Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 26 October 2026

M. F. Callec
Independent

24 April 2026

Domain-based Integrity Verification Enforcement (DIVE) Version 0.1
draft-callec-dive-01

Abstract

Domain-based Integrity Verification Enforcement (DIVE) is an application-layer protocol that provides cryptographic integrity and authenticity verification of HTTP response bodies by leveraging the Domain Name System Security Extensions (DNSSEC) as an out-of-band distribution channel for public keys.

DIVE has two independent components: (1) an object-security layer, which uses HTTP Message Signatures (RFC 9421) to carry per-resource signatures in HTTP response headers, and (2) a DNS key-distribution layer, which publishes public keys and policy in DNSSEC-protected TXT records. A client implementing DIVE verifies each covered resource against the corresponding DNS-published public key before accepting it. An attacker must therefore compromise both the DNS infrastructure and the origin server simultaneously to deliver a tampered resource to a DIVE-compliant client.

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 26 October 2026.

Copyright Notice

Table of Contents

1. Introduction

Transport-layer security protects data in transit but does not protect against a compromised origin server that serves malicious content over a legitimate TLS session. Subresource Integrity (SRI) embeds expected hashes in HTML markup, but because that markup is itself served by the potentially compromised host, it provides limited security during a full infrastructure breach.

DIVE addresses this threat by separating two concerns:

* *Object security*: HTTP Message Signatures [RFC9421] carry a cryptographic signature over the response body. The signature travels with the resource and can be verified at any point after receipt.

* *Key distribution*: DNSSEC-protected DNS TXT records publish the authoritative public keys and policy. Because DNS is administered independently from the origin server, an attacker who controls only the origin cannot forge a valid DNS-published key.

DIVE is designed for non-browser automated clients such as package managers, CLI tools, and software-update agents. Browser clients MUST NOT implement or enforce DIVE.

DIVE is incrementally deployable: servers add DNS records and HTTP signatures; clients that do not implement DIVE are unaffected.

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.

DIVE client: An HTTP client that implements the verification algorithm defined in this document.

DIVE server: An HTTP origin server that publishes DIVE DNS records and includes DIVE-conformant HTTP Message Signatures on covered resources.

Resource: A single HTTP response body received with status code 200.

Scope: A named category of resources subject to DIVE verification (Section 4).

Key ID: An operator-assigned label that names a specific DNS key record and is referenced in HTTP Message Signatures via the keyid parameter ([RFC9421] Section 2.3).

Policy record: The _dive DNS TXT record that publishes the DIVE configuration for a domain (Section 5.3).

Key record: A DNS TXT record that publishes a public key, placed at <Key-ID>._divekey.<domain> (Section 5.4).

Structured Field: A value encoded per [RFC9651].

Unix timestamp: Seconds since 1970-01-01T00:00:00Z (UTC), excluding leap seconds, as a signed integer.

3. Architecture

DIVE separates object security from key distribution:

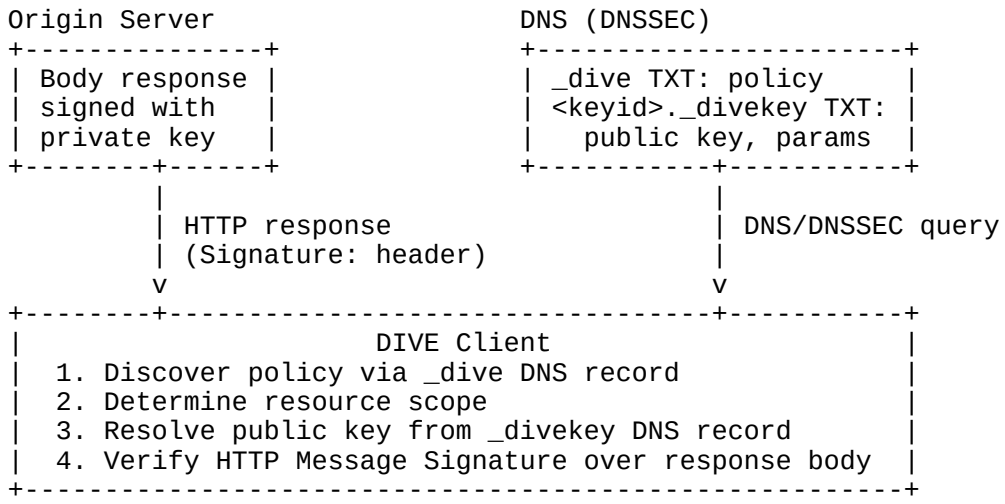

Figure 1: DIVE Architecture and Verification Flow

The Signature and Signature-Input headers are defined by [RFC9421]. The DNS records are defined by this document.

4. Scopes

A scope identifies the category of resources to which DIVE verification applies.

4.1. Standard Scopes

strict: DIVE verification MUST be applied to ALL resources served under the domain covered by the policy record.

4.2. Custom Scopes

Operators MAY define custom scopes for application-specific resource categories. Custom scope names MUST begin with x-, be entirely lowercase, and contain only a-z and - after the prefix.

Custom scopes are intended for closed environments where server and client are under the same operator’s control. Detection logic for custom scopes is application-defined.

A DIVE client that does not recognise a custom scope MUST ignore it.

5. DNS Configuration

5.1. DNSSEC Requirement

A DIVE server SHOULD enable DNSSEC [RFC4033] for all zones publishing DIVE records. If a client retrieves a _dive record without DNSSEC validation, it MUST treat DIVE as not supported for that domain.

5.2. Record Format

All DIVE DNS records are DNS TXT records. Values MUST be formatted as Structured Field Values [RFC9651]. Parameter names and values MUST be lowercase unless otherwise stated. Timestamps are Unix timestamps represented as Structured Field Integers.

5.3. Policy Record (_dive)

The _dive TXT record is placed at the _dive label of the domain or subdomain it governs (e.g., _dive.example.com). A record at a given level applies to all subordinate labels unless overridden by a more specific record.

5.3.1. Example

```
_dive.example.com.  300  IN  TXT  (
  "v=\"dive-draft-01\", "
  "scopes=(\"strict\"), "
  "directives=(\"https-required\"), "
  "cache=300, "
  "invalidate-keys-cache=1700000000, "
  "report-to=\"https://reports.example.com/dive\""
)
```

Figure 2: Example of a DIVE Policy Record

5.3.2. Parameters

v (REQUIRED): Protocol version string. MUST be "dive-draft-01". If absent, unrecognised, or unparseable, the client MUST treat DIVE as not supported for this domain.

scopes (OPTIONAL): Structured Field List of scope names (Strings). If absent or empty, no resource-level verification is performed, though other directives still apply.

directives (OPTIONAL): Structured Field List of behavioural directives (Strings):

* "https-required": the client MUST refuse or upgrade plain-HTTP requests for resources under the covered domain.

* "report-only": the client MUST NOT block resources that fail DIVE verification; it MUST report failures per Section 7.6 instead.

Unrecognised directive values MUST be ignored.

cache (OPTIONAL): Structured Field Integer. Number of seconds the client MAY cache this record (default: 0). MUST NOT exceed 3600 (Section 7.7).

invalidate-keys-cache (OPTIONAL): Structured Field Integer (Unix timestamp). When present, the client MUST purge cached key records for the domain stored at or before this timestamp. If the timestamp is in the future, the client MUST issue a fresh DNS query on each verification attempt until the timestamp has passed.

report-to (OPTIONAL): Structured Field String. An absolute HTTPS URL to which failure reports MUST be sent (Section 7.6). Plain HTTP URLs MUST be ignored.

Unrecognised parameters MUST be ignored.

Operational note: Operators SHOULD set the DNS TTL of _dive records to 300 seconds.

5.4. Key Records

When one or more scopes are declared, at least one key record MUST be present and valid. If no valid key record is reachable, the client MUST refuse all resources in the declared scopes.

Key records are DNS TXT records at <Key-ID>._divekey.<domain>. A Key ID MAY contain A-Z, a-z, 0-9, and _; it MUST NOT contain other characters; it is case-sensitive.

5.4.1. Example

```
keyABC._divekey.example.com.  900  IN  TXT  (
  "sig=\"ed25519\", "
  "key=:BASE64RAWKEY:, "
  "allowed-hash=(\"sha256\" \"sha384\" \"sha3-256\"), "
  "cache=900"
)
```

Figure 3: Example of a DIVE Key Record

5.4.2. Parameters

sig (REQUIRED): Signature algorithm. MUST be one of "ed25519" ([RFC8032] Section 5.1) or "ed448" ([RFC8032] Section 5.2). All other algorithms MUST be rejected.

key (REQUIRED): Structured Field Byte Sequence containing the raw public key bytes for the declared algorithm (32 bytes for Ed25519, 57 bytes for Ed448).

allowed-hash (OPTIONAL): Structured Field List of permitted hash algorithms (Strings). Permitted values: "sha256", "sha384", "sha512", "sha3-256", "sha3-384", "sha3-512". MD5, CRC32, and SHA-1 MUST NOT be listed and MUST be rejected. When present, the hash algorithm used in the corresponding signature MUST appear in this list; otherwise verification MUST be treated as failed.

cache (OPTIONAL): Structured Field Integer. Number of seconds the client MAY cache this record (default: 0). MUST NOT exceed 86400 (Section 7.7).

Unrecognised parameters MUST be ignored.

Operational note: Operators SHOULD set the DNS TTL of key records to 900 seconds and SHOULD perform regular key rotation (Section 8.1). A Key ID SHOULD NOT be reused after its associated record has been removed.

5.5. Subdomain-Specific Records

Operators MAY publish policy and key records at any subdomain level. The most specific matching record takes precedence.

6. HTTP Signatures

DIVE uses HTTP Message Signatures [RFC9421] to carry per-resource signatures. A DIVE server MUST include Signature and Signature-Input response headers on all 200 responses for resources within a declared scope.

6.1. Signature Coverage

Each DIVE signature MUST cover the content-digest derived component ([RFC9421] Section 2.4), which commits the signature to the response body. Servers MUST include a Content-Digest header [RFC9530] in each covered response.

6.2. Key Identification and Multiple Signatures

The keyid parameter in Signature-Input MUST be set to the Key ID of the DNS key record used to create the signature. The alg parameter MUST be set to "ed25519" or "ed448" as appropriate.

To support key rotation (Section 8.1), a server MAY include multiple signatures in a single response by providing multiple Signature and Signature-Input entries ([RFC9421] Section 4.2), each referencing a different Key ID.

A DIVE client MUST attempt verification using the available signature entries in order. The client MUST NOT assume prior knowledge of key ordering semantics beyond the order provided in the response. If the client has previously cached a public key matching a given keyid, it SHOULD prefer using the cached key for verification of the corresponding signature entry. Otherwise, the client MUST use the first available signature entry, and proceed to the next entry only if verification fails. The resource MUST be accepted as soon as one verification succeeds.

6.2.1. Example

The following example illustrates two concurrent signatures for key rotation. keyDEF is the newly introduced signing key being rolled in, while keyABC is the previously active key.

```
Content-Digest: sha-256=:BASE64DIGEST:
Signature-Input: sigDEF=("content-digest");keyid="keyDEF"; \
                 alg="ed25519", \
                 sigABC=("content-digest");keyid="keyABC"; \
                 alg="ed25519"
Signature: sigDEF=:BASE64SIG2:, \
           sigABC=:BASE64SIG1:
```

Figure 4: Example of Concurrent Signatures for Key Rotation

Clients that have already cached keyDEF will verify using sigDEF directly. Clients that have not cached any key will attempt verification starting with the first signature entry (sigDEF), and will fall back to sigABC if needed. Clients that have cached keyABC MAY directly use sigABC for verification, but MUST still follow the ordered fallback behavior if verification fails.

The list of signatures SHOULD contain no more than three entries to maintain compatibility with HTTP implementations that impose header-length limits.

6.3. Hash Algorithm Binding

The hash algorithm used to compute Content-Digest MUST be consistent with the allowed-hash parameter of the key record (Section 5.4), when that parameter is present.

Permitted hash algorithms for Content-Digest: sha-256, sha-384, sha-512, sha3-256, sha3-384, sha3-512. MD5, CRC32, and SHA-1 MUST NOT be used and MUST be rejected by the client.

7. Client Implementation

7.1. Step 1: Policy Discovery

The client MUST locate the applicable _dive TXT record by querying from the resource's full FQDN upward, one label at a time, until a record is found or no labels remain. The most specific (deepest) record found applies.

If no record is found, DIVE is not supported.

If the policy record was retrieved without DNSSEC validation, the client MUST treat DIVE as not supported.

If a valid cached copy of the policy record has not expired, the client MUST use it.

Upon retrieving the record, the client MUST verify the v parameter and parsability. It MUST apply invalidate-keys-cache and the https-required directive as specified in Section 5.3. It MUST cache the record per the cache parameter, subject to the 3600-second cap.

DIVE verification is based on the resource's own origin. If a resource is fetched from a third-party domain, the client MUST look up that domain's _dive record, not the referring domain's.

If DIVE is not supported, the client MAY choose to block the resource based on its own requirements.

7.2. Step 2: Scope Determination

If scopes is absent or empty, no resource-level verification is performed. Other directives (e.g., https-required) still apply.

If the resource falls within at least one declared scope, proceed to Step 3. Standard scope detection:

* strict: all resources under the covered domain are in scope.

Custom scope detection is application-defined.

7.3. Step 3: Signature Header Validation

The client MUST verify that Signature and Signature-Input headers are present and syntactically conformant per [RFC9421]. If either header is absent or invalid, the client MUST refuse the resource.

The client MUST also verify that a Content-Digest header is present and parseable per [RFC9530].

7.4. Step 4: Key Resolution

For each signature entry in Signature-Input, the client resolves the key record as follows:

If keyid contains an @-qualified FQDN (e.g., keyABC@example.com), the client MUST verify that the specified FQDN is equal to or a parent of the resource's origin FQDN. If not, the entry MUST be treated as invalid. The client MUST query exactly <Key-ID>._divekey.<fqdn> and MUST NOT search at levels above the specified FQDN.

Otherwise, the client queries from the resource's FQDN upward, label by label, stopping at the level of the applicable policy record, querying <Key-ID>._divekey.<level> at each step. The first record found is used.

In all cases, the DNS query MUST be DNSSEC-validated. A record retrieved without DNSSEC validation MUST be treated as absent.

If no key record is found after cache eviction and a fresh DNS query, the Key ID is unresolvable and verification fails for that entry.

The client MUST cache a valid key record per its cache parameter, subject to the 86400-second cap and any invalidate-keys-cache constraint.

7.5. Step 5: Signature Verification

For each signature entry, the client MUST:

1. Recompute the Content-Digest over the received response body using the hash algorithm declared in the Content-Digest header and verify it matches.

2. Reconstruct the signature base as defined by [RFC9421] Section 2.5 from the Signature-Input components.

3. Retrieve the public key from the resolved key record.

4. If allowed-hash is present in the key record, verify the hash algorithm in Content-Digest is listed; if not, fail this entry.

5. Verify the decoded signature over the signature base using the algorithm declared in sig of the key record.

If at least one entry verifies successfully, the resource MUST be accepted.

If all entries fail:

* By default, the resource MUST be rejected.

* If report-to is present, a report MUST be sent per Section 7.6.

* If report-only is present, the resource MUST be accepted.

The client MUST NOT act upon the resource body before completing verification. The body MAY be downloaded concurrently with DNS resolution, but MUST NOT be delivered to the application until verification is complete.

7.6. Step 6: Reporting

When a resource fails verification (whether blocked or allowed under report-only), and report-to is set, the client MUST POST to that URL with Content-Type: application/json:

```
{
  "report-version": "0.1",
  "timestamp": 1700000000,
  "client": {
    "user-agent": "ExampleClient/1.0"
  },
  "policy": {
    "domain": "example.com",
    "fqdn": "sub.example.com",
    "dnssec-validated": true
  },
  "resource": {
    "url": "https://sub.example.com/downloads/invalid.zip",
    "method": "GET",
    "status-code": 200,
    "scope": "strict"
  },
  "headers-received": {
    "signature-input":
      "sig1=(\"content-digest\");keyid=\"keyABC\";alg=\"ed25519\"",
    "signature": "sig1=:BASE64SIG:",
    "content-digest": "sha-256=:BASE64DIGEST:"
  },
  "key-resolution": [
    {
      "key-id": "keyABC",
      "fqdn-queried": "keyABC._divekey.sub.example.com",
      "found": true,
      "dnssec-validated": true,
      "sig-algorithm": "ed25519"
    }
  ],
  "validation": {
    "hash-algorithm": "sha-256",
    "hash-computed": "BASE64HASHVALUE",
    "signature-valid": false,
    "failure-reason": "signature-mismatch",
    "final-decision": "blocked"
  }
}
```

Figure 5: Example of a DIVE Issue Report

Fields that are absent or not applicable MUST be set to JSON null.

failure-reason permitted values: "missing-headers", "key-not-found", "key-invalid", "dnssec-unavailable", "hash-algorithm-not-allowed", "signature-mismatch", "no-valid-key".

final-decision permitted values: "blocked", "allowed-report-only".

Failure to deliver a report MUST NOT affect the resource acceptance decision.

7.7. Cache Management

Clients MUST enforce an absolute maximum cache duration of 3600 seconds for all _dive records and 86400 seconds for all _divekey records, regardless of the cache parameter. When a key record cannot be resolved, the client MUST evict any cached entry and issue a fresh DNS query before failing.

8. Operational Security

8.1. Key Rotation

To rotate a signing key without service disruption:

1. Generate a new key pair and publish the new public key under a new Key ID in DNS.

2. Wait for the new key record to propagate throughout the DNS.

3. Begin including both the old and new signatures in HTTP responses (using multiple Signature entries per Section 6).

4. Once the new signature has been successfully validated across all resources (ensuring no service disruption), remove the old signature from HTTP responses.

5. Remove the old key record from DNS.

Key IDs MUST NOT be reused after their associated key records have been removed from DNS.

8.2. Response to Key Compromise

Upon discovering a compromised private key, the operator MUST:

1. Immediately begin key rotation (Section 8.1).

2. Set invalidate-keys-cache in all applicable _dive policy records to a timestamp at or after the time of compromise.

3. Remove the compromised key record from DNS as soon as the new key is operational.

4. Maintain the invalidate-keys-cache directive for at least 86400 seconds to ensure all clients have flushed the compromised key, then remove it from the policy record to restore normal caching behavior.

Operators MUST NOT set report-only as a temporary measure during key compromise remediation.

8.3. Private Key Storage

Operators SHOULD NOT store private signing keys on HTTP servers. Private keys SHOULD be stored in offline environments or within Hardware Security Modules (HSMs). However, the security of the signing path itself MUST also be enforced: HSM access interfaces and any remote signing services MUST be strongly authenticated, authorized, rate-limited, and audited, since an attacker who cannot retrieve the private key material may still obtain unauthorized signatures by abusing or compromising a signing endpoint. Signatures SHOULD be pre-computed where feasible and injected at deployment time.

9. Security Considerations

9.1. Threat Model

DIVE protects against an attacker who has compromised the origin web server but not the DNS infrastructure. Such an attacker cannot publish a forged DNSSEC-validated key record, so a DIVE-compliant client will reject any response not signed with a DNS-published key.

DIVE does NOT protect against simultaneous compromise of both the DNS infrastructure and the origin server, or against compromise of the private signing keys.

9.2. DNSSEC as Trust Anchor

DNSSEC is the root of trust for DIVE. Clients MUST obtain DNSSEC validation status for all DNS records they retrieve. Two models are recognised:

* *Stub validator* (RECOMMENDED): the client performs DNSSEC verification itself.

* *Validating resolver*: the client trusts the AD bit from a resolver. The connection to the resolver MUST be over DoH [RFC8484] or DoT [RFC7858].

A record for which DNSSEC validation cannot be confirmed MUST be treated as absent.

9.3. Algorithm Restrictions

Only Ed25519 and Ed448 are permitted for signing. Only SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384, and SHA3-512 are permitted for hashing. Clients MUST reject records or responses referencing any other algorithm. This prevents downgrade attacks.

9.4. Cache Poisoning

Enforcing bounded cache lifetimes for _dive and _divekey records limits the impact of cache-poisoning attacks in which a malicious record with an artificially long cache value is injected. DNSSEC substantially mitigates this attack.

9.5. Privacy

DNS queries for _dive and _divekey records may reveal resource-access patterns to the resolver. Clients SHOULD use DoH or DoT.

Failure reports sent to report-to include the resource URL and User-Agent. Operators MUST handle report data in accordance with applicable privacy regulations.

9.6. Scope of Protection

DIVE verifies response body integrity and authenticity. It does not protect HTTP response headers other than those defined in [RFC9421] and [RFC9530], nor request parameters or cookies.

9.7. Interaction with HTTP Caches

The client MUST NOT hide, suppress, or otherwise obscure HTTP resources or HTTP header fields.

10. IANA Considerations

10.1. DIVE Scope Registry

IANA is requested to create the registry "DIVE Scope Names" under a new registry group "Domain-based Integrity Verification Enforcement (DIVE)", with policy "Specification Required" [RFC8126].

Initial contents:

Scope Name	Description	Detection Criterion	Reference
strict	All resources in the covered domain	Applies to all resources.	This document

Table 1: Initial Contents of the DIVE Scope Names Registry

Custom scopes using the x- prefix are not subject to IANA registration.

10.2. DIVE Directive Registry

IANA is requested to create the registry "DIVE Directive Names" under the same registry group, with policy "Specification Required" [RFC8126].

Initial contents:

Directive Name	Description	Reference
https-required	Client MUST NOT issue plain-HTTP requests; MUST upgrade or abort.	This document
report-only	Client MUST NOT block failures; MUST report them instead.	This document

Table 2: Initial Contents of the DIVE Directive Names Registry

10.3. Hash Algorithms for HTTP Digest Fields

DIVE requires hash algorithms beyond those currently registered in the IANA "Hash Algorithms for HTTP Digest Fields" registry [RFC9530]. IANA is requested to add the following entries to that registry:

Key	Status	Description	Reference
sha-384	Active	SHA-384	[RFC6234]
sha3-256	Active	SHA3-256	[FIPS202]
sha3-384	Active	SHA3-384	[FIPS202]
sha3-512	Active	SHA3-512	[FIPS202]

Table 3: Additions to the Hash Algorithms for HTTP Digest Fields Registry

These algorithms are required to satisfy the cryptographic posture mandated by CNSA Suite 2.0 [CNSA2] and to provide algorithm diversity between the SHA-2 and SHA-3 families, ensuring continued integrity guarantees in the event of a weakness discovered in either family.

SHA-384 provides a stronger security margin than SHA-256 within the SHA-2 family at acceptable performance cost. The SHA-3 family (Keccak) has an entirely different internal construction from SHA-2, meaning a structural break in one family does not compromise the other. For high-assurance environments, the ability to mandate SHA-3 exclusively via the allowed-hash key record parameter is a deliberate design goal of DIVE.

MD5, CRC32, and SHA-1 MUST NOT be registered or used within DIVE.

10.4. DNS Resource Record Types

No new DNS resource record types are defined. DIVE uses DNS TXT records (type 16) [RFC1035].

11. Implementation Status

An experimental implementation of the DIVE protocol is available:

* OpenDIVE Client: https://github.com/diveprotocol/opendive-client (https://github.com/diveprotocol/opendive-client)

* Protocol information: https://diveprotocol.org (https://diveprotocol.org)

12. References

12.1. Normative References

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/info/rfc9421>.

[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>.

[RFC9651] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 9651, DOI 10.17487/RFC9651, September 2024, <https://www.rfc-editor.org/info/rfc9651>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

[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/info/rfc8032>.

[RFC9530] Polli, R. and L. Pardue, "Digest Fields", RFC 9530, DOI 10.17487/RFC9530, February 2024, <https://www.rfc-editor.org/info/rfc9530>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <https://www.rfc-editor.org/info/rfc1035>.

12.2. Informative References

[FIPS202] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", DOI 10.6028/NIST.FIPS.202, August 2015, <https://doi.org/10.6028/NIST.FIPS.202>.

[CNSA2] National Security Agency, "Commercial National Security Algorithm Suite 2.0", September 2022, <https://media.defense.gov/2025/May/30/2003728741/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS.PDF>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/info/rfc8484>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/info/rfc7858>.

Acknowledgements

The author would like to thank Benjamin Schwartz for his review and constructive feedback on earlier versions of this document.

Author’s Address

Mateo Florian Callec
Independent
France
Email: mateo@callec.net