no change to the stream of packets being authenticated, possibly enabling interoperability with legacy receivers.

2.1.2. RTP Sequence Number

Sequence number from Section 5.1 of [RFC3550].

TBD: discussion of security consequences of using 16 bits- recommend a bigger hash in manifests for this case?

2.1.3. SRTP Sequence Number

Packet Index from Section 3.3.1 of [RFC3711].

2.1.4. UDP Option

Define a new UDP option [I-D.ietf-tsvwg-udp-options] (TBD2).

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
| UDP Kind=TBD2 | Length=6 | 32-bit sequence number |
+-----+-----+-----+-----+-----+-----+-----+-----+
|                                     |
+-----+-----+-----+-----+-----+-----+-----+-----+

```


2.2. Anchor Message

2.2.1. Overview

An anchor message provides the information that makes it possible to associate the manifests with the data packets they authenticate. ID values that appear as text integers in the anchor message also appear in the manifest binary data, with the anchor message providing context on how to interpret the values.

An anchor message MAY be discovered and transmitted by any means which provides adequate source authentication and data integrity to meet the security needs of the receiver.

In order to support middle-box authentication, it is RECOMMENDED that senders arrange to distribute anchor messages according to the method outlined in Section 2.2.2.

2.2.2. DNS-based Anchor URI Bootstrap

When a middle box tries to process a join for a specific source, if it is configured to perform authentication on SSM multicast channels it can forward, it SHOULD make a DNS request for `ambi.(reverse-source-ip).ip6.arpa` or `ambi.(reverse-source-ip).in-addr.arpa`, for IPv6 or IPv4 source addresses.

When AMBI is provided to authenticate traffic from this source IP, this domain name SHOULD be configured with a TXT field which contains a URI that can be used to securely fetch an anchor message that describes all the AMBI- authenticatable traffic from this source IP.

Other methods MAY be used to discover and transfer an anchor message. The description of alternate methods is out of scope for this document.

TBD: consider breaking up anchor message to avoid large, frequently changing anchors for sources with many groups.

TBD: consider graceful rollover for anchors, instead of synchronized update of anchor hash.

2.2.3. Anchor Message YANG model

```
<CODE BEGINS> file "ietf-ambi-anchor.yang"
module ietf-ambi-anchor {
  yang-version 1.1;

  namespace "urn:ietf:params:xml:ns:yang:ietf-ambi-anchor";
```

```
prefix "ambi";

import ietf-yang-types {
    prefix "yang";
    reference "RFC6991 Section 3";
}

import ietf-inet-types {
    prefix "inet";
    reference "RFC6991 Section 4";
}

import ietf-routing-types {
    prefix "rt-types";
    reference "RFC8294";
}

organization "IETF";

contact
    "Author:    Jake Holland
               <mailto:jholland@akamai.com>
    Author:    Kyle Rose
               <mailto:krose@akamai.com>
    ";

description
    "This module contains the definition for the AMBI anchor
    message data type.

    Copyright (c) 2018 IETF Trust and the persons identified as
    authors of the code. All rights reserved.

    Redistribution and use in source and binary forms, with or
    without modification, is permitted pursuant to, and subject
    to the license terms contained in, the Simplified BSD License
    set forth in Section 4.c of the IETF Trust's Legal Provisions
    Relating to IETF Documents
    (http://trustee.ietf.org/license-info).

    This version of this YANG module is part of
    draft-jholland-mboned-ambi,
    see the internet draft itself for full legal notices.";

revision 2018-06-27 {
    description "Initial revision.";
    reference
        "";
```

```
}

/* TBD: copied some from https://tools.ietf.org/html/rfc8177,
   but the model doesn't seem to match what I want. is there another
   I can import instead of making these here? Or a registry
   to reference? */
identity crypto-hash {
    description
        "Base identity of cryptographic hash options. ";
}

identity sha-256 {
    base crypto-hash;
    description
        "The SHA-256 algorithm.";
}

identity blake2b {
    base crypto-hash;
    description
        "The BLAKE2b algorithm.";
}

identity crypto-signature {
    description
        "Base identity of cryptographic asymmetric signature
        options.";
}

identity ed25519 {
    base crypto-signature;
    description
        "The Ed25519 algorithm.";
}

identity rsa {
    base crypto-signature;
    description
        "The RSA algorithm.";
}

identity sequence-type {
    description
        "Base identity for sequence number type options.";
}

identity rtp {
    base sequence-type;
```

```
        description
            "The sequence number from RTP.";
    }

    identity srtp {
        base sequence-type;
        description
            "The sequence number from SRTP.";
    }

    identity udp {
        base sequence-type;
        description
            "The sequence number from UDP.";
    }

    typedef key-identifier {
        type uint16 {
            range 1..65535;
        }
        description "Key identifier within a manifest";
    }

    typedef bitrate {
        type string {
            pattern '[1-9][0-9]*[GMK]?bps';
        }
        description "Bit-rate of a data stream";
    }

    typedef packetrate {
        type string {
            pattern '[1-9][0-9]*[GMK]?pps';
        }
        description "Packet rate of a data stream";
    }

    typedef manifest-transport {
        type union {
            type leafref {
                path "/anchor/data_stream/id";
            }
            type inet:uri;
        }
        description "Transport method for a manifest stream";
    }

    container anchor {
        container self {
```

```
presence "An anchor message exists";
description
  "Self-referential properties about the anchor message";
leaf uri {
  type inet:uri;
  mandatory true;
  description
    "The canonical URI for this anchor message.";
}
leaf version {
  type uint16;
  mandatory true;
  description
    "The version number for this anchor message.";
}
leaf hash_algorithm {
  type identityref {
    base crypto-hash;
  }
  mandatory true;
  description
    "The algorithm for the anchor message hash provided
    in a manifest.";
}
leaf hash_bits {
  type uint16;
  mandatory true;
  description
    "The number of bits for the anchor's hash provided
    in a manifest.";
}
leaf expires {
  type yang:date-and-time;
  mandatory true;
  description
    "The expiration time for this anchor message.";
}
}
description "Anchor message for AMBI";

list public_key {
  key id;
  description "Public key for non-recursive manifest";
  leaf id {
    type key-identifier;
    mandatory true;
    description
      "The key identifier referenced in a manifest.";
  }
}
```

```
    }
    leaf algorithm {
      type identityref {
        base crypto-signature;
      }
      mandatory true;
      description
        "The signature algorithm for use with this key.";
    }
    leaf signature_bits {
      type uint16;
      mandatory true;
      description
        "The length of the signature provided in manifests
        signed with this key.";
    }
    leaf value {
      type string;
      mandatory true;
      description
        "The base64-encoded value of the public key.";
    }
  }
}

list data_stream {
  key id;
  unique "source destination port";
  description "Stream of data packets to be authenticated";
  leaf id {
    type uint16;
    mandatory true;
    description
      "The datastream_id referenced by a
      manifest_stream.";
  }
  leaf source {
    type inet:ip-address;
    mandatory true;
    description
      "The source IP address of the authenticated data
      stream.";
  }
  leaf destination {
    type rt-types:ip-multicast-group-address;
    mandatory true;
    description
      "The destination group IP address of the
      authenticated data stream.";
  }
}
```

```
    }
    leaf port {
        type uint16;
        mandatory true;
        description
            "The destination UDP port of the authenticated data
            stream.";
    }
    leaf max_bitrate {
        type bitrate;
        mandatory true;
        description
            "The maximum bitrate expected for this data
            stream.";
    }
    leaf max_packetrate {
        type packetrate;
        mandatory true;
        description
            "The maximum packetrate expected for this data
            stream.";
    }
    list authenticator {
        key manifest_id;
        description
            "A manifest stream that authenticates this data";
        leaf manifest_id {
            type leafref {
                path "/anchor/manifest_stream/id";
            }
            mandatory true;
            description
                "The ID of a manifest stream that provides
                authentication for this data stream.";
        }
    }
}

list manifest_stream {
    key id;
    description "Stream of manifests";
    leaf id {
        type uint16;
        mandatory true;
        description "The Manifest ID referenced in a manifest.";
    }
    leaf transport {
        type manifest-transport;
    }
}
```

```

        mandatory true;
        description
            "The ID of the data stream that carries this
             manifest stream or a uri that provides a websocket
             with the stream of manifests.";
    }
    leaf hash_algorithm {
        type identityref {
            base crypto-hash;
        }
        mandatory true;
        description
            "The hash algorithm for the packet hashes within
             manifests in this stream.";
    }
    leaf hash_bits {
        type uint16;
        mandatory true;
        description
            "The number of bits of hash provided for packet
             hashes.";
    }
    leaf sequence_type {
        type identityref {
            base sequence-type;
        }
        mandatory true;
        description
            "The linkage to the data packet sequence numbers in
             the manifest.";
    }
}

}

}
<CODE ENDS>
```

Figure 1: Anchor Message YANG model

2.2.4. Example Anchor Message

```
{
  "ietf-ambi-anchor:anchor": {
    "self": {
      "uri": "https://example.com/ambi/anchor/example_1.json",
      "version": 1,
      "hash_algorithm": "blake2b",
      "hash_bits": 256,
      "expires": "2018-03-05T23:59:59Z"
    }
  }
}
```



```
    },
    "public_key": [
      {
        "id": 1,
        "algorithm": "ed25519",
        "signature_bits": 256,
        "value": "VGhpcyBpcyBub3QgYSBnb29kIGtleSB0byB1c2UuLi4NCg=="
      }
    ],
    "data_stream": [
      {
        "id": 10,
        "source": "192.0.2.10",
        "destination": "232.10.10.1",
        "port": 18001,
        "max_bitrate": "10Mbps",
        "max_packetrate": "1Kpps",
        "authenticator": [
          {
            "manifest_id": 1
          }
        ]
      }
    ],
    {
      "id": 20,
      "source": "192.0.2.10",
      "destination": "232.10.10.1",
      "port": 18002,
      "max_bitrate": "400Kbps",
      "max_packetrate": "40pps",
      "authenticator": [
        {
          "manifest_id": 2
        }
      ]
    }
  ],
  "manifest_stream": [
    {
      "id": 1,
      "transport": 20,
      "hash_algorithm": "blake2b",
      "hash_bits": 80,
      "sequence_type": "udp"
    },
    {
      "id": 2,
      "transport": "https://example.com/ambi/manifest_stream/3",
```

```
        "hash_algorithm": "blake2b",  
        "hash_bits": 80,  
        "sequence_type": "udp"  
      }  
    ]  
  }  
}
```

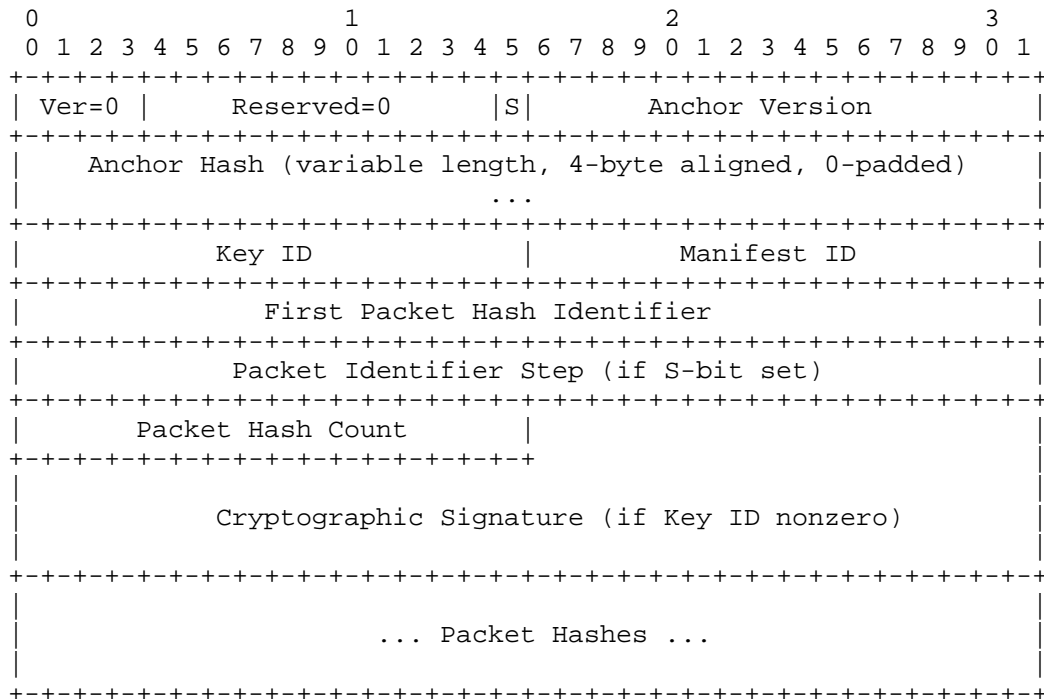
Figure 2: Example Anchor Message

2.3. Manifests

2.3.1. Overview

A manifest cannot be interpreted except in context of a known anchor message. In order for a manifest to be considered as potentially authenticating a set of packets, the anchor version **MUST** match the value in a known unexpired anchor message, and the hash **MUST** match the hash of the contents of that anchor message, according to the `/anchor/self/hash_algorithm` and `/anchor/self/hash_bits` fields, in order for a manifest to be accepted for use as evidence of authenticity and integrity.

2.3.2. Manifest Layout



3. IANA Considerations

TBD1: Request a "Specification Required" registry for Packet identifier methods, from <https://tools.ietf.org/html/rfc5226#section-4.1> . Reserve anything beginning with "experimental:"?

TBD2: Add a new entry to the "UDP Option Kind" numbers registry: <https://tools.ietf.org/html/draft-ietf-tsvwg-udp-options-02#section-14>

TBD: check guidelines in <https://tools.ietf.org/html/rfc5226> and remove this paragraph

Example from: <https://tools.ietf.org/html/rfc5226#section-5.1>

[TO BE REMOVED: Please add the yang model in Section 2.2.3 to: <https://www.iana.org/assignments/yang-parameters/yang-parameters.xhtml>

Name:ietf-ambi-anchor Maintained by IANA: N Namespace:
urn:ietf:params:xml:ns:yang:ietf-ambi-anchor Prefix: anchor
Reference: I-D.draft-jholland-mboned-ambi Notes:]

4. Security Considerations

4.1. Packet Identifiers

TBD: explain attack from generating malicious packets and then looking for collisions, as opposed to having to generate a collision including a sequence number and then hitting a match

TBD: DNSSEC vis-a-vis anchor url discovery. (we need a diagram about for middle-box handling of a revers-path propagated join?) Explain why malicious DNS could deny service, but cannot cause accepting attack packets.

TBD: follow the rest of the guidelines: <https://tools.ietf.org/html/rfc3552>

5. References

5.1. Normative References

- [I-D.ietf-tsvwg-udp-options] Touch, J., "Transport Options for UDP", draft-ietf-tsvwg-udp-options-02 (work in progress), January 2018.
- [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>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<https://www.rfc-editor.org/info/rfc3711>>.

5.2. Informative References

- [DIGSIGN] Rohatgi, P., "How to Sign Digital Streams", 1997, <<https://link.springer.com/content/pdf/10.1007/BFb0052235.pdf>>.
- Published in CRYPTO '97 "Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology", Pages 180-197

- [RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, DOI 10.17487/RFC4082, June 2005, <<https://www.rfc-editor.org/info/rfc4082>>.
- [RFC4383] Baugher, M. and E. Carrara, "The Use of Timed Efficient Stream Loss-Tolerant Authentication (TESLA) in the Secure Real-time Transport Protocol (SRTP)", RFC 4383, DOI 10.17487/RFC4383, February 2006, <<https://www.rfc-editor.org/info/rfc4383>>.
- [RFC5776] Roca, V., Francillon, A., and S. Faurite, "Use of Timed Efficient Stream Loss-Tolerant Authentication (TESLA) in the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", RFC 5776, DOI 10.17487/RFC5776, April 2010, <<https://www.rfc-editor.org/info/rfc5776>>.
- [RFC6584] Roca, V., "Simple Authentication Schemes for the Asynchronous Layered Coding (ALC) and NACK-Oriented Reliable Multicast (NORM) Protocols", RFC 6584, DOI 10.17487/RFC6584, April 2012, <<https://www.rfc-editor.org/info/rfc6584>>.

Authors' Addresses

Jake Holland
Akamai Technologies, Inc.
150 Broadway
Cambridge, MA 02144
United States of America

Email: jakeholland.net@gmail.com

Kyle Rose
Akamai Technologies, Inc.
150 Broadway
Cambridge, MA 02144
United States of America

Email: krose@krose.org

WebAuthn Working Group
Internet-Draft
Intended status: Informational
Expires: December 1, 2018

M. Jones
Microsoft
May 30, 2018

Using secp256k1 with JOSE and COSE
draft-jones-webauthn-secp256k1-00

Abstract

This specification defines algorithm encodings and representations enabling the Standards for Efficient Cryptography Group (SECG) elliptic curve "secp256k1" to be used for JSON Object Signing and Encryption (JOSE) and CBOR Object Signing and Encryption (COSE) messages.

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 December 1, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Notation and Conventions	2
2. JOSE and COSE secp256k1 Curve Key Representations	2
3. ECDSA Signature with secp256k1 Curve	3
4. IANA Considerations	3
4.1. JSON Web Key Elliptic Curve Registration	3
4.2. JOSE Algorithm Registration	3
4.3. COSE Elliptic Curves Registration	4
4.4. COSE Algorithm Registration	4
5. Security Considerations	4
6. References	4
6.1. Normative References	4
6.2. Informative References	5
Acknowledgements	6
Document History	6
Author's Address	6

1. Introduction

This specification defines algorithm encodings and representations enabling the Standards for Efficient Cryptography Group (SECG) elliptic curve "secp256k1" [SEC2] to be used for JSON Object Signing and Encryption (JOSE) [RFC7515] and CBOR Object Signing and Encryption (COSE) [RFC8152] messages. The elliptic curve and associated algorithm are registered in appropriate IANA JOSE and COSE registries.

1.1. Requirements Notation and Conventions

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.

2. JOSE and COSE secp256k1 Curve Key Representations

The Standards for Efficient Cryptography Group (SECG) elliptic curve "secp256k1" [SEC2] is represented in a JSON Web Key (JWK) [RFC7517] using these values:

- o "kty": "EC"
- o "crv": "P-256K"

plus "x" and "y" values to represent the curve point for the key. Other optional values such as "alg" MAY also be present.

It is represented in a COSE_Key [RFC8152] using these values:

- o "kty" (1): "EC2" (2)
- o "crv" (-1): "P-256K" (TBD - requested assignment 8)

plus "x" (-2) and "y" (-3) values to represent the curve point for the key. Other optional values such as "alg" (3) MAY also be present.

3. ECDSA Signature with secp256k1 Curve

The ECDSA signature algorithm is defined in [DSS]. Implementations need to check that the key type is "EC" for JOSE or "EC2" (2) for COSE when creating or verifying a signature.

The ECDSA algorithm specified in this document is:

JOSE Alg Name	COSE Alg Value	Description
ES256K	TBD (requested assignment -43)	ECDSA w/ secp256k1 Curve

Table 1: ECDSA Algorithm Values

4. IANA Considerations

4.1. JSON Web Key Elliptic Curve Registration

This section registers the following value in the IANA "JSON Web Key Elliptic Curve" registry [IANA.JOSE.Curves].

- o Curve Name: P-256K
- o Curve Description: SECG secp256k1 Curve
- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Specification Document(s): Section 2 of [[this specification]]

4.2. JOSE Algorithm Registration

This section registers the following value in the IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE.Algorithms].

- o Algorithm Name: ES256K
- o Algorithm Description: ECDSA w/ secp256k1 Curve
- o Algorithm Usage Locations: alg

- o JOSE Implementation Requirements: Optional
- o Change Controller: IESG
- o Reference: Section 3 of [[this specification]]
- o Algorithm Analysis Document(s): [SEC2]

4.3. COSE Elliptic Curves Registration

This section registers the following value in the IANA "COSE Elliptic Curves" registry [IANA.COSE.Curves].

- o Name: P-256K
- o Value: TBD (requested assignment 8)
- o Key Type: EC2
- o Description: SECG secp256k1 Curve
- o Change Controller: IESG
- o Reference: Section 2 of [[this specification]]
- o Recommended: Yes

4.4. COSE Algorithm Registration

This section registers the following value in the IANA "COSE Algorithms" registry [IANA.COSE.Algorithms].

- o Name: ES256K
- o Value: TBD (requested assignment -43)
- o Description: ECDSA w/ secp256k1 Curve
- o Reference: Section 3 of this document
- o Recommended: Yes

5. Security Considerations

Care should be taken that a secp256k1 key not be mistaken for a P-256 key, given that their representations are the same except for the "crv" value.

The procedures and security considerations described in the [SEC1], [SEC2], and [DSS] specifications apply to implementations of this specification.

6. References

6.1. Normative References

- [DSS] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", FIPS PUB 186-4, July 2013, <<http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>>.

- [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>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <<https://www.rfc-editor.org/info/rfc7517>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [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>>.
- [SEC1] Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography", Version 2.0, May 2009, <<http://www.secg.org/sec1-v2.pdf>>.
- [SEC2] Standards for Efficient Cryptography Group, "SEC 2: Recommended Elliptic Curve Domain Parameters", Version 2.0, January 2010, <<http://www.secg.org/sec2-v2.pdf>>.

6.2. Informative References

- [IANA.COSE.Algorithms] IANA, "COSE Algorithms", <<https://www.iana.org/assignments/cose/cose.xhtml#algorithms>>.
- [IANA.COSE.Curves] IANA, "COSE Elliptic Curves", <<https://www.iana.org/assignments/cose/cose.xhtml#elliptic-curves>>.
- [IANA.JOSE.Algorithms] IANA, "JSON Web Signature and Encryption Algorithms", <<https://www.iana.org/assignments/jose/jose.xhtml#web-signature-encryption-algorithms>>.

[IANA.JOSE.Curves]

IANA, "JSON Web Key Elliptic Curve",
<[https://www.iana.org/assignments/jose/
jose.xhtml#web-key-elliptic-curve](https://www.iana.org/assignments/jose/jose.xhtml#web-key-elliptic-curve)>.

Acknowledgements

TBD

Document History

[[to be removed by the RFC Editor before publication as an RFC]]

-00

- o Initial version.

Author's Address

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: <http://self-issued.info/>

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 20, 2018

G. Mandyam
L. Lundblade
M. Ballesteros
J. O'Donoghue
Qualcomm Technologies Inc.
May 19, 2018

The Entity Attestation Token (EAT)
draft-mandyam-eat-00

Abstract

An attestation format based on concise binary object representation (CBOR) is proposed that is suitable for inclusion in a CBOR Web Token (CWT), known as the Entity Attestation Token (EAT). The associated data can be used by a relying party to assess the security state of a remote device or module.

Contributing

TBD

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 November 20, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Entity Overview	4
1.2. Use of CBOR and COSE	4
1.3. EAT Operating Models	5
1.4. What is Not Standardized	6
1.4.1. Transmission Protocol	6
1.4.2. Signing Scheme	6
2. Terminology	7
3. The Claims	8
3.1. Universal Entity ID (UEID) Claim	8
3.2. Origination (origination) Claims	10
3.3. OEM identification by IEEE OUI	10
3.4. Security Level (seclevel) Claim	11
3.5. Nonce (nonce) Claim	12
3.6. Secure Boot and Debug Enable State Claims	12
3.6.1. Secure Boot Enabled (secbootenabled) Claim	12
3.6.2. Debug Disabled (debugdisabled) Claim	12
3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot) Claim	12
3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim	12
3.6.5. Debug Full Permanent Disable (debugfullpermanentdisable) Claim	13
3.7. Location (loc) Claim	13
3.7.1. lat (latitude) claim	13
3.7.2. long (longitude) claim	13
3.7.3. alt (altitude) claim	13
3.7.4. acc (accuracy) claim	13
3.7.5. altacc (altitude accuracy) claim	14
3.7.6. heading claim	14
3.7.7. speed claim	14
3.8. ts (timestamp) claim	14
3.9. age claim	14
3.10. uptime claim	14
3.11. The submods Claim	15
3.11.1. The submod_name Claim	15
3.11.2. Nested EATs, the eat Claim	15
4. IANA Considerations	15
4.1. Reuse of CBOR Web Token (CWT) Claims Registry	15
4.1.1. Claims Registered by This Document	16

4.2. EAT CBOR Tag Registration	16
4.2.1. Tag Registered by This Document	16
5. Privacy Considerations	17
5.1. UEID Privacy Considerations	17
6. Security Considerations	18
7. References	18
7.1. Normative References	18
7.2. Informative References	19
Appendix A. Examples	20
A.1. Very Simple EAT	20
A.2. Example with Submodules, Nesting and Security Levels . .	20
Authors' Addresses	21

1. Introduction

Remote device attestation is fundamental service that allows a remote device such as a mobile phone, an Internet-of-Things (IoT) device, or other endpoint to prove itself to a relying party, a server or a service. This allows the relying party to know some characteristics about the device and decide whether it trusts the device.

Remote attestation is a fundamental service that can underly other protocols and services that need to know about the trustworthiness of the device before proceeding. One good example is biometric authentication where the biometric matching is done on the device. The relying party needs to know that the device is one that is known to do biometric matching correctly. Another example is content protection where the relying party wants to know the device will protect the data. This generalizes on to corporate enterprises that might want to know that a device is trustworthy before allowing corporate data to be accessed by it.

The notion of attestation here is large and may include, but is not limited to the following:

- o Proof of the make and model of the device hardware (HW)
- o Proof of the make and model of the device processor, particularly for security oriented chips
- o Measurement of the software (SW) running on the device
- o Configuration and state of the device
- o Environmental characteristics of the device such as its GPS location

The required data format should be general purpose and extensible so that it can work across many use cases. This is why CBOR (see [RFC7049]) is chosen as the format -- it already supports a rich set of data types, and is both expressive and extensible. It translates well to JSON for good interoperability with web technology. It is compact and can work on very small IoT device. The format proposed here is small enough that a limited version can be implemented in pure hardware gates with no software at all. Moreover, the attestation data is defined in the form of claims that is the same as CBOR Web Token (CWT, see [RFC8392]). This is the motivation for defining the Entity Attestation Token, i.e. EAT.

1.1. Entity Overview

An "entity" can be any device or device subassembly ("submodule") that can generate its own attestation in the form of an EAT. The attestation should be cryptographically verifiable by the EAT consumer. An EAT at the device-level can be composed of several submodule EAT's. It is assumed that any entity that can create an EAT does so by means of a dedicated root-of-trust (RoT).

Modern devices such as a mobile phone have many different execution environments operating with different security levels. For example it is common for a mobile phone to have an "apps" environment that runs an operating system (OS) that hosts a plethora of downloadable apps. It may also have a TEE (Trusted Execution Environment) that is distinct, isolated, and hosts security-oriented functionality like biometric authentication. Additionally it may have an eSE (embedded Secure Element) - a high security chip with defenses against HW attacks that can serve as a RoT. This device attestation format allows the attested data to be tagged at a security level from which it originates. In general, any discrete execution environment that has an identifiable security level can be considered an entity.

1.2. Use of CBOR and COSE

Fundamentally this attestation format is a verifiable data format. It is a collection of data items that can be signed by an attestation key, hashed, and/or encrypted. As per Section 7 of [RFC8392], the verification method is in the CWT using the CBOR Object Signing and Encryption (COSE) methodology (see [RFC8152]).

In addition, the reported attestation data could be determined within the secure operating environment or written to it from an external and presumably less trusted entity on the device. In either case, the source of the reported data must be identifiable by the relying party.

This attestation format is a single relatively simple signed message. It is designed to be incorporated into many other protocols and many other transports. It is also designed such that other SW and apps can add their own data to the message such that it is also attested.

1.3. EAT Operating Models

At least the following three participants exist in all EAT operating models. Some operating models have additional participants.

The Entity. This is the phone, the IoT device, the sensor, the sub-assembly or such that the attestation provides information about.

The Manufacturer. The company that made the entity. This may be a chip vendor, a circuit board module vendor or a vendor of finished consumer products.

The Relying Party. The server, service or company that makes use of the information in the EAT about the entity.

In all operating models, the manufacturer provisions some secret attestation key material (AKM) into the entity during manufacturing. This might be during the manufacturer of a chip at a fabrication facility (fab) or during final assembly of a consumer product or any time in between. This attestation key material is used for signing EATs.

In all operating models, hardware and/or software on the entity create an EAT of the format described in this document. The EAT is always signed by the attestation key material provisioned by the manufacturer.

In all operating models, the relying party must end up knowing that the signature on the EAT is valid and consistent with data from claims in the EAT. This can happen in many different ways. Here are some examples.

- o The EAT is transmitted to the relying party. The relying party gets corresponding key material (e.g. a root certificate) from the manufacturer. The relying party performs the verification.
- o The EAT is transmitted to the relying party. The relying party transmits the EAT to a verification service offered by the manufacturer. The server returns the validated claims.
- o The EAT is transmitted directly to a verification service, perhaps operated by the manufacturer or perhaps by another party. It verifies the EAT and makes the validated claims available to the

relying party. It may even modify the claims in some way and re-sign the EAT (with a different signing key).

This standard supports all these operating models and does not prefer one over the other. It is important to support this variety of operating models to generally facilitate deployment and to allow for some special scenarios. One special scenario has a validation service that is monetized, most likely by the manufacturer. In another, a privacy proxy service processes the EAT before it is transmitted to the relying party. In yet another, symmetric key material is used for signing. In this case the manufacturer should perform the verification, because any release of the key material would enable a participant other than the entity to create valid signed EATs.

1.4. What is Not Standardized

1.4.1. Transmission Protocol

EATs may be transmitted by any protocol. For example, they might be added in extension fields of other protocols, bundled into an HTTP header, or just transmitted as files. This flexibility is intentional to allow broader adoption. This flexibility is possible because EAT's are self-secured with signing (and possibly additionally with encryption and anti-replay). The transmission protocol is not required to fulfill any additional security requirements.

For certain devices, a direct connection may not exist between the EAT-producing device and the Relying Party. In such cases, the EAT should be protected against malicious access. The use of COSE allows for signing and encryption of the EAT. Therefore even if the EAT is conveyed through intermediaries between the device and Relying Party, such intermediaries cannot easily modify the EAT payload or alter the signature.

1.4.2. Signing Scheme

The term "signing scheme" is used to refer to the system that includes end-end process of establishing signing attestation key material in the entity, signing the EAT, and verifying it. This might involve key IDs and X.509 certificate chains or something similar but different. The term "signing algorithm" refers just to the algorithm ID in the COSE signing structure. No particular signing algorithm or signing scheme is required by this standard.

There are three main implementation issues driving this. First, secure non-volatile storage space in the entity for the attestation

key material may be highly limited, perhaps to only a few hundred bits, on some small IoT chips. Second, the factory cost of provisioning key material in each chip or device may be high, with even millisecond delays adding to the cost of a chip. Third, privacy-preserving signing schemes like ECDA (Elliptic Curve Direct Anonymous Attestation) are complex and not suitable for all use cases.

Eventually some form of standardization of the signing scheme may be required. This might come in the form of another standard that adds to this document, or when there is clear convergence on a small number of signing schemes this standard can be updated.

2. Terminology

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.

This document reuses terminology from JWT [RFC7519], COSE [RFC8152], and CWT [RFC8392].

StringOrURI. The "StringOrURI" term in this specification has the same meaning and processing rules as the JWT "StringOrURI" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR text string instead of a JSON text string.

NumericDate. The "NumericDate" term in this specification has the same meaning and processing rules as the JWT "NumericDate" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR numeric date (from Section 2.4.1 of [RFC7049]) instead of a JSON number. The encoding is modified so that the leading tag 1 (epoch-based date/time) MUST be omitted.

Claim Name. The human-readable name used to identify a claim.

Claim Key. The CBOR map key used to identify a claim.

Claim Value. The CBOR map value representing the value of the claim.

CWT Claims Set. The CBOR map that contains the claims conveyed by the CWT.

FloatOrNumber. The "FloatOrNumber" term in this specification is the type of a claim that is either a CBOR positive integer, negative integer or floating point number.

Attestation Key Material (AKM). The key material used to sign the EAT token. If it is done symmetrically with HMAC, then this is a simple symmetric key. If it is done with ECC, such as an IEEE DevID [IDevID], then this is the private part of the EC key pair. If ECDAAs are used, (e.g., as used by Enhanced Privacy ID, i.e. EPID) then it is the key material needed for ECDAAs.

3. The Claims

3.1. Universal Entity ID (UEID) Claim

UEIDs identify individual manufactured entities / devices such as a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. It may identify the entire device or a submodule or subsystem. It does not identify types, models or classes of devices. It is akin to a serial number, though it does not have to be sequential.

It is identified by Claim Key X (X is TBD).

UEIDs must be universally and globally unique across manufacturers and countries. UEIDs must also be unique across protocols and systems, as tokens are intended to be embedded in many different protocols and systems. No two products anywhere, even in completely different industries made by two different manufacturers in two different countries, should have the same UEID (if they are not global and universal in this way then relying parties receiving them will have to track other characteristics of the device to keep devices distinct between manufacturers).

The UEID should be permanent. It should never change for a given device / entity. In addition, it should not be reprogrammable.

UEIDs are binary byte-strings (resulting in a smaller size than text strings). When handled in text-based protocols, they should be base-64 encoded.

UEIDs are variable length with a maximum size of 33 bytes (1 type byte and 256 bits). A receiver of a token with UEIDs may reject the token if a UEID is larger than 33 bytes.

UEIDs are not designed for direct use by humans (e.g., printing on the case of a device), so no textual representation is defined.

A UEID is a byte string. From the consumer's view (the relying party) it is opaque with no bytes having any special meaning.

When the entity constructs the UEID, the first byte is a type and the following bytes the ID for that type. Several types are allowed to accommodate different industries and different manufacturing processes and to give options to avoid paying fees for certain types of manufacturer registrations.

Type Byte	Type Name	Specification
0x01	GUID	This is a 128 to 256 bit random number generated once and stored in the device. The GUID may be constructed from various identifiers on the device using a hash function or it may be just the raw random number. In any case, the random number must have entropy of at least 128 bits as this is what gives the global
0x02	IEEE EUI	This makes use of the IEEE company identification registry. An EUI is made up of an OUI and OUI-36 or a CID, different registered company identifiers, and some unique per-device identifier. EUIs are often the same as or similar to MAC addresses. (Note that while devices with multiple network interfaces may have multiple MAC addresses, there is only one UEID for a device) TODO: normative references to IEEE.
0x03	IMEI	TODO: figure how to specify IMEIs

Table 1: UEID Composition Types

The consumer (the Relying Party) of a UEID should treat a UEID as a completely opaque string of bytes and not make any use of its internal structure. For example they should not use the OUI part of a type 0x02 UEID to identify the manufacturer of the device. Instead they should use the OUI claim that is defined elsewhere. The reasons for this are:

- o UEIDs types may vary freely from one manufacturer to the next.
- o New types of UEIDs may be created. For example a type 0x04 UEID may be created based on some other manufacturer registration scheme.
- o Device manufacturers are allowed to change from one type of UEID to another anytime they want. For example they may find they can optimize their manufacturing by switching from type 0x01 to type

0x02 or vice versa. The main requirement on the manufacturer is that UEIDs be universally unique.

3.2. Origination (origination) Claims

This claim describes the parts of the device or entity that are creating the EAT. Often it will be tied back to the device or chip manufacturer. The following table gives some examples:

Name	Description
Acme-TEE	The EATs are generated in the TEE authored and configured by "Acme"
Acme-TPM	The EATs are generated in a TPM manufactured by "Acme"
Acme-Linux-Kernel	The EATs are generated in a Linux kernel configured and shipped by "Acme"
Acme-TA	The EATs are generated in a Trusted Application (TA) authored by "Acme"

The claim is represented by Claim Key X+15. It is type StringOrURI.

TODO: consider a more structure approach where the name and the URI and other are in separate fields.

TODO: This needs refinement. It is somewhat parallel to issuer claim in CWT in that it describes the authority that created the token.

3.3. OEM identification by IEEE OUI

This claim identifies a device OEM by the IEEE OUI. Reference TBD. It is a byte string representing the OUI in binary form in network byte order (TODO: confirm details).

Companies that have more than one IEEE OUI registered with IEEE should pick one and prefer that for all their devices.

Note that the OUI is in common use as a part of MAC Address. This claim is only the first bits of the MAC address that identify the manufacturer. The IEEE maintains a registry for these in which many companies participate. This claim is represented by Claim Key TBD.

3.4. Security Level (seclevel) Claim

EATs have a claim that roughly characterizes the device / entities ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is done by roughly defining four security levels as described below. This is similar to the security levels defined in the Metadata Service defined by the Fast Identity Online (FIDO) Alliance (TODO: reference).

These claims describe security environment and countermeasures available on the end-entity / client device where the attestation key reside and the claims originate.

This claim is identified by Claim Key X+2. The value is an integer between 1 and 4 as defined below.

- 1 - Unrestricted There is some expectation that implementor will protect the attestation signing keys at this level. Otherwise the EAT provides no meaningful security assurances.
- 2- Restricted Entities at this level should not be general-purpose operating environments that host features such as app download systems, web browsers and complex productivity applications. It is akin to the Secure Restricted level (see below) without the security orientation. Examples include a WiFi subsystem, an IoT camera, or sensor device.
- 3 - Secure Restricted Entities at this level must meet the criteria defined by FIDO Allowed Restricted Operating Environments (TODO: reference). Examples include TEE's and schemes using virtualization-based security. Like the FIDO security goal, security at this level is aimed at defending well against large-scale network / remote attacks against the device.
- 4 - Hardware Entities at this level must include substantial defense against physical or electrical attacks against the device itself. It is assumed any potential attacker has captured the device and can disassemble it. Example include TPMs and Secure Elements.

This claim is not intended as a replacement for a proper end-device security certification schemes such as those based on FIPS (TODO: reference) or those based on Common Criterion (TODO: reference). The claim made here is solely a self-claim made by the Entity Originator.

3.5. Nonce (nonce) Claim

The "nonce" (Nonce) claim represents a random value that can be used to avoid replay attacks. This would be ideally generated by the CWT consumer. This value is intended to be a CWT companion claim to the existing JWT claim `**_IANAJWT_` (TODO: fix this reference). The nonce claim is identified by Claim Key X+3.

3.6. Secure Boot and Debug Enable State Claims

3.6.1. Secure Boot Enabled (secbootenabled) Claim

The "secbootenabled" (Secure Boot Enabled) claim represents a boolean value that indicates whether secure boot is enabled either for an entire device or an individual submodule. If it appears at the device level, then this means that secure boot is enabled for all submodules. Secure boot enablement allows a secure boot loader to authenticate software running either in a device or a submodule prior allowing execution. This claim is identified by Claim Key X+4.

3.6.2. Debug Disabled (debugdisabled) Claim

The "debugdisabled" (Debug Disabled) claim represents a boolean value that indicates whether debug capabilities are disabled for an entity (i.e. value of 'true'). Debug disablement is considered a prerequisite before an entity is considered operational. This claim is identified by Claim Key X+5.

3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot) Claim

The "debugdisabledsinceboot" (Debug Disabled Since Boot) claim represents a boolean value that indicates whether debug capabilities for the entity were not disabled in any way since boot (i.e. value of 'true'). This claim is identified by Claim Key X+6.

3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim

The "debugpermanentdisable" (Debug Permanent Disable) claim represents a boolean value that indicates whether debug capabilities for the entity are permanently disabled (i.e. value of 'true'). This value can be set to 'true' also if only the manufacturer is allowed to enabled debug, but the end user is not. This claim is identified by Claim Key X+7.

3.6.5. Debug Full Permanent Disable (debugfullpermanentdisable) Claim

The "debugfullpermanentdisable" (Debug Full Permanent Disable) claim represents a boolean value that indicates whether debug capabilities for the entity are permanently disabled (i.e. value of 'true'). This value can only be set to 'true' if no party can enable debug capabilities for the entity. Often this is implemented by blowing a fuse on a chip as fuses cannot be restored once blown. This claim is identified by Claim Key X+8.

3.7. Location (loc) Claim

The "loc" (location) claim is a CBOR-formatted object that describes the location of the device entity from which the attestation originates. It is identified by Claim Key X+10. It is comprised of an array of additional subclaims that represent the actual location coordinates (latitude, longitude and altitude). The location coordinate claims are consistent with the WGS84 coordinate system [WGS84]. In addition, a subclaim providing the estimated accuracy of the location measurement is defined.

3.7.1. lat (latitude) claim

The "lat" (latitude) claim contains the value of the device location corresponding to its latitude coordinate. It is of data type FloatOrNumber and identified by Claim Key X+11.

3.7.2. long (longitude) claim

The "long" (longitude) claim contains the value of the device location corresponding to its longitude coordinate. It is of data type FloatOrNumber and identified by Claim Key X+12.

3.7.3. alt (altitude) claim

The "alt" (altitude) claim contains the value of the device location corresponding to its altitude coordinate (if available). It is of data type FloatOrNumber and identified by Claim Key X+13.

3.7.4. acc (accuracy) claim

The "acc" (accuracy) claim contains a value that describes the location accuracy. It is non-negative and expressed in meters. It is of data type FloatOrNumber and identified by Claim Key X+14.

3.7.5. altacc (altitude accuracy) claim

The "altacc" (altitude accuracy) claim contains a value that describes the altitude accuracy. It is non-negative and expressed in meters. It is of data type FloatOrNumber and identified by Claim Key X+15.

3.7.6. heading claim

The "heading" claim contains a value that describes direction of motion for the entity. Its value is specified in degrees, between 0 and 360. It is of data type FloatOrNumber and identified by Claim Key X+16.

3.7.7. speed claim

The "speed" claim contains a value that describes the velocity of the entity in the horizontal direction. Its value is specified in meters/second and must be non-negative. It is of data type FloatOrNumber and identified by Claim Key X+17.

3.8. ts (timestamp) claim

The "ts" (timestamp) claim contains a timestamp derived using the same time reference as is used to generate an "iat" claim (see Section 3.1.6 of [RFC8392]). It is of the same type as "iat" (integer or floating-point), and is identified by Claim Key X+18. It is meant to designate the time at which a measurement was taken, when a location was obtained, or when a token was actually transmitted. The timestamp would be included as a subclaim under the "submod" or "loc" claims (in addition to the existing respective subclaims), or at the device level.

3.9. age claim

The "age" claim contains a value that represents the number of seconds that have elapsed since the token was created, measurement was made, or location was obtained. Typical attestable values are sent as soon as they are obtained. However in the case that such a value is buffered and sent at a later time and a sufficiently accurate time reference is unavailable for creation of a timestamp, then the age claim is provided. It is identified by Claim Key X+19.

3.10. uptime claim

The "uptime" claim contains a value that represents the number of seconds that have elapsed since the entity or submod was last booted. It is identified by Claim Key X+20.

3.11. The submods Claim

Some devices are complex, having many subsystems or submodules. A mobile phone is a good example. It may have several connectivity submodules for communications (e.g., WiFi and cellular). It may have sub systems for low-power audio and video playback. It may have one or more security-oriented subsystems like a TEE or a Secure Element.

The claims for each these can be grouped together in a submodule.

Specifically, the "submods" claim is an array. Each item in the array is a CBOR map containing all the claims for a particular submodule. It is identified by Claim Key X+22.

The security level of the submod is assumed to be at the same level as the main entity unless there is a security level claim in that submodule indicating otherwise. The security level of a submodule can never be higher (more secure) than the security level of the EAT it is a part of.

3.11.1. The submod_name Claim

Each submodule should have a submod_name claim that is descriptive name. This name should be the CBOR txt type.

3.11.2. Nested EATs, the eat Claim

It is allowed for one EAT to be embedded in another. This is for complex devices that have more than one subsystem capable of generating an EAT. Typically one will be the device-wide EAT that is low to medium security and another from a Secure Element or similar that is high security.

The contents of the "eat" claim must be a fully signed, optionally encrypted, EAT token. It is identified by Claim Key X+23.

4. IANA Considerations

4.1. Reuse of CBOR Web Token (CWT) Claims Registry

Claims defined for EAT are compatible with those of CWT so the CWT Claims Registry is re used. New new IANA registry is created. All EAT claims should be registered in the CWT Claims Registry.

4.1.1. Claims Registered by This Document

- o Claim Name: UEID
- o Claim Description: The Universal Entity ID
- o JWT Claim Name: N/A
- o Claim Key: X
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): *this document*

TODO: add the rest of the claims in here

4.2. EAT CBOR Tag Registration

How an EAT consumer determines whether received CBOR-formatted data actually represents a valid EAT is application-dependent, much like a CWT. For instance, a specific MIME type associated with the EAT such as "application/eat" could be sufficient for identification of the EAT. Note however that EAT's can include other EAT's (e.g. a device EAT comprised of several submodule EAT's). In this case, a CBOR tag dedicated to the EAT will be required at least for the submodule EAT's and the tag must be a valid CBOR tag. In other words - the EAT CBOR tag can optionally prefix a device-level EAT, but a EAT CBOR tag must always prefix a submodule EAT. The proposed EAT CBOR tag is 71.

4.2.1. Tag Registered by This Document

- o CBOR Tag: 71
- o Data Item: Entity Attestation Token (EAT)
- o Semantics: Entity Attestation Token (CWT), as defined in *this_doc*
- o Reference: *this_doc*
- o Point of Contact: Giridhar Mandyam, mandyam@qti.qualcomm.com

5. Privacy Considerations

Certain EAT claims can be used to track the owner of an entity and therefore implementations should consider providing privacy-preserving options dependent on the intended usage of the EAT. Examples would include suppression of location claims for EAT's provided to unauthenticated consumers.

5.1. UEID Privacy Considerations

A UEID is usually not privacy preserving. Any set of relying parties that receives tokens that happen to be from a single device will be able to know the tokens are all from the same device and be able to track the device. Thus, in many usage situations ueid violates governmental privacy regulation. In other usage situations UEID will not be allowed for certain products like browsers that give privacy for the end user. It will often be the case that tokens will not have a UEID for these reasons.

There are several strategies that can be used to still be able to put UEID's in tokens:

- o The device obtains explicit permission from the user of the device to use the UEID. This may be through a prompt. It may also be through a license agreement. For example, agreements for some online banking and brokerage services might already cover use of a UEID.
- o The UEID is used only in a particular context or particular use case. It is used only by one relying party.
- o The device authenticates the relying party and generates a derivate UEID just for that particular relying party. For example, the relying party could prove their identity cryptographically to the device, then the device generates a UEID just for that relying party by hashing a proofed relying party ID with the main device UEID.

Note that some of these privacy preservation strategies result in multiple UEIDs per device. Each UEID is used in a different context, use case or system on the device. However, from the view of the relying party, there is just one UEID and it is still globally universal across manufacturers.

6. Security Considerations

TODO: Perhaps this can be the same as CWT / COSE, but not sure yet because it involves so much entity / device security that those do not.

7. References

7.1. Normative References

- [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>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [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>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [TIME_T] The Open Group Base Specifications, "Vol. 1: Base Definitions, Issue 7", Section 4.15 'Seconds Since the Epoch', IEEE Std 1003.1, 2013 Edition, 2013, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_15>.
- [WGS84] National Imagery and Mapping Agency, "National Imagery and Mapping Agency Technical Report 8350.2, Third Edition", 2000, <<http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf>>.

7.2. Informative References

- [ASN.1] International Telecommunication Union, "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, 1994.
- [IDevID] "IEEE Standard, "IEEE 802.1AR Secure Device Identifier", December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.
- [Webauthn] Worldwide Web Consortium, "Web Authentication: A Web API for accessing scoped credentials", 2016.

Appendix A. Examples

A.1. Very Simple EAT

This is shown in CBOR diagnostic form. Only the payload signed by COSE is shown.

```
{
  / nonce /                11:h'948f8860d13a463e8e',
  / UEID /                 8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
  / secbootenabled /      13:true,
  / debugpermanentdisable / 15:true,
  / ts /                  21:1526542894,
}
```

A.2. Example with Submodules, Nesting and Security Levels

```
{
  / nonce /                11:h'948f8860d13a463e8e',
  / UEID /                 8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
  / secbootenabled /      13:true,
  / debugpermanentdisable / 15:true,
  / ts /                  21:1526542894,
  / seclevel /            10:3, / secure restricted OS /

  / submods / 30:
  [
    / 1st submod, an Android Application / {
      / submod_name /      30:'Android App "Foo"',
      / seclevel /        10:1, / unrestricted /
      / app data /        -70000:'text string'
    },
    / 2nd submod, A nested EAT from a secure element / {
      / submod_name /      30:'Secure Element EAT',
      / eat /              31:71( 18(
        / an embedded EAT / [ /...COSE_Sign1 bytes with payload.../ ]
      ) )
    },
    / 3rd submod, information about Linux Android / {
      / submod_name /      30:'Linux Android',
      / seclevel /        10:1, / unrestricted /
      / custom - release / -80000:'8.0.0',
      / custom - version / -80001:'4.9.51+'
    }
  ]
}
```

Authors' Addresses

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 7200
EMail: mandyam@qti.qualcomm.com

Laurence Lundblade
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 658 3584
EMail: llundbla@qti.qualcomm.com

Miguel Ballesteros
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 4299
EMail: mballest@qti.qualcomm.com

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough GU14 7LS
United Kingdom

Phone: +44 1252 363189
EMail: jodonogh@qti.qualcomm.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 23, 2019

G. Mandyam
Qualcomm Technologies Inc.
L. Lundblade
Security Theory LLC
M. Ballesteros
J. O'Donoghue
Qualcomm Technologies Inc.
November 19, 2018

The Entity Attestation Token (EAT)
draft-mandyam-eat-01

Abstract

An attestation format based on concise binary object representation (CBOR) is proposed that is suitable for inclusion in a CBOR Web Token (CWT), known as the Entity Attestation Token (EAT). The associated data can be used by a relying party to assess the security state of a remote device or module.

Contributing

TBD

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 May 23, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Entity Overview	4
1.2. Use of CBOR and COSE	5
1.3. EAT Operating Models	5
1.4. What is Not Standardized	6
1.4.1. Transmission Protocol	6
1.4.2. Signing Scheme	7
2. Terminology	7
3. The Claims	8
3.1. Universal Entity ID (UEID) Claim	8
3.2. Origination (origination) Claims	10
3.3. OEM identification by IEEE OUI	10
3.4. Security Level (seclevel) Claim	11
3.5. Nonce (nonce) Claim	12
3.6. Secure Boot and Debug Enable State Claims	12
3.6.1. Secure Boot Enabled (secbootenabled) Claim	12
3.6.2. Debug Disabled (debugdisabled) Claim	12
3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot) Claim	12
3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim	12
3.6.5. Debug Full Permanent Disable (debugfullpermanentdisable) Claim	13
3.7. Location (loc) Claim	13
3.7.1. lat (latitude) claim	13
3.7.2. long (longitude) claim	13
3.7.3. alt (altitude) claim	13
3.7.4. acc (accuracy) claim	13
3.7.5. altacc (altitude accuracy) claim	14
3.7.6. heading claim	14
3.7.7. speed claim	14
3.8. ts (timestamp) claim	14
3.9. age claim	14
3.10. uptime claim	14
3.11. The submods Claim	15
3.11.1. The submod_name Claim	15
3.11.2. Nested EATs, the eat Claim	15

4.	CBOR Interoperability	15
4.1.	Integer Encoding (major type 0 and 1)	16
4.2.	String Encoding (major type 2 and 3)	16
4.3.	Map and Array Encoding (major type 4 and 5)	16
4.4.	Date and Time	16
4.5.	URIs	16
4.6.	Floating Point	16
4.7.	Other types	16
5.	IANA Considerations	17
5.1.	Reuse of CBOR Web Token (CWT) Claims Registry	17
5.1.1.	Claims Registered by This Document	17
5.2.	EAT CBOR Tag Registration	17
5.2.1.	Tag Registered by This Document	17
6.	Privacy Considerations	18
6.1.	UEID Privacy Considerations	18
7.	Security Considerations	19
8.	References	19
8.1.	Normative References	19
8.2.	Informative References	20
Appendix A.	Examples	21
A.1.	Very Simple EAT	21
A.2.	Example with Submodules, Nesting and Security Levels	21
Authors' Addresses	22

1. Introduction

Remote device attestation is fundamental service that allows a remote device such as a mobile phone, an Internet-of-Things (IoT) device, or other endpoint to prove itself to a relying party, a server or a service. This allows the relying party to know some characteristics about the device and decide whether it trusts the device.

Remote attestation is a fundamental service that can underlie other protocols and services that need to know about the trustworthiness of the device before proceeding. One good example is biometric authentication where the biometric matching is done on the device. The relying party needs to know that the device is one that is known to do biometric matching correctly. Another example is content protection where the relying party wants to know the device will protect the data. This generalizes on to corporate enterprises that might want to know that a device is trustworthy before allowing corporate data to be accessed by it.

The notion of attestation here is large and may include, but is not limited to the following:

- o Proof of the make and model of the device hardware (HW)

- o Proof of the make and model of the device processor, particularly for security oriented chips
- o Measurement of the software (SW) running on the device
- o Configuration and state of the device
- o Environmental characteristics of the device such as its GPS location

The required data format should be general purpose and extensible so that it can work across many use cases. This is why CBOR (see [RFC7049]) was chosen as the format -- it already supports a rich set of data types, and is both expressive and extensible. It translates well to JSON for good interoperability with web technology. It is compact and can work on very small IoT device. The format proposed here is small enough that a limited version can be implemented in pure hardware gates with no software at all. Moreover, the attestation data is defined in the form of claims that is the same as CBOR Web Token (CWT, see [RFC8392]). This is the motivation for defining the Entity Attestation Token, i.e. EAT.

1.1. Entity Overview

An "entity" can be any device or device subassembly ("submodule") that can generate its own attestation in the form of an EAT. The attestation should be cryptographically verifiable by the EAT consumer. An EAT at the device-level can be composed of several submodule EAT's. It is assumed that any entity that can create an EAT does so by means of a dedicated root-of-trust (RoT).

Modern devices such as a mobile phone have many different execution environments operating with different security levels. For example it is common for a mobile phone to have an "apps" environment that runs an operating system (OS) that hosts a plethora of downloadable apps. It may also have a TEE (Trusted Execution Environment) that is distinct, isolated, and hosts security-oriented functionality like biometric authentication. Additionally it may have an eSE (embedded Secure Element) - a high security chip with defenses against HW attacks that can serve as a RoT. This device attestation format allows the attested data to be tagged at a security level from which it originates. In general, any discrete execution environment that has an identifiable security level can be considered an entity.

1.2. Use of CBOR and COSE

Fundamentally this attestation format is a verifiable data format. It is a collection of data items that can be signed by an attestation key, hashed, and/or encrypted. As per Section 7 of [RFC8392], the verification method is in the CWT using the CBOR Object Signing and Encryption (COSE) methodology (see [RFC8152]).

In addition, the reported attestation data could be determined within the secure operating environment or written to it from an external and presumably less trusted entity on the device. In either case, the source of the reported data must be identifiable by the relying party.

This attestation format is a single relatively simple signed message. It is designed to be incorporated into many other protocols and many other transports. It is also designed such that other SW and apps can add their own data to the message such that it is also attested.

1.3. EAT Operating Models

At least the following three participants exist in all EAT operating models. Some operating models have additional participants.

The Entity. This is the phone, the IoT device, the sensor, the sub-assembly or such that the attestation provides information about.

The Manufacturer. The company that made the entity. This may be a chip vendor, a circuit board module vendor or a vendor of finished consumer products.

The Relying Party. The server, service or company that makes use of the information in the EAT about the entity.

In all operating models, the manufacturer provisions some secret attestation key material (AKM) into the entity during manufacturing. This might be during the manufacturer of a chip at a fabrication facility (fab) or during final assembly of a consumer product or any time in between. This attestation key material is used for signing EATs.

In all operating models, hardware and/or software on the entity create an EAT of the format described in this document. The EAT is always signed by the attestation key material provisioned by the manufacturer.

In all operating models, the relying party must end up knowing that the signature on the EAT is valid and consistent with data from

claims in the EAT. This can happen in many different ways. Here are some examples.

- o The EAT is transmitted to the relying party. The relying party gets corresponding key material (e.g. a root certificate) from the manufacturer. The relying party performs the verification.
- o The EAT is transmitted to the relying party. The relying party transmits the EAT to a verification service offered by the manufacturer. The server returns the validated claims.
- o The EAT is transmitted directly to a verification service, perhaps operated by the manufacturer or perhaps by another party. It verifies the EAT and makes the validated claims available to the relying party. It may even modify the claims in some way and re-sign the EAT (with a different signing key).

This standard supports all these operating models and does not prefer one over the other. It is important to support this variety of operating models to generally facilitate deployment and to allow for some special scenarios. One special scenario has a validation service that is monetized, most likely by the manufacturer. In another, a privacy proxy service processes the EAT before it is transmitted to the relying party. In yet another, symmetric key material is used for signing. In this case the manufacturer should perform the verification, because any release of the key material would enable a participant other than the entity to create valid signed EATs.

1.4. What is Not Standardized

1.4.1. Transmission Protocol

EATs may be transmitted by any protocol. For example, they might be added in extension fields of other protocols, bundled into an HTTP header, or just transmitted as files. This flexibility is intentional to allow broader adoption. This flexibility is possible because EAT's are self-secured with signing (and possibly additionally with encryption and anti-replay). The transmission protocol is not required to fulfill any additional security requirements.

For certain devices, a direct connection may not exist between the EAT-producing device and the Relying Party. In such cases, the EAT should be protected against malicious access. The use of COSE allows for signing and encryption of the EAT. Therefore even if the EAT is conveyed through intermediaries between the device and Relying Party,

such intermediaries cannot easily modify the EAT payload or alter the signature.

1.4.2. Signing Scheme

The term "signing scheme" is used to refer to the system that includes end-end process of establishing signing attestation key material in the entity, signing the EAT, and verifying it. This might involve key IDs and X.509 certificate chains or something similar but different. The term "signing algorithm" refers just to the algorithm ID in the COSE signing structure. No particular signing algorithm or signing scheme is required by this standard.

There are three main implementation issues driving this. First, secure non-volatile storage space in the entity for the attestation key material may be highly limited, perhaps to only a few hundred bits, on some small IoT chips. Second, the factory cost of provisioning key material in each chip or device may be high, with even millisecond delays adding to the cost of a chip. Third, privacy-preserving signing schemes like ECDA (Elliptic Curve Direct Anonymous Attestation) are complex and not suitable for all use cases.

Eventually some form of standardization of the signing scheme may be required. This might come in the form of another standard that adds to this document, or when there is clear convergence on a small number of signing schemes this standard can be updated.

2. Terminology

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.

This document reuses terminology from JWT [RFC7519], COSE [RFC8152], and CWT [RFC8392].

StringOrURI. The "StringOrURI" term in this specification has the same meaning and processing rules as the JWT "StringOrURI" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR text string instead of a JSON text string.

NumericDate. The "NumericDate" term in this specification has the same meaning and processing rules as the JWT "NumericDate" term defined in Section 2 of [RFC7519], except that it is represented as a CBOR numeric date (from Section 2.4.1 of [RFC7049]) instead

of a JSON number. The encoding is modified so that the leading tag 1 (epoch-based date/time) MUST be omitted.

Claim Name. The human-readable name used to identify a claim.

Claim Key. The CBOR map key used to identify a claim.

Claim Value. The CBOR map value representing the value of the claim.

CWT Claims Set. The CBOR map that contains the claims conveyed by the CWT.

FloatOrNumber. The "FloatOrNumber" term in this specification is the type of a claim that is either a CBOR positive integer, negative integer or floating point number.

Attestation Key Material (AKM). The key material used to sign the EAT token. If it is done symmetrically with HMAC, then this is a simple symmetric key. If it is done with ECC, such as an IEEE DevID [IDevID], then this is the private part of the EC key pair. If ECDAAs are used, (e.g., as used by Enhanced Privacy ID, i.e. EPID) then it is the key material needed for ECDAAs.

3. The Claims

3.1. Universal Entity ID (UEID) Claim

UEIDs identify individual manufactured entities / devices such as a mobile phone, a water meter, a Bluetooth speaker or a networked security camera. It may identify the entire device or a submodule or subsystem. It does not identify types, models or classes of devices. It is akin to a serial number, though it does not have to be sequential.

It is identified by Claim Key X (X is TBD).

UEIDs must be universally and globally unique across manufacturers and countries. UEIDs must also be unique across protocols and systems, as tokens are intended to be embedded in many different protocols and systems. No two products anywhere, even in completely different industries made by two different manufacturers in two different countries, should have the same UEID (if they are not global and universal in this way then relying parties receiving them will have to track other characteristics of the device to keep devices distinct between manufacturers).

The UEID should be permanent. It should never change for a given device / entity. In addition, it should not be reprogrammable.

UEID's are binary byte-strings (resulting in a smaller size than text strings). When handled in text-based protocols, they should be base-64 encoded.

UEID's are variable length with a maximum size of 33 bytes (1 type byte and 256 bits). A receivers of a token with UEIDs may reject the token if a UEID is larger than 33 bytes.

UEID's are not designed for direct use by humans (e.g., printing on the case of a device), so no textual representation is defined.

A UEID is a byte string. From the consumer's view (the rely party) it is opaque with no bytes having any special meaning.

When the entity constructs the UEID, the first byte is a type and the following bytes the ID for that type. Several types are allowed to accommodate different industries and different manufacturing processes and to give options to avoid paying fees for certain types of manufacturer registrations.

Type Byte	Type Name	Specification
0x01	GUID	This is a 128 to 256 bit random number generated once and stored in the device. The GUID may be constructed from various identifiers on the device using a hash function or it may be just the raw random number. In any case, the random number must have entropy of at least 128 bits as this is what gives the global
0x02	IEEE EUI	This makes use of the IEEE company identification registry. An EUI is made up of an OUI and OUI-36 or a CID, different registered company identifiers, and some unique per-device identifier. EUIs are often the same as or similar to MAC addresses. (Note that while devices with multiple network interfaces may have multiple MAC addresses, there is only one UEID for a device) TODO: normative references to IEEE.
0x03	IMEI	TODO: figure how to specify IMEIs

Table 1: UEID Composition Types

The consumer (the Relying Party) of a UEID should treat a UEID as a completely opaque string of bytes and not make any use of its internal structure. For example they should not use the OUI part of

a type 0x02 UEID to identify the manufacturer of the device. Instead they should use the OUI claim that is defined elsewhere. The reasons for this are:

- o UEIDs types may vary freely from one manufacturer to the next.
- o New types of UEIDs may be created. For example a type 0x04 UEID may be created based on some other manufacturer registration scheme.
- o Device manufacturers are allowed to change from one type of UEID to another anytime they want. For example they may find they can optimize their manufacturing by switching from type 0x01 to type 0x02 or vice versa. The main requirement on the manufacturer is that UEIDs be universally unique.

3.2. Origination (origination) Claims

This claim describes the parts of the device or entity that are creating the EAT. Often it will be tied back to the device or chip manufacturer. The following table gives some examples:

Name	Description
Acme-TEE	The EATs are generated in the TEE authored and configured by "Acme"
Acme-TPM	The EATs are generated in a TPM manufactured by "Acme"
Acme-Linux-Kernel	The EATs are generated in a Linux kernel configured and shipped by "Acme"
Acme-TA	The EATs are generated in a Trusted Application (TA) authored by "Acme"

The claim is represented by Claim Key X+1. It is type StringOrURI.

TODO: consider a more structure approach where the name and the URI and other are in separate fields.

TODO: This needs refinement. It is somewhat parallel to issuer claim in CWT in that it describes the authority that created the token.

3.3. OEM identification by IEEE OUI

This claim identifies a device OEM by the IEEE OUI. Reference TBD. It is a byte string representing the OUI in binary form in network byte order (TODO: confirm details).

Companies that have more than one IEEE OUI registered with IEEE should pick one and prefer that for all their devices.

Note that the OUI is in common use as a part of MAC Address. This claim is only the first bits of the MAC address that identify the manufacturer. The IEEE maintains a registry for these in which many companies participate. This claim is represented by Claim Key TBD.

3.4. Security Level (seclevel) Claim

EATs have a claim that roughly characterizes the device / entities ability to defend against attacks aimed at capturing the signing key, forging claims and at forging EATs. This is done by roughly defining four security levels as described below. This is similar to the security levels defined in the Metadata Service defined by the Fast Identity Online (FIDO) Alliance (TODO: reference).

These claims describe security environment and countermeasures available on the end-entity / client device where the attestation key reside and the claims originate.

This claim is identified by Claim Key X+2. The value is an integer between 1 and 4 as defined below.

- 1 - Unrestricted There is some expectation that implementor will protect the attestation signing keys at this level. Otherwise the EAT provides no meaningful security assurances.
- 2- Restricted Entities at this level should not be general-purpose operating environments that host features such as app download systems, web browsers and complex productivity applications. It is akin to the Secure Restricted level (see below) without the security orientation. Examples include a WiFi subsystem, an IoT camera, or sensor device.
- 3 - Secure Restricted Entities at this level must meet the criteria defined by FIDO Allowed Restricted Operating Environments (TODO: reference). Examples include TEE's and schemes using virtualization-based security. Like the FIDO security goal, security at this level is aimed at defending well against large-scale network / remote attacks against the device.
- 4 - Hardware Entities at this level must include substantial defense against physical or electrical attacks against the device itself. It is assumed any potential attacker has captured the device and can disassemble it. Example include TPMs and Secure Elements.

This claim is not intended as a replacement for a proper end-device security certification schemes such as those based on FIPS (TODO: reference) or those based on Common Criteria (TODO: reference). The claim made here is solely a self-claim made by the Entity Originator.

3.5. Nonce (nonce) Claim

The "nonce" (Nonce) claim represents a random value that can be used to avoid replay attacks. This would be ideally generated by the CWT consumer. This value is intended to be a CWT companion claim to the existing JWT claim `**_IANAJWT_` (TODO: fix this reference). The nonce claim is identified by Claim Key X+3.

3.6. Secure Boot and Debug Enable State Claims

3.6.1. Secure Boot Enabled (secbootenabled) Claim

The "secbootenabled" (Secure Boot Enabled) claim represents a boolean value that indicates whether secure boot is enabled either for an entire device or an individual submodule. If it appears at the device level, then this means that secure boot is enabled for all submodules. Secure boot enablement allows a secure boot loader to authenticate software running either in a device or a submodule prior allowing execution. This claim is identified by Claim Key X+4.

3.6.2. Debug Disabled (debugdisabled) Claim

The "debugdisabled" (Debug Disabled) claim represents a boolean value that indicates whether debug capabilities are disabled for an entity (i.e. value of 'true'). Debug disablement is considered a prerequisite before an entity is considered operational. This claim is identified by Claim Key X+5.

3.6.3. Debug Disabled Since Boot (debugdisabledsinceboot) Claim

The "debugdisabledsinceboot" (Debug Disabled Since Boot) claim represents a boolean value that indicates whether debug capabilities for the entity were not disabled in any way since boot (i.e. value of 'true'). This claim is identified by Claim Key X+6.

3.6.4. Debug Permanent Disable (debugpermanentdisable) Claim

The "debugpermanentdisable" (Debug Permanent Disable) claim represents a boolean value that indicates whether debug capabilities for the entity are permanently disabled (i.e. value of 'true'). This value can be set to 'true' also if only the manufacturer is allowed to enable debug, but the end user is not. This claim is identified by Claim Key X+7.

3.6.5. Debug Full Permanent Disable (debugfullpermanentdisable) Claim

The "debugfullpermanentdisable" (Debug Full Permanent Disable) claim represents a boolean value that indicates whether debug capabilities for the entity are permanently disabled (i.e. value of 'true'). This value can only be set to 'true' if no party can enable debug capabilities for the entity. Often this is implemented by blowing a fuse on a chip as fuses cannot be restored once blown. This claim is identified by Claim Key X+8.

3.7. Location (loc) Claim

The "loc" (location) claim is a CBOR-formatted object that describes the location of the device entity from which the attestation originates. It is identified by Claim Key X+10. It is comprised of an array of additional subclaims that represent the actual location coordinates (latitude, longitude and altitude). The location coordinate claims are consistent with the WGS84 coordinate system [WGS84]. In addition, a subclaim providing the estimated accuracy of the location measurement is defined.

3.7.1. lat (latitude) claim

The "lat" (latitude) claim contains the value of the device location corresponding to its latitude coordinate. It is of data type FloatOrNumber and identified by Claim Key X+11.

3.7.2. long (longitude) claim

The "long" (longitude) claim contains the value of the device location corresponding to its longitude coordinate. It is of data type FloatOrNumber and identified by Claim Key X+12.

3.7.3. alt (altitude) claim

The "alt" (altitude) claim contains the value of the device location corresponding to its altitude coordinate (if available). It is of data type FloatOrNumber and identified by Claim Key X+13.

3.7.4. acc (accuracy) claim

The "acc" (accuracy) claim contains a value that describes the location accuracy. It is non-negative and expressed in meters. It is of data type FloatOrNumber and identified by Claim Key X+14.

3.7.5. altacc (altitude accuracy) claim

The "altacc" (altitude accuracy) claim contains a value that describes the altitude accuracy. It is non-negative and expressed in meters. It is of data type FloatOrNumber and identified by Claim Key X+15.

3.7.6. heading claim

The "heading" claim contains a value that describes direction of motion for the entity. Its value is specified in degrees, between 0 and 360. It is of data type FloatOrNumber and identified by Claim Key X+16.

3.7.7. speed claim

The "speed" claim contains a value that describes the velocity of the entity in the horizontal direction. Its value is specified in meters/second and must be non-negative. It is of data type FloatOrNumber and identified by Claim Key X+17.

3.8. ts (timestamp) claim

The "ts" (timestamp) claim contains a timestamp derived using the same time reference as is used to generate an "iat" claim (see Section 3.1.6 of [RFC8392]). It is of the same type as "iat" (integer or floating-point), and is identified by Claim Key X+18. It is meant to designate the time at which a measurement was taken, when a location was obtained, or when a token was actually transmitted. The timestamp would be included as a subclaim under the "submod" or "loc" claims (in addition to the existing respective subclaims), or at the device level.

3.9. age claim

The "age" claim contains a value that represents the number of seconds that have elapsed since the token was created, measurement was made, or location was obtained. Typical attestable values are sent as soon as they are obtained. However in the case that such a value is buffered and sent at a later time and a sufficiently accurate time reference is unavailable for creation of a timestamp, then the age claim is provided. It is identified by Claim Key X+19.

3.10. uptime claim

The "uptime" claim contains a value that represents the number of seconds that have elapsed since the entity or submod was last booted. It is identified by Claim Key X+20.

3.11. The submods Claim

Some devices are complex, having many subsystems or submodules. A mobile phone is a good example. It may have several connectivity submodules for communications (e.g., WiFi and cellular). It may have sub systems for low-power audio and video playback. It may have one or more security-oriented subsystems like a TEE or a Secure Element.

The claims for each these can be grouped together in a submodule.

Specifically, the "submods" claim is an array. Each item in the array is a CBOR map containing all the claims for a particular submodule. It is identified by Claim Key X+22.

The security level of the submod is assumed to be at the same level as the main entity unless there is a security level claim in that submodule indicating otherwise. The security level of a submodule can never be higher (more secure) than the security level of the EAT it is a part of.

3.11.1. The submod_name Claim

Each submodule should have a submod_name claim that is descriptive name. This name should be the CBOR txt type.

3.11.2. Nested EATs, the eat Claim

It is allowed for one EAT to be embedded in another. This is for complex devices that have more than one subsystem capable of generating an EAT. Typically one will be the device-wide EAT that is low to medium security and another from a Secure Element or similar that is high security.

The contents of the "eat" claim must be a fully signed, optionally encrypted, EAT token. It is identified by Claim Key X+23.

4. CBOR Interoperability

EAT is a one-way protocol. It only defines a single message that goes from the entity to the server. The entity implementation will often be in a contained environment with little RAM and the server will usually not be. The following requirements for interoperability take that into account. The entity can generally use whatever encoding it wants. The server is required to support just about every encoding.

Canonical CBOR encoding is explicitly NOT required as it would place an unnecessary burden on the entity implementation.

4.1. Integer Encoding (major type 0 and 1)

The entity may use any integer encoding allowed by CBOR. The server MUST accept all integer encodings allowed by CBOR.

4.2. String Encoding (major type 2 and 3)

The entity can use any string encoding allowed by CBOR including indefinite lengths. It may also encode the lengths of strings in any way allowed by CBOR. The server must accept all string encodings.

Major type 2, bstr, SHOULD be have tag 21, 22 or 23 to indicate conversion to base64 or such when converting to JSON.

4.3. Map and Array Encoding (major type 4 and 5)

The entity can use any array or map encoding allowed by CBOR including indefinite lengths. Sorting of map keys is not required. Duplicate map keys are not allowed. The server must accept all array and map encodings. The server may reject maps with duplicate map keys.

4.4. Date and Time

The entity should send dates as tag 1 encoded as 64-bit or 32-bit integers. The entity may not send floating point dates. The server must support tag 1 epoch based dates encoded as 64-bit or 32-bit integers.

The entity may send tag 0 dates, however tag 1 is preferred. The server must support tag 0 UTC dates.

4.5. URIs

URIs should be encoded as text strings and marked with tag 32.

4.6. Floating Point

Encoding data in floating point is to be used only if necessary. Location coordinates are always in floating point. The server must support decoding of all types of floating point.

4.7. Other types

Use of Other types like bignums, regular expressions and so SHOULD NOT be used. The server MAY support them, but is not required to. Use of these tags is

5. IANA Considerations

5.1. Reuse of CBOR Web Token (CWT) Claims Registry

Claims defined for EAT are compatible with those of CWT so the CWT Claims Registry is re used. New new IANA registry is created. All EAT claims should be registered in the CWT Claims Registry.

5.1.1. Claims Registered by This Document

- o Claim Name: UEID
- o Claim Description: The Universal Entity ID
- o JWT Claim Name: N/A
- o Claim Key: X
- o Claim Value Type(s): byte string
- o Change Controller: IESG
- o Specification Document(s): *this document*

TODO: add the rest of the claims in here

5.2. EAT CBOR Tag Registration

How an EAT consumer determines whether received CBOR-formatted data actually represents a valid EAT is application-dependent, much like a CWT. For instance, a specific MIME type associated with the EAT such as "application/eat" could be sufficient for identification of the EAT. Note however that EAT's can include other EAT's (e.g. a device EAT comprised of several submodule EAT's). In this case, a CBOR tag dedicated to the EAT will be required at least for the submodule EAT's and the tag must be a valid CBOR tag. In other words - the EAT CBOR tag can optionally prefix a device-level EAT, but a EAT CBOR tag must always prefix a submodule EAT. The proposed EAT CBOR tag is 71.

5.2.1. Tag Registered by This Document

- o CBOR Tag: 71
- o Data Item: Entity Attestation Token (EAT)
- o Semantics: Entity Attestation Token (CWT), as defined in *this_doc*

- o Reference: *this_doc*
- o Point of Contact: Giridhar Mandyam, mandyam@qti.qualcomm.com

6. Privacy Considerations

Certain EAT claims can be used to track the owner of an entity and therefore implementations should consider providing privacy-preserving options dependent on the intended usage of the EAT. Examples would include suppression of location claims for EAT's provided to unauthenticated consumers.

6.1. UEID Privacy Considerations

A UEID is usually not privacy preserving. Any set of relying parties that receives tokens that happen to be from a single device will be able to know the tokens are all from the same device and be able to track the device. Thus, in many usage situations ueid violates governmental privacy regulation. In other usage situations UEID will not be allowed for certain products like browsers that give privacy for the end user. it will often be the case that tokens will not have a UEID for these reasons.

There are several strategies that can be used to still be able to put UEID's in tokens:

- o The device obtains explicit permission from the user of the device to use the UEID. This may be through a prompt. It may also be through a license agreement. For example, agreements for some online banking and brokerage services might already cover use of a UEID.
- o The UEID is used only in a particular context or particular use case. It is used only by one relying party.
- o The device authenticates the relying party and generates a derived UEID just for that particular relying party. For example, the relying party could prove their identity cryptographically to the device, then the device generates a UEID just for that relying party by hashing a proofed relying party ID with the main device UEID.

Note that some of these privacy preservation strategies result in multiple UEIDs per device. Each UEID is used in a different context, use case or system on the device. However, from the view of the relying party, there is just one UEID and it is still globally universal across manufacturers.

7. Security Considerations

TODO: Perhaps this can be the same as CWT / COSE, but not sure yet because it involves so much entity / device security that those do not.

8. References

8.1. Normative References

- [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>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <<https://www.rfc-editor.org/info/rfc7519>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [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>>.
- [RFC8392] Jones, M., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "CBOR Web Token (CWT)", RFC 8392, DOI 10.17487/RFC8392, May 2018, <<https://www.rfc-editor.org/info/rfc8392>>.
- [TIME_T] The Open Group Base Specifications, "Vol. 1: Base Definitions, Issue 7", Section 4.15 'Seconds Since the Epoch', IEEE Std 1003.1, 2013 Edition, 2013, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_15>.
- [WGS84] National Imagery and Mapping Agency, "National Imagery and Mapping Agency Technical Report 8350.2, Third Edition", 2000, <<http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf>>.

8.2. Informative References

- [ASN.1] International Telecommunication Union, "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, 1994.
- [IDevID] "IEEE Standard, "IEEE 802.1AR Secure Device Identifier", December 2009, <<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>>.
- [Webauthn] Worldwide Web Consortium, "Web Authentication: A Web API for accessing scoped credentials", 2016.

Appendix A. Examples

A.1. Very Simple EAT

This is shown in CBOR diagnostic form. Only the payload signed by COSE is shown.

```
{
  / nonce /                11:h'948f8860d13a463e8e',
  / UEID /                 8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
  / secbootenabled /      13:true,
  / debugpermanentdisable / 15:true,
  / ts /                  21:1526542894,
}
```

A.2. Example with Submodules, Nesting and Security Levels

```
{
  / nonce /                11:h'948f8860d13a463e8e',
  / UEID /                 8:h'0198f50a4ff6c05861c8860d13a638ea4fe2f',
  / secbootenabled /      13:true,
  / debugpermanentdisable / 15:true,
  / ts /                  21:1526542894,
  / seclevel /            10:3, / secure restriced OS /

  / submods / 30:
  [
    / 1st submod, an Android Application / {
      / submod_name /    30:'Android App "Foo"',
      / seclevel /      10:1, / unrestricted /
      / app data /      -70000:'text string'
    },
    / 2nd submod, A nested EAT from a secure element / {
      / submod_name /    30:'Secure Element EAT',
      / eat /            31:71( 18(
        / an embedded EAT / [ /...COSE_Sign1 bytes with payload.../ ]
      ) )
    }
    / 3rd submod, information about Linux Android / {
      / submod_name/    30:'Linux Android',
      / seclevel /     10:1, / unrestricted /
      / custom - release / -80000:'8.0.0',
      / custom - version / -80001:'4.9.51+'
    }
  ]
}
```

Authors' Addresses

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 7200
EMail: mandyam@qti.qualcomm.com

Laurence Lundblade
Security Theory LLC

EMail: lgl@island-resort.com

Miguel Ballesteros
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California
USA

Phone: +1 858 651 4299
EMail: mballest@qti.qualcomm.com

Jeremy O'Donoghue
Qualcomm Technologies Inc.
279 Farnborough Road
Farnborough GU14 7LS
United Kingdom

Phone: +44 1252 363189
EMail: jodonogh@qti.qualcomm.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 30, 2018

N. Mavrogiannopoulos
Red Hat
March 29, 2018

Storing validation parameters in PKCS#8
draft-mavrogiannopoulos-pkcs8-validated-parameters-02

Abstract

This memo describes a method of storing parameters needed for private key validation in the Private-Key Information Syntax Specification as defined in RFC5208 (PKCS#8) format. It is equally applicable the alternative implementation of the Private-Key Information Syntax Specification as defined in RFC 5958.

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 <http://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 September 30, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

RSA or DSA private keys generated using the Shawe-Taylor prime generation algorithm described in [FIPS186-4] allow for parameter validation, i.e., verify whether the primes are actually prime, and were correctly generated. That is done by generating the parameters from a known seed and a selected hash algorithm.

Storing these parameters in a private key format such as the RSA Private Key Syntax from PKCS#1 [RFC8017], or common representations for DSA private keys, does not allow attaching information on the parameters needed for validation. The purpose of the document is to describe such a method using the Private-Key Information Syntax Specification as defined in [RFC5208], as well as on the alternative specification on [RFC5958].

2. Terminology

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 BCP14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

3. ValidationParams attribute

The information related to the validation parameters is stored as an attribute in the PrivateKeyInfo structure. The attribute is identified by the id-attr-validation-parameters object identifier and contains as AttributeValue a single ValidationParams structure.

```
id-attr-validation-parameters OBJECT IDENTIFIER ::=
    {1 3 6 1 4 1 2312 18 8 1}

ValidationParams ::= SEQUENCE {
    hashAlgo OBJECT IDENTIFIER,
    seed OCTET STRING
}
```

The algorithm identifier in the ValidationParams should be a hash algorithm identifier for the [FIPS186-4] methods. The ValidationParams sequence must be DER encoded [CCITT.X690.2002].

4. Example Structure

The following structure contains an RSA key generated using the [FIPS186-4] section B.3.3 algorithm with SHA2-384 hash. The seed used is

'8af4328c87bebec31e303b8f5537effcb6a91d947084d99a369823b36f01462'
(hex encoded).

-----BEGIN PRIVATE KEY-----
MIIE/gIBADANBgkqhkiG9w0BAQEFAASCBCwggSjAgEAAoIBAQCpPwXwfHdsWA3q
jN2BWglxfDjvZDVNfgTV/b95g304Aty3z13xPXAhHZ3ROW3pgPxTj9fiq7ZMy4Ua
gMpPK81v3pHXluokC2KcGXbgbAq2Q8ClxSXgEJl1RwDENufjEdV10gArt8N1IP0N
lotalkQUuI1DMSqc5DTIa35Nq4j1GW+KmltP0kCrGq9fMGwJdbPEpSp9DTquEMHJ
o7kyJIjB+93ikLvBUTgbxr+jcnTLXuhA8rC8r+KXre4NPPNPRyefRcALLt/URvfA
rTvFOQfi3viJnHBZL5FdC+FVar5QnF3r2+cuDPbnczr4/rr81kzFGWrwyAgF5FWu
pFtB5IYDAgMBAAECggeAHZ88vGNsNdmRkfhWupGW4cKCuo+Y7re8Q/H2Jd/4Nin2
FKvUPuloaztiSGDbVm+vejama/Nu5FEIumNJRYMeoVJcx2DDuUxO1ZBlaIEwfMct
/DWd0/JDzuCXB0Cu5GTWLhlz0zMGHXihIdQ0DtGKt++3Ncg5gy1D+cIqqJB515/z
jYdZmb0Wqgz7H3DisuxvnhiCAOuNrjcDau80hpMA9TQlb+XKNGHIBgKpJe6lnB0P
MsS/AjDiDoEpP9GG9mv9+96rAga4Nos6avYlwWwbC6d+hHIWvWEWsmrDfcJlm2gN
tjvG8omj00t5dAt7qGhfOoNDGr5tvJVo/g960/0I8QKBgQDdzytVRulo9aKVdAYW
/Nj04thtnRaqsTyFH+7ibEVwNIUuld/Bp6NnuGrY+Klsix8+zA9f8mKxuXXV9KK4
O89Ypw9js2BxM7VYO9Gmp6elRY3Rrd8w7pG7/KqoPWXkuixTay9eybrJMWu3TT36
q7NheNmBHqcFmSQQuUwEmvp3MQKBgQDDVaisMJkc/sIyQh3XrlfzmMLK+G1PDucD
w5e50fh18Q5PmTcP20zVLhTevffCqeItSyeAno94Xdzc9vZ/rt69410kJEHyB09L
CmhtYz94wvSdRhbqf4VzAl2WU184sIYiIZDGsnGScgIYvo6v6mITjRhc8AMdYoPR
rL6xp6frcwKBgFil+avCj6mFzD+fxqu89nyCmXLfiAI+nmjTy7PM/7yPlNB76qDG
Dil2bw1Xj+y/1R9ld6S1CVnxRbqLe+TZLuVS82m5nRHJT3b5fbD8jquGJOE++xT
DgA0XoCpBa6D8yRt0uVDIyxCUSVd5DL0JusN7VehzcUEaZMyuL+CyDeRAoGBAImB
qH6mq3Kc6K0mnw1w4ttJ436sxrlvuTKOIyYdZBNB0Zg5PGi+MWU0z15LDroLi3vl
FwbVGBxcvxkSBU63FhhKMqW7Ne0gii+iQQcYQdtKKpb4ezNS1+exd55WTicExtgL
tvYZMhgsh8tRgflWpXor7kWmdBrgeflfiOxZIL1/AoGAeBP7sdE+gzsh8jqFnVRj
7nOg+Y1l1JAlWsf7cTH4pLIy2Eo9D+cNjhL9LK6RaAd7PSZladm8HfaroA2cfCm84
RI4c7Ue0G+N6LZiFvC0Bfi5SaPVAExXoty8UqjOCozavSaXBPuNcTXZuzswcgbxI
G5/kaJNHoeCdlVsPsYWKRNKGpZa9BgorBgEEAZIIEggBMS8wLQYJYIZIAWUDBAIC
BCKK9DKMh7687DHJA7j1U37/y2qR2UcITZmjaYI7NvAUyg==
-----END PRIVATE KEY-----

5. Compatibility notes

For compatibility it is RECOMMENDED that implementations following this document, support generation and validation using the SHA2-384 hash algorithm.

The extension defined in this document is applicable both to the Private-Key Information Syntax Specification defined in [RFC5958] and PKCS#8 [RFC5208].

6. Security Considerations

All the considerations in [RFC5208] and [RFC5958] apply.

7. IANA Considerations

None.

8. References

8.1. Normative References

- [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>>.
- [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>>.
- [RFC5208] Kaliski, B., "Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2", RFC 5208, DOI 10.17487/RFC5208, May 2008, <<https://www.rfc-editor.org/info/rfc5208>>.
- [CCITT.X680.2002] International International Telephone and Telegraph Consultative Committee, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", CCITT Recommendation X.680, July 2002.
- [CCITT.X690.2002] International International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.
- [FIPS186-4] Kerry, C. and P. Gallagher, "FIPS PUB 186-4: Digital Signature Standard (DSS)", FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION , July 2013.
- [RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <<https://www.rfc-editor.org/info/rfc5958>>.

8.2. Informative References

- [RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <<https://www.rfc-editor.org/info/rfc8017>>.
- [RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <<https://www.rfc-editor.org/info/rfc5912>>.

Appendix A. Acknowledgements

The author would like to thank Russ Housley for his comments and for the ASN.1 module appendix.

Appendix B. ASN.1 module

This appendix provides non-normative ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [CCITT.X680.2002] and [RFC5912].

```
PrivateKeyValidationAttrV1
  { iso(1) identified-organization(3) dod(6) internet(1)
    private(4) enterprise(1) 2312 18 1 1 }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL

IMPORTS

ATTRIBUTE
  FROM PKIX-CommonTypes-2009 -- [RFC5912]
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-pkixCommon-02(57) } ;

-- PrivateKeyInfo is defined in [RFC5208].
-- This definition adds the validation parameters attribute
-- to the set of allowed attributes.

PrivateKeyInfo ATTRIBUTE ::= {
  at-validation-parameters, ... }

at-validation-parameters ATTRIBUTE ::= {
  TYPE ValidationParams
  IDENTIFIED BY id-attr-validation-parameters }

id-attr-validation-parameters OBJECT IDENTIFIER ::=
  { 1 3 6 1 4 1 2312 18 8 1 }

ValidationParams ::= SEQUENCE {
  hashAlg OBJECT IDENTIFIER,
  seed OCTET STRING }

END
```

Author's Address

Nikos Mavrogiannopoulos
Red Hat, Inc.
Brno
Czech Republic

Email: nmav@redhat.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: February 22, 2019

N. Mavrogiannopoulos
Red Hat
August 21, 2018

Storing validation parameters in PKCS#8
draft-mavrogiannopoulos-pkcs8-validated-parameters-04

Abstract

This memo describes a method of storing parameters needed for private key validation in the Private-Key Information Syntax Specification as defined in RFC5208 (PKCS#8) format. It is equally applicable to the alternative implementation of the Private-Key Information Syntax Specification as defined in RFC 5958.

The approach described in this document encodes the parameters under a private enterprise extension and does not form part of a formal standard.

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 <http://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 February 22, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

RSA or DSA private keys generated using the Shawe-Taylor prime generation algorithm described in [FIPS186-4] allow for parameter validation, i.e., verify whether the primes are actually prime, and were correctly generated. That is done by generating the parameters from a known seed and a selected hash algorithm.

Storing these parameters in a private key format such as the RSA Private Key Syntax from PKCS#1 [RFC8017], or common representations for DSA private keys, does not allow attaching information on the parameters needed for validation. The purpose of the document is to describe such a method using the Private-Key Information Syntax Specification as defined in [RFC5208], as well as on the alternative specification on [RFC5958].

The approach described in this document encodes the parameters under a private enterprise extension and does not form part of a formal standard. The encoding can be used as is, or could be used as the basis for a standard at a later time.

2. ValidationParams attribute

The information related to the validation parameters is stored as an attribute in the PrivateKeyInfo structure. The attribute is identified by the id-attr-validation-parameters object identifier and contains as AttributeValue a single ValidationParams structure.

```
id-attr-validation-parameters OBJECT IDENTIFIER ::=
    {1 3 6 1 4 1 2312 18 8 1}

ValidationParams ::= SEQUENCE {
    hashAlgo OBJECT IDENTIFIER,
    seed OCTET STRING
}
```

The algorithm identifier in the ValidationParams should be a hash algorithm identifier for the [FIPS186-4] methods. The ValidationParams sequence must be DER encoded [CCITT.X690.2002].

3. Example Structure

The following structure contains an RSA key generated using the [FIPS186-4] section B.3.3 algorithm with SHA2-384 hash. The seed used is

```
'8af4328c87bebcec31e303b8f5537effcb6a91d947084d99a369823b36f01462'  
(hex encoded).
```

```
-----BEGIN PRIVATE KEY-----  
MIIE/gIBADANBgkqhkiG9w0BAQEFAASCBCwggSjAgEAAoIBAQCpPwXfhdSWA3q  
jN2BWglxfDjvZDVNfgTV/b95g304Aty3z13xPXAhHZ3ROW3pgPxTj9fiq7ZMy4Ua  
gMpPK81v3pHX1uokC2KcGXbgbAq2Q8ClxSXgEJllRwDENufjEdV10gArt8NlIP0N  
lotalkQUuI1DMSqc5DTIa35Nq4j1GW+KmltP0kCrGq9fMGwjDbPEpSp9DTquEMHJ  
o7kyJIjB+93ikLvBUTgboxr+jcnTLXuhA8rC8r+KXre4NPPNPRyefRcALLt/URvFA  
rTvFOQfi3vIjNhBZL5FdC+FVAR5QnF3r2+cuDPbnczr4/rr81kzFGWrwyAgF5FWu  
pFtB5IYDagMBAAECggEAHZ88vGNsNdmRkfHwupGW4cKCuo+Y7reQ/H2Jd/4Nin2  
FKvUPuloaztiSGDbVm+vejama/Nu5FEIumNJRYMeoVJcx2DDuUx01ZB1aIEwfMct  
/DWD0/JDzuCXB0Cu5GTWLhlz0zMGHXihIdQ0DtGkt++3Ncg5gy1D+cIqqJB515/z  
jYdZmb0Wqmz7H3DisuxvnhICAouNrjcDau80hpMA9TQlb+XKNGHIBgKpJe6lnB0P  
MsS/AjDiDoEpP9GG9mv9+96rAga4Nos6avYlwWwbC6d+hHIWvWEWsmrDfcJlm2gN  
tjvG8omj00t5dAt7qGhfOoNDGr5tvJVo/g960/0I8QKBgQDdzytVRulo9aKVdAYW  
/Nj04thtnRaqsTyFH+7ibEVwNIUuld/Bp6NnuGrY+K1siX8+zA9f8mKxuXXV9KK4  
O89Ypw9js2BxM7VYO9Gmp6e1RY3Rrd8w7pG7/KqoPWXkuixTay9eybrJMWu3TT36  
q7NheNmBHqcFmSQQQuUwEmvp3MQKBgQDDVaisMJkc/sIyQh3XrlfzmMLK+G1PDucD  
w5e50fHl8Q5PmTcP20zVLhTevffCqeItSyeAno94Xdzc9vZ/rt69410kJEHyB09L  
CmhtYz94wvSdRhbqf4VzAl2WU184sIYiIZDGsnGScgIYvo6v6mITjRhc8AMdYoPR  
rL6xp6frcwKBgFil+avCj6mFzD+fxqu89nyCmXLfiAI+nmjTy7PM/7yPlNB76qDG  
Dil2bw1Xj+y/1R9ld6S1CVnxRbqLe+TZLuVS82m5nRHJT3b5fbD8jquGJOE+e+xT  
DgA0XoCpBa6D8yRt0uVDIyxCUSVd5DL0JusN7VehzcUEaZMyuL+CyDeRAoGBAImB  
qH6mq3Kc6Konnw1w4ttJ436sxrlvuTKOIyYdZBNB0Zg5PGi+MWU0zl5LDroLi3vl  
FwbVGBxcvXkSBU63FHhKMqW7Ne0gii+iQqCYQdtKKpb4ezNS1+exd55WTicExtgL  
tvYZMhgsh8tRgflWpXor7kWmdBrgef1FiOxZIL1/AoGAeBP7sdE+gzsh8jqFnVRj  
7nOg+Y11JAlWsf7cTH4pLIy2Eo9D+cNjhL9LK6RaAd7PSZladm8HfaROA2cfCm84  
RI4c7Ue0G+N6LZiFvC0Bfi5SaPVAExXoty8UqjOCozavSaXBPuNcTXZuzswcgbxI  
G5/kaJNHoeCd1VsPsYWKRNKGPza9BgorBgEEAZIIEggBMS8wLQYJYIZIAWUDBAIC  
BCKK9DKMh7687DHja7j1U37/y2qr2UcITZmjaYI7NvAUYg==  
-----END PRIVATE KEY-----
```

4. Compatibility notes

For compatibility it is recommended that implementations following this document, support generation and validation using the SHA2-384 hash algorithm.

The extension defined in this document is applicable both to the Private-Key Information Syntax Specification defined in [RFC5958] and PKCS#8 [RFC5208].

5. Security Considerations

All the considerations in [RFC5208] and [RFC5958] apply.

6. IANA Considerations

None.

7. References

7.1. Normative References

[RFC5208] Kaliski, B., "Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2", RFC 5208, DOI 10.17487/RFC5208, May 2008, <<https://www.rfc-editor.org/info/rfc5208>>.

[CCITT.X680.2002] International International Telephone and Telegraph Consultative Committee, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", CCITT Recommendation X.680, July 2002.

[CCITT.X690.2002] International International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.

[FIPS186-4] Kerry, C. and P. Gallagher, "FIPS PUB 186-4: Digital Signature Standard (DSS)", FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION , July 2013.

[RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <<https://www.rfc-editor.org/info/rfc5958>>.

7.2. Informative References

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <<https://www.rfc-editor.org/info/rfc8017>>.

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <<https://www.rfc-editor.org/info/rfc5912>>.

Appendix A. Acknowledgements

The author would like to thank Russ Housley for his comments and for the ASN.1 module appendix.

Appendix B. ASN.1 module

This appendix provides non-normative ASN.1 definitions for the structures described in this specification using ASN.1 as defined in [CCITT.X680.2002] and [RFC5912].

```
PrivateKeyValidationAttrV1
  { iso(1) identified-organization(3) dod(6) internet(1)
    private(4) enterprise(1) 2312 18 1 1 }

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL

IMPORTS

ATTRIBUTE
  FROM PKIX-CommonTypes-2009 -- [RFC5912]
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-pkixCommon-02(57) } ;

-- PrivateKeyInfo is defined in [RFC5208].
-- This definition adds the validation parameters attribute
-- to the set of allowed attributes.

PrivateKeyInfo ATTRIBUTE ::= {
  at-validation-parameters, ... }

at-validation-parameters ATTRIBUTE ::= {
  TYPE ValidationParams
  IDENTIFIED BY id-attr-validation-parameters }

id-attr-validation-parameters OBJECT IDENTIFIER ::=
  { 1 3 6 1 4 1 2312 18 8 1 }

ValidationParams ::= SEQUENCE {
  hashAlg OBJECT IDENTIFIER,
  seed OCTET STRING }

END
```

Author's Address

Nikos Mavrogiannopoulos
Red Hat, Inc.
Brno
Czech Republic

Email: nmav@redhat.com

ACME
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2018

Y. Sheffer
Intuit
D. Lopez
O. Gonzalez de Dios
A. Pastor Perales
Telefonica I+D
T. Fossati
Nokia
June 29, 2018

Generating Certificate Requests for Short-Term, Automatically-Renewed
(STAR) Certificates
draft-sheffer-acme-star-request-02

Abstract

This memo proposes a protocol that allows a domain name owner to delegate to a third party (such as a CDN) control over a certificate that bears one or more names in that domain. Specifically the third party creates a Certificate Signing Request for the domain, which can then be used by the domain owner to request a short term and automatically renewed (STAR) certificate.

This is a component in a solution where a third-party such as a CDN can terminate TLS sessions on behalf of a domain name owner (e.g., a content provider), and the domain owner can cancel this delegation at any time without having to rely on certificate revocation mechanisms.

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 December 31, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Terminology	3
1.2. Conventions used in this document	4
2. Protocol Flow	4
2.1. Preconditions	4
2.2. Bootstrap	4
2.3. Refresh	6
2.4. Termination	7
3. Protocol Details	8
3.1. STAR API	8
3.1.1. Creating a Delegation Request	8
3.1.2. Polling the Delegation Request	10
3.2. Transport Security for the STAR Protocol	11
4. CDNI Use Cases	11
4.1. Multiple Parallel Delegates	11
4.2. Chained Delegation	11
5. Security Considerations	12
5.1. STAR Protocol Authentication	12
6. Acknowledgments	12
7. References	12
7.1. Normative References	12
7.2. Informative References	13
Appendix A. Document History	14
A.1. draft-sheffer-acme-star-request-02	14
A.2. draft-sheffer-acme-star-request-01	14
A.3. draft-sheffer-acme-star-request-00	14
Authors' Addresses	14

1. Introduction

This document is a companion document to [I-D.ietf-acme-star]. To avoid duplication, we give here a bare-bones description of the motivation for this solution. For more details and further use cases, please refer to the introductory sections of [I-D.ietf-acme-star].

A content provider (referred to in this document as Domain Name Owner, DNO, or more generally as Identity Owner, IdO) has agreements in place with one or more Content Delivery Networks (CDNs) that are contracted to serve its content over HTTPS. The CDN terminates the HTTPS connection at one of its edge cache servers and needs to present its clients (browsers, set-top-boxes) a certificate whose name matches the authority of the URL that is requested, i.e. that of the DNO. However, many DNOs balk at sharing their long-term private keys with another organization and, equally, delegates (henceforth referred to as NDC, Name Delegation Consumer) would rather not have to handle other parties' long-term secrets.

This document describes a protocol where the IdO and the NDC agree on a CSR template and the NDC generates a CSR for a private key that it holds. The IdO then uses the ACME protocol (as extended in [I-D.ietf-acme-star]) to issue the STAR certificate.

The generated short-term certificate is automatically renewed by an ACME Certification Authority (CA) [I-D.ietf-acme-acme] and periodically fetched into the NDC and used for HTTPS connections. The IdO can end the delegation at any time by simply instructing the CA to stop the automatic renewal and letting the certificate expire shortly thereafter.

1.1. Terminology

IdO Identity Owner, the owner of an identity (e.g., a domain name) that needs to be delegated.

DNO Domain Name Owner, a specific kind of IdO whose identity is a domain name.

NDC Name Delegation Consumer, the entity to which the domain name is delegated for a limited time. This is often a CDN (in fact, readers may note the similarity of the two acronyms).

CDN Content Delivery Network, a widely distributed network that serves the domain's web content to a wide audience at high performance.

STAR Short-Term, Automatically Renewed X.509 certificates.

ACME The IETF Automated Certificate Management Environment, a certificate management protocol.

CA A Certificate Authority that implements the ACME protocol.

1.2. Conventions used in this document

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 [RFC2119].

2. Protocol Flow

This section presents the protocol flow. For completeness, we include the STAR Interface proposed in this draft, as well as the extended ACME protocol as described in [I-D.ietf-acme-star].

2.1. Preconditions

The protocol assumes the following preconditions are met:

- o A mutually authenticated channel between NDC and IdO pre-exists. This is called "STAR channel" and all STAR protocol exchanges between NDC and IdO are run over it. It provides the guarantee that requests and responses are authentic.
- o NDC and IdO have agreed on a "CSR template" to use, including at a minimum:
 - Subject name (e.g., "somesite.example.com"),
 - Requested algorithms,
 - Key length,
 - Key usage.

The NDC is required to use this template for every CSR created under the same delegation.

- o IdO has registered through the ACME interface exposed by the Certificate Authority (CA) using the usual ACME registration procedure. In ACME terms, the IdO has an Account on the server and is ready to issue Orders.

2.2. Bootstrap

The NDC (STAR Client) generates a key-pair, wraps it into a Certificate Signing Request (CSR) according to the agreed upon CSR template, and sends it to the IdO (STAR Proxy) over the pre-established STAR channel. The IdO uses the NDC identity provided on the STAR channel to look up the CSR template that applies to the requesting NDC and decides whether or not to accept the request. Assuming everything is in order, it then "forwards" the NDC request to the ACME CA by means of the usual ACME application procedure. Specifically, the IdO, in its role as an ACME client, requests the CA to issue a STAR certificate, i.e., one that:

- o Has a short validity (e.g., 24 to 72 hours);
- o Is automatically renewed by the CA for a certain period of time;
- o Is downloadable from a (highly available) public link without requiring any special authorization.

Other than that, the ACME protocol flows as normal between IdO and CA, in particular IdO is responsible for satisfying the requested ACME challenges until the CA is willing to issue the requested certificate. Per normal ACME processing, the IdO is given back an Order ID for the issued STAR certificate to be used in subsequent interaction with the CA (e.g., if the certificate needs to be terminated.)

Concurrently, a response is sent back to the NDC with an endpoint to poll for completion of the certificate generation process.

The bootstrap phase ends when the IdO obtains the OK from the ACME CA and posts the certificate's URL to the "completion endpoint" where the NDC can retrieve it.

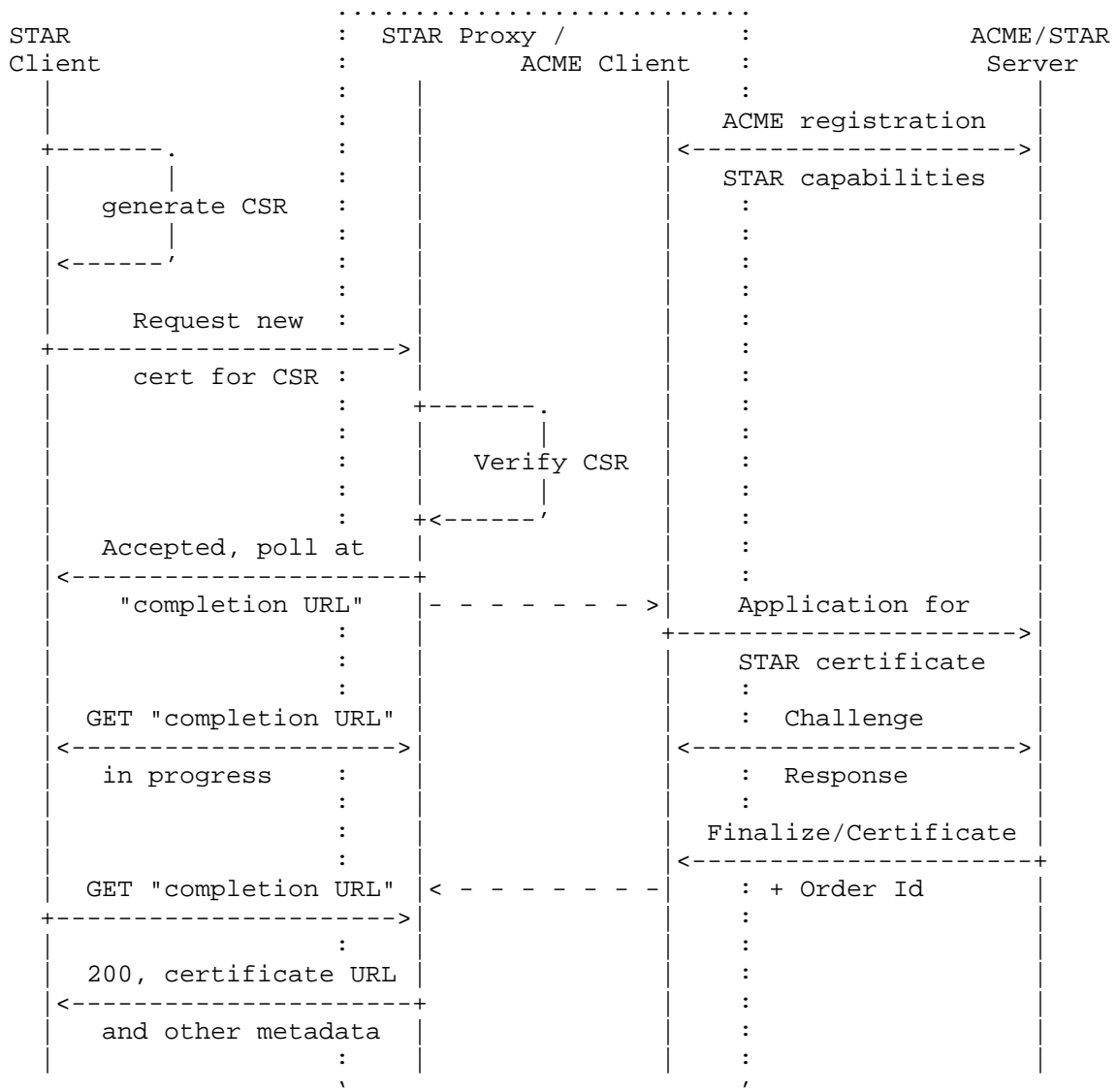


Figure 1: Bootstrap

2.3. Refresh

The CA automatically re-issues the certificate (using the same CSR) before it expires and publishes it to the URL that the NDC has come to know at the end of the bootstrap phase. The NDC downloads and installs it. This process goes on until either:

- o IdO terminates the delegation, or
- o Automatic renewal expires.

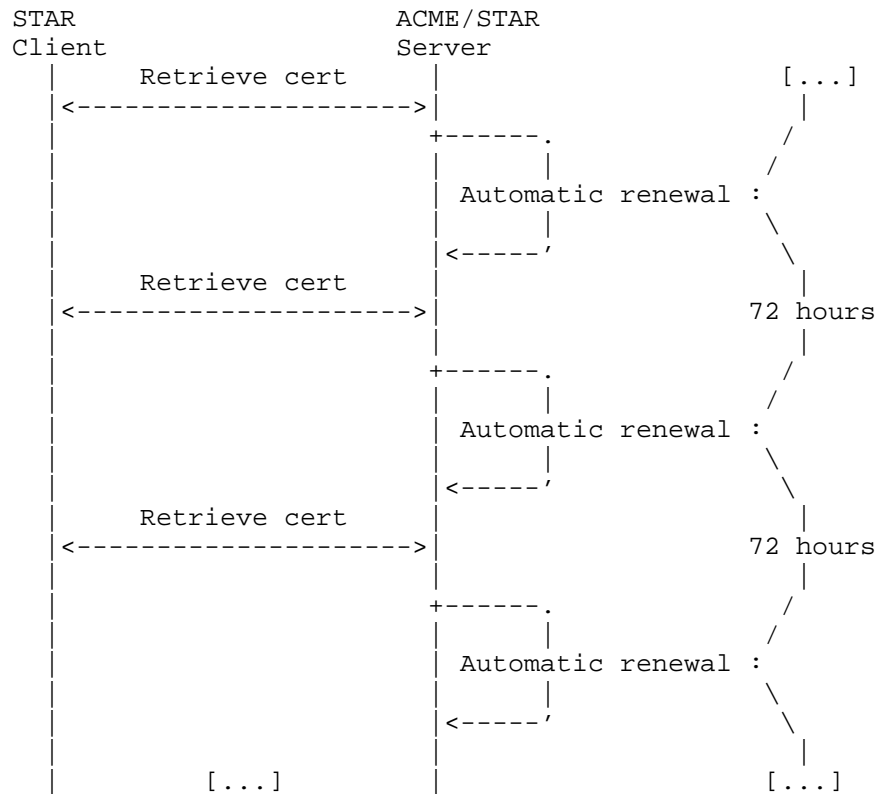


Figure 2: Auto renewal

2.4. Termination

The IdO may request early termination of the STAR certificate by including the Order ID in a certificate termination request to the ACME interface, defined below. After the CA receives and verifies the request, it shall:

- o Cancel the automatic renewal process for the STAR certificate;
- o Change the certificate publication resource to return an error indicating the termination of the delegation to external clients, including the NDC.

Note that it is not necessary to explicitly revoke the short-term certificate.

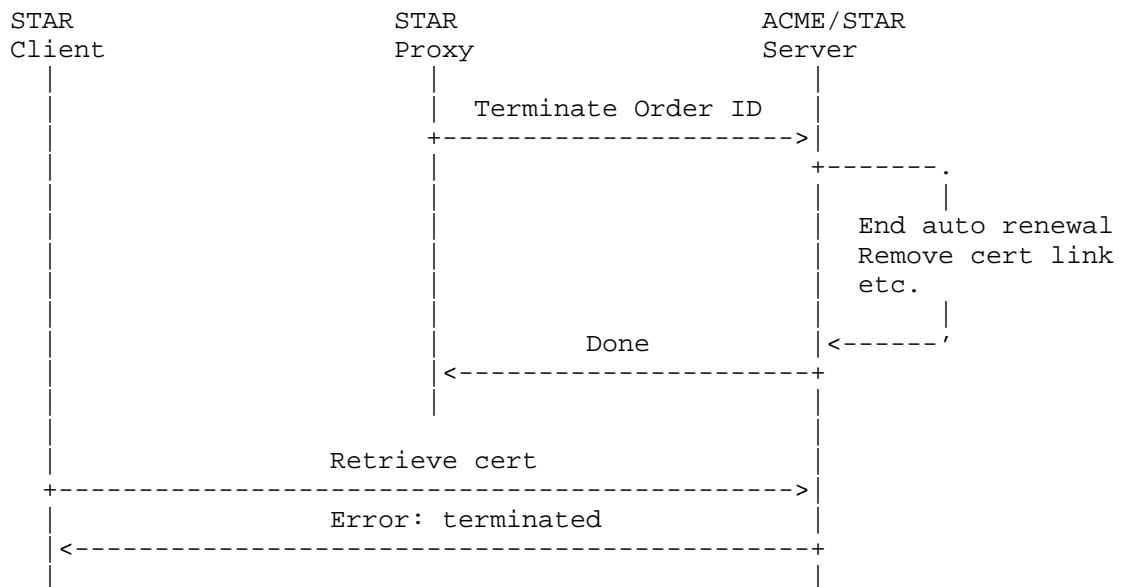


Figure 3: Termination

No facility is provided for the NDC to directly initiate the termination of a STAR certificate.

3. Protocol Details

This section describes the STAR API between the STAR Client and the STAR Proxy.

3.1. STAR API

This API allows an IdO (STAR Proxy) to control the long-term delegation of one of its names to an authorized third-party (STAR Client).

3.1.1. Creating a Delegation Request

To create a new delegation request, the client wraps the following parameters in a POST to the `'/star/delegation'` path:

- o `csr` (required, string): A CSR encoding the parameters for the certificate being requested [RFC2986]. The CSR is sent in the base64url-encoded version of the DER format. (Note: Because this field uses base64url, and does not include headers, it is different from PEM.)

- o duration (optional, integer): How long the delegation should last (in seconds). If not specified, a local default applies.
- o certificate-lifetime (optional, integer): How long each short-term certificate should last (in seconds). If not specified, a local default applies.

Note that the STAR Proxy MAY treat both "duration" and "certificate-lifetime" as hints, and MAY update any of them due to local policy decisions or as a result of the interaction with the ACME server.

```
POST /star/delegation
Host: star-proxy.example.net
Content-Type: application/json
```

```
{
  "csr": "jcRf4uXra7FGYW5ZMewvV...rhlnznwy8YbpMGqwidEXfE",
  "duration": 31536000,
  "certificate-lifetime": 604800
}
```

On success, the service returns a 201 Created status with the URL of the newly generated delegation order in the Location header field. The current state of the delegation order is returned in the body of the response in JSON format:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: http://example.net/star/delegation/567
```

```
{
  "id": "567",
  "certificate-lifetime": 604800,
  "duration": 31536000,
  "status": "new"
}
```

If an error occurs, an error response (4XX or 5XX) is generated with an appropriate problem detail [RFC7807] body, e.g.:

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

{
  "type": "https://example.net/validation-error",
  "title": "Your request parameters didn't validate.",
  "invalid-params": [ {
    "name": "csr",
    "reason": "missing mandatory parameter"
  } ]
}
```

3.1.2. Polling the Delegation Request

The returned delegation order URL can be polled until the dialog between the STAR Proxy and the ACME server is complete (i.e., the "status" attribute changes from "new" or "pending" to one of "failed" or "success"):

```
GET /star/delegation/567
Host: star-proxy.example.net
```

In responding to poll requests while the validation is still in progress, the server MUST return a 200 (OK) response and MAY include a Retry-After header field to suggest a polling interval to the client. The Retry-After value MUST be expressed in seconds. If the Retry-After header is present, in order to avoid surprising interactions with heuristic expiration times, a max-age Cache-Control SHOULD also be present and set to a value slightly smaller than the Retry-After value:

```
HTTP/1.1 200 OK
Content-Type: application/json
Retry-After: 10
Cache-Control: max-age=9

{
  "id": "5",
  "certificate-lifetime": 604800,
  "creation-date": "2017-11-12T01:38:09Z",
  "duration": 31536000,
  "status": "pending"
}
```

When the operation is successfully completed, the ACME Proxy returns:

HTTP/1.1 200 OK

```
{
  "status": "success", // or "failed"
  "lifetime": 365,      // lifetime of the registration in days,
                        // possibly less than requested
  "certificates": "https://ca.example.org/certificates/A51A3"
}
```

The "certificates" attribute contains a URL of the certificate pull endpoint, received from the ACME Server.

If the registration fails for any reason, the server returns a "200 OK" response, with the status as "failed" and a "reason" attribute containing a human readable error message.

3.2. Transport Security for the STAR Protocol

Traffic between the STAR Client and the STAR Proxy MUST be protected with HTTPS. For interoperability, all implementations MUST support HTTP Basic Authentication [RFC7617]. However some deployments MAY prefer mutually-authenticated HTTPS or two-legged OAUTH.

4. CDNI Use Cases

Members of the IETF CDNI (Content Delivery Network Interconnection) working group are interested in delegating authority over web content to CDNs. Their requirements are described in a draft [I-D.fieau-cdni-https-delegation] that compares several solutions. This section discusses two particular requirements in the context of the STAR protocol.

4.1. Multiple Parallel Delegates

In some cases the DNO would like to delegate authority over a web site to multiple CDNs. This could happen if the DNO has agreements in place with different regional CDNs for different geographical regions. STAR enables this use case naturally, since each CDN can authenticate separately to the DNO specifying its CSR, and the DNO is free to allow or deny each certificate request according to its own policy.

4.2. Chained Delegation

In other cases, a content owner (DNO) delegates some domains to a large CDN (CDN1), which in turn delegates to a smaller regional CDN, CDN2. The DNO has a contractual relationship with CDN1, and CDN1 has

a similar relationship with CDN2. However DNO may not even know about CDN2.

The STAR protocol does not prevent this use case, although there is no special support for it. CDN1 can forward requests from CDN2 to DNO, and forward responses back to CDN2. Whether such proxying is allowed is governed by policy and contracts between the parties.

5. Security Considerations

5.1. STAR Protocol Authentication

The STAR protocol allows its client to obtain certificates bearing the IdO's identity. Therefore strong client authentication is mandatory.

When multiple NDCs may connect to the same IdO, the STAR protocol's authentication MUST allow the IdO to distinguish between different NDCs, and the IdO MUST associate different Registration objects to different clients. Among other benefits, this allows the IdO to cancel a STAR registration for one of its clients instead of all of them.

6. Acknowledgments

This work is partially supported by the European Commission under Horizon 2020 grant agreement no. 688421 Measurement and Architecture for a Middleboxed Internet (MAMI). This support does not imply endorsement.

7. References

7.1. Normative References

[I-D.ietf-acme-acme]

Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", draft-ietf-acme-acme-12 (work in progress), April 2018.

[I-D.ietf-acme-star]

Sheffer, Y., Lopez, D., Dios, O., Pastor, A., and T. Fossati, "Support for Short-Term, Automatically-Renewed (STAR) Certificates in Automated Certificate Management Environment (ACME)", draft-ietf-acme-star-03 (work in progress), March 2018.

- [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>>.
- [RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <<https://www.rfc-editor.org/info/rfc2986>>.
- [RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <<https://www.rfc-editor.org/info/rfc7617>>.
- [RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <<https://www.rfc-editor.org/info/rfc7807>>.

7.2. Informative References

- [I-D.fieau-cdni-https-delegation]
Fieau, F., Emile, S., and S. Mishra, "HTTPS delegation in CDNI", draft-fieau-cdni-https-delegation-02 (work in progress), July 2017.

Appendix A. Document History

[[Note to RFC Editor: please remove before publication.]]

A.1. draft-sheffer-acme-star-request-02

- o Clarifications and minor changes based on implementation experience.
- o More detail on error cases.

A.2. draft-sheffer-acme-star-request-01

- o Correct reference to WG draft.

A.3. draft-sheffer-acme-star-request-00

- o Initial version, the STAR API extracted from draft-sheffer-acme-star-02.

Authors' Addresses

Yaron Sheffer
Intuit

EMail: yaronf.ietf@gmail.com

Diego Lopez
Telefonica I+D

EMail: diego.r.lopez@telefonica.com

Oscar Gonzalez de Dios
Telefonica I+D

EMail: oscar.gonzalezdedios@telefonica.com

Antonio Agustin Pastor Perales
Telefonica I+D

EMail: antonio.pastorperales@telefonica.com

Thomas Fossati
Nokia

EMail: thomas.fossati@nokia.com