Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 3 October 2026

M. F. Callec
Independent

1 April 2026

Domain-based Integrity Verification Enforcement (DIVE) Version 0.1
draft-callec-dive-00

Abstract

Domain-based Integrity Verification Enforcement (DIVE) version 0.1 is an application-layer protocol that provides cryptographic integrity and authenticity verification of web resources by leveraging the Domain Name System Security Extensions (DNSSEC) as an independent verification channel.

DIVE operates as an additional security layer above HTTP and HTTPS. Public keys and policy configuration are published as DNS TXT records protected by DNSSEC, while HTTP response headers carry per-resource cryptographic signatures. 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 in order to deliver a tampered resource to a DIVE-compliant client.

This document defines the DNS record format, the HTTP header format, the client verification algorithm, the reporting mechanism, and the operational recommendations for the DIVE 0.1 protocol.

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 3 October 2026.

Table of Contents

1. Introduction

Content-delivery infrastructures are a high-value target for attackers. When a web server is compromised, an attacker can silently replace or modify resources served to clients. Transport-layer security (TLS/HTTPS) protects the confidentiality and integrity of data in transit but does not protect against a compromised origin server that serves malicious content over a legitimate TLS session.

Existing mechanisms like Subresource Integrity (SRI) embed expected hashes directly into the HTML markup. However, because this markup is delivered by the primary host, it provides limited security during a full infrastructure breach. The compromised origin can simply alter the hashes to match malicious third-party scripts.

DIVE addresses this threat model by publishing the authoritative policy and public keys through DNSSEC-protected DNS records, which constitute an independent trust anchor. A resource is accepted by a DIVE client only when its signature, carried in HTTP response headers, can be successfully verified against a public key retrieved from DNSSEC-validated DNS. Because DNS and the origin server are typically administered independently, an attacker must control both simultaneously to bypass DIVE.

DIVE is designed to be:

* *Incrementally deployable*: DIVE adds optional DNS records and HTTP response headers; existing infrastructure that does not implement DIVE continues to function normally for non-DIVE clients.

* *Scope-limited*: operators can restrict DIVE verification to specific categories of resources (scopes) rather than all resources.

* *Transparent on failure*: a report-only mode allows operators to monitor verification failures before enforcing blocking.

This document is structured as follows. Section 2 defines terms used throughout the document. Section 3 defines the scope mechanism. Section 4 specifies the DNS record format. Section 5 specifies the HTTP response headers. Section 6 defines the client verification algorithm. Section 8 discusses security properties and threats. Section 9 addresses IANA registration requirements.

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.

The following terms are used in this document:

DIVE client: Any HTTP client (library, tool, or automated agent) that implements the verification algorithm defined in this document.

DIVE server: An HTTP origin server that publishes DIVE DNS records and includes DIVE HTTP response headers on covered resources.

Resource: A single HTTP response body identified by its request URL and received with HTTP status code 200.

Scope: A named category of resources subject to DIVE verification, as defined in Section 3.

Key record: A DNS TXT record that publishes a public key under the naming convention defined in Section 4.4.

Policy record: The _dive DNS TXT record that publishes the DIVE

configuration for a domain or subdomain, as defined in Section 4.3.

Key ID: An operator-assigned identifier that names a specific key record and is referenced in HTTP response headers.

FQDN: Fully Qualified Domain Name.

Structured Field: A value encoded according to the Structured Field Values for HTTP specification [RFC9651].

DNSSEC: DNS Security Extensions [RFC4033] [RFC4034] [RFC4035].

Unix timestamp: The number of seconds elapsed since 1970-01-01T00:00:00Z (UTC), excluding leap seconds, represented as a signed integer.

3. Scopes

A scope identifies a category of resources to which DIVE verification applies. Scopes are declared in the scopes parameter of the policy record (Section 4.3).

3.1. Standard Scopes

This document defines the following standard scope:

strict: DIVE verification MUST be applied to ALL resources served under the domain covered by the applicable policy record, regardless of content type or response headers.

3.2. Custom Scopes

Operators MAY define custom scopes to accommodate application-specific resource categories. Custom scope names:

* MUST begin with the prefix x- (e.g., x-myscope);
* MUST be entirely lowercase;
* MUST contain only the characters a-z and -;
* MUST NOT begin or end with - after the mandatory x- prefix.

Custom scopes are intended for use in closed environments where both the server and the client are controlled by the same operator. The detection logic for custom scopes (i.e., how a client determines whether a given resource falls within a custom scope) is application-defined. Interoperability between different client implementations for custom scopes is explicitly out of scope of this specification.

Custom scopes are intended to cover downloaded artifacts, such as packages, binaries, or other files fetched by a purpose-built client.

A DIVE client that does not recognise a custom scope MUST ignore it and MUST NOT apply DIVE verification to resources solely on the basis of an unrecognised scope.

4. DNS Configuration

4.1. DNSSEC Requirement

A DIVE server SHOULD enable DNSSEC [RFC4033] for all zones that publish DIVE DNS records to ensure record integrity. If a DIVE client retrieves a _dive policy record without DNSSEC validation, it MAY still process the record and perform DIVE verification, though it SHOULD treat the policy as unauthenticated.

4.2. Record Format

All DIVE DNS records are DNS TXT records. Their values MUST be formatted as Structured Field Values as defined in [RFC9651]. Specifically, list values use the Structured Field List syntax and string values use the Structured Field String syntax.

All timestamps used in DIVE DNS records are Unix timestamps represented as Structured Field Integers.

Parameter names and their values, unless otherwise stated, MUST be lowercase.

4.3. The Policy Record (_dive)

The _dive TXT record publishes the DIVE policy for a domain or subdomain. It MUST be placed at the _dive label of the zone or subdomain it governs (e.g., _dive.example.com or _dive.sub.example.com).

A policy record applies to the label at which it is placed and to all subordinate labels, unless a more specific policy record exists at a deeper level. A more specific record always takes precedence over a less specific one.

4.3.1. Example

```
_dive.example.com.  900   IN  TXT  (
  "v=\"dive-draft-00\", "
  "scopes=(\"strict\"), "
  "directives=(\"https-required\"), "
  "cache=900, "
  "invalidate-keys-cache=1700000000, "
  "report-to=\"https://reports.example.com/dive\""
)
```

Figure 1: Example of the ‘_dive‘ DNS record

4.3.2. Parameters

v (REQUIRED): A Structured Field String identifying the protocol version. For this specification the value MUST be "dive-draft-00". Both the parameter name and its value MUST be lowercase. If the v parameter is absent, its value is not "dive-draft-00", or the record cannot be parsed as valid Structured Field syntax, the client MUST treat DIVE as not supported for this domain and MUST NOT attempt to interpret any other parameter in the record.

scopes (OPTIONAL): A Structured Field List of Structured Field Strings naming the scopes for which DIVE verification is enforced (see Section 3). Both the parameter name and all list values MUST be lowercase. If this parameter is absent, no resource scope is enforced; the directives parameter (see below) MAY still apply, but no resource integrity verification will be performed. Operators SHOULD include at least one scope when deploying DIVE.

directives (OPTIONAL): A Structured Field List of Structured Field Strings conveying behavioural directives to clients. Both the parameter name and all list values MUST be lowercase. Defined directive values are:

* "https-required": the client MUST refuse to load any resource under the covered domain over plain HTTP and MUST upgrade or block such requests.

* "report-only": the client MUST NOT block resources that fail DIVE verification; instead it MUST report the failure as defined in Section 6.7 if a report-to URL is present. Verification is still performed; only the enforcement action is suppressed.

A client that encounters an unrecognised directive value MUST ignore that value and continue processing.

cache (OPTIONAL): A Structured Field Integer specifying, in seconds, how long the client MAY cache the policy record, measured from the moment the client stores the record. The value MUST be a non-negative integer. The default value when the parameter is absent is 0 (no caching). Clients MUST NOT cache the policy record for longer than 86400 seconds (24 hours), regardless of the value specified (see Section 6.8).

invalidate-keys-cache (OPTIONAL): A Structured Field Integer carrying a Unix timestamp. When present, the client MUST purge from its local cache all key records for the domain and all subordinate domains covered by this policy record whose cache-storage timestamp is less than or equal to the specified value. If the specified timestamp is in the future, the client MUST NOT serve key records from its cache until that timestamp has passed; it MUST instead issue a fresh DNS query on each verification attempt until the timestamp is reached. The parameter name MUST be lowercase.

report-to (OPTIONAL): A Structured Field String containing an absolute URL to which the client MUST send verification failure reports as defined in Section 6.7. The parameter name MUST be lowercase. The URL MAY refer to a third-party domain. The URL MUST use HTTPS. A plain HTTP URL MUST be ignored by the client.

Unrecognised parameters MUST be ignored by the client.

4.3.3. Operational Recommendations

Operators SHOULD set the DNS TTL of the _dive record to 900 seconds to balance caching performance with the ability to propagate invalidate-keys-cache updates in a timely manner.

4.4. Key Records

When one or more scopes are declared in the policy record, 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 placed at the label <Key-ID>._divekey.<domain>, where <domain> is the domain or subdomain to which the key applies. The _divekey label is distinct from the _dive label used for policy records and MUST NOT be used interchangeably with it.

A Key ID:

* MAY contain uppercase letters (A-Z), lowercase letters (a-z), decimal digits (0-9), and underscores (_);

* MUST NOT contain any other character;

* is case-sensitive.

A key record at a given subdomain level applies only to that level and its subordinate labels. Its Key ID MUST NOT collide with a Key ID used by a parent domain's key record in a manner that would create ambiguity during hierarchical resolution (see Section 6.5).

4.4.1. Example

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

Figure 2: Example of the '_divekey' DNS record

4.4.2. Parameters

sig (REQUIRED): A Structured Field String identifying the digital signature algorithm corresponding to the published public key. Both the parameter name and its value MUST be lowercase. Permitted values are:

* "ed25519": Edwards-curve Digital Signature Algorithm over Curve25519 [RFC8032];

* "ed448": Edwards-curve Digital Signature Algorithm over Curve448 [RFC8032].

Algorithms including but not limited to DSA, ECDSA (any curve), and RSA MUST NOT be used. A client that encounters a sig value other than the permitted values MUST treat the key record as invalid.

key (REQUIRED): A Structured Field Byte Sequence (encoded as base64 within colons as per [RFC9651] Section 3.3.5) containing the raw public key bytes for the algorithm identified by sig. For Ed25519, this is the 32-byte public key encoding defined in [RFC8032] Section 5.1.5. For Ed448, this is the 57-byte public key encoding defined in [RFC8032] Section 5.2.5.

allowed-hash (OPTIONAL): A Structured Field List of Structured Field

Strings enumerating the hash algorithms that the server is permitted to use when computing the digest for this key. Both the parameter name and all list values MUST be lowercase. Permitted values are:

* "sha256": Secure Hash Algorithm 2 (SHA-2) with a 256-bit output, as defined in FIPS 180-4 (https://csrc.nist.gov/pubs/fips/180-4/upd1/final) and widely used in IETF protocols such as [RFC6234];

* "sha384": Secure Hash Algorithm 2 (SHA-2) with a 384-bit output, as defined in FIPS 180-4 (https://csrc.nist.gov/pubs/fips/180-4/upd1/final) and widely used in IETF protocols such as [RFC6234];

* "sha512": Secure Hash Algorithm 2 (SHA-2) with a 512-bit output, as defined in FIPS 180-4 (https://csrc.nist.gov/pubs/fips/180-4/upd1/final) and widely used in IETF protocols such as [RFC6234];

* "sha3-256": Secure Hash Algorithm 3 (SHA-3) with a 256-bit output, as defined in FIPS 202 (https://csrc.nist.gov/pubs/fips/202/final);

* "sha3-384": Secure Hash Algorithm 3 (SHA-3) with a 384-bit output, as defined in FIPS 202 (https://csrc.nist.gov/pubs/fips/202/final);

* "sha3-512": Secure Hash Algorithm 3 (SHA-3) with a 512-bit output, as defined in FIPS 202 (https://csrc.nist.gov/pubs/fips/202/final).

Algorithms including but not limited to MD5, CRC32, and SHA-1 MUST NOT be listed and MUST be rejected by the client. When this parameter is present, the hash algorithm indicated in the DIVE-Sig response header for this Key ID MUST appear in the list; if it does not, the client MUST treat verification as failed for that key. When this parameter is absent, any permitted hash algorithm MAY be used.

cache (OPTIONAL): A Structured Field Integer specifying, in seconds, how long the client MAY cache this key record, measured from the moment the client stores the record. The value MUST be a non-negative integer. The default value when the parameter is absent is 0 (no caching). Clients MUST NOT cache key records for longer than 86400 seconds (24 hours) (see Section 6.8).

Unrecognised parameters MUST be ignored by the client.

4.4.3. Operational Recommendations

Operators SHOULD set the DNS TTL of key records to 900 seconds to allow rapid removal of a compromised key from resolver caches.

Operators SHOULD perform regular key rotation following the procedure described in Section 7.1. A Key ID SHOULD NOT be reused once its associated key record has been removed, to avoid cache-based confusion on clients that still hold the old record.

4.5. Subdomain-Specific Records

Operators MAY publish policy records and key records at any subdomain level. For policy records, the record at the most specific _dive label that matches the resource's FQDN takes precedence. For key records, the record at the most specific _divekey label applicable to the resource's FQDN takes precedence. This allows per-subdomain policy and key overrides without affecting the parent domain.

5. HTTP Response Headers

A DIVE server MUST include the DIVE-Sig HTTP response header on all responses with status code 200 for resources that fall within a declared scope.

5.1. DIVE-Sig

A comma-separated list of entries, each encoding a Key ID, an optional FQDN qualifier, a hash algorithm, and a Base64-encoded signature. Two forms are defined:

* Without FQDN qualifier: keyID:hash-algorithm:BASE64SIG

* With FQDN qualifier: keyID@fqdn:hash-algorithm:BASE64SIG

Example:

```
DIVE-Sig: keyABC:sha256:BASE64SIG1, \
          keyDEF@sub.example.com:sha384:BASE64SIG2
```

Figure 3: Example of the 'DIVE-Sig' HTTP header

The Key ID identifies the key record used to produce the signature. Key IDs are case-sensitive. Each Key ID MUST appear at most once. The list SHOULD contain no more than three entries to ensure compatibility with HTTP implementations that impose limits on header length.

The @fqdn qualifier, when present, instructs the client to resolve the key record for this Key ID starting at <Key-ID>._divekey.<fqdn> rather than at the resource's own FQDN level. The FQDN MUST be the same domain as the resource's origin or a parent domain thereof. The client MUST reject any FQDN that is not equal to or a strict parent of the resource's origin (i.e., child subdomains are not permitted).

The hash algorithm field identifies the algorithm used to compute the digest that was signed. Permitted values are: sha256, sha384, sha512, sha3-256, sha3-384, sha3-512. Algorithms including but not limited to MD5, CRC32, and SHA-1 MUST NOT be used and MUST be rejected by the client.

At least one entry MUST be present.

The signature input is constructed as defined in Section 6.6.1.

5.2. Duplicate Headers

When a client receives duplicate instances of DIVE-Sig (i.e., multiple headers with the same field name), it MUST concatenate their values with a comma separator, as permitted by [RFC9110] Section 5.2, and treat the result as a single header value. If a Key ID appears more than once across the combined value, the client MUST use the first occurrence and MUST ignore all subsequent occurrences of the same Key ID.

6. Client Implementation

6.1. Overview

A DIVE client MUST implement the verification algorithm described in this section for all resources it downloads over HTTP or HTTPS. DIVE is exclusively intended for non-browser clients such as package managers, CLI tools, or automated agents. Browser clients MUST NOT implement or enforce DIVE. Implementors SHOULD support all scopes declared in the applicable policy record.

DIVE verification applies based on the origin of the resource being fetched. If a resource is fetched from a third-party domain, the client MUST look up the DIVE policy record of that third-party domain, not the policy of the domain that triggered the request. A domain that does not publish a valid _dive record is treated as not supporting DIVE, regardless of whether the referring domain supports it.

6.2. Step 1: Policy Discovery

The client MUST attempt to locate the applicable _dive policy record for the resource's FQDN using the following procedure:

1. Starting from the full FQDN of the resource (e.g., _dive.a.b.example.com), query for a _dive TXT record.

2. If no record is found, remove the leftmost label of the domain and repeat the query (e.g., _dive.b.example.com, then _dive.example.com).

3. Continue until a record is found or no more labels remain.

4. The first (most specific) record found is the applicable policy record.

If no _dive record is found at any level, DIVE is not supported for this resource. The client MUST proceed as if DIVE does not exist and MUST NOT block the resource on DIVE grounds.

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

If the client holds a cached copy of the policy record that has not expired, it MUST use the cached copy and MUST NOT issue a DNS query.

Upon retrieving the record, the client MUST verify:

* The v parameter is present and its value is exactly "dive-draft-00".

* The record is parseable as valid Structured Field syntax.

If either condition is not met, the client MUST treat DIVE as not supported for this domain. Unrecognised parameters MUST be ignored.

The client MUST then apply the invalidate-keys-cache directive if present, as described in Section 4.3.

The client MUST apply the directives, including https-required, as described in Section 4.3.

The client MUST cache the policy record for the duration indicated by the cache parameter, subject to the 86400-second maximum defined in Section 6.8.

6.3. Step 2: Scope Determination

The client MUST determine whether the resource falls within any scope listed in the scopes parameter of the applicable policy record.

If the scopes parameter is absent or empty, no resource-level verification is performed for this domain. The client MUST NOT block the resource on DIVE grounds (though other directives such as https-required continue to apply).

If the resource falls within at least one declared scope, the client MUST proceed to Step 3. If the resource does not fall within any declared scope, the client MUST proceed as if DIVE does not apply to this resource.

For standard scope detection:

* strict: every resource served under the covered domain is in scope.

For custom scopes, detection logic is application-defined (see Section 3.2).

6.4. Step 3: Header Validation

For any resource that falls within a declared scope, the client MUST verify that a DIVE-Sig header is present in the HTTP response and syntactically conformant. If the header is absent or invalid, the client MUST refuse the resource.

Duplicate headers MUST be consolidated as described in Section 5.

If the header is absent or syntactically invalid, the client MUST refuse the resource.

6.5. Step 4: Key Resolution

For each entry listed in DIVE-Sig, the client MUST attempt to resolve the corresponding key record as follows:

1. Determine the starting FQDN for resolution:

    * If the entry includes an explicit FQDN qualifier (keyID@fqdn), the client MUST verify that the specified FQDN is the same domain as the resource’s origin or a parent domain thereof. If this condition is not satisfied, the client MUST treat this Key ID as invalid. The client MUST then query for <Key-ID>._divekey.<fqdn>. If the key record is not found at that

exact level, the client MUST NOT search at levels above the specified FQDN. The client MAY search at more specific levels below the specified FQDN that are still applicable to the resource.

* If no FQDN qualifier is present, the client begins resolution at the most specific subdomain applicable to the resource (the level of the resource's FQDN) and works upward, label by label, until reaching the domain covered by the applicable policy record. At each level, the client queries for <Key-ID>._divekey.<level>.

2. Issue a DNSSEC-validated DNS TXT query for <Key-ID>._divekey.<level> at each level in the resolution order. The first record found is used.

3. If no key record is found at any level, the client MUST treat this Key ID as unresolvable. It MUST remove any cached entry for this Key ID and issue a fresh DNS query. If the fresh query also yields no result, the client MUST consider verification as failed for this Key ID.

4. If a key record is found but its content is not conformant with the format defined in Section 4.4, the client MUST treat this Key ID as invalid. Unrecognised parameters MUST be ignored.

5. If the key record was retrieved without successful DNSSEC validation, the client MUST treat this Key ID as invalid.

The client MUST cache a successfully retrieved and valid key record for the duration indicated by the cache parameter of the key record, subject to the 86400-second maximum, UNLESS the invalidate-keys-cache directive requires otherwise.

6.6. Step 5: Signature Verification

6.6.1. Signature Input Construction

For each Key ID being verified, the client MUST construct the signature input as follows:

1. Obtain the raw bytes of the downloaded file as written to disk or delivered to the application layer.

2. Compute the digest of that content using the hash algorithm declared for this Key ID in DIVE-Sig.

3. Construct the byte string to be verified:

```
input = hash_algorithm_name || ":" || hash_bytes_raw
```

Figure 4: Construction of the verification input

where hash_algorithm_name is the ASCII encoding of the algorithm name (e.g., sha256), ":" is the ASCII colon character (0x3A), and hash_bytes_raw is the raw (binary, not hex or base64) digest output.

Example for SHA-256:

```
input = "sha256:" || <32 raw bytes of SHA-256 digest>
```

Figure 5: SHA-256 Input Example

6.6.2. Verification Procedure

1. Retrieve the Base64-encoded signature for this Key ID from DIVE-Sig and decode it.

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

3. Verify the decoded signature over the input byte string using the algorithm declared in the sig parameter of the key record.

4. If the allowed-hash parameter is present in the key record, verify that the hash algorithm declared in DIVE-Sig for this Key ID is listed in allowed-hash. If it is not, the verification MUST be treated as failed for this Key ID.

5. If signature verification succeeds and the hash algorithm is permitted, the resource is considered authentic. The client MUST accept the resource.

6. If verification fails for this Key ID, the client MUST proceed to the next entry listed in DIVE-Sig (if any) and repeat Steps 4-5 of this section. The reason for failure (invalid signature, unresolvable key, etc.) does not affect this fallback behaviour.

If at least one Key ID produces a successful verification, the resource MUST be accepted. If all Key IDs have been tried and none produced a successful verification, the client MUST refuse the resource, subject to the report-only directive.

If at least one Key ID successfully verifies the signature, the resource MUST be accepted.

If all Key IDs fail verification:

* By default, the resource MUST be rejected.

* If the report-only directive is present, the resource MUST still be accepted, and a report MUST be sent to the URL specified in the report-to field of the corresponding _dive record (if present).

The client MUST NOT process (execute, render, or pass to the application layer) the resource body before completing signature verification. The body MAY be downloaded concurrently with DNS resolution (see Section 6.9), but MUST NOT be acted upon until verification is complete.

6.7. Step 6: Enforcement and Reporting

If the resource is refused and the report-only directive is present in the applicable policy record, the client MUST allow the resource to be used as if DIVE were not in effect, and MUST send a report as described below.

If the resource is refused and the report-only directive is absent, the client MUST block the resource.

In either case (blocked or blocked-but-allowed-by-report-only), if the report-to parameter is present in the applicable policy record, the client MUST send an HTTP POST request to the specified URL with Content-Type: application/json and the following JSON body:

```
{
  "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/static/app.js",
    "method": "GET",
    "status-code": 200,
    "scope": "strict"
  },
  "headers-received": {
    "dive-sig": "key1:sha256:BASE64SIG1,key2:sha384:BASE64SIG2"
  },
  "key-resolution": [
    {
      "key-id": "key1",
      "fqdn-queried": "key1._divekey.sub.example.com",
      "found": true,
      "dnssec-validated": true,
      "sig-algorithm": "ed25519"
    }
  ],
  "validation": {
    "hash-algorithm": "sha256",
    "hash-computed": "BASE64HASHVALUE",
    "signature-valid": false,
    "failure-reason": "signature-mismatch",
    "final-decision": "blocked"
  }
}
```

Figure 6: Example of a DIVE Report JSON body

Fields that are absent, unavailable, or not applicable MUST be set to the JSON null value.

6.7.1. Report Field Definitions

report-version: The version of the report format. For this specification the value MUST be "0.1".

timestamp: Unix timestamp of the moment the report is generated.

client.user-agent: The User-Agent string of the DIVE client.

policy.domain: The apex domain of the applicable _dive policy record.

policy.fqdn: The FQDN of the resource being verified.

policy.dnssec-validated: Boolean. true if the policy record was retrieved with successful DNSSEC validation.

resource.url: The absolute URL of the resource that failed verification.

resource.method: The HTTP method used to retrieve the resource.

resource.status-code: The HTTP status code of the response.

resource.scope: The scope under which the resource was evaluated.

headers-received: The raw value of the DIVE-Sig response header as received, after duplicate consolidation.

key-resolution: An array containing one object per Key ID attempted, in the order they were tried. Each object contains:

* key-id: the Key ID string;

* fqdn-queried: the FQDN at which the DNS query was issued (of the form <Key-ID>._divekey.<domain>);

* found: boolean, true if a key record was retrieved;

* dnssec-validated: boolean, true if the key record was DNSSEC-validated;

* sig-algorithm: the value of the sig parameter of the key record, or null if the record was not found or not conformant.

validation.hash-algorithm: The hash algorithm used for the final verification attempt.

validation.hash-computed: The Base64 encoding of the computed digest.

validation.signature-valid: Boolean. true if at least one signature verification succeeded.

validation.failure-reason: A string identifying the reason for the verification failure. Permitted values are:

* "missing-headers": the DIVE-Sig response header is absent or syntactically invalid;

* "key-not-found": no key record was found in DNS for any listed Key ID;

* "key-invalid": a key record was found but is not conformant with this specification;

* "dnssec-unavailable": DNSSEC validation could not be performed or failed;

* "hash-algorithm-not-allowed": the hash algorithm declared in DIVE-Sig is not listed in the allowed-hash parameter of the key record;

* "signature-mismatch": the signature did not verify against the computed digest;

* "no-valid-key": all listed Key IDs were attempted and none produced a successful verification.

validation.final-decision: A string indicating the enforcement outcome. Permitted values are:

* "blocked": the resource was refused;

* "allowed-report-only": the resource was refused by DIVE but allowed due to the report-only directive.

If the report-to URL is unreachable or the POST request fails, the client MUST NOT treat this as a DIVE verification error. The failure to deliver the report has no effect on the acceptance or refusal of the resource.

The report-to URL MAY be on a third-party domain.

6.8. Cache Management

Clients MUST enforce an absolute maximum cache duration of 86400 seconds (24 hours) for all DIVE records (both policy records and key records), regardless of the value of the cache parameter. This limit mitigates denial-of-service attacks in which an attacker publishes a malicious record with an artificially long cache duration.

When a key record cannot be resolved (e.g., the record has been removed from DNS), the client MUST evict any cached entry for that Key ID and issue a fresh DNS query. If the fresh query yields a different valid record, the client MUST restart the verification procedure from Step 4 using the new record. If the new record produces a different verification outcome than the cached record would have, the process continues with the new result.

6.9. Parallelism

To minimise latency, a client MAY initiate DNS resolution for key records concurrently with the download of the resource body. However, the client MUST NOT deliver or act upon the resource body until all required DNS queries have completed and the verification procedure defined in Section 6.6 has concluded.

7. Operational Security Recommendations

7.1. Key Rotation

To rotate a signing key without service disruption, operators SHOULD follow this procedure:

1. Generate a new key pair and publish the new public key under a new Key ID in DNS at the appropriate <Key-ID>._divekey.<domain> label.

2. Wait for the DNS TTL of the old key record to expire, so that all resolver caches have had the opportunity to observe both the old and the new key records.

3. Update all resources to carry signatures and headers referencing the new Key ID.

4. Remove the old key record from DNS only after all affected resources have been updated.

Operators MUST NOT reuse a Key ID after its associated key record has been removed from DNS. Reuse of Key IDs may cause stale cached records on clients to be applied to the wrong key material.

7.2. Response to Key Compromise

Upon discovering that a private key has been compromised, the operator MUST:

1. Immediately begin the key rotation procedure (Section 7.1).

2. Set the invalidate-keys-cache parameter in ALL applicable _dive policy records to a Unix timestamp greater than or equal to the moment at which the key is considered compromised, accounting for DNS TTL propagation delay.

3. Remove the compromised key record from DNS (i.e., remove the <Key-ID>._divekey.<domain> TXT record) as soon as the new key is operational. Until the compromised key record is removed, DIVE clients will continue to accept resources signed with the compromised key. CDN or cache purges of signed resources do not substitute for key record removal.

Operators MUST NOT set the report-only directive as a temporary measure during key compromise remediation. Doing so would disable enforcement and allow an attacker in possession of the compromised private key to serve malicious resources to DIVE clients without triggering a block.

7.3. Private Key Storage

Operators SHOULD NOT store private signing keys on servers that generate or serve HTTP responses. Keys SHOULD be kept in dedicated offline or hardware security module (HSM) environments, with signatures pre-computed and injected at deployment time.

8. Security Considerations

8.1. Threat Model

DIVE is designed to protect against an attacker who has compromised the origin web server but not the DNS infrastructure. Such an attacker can serve arbitrary HTTP response bodies and headers but cannot forge DNSSEC-validated DNS records. Consequently, the attacker cannot publish a new public key or modify the DIVE policy, and a DIVE-compliant client will reject any resource that is not signed with a key whose public counterpart is published in DNSSEC-validated DNS under the _divekey label.

DIVE does NOT protect against an attacker who controls both the DNS infrastructure and the origin server, nor does it protect against the compromise of the private keys or the discovery of a vulnerability in the underlying cryptographic algorithms.

8.2. DNSSEC as a Trust Anchor

The primary security model of DIVE leverages DNSSEC for maximum integrity. If DNSSEC validation is absent or bypassed, a client SHOULD still perform DIVE verification using the available DNS records, although it MUST treat the policy and public keys as unauthenticated (see Section 6.2 and Section 6.5). Operators MUST ensure that DNSSEC is correctly deployed and maintained for all zones publishing DIVE records, including both _dive policy records and _divekey key records.

A DIVE client MUST obtain DNSSEC validation status for all DNS records it retrieves. Two deployment models are recognised:

* *Stub validator*: the client performs DNSSEC signature verification itself, independently of the resolver. This model provides the strongest guarantees and is RECOMMENDED for all DIVE client implementations.

* *Validating resolver*: the client delegates DNSSEC validation to a resolver that sets the Authenticated Data (AD) bit in its responses [RFC4035]. This model is tolerated for implementations where stub validation is impractical. When relying on a validating resolver, the client MUST communicate with the resolver over an encrypted channel (DoH [RFC8484] or DoT [RFC7858]). The client MUST NOT trust the AD bit over an unencrypted connection or when the resolver is not under the client’s control.

In both cases, a record for which DNSSEC validation cannot be confirmed MUST be treated as absent.

8.3. Algorithm Agility and Prohibited Algorithms

DIVE explicitly prohibits weak or legacy algorithms. Signature algorithms other than Ed25519 and Ed448 MUST NOT be accepted. Hash algorithms other than SHA-256, SHA-384, SHA-512, SHA3-256, SHA3-384 and SHA3-512 MUST NOT be accepted. Clients MUST reject resources or key records that reference prohibited algorithms. This prevents downgrade attacks where an attacker replaces headers to reference a weaker algorithm.

8.4. Cache Poisoning and Denial of Service

An attacker who can inject or modify DNS responses may attempt to poison the client’s DIVE record cache with a record that has a very long cache value, causing the client to use stale or malicious configuration for an extended period. The 86400-second cache cap defined in Section 6.8 limits the window of such an attack.

This attack is substantially mitigated by the DNSSEC requirement: an attacker without control of the DNS signing key cannot inject DNSSEC-validated records.

8.5. Privacy Considerations

The client sends DNS queries for _dive policy records and _divekey key records, which may reveal browsing activity to the DNS resolver. Operators and implementors SHOULD consider using DNS-over-HTTPS (DoH) [RFC8484] or DNS-over-TLS (DoT) [RFC7858] to protect the confidentiality of these queries.

The verification failure report sent to the report-to endpoint includes the resource URL, the client User-Agent, and other request metadata. Operators MUST ensure that the report endpoint is operated in accordance with applicable privacy regulations.

8.6. Scope of Protection

DIVE verifies the integrity and authenticity of resource bodies. It does not protect HTTP response headers other than the DIVE-Sig header defined in this document, nor does it protect HTTP request parameters, cookies, or any content rendered inside the resource body (e.g., URLs embedded in HTML).

8.7. Interaction with HTTP Caches

If an HTTP cache returns a cached response, the DIVE headers in that cached response MUST be verified against the DNS records at the time the cached response is used. If the DNS records have changed (e.g., the key has been rotated), the cached response may fail verification and be refused.

DIVE clients MUST NOT bypass intermediate HTTP caches solely for the purpose of retrieving fresh DIVE headers. Standard HTTP cache semantics [RFC9111] apply to DIVE-protected responses.

Operators MUST be aware that intermediate caches (proxies, CDNs) may serve stale DIVE headers after a key rotation or key revocation. Operators SHOULD purge affected cache entries as part of the key rotation procedure (Section 7.1) to prevent verification failures on clients that receive cached responses signed with a key that is no longer present in DNS.

9. IANA Considerations

9.1. HTTP Response Header Fields

IANA is requested to register the following header field names in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" maintained at https://www.iana.org/assignments/http-fields/ (https://www.iana.org/assignments/http-fields/):

Header Field Name	Status	Reference
DIVE-Sig	provisional	This document

Table 1

9.2. DNS Resource Record Types

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

9.3. DIVE Scope Registry

IANA is requested to create a new registry named "DIVE Scope Names" under a new registry group "Domain-based Integrity Verification Enforcement (DIVE)".

The policy for registration in this registry is "Specification Required" as defined in [RFC8126].

The initial contents of the registry are:

Scope Name	Description	Detection Criterion	Reference
strict	All resources in the covered domain	No additional criterion; applies to all resources.	This document

Table 2

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

9.4. DIVE Directive Registry

IANA is requested to create a new registry named "DIVE Directive Names" under the "Domain-based Integrity Verification Enforcement (DIVE)" registry group.

The policy for registration in this registry is "Specification Required" as defined in [RFC8126].

The initial contents of the registry are:

Directive Name	Description	Reference
https-required	The client MUST NOT issue plain-HTTP requests for resources covered by a policy containing the https-required directive. If a request is attempted over plain-HTTP, the client MUST either upgrade the request to HTTPS or abort it.	This document
report-only	Client MUST NOT block failures; MUST report them instead.	This document

Table 3

10. Implementation Status

This section is provided for informational purposes only and is not normative.

An experimental implementation of the DIVE protocol is available:

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

Additional information about the protocol is available at:

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

{backmatter}

11. References

11.1. 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>.

[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>.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, DOI 10.17487/RFC4034, March 2005, <https://www.rfc-editor.org/info/rfc4034>.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, DOI 10.17487/RFC4035, March 2005, <https://www.rfc-editor.org/info/rfc4035>.

[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>.

[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>.

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

[RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[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>.

[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>.

11.2. Informative References

[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>.

Author’s Address

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