
Stream: Internet Engineering Task Force (IETF)
RFC: [9729](#)
Category: Standards Track
Published: January 2025
ISSN: 2070-1721
Authors: D. Schinazi D. Oliver J. Hoyland
Google LLC Guardian Project Cloudflare Inc.

RFC 9729

The Concealed HTTP Authentication Scheme

Abstract

Most HTTP authentication schemes are probeable in the sense that it is possible for an unauthenticated client to probe whether an origin serves resources that require authentication. It is possible for an origin to hide the fact that it requires authentication by not generating Unauthorized status codes; however, that only works with non-cryptographic authentication schemes: cryptographic signatures require a fresh nonce to be signed. Prior to this document, there was no existing way for the origin to share such a nonce without exposing the fact that it serves resources that require authentication. This document defines a new non-probeable cryptographic authentication scheme.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <https://www.rfc-editor.org/info/rfc9729>.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

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

with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
1.1. Conventions and Definitions	3
2. The Concealed HTTP Authentication Scheme	4
3. Client Handling	4
3.1. Key Exporter Context	4
3.1.1. Public Key Encoding	6
3.2. Key Exporter Output	6
3.3. Signature Computation	6
4. Authentication Parameters	7
4.1. The k Parameter	8
4.2. The a Parameter	8
4.3. The p Parameter	8
4.4. The s Parameter	8
4.5. The v Parameter	8
5. Example	8
6. Server Handling	9
6.1. Frontend Handling	9
6.2. Communication Between Frontend and Backend	9
6.3. Backend Handling	10
6.4. Non-Probeable Server Handling	11
7. Requirements on TLS Usage	11
8. Security Considerations	12
9. IANA Considerations	12
9.1. HTTP Authentication Schemes Registry	12
9.2. TLS Keying Material Exporter Labels	13
9.3. HTTP Field Name	13

10. References	13
10.1. Normative References	13
10.2. Informative References	14
Acknowledgments	15
Authors' Addresses	15

1. Introduction

HTTP authentication schemes (see [Section 11](#) of [\[HTTP\]](#)) allow origins to restrict access for some resources to only authenticated requests. While these schemes commonly involve a challenge where the origin asks the client to provide authentication information, it is possible for clients to send such information unprompted. This is particularly useful in cases where an origin wants to offer a service or capability only to "those who know", while all others are given no indication the service or capability exists. Such designs rely on an externally defined mechanism by which keys are distributed. For example, a company might offer remote employee access to company services directly via its website using their employee credentials or offer access to limited special capabilities for specific employees while making discovering (or probing for) such capabilities difficult. As another example, members of less well-defined communities might use more ephemeral keys to acquire access to geography- or capability-specific resources, as issued by an entity whose user base is larger than the available resources can support (by having that entity metering the availability of keys temporally or geographically).

While digital-signature-based HTTP authentication schemes already exist (e.g., [\[HOBA\]](#)), they rely on the origin explicitly sending a fresh challenge to the client, to ensure that the signature input is fresh. That makes the origin probeable as it sends the challenge to unauthenticated clients. This document defines a new signature-based authentication scheme that is not probeable.

1.1. 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.

This document uses the notation from [Section 1.3](#) of [\[QUIC\]](#).

Various examples in this document contain long lines that may be folded, as described in [\[RFC8792\]](#).

2. The Concealed HTTP Authentication Scheme

This document defines the "Concealed" HTTP authentication scheme. It uses asymmetric cryptography. Clients possess a key ID and a public/private key pair, and origin servers maintain a mapping of authorized key IDs to associated public keys.

The client uses a TLS keying material exporter to generate data to be signed (see [Section 3](#)) then sends the signature using the Authorization (or Proxy-Authorization) header field (see [Section 11](#) of [\[HTTP\]](#)). The signature and additional information are exchanged using authentication parameters (see [Section 4](#)). Once the server receives these, it can check whether the signature validates against an entry in its database of known keys. The server can then use the validation result to influence its response to the client, for example, by restricting access to certain resources.

3. Client Handling

When a client wishes to use the Concealed HTTP authentication scheme with a request, it **SHALL** compute the authentication proof using a TLS keying material exporter with the following parameters:

- The label is set to "EXPORTER-HTTP-Concealed-Authentication".
- The context is set to the structure described in [Section 3.1](#).
- The exporter output length is set to 48 bytes (see [Section 3.2](#)).

Note that TLS 1.3 keying material exporters are defined in [Section 7.5](#) of [\[TLS\]](#), while TLS 1.2 keying material exporters are defined in [\[KEY-EXPORT\]](#).

3.1. Key Exporter Context

The TLS key exporter context is described in [Figure 1](#), using the notation from [Section 1.3](#) of [\[QUIC\]](#):

```
Signature Algorithm (16),
Key ID Length (i),
Key ID (..),
Public Key Length (i),
Public Key (..),
Scheme Length (i),
Scheme (..),
Host Length (i),
Host (..),
Port (16),
Realm Length (i),
Realm (..),
```

Figure 1: Key Exporter Context Format

The key exporter context contains the following fields:

Signature Algorithm: The signature scheme sent in the `s` Parameter (see [Section 4.4](#)).

Key ID: The key ID sent in the `k` Parameter (see [Section 4.1](#)).

Public Key: The public key used by the server to validate the signature provided by the client. Its encoding is described in [Section 3.1.1](#).

Scheme: The scheme for this request, encoded using the format of the scheme portion of a URI as defined in [Section 3.1](#) of [URI].

Host: The host for this request, encoded using the format of the host portion of a URI as defined in [Section 3.2.2](#) of [URI].

Port: The port for this request, encoded in network byte order. Note that the port is either included in the URI or is the default port for the scheme in use; see [Section 3.2.3](#) of [URI].

Realm: The realm of authentication that is sent in the realm authentication parameter ([Section 11.5](#) of [HTTP]). If the realm authentication parameter is not present, this **SHALL** be empty. This document does not define a means for the origin to communicate a realm to the client. If a client is not configured to use a specific realm, it **SHALL** use an empty realm and **SHALL NOT** send the realm authentication parameter.

The Signature Algorithm and Port fields are encoded as unsigned 16-bit integers in network byte order. The Key ID, Public Key, Scheme, Host, and Realm fields are length-prefixed strings; they are preceded by a Length field that represents their length in bytes. These length fields are encoded using the variable-length integer encoding from [Section 16](#) of [QUIC] and **MUST** be encoded in the minimum number of bytes necessary.

3.1.1. Public Key Encoding

Both the "Public Key" field of the TLS key exporter context (see above) and the `a` Parameter (see [Section 4.2](#)) carry the same public key. The encoding of the public key is determined by the signature algorithm in use as follows:

RSASSA-PSS algorithms: The public key is an `RSAPublicKey` structure [[PKCS1](#)] encoded in DER [[X.690](#)]. BER encodings that are not DER **MUST** be rejected.

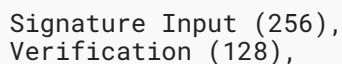
ECDSA algorithms: The public key is an `UncompressedPointRepresentation` structure defined in [Section 4.2.8.2](#) of [[TLS](#)], using the curve specified by the `SignatureScheme`.

Eddsa algorithms: The public key is the byte string encoding defined in [[EdDSA](#)].

This document does not define the public key encodings for other algorithms. In order for a `SignatureScheme` to be usable with the Concealed HTTP authentication scheme, its public key encoding needs to be defined in a corresponding document.

3.2. Key Exporter Output

The key exporter output is 48 bytes long. Of those, the first 32 bytes are part of the input to the signature and the next 16 bytes are sent alongside the signature. This allows the recipient to confirm that the exporter produces the right values. This is described in [Figure 2](#), using the notation from [Section 1.3](#) of [[QUIC](#)]:



```
Signature Input (256),
Verification (128),
```

Figure 2: Key Exporter Output Format

The key exporter output contains the following fields:

Signature Input: This is part of the data signed using the client's chosen asymmetric private key (see [Section 3.3](#)).

Verification: The verification is transmitted to the server using the `v` Parameter (see [Section 4.5](#)).

3.3. Signature Computation

Once the Signature Input has been extracted from the key exporter output (see [Section 3.2](#)), it is prefixed with static data before being signed. The signature is computed over the concatenation of:

- A string that consists of octet 32 (0x20) repeated 64 times

- The context string "HTTP Concealed Authentication"
- A single 0 byte that serves as a separator
- The Signature Input extracted from the key exporter output (see [Section 3.2](#))

For example, if the Signature Input has all its 32 bytes set to 01, the content covered by the signature (in hexadecimal format) would be:

```
2020202020202020202020202020202020202020202020202020202020202020
2020202020202020202020202020202020202020202020202020202020202020
48545450205369676E61747572652041757468656E7469636174696F6E
00
0101010101010101010101010101010101010101010101010101010101010101
```

Figure 3: Example Content Covered by Signature

The purpose of this static prefix is to mitigate issues that could arise if authentication asymmetric keys were accidentally reused across protocols (even though this is forbidden, see [Section 8](#)). This construction mirrors that of the TLS 1.3 CertificateVerify message defined in [Section 4.4.3](#) of [TLS].

The resulting signature is then transmitted to the server using the p Parameter (see [Section 4.3](#)).

4. Authentication Parameters

This specification defines the following authentication parameters.

All of the byte sequences below are encoded using base64url (see [Section 5](#) of [BASE64]) without quotes and without padding. In other words, the values of these byte-sequence authentication parameters **MUST NOT** include any characters other than ASCII letters, digits, dash, and underscore.

The integer below is encoded without a minus and without leading zeroes. In other words, the value of this integer authentication parameter **MUST NOT** include any characters other than digits and **MUST NOT** start with a zero unless the full value is "0".

Using the syntax from [ABNF]:

```
concealed-byte-sequence-param-value = *( ALPHA / DIGIT / "-" / "_" )
concealed-integer-param-value = %x31-39 1*4( DIGIT ) / "0"
```

Figure 4: Authentication Parameter Value ABNF

4.1. The k Parameter

The **REQUIRED** "k" (key ID) Parameter is a byte sequence that identifies which key the client wishes to use to authenticate. This is used by the backend to point to an entry in a server-side database of known keys; see [Section 6.3](#).

4.2. The a Parameter

The **REQUIRED** "a" (public key) Parameter is a byte sequence that specifies the public key used by the server to validate the signature provided by the client. This avoids key confusion issues (see [\[SEEMS-LEGIT\]](#)). The encoding of the public key is described in [Section 3.1.1](#).

4.3. The p Parameter

The **REQUIRED** "p" (proof) Parameter is a byte sequence that specifies the proof that the client provides to attest to possessing the credential that matches its key ID.

4.4. The s Parameter

The **REQUIRED** "s" (signature) Parameter is an integer that specifies the signature scheme used to compute the proof transmitted in the p Parameter. Its value is an integer between 0 and 65535 inclusive from the IANA "TLS SignatureScheme" registry maintained at <https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-signaturescheme>.

4.5. The v Parameter

The **REQUIRED** "v" (verification) Parameter is a byte sequence that specifies the verification that the client provides to attest to possessing the key exporter output (see [Section 3.2](#) for details). This avoids issues with signature schemes where certain keys can generate signatures that are valid for multiple inputs (see [\[SEEMS-LEGIT\]](#)).

5. Example

For example, a client using the key ID "basement" and the signature algorithm Ed25519 [\[ED25519\]](#) could produce the following header field:


```
NOTE: '\' line wrapping per RFC 8792

Authorization: Concealed \
k=YmFzZW11bnQ, \
a=VGhpcyBpcyBh-HB1YmXpYyBrZXkgaW4gdXNl_GhlcmU, \
s=2055, \
v=dmVyaWZpY2F0aW9u_zE2Qg, \
p=QzpcV2luZG93c_xTeXN0ZW0zM1xkcm12ZXJz-ENyb3dkU\
3RyaWt1XEMtMDAwMDAwMDAyOTEtMD-wMC0w_DAwLnN5cw
```

Figure 5: Example Header Field

6. Server Handling

In this section, we subdivide the server role in two:

- The "frontend" runs in the HTTP server that terminates the TLS or QUIC connection created by the client.
- The "backend" runs in the HTTP server that has access to the database of accepted key identifiers and public keys.

In most deployments, we expect both the frontend and backend roles to be implemented in a single HTTP origin server (as defined in [Section 3.6](#) of [HTTP]). However, these roles can be split such that the frontend is an HTTP gateway (as defined in [Section 3.7](#) of [HTTP]) and the backend is an HTTP origin server.

6.1. Frontend Handling

If a frontend is configured to check the Concealed HTTP authentication scheme, it will parse the Authorization (or Proxy-Authorization) header field. If the authentication scheme is set to "Concealed", the frontend **MUST** validate that all the required authentication parameters are present and can be parsed correctly as defined in [Section 4](#). If any parameter is missing or fails to parse, the frontend **MUST** ignore the entire Authorization (or Proxy-Authorization) header field.

The frontend then uses the data from these authentication parameters to compute the key exporter output, as defined in [Section 3.2](#). The frontend then shares the header field and the key exporter output with the backend.

6.2. Communication Between Frontend and Backend

If the frontend and backend roles are implemented in the same machine, this can be handled by a simple function call.

If the roles are split between two separate HTTP servers, then the backend won't be able to directly access the TLS keying material exporter from the TLS connection between the client and frontend, so the frontend needs to explicitly send it. This document defines the "Concealed-Auth-Export" request header field for this purpose. The Concealed-Auth-Export header field's value is a

Structured Field Byte Sequence (see [Section 3.3.5](#) of [STRUCTURED-FIELDS]) that contains the 48-byte key exporter output (see [Section 3.2](#)), without any parameters. Note that Structured Field Byte Sequences are encoded using the non-URL-safe variant of base64. For example:

```
NOTE: '\' line wrapping per RFC 8792

Concealed-Auth-Export: :VGhpc+BleGFtcGx1IFRMU/BleHBvcn\
  Rlc+BvdXRwdXQ/aXMgNDggYn10ZXMgI/+h:
```

Figure 6: Example Concealed-Auth-Export Header Field

The frontend **SHALL** forward the HTTP request to the backend, including the original unmodified Authorization (or Proxy-Authorization) header field and the newly added Concealed-Auth-Export header field.

Note that, since the security of this mechanism requires the key exporter output to be correct, backends need to trust frontends to send it truthfully. This trust relationship is common because the frontend already needs access to the TLS certificate private key in order to respond to requests. HTTP servers that parse the Concealed-Auth-Export header field **MUST** ignore it unless they have already established that they trust the sender. Similarly, frontends that send the Concealed-Auth-Export header field **MUST** ensure that they do not forward any Concealed-Auth-Export header field received from the client.

6.3. Backend Handling

Once the backend receives the Authorization (or Proxy-Authorization) header field and the key exporter output, it looks up the key ID in its database of public keys. The backend **SHALL** then perform the following checks:

- validate that all the required authentication parameters are present and can be parsed correctly as defined in [Section 4](#)
- ensure the key ID is present in the backend's database and maps to a corresponding public key
- validate that the public key from the database is equal to the one in the Authorization (or Proxy-Authorization) header field
- validate that the verification field from the Authorization (or Proxy-Authorization) header field matches the one extracted from the key exporter output
- verify the cryptographic signature as defined in [Section 3.3](#)

If all of these checks succeed, the backend can consider the request to be properly authenticated and can reply accordingly (the backend can also forward the request to another HTTP server).

If any of the above checks fail, the backend **MUST** treat it as if the Authorization (or Proxy-Authorization) header field was missing.

6.4. Non-Probeable Server Handling

Servers that wish to introduce resources whose existence cannot be probed need to ensure that they do not reveal any information about those resources to unauthenticated clients. In particular, such servers **MUST** respond to authentication failures with the exact same response that they would have used for nonexistent resources. For example, this can mean using HTTP status code 404 (Not Found) instead of 401 (Unauthorized).

The authentication checks described above can take time to compute, and an attacker could detect use of this mechanism if that time is observable by comparing the timing of a request for a known nonexistent resource to the timing of a request for a potentially authenticated resource. Servers can mitigate this observability by slightly delaying responses to some nonexistent resources such that the timing of the authentication verification is not observable. This delay needs to be carefully considered to avoid having the delay itself leak the fact that this origin uses this mechanism at all.

Non-probeable resources also need to be non-discoverable for unauthenticated users. For example, if a server operator wishes to hide an authenticated resource by pretending it does not exist to unauthenticated users, then the server operator needs to ensure there are no unauthenticated pages with links to that resource and no other out-of-band ways for unauthenticated users to discover this resource.

7. Requirements on TLS Usage

This authentication scheme is only defined for uses of HTTP with TLS [TLS]. This includes any use of HTTP over TLS as typically used for HTTP/2 [HTTP/2], or HTTP/3 [HTTP/3] where the transport protocol uses TLS as its authentication and key exchange mechanism [QUIC-TLS].

Because the TLS keying material exporter is only secure for authentication when it is uniquely bound to the TLS session [RFC7627], the Concealed authentication scheme requires either one of the following properties:

- The TLS version in use is greater than or equal to 1.3 [TLS].
- The TLS version in use is 1.2, and the extended master secret extension [RFC7627] has been negotiated.

Clients **MUST NOT** use the Concealed HTTP authentication scheme on connections that do not meet one of the two properties above. If a server receives a request that uses this authentication scheme on a connection that meets neither of the above properties, the server **MUST** treat the request as if the authentication were not present.

8. Security Considerations

The Concealed HTTP authentication scheme allows a client to authenticate to an origin server while guaranteeing freshness and without the need for the server to transmit a nonce to the client. This allows the server to accept authenticated clients without revealing that it supports or expects authentication for some resources. It also allows authentication without the client leaking the presence of authentication to observers due to cleartext TLS Client Hello extensions.

Since the freshness described above is provided by a TLS key exporter, it can be as old as the underlying TLS connection. Servers can require better freshness by forcing clients to create new connections using mechanisms such as the GOAWAY frame (see [Section 5.2](#) of [HTTP/3]).

The authentication proofs described in this document are not bound to individual HTTP requests; if the key is used for authentication proofs on multiple requests on the same connection, they will all be identical. This allows for better compression when sending over the wire, but it implies that client implementations that multiplex different security contexts over a single HTTP connection need to ensure that those contexts cannot read each other's header fields. Otherwise, one context would be able to replay the Authorization header field of another. This constraint is met by modern web browsers. If an attacker were to compromise the browser such that it could access another context's memory, the attacker might also be able to access the corresponding key, so binding authentication to requests would not provide much benefit in practice.

Authentication asymmetric keys used for the Concealed HTTP authentication scheme **MUST NOT** be reused in other protocols. Even though we attempt to mitigate these issues by adding a static prefix to the signed data (see [Section 3.3](#)), reusing keys could undermine the security guarantees of the authentication.

Origins offering this scheme can link requests that use the same key. However, requests are not linkable across origins if the keys used are specific to the individual origins using this scheme.

9. IANA Considerations

9.1. HTTP Authentication Schemes Registry

IANA has registered the following entry in the "HTTP Authentication Schemes" registry maintained at <https://www.iana.org/assignments/http-authschemes>:

Authentication Scheme Name: Concealed

Reference: RFC 9729

Notes: None

9.2. TLS Keying Material Exporter Labels

IANA has registered the following entry in the "TLS Exporter Labels" registry maintained at <<https://www.iana.org/assignments/tls-parameters#exporter-labels>>:

Value: EXPORTER-HTTP-Concealed-Authentication

DTLS-OK: N

Recommended: Y

Reference: RFC 9729

9.3. HTTP Field Name

IANA has registered the following entry in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" maintained at <<https://www.iana.org/assignments/http-fields/http-fields.xhtml>>:

Field Name: Concealed-Auth-Export

Status: permanent

Structured Type: Item

Reference: RFC 9729

Comments: None

10. References

10.1. Normative References

- [ABNF]** Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/rfc/rfc5234>>.
- [BASE64]** Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.
- [EdDSA]** 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>>.
- [HTTP]** 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/rfc/rfc9110>>.

- [KEY-EXPORT]** Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/rfc/rfc5705>>.
- [PKCS1]** Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <<https://www.rfc-editor.org/rfc/rfc8017>>.
- [QUIC]** Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <<https://www.rfc-editor.org/rfc/rfc9000>>.
- [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>>.
- [RFC7627]** Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <<https://www.rfc-editor.org/rfc/rfc7627>>.
- [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>>.
- [RFC8792]** Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <<https://www.rfc-editor.org/rfc/rfc8792>>.
- [STRUCTURED-FIELDS]** Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 9651, DOI 10.17487/RFC9651, September 2024, <<https://www.rfc-editor.org/rfc/rfc9651>>.
- [TLS]** Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/rfc/rfc8446>>.
- [URI]** Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/rfc/rfc3986>>.
- [X.690]** ITU-T, "Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X690, ISO/IEC 8825-1:2021, February 2021, <<https://www.itu.int/rec/T-REC-X.690>>.

10.2. Informative References

- [ED25519]** 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>>.

- [HOBA]** Farrell, S., Hoffman, P., and M. Thomas, "HTTP Origin-Bound Authentication (HOBA)", RFC 7486, DOI 10.17487/RFC7486, March 2015, <<https://www.rfc-editor.org/rfc/rfc7486>>.
- [HTTP/2]** Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <<https://www.rfc-editor.org/rfc/rfc9113>>.
- [HTTP/3]** Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <<https://www.rfc-editor.org/rfc/rfc9114>>.
- [MASQUE-ORIGINAL]** Schinazi, D., "The MASQUE Protocol", Work in Progress, Internet-Draft, draft-schinazi-masque-00, 28 February 2019, <<https://datatracker.ietf.org/doc/html/draft-schinazi-masque-00>>.
- [QUIC-TLS]** Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <<https://www.rfc-editor.org/rfc/rfc9001>>.
- [SEEMS-LEGIT]** Jackson, D., Cremers, C., Cohn-Gordon, K., and R. Sasse, "Seems Legit: Automated Analysis of Subtle Attacks on Protocols That Use Signatures", CCS '19: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 2165-2180, DOI 10.1145/3319535.3339813, November 2019, <<https://doi.org/10.1145/3319535.3339813>>.

Acknowledgments

The authors would like to thank many members of the IETF community, as this document is the fruit of many hallway conversations. In particular, the authors would like to thank David Benjamin, Reese Enghardt, Nick Harper, Dennis Jackson, Ilari Liusvaara, François Michel, Lucas Pardue, Justin Richer, Ben Schwartz, Martin Thomson, and Chris A. Wood for their reviews and contributions. The mechanism described in this document was originally part of the first iteration of MASQUE [MASQUE-ORIGINAL].

Authors' Addresses

David Schinazi

Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America
Email: dschinazi.ietf@gmail.com

David M. Oliver

Guardian Project
Email: david@guardianproject.info
URI: <https://guardianproject.info>

Jonathan Hoyland

Cloudflare Inc.

Email: jonathan.hoyland@gmail.com