Internet Engineering Task Force (IETF)
Internet-Draft
Updates: RFC1928
Intended status: Standards Track
Expires: September 16, 2025

T. Lancaster
Independent

March 16, 2025

SOCKS Protocol Version 5a
draft-lancaster-socks-5a-00

Abstract

This document describes SOCKS Protocol Version 5a (SOCKS5a), an evolution of the SOCKS Version 5 protocol [RFC1928] designed to enhance security, authentication flexibility, and resistance to traffic analysis while maintaining complete backward compatibility. SOCKS5a introduces optional extensions that allow existing SOCKS5 clients and servers to interoperate with newer, more secure implementations.

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 September 16, 2025.

Copyright Notice

Table of Contents

1. Introduction 3
2. Requirements Language 4
3. Protocol Overview 4
    3.1. Version Negotiation 5
    3.2. Authentication Framework 6
    3.3. Traffic Obfuscation 7
4. Enhanced Authentication Framework (EAF) 8

4.1. Method Selection 9
4.2. Authentication Methods 10
4.3. Authentication Metadata 11
5. Traffic Obfuscation Layer (TOL) 12
6. Extended Commands 14
7. Extended Address Types 15
8. Reply Codes and Error Handling 16
9. Backward Compatibility 17
10. Security Considerations 18
11. IANA Considerations 19
12. References 20
12.1. Normative References 20
12.2. Informative References 20
Authors' Addresses 21

1. Introduction

The SOCKS protocol [RFC1928] has served as a fundamental component for network access control and firewall traversal since its standardization. This document specifies SOCKS Protocol Version 5a (SOCKS5a), which builds upon SOCKS Version 5 to address modern security requirements while maintaining backward compatibility.

Key improvements in SOCKS5a include:

- Enhanced authentication framework supporting modern authentication methods and multi-factor authentication
- Traffic obfuscation capabilities to resist traffic analysis
- Extended command set for improved operational flexibility
- Additional address types supporting modern deployment scenarios
- Improved error reporting and handling

The primary motivation for these enhancements is to address the evolving security landscape while ensuring that existing SOCKS5 implementations can continue to function without modification.

SOCKS5a maintains complete backward compatibility with SOCKS5 while introducing new capabilities through optional extensions. This approach allows for gradual adoption of the enhanced features without breaking

existing deployments.

This document updates RFC1928 by adding new method codes, address types, and reply codes, while preserving all existing functionality defined in the original specification.

The key design principles of SOCKS5a are:

1. Backward Compatibility
   Existing SOCKS5 clients and servers must continue to work without modification.

2. Optional Enhancement
   All new features are optional and must be explicitly negotiated.

3. Security First
   New features prioritize security and privacy improvements.

4. Extensibility
   The protocol framework allows for future extensions and improvements.

2. Requirements Language

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.

3. Protocol Overview

SOCKS5a extends SOCKS5 by introducing three major components:

1. Enhanced Authentication Framework (EAF)
   Provides support for modern authentication methods, multi-factor authentication, and authentication metadata exchange.

2. Traffic Obfuscation Layer (TOL)
   Adds capabilities to resist traffic analysis and protocol fingerprinting.

3. Extended Command Set
   Introduces new commands for advanced functionality while maintaining compatibility with existing commands.

These components are designed to be optional and backward compatible, allowing gradual adoption while maintaining interoperability with existing SOCKS5 implementations.

The protocol operates in distinct phases:

1. Version and Method Selection
   Compatible with SOCKS5, introduces new method codes.

2. Authentication
   Enhanced framework when using new methods, backward compatible

with existing methods.

3. Request Processing
   Extends SOCKS5 with new commands and address types.

4. Data Transfer
   Adds optional traffic obfuscation capabilities.

3.1. Version Negotiation

The version negotiation process remains compatible with SOCKS5:

```
+----+----------+----------+
|VER | NMETHODS | METHODS  |
+----+----------+----------+
| 1  |    1     | 1 to 255 |
+----+----------+----------+
```

The VER field MUST be X'05' to maintain compatibility with SOCKS5. SOCKS5a introduces new method codes in the range X'80' to X'FE':

X'80' - Enhanced Authentication Framework (EAF)
Enables the use of modern authentication methods and multi-factor authentication.

X'81' - Traffic Obfuscation Layer (TOL)
Activates traffic analysis resistance features.

X'82' - Authentication Method Chaining (AMC)
Allows multiple authentication methods to be used in sequence.

X'83' to X'FE' - Reserved for future extensions

The client sends a list of methods it supports, which may include both traditional SOCKS5 methods and new SOCKS5a methods. The server selects one method from the list by responding with a single byte.

If the server selects a SOCKS5a method (X'80' to X'FE'), all subsequent communication MUST follow the corresponding protocol extension specifications defined in this document.

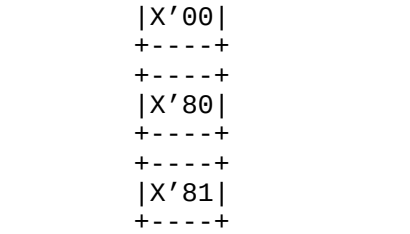
Example version negotiation with SOCKS5a support:

Client request:

```
+----+----------+--------------------+
|X'05|    3     | X'00' X'80' X'81' |
+----+----------+--------------------+
```

Possible server responses:

```
+----+                    (Select NO AUTHENTICATION REQUIRED)
```

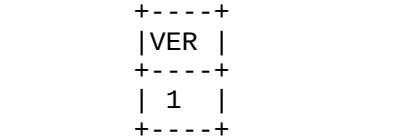


3.2. Authentication Framework

When Enhanced Authentication Framework (EAF) is selected (method X'80'), the authentication process follows an enhanced format that supports:

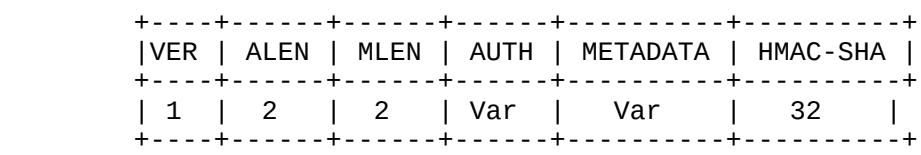
1. Multiple authentication factors
2. Modern cryptographic methods
3. Authentication metadata exchange
4. Forward secrecy

The EAF begins with its own version negotiation:



The VER field indicates the EAF protocol version. This document defines version 1 (X'01'). Future versions of EAF may use different version numbers.

Following version negotiation, authentication data is exchanged using the EAF data format:



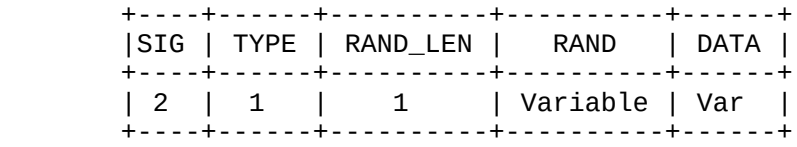
Where:

- VER: Protocol version (X'01' for EAF v1)
- ALEN: Length of the AUTH field in bytes (unsigned 16-bit integer)
- MLEN: Length of the METADATA field in bytes (unsigned 16-bit integer)
- AUTH: Authentication data
- METADATA: Authentication metadata
- HMAC-SHA: HMAC-SHA256 of all preceding fields

The HMAC-SHA field provides integrity protection for the authentication exchange. The key used for HMAC calculation MUST be securely derived during the authentication process.

3.3. Traffic Obfuscation

The Traffic Obfuscation Layer (TOL) provides mechanisms to resist traffic analysis. When method X'81' is selected, all subsequent messages are encapsulated in the TOL format:



SIG	TYPE	RAND_LEN	RAND	DATA
2	1	1	Variable	Var

Where:

SIG (2 bytes):
A random-looking but verifiable signature. This field helps identify valid TOL packets while avoiding obvious patterns. The signature SHOULD be derived from a session-specific key.

TYPE (1 byte):
Indicates the obfuscation method applied to the DATA field:

X'00' - No obfuscation (pass-through)
X'01' - Random padding
X'02' - Timing jitter
X'03' - Traffic pattern masking
X'04' - Full obfuscation (all methods)
X'05' to X'FF' - Reserved for future extensions

RAND_LEN (1 byte):
Length of the RAND field (0 to 255 bytes)

RAND (variable):
Random padding data. The content SHOULD be cryptographically random to resist statistical analysis.

DATA (variable):
The encapsulated SOCKS message.

Each obfuscation type provides different protections:

No Obfuscation (X'00'):
Basic encapsulation without additional obfuscation. Used for control messages or when obfuscation is not required.

Random Padding (X'01'):
Adds variable-length random padding to mask true message sizes.

4. Enhanced Authentication Framework (EAF)

The Enhanced Authentication Framework (EAF) provides a flexible foundation for implementing modern authentication mechanisms. It supports:

- Multi-factor authentication
- Public key cryptography
- Challenge-response protocols
- Authentication metadata exchange
- Forward secrecy
- Authentication method chaining

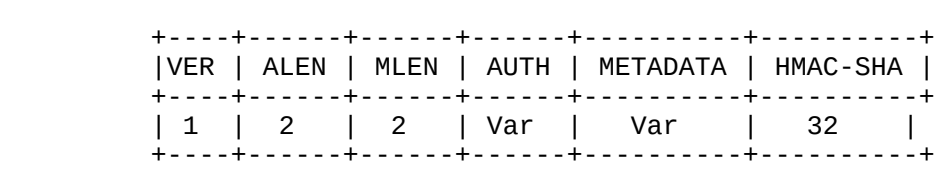
The EAF authentication process consists of three phases:

1. Version Negotiation
   Ensures compatibility between client and server implementations.

2. Authentication Data Exchange
   Transfers authentication credentials and related metadata.

3. Verification
   Confirms the authenticity of both parties.

All EAF messages use the following general format:



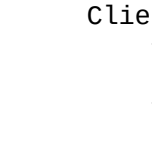
The HMAC-SHA field protects the integrity of the entire message and provides authentication of the sender. The key used for the HMAC calculation MUST be derived using a secure key exchange mechanism or pre-shared key.

Implementations MUST validate the HMAC before processing any authentication data to prevent tampering.

4.1. Method Selection

After selecting the EAF method (X’80’), the authentication process begins with EAF version negotiation:

Client sends:



Server responds:

Both fields MUST contain X'01' for the version defined in this document. If the server does not support the client's version, it SHOULD respond with X'FF' and terminate the connection.

Following successful version negotiation, the client initiates the authentication exchange with an EAF message containing its selected authentication method and any required metadata.

Authentication methods are identified by method codes in the AUTH field:

X'01' - Public Key Authentication
  Uses digital signatures and certificates

X'02' - Challenge-Response
  Server issues challenge, client provides response

X'03' - Time-based One-Time Password (TOTP)
  Compatible with RFC 6238

X'04' - Multi-Factor Combined
  Multiple authentication factors in single exchange

X'05' - Token Authentication
  Bearer token or API key based authentication

X'06' to X'FF' - Reserved for future methods

4.2. Authentication Methods

Each authentication method defines its own structure for the AUTH field. The following sections describe the standard methods:

4.2.1. Public Key Authentication (X'01')

The AUTH field format for public key authentication:

MID	KLEN	SLEN	KEY	SIG
1	2	2	Variable	Variable

Where:

MID - Method identifier (X'01')
KLEN - Length of the KEY field
SLEN - Length of the SIG field
KEY - Public key or certificate
SIG - Digital signature

4.2.2. Challenge-Response (X'02')

Challenge format (server to client):

MID	CLEN	CHALLENGE

```
    +----+------+----------+
    | 1  |  2   | Variable |
    +----+------+----------+
```

Response format (client to server):

```
    +----+------+----------+
    |MID | RLEN | RESPONSE |
    +----+------+----------+
    | 1  |  2   | Variable |
    +----+------+----------+
```

Where:

MID - Method identifier (X’02’)
CLEN - Challenge length
RLEN - Response length

4.3. Authentication Metadata

The METADATA field uses a TLV (Type-Length-Value) format to provide extensible authentication context:

```
    +----+------+----------+
    |TYPE| LEN  |  VALUE   |
    +----+------+----------+
    | 1  |  2   | Variable |
    +----+------+----------+
```

Standard metadata types:

X’01’ - Key Exchange Parameters
    Contains parameters for key exchange protocols.
    Format depends on the chosen key exchange method.

X’02’ - Client Capabilities
    Indicates supported authentication methods and features.
    Bitmap of supported capabilities.

X’03’ - Server Capabilities
    Server’s supported features and requirements.
    Bitmap of supported capabilities.

X’04’ - Authentication Context
    Additional context for the authentication process.
    Method-specific format.

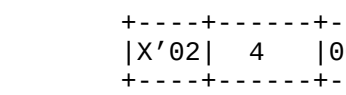
X’05’ - Session Parameters
    Parameters for the authenticated session.
    Includes timeout values, feature flags, etc.

X’06’ - Chain Control
    Controls authentication method chaining.
    Indicates next required authentication method.

Multiple metadata entries MAY be present in a single message. The

total length of all metadata is indicated by the MLEN field in the EAF header.

Example metadata for client capabilities:



X'02'	4	00 00 0F FF

5. Traffic Obfuscation Layer (TOL)

5.1. TOL Header Format

When the TOL method (X'81') is selected, all subsequent messages MUST use the TOL encapsulation format:

SIG	TYPE	RAND_LEN	RAND	DATA
2	1	1	Variable	Var

The SIG field MUST be calculated as:

SIG = HMAC-SHA256-128(SessionKey, TYPE || RAND_LEN || RAND || DATA)

Where:
- SessionKey is derived during authentication
- || denotes concatenation
- Only the first 16 bits of the HMAC are used

5.2. Obfuscation Types

TYPE field values indicate the obfuscation methods:

X'00' - No Obfuscation
Basic encapsulation without additional protections.
Used for control messages or when obfuscation isn't needed.

X'01' - Random Padding
Adds variable-length random padding to mask message sizes.
RAND_LEN and RAND fields contain the padding.

X'02' - Timing Jitter
Introduces randomized delays in message processing.
RAND field contains timing parameters.

X'03' - Traffic Pattern Masking
Modifies traffic patterns to resist analysis.
Includes dummy messages and timing adjustments.

X'04' - Full Obfuscation
Combines all available obfuscation methods.
Maximum protection at the cost of performance.

5.3. Obfuscation Implementations

Each obfuscation type requires specific implementation considerations:

5.3.1. Random Padding (X'01')

When using random padding:

- RAND_LEN MUST be randomly chosen for each message
- RAND content MUST be cryptographically random
- Padding size SHOULD vary independently of message size
- Implementation SHOULD maintain minimum throughput requirements

Example padding distribution:

```
+----------------+-------------------+
| Message Size   | Padding Range     |
+----------------+-------------------+
| 1-128 bytes    | 0-255 bytes       |
| 129-512 bytes  | 0-512 bytes       |
| 513+ bytes     | 0-1024 bytes      |
+----------------+-------------------+
```

5.3.2. Timing Jitter (X'02')

Timing jitter implementation requirements:

- Delay intervals MUST be randomly distributed
- Maximum delay SHOULD be configurable
- Delays SHOULD be applied to both sending and receiving
- Critical messages MAY bypass timing delays

The RAND field contains timing parameters:

```
+----+----+----+----------+
|MIN |MAX |MODE| RESERVED |
+----+----+----+----------+
| 2  | 2  | 1  |    3     |
+----+----+----+----------+
```

Where:
- MIN: Minimum delay in milliseconds
- MAX: Maximum delay in milliseconds
- MODE: Distribution type (0=uniform, 1=normal, 2=exponential)

6. Extended Commands

SOCKS5a introduces new commands while maintaining compatibility with existing SOCKS5 commands. The command format remains unchanged:

```
+----+-----+-------+------+----------+----------+
|VER | CMD |  RSV  | ATYP | DST.ADDR | DST.PORT |
+----+-----+-------+------+----------+----------+
| 1  |  1  | X'00' |  1   | Variable |    2     |
+----+-----+-------+------+----------+----------+
```

New command codes:

X'04' - KEEP_ALIVE
Maintain connection and verify peer availability.
No address required (ATYP = X'00').

X'05' - DNS_RESOLVE
Perform DNS resolution through the SOCKS server.
DST.ADDR contains the hostname to resolve.

X'06' - POOL_NEW
Create a new connection in the connection pool.
Standard address format applies.

X'07' - POOL_GET
Retrieve a connection from the pool.
DST.ADDR contains the pool identifier.

X'08' - UDP_ASSOCIATE_EXT
Enhanced UDP association with additional options.
Extends the original UDP_ASSOCIATE command.

Server replies use the standard SOCKS5 response format with additional reply codes defined in Section 8.

7. Extended Address Types

SOCKS5a adds new address types while maintaining support for existing SOCKS5 address types. Address types are indicated by the ATYP field:

X'01' - IPv4 address (4 bytes)
X'03' - Domain name (Variable)
X'04' - IPv6 address (16 bytes)
X'05' - Unix domain socket path
X'06' - Abstract Unix domain socket
X'07' - URL
X'08' to X'FF' - Reserved for future use

7.1. Unix Domain Socket Path (X'05')

Format for Unix domain socket paths:

```
+----+----------+
|LEN | PATH     |
+----+----------+
| 1  | Variable |
```

```
    +----+----------+
```

- LEN: Length of the path (1-255 bytes)
- PATH: Unix domain socket path

The path MUST be a valid filesystem path to a Unix domain socket.

7.2. Abstract Unix Domain Socket (X'06')

Format for abstract Unix domain sockets:

```
    +----+----------+
    |LEN | NAME     |
    +----+----------+
    | 1  | Variable |
    +----+----------+
```

- LEN: Length of the abstract name (1-255 bytes)
- NAME: Abstract socket name

The name MUST NOT contain null bytes except for the leading null that identifies it as an abstract socket name.

7.3. URL Address Type (X'07')

The URL address type allows direct specification of URLs:

```
    +----+------+----------+
    |VER | ULEN | URL      |
    +----+------+----------+
    | 1  |  2   | Variable |
    +----+------+----------+
```

- VER: URL format version (X'01')
- ULEN: URL length (2 bytes)
- URL: The URL string

The URL MUST be properly encoded according to RFC 3986.
Supported schemes include:
- http
- https
- ws
- wss
- ftp
- sftp

Example URL address:

```
    +----+------+---------------------------------+
    |X'01|  23  | https://example.com/path?q=123 |
    +----+------+---------------------------------+
```

7.4. Address Type Compatibility

When communicating with SOCKS5 servers that don't support extended address types, clients SHOULD:

1. Resolve Unix domain socket paths to IP addresses if possible
2. Convert URLs to hostnames using the authority component
3. Fall back to IPv4/IPv6 or domain name address types

Servers MUST return a "Address type not supported" (X'08') reply if they receive an unsupported address type.

8. Reply Codes and Error Handling

SOCKS5a extends the reply codes defined in SOCKS5 while maintaining backward compatibility. New reply codes provide more detailed error information and support for new features.

8.1. Extended Reply Codes

New reply codes (X'09' to X'FF'):

X'09' - Authentication expired
    The authentication credentials have expired.
    Client should re-authenticate.

X'0A' - Rate limit exceeded
    Too many requests or connections.
    Includes rate limit metadata.

X'0B' - Protocol violation
    Invalid message format or sequence.
    Includes error details in response.

X'0C' - Server temporary error
    Server-side temporary failure.
    Client may retry after delay.

X'0D' - Client configuration error
    Client configuration is invalid.
    Includes configuration guidance.

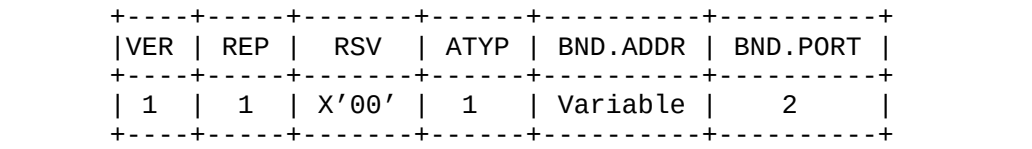
X'0E' - Network configuration error
    Network-related configuration issue.
    Details provided in response.

X'0F' - Permission denied
    Access denied due to permissions.
    May include required permissions.

X'10' - Resource exhausted
    Server resources exhausted.
    Includes resource limits.

8.2. Reply Format

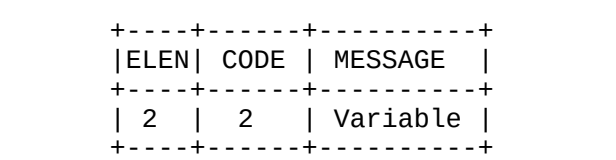
All replies use the standard SOCKS5 reply format:



VER	REP	RSV	ATYP	BND.ADDR	BND.PORT
1	1	X'00'	1	Variable	2

8.3. Error Response Details

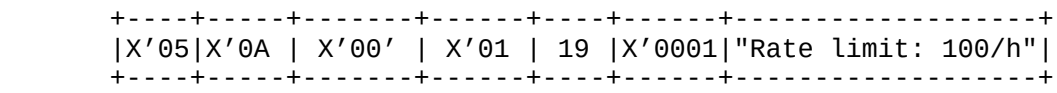
For extended reply codes (X'09' and above), additional error details MAY be provided in the BND.ADDR field using the following format:



ELEN	CODE	MESSAGE
2	2	Variable

Where:
- ELEN: Total length of the error details
- CODE: Detailed error code
- MESSAGE: Human-readable error message

Example error response for rate limit exceeded:



X'05	X'0A	X'00'	X'01	19	X'0001	"Rate limit: 100/h"

9. Backward Compatibility

9.1. Version Negotiation

SOCKS5a maintains full backward compatibility through careful version and method negotiation:

1. Clients MUST start with SOCKS5 version identifier (X'05')
2. New methods are allocated in the reserved range (X'80' to X'FE')
3. Servers MAY choose standard SOCKS5 methods if available
4. Clients MUST support at least one standard SOCKS5 method

9.2. Feature Negotiation

For SOCKS5a-specific features:

1. All extensions are optional
2. Features MUST be explicitly negotiated
3. Fallback to basic functionality MUST be supported
4. Servers MUST handle unknown extensions gracefully

9.3. Address Type Handling

When using extended address types:

1. Clients SHOULD detect server capabilities
2. Fallback to standard types SHOULD be implemented
3. Resolution to supported types MAY be attempted

10. Security Considerations

10.1. Authentication Security

SOCKS5a implementations MUST consider the following authentication security requirements:

1. Strong Authentication Methods
   - Public key cryptography SHOULD be preferred
   - Password authentication SHOULD use secure password hashing
   - Multi-factor authentication SHOULD be supported
   - Authentication tokens MUST have limited lifetimes

2. Authentication Data Protection
   - Credentials MUST be protected in transit
   - Session keys MUST be securely generated
   - Forward secrecy SHOULD be implemented
   - Key material MUST be properly destroyed

3. Implementation Requirements
   - Implementations MUST validate all input
   - Buffer sizes MUST be properly checked
   - Authentication bypass attempts MUST be logged
   - Rate limiting SHOULD be implemented

10.2. Traffic Analysis Resistance

When using the Traffic Obfuscation Layer:

1. Padding Implementation
   - Padding size SHOULD be independent of content
   - Padding MUST use cryptographically secure random data
   - Fixed-size padding SHOULD be avoided
   - Padding distribution SHOULD be configurable

2. Timing Considerations
   - Message timing SHOULD be randomized
   - Fixed delays SHOULD be avoided
   - Processing time SHOULD be normalized
   - Bulk data transfer SHOULD use variable rates

3. Pattern Masking
   - Traffic patterns SHOULD be masked
   - Keep-alive messages SHOULD be randomized
   - Connection pooling SHOULD be used
   - Dummy traffic MAY be generated

4. Implementation Guidelines
   - All random numbers MUST be cryptographically secure
   - State tracking MUST be efficient
   - Resource limits MUST be enforced

11. IANA Considerations

11.1. Method Codes

This document defines new method codes in the range X’80’ to X’FE’. IANA is requested to create a new registry "SOCKS5a Method Codes" with the following initial entries:

X’80’ - Enhanced Authentication Framework (EAF)
X’81’ - Traffic Obfuscation Layer (TOL)
X’82’ - Authentication Method Chaining (AMC)
X’83’ to X’FE’ - Unassigned

11.2. Reply Codes

IANA is requested to create a new registry "SOCKS5a Reply Codes" for reply codes X’09’ through X’FF’ with initial assignments as specified in Section 8.1.

11.3. Address Types

IANA is requested to create a new registry "SOCKS5a Address Types" for address types X’05’ through X’FF’ with initial assignments as specified in Section 7.

12. References

12.1. Normative References

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <https://www.rfc-editor.org/info/rfc1928>.

[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>.

Author’s Address

Torin Lancaster
Anonymous

Email: admin@wikiped.me