Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 30 November 2026

M. Sobolyev
Sippy Software, Inc.

29 May 2026

SIP Digest Authentication with X25519 Shared Secrets and Ristretto255 Schnorr Proofs

draft-sip-digest-auth-x25519-ristretto255-schnorr-00

Abstract

This document defines three Session Initiation Protocol (SIP) Digest authentication algorithms that replace password-derived Digest secrets with public-key-based authentication material. Two algorithms derive a response secret from an X25519 shared secret, and one algorithm uses a Fiat-Shamir Schnorr proof over the ristretto255 group.

The mechanisms defined here preserve the existing SIP Digest challenge and authorization header flow while adding Digest parameters that carry public keys and, optionally, an authenticated server challenge proof.

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 30 November 2026.

Copyright Notice

Table of Contents

1. Introduction

This document defines the following SIP Digest authentication algorithm tokens:

* X25519-HKDF-SHA256

* X25519-HMAC-SHA256

* R25519-SCHNORR-SHA256

The algorithms are intended for deployments where the User Agent Client (UAC) and User Agent Server (UAS), or a proxy acting as the authenticating server, have pre-provisioned trust in each other's public keys. They do not define certificate discovery, enrollment, or authorization policy.

The X25519 algorithms derive authentication material from the shared secret computed by [RFC7748] X25519. The ristretto255 algorithm uses a Schnorr-style non-interactive proof over the group defined by [RFC9496].

X25519-HKDF-SHA256 and X25519-HMAC-SHA256 preserve the original SIP Digest flow: the UAS sends a challenge and the UAC sends an authorization response after validating the challenged server public key against local policy. R25519-SCHNORR-SHA256 adds a new capability: the UAC can request that the UAS pre-authenticate the challenge using server-response before the UAC generates its own proof. This prevents an unauthenticated man-in-the-middle attacker from causing the UAC to perform private-key operations for attacker-generated challenges, reducing exposure to private-key extraction attempts that rely on side channels or fault behavior during proof generation.

The existing SIP Digest parameters, including realm, username, nonce, uri, qop, nc, cnonce, and response, retain their existing Digest roles. This document defines four new Digest parameters: server-pubkey, client-pubkey, client-challenge, and server-response.

2. Motivation

Traditional SIP Digest authentication is based on a username and a shared password or password-equivalent secret. This model is well suited to human subscriber authentication and legacy provisioning systems, but it is a poor fit for many machine-to-machine SIP deployments.

Machine-to-machine SIP authentication is often performed between systems, services, trunks, gateways, proxies, application servers, Session Border Controllers (SBCs), and other automated endpoints. These entities do not always naturally correspond to human usernames and passwords. Password-based provisioning also creates operational issues: shared secrets must be generated, distributed, rotated, stored, and protected by both sides. A compromise of the verifier-side secret database can expose password-equivalent material that can be used for impersonation.

Public-key authentication is often a better fit for these environments. Each endpoint can be provisioned with a private key and a corresponding trusted public key. The private key does not need to be shared with the peer, and the public key can be distributed through configuration, inventory, orchestration, certificates, or another trust-management system.

The mechanisms in this document use modern elliptic-curve primitives rather than older RSA-based public-key schemes. X25519 and ristretto255 provide compact public keys and authentication material, efficient constant-time implementations, and simpler fixed-size encodings. These properties are useful for SIP deployments where authentication data is carried in header fields and where endpoints may need to perform many authentication operations per second.

The goal of these mechanisms is to support deployments where authorization is tied to possession of a configured private key rather than knowledge of a shared password, while preserving SIP Digest processing. This model also supports deployments where a request is not accepted merely because it arrives from a particular network, IP address, interface, trunk, or transport path. Instead, the receiver verifies cryptographic proof that the sender possesses the private key corresponding to a trusted public key, and then applies local authorization policy for that key, realm, endpoint, account, route, or service.

SIP over TLS can provide transport confidentiality, transport integrity, and transport-layer peer authentication. However, TLS authentication is tied to the transport connection and does not by itself define SIP Digest identities, Digest realm scoping, SIP

authorization policy, or authentication of individual SIP requests and entity bodies through the Digest qop model. The mechanisms in this document are therefore complementary to TLS rather than replacements for it.

3. Conventions and 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.

UAC SIP User Agent Client.

UAS SIP User Agent Server.

X25519 public key A 32-octet raw X25519 public key as specified by [RFC7748].

ristretto255 public key A 32-octet compressed ristretto255 group element as specified by [RFC9496].

base64url Base64url encoding without padding, using the URL and filename safe alphabet from [RFC4648] Section 5.

digest-uri The value of the Digest uri parameter.

entity-body The SIP message body used for the qop=auth-int calculation.

SHA-256 The SHA-256 hash function specified by [RFC6234].

4. Digest Parameters and Syntax

The parameters in this section extend the Digest auth-param syntax used by SIP Digest authentication [RFC3261] [RFC8760]. Unless otherwise stated, each value is carried as a quoted string containing an unpadded base64url value.

```
base64url-char     = ALPHA / DIGIT / "-" / "_"
base64url-value    = 1*base64url-char
b64q               = DQUOTE base64url-value DQUOTE

server-pubkey      = "server-pubkey" EQUAL b64q
client-pubkey      = "client-pubkey" EQUAL b64q
client-challenge   = "client-challenge" EQUAL b64q
server-response    = "server-response" EQUAL b64q
```

ALPHA, DIGIT, and DQUOTE are defined by [RFC5234]. EQUAL is inherited from the SIP grammar in [RFC3261].

4.1. server-pubkey

The server-pubkey parameter is sent by the UAS in WWW-Authenticate, or by a proxy in Proxy-Authenticate.

For X25519-HKDF-SHA256 and X25519-HMAC-SHA256, it contains the raw 32-octet X25519 public key of the authenticating server. For R25519-SCHNORR-SHA256, it contains the 32-octet ristretto255 public key of the authenticating server.

A UAC receiving this parameter MUST verify that the decoded public key is trusted for the challenged SIP realm, peer, route, outbound proxy, or configured authentication domain before generating an authorization response.

4.2. client-pubkey

The client-pubkey parameter is sent by the UAC in Authorization or Proxy-Authorization.

For X25519-HKDF-SHA256 and X25519-HMAC-SHA256, it contains the raw 32-octet X25519 public key of the authenticating client. For R25519-SCHNORR-SHA256, it contains the 32-octet ristretto255 public key whose corresponding private scalar is proven by the UAC.

A UAS receiving this parameter MUST verify that the decoded public key is trusted for the claimed username, if present, and realm. If username is absent, the UAS MUST verify that client-pubkey is trusted for the realm and for the applicable account, subscriber, endpoint, authorization identity, or other local authorization record.

4.3. client-challenge

The client-challenge parameter is sent by the UAC in an initial Authorization or Proxy-Authorization header when requesting an authenticated server challenge. It contains at least 128 bits of client-generated randomness.

The parameter is used by R25519-SCHNORR-SHA256 to allow the UAS to prove possession of the private scalar corresponding to server-pubkey in the 401 or 407 challenge response. The UAS MUST NOT echo client-challenge in WWW-Authenticate or Proxy-Authenticate.

The UAC MUST verify server-response using the locally remembered client-challenge value that it generated and sent. The UAC MUST NOT use any reflected or received client-challenge value from a response header when verifying server-response.

4.4. server-response

The server-response parameter is sent by the UAS in WWW-Authenticate, or by a proxy in Proxy-Authenticate, when responding to a request that contained client-challenge.

The value is base64url(R || s), where R is a compressed ristretto255 commitment of 32 octets and s is a canonical scalar modulo the ristretto255 group order, encoded as 32 octets. The decoded value is therefore exactly 64 octets.

5. Transcript Encoding

All cryptographic transcripts defined by this document use the following deterministic encoding to avoid ambiguity between adjacent SIP and Digest fields.

Transcript(label, field-list) is the octet string formed by concatenating the US-ASCII label, a line feed (%x0A), and then, for each field in order:

1. the US-ASCII field name,
2. a colon (%x3A),
3. the decimal octet length of the field value encoded in US-ASCII without leading zeroes,
4. another colon,
5. the field value octets, and
6. a line feed.

Unless otherwise stated, SIP and Digest string fields are encoded exactly as their field values appear after Digest quoted-string unescaping. Binary fields, including public keys, shared secrets, commitments, scalars, and hash outputs, are encoded as their raw octets when they are transcript fields. The final line feed is part of the transcript.

For qop=auth, the body-hash transcript field is the zero-length value. For qop=auth-int, it is the raw 32-octet SHA-256 digest of the entity body.

6. Common Digest Fields

The Digest fields realm, username, nonce, uri, qop, nc, cnonce, and response retain their usual meanings in SIP Digest authentication, except as explicitly stated in this section. For all algorithms in this document, realm is REQUIRED and qop=auth and qop=auth-int are supported.

For the algorithms defined by this document, username is OPTIONAL. For all calculations and transcripts defined by this document, an absent username is equivalent to an explicitly empty username. All formulas that reference username use the received username value if present, or a zero-length string if absent.

If username is present, it is an identity hint. The verifier MAY use it to select an account, subscriber, endpoint, or authorization record. The verifier MUST still verify that client-pubkey is trusted for the claimed username and realm. If username is absent, the verifier MUST identify the peer by client-pubkey and realm using local policy.

When qop=auth-int is used, the authentication calculation includes the entity-body hash. For SIP INVITE requests carrying Session Description Protocol (SDP), this provides integrity protection for the SDP body against attackers that cannot produce the required authentication response.

The response value is algorithm-specific. The X25519-HKDF-SHA256 and X25519-HMAC-SHA256 algorithms encode response as lowercase hexadecimal. The R25519-SCHNORR-SHA256 algorithm uses a different format: response is an unpadded base64url encoding of the 64-octet Schnorr proof R_c || s_c.

7. Algorithm: X25519-HKDF-SHA256

7.1. Challenge

The UAS sends a Digest challenge with algorithm=X25519-HKDF-SHA256 and server-pubkey.

```
WWW-Authenticate: Digest
    realm="sip.example.net",
    algorithm=X25519-HKDF-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    qop="auth,auth-int",
    server-pubkey="xRbeh9_DZBrONNFs7P8rhZhcLlXSw79RU5frhaOvIZc"
```

7.2. Authorization Response

The UAC sends a Digest authorization response with client-pubkey. The username parameter MAY be omitted.

Example with username present:

```
Authorization: Digest
    username="alice",
    realm="sip.example.net",
    algorithm=X25519-HKDF-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    uri="sip:bob@example.net",
    qop=auth-int,
    nc=00000001,
    cnonce="q1w2e3r4t5y6",
    client-pubkey="fPEDas5tsJgGPG_QMPVECsTySsC0TFGHRVSNcWSmABQ",
    response="281847efc4a3739d638de9f729fb6b5565bb2f2a006f1719"
```

Example with username absent:

```
Authorization: Digest
    realm="sip.example.net",
    algorithm=X25519-HKDF-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    uri="sip:bob@example.net",
    qop=auth-int,
    nc=00000001,
    cnonce="q1w2e3r4t5y6",
    client-pubkey="zLhVVAHkZJ8oNU-v2DKauUB_TvWBn1dpcBFtQL6kd6s",
    response="fb2e80bda9f83361e995f1c7662b5cc32ee259f3c4ea9d66"
```

7.3. Shared Secret Calculation

The UAC computes Z = X25519(client-private-key, server-pubkey). The UAS computes Z = X25519(server-private-key, client-pubkey). Both sides MUST reject the exchange if the computed X25519 shared secret is the all-zero value.

The Digest response key K is derived with HKDF-SHA256 [RFC5869]:

```
salt = Transcript("SIP-Digest-X25519-HKDF-SHA256-salt-v1",
                  nonce, cnonce)

info = Transcript("SIP-Digest-X25519-HKDF-SHA256-info-v1",
                  algorithm, username, realm, nonce, cnonce,
                  server-pubkey, client-pubkey)

K = HKDF-SHA256(IKM = Z, salt = salt, info = info, L = 32)
```

7.4. Response Calculation

The response value is computed as follows:

```
body-hash = ""                   ; for qop=auth
body-hash = SHA-256(entity-body) ; for qop=auth-int

HA1 = SHA-256(Transcript(
          "SIP-Digest-X25519-HKDF-SHA256-HA1-v1",
          username, realm, K))

HA2 = SHA-256(Transcript(
          "SIP-Digest-X25519-HKDF-SHA256-HA2-v1",
          method, digest-uri, qop, body-hash))

response = SHA-256(Transcript(
          "SIP-Digest-X25519-HKDF-SHA256-response-v1",
          HA1, nonce, nc, cnonce, qop, HA2))
```

The response value is encoded as lowercase hexadecimal SHA-256 output.

8. Algorithm: X25519-HMAC-SHA256

The X25519-HMAC-SHA256 algorithm uses the same server-pubkey and client-pubkey parameters as X25519-HKDF-SHA256, but computes the final response as an HMAC [RFC2104] rather than as a Digest-style hash chain.

8.1. Challenge

The UAS sends a Digest challenge with algorithm=X25519-HMAC-SHA256 and server-pubkey.

```
WWW-Authenticate: Digest
    realm="sip.example.net",
    algorithm=X25519-HMAC-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    qop="auth,auth-int",
    server-pubkey="bzeK9qEcpcMStQYlqQbVs5NfjMMu76AlsKV587PvlT8"
```

8.2. Authorization Response

The UAC sends a Digest authorization response with client-pubkey. The username parameter MAY be omitted.

Example with username present:

```
Authorization: Digest
    username="alice",
    realm="sip.example.net",
    algorithm=X25519-HMAC-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    uri="sip:bob@example.net",
    qop=auth-int,
    nc=00000001,
    cnonce="q1w2e3r4t5y6",
    client-pubkey="hDX3Ky19NCnnzATsdgvdu5BvPqwGdOOP1koEUWU6gZ4",
    response="3401ac99fa30dacdeeb245749639dcd46c504611df2398dd"
```

Example with username absent:

```
Authorization: Digest
    realm="sip.example.net",
    algorithm=X25519-HMAC-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    uri="sip:bob@example.net",
    qop=auth-int,
    nc=00000001,
    cnonce="q1w2e3r4t5y6",
    client-pubkey="dywz9UEYmTmzGXuNrfSZGw1maX0vGJ3i4kMPaHmRyzo",
    response="14ec53f746359ba4d64ff8cf8d5667cbe64b14c9b50c4dd4"
```

8.3. Shared Secret Calculation

The UAC computes Z = X25519(client-private-key, server-pubkey). The UAS computes Z = X25519(server-private-key, client-pubkey). Both sides MUST reject the exchange if the computed X25519 shared secret is the all-zero value.

```
K = SHA-256(Transcript(
        "SIP-Digest-X25519-HMAC-SHA256-key-v1",
        Z, algorithm, username, realm, nonce, cnonce,
        server-pubkey, client-pubkey))
```

8.4. HMAC Transcript

```
body-hash = ""                  ; for qop=auth
body-hash = SHA-256(entity-body) ; for qop=auth-int

transcript = Transcript(
        "SIP-Digest-X25519-HMAC-SHA256-response-v1",
        username, realm, nonce, nc, cnonce, qop,
        method, digest-uri, body-hash, server-pubkey, client-pubkey)

response = HMAC-SHA256(K, transcript)
```

The response value is encoded as lowercase hexadecimal HMAC-SHA256 output.

9. Algorithm: R25519-SCHNORR-SHA256

The R25519-SCHNORR-SHA256 algorithm defines a Schnorr-style Fiat-Shamir non-interactive proof of knowledge over the ristretto255 group. The UAC proves knowledge of a scalar x_c corresponding to client-pubkey = x_c * G, where G is the canonical ristretto255 base point.

9.1. Optional Authenticated Server Challenge Request

A UAC MAY request an authenticated server challenge by including client-challenge in an initial request.

```
INVITE sip:bob@example.net SIP/2.0
Authorization: Digest
    algorithm=R25519-SCHNORR-SHA256,
    client-challenge="0Xc7ag8QRtRWlamLDDozsA"
```

This header does not authenticate the UAC. It only supplies freshness for an authenticated server challenge. A UAC that sends client-challenge MUST remember the value locally until it receives and verifies the corresponding WWW-Authenticate or Proxy-Authenticate challenge, or until the transaction is abandoned.

9.2. Challenge

The UAS sends a Digest challenge with algorithm=R25519-SCHNORR-SHA256 and server-pubkey. If the request contained a valid client-challenge, the UAS MAY include server-response.

```
WWW-Authenticate: Digest
    realm="sip.example.net",
    algorithm=R25519-SCHNORR-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    qop="auth,auth-int",
    server-pubkey="xBiXzi82PKyiSqcRBXJauiNECbQDQZfzt-RRwzsKAXs",
    server-response="neTvyoW2SLh32ob-UTteu_ragDBe2Ub3wqhm969gjlw"
```

9.3. Authenticated Server Challenge Proof

When server-response is present, it proves possession of the private scalar corresponding to server-pubkey. The UAS has private scalar x_s and public key A_s = x_s * G, where A_s is server-pubkey.

```
T_srv_chal = Transcript(
    "SIP-Digest-R25519-SCHNORR-SHA256-ServerChallenge-v1",
    algorithm, method, digest-uri, realm, nonce, qop-list,
    server-pubkey, client-challenge)

R_s = r_s * G
c_s = SHA-256(Transcript(
          "SIP-Digest-R25519-SCHNORR-SHA256-ServerChallenge-c-v1",
          T_srv_chal, R_s)) mod L
s_s = r_s + c_s * x_s mod L

server-response = base64url(R_s || s_s)
```

r_s is a fresh random scalar and L is the ristretto255 group order. The UAC verifies by recomputing the transcript and checking s_s * G == R_s + c_s * A_s.

The UAC MUST reject the challenge if server-response is required by local policy and is absent, malformed, invalid, not bound to the locally remembered client-challenge, or not valid for the received server-pubkey.

9.4. Authorization Proof

The UAC sends a Digest authorization response with client-pubkey. The username parameter MAY be omitted. The response parameter contains base64url(R_c || s_c), where R_c is a compressed ristretto255 commitment and s_c is a canonical scalar modulo the group order. This differs from the hexadecimal Digest response values used by the X25519 algorithms. The decoded value is exactly 64 octets.

```
A_c = x_c * G
R_c = r_c * G

body-hash = ""                   ; for qop=auth
body-hash = SHA-256(entity-body) ; for qop=auth-int

T_uac = Transcript(
    "SIP-Digest-R25519-SCHNORR-SHA256-UAC-v1",
    algorithm, username, realm, nonce, nc, cnonce, qop,
    method, digest-uri, body-hash, server-pubkey, client-pubkey)

c_c = SHA-256(Transcript(
          "SIP-Digest-R25519-SCHNORR-SHA256-UAC-c-v1",
          T_uac, R_c)) mod L
s_c = r_c + c_c * x_c mod L

response = base64url(R_c || s_c)
```

The UAS decodes A_c from client-pubkey, reconstructs T_uac from the received request and Digest parameters, and accepts only if s_c * G == R_c + c_c * A_c. The UAS MUST reject malformed ristretto255 encodings, non-canonical scalars, invalid proof lengths, and proofs that are not valid for the exact received transcript.

A proof generated for one method, URI, nonce, cnonce, nonce-count, qop value, body, realm, server public key, client public key, or username value MUST NOT be accepted for another.

10. Verification Rules

A receiver MUST reject authentication if any of the following are true:

* server-pubkey is absent from the challenge;

* client-pubkey is absent from the authorization response;

* realm is absent from the challenge or authorization response;

* a public key is malformed for the selected algorithm;

* the peer public key is not trusted for the claimed identity or realm;

* nonce is expired, unknown, malformed, or already consumed;

* nc does not increase monotonically for the nonce and client identity pair;

* cnonce is absent when qop is present;

* qop is unsupported;

* response is malformed; or

* response does not match or verify against the locally computed value.

For R25519-SCHNORR-SHA256, a receiver MUST also reject authentication if server-response is malformed when present, is absent when required by local policy, or does not verify against the locally remembered client-challenge.

A UAS MUST NOT authenticate a UAC merely because client-pubkey is present. The client-pubkey value MUST be bound to a trusted identity, such as the username if present, configured endpoint, account, subscriber, realm, or equivalent local authorization record.

A UAC MUST NOT answer a challenge merely because server-pubkey is present. The server-pubkey value MUST be present in the UAC's trusted key list for the challenged peer, realm, outbound proxy, or service domain.

Implementations MUST reject algorithm identifiers other than those explicitly defined by this document or locally configured for this mechanism.

11. Replay Protection

The UAS MUST generate nonces with sufficient entropy and unpredictability. The UAS MUST reject replayed tuples of client-pubkey, nonce, nc, and cnonce. If username is present, the UAS MAY also include username in the replay cache key. The UAS SHOULD expire nonces after a short interval and SHOULD bind nonces to the challenged realm, selected algorithm, and server-pubkey.

The UAC SHOULD generate a fresh client-challenge for each initial authenticated challenge request. The UAC MUST NOT accept a server-response generated for a different client-challenge.

12. Body Integrity

When qop=auth-int is used, the entity-body hash is included in the authentication calculation. For SIP requests carrying SDP, this protects the SDP body against modification by an attacker that does not know the X25519-derived authentication secret or cannot generate the required ristretto255 Schnorr proof.

If intermediaries are expected to modify the SDP body, qop=auth-int can fail unless the modification happens before the UAC computes the authorization response or unless the deployment explicitly permits such behavior.

13. Examples

13.1. Initial Request Asking for Authenticated Server Challenge

```
INVITE sip:bob@example.net SIP/2.0
Via: SIP/2.0/TLS client.example.org;branch=z9hG4bK776asdhds
From: <sip:alice@example.org>;tag=1928301774
To: <sip:bob@example.net>
Call-ID: a84b4c76e66710@example.org
CSeq: 314159 INVITE
Authorization: Digest algorithm=R25519-SCHNORR-SHA256,
    client-challenge="QG7xYpk5XlVz9hHMKx3uRg"
Content-Length: 0
```

13.2. Authenticated R25519-SCHNORR-SHA256 Challenge

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/TLS client.example.org;branch=z9hG4bK776asdhds
From: <sip:alice@example.org>;tag=1928301774
To: <sip:bob@example.net>;tag=a6c85cf
Call-ID: a84b4c76e66710@example.org
CSeq: 314159 INVITE
WWW-Authenticate: Digest realm="sip.example.net",
    algorithm=R25519-SCHNORR-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    qop="auth,auth-int",
    server-pubkey="V8nM6R1l7Pp4sLdbZSyGx6Fv5F1LxPvV9gzLg4YhV2I",
    server-response="Jvx3L4ZPTVq43cacYBONePT4nInA-F8FPqL1mv7ORC0"
Content-Length: 0
```

13.3. R25519-SCHNORR-SHA256 Authorization with Username

```
INVITE sip:bob@example.net SIP/2.0
Via: SIP/2.0/TLS client.example.org;branch=z9hG4bK776asdhds
From: <sip:alice@example.org>;tag=1928301774
To: <sip:bob@example.net>
Call-ID: a84b4c76e66710@example.org
CSeq: 314160 INVITE
Contact: <sip:alice@client.example.org>
Authorization: Digest username="alice",
    realm="sip.example.net",
    algorithm=R25519-SCHNORR-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    uri="sip:bob@example.net",
    qop=auth-int,
    nc=00000001,
    cnonce="q1w2e3r4t5y6",
    client-pubkey="LKz2bq0TLeHqkCJ2m6v9MGWQp9WnZtDZ9pYyHk4IoX0",
    response="mU7Wgqm2wHIAk993xXo6OXKMQBNtgl-mFJQ_-Rgo8oI"
Content-Type: application/sdp
Content-Length: ...
```

13.4. R25519-SCHNORR-SHA256 Authorization without Username

```
INVITE sip:bob@example.net SIP/2.0
Via: SIP/2.0/TLS client.example.org;branch=z9hG4bK776asdhds
From: <sip:alice@example.org>;tag=1928301774
To: <sip:bob@example.net>
Call-ID: a84b4c76e66710@example.org
CSeq: 314160 INVITE
Contact: <sip:alice@client.example.org>
Authorization: Digest
    realm="sip.example.net",
    algorithm=R25519-SCHNORR-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    uri="sip:bob@example.net",
    qop=auth-int,
    nc=00000001,
    cnonce="q1w2e3r4t5y6",
    client-pubkey="a0t5YmU8Ppgp0zLGgwsVXnxViPImkjyyzWTDj5wfmxE",
    response="bde_KoYDFV3SYJa55WppUdlgLjhgM3vqT5U7qa6YR_o"
Content-Type: application/sdp
Content-Length: ...
```

13.5. X25519-HKDF-SHA256 Challenge

```
SIP/2.0 401 Unauthorized
Via: SIP/2.0/TLS client.example.org;branch=z9hG4bK776asdhds
From: <sip:alice@example.org>;tag=1928301774
To: <sip:bob@example.net>;tag=a6c85cf
Call-ID: a84b4c76e66710@example.org
CSeq: 314159 INVITE
WWW-Authenticate: Digest realm="sip.example.net",
    algorithm=X25519-HKDF-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    qop="auth,auth-int",
    server-pubkey="V8nM6R1l7Pp4sLdbZSyGx6Fv5F1LxPvV9gzLg4YhV2I"
Content-Length: 0
```

13.6. X25519-HKDF-SHA256 Authorization with Username

```
INVITE sip:bob@example.net SIP/2.0
Via: SIP/2.0/TLS client.example.org;branch=z9hG4bK776asdhds
From: <sip:alice@example.org>;tag=1928301774
To: <sip:bob@example.net>
Call-ID: a84b4c76e66710@example.org
CSeq: 314160 INVITE
Contact: <sip:alice@client.example.org>
Authorization: Digest username="alice",
    realm="sip.example.net",
    algorithm=X25519-HKDF-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    uri="sip:bob@example.net",
    qop=auth-int,
    nc=00000001,
    cnonce="q1w2e3r4t5y6",
    client-pubkey="LKz2bq0TLeHqkCJ2m6v9MGWQp9WnZtDZ9pYyHk4IoX0",
    response="92eab8507440bf720bfffdffcdab35c785ddee69d419db58"
Content-Type: application/sdp
Content-Length: ...
```

13.7. X25519-HKDF-SHA256 Authorization without Username

```
INVITE sip:bob@example.net SIP/2.0
Via: SIP/2.0/TLS client.example.org;branch=z9hG4bK776asdhds
From: <sip:alice@example.org>;tag=1928301774
To: <sip:bob@example.net>
Call-ID: a84b4c76e66710@example.org
CSeq: 314160 INVITE
Contact: <sip:alice@client.example.org>
Authorization: Digest
    realm="sip.example.net",
    algorithm=X25519-HKDF-SHA256,
    nonce="NQ7x0vR3VnP0aK9fW6tDHA",
    uri="sip:bob@example.net",
    qop=auth-int,
    nc=00000001,
    cnonce="q1w2e3r4t5y6",
    client-pubkey="q6wi4GXRx1poYAVvqcp7nmC4FU0MDxnvooatixWp1mY",
    response="d857749505ba1cc5262d945bfd793d7988f6a0d675eab113"
Content-Type: application/sdp
Content-Length: ...
```

14. IANA Considerations

This document requests registration of the following Digest algorithm tokens in the applicable Digest algorithm registry:

* X25519-HKDF-SHA256

* X25519-HMAC-SHA256

* R25519-SCHNORR-SHA256

The following Digest authentication parameters are also defined by this document: server-pubkey, client-pubkey, client-challenge, and server-response.

15. Implementation Notes

Implementations SHOULD use well-reviewed cryptographic libraries for X25519, HKDF-SHA256, HMAC-SHA256, SHA-256, and ristretto255. Implementations SHOULD avoid accepting algorithm aliases.

Implementations SHOULD domain-separate every proof or MAC input exactly as specified.

Implementations SHOULD log authentication failures without logging private keys, shared secrets, raw proof scalars, derived keys, or complete authentication transcripts.

Implementations SHOULD treat server-pubkey and client-pubkey as identity-bearing material and apply local authorization policy before accepting a request.

Implementations need to account for generic Digest parsers that assume username is always present. Implementations of this specification MUST support the absence of username for the algorithms defined here.

16. Security Considerations

This mechanism prevents impersonation only when both sides have securely provisioned peer public keys, bind those keys to the applicable realm and authorization identity, and enforce the replay protections required by this document. X25519 by itself is not authentication. Authentication is achieved only by deriving a secret from a trusted peer public key and proving possession of the corresponding private key through the Digest response.

For X25519-HKDF-SHA256 and X25519-HMAC-SHA256, the UAC trusts the UAS X25519 public key and the UAS trusts the UAC X25519 public key. For R25519-SCHNORR-SHA256, the UAS trusts the UAC ristretto255 public key, and the UAC trusts the UAS ristretto255 public key before answering the challenge. The server-response parameter provides authenticated server challenges when required by local policy.

The ristretto255 Schnorr algorithm authenticates the UAC by proving knowledge of the private scalar corresponding to client-pubkey, bound to the SIP Digest transcript. The server-response parameter authenticates the server challenge by proving knowledge of the private scalar corresponding to server-pubkey, bound to a UAC-generated client-challenge.

Nonce freshness, nonce-count validation, client nonce validation, and replay-cache enforcement are part of the security of this mechanism. An implementation that accepts replayed Digest responses can authenticate a stale request even when the cryptographic proof or MAC is otherwise valid.

These algorithms authenticate SIP Digest exchanges and, when qop=auth-int is used, provide integrity protection for the authenticated SIP entity body. They do not establish a confidential transport channel, protect SIP metadata, or provide media security. Deployments requiring confidential signaling, SIP metadata protection, transport-layer peer authentication, or media security SHOULD use SIP over TLS and appropriate media-security mechanisms.

Compromise of a provisioned private key enables impersonation for the identities and realms authorized for the corresponding public key. Deployments need operational procedures for protecting, rotating, and revoking keys according to local policy.

Schnorr nonces MUST be generated with high-quality randomness or by a deterministic nonce generation construction with equivalent security. Reusing a Schnorr nonce with the same private scalar can reveal the private scalar.

Implementations MUST use constant-time comparison when checking MAC or hash responses and SHOULD use constant-time scalar and group operations where provided by the cryptographic library.

17. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[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>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, July 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5234] 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/info/rfc5234>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[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>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[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>.

[RFC8760] Shekh-Yusef, R., "The Session Initiation Protocol (SIP) Digest Access Authentication Scheme", RFC 8760, DOI 10.17487/RFC8760, March 2020, <https://www.rfc-editor.org/info/rfc8760>.

[RFC9496] de Valence, H., Grigg, J., Hamburg, M., Lovecruft, I., Tankersley, G., and F. Valsorda, "The ristretto255 and decaf448 Groups", RFC 9496, DOI 10.17487/RFC9496, December 2023, <https://www.rfc-editor.org/info/rfc9496>.

Author’s Address

Maksym Sobolyev
Sippy Software, Inc.
Email: sobomax@sippysoft.com