Workgroup: Interpeer Project
Published: 14 June 2023
Author: J. Finkhaeuser
Interpeer

CAProck Distributed Authorization Scheme

Abstract

CAProck is a distributed authorization scheme based on cryptographic capabilities [I-D.draft-jfinkhaeuser-caps-for-distributed-auth]. This document describes the schemes additional constraints over the base document, and introduces a method for dealing with revocation of authorization. The result is a complete distributed authorization scheme.

The RFC Editor will remove this note

About This Document

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

The latest revision of this draft can be found at https://specs.interpeer.org/draft-jfinkhaeuser-caprock-auth-scheme/.

Discussion of this document takes place on the Interpeer mailing list interpeer@lists.interpeer.io, which is archived at https://lists.interpeer.io/pipermail/interpeer/. Subscribe at https://lists.interpeer.io/mailman/listinfo/interpeer.

Source for this draft and an issue tracker can be found at https://codeberg.org/interpeer/specs.

Status of This Memo

Drafts are working documents of the Interpeer Project. The list of current Drafts is at https://specs.interpeer.org/. Drafts may be updated, replaced, or obsoleted by other documents at any time. It is inadvisable to use Drafts as reference material or to cite them other than as "work in progress."

Copyright Notice

Table of Contents

1. Introduction

CAProck addresses distributed authorization by providing a framework for applications to define and verify their own privilege schemes via cryptographic capabilities.

Capabilities in essence provide cryptographic verification for an authorization tuple, and thus confirm a relationship between a subject, a prilege, and an optional object. The privilege itself is an opaque piece of information for this scheme; it only has application defined meaning.

As such, CAProck can be viewed as a kind of envelope for distributed authorization schemes. To do this, however, this document needs to define some constraints for identifiers used in this scheme.

Not in scope of this document are wire encodings (serialization formats) for capabilities. This is because different applications may have differing requirements, making one kind of encoding more suitable than the other.

2. Conventions and Definitions

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.

In order to respect inclusive language guidelines from [NIST.IR.8366] and [I-D.draft-knodel-terminology-10], this document uses plural pronouns.

2.1. Terminology

Most terminology is taken from [I-D.draft-jfinkhaeuser-caps-for-distributed-auth], Section 2, in which elements of a cryptographic, distributed authorization scheme are listed. Below are alternative names for some of those elements as used in CAProck.

Claim: In CAProck, authorization tuples are called a *claim*, because by presenting them, a claim is made about the relationship of its elements. The tuple consists of a subject identifier, a predicate, and an object identifier.

Predicate: Where the overview draft above speaks about *privileges*, CAProck uses the more general *predicate* term. This is terminology from [RDF] to indicate that a predicate can itself be arbitrarily complex, though for authorization purposes must describe some permission relationship.

Issuer: The grantee role described in the more general document is called the issuer in CAProck, for consistency with terminology from [X.509].

Furthermore, CAProck defines the following terms:

Token: A token is a serialized capability, as passed over the network. Specific serialization formats are out of scope for this document, but two tokens serialized in different formats but with identical contents **MUST** be considered equivalent.

Grant: A grant is a token that makes claims about granting some permissions.

Revocation: Conversely, a revocation makes claims about revoking some permissions.

2.2. Use Case

CAProck was designed with a 0-RTT handshake in mind that establishes authorization for some resource. This scenario is adequately described in [I-D.draft-jfinkhaeuser-caps-for-distributed-auth].

Suffice to add that a 0-RTT handshake over IP requires that authentication information as well as a caprock token be transmitted in a single packet. That in turn requires the data to be transmitted, and the encoding used on the wire, to be as sparse as feasible.

Where possible, CAProck uses compact data representations for this reason. However, as wire encodings are not in scope for this document, this is mostly discussed in the abstract.

Furthermore, CAProck is designed with a fully distributed system in mind. The basic assumption is that an issuer of a capability cannot be used at the time the capability is to be verified.

3. CAProck

The basic functionality of CAProck is to provide a cryptographic signature over a number of claims with the intent that it can be verified at a later date, when actions based on those claims are to be performed.

This temporal decouping between the time of signature and time of verification represents both the greatest strength and weakness of capability based schemes, as it raises the question for what time period these claims are to be considered valid. Rare is the case where a privilege should be granted to a subject for perpetuity, but choose the time period too small, and a claim may not be usable.

Furthermore, this temporal decoupling introduces the issue of what should happen when a privilege is granted and the associated capability transmitted, but subsequently trust in the subject is lost. This loss of trust should ideally be known to the party that is supposed to execute an action on behalf of the subject.

The typical approach is to verify via some real-time blocklist query. However, the design goals for CAProck prohibit such an approach. Note, however, that systems using CAProck may still employ this method. The requirement is included to drive the design to broader applicability also in situations where real-time queries are infeasible.

To these ends, CAProck defines additional metadata beyond that described in [I-D.draft-jfinkhaeuser-caps-for-distributed-auth], namely:

- A validity timespan (scope)
- An expiry policy
- Grant as well as revocation markers

Furthermore, this document defines a conflict resolution scheme for resolving multiple capabilities.

3.1. Identifiers

Generally speaking, a capability based distributed authorization scheme is agnostic to the specific contents of identifiers used for the capability issuer, subject and object. However, such a scheme requires that issuer and subject identifiers can be uniquely mapped to public keys.

The issuer public key is required to verify the capability signature. At the same time, accepting and executing an action requires that the capability subject is authenticated, which typically involves use of its public key as well.

The space saving concerns suggest that full public keys should only be used in authentication, while shorter identifiers derived from these keys are sufficient for use in capabilities. To that end, CAProck defines two kinds of identifiers for issuers and subjects:

SHA-3 Identifiers: Identifiers may be SHA-3 hashes ([NIST.FIPS.202]) over DER encoded ([X.690]) public keys.

Raw Identifiers: If the key scheme supports it, identifiers may also be raw public keys. This identifier **MUST NOT** be chosen if it is longer than the longest identifier generated under the SHA-3 scheme for the same key, but **SHOULD** be used if the result is shorter than the SHA-3 scheme produces.

Algorithm	Identifier	Preferred Scheme	Reference
PureEdDSA for curve25519	ed 2 5 5 1 9	raw	[RFC8410], [RFC8032]
PureEdDSA for curve448	ed 4 4 8	raw	[RFC8410], [RFC8032]
DSA	dsa	SHA-3	[NIST.FIPS.186-4]
RSA	rsa	SHA-3	[NIST.FIPS.186-4]
ECDSA	ecdsa	SHA-3	[NIST.FIPS.186-4]

Table 1: Identifier Schemes

3.1.1. Object Identifiers

Unlike with issuer and subject identifiers, object identifiers do not require mapping to public keys, though it may be in the application's interest to also define object identifiers in this fashion.

CAProck places only a single constraint on object identifiers: that they have the same length as issuer and subject identifiers. More specifically, object identifers **MUST** be between 28 and 64 octets in length, and **SHOULD** be one of the sizes yielded by either of the issuer and subject identifier schemes.

In practice, this constraint suggests to use a default SHA-3 digest size for identifiers of any kind, unless the use of EdDSA allows shorter issuer or subject identifiers.

3.1.2. Group Identifiers

As CAProck does not define much about identifiers, it is also agnostic as to whether any identifier refers to an individual or group. Such definitions are part of the application concern.

Note, however, that as subject and issuer identifiers must be mappable to a public key, the use of groups with a CAProck based scheme essentially becomes a public key management problem.

3.2. Predicates

Predicate formats are outside of the scope of this document. This is to permit maximum flexibility for the application concern.

However, arbitrary length predicates are in conflict with the space saving requirements, while short predicates may limit the application in how to apply them.

CAProck takes a compromise approach here: predicates **MUST NOT** be larger than 2^16 = 65,536 octets. This limit is far too generous for 0-RTT handshakes, so predicates **SHOULD** be limited to significantly shorter lengths, such that an entire capability fits comfortably into an IP packet with authentication information.

3.3. Claims

CAProck's capabilities require that one or more claims are provided to create them, i.e. tuples of subject identifier and predicate, and an optional object identifier.

Note that a claim that includes an object identifier has a signficantly different semantic from one that does not.

1. Authorization tuples with an object identifier assert that the predicate describes the relationship between the given subject and object.
2. Authorization tuples without an object assert that the predicate describes an aspect of the subject itself.

3.3.1. Wildcards

Furthermore, in CAProck each claim component may be a wildcard. As predicate formats are outside of this document's scope, predicates may also *contain* a wildcard, but the semantics of that must be defined with the predicate scheme to use.

Wildcards make more general statements than a claim typically does, and due to this genericity, implementations **MUST** take great care with their use. Broadly speaking:

1. A wildcard subject states that the predicate applies to an object for any subject. Such a statement may imply that authenticating a subject is no longer of interest, i.e. grant public privileges.
2. A wildcard predicate states that all relationships between the given subject and object are affected. This may e.g. be used to revoke all privileges from a subject.
3. A wildcard object is not the same as omitting an object altogether; instead, it states that the subject and predicate combination applies to all objects. Note, though, that the issuer does not have authority over all possible objects, so this is naturally limited to only those objects the issuer has authority over.

Combinations of wildcard fields in a single claim may have sense within the application's privilege scheme, but can also be increasingly confusing, and thus prone to creating security flaws. For that reason, public privilege schemes **MUST** document the semantics for every combination of wildcard elements they support, and **SHOULD** limit such combinations to a minimum required amount. Other combinations of wildcards **SHOULD** be rejected as invalid.

3.4. Metadata

The most basic data encoded into a CAProck token is mostly described in [I-D.draft-jfinkhaeuser-caps-for-distributed-auth] and consists of the following fields:

1. An issuer identifier (also see Section 3.1).
2. One or more claims (Section 3.3), each consisting of:
    1. A subject identifier (Section 3.1).

2. A predicate (Section 3.2).
3. An optional object identifier (Section 3.1.1).

3. A validity range (Section 3.4.1).
4. An expiry policy (Section 3.4.2).
5. A signature by the issuer over a serialized version of the previous fields (Section 3.4.3).

Please note that the specific encoding or serialization format for the data affected by the signature is not defined in this document; this is because alternative encodings are possible. For this reason, documents specifying such encodings **MUST** be clear on how this payload data is serialized, and how the signature is added to the token.

3.4.1. Validity Range/Temporal Scope

The validty range or temporal scope for a token is defined by a tuple of timestamps, much as in [X.509] certificates. In these certificates, the validity range is defined by rather awkwardly named "not before" and "not after" fields, CAProck prefers the "from" and "to" fields.

The "from" timestamp **MUST** be provided, while the "to" timestamp is optional. Tokens without the "to" timestamp are valid in perpetuity, after the "from" time has been reached.

The range from "from" to "to" is *inclusive*, i.e. the token is valid not before the "from" date, but both at and after the "from" timestamp. The same logic applies to the "to" timestamp.

The wire encoding specifies how to represent these fields in the token. However, implementations **MUST** accept the subset of ISO-8601 time stamp formats as defined in [RFC3339] as input to either field, in order to set consistent expectations with the user base.

3.4.2. Expiry Policy

Distributed authorization in general and CAProck in particular are designed for scenarios in which connectivity is not always given. This can mean that having a access to a synchronized clock is impossible at the time of validation, which would imply that evaluating the valdity range timestamps above is impossible.

To mitigate this, tokens **MAY** contain an optional expiry policy field. This field can take the symbolic values of `issuer` and `local`. If omitted, the value **MUST** be assumed to be `issuer`. The semantics of each value are as follows:

Issuer: If the expiry policy is set to `issuer`, then the valdity scope of the issuer **MUST** be respected. An unsynchronize clock may lead to failures, but that is the issuer's wish.

Local: The verifier is permitted to apply local policies for failures. That is, a CAProck system must now query the application whether to accept or reject a token.

Leaving the possibility to defer to the applications permits resolving clock conflicts by means outside the ability of CAProck to influence. One such method might be to simply ask the application user.

As this behaviour is a potential security risk, implementations **MAY** reject tokens with `local` expiry policy outright.

3.4.3. Signature

Asymmetric signatures are made over hashes of content; the hash algorithm to use depends on the key type used for signing.

- The EdDSA algorithm defines what signature hash function to use in [RFC8032].
- The DSA algorithm requires the selection of a hashing algorithm. Implementations **MUST** use one of the SHA-2 or SHA-3 family of hash functions (see [NIST.FIPS.180-4]).
- The ECDSA algorithm likewise requires the selection of a hashing algorithm. We also use SHA-2 or SHA-3 here. For ECDSA, [NIST.FIPS.180-4] specifies that the hash length shall be no less than the ECDSA key bit length.
- The RSA algorithm also requires specification of a signature hash. Implementations **MUST** select one of the SHA-3 family of hash functions (see [NIST.FIPS.202]).

CAProck does not specify which digest sizes to use for SHA-2/SHA-3. However, wire encodings may restrict the choices here further.

3.5. Grants and Revocations

Capabilities in CAProck either grant privileges or revoke them; these are called grants or grant tokens and revocations or revocation tokens, respectively. As each token contains one or more claims, the implication is that each token either grants all of the claimed privilege or revokes all.

Other metadata (see Section 3.4) applies equally to the entire set of claims. But individual tokens may contain differing metadata. If the metadata differs, this can create a set a tokens that make conflicting claims.

Consider the following scenario:

1. Grant G_1 contains claims C_1 and C_2 for a period of $[S_1, E_1]$ defined by a start S_1 and end E_1 timestamp.
2. Revocation R_1 revokes claim C_1 for a new, shorter time period $[S_2, E_2]$ where $S_2 > S_1$ and $E_2 < E_1$.

In this simple scenario, whether C_1 is granted or revoked already depends on whether one looks at time period $[S_1, S_2)$, $[S_1, E_2]$ or $(E_2, E_1]$ - or some period before S_1 or after E_1.

If the two tokens are examined in the order listed, that is all there is to it. However, what should happen if the tokens arrive in reverse order? For this, we need to define a conflict resolution algorithm (Section 3.5.1).

Things only gain in complexity when more claims are processed, and revocations are themselves further "revoked" by issuing another grant for a different time period.

3.5.1. Conflict Resolution

While privileges encoded in predicates are arbitrarily complex, whether a token represents a grant or revocation is essentially a binary value. It is therefore possible to view a set of grants and revocatons for a given claim as a bit stream, in which only the latest bit has any significance. The question then is how to bring order into this token set.

To this end, CAProck introduces a simple counter. This is represented by an unsigned 64 bit integer value. The only requirement on the counter is that tokens issued earlier have smaller values than tokens issued at a later date. The counter value is scoped to the issuer that issues the token; therefore, no synchronization of counters between issuers is required.

In order to query the validity of a claim, then, a time point must be provided, and the following algorithm **MUST** be used:

1. Start with the state that the claim is invalid.
2. Assume a token store containing only tokens with valid cryptographic signatures.
3. From the token store, pick all tokens pertaining to the claim being queried. Note that this may include claims containing wildcards.
4. Order the picked tokens by the counter value from lowest value to highest value, essentially ordering by age of the token.
5. Process each token in order, and compare it's validity range to the given time point:
    1. Discard tokens for which the time point does not lie in the validity range.
        1. If the token expiry policy is `local`, consult the local policy whether to discard it. See section Section 3.4.2 for details.
    2. If the time point lies in the validity range, set the current state to valid for grants, and invalid for revocations.
6. When all tokens are processed, the current state is the final state of processing. If that state is invalid, reject the claim.
7. For valid end states, process the claim by application specified rules for the predicate. The information contained here may still make the claim invalid, but this is beyond the scope of CAProck.

Note that CAProck explicitly does not introduce timestamps for ordering tokens. This is due to several considerations.

First, using a counter permits an implementation to update the value more sparsely. Between issuing two tokens, a lot of time may pass, which would increment a timestamp by a significant number. By contrast, a counter may use the value space more efficiently by being incremented only by one each time.

Second, the extra information a timestamp imparts is not necessary for deconflicting tokens. However, it is possible to infer information from it that may best be hidden, namely when the token was issued. This permits for analysis on the time periods an issuer was active, and may imply further communications between the issuer and subject.

Finally, using timestamps introduces a hard dependency on a valid understanding of the current time for validation. In early boot processes or handshakes for reading a clock, such a dependency cannot necessarily be fulfilled.

3.5.1.1. State Compression

The above algorithm makes it possible to compress state. First, any expired tokens can be discarded, as they are no longer consulted at all. They have the same semantics as a revocation for the claim.

Furthermore, subsequent tokens that completely overrule a prior token make it unnecessary to keep storing the prior token.

Implementations are not required to perform any state compression, and **MAY** find additional means to compress state. However, such compressions **MUST NOT** affect the results as would be achieved by the above algorithm.

3.6. Confidentiality

All of the data fields described here are required for a party validating a capability or a set thereof; therefore, CAProck cannot provide confidentiality for this data itself.

However, individual permissions may well be in need of protection. If any subject is issued a privilege, this should not be visibly to any party other than the issuer, the verifier, and most likely the subject itself.

Implementations **SHOULD** therefore transport capability tokens in such a way that confidentiality is preserved. This leaves room for potentially replacing the cryptographic signature with an authenticated encryption method. Such a scheme, however, should be considered an extension to CAProck and defined in a separate document.

3.7. Delegation

Delegation of authority comes in two distinct flavors in a distributed authorization system. CAProck addreses the set of scenarios in which a subject wishes to perform an action (on an object), and presents its authorization to do so.

The other set of scenarios involves delegating the authority to grant priveleges. This is not explicitly addressed in CAProck, though the following may provide some guidance.

First, a predicate that states a subject may itself create subgrants is likely more complex than predicates permitting other actions. At minimum, such a predicate should provide limits on which subgrants may be issued; such limits may include a list of permitted predicates, different

time slots for the subgranted privilege, and a limit on whether grants or revocations or both may be issued. Effectively, such a predicate might contain the same information as a CAProck token, or even more.

Second, a subgrant may also be issued to grant further subgrants. If this is the case, there is likely need for bounding such subsubgrants to a certain depth, otherwise it is likely that subjects receive grants that the original issuer never intended to issue grants to.

4. Related Considerations

4.1. Human Rights Considerations

[I-D.draft-jfinkhaeuser-caps-for-distributed-auth] contains a list of objectives derived from [RFC8280], each with a statement on how these concerns are addressed. This section lists any modifications or additions to that list that is specific to CAProck.

4.1.1. In Scope

Content agnosticism: Where the base document is entirely content agnostic, CAProck restricts this to content agnosticism about predicates. Without such a restriction, CAProck would not be able to add signficant semantics to the base document.

Localization: This document refers to time in [RFC3339] timestamps in order to provide timestamps unambiguous in any locale.

4.1.2. Out of Scope

Confidentiality: Confidentiality remains out of scope, but see Section 3.6 for some additional considerations.

4.2. Protocol Considerations

There are no additional protocol considerations for this document.

4.3. Security Considerations

This document does not specify a network protocol. In fact, it deliberately requires no specific protocol for transmitting capabilities. As such, much of [BCP72] does not apply.

Similar to Section 4.1, below we list changes to the base document at [I-D.draft-jfinkhaeuser-caps-for-distributed-auth].

4.3.1. Confidentiality

As above, confidentiality is out of scope, but see Section 3.6.

4.3.2. Message Deletion

Deletion of individual tokens in a ordered stream relating to the same claim may result in the wrong verification state at the end of the algorithm presented in Section 3.5.1. Mitigation against this boils down to knowing that the full set of tokens has been communicated that an issuer generated for a given claim.

It is possible to use CAProck in this fashion, but this document assumes other uses are equally valid. To provide mitigation against message deletion, follow the steps below:

1. Start issuer counters at some fixed value, e.g. 0, and strictly increment the counter by one for each token issued.
2. Only provide exactly one claim per token.
3. Communicate the last counter issued as part of the token transmission. Validate this by means out of scope for this document.

Given the above information, a recipient of a token set can be certain that they know the full set of tokens, *at the point in time when the token set was sent*. If the transmission of the token set is synchronous, this corresponds to the point in time at which a claim is checked for validity.

However, these additional assumptions counter some of the gains for distributed systems, to the point where a traditional, centralized authorization scheme may be the better choice. It therefore depends strongly on the threat model and application requirements whether to use centralized authorization, distributed authorization, or a mixed model as outlined above.

4.4. IANA Considerations

This document has no IANA actions.

5. References

5.1. Normative References

[BCP72] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/rfc/rfc3552>.

[I-D.draft-jfinkhaeuser-caps-for-distributed-auth] Finkhäuser, J. and S. P. ISEP, "Capabilities for Distributed Authorization", Work in Progress, Internet-Draft, draft-jfinkhaeuser-caps-for-distributed-auth-01, 1 June 2023, <https://datatracker.ietf.org/doc/html/draft-jfinkhaeuser-caps-for-distributed-auth-01>.

[NIST.FIPS.180-4] "Secure hash standard", National Institute of Standards and Technology (U.S.), DOI 10.6028/nist.fips.180-4, 2015, <https://doi.org/10.6028/nist.fips.180-4>.

[NIST.FIPS.186-4] "Digital signature standard (DSS)", National Institute of Standards and Technology (U.S.), DOI 10.6028/nist.fips.186-4, 2013, <https://doi.org/10.6028/nist.fips.186-4>.

[NIST.FIPS.202] "SHA-3 standard :: permutation-based hash and extendable-output functions", National Institute of Standards and Technology (U.S.), DOI 10.6028/nist.fips.202, 2015, <https://doi.org/10.6028/nist.fips.202>.

[NIST.IR.8366] Miller, K., Alderman, D., Carnahan, L., Chen, L., Foti, J., Goldstein, B., Hogan, M., Marshall, J., Reczek, K., Rioux, N., Theofanos, M., and D. Wollman, "Guidance for NIST staff on using inclusive language in documentary standards", National Institute of Standards and Technology (U.S.), DOI 10.6028/nist.ir.8366, April 2021, <https://doi.org/10.6028/nist.ir.8366>.

[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/rfc/rfc2119>.

[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>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[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/rfc/rfc8174>.

[RFC8280] ten Oever, N. and C. Cath, "Research into Human Rights Protocol Considerations", RFC 8280, DOI 10.17487/RFC8280, October 2017, <https://www.rfc-editor.org/rfc/rfc8280>.

[RFC8410] Josefsson, S. and J. Schaad, "Algorithm Identifiers for Ed25519, Ed448, X25519, and X448 for Use in the Internet X.509 Public Key Infrastructure", RFC 8410, DOI 10.17487/RFC8410, August 2018, <https://www.rfc-editor.org/rfc/rfc8410>.

[X.690] International Telecommunications 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.

5.2. Informative References

[I-D.draft-knodel-terminology-10] Knodel, M. and N. ten Oever, "Terminology, Power, and Inclusive Language in Internet-Drafts and RFCs", Work in Progress, Internet-Draft, draft-knodel-terminology-10, 11 July 2022, <https://datatracker.ietf.org/doc/html/draft-knodel-terminology-10>.

[ISOC-FOUNDATION] Internet Society Foundation, "Internet Society Foundation", n.d., <https://www.isocfoundation.org/>.

[RDF] RDF Working Group of the World Wide Web Consortium (W3C), "RDF 1.1 Concepts and Abstract Syntax", 25 February 2014, <https://www.w3.org/TR/rdf11-concepts/>.

[X.509] International Telecommunications Union, "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks", ITU-T Recommendation X.509, ISO Standard 9594-8, March 2000.

Acknowledgments

Jens Finkhäuser's authorship of this document was performed as part of work undertaken under a grant agreement with the Internet Society Foundation [ISOC-FOUNDATION].

Index

Author's Address

Jens Finkhäuser
Interpeer gUG (haftungsbeschraenkt)
Email: ietf@interpeer.io
URI: https://interpeer.io/