Internet Engineering Task Force (IETF)
Request for Comments: 8489
Obsoletes: 5389
Category: Standards Track
ISSN: 2070-1721

M. Petit-Huguenin
Impedance Mismatch
G. Salgueiro
Cisco
J. Rosenberg
Five9
D. Wing
Citrix
R. Mahy
Unaffiliated
P. Matthews
Nokia

February 2020

Session Traversal Utilities for NAT (STUN)

Abstract

Session Traversal Utilities for NAT (STUN) is a protocol that serves as a tool for other protocols in dealing with NAT traversal. It can be used by an endpoint to determine the IP address and port allocated to it by a NAT. It can also be used to check connectivity between two endpoints and as a keep-alive protocol to maintain NAT bindings. STUN works with many existing NATs and does not require any special behavior from them.

STUN is not a NAT traversal solution by itself. Rather, it is a tool to be used in the context of a NAT traversal solution.

This document obsoletes RFC 5389.

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/rfc8489.

Copyright Notice

Copyright (c) 2020 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 Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

The protocol defined in this specification, Session Traversal Utilities for NAT (STUN), provides a tool for dealing with Network Address Translators (NATs). It provides a means for an endpoint to determine the IP address and port allocated by a NAT that corresponds to its private IP address and port. It also provides a way for an endpoint to keep a NAT binding alive. With some extensions, the protocol can be used to do connectivity checks between two endpoints [RFC8445] or to relay packets between two endpoints [RFC5766].

In keeping with its tool nature, this specification defines an extensible packet format, defines operation over several transport protocols, and provides for two forms of authentication.

STUN is intended to be used in the context of one or more NAT traversal solutions. These solutions are known as "STUN Usages". Each usage describes how STUN is utilized to achieve the NAT traversal solution. Typically, a usage indicates when STUN messages get sent, which optional attributes to include, what server is used, and what authentication mechanism is to be used. Interactive Connectivity Establishment (ICE) [RFC8445] is one usage of STUN. SIP Outbound [RFC5626] is another usage of STUN. In some cases, a usage will require extensions to STUN. A STUN extension can be in the form of new methods, attributes, or error response codes. More information on STUN Usages can be found in Section 13.

2. Overview of Operation

This section is descriptive only.

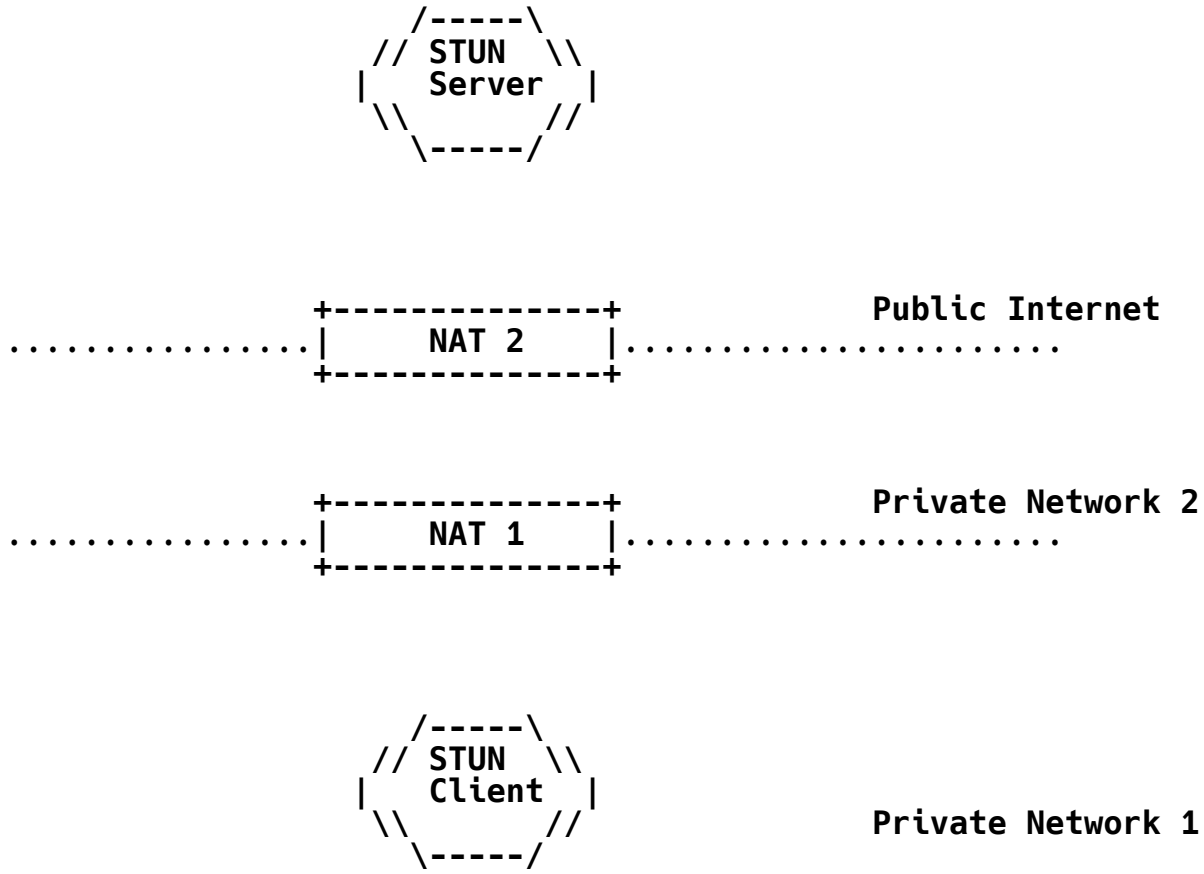

Figure 1: One Possible STUN Configuration

One possible STUN configuration is shown in Figure 1. In this configuration, there are two entities (called STUN agents) that implement the STUN protocol. The lower agent in the figure is the client, which is connected to private network 1. This network connects to private network 2 through NAT 1. Private network 2 connects to the public Internet through NAT 2. The upper agent in the figure is the server, which resides on the public Internet.

STUN is a client-server protocol. It supports two types of transactions. One is a request/response transaction in which a client sends a request to a server, and the server returns a response. The second is an indication transaction in which either agent -- client or server -- sends an indication that generates no response. Both types of transactions include a transaction ID, which

is a randomly selected 96-bit number. For request/response transactions, this transaction ID allows the client to associate the response with the request that generated it; for indications, the transaction ID serves as a debugging aid.

All STUN messages start with a fixed header that includes a method, a class, and the transaction ID. The method indicates which of the various requests or indications this is; this specification defines just one method, Binding, but other methods are expected to be defined in other documents. The class indicates whether this is a request, a success response, an error response, or an indication. Following the fixed header comes zero or more attributes, which are Type-Length-Value extensions that convey additional information for the specific message.

This document defines a single method called "Binding". The Binding method can be used either in request/response transactions or in indication transactions. When used in request/response transactions, the Binding method can be used to determine the particular binding a NAT has allocated to a STUN client. When used in either request/response or in indication transactions, the Binding method can also be used to keep these bindings alive.

In the Binding request/response transaction, a Binding request is sent from a STUN client to a STUN server. When the Binding request arrives at the STUN server, it may have passed through one or more NATs between the STUN client and the STUN server (in Figure 1, there are two such NATs). As the Binding request message passes through a NAT, the NAT will modify the source transport address (that is, the source IP address and the source port) of the packet. As a result, the source transport address of the request received by the server will be the public IP address and port created by the NAT closest to the server. This is called a "reflexive transport address". The STUN server copies that source transport address into an XOR-MAPPED-ADDRESS attribute in the STUN Binding response and sends the Binding response back to the STUN client. As this packet passes back through a NAT, the NAT will modify the destination transport address in the IP header, but the transport address in the XOR-MAPPED-ADDRESS attribute within the body of the STUN response will remain untouched. In this way, the client can learn its reflexive transport address allocated by the outermost NAT with respect to the STUN server.

In some usages, STUN must be multiplexed with other protocols (e.g., [RFC8445] and [RFC5626]). In these usages, there must be a way to inspect a packet and determine if it is a STUN packet or not. STUN provides three fields in the STUN header with fixed values that can

be used for this purpose. If this is not sufficient, then STUN packets can also contain a FINGERPRINT value, which can further be used to distinguish the packets.

STUN defines a set of optional procedures that a usage can decide to use, called "mechanisms". These mechanisms include DNS discovery, a redirection technique to an alternate server, a fingerprint attribute for demultiplexing, and two authentication and message-integrity exchanges. The authentication mechanisms revolve around the use of a username, password, and message-integrity value. Two authentication mechanisms, the long-term credential mechanism and the short-term credential mechanism, are defined in this specification. Each usage specifies the mechanisms allowed with that usage.

In the long-term credential mechanism, the client and server share a pre-provisioned username and password and perform a digest challenge/response exchange inspired by the one defined for HTTP [RFC7616] but differing in details. In the short-term credential mechanism, the client and the server exchange a username and password through some out-of-band method prior to the STUN exchange. For example, in the ICE usage [RFC8445], the two endpoints use out-of-band signaling to exchange a username and password. These are used to integrity protect and authenticate the request and response. There is no challenge or nonce used.

3. 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.

4. Definitions

STUN Agent: A STUN agent is an entity that implements the STUN protocol. The entity can be either a STUN client or a STUN server.

STUN Client: A STUN client is an entity that sends STUN requests and receives STUN responses and STUN indications. A STUN client can also send indications. In this specification, the terms "STUN client" and "client" are synonymous.

STUN Server: A STUN server is an entity that receives STUN requests and STUN indications and that sends STUN responses. A STUN server can also send indications. In this specification, the terms "STUN server" and "server" are synonymous.

Transport Address: The combination of an IP address and port number (such as a UDP or TCP port number).

Reflexive Transport Address: A transport address learned by a client that identifies that client as seen by another host on an IP network, typically a STUN server. When there is an intervening NAT between the client and the other host, the reflexive transport address represents the mapped address allocated to the client on the public side of the NAT. Reflexive transport addresses are learned from the mapped address attribute (MAPPED-ADDRESS or XOR-MAPPED-ADDRESS) in STUN responses.

Mapped Address: Same meaning as reflexive address. This term is retained only for historic reasons and due to the naming of the MAPPED-ADDRESS and XOR-MAPPED-ADDRESS attributes.

Long-Term Credential: A username and associated password that represent a shared secret between client and server. Long-term credentials are generally granted to the client when a subscriber enrolls in a service and persist until the subscriber leaves the service or explicitly changes the credential.

Long-Term Password: The password from a long-term credential.

Short-Term Credential: A temporary username and associated password that represent a shared secret between client and server. Short-term credentials are obtained through some kind of protocol mechanism between the client and server, preceding the STUN exchange. A short-term credential has an explicit temporal scope, which may be based on a specific amount of time (such as 5 minutes) or on an event (such as termination of a Session Initiation Protocol (SIP) [RFC3261] dialog). The specific scope of a short-term credential is defined by the application usage.

Short-Term Password: The password component of a short-term credential.

STUN Indication: A STUN message that does not receive a response.

Attribute: The STUN term for a Type-Length-Value (TLV) object that can be added to a STUN message. Attributes are divided into two types: comprehension-required and comprehension-optional. STUN agents can safely ignore comprehension-optional attributes they don't understand but cannot successfully process a message if it contains comprehension-required attributes that are not understood.

RTO: Retransmission TimeOut, which defines the initial period of time between transmission of a request and the first retransmit of that request.

5. STUN Message Structure

STUN messages are encoded in binary using network-oriented format (most significant byte or octet first, also commonly known as big-endian). The transmission order is described in detail in Appendix B of [RFC0791]. Unless otherwise noted, numeric constants are in decimal (base 10).

All STUN messages comprise a 20-byte header followed by zero or more attributes. The STUN header contains a STUN message type, message length, magic cookie, and transaction ID.

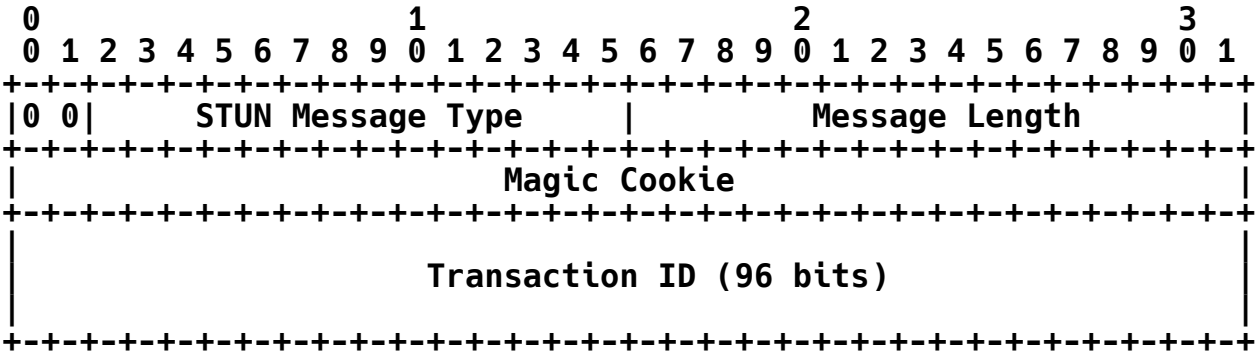

Figure 2: Format of STUN Message Header

The most significant 2 bits of every STUN message MUST be zeroes. This can be used to differentiate STUN packets from other protocols when STUN is multiplexed with other protocols on the same port.

The message type defines the message class (request, success response, error response, or indication) and the message method (the primary function) of the STUN message. Although there are four message classes, there are only two types of transactions in STUN: request/response transactions (which consist of a request message and a response message) and indication transactions (which consist of a single indication message). Response classes are split into error and success responses to aid in quickly processing the STUN message.

The STUN Message Type field is decomposed further into the following structure:

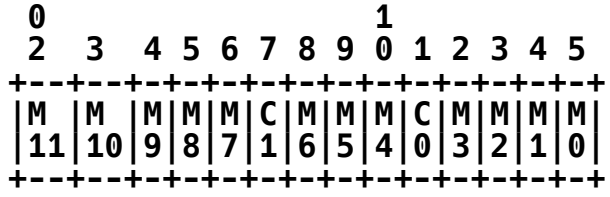

Figure 3: Format of STUN Message Type Field

Here the bits in the STUN Message Type field are shown as most significant (M11) through least significant (M0). M11 through M0 represent a 12-bit encoding of the method. C1 and C0 represent a 2-bit encoding of the class. A class of 0b00 is a request, a class of 0b01 is an indication, a class of 0b10 is a success response, and a class of 0b11 is an error response. This specification defines a single method, Binding. The method and class are orthogonal, so that for each method, a request, success response, error response, and indication are possible for that method. Extensions defining new methods MUST indicate which classes are permitted for that method.

For example, a Binding request has class=0b00 (request) and method=0b000000000001 (Binding) and is encoded into the first 16 bits as 0x0001. A Binding response has class=0b10 (success response) and method=0b000000000001 and is encoded into the first 16 bits as 0x0101.

> Note: This unfortunate encoding is due to assignment of values in [RFC3489] that did not consider encoding indication messages, success responses, and errors responses using bit fields.

The Magic Cookie field MUST contain the fixed value 0x2112A442 in network byte order. In [RFC3489], the 32 bits comprising the Magic Cookie field were part of the transaction ID; placing the magic cookie in this location allows a server to detect if the client will understand certain attributes that were added to STUN by [RFC5389]. In addition, it aids in distinguishing STUN packets from packets of other protocols when STUN is multiplexed with those other protocols on the same port.

The transaction ID is a 96-bit identifier, used to uniquely identify STUN transactions. For request/response transactions, the transaction ID is chosen by the STUN client for the request and echoed by the server in the response. For indications, it is chosen by the agent sending the indication. It primarily serves to correlate requests with responses, though it also plays a small role

in helping to prevent certain types of attacks. The server also uses the transaction ID as a key to identify each transaction uniquely across all clients. As such, the transaction ID MUST be uniformly and randomly chosen from the interval 0 .. 2**96-1 and MUST be cryptographically random. Resends of the same request reuse the same transaction ID, but the client MUST choose a new transaction ID for new transactions unless the new request is bit-wise identical to the previous request and sent from the same transport address to the same IP address. Success and error responses MUST carry the same transaction ID as their corresponding request. When an agent is acting as a STUN server and STUN client on the same port, the transaction IDs in requests sent by the agent have no relationship to the transaction IDs in requests received by the agent.

The message length MUST contain the size of the message in bytes, not including the 20-byte STUN header. Since all STUN attributes are padded to a multiple of 4 bytes, the last 2 bits of this field are always zero. This provides another way to distinguish STUN packets from packets of other protocols.

Following the STUN fixed portion of the header are zero or more attributes. Each attribute is TLV (Type-Length-Value) encoded. Details of the encoding and the attributes themselves are given in Section 14.

6. Base Protocol Procedures

This section defines the base procedures of the STUN protocol. It describes how messages are formed, how they are sent, and how they are processed when they are received. It also defines the detailed processing of the Binding method. Other sections in this document describe optional procedures that a usage may elect to use in certain situations. Other documents may define other extensions to STUN, by adding new methods, new attributes, or new error response codes.

6.1. Forming a Request or an Indication

When formulating a request or indication message, the agent MUST follow the rules in Section 5 when creating the header. In addition, the message class MUST be either "Request" or "Indication" (as appropriate), and the method must be either Binding or some method defined in another document.

The agent then adds any attributes specified by the method or the usage. For example, some usages may specify that the agent use an authentication method (Section 9) or the FINGERPRINT attribute (Section 7).

If the agent is sending a request, it SHOULD add a SOFTWARE attribute to the request. Agents MAY include a SOFTWARE attribute in indications, depending on the method. Extensions to STUN should discuss whether SOFTWARE is useful in new indications. Note that the inclusion of a SOFTWARE attribute may have security implications; see Section 16.1.2 for details.

For the Binding method with no authentication, no attributes are required unless the usage specifies otherwise.

All STUN messages sent over UDP or DTLS-over-UDP [RFC6347] SHOULD be less than the path MTU, if known.

If the path MTU is unknown for UDP, messages SHOULD be the smaller of 576 bytes and the first-hop MTU for IPv4 [RFC1122] and 1280 bytes for IPv6 [RFC8200]. This value corresponds to the overall size of the IP packet. Consequently, for IPv4, the actual STUN message would need to be less than 548 bytes (576 minus 20-byte IP header, minus 8-byte UDP header, assuming no IP options are used).

If the path MTU is unknown for DTLS-over-UDP, the rules described in the previous paragraph need to be adjusted to take into account the size of the (13-byte) DTLS Record header, the Message Authentication Code (MAC) size, and the padding size.

STUN provides no ability to handle the case where the request is smaller than the MTU but the response is larger than the MTU. It is not envisioned that this limitation will be an issue for STUN. The MTU limitation is a SHOULD, not a MUST, to account for cases where STUN itself is being used to probe for MTU characteristics [RFC5780]. See also [STUN-PMTUD] for a framework that uses STUN to add Path MTU Discovery to protocols that lack such a mechanism. Outside of this or similar applications, the MTU constraint MUST be followed.

6.2. Sending the Request or Indication

The agent then sends the request or indication. This document specifies how to send STUN messages over UDP, TCP, TLS-over-TCP, or DTLS-over-UDP; other transport protocols may be added in the future. The STUN Usage must specify which transport protocol is used and how the agent determines the IP address and port of the recipient. Section 8 describes a DNS-based method of determining the IP address and port of a server that a usage may elect to use.

At any time, a client MAY have multiple outstanding STUN requests with the same STUN server (that is, multiple transactions in progress, with different transaction IDs). Absent other limits to

the rate of new transactions (such as those specified by ICE for connectivity checks or when STUN is run over TCP), a client SHOULD limit itself to ten outstanding transactions to the same server.

6.2.1. Sending over UDP or DTLS-over-UDP

When running STUN over UDP or STUN over DTLS-over-UDP [RFC7350], it is possible that the STUN message might be dropped by the network. Reliability of STUN request/response transactions is accomplished through retransmissions of the request message by the client application itself. STUN indications are not retransmitted; thus, indication transactions over UDP or DTLS-over-UDP are not reliable.

A client SHOULD retransmit a STUN request message starting with an interval of RTO ("Retransmission TimeOut"), doubling after each retransmission. The RTO is an estimate of the round-trip time (RTT) and is computed as described in [RFC6298], with two exceptions. First, the initial value for RTO SHOULD be greater than or equal to 500 ms. The exception cases for this "SHOULD" are when other mechanisms are used to derive congestion thresholds (such as the ones defined in ICE for fixed-rate streams) or when STUN is used in non-Internet environments with known network capacities. In fixed-line access links, a value of 500 ms is RECOMMENDED. Second, the value of RTO SHOULD NOT be rounded up to the nearest second. Rather, a 1 ms accuracy SHOULD be maintained. As with TCP, the usage of Karn's algorithm is RECOMMENDED [KARN87]. When applied to STUN, it means that RTT estimates SHOULD NOT be computed from STUN transactions that result in the retransmission of a request.

The value for RTO SHOULD be cached by a client after the completion of the transaction and used as the starting value for RTO for the next transaction to the same server (based on equality of IP address). The value SHOULD be considered stale and discarded if no transactions have occurred to the same server in the last 10 minutes.

Retransmissions continue until a response is received or until a total of Rc requests have been sent. Rc SHOULD be configurable and SHOULD have a default of 7. If, after the last request, a duration equal to Rm times the RTO has passed without a response (providing ample time to get a response if only this final request actually succeeds), the client SHOULD consider the transaction to have failed. Rm SHOULD be configurable and SHOULD have a default of 16. A STUN transaction over UDP or DTLS-over-UDP is also considered failed if there has been a hard ICMP error [RFC1122]. For example, assuming an RTO of 500 ms, requests would be sent at times 0 ms, 500 ms, 1500 ms, 3500 ms, 7500 ms, 15500 ms, and 31500 ms. If the client has not received a response after 39500 ms, the client will consider the transaction to have timed out.

6.2.2. Sending over TCP or TLS-over-TCP

For TCP and TLS-over-TCP [RFC8446], the client opens a TCP connection to the server.

In some usages of STUN, STUN is the only protocol over the TCP connection. In this case, it can be sent without the aid of any additional framing or demultiplexing. In other usages, or with other extensions, it may be multiplexed with other data over a TCP connection. In that case, STUN MUST be run on top of some kind of framing protocol, specified by the usage or extension, which allows for the agent to extract complete STUN messages and complete application-layer messages. The STUN service running on the well-known port or ports discovered through the DNS procedures in Section 8 is for STUN alone, and not for STUN multiplexed with other data. Consequently, no framing protocols are used in connections to those servers. When additional framing is utilized, the usage will specify how the client knows to apply it and what port to connect to. For example, in the case of ICE connectivity checks, this information is learned through out-of-band negotiation between client and server.

Reliability of STUN over TCP and TLS-over-TCP is handled by TCP itself, and there are no retransmissions at the STUN protocol level. However, for a request/response transaction, if the client has not received a response by Ti seconds after it sent the request message, it considers the transaction to have timed out. Ti SHOULD be configurable and SHOULD have a default of 39.5 s. This value has been chosen to equalize the TCP and UDP timeouts for the default initial RTO.

In addition, if the client is unable to establish the TCP connection, or the TCP connection is reset or fails before a response is received, any request/response transaction in progress is considered to have failed.

The client MAY send multiple transactions over a single TCP (or TLS-over-TCP) connection, and it MAY send another request before receiving a response to the previous request. The client SHOULD keep the connection open until it:

- has no further STUN requests or indications to send over that connection,
- has no plans to use any resources (such as a mapped address (MAPPED-ADDRESS or XOR-MAPPED-ADDRESS) or relayed address [RFC5766]) that were learned though STUN requests sent over that connection,

- if multiplexing other application protocols over that port, has finished using those other protocols,
- if using that learned port with a remote peer, has established communications with that remote peer, as is required by some TCP NAT traversal techniques (e.g., [RFC6544]).

The details of an eventual keep-alive mechanism are left to each STUN Usage. In any case, if a transaction fails because an idle TCP connection doesn't work anymore, the client SHOULD send a RST and try to open a new TCP connection.

At the server end, the server SHOULD keep the connection open and let the client close it, unless the server has determined that the connection has timed out (for example, due to the client disconnecting from the network). Bindings learned by the client will remain valid in intervening NATs only while the connection remains open. Only the client knows how long it needs the binding. The server SHOULD NOT close a connection if a request was received over that connection for which a response was not sent. A server MUST NOT ever open a connection back towards the client in order to send a response. Servers SHOULD follow best practices regarding connection management in cases of overload.

6.2.3. Sending over TLS-over-TCP or DTLS-over-UDP

When STUN is run by itself over TLS-over-TCP or DTLS-over-UDP, the TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 and TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 ciphersuites MUST be implemented (for compatibility with older versions of this protocol), except if deprecated by rules of a specific STUN usage. Other ciphersuites MAY be implemented. Note that STUN clients and servers that implement TLS version 1.3 [RFC8446] or subsequent versions are also required to implement mandatory ciphersuites from those specifications and SHOULD disable usage of deprecated ciphersuites when they detect support for those specifications. Perfect Forward Secrecy (PFS) ciphersuites MUST be preferred over non-PFS ciphersuites. Ciphersuites with known weaknesses, such as those based on (single) DES and RC4, MUST NOT be used. Implementations MUST disable TLS-level compression.

These recommendations are just a part of the recommendations in [BCP195] that implementations and deployments of a STUN Usage using TLS or DTLS MUST follow.

When it receives the TLS Certificate message, the client MUST verify the certificate and inspect the site identified by the certificate. If the certificate is invalid or revoked, or if it does not identify

the appropriate party, the client MUST NOT send the STUN message or otherwise proceed with the STUN transaction. The client MUST verify the identity of the server. To do that, it follows the identification procedures defined in [RFC6125], with a certificate containing an identifier of type DNS-ID or CN-ID, optionally with a wildcard character as the leftmost label, but not of type SRV-ID or URI-ID.

When STUN is run multiplexed with other protocols over a TLS-over-TCP connection or a DTLS-over-UDP association, the mandatory ciphersuites and TLS handling procedures operate as defined by those protocols.

6.3. Receiving a STUN Message

This section specifies the processing of a STUN message. The processing specified here is for STUN messages as defined in this specification; additional rules for backwards compatibility are defined in Section 11. Those additional procedures are optional, and usages can elect to utilize them. First, a set of processing operations is applied that is independent of the class. This is followed by class-specific processing, described in the subsections that follow.

When a STUN agent receives a STUN message, it first checks that the message obeys the rules of Section 5. It checks that the first two bits are 0, that the Magic Cookie field has the correct value, that the message length is sensible, and that the method value is a supported method. It checks that the message class is allowed for the particular method. If the message class is "Success Response" or "Error Response", the agent checks that the transaction ID matches a transaction that is still in progress. If the FINGERPRINT extension is being used, the agent checks that the FINGERPRINT attribute is present and contains the correct value. If any errors are detected, the message is silently discarded. In the case when STUN is being multiplexed with another protocol, an error may indicate that this is not really a STUN message; in this case, the agent should try to parse the message as a different protocol.

The STUN agent then does any checks that are required by a authentication mechanism that the usage has specified (see Section 9).

Once the authentication checks are done, the STUN agent checks for unknown attributes and known-but-unexpected attributes in the message. Unknown comprehension-optional attributes MUST be ignored by the agent. Known-but-unexpected attributes SHOULD be ignored by the agent. Unknown comprehension-required attributes cause processing that depends on the message class and is described below.

At this point, further processing depends on the message class of the request.

6.3.1. Processing a Request

If the request contains one or more unknown comprehension-required attributes, the server replies with an error response with an error code of 420 (Unknown Attribute) and includes an UNKNOWN-ATTRIBUTES attribute in the response that lists the unknown comprehension-required attributes.

Otherwise, the server then does any additional checking that the method or the specific usage requires. If all the checks succeed, the server formulates a success response as described below.

When run over UDP or DTLS-over-UDP, a request received by the server could be the first request of a transaction or could be a retransmission. The server MUST respond to retransmissions such that the following property is preserved: if the client receives the response to the retransmission and not the response that was sent to the original request, the overall state on the client and server is identical to the case where only the response to the original retransmission is received or where both responses are received (in which case the client will use the first). The easiest way to meet this requirement is for the server to remember all transaction IDs received over UDP or DTLS-over-UDP and their corresponding responses in the last 40 seconds. However, this requires the server to hold state and is inappropriate for any requests that are not authenticated. Another way is to reprocess the request and recompute the response. The latter technique MUST only be applied to requests that are idempotent (a request is considered idempotent when the same request can be safely repeated without impacting the overall state of the system) and result in the same success response for the same request. The Binding method is considered to be idempotent. Note that there are certain rare network events that could cause the reflexive transport address value to change, resulting in a different mapped address in different success responses. Extensions to STUN MUST discuss the implications of request retransmissions on servers that do not store transaction state.

6.3.1.1. Forming a Success or Error Response

When forming the response (success or error), the server follows the rules of Section 6. The method of the response is the same as that of the request, and the message class is either "Success Response" or "Error Response".

For an error response, the server MUST add an ERROR-CODE attribute containing the error code specified in the processing above. The reason phrase is not fixed but SHOULD be something suitable for the error code. For certain errors, additional attributes are added to the message. These attributes are spelled out in the description where the error code is specified. For example, for an error code of 420 (Unknown Attribute), the server MUST include an UNKNOWN-ATTRIBUTES attribute. Certain authentication errors also cause attributes to be added (see Section 9). Extensions may define other errors and/or additional attributes to add in error cases.

If the server authenticated the request using an authentication mechanism, then the server SHOULD add the appropriate authentication attributes to the response (see Section 9).

The server also adds any attributes required by the specific method or usage. In addition, the server SHOULD add a SOFTWARE attribute to the message.

For the Binding method, no additional checking is required unless the usage specifies otherwise. When forming the success response, the server adds an XOR-MAPPED-ADDRESS attribute to the response; this attribute contains the source transport address of the request message. For UDP or DTLS-over-UDP, this is the source IP address and source UDP port of the request message. For TCP and TLS-over-TCP, this is the source IP address and source TCP port of the TCP connection as seen by the server.

6.3.1.2. Sending the Success or Error Response

The response (success or error) is sent over the same transport as the request was received on. If the request was received over UDP or DTLS-over-UDP, the destination IP address and port of the response are the source IP address and port of the received request message, and the source IP address and port of the response are equal to the destination IP address and port of the received request message. If the request was received over TCP or TLS-over-TCP, the response is sent back on the same TCP connection as the request was received on.

The server is allowed to send responses in a different order than it received the requests.

6.3.2. Processing an Indication

If the indication contains unknown comprehension-required attributes, the indication is discarded and processing ceases.

Otherwise, the agent then does any additional checking that the method or the specific usage requires. If all the checks succeed, the agent then processes the indication. No response is generated for an indication.

For the Binding method, no additional checking or processing is required, unless the usage specifies otherwise. The mere receipt of the message by the agent has refreshed the bindings in the intervening NATs.

Since indications are not re-transmitted over UDP or DTLS-over-UDP (unlike requests), there is no need to handle re-transmissions of indications at the sending agent.

6.3.3. Processing a Success Response

If the success response contains unknown comprehension-required attributes, the response is discarded and the transaction is considered to have failed.

Otherwise, the client then does any additional checking that the method or the specific usage requires. If all the checks succeed, the client then processes the success response.

For the Binding method, the client checks that the XOR-MAPPED-ADDRESS attribute is present in the response. The client checks the address family specified. If it is an unsupported address family, the attribute SHOULD be ignored. If it is an unexpected but supported address family (for example, the Binding transaction was sent over IPv4, but the address family specified is IPv6), then the client MAY accept and use the value.

6.3.4. Processing an Error Response

If the error response contains unknown comprehension-required attributes, or if the error response does not contain an ERROR-CODE attribute, then the transaction is simply considered to have failed.

Otherwise, the client then does any processing specified by the authentication mechanism (see Section 9). This may result in a new transaction attempt.

The processing at this point depends on the error code, the method, and the usage; the following are the default rules:

- If the error code is 300 through 399, the client SHOULD consider the transaction as failed unless the ALTERNATE-SERVER extension (Section 10) is being used.

- If the error code is 400 through 499, the client declares the transaction failed; in the case of 420 (Unknown Attribute), the response should contain a UNKNOWN-ATTRIBUTES attribute that gives additional information.

- If the error code is 500 through 599, the client MAY resend the request; clients that do so MUST limit the number of times they do this. Unless a specific error code specifies a different value, the number of retransmissions SHOULD be limited to 4.

Any other error code causes the client to consider the transaction failed.

7. FINGERPRINT Mechanism

This section describes an optional mechanism for STUN that aids in distinguishing STUN messages from packets of other protocols when the two are multiplexed on the same transport address. This mechanism is optional, and a STUN Usage must describe if and when it is used. The FINGERPRINT mechanism is not backwards compatible with RFC 3489 and cannot be used in environments where such compatibility is required.

In some usages, STUN messages are multiplexed on the same transport address as other protocols, such as the Real-Time Transport Protocol (RTP). In order to apply the processing described in Section 6, STUN messages must first be separated from the application packets.

Section 5 describes three fixed fields in the STUN header that can be used for this purpose. However, in some cases, these three fixed fields may not be sufficient.

When the FINGERPRINT extension is used, an agent includes the FINGERPRINT attribute in messages it sends to another agent. Section 14.7 describes the placement and value of this attribute.

When the agent receives what it believes is a STUN message, then, in addition to other basic checks, the agent also checks that the message contains a FINGERPRINT attribute and that the attribute contains the correct value. Section 6.3 describes when in the overall processing of a STUN message the FINGERPRINT check is performed. This additional check helps the agent detect messages of other protocols that might otherwise seem to be STUN messages.

8. DNS Discovery of a Server

This section describes an optional procedure for STUN that allows a client to use DNS to determine the IP address and port of a server. A STUN Usage must describe if and when this extension is used. To

use this procedure, the client must know a STUN URI [RFC7064]; the usage must also describe how the client obtains this URI. Hard-coding a STUN URI into software is NOT RECOMMENDED in case the domain name is lost or needs to change for legal or other reasons.

When a client wishes to locate a STUN server on the public Internet that accepts Binding request/response transactions, the STUN URI scheme is "stun". When it wishes to locate a STUN server that accepts Binding request/response transactions over a TLS or DTLS session, the URI scheme is "stuns".

The syntax of the "stun" and "stuns" URIs is defined in Section 3.1 of [RFC7064]. STUN Usages MAY define additional URI schemes.

8.1. STUN URI Scheme Semantics

If the <host> part of a "stun" URI contains an IP address, then this IP address is used directly to contact the server. A "stuns" URI containing an IP address MUST be rejected. A future STUN extension or usage may relax this requirement, provided it demonstrates how to authenticate the STUN server and prevent man-in-the-middle attacks.

If the URI does not contain an IP address, the domain name contained in the <host> part is resolved to a transport address using the SRV procedures specified in [RFC2782]. The DNS SRV service name is the content of the <scheme> part. The protocol in the SRV lookup is the transport protocol the client will run STUN over: "udp" for UDP and "tcp" for TCP.

The procedures of RFC 2782 are followed to determine the server to contact. RFC 2782 spells out the details of how a set of SRV records is sorted and then tried. However, RFC 2782 only states that the client should "try to connect to the (protocol, address, service)" without giving any details on what happens in the event of failure. When following these procedures, if the STUN transaction times out without receipt of a response, the client SHOULD retry the request to the next server in the order defined by RFC 2782. Such a retry is only possible for request/response transmissions, since indication transactions generate no response or timeout.

In addition, instead of querying either the A or the AAAA resource records for a domain name, a dual-stack IPv4/IPv6 client MUST query both and try the requests with all the IP addresses received, as specified in [RFC8305].

The default port for STUN requests is 3478, for both TCP and UDP. The default port for STUN over TLS and STUN over DTLS requests is 5349. Servers can run STUN over DTLS on the same port as STUN over

UDP if the server software supports determining whether the initial message is a DTLS or STUN message. Servers can run STUN over TLS on the same port as STUN over TCP if the server software supports determining whether the initial message is a TLS or STUN message.

Administrators of STUN servers SHOULD use these ports in their SRV records for UDP and TCP. In all cases, the port in DNS MUST reflect the one on which the server is listening.

If no SRV records are found, the client performs both an A and AAAA record lookup of the domain name, as described in [RFC8305]. The result will be a list of IP addresses, each of which can be simultaneously contacted at the default port using UDP or TCP, independent of the STUN Usage. For usages that require TLS, the client connects to the IP addresses using the default STUN over TLS port. For usages that require DTLS, the client connects to the IP addresses using the default STUN over DTLS port.

9. Authentication and Message-Integrity Mechanisms

This section defines two mechanisms for STUN that a client and server can use to provide authentication and message integrity; these two mechanisms are known as the short-term credential mechanism and the long-term credential mechanism. These two mechanisms are optional, and each usage must specify if and when these mechanisms are used. Consequently, both clients and servers will know which mechanism (if any) to follow based on knowledge of which usage applies. For example, a STUN server on the public Internet supporting ICE would have no authentication, whereas the STUN server functionality in an agent supporting connectivity checks would utilize short-term credentials. An overview of these two mechanisms is given in Section 2.

Each mechanism specifies the additional processing required to use that mechanism, extending the processing specified in Section 6. The additional processing occurs in three different places: when forming a message, when receiving a message immediately after the basic checks have been performed, and when doing the detailed processing of error responses.

Note that agents MUST ignore all attributes that follow MESSAGE-INTEGRITY, with the exception of the MESSAGE-INTEGRITY-SHA256 and FINGERPRINT attributes. Similarly, agents MUST ignore all attributes that follow the MESSAGE-INTEGRITY-SHA256 attribute if the MESSAGE-INTEGRITY attribute is not present, with the exception of the FINGERPRINT attribute.

9.1. Short-Term Credential Mechanism

The short-term credential mechanism assumes that, prior to the STUN transaction, the client and server have used some other protocol to exchange a credential in the form of a username and password. This credential is time-limited. The time limit is defined by the usage. As an example, in the ICE usage [RFC8445], the two endpoints use out-of-band signaling to agree on a username and password, and this username and password are applicable for the duration of the media session.

This credential is used to form a message-integrity check in each request and in many responses. There is no challenge and response as in the long-term mechanism; consequently, replay is limited by virtue of the time-limited nature of the credential.

9.1.1. HMAC Key

For short-term credentials, the Hash-Based Message Authentication Code (HMAC) key is defined as follow:

```
key = OpaqueString(password)
```

where the OpaqueString profile is defined in [RFC8265]. The encoding used is UTF-8 [RFC3629].

9.1.2. Forming a Request or Indication

For a request or indication message, the agent MUST include the USERNAME, MESSAGE-INTEGRITY-SHA256, and MESSAGE-INTEGRITY attributes in the message unless the agent knows from an external mechanism which message integrity algorithm is supported by both agents. In this case, either MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 MUST be included in addition to USERNAME. The HMAC for the MESSAGE-INTEGRITY attribute is computed as described in Section 14.5, and the HMAC for the MESSAGE-INTEGRITY-SHA256 attributes is computed as described in Section 14.6. Note that the password is never included in the request or indication.

9.1.3. Receiving a Request or Indication

After the agent has done the basic processing of a message, the agent performs the checks listed below in the order specified:

- If the message does not contain 1) a MESSAGE-INTEGRITY or a MESSAGE-INTEGRITY-SHA256 attribute and 2) a USERNAME attribute:

* If the message is a request, the server MUST reject the request with an error response. This response MUST use an error code of 400 (Bad Request).

* If the message is an indication, the agent MUST silently discard the indication.

- If the USERNAME does not contain a username value currently valid within the server:

  * If the message is a request, the server MUST reject the request with an error response. This response MUST use an error code of 401 (Unauthenticated).

  * If the message is an indication, the agent MUST silently discard the indication.

- If the MESSAGE-INTEGRITY-SHA256 attribute is present, compute the value for the message integrity as described in Section 14.6, using the password associated with the username. If the MESSAGE-INTEGRITY-SHA256 attribute is not present, then use the same password to compute the value for the message integrity as described in Section 14.5. If the resulting value does not match the contents of the corresponding attribute (MESSAGE-INTEGRITY-SHA256 or MESSAGE-INTEGRITY):

  * If the message is a request, the server MUST reject the request with an error response. This response MUST use an error code of 401 (Unauthenticated).

  * If the message is an indication, the agent MUST silently discard the indication.

If these checks pass, the agent continues to process the request or indication. Any response generated by a server to a request that contains a MESSAGE-INTEGRITY-SHA256 attribute MUST include the MESSAGE-INTEGRITY-SHA256 attribute, computed using the password utilized to authenticate the request. Any response generated by a server to a request that contains only a MESSAGE-INTEGRITY attribute MUST include the MESSAGE-INTEGRITY attribute, computed using the password utilized to authenticate the request. This means that only one of these attributes can appear in a response. The response MUST NOT contain the USERNAME attribute.

If any of the checks fail, a server MUST NOT include a MESSAGE-INTEGRITY-SHA256, MESSAGE-INTEGRITY, or USERNAME attribute in the error response. This is because, in these failure cases, the server cannot determine the shared secret necessary to compute the MESSAGE-INTEGRITY-SHA256 or MESSAGE-INTEGRITY attributes.

9.1.4. Receiving a Response

The client looks for the MESSAGE-INTEGRITY or the MESSAGE-INTEGRITY-SHA256 attribute in the response. If present and if the client only sent one of the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attributes in the request (because of the external indication in Section 9.1.2 or because this is a subsequent request as defined in Section 9.1.5), the algorithm in the response has to match; otherwise, the response MUST be discarded.

The client then computes the message integrity over the response as defined in Section 14.5 for the MESSAGE-INTEGRITY attribute or Section 14.6 for the MESSAGE-INTEGRITY-SHA256 attribute, using the same password it utilized for the request. If the resulting value matches the contents of the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute, respectively, the response is considered authenticated. If the value does not match, or if both MESSAGE-INTEGRITY and MESSAGE-INTEGRITY-SHA256 are absent, the processing depends on whether the request was sent over a reliable or an unreliable transport.

If the request was sent over an unreliable transport, the response MUST be discarded, as if it had never been received. This means that retransmits, if applicable, will continue. If all the responses received are discarded, then instead of signaling a timeout after ending the transaction, the layer MUST signal that the integrity protection was violated.

If the request was sent over a reliable transport, the response MUST be discarded, and the layer MUST immediately end the transaction and signal that the integrity protection was violated.

9.1.5. Sending Subsequent Requests

A client sending subsequent requests to the same server MUST send only the MESSAGE-INTEGRITY-SHA256 or the MESSAGE-INTEGRITY attribute that matches the attribute that was received in the response to the initial request. Here, "same server" means same IP address and port number, not just the same URI or SRV lookup result.

9.2. Long-Term Credential Mechanism

The long-term credential mechanism relies on a long-term credential, in the form of a username and password that are shared between client and server. The credential is considered long-term since it is assumed that it is provisioned for a user and remains in effect until the user is no longer a subscriber of the system or until it is changed. This is basically a traditional "log-in" username and password given to users.

Because these usernames and passwords are expected to be valid for extended periods of time, replay prevention is provided in the form of a digest challenge. In this mechanism, the client initially sends a request, without offering any credentials or any integrity checks. The server rejects this request, providing the user a realm (used to guide the user or agent in selection of a username and password) and a nonce. The nonce provides a limited replay protection. It is a cookie, selected by the server and encoded in such a way as to indicate a duration of validity or client identity from which it is valid. Only the server needs to know about the internal structure of the cookie. The client retries the request, this time including its username and the realm and echoing the nonce provided by the server. The client also includes one of the message-integrity attributes defined in this document, which provides an HMAC over the entire request, including the nonce. The server validates the nonce and checks the message integrity. If they match, the request is authenticated. If the nonce is no longer valid, it is considered "stale", and the server rejects the request, providing a new nonce.

In subsequent requests to the same server, the client reuses the nonce, username, realm, and password it used previously. In this way, subsequent requests are not rejected until the nonce becomes invalid by the server, in which case the rejection provides a new nonce to the client.

Note that the long-term credential mechanism cannot be used to protect indications, since indications cannot be challenged. Usages utilizing indications must either use a short-term credential or omit authentication and message integrity for them.

To indicate that it supports this specification, a server MUST prepend the NONCE attribute value with the character string composed of "obMatJos2" concatenated with the (4-character) base64 [RFC4648] encoding of the 24-bit STUN Security Features as defined in Section 18.1. The 24-bit Security Feature set is encoded as 3 bytes, with bit 0 as the most significant bit of the first byte and bit 23 as the least significant bit of the third byte. If no security features are used, then a byte array with all 24 bits set to zero

MUST be encoded instead. For the remainder of this document, the term "nonce cookie" will refer to the complete 13-character string prepended to the NONCE attribute value.

Since the long-term credential mechanism is susceptible to offline dictionary attacks, deployments SHOULD utilize passwords that are difficult to guess. In cases where the credentials are not entered by the user, but are rather placed on a client device during device provisioning, the password SHOULD have at least 128 bits of randomness. In cases where the credentials are entered by the user, they should follow best current practices around password structure.

9.2.1. Bid-Down Attack Prevention

This document introduces two new security features that provide the ability to choose the algorithm used for password protection as well as the ability to use an anonymous username. Both of these capabilities are optional in order to remain backwards compatible with previous versions of the STUN protocol.

These new capabilities are subject to bid-down attacks whereby an attacker in the message path can remove these capabilities and force weaker security properties. To prevent these kinds of attacks from going undetected, the nonce is enhanced with additional information.

The value of the "nonce cookie" will vary based on the specific STUN Security Feature bits selected. When this document makes reference to the "nonce cookie" in a section discussing a specific STUN Security Feature it is understood that the corresponding STUN Security Feature bit in the "nonce cookie" is set to 1.

For example, when the PASSWORD-ALGORITHMS security feature (defined in Section 9.2.4) is used, the corresponding "Password algorithms" bit (defined in Section 18.1) is set to 1 in the "nonce cookie".

9.2.2. HMAC Key

For long-term credentials that do not use a different algorithm, as specified by the PASSWORD-ALGORITHM attribute, the key is 16 bytes:

```
key = MD5(username ":" OpaqueString(realm)
  ":" OpaqueString(password))
```

Where MD5 is defined in [RFC1321] and [RFC6151], and the OpaqueString profile is defined in [RFC8265]. The encoding used is UTF-8 [RFC3629].

The 16-byte key is formed by taking the MD5 hash of the result of concatenating the following five fields: (1) the username, with any quotes and trailing nulls removed, as taken from the USERNAME attribute (in which case OpaqueString has already been applied); (2) a single colon; (3) the realm, with any quotes and trailing nulls removed and after processing using OpaqueString; (4) a single colon; and (5) the password, with any trailing nulls removed and after processing using OpaqueString. For example, if the username is 'user', the realm is 'realm', and the password is 'pass', then the 16-byte HMAC key would be the result of performing an MD5 hash on the string 'user:realm:pass', the resulting hash being 0x8493fbc53ba582fb4c044c456bdc40eb.

The structure of the key when used with long-term credentials facilitates deployment in systems that also utilize SIP [RFC3261]. Typically, SIP systems utilizing SIP's digest authentication mechanism do not actually store the password in the database. Rather, they store a value called "H(A1)", which is equal to the key defined above. For example, this mechanism can be used with the authentication extensions defined in [RFC5090].

When a PASSWORD-ALGORITHM is used, the key length and algorithm to use are described in Section 18.5.1.

9.2.3. Forming a Request

The first request from the client to the server (as identified by hostname if the DNS procedures of Section 8 are used and by IP address if not) is handled according to the rules in Section 9.2.3.1. When the client initiates a subsequent request once a previous request/response transaction has completed successfully, it follows the rules in Section 9.2.3.2. Forming a request as a consequence of a 401 (Unauthenticated) or 438 (Stale Nonce) error response is covered in Section 9.2.5 and is not considered a "subsequent request" and thus does not utilize the rules described in Section 9.2.3.2. Each of these types of requests have a different mandatory attributes.

9.2.3.1. First Request

If the client has not completed a successful request/response transaction with the server, it MUST omit the USERNAME, USERHASH, MESSAGE-INTEGRITY, MESSAGE-INTEGRITY-SHA256, REALM, NONCE, PASSWORD-ALGORITHMS, and PASSWORD-ALGORITHM attributes. In other words, the first request is sent as if there were no authentication or message integrity applied.

9.2.3.2. Subsequent Requests

Once a request/response transaction has completed, the client will have been presented a realm and nonce by the server and selected a username and password with which it authenticated. The client SHOULD cache the username, password, realm, and nonce for subsequent communications with the server. When the client sends a subsequent request, it MUST include either the USERNAME or USERHASH, REALM, NONCE, and PASSWORD-ALGORITHM attributes with these cached values. It MUST include a MESSAGE-INTEGRITY attribute or a MESSAGE-INTEGRITY-SHA256 attribute, computed as described in Sections 14.5 and 14.6 using the cached password. The choice between the two attributes depends on the attribute received in the response to the first request.

9.2.4. Receiving a Request

After the server has done the basic processing of a request, it performs the checks listed below in the order specified. Note that it is RECOMMENDED that the REALM value be the domain name of the provider of the STUN server:

- If the message does not contain a MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute, the server MUST generate an error response with an error code of 401 (Unauthenticated). This response MUST include a REALM value. The response MUST include a NONCE, selected by the server. The server MUST NOT choose the same NONCE for two requests unless they have the same source IP address and port. The server MAY support alternate password algorithms, in which case it can list them in preferential order in a PASSWORD-ALGORITHMS attribute. If the server adds a PASSWORD-ALGORITHMS attribute, it MUST set the STUN Security Feature "Password algorithms" bit to 1. The server MAY support anonymous username, in which case it MUST set the STUN Security Feature "Username anonymity" bit set to 1. The response SHOULD NOT contain a USERNAME, USERHASH, MESSAGE-INTEGRITY, or MESSAGE-INTEGRITY-SHA256 attribute.

  Note: Reusing a NONCE for different source IP addresses or ports was not explicitly forbidden in [RFC5389].

- If the message contains a MESSAGE-INTEGRITY or a MESSAGE-INTEGRITY-SHA256 attribute, but is missing either the USERNAME or USERHASH, REALM, or NONCE attribute, the server MUST generate an error response with an error code of 400 (Bad Request). This response SHOULD NOT include a USERNAME, USERHASH, NONCE, or REALM

attribute. The response cannot contain a MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute, as the attributes required to generate them are missing.

- If the NONCE attribute starts with the "nonce cookie" with the STUN Security Feature "Password algorithms" bit set to 1, the server performs these checks in the order specified:
  - If the request contains neither the PASSWORD-ALGORITHMS nor the PASSWORD-ALGORITHM algorithm, then the request is processed as though PASSWORD-ALGORITHM were MD5.
  - Otherwise, unless (1) PASSWORD-ALGORITHM and PASSWORD-ALGORITHMS are both present, (2) PASSWORD-ALGORITHMS matches the value sent in the response that sent this NONCE, and (3) PASSWORD-ALGORITHM matches one of the entries in PASSWORD-ALGORITHMS, the server MUST generate an error response with an error code of 400 (Bad Request).
- If the value of the USERNAME or USERHASH attribute is not valid, the server MUST generate an error response with an error code of 401 (Unauthenticated). This response MUST include a REALM value. The response MUST include a NONCE, selected by the server. The response MUST include a PASSWORD-ALGORITHMS attribute. The response SHOULD NOT contain a USERNAME or USERHASH attribute. The response MAY include a MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute, using the previous key to calculate it.
- If the MESSAGE-INTEGRITY-SHA256 attribute is present, compute the value for the message integrity as described in Section 14.6, using the password associated with the username. Otherwise, using the same password, compute the value for the MESSAGE-INTEGRITY attribute as described in Section 14.5. If the resulting value does not match the contents of the MESSAGE-INTEGRITY attribute or the MESSAGE-INTEGRITY-SHA256 attribute, the server MUST reject the request with an error response. This response MUST use an error code of 401 (Unauthenticated). It MUST include the REALM and NONCE attributes and SHOULD NOT include the USERNAME, USERHASH, MESSAGE-INTEGRITY, or MESSAGE-INTEGRITY-SHA256 attribute.
- If the NONCE is no longer valid, the server MUST generate an error response with an error code of 438 (Stale Nonce). This response MUST include NONCE, REALM, and PASSWORD-ALGORITHMS attributes and SHOULD NOT include the USERNAME and USERHASH attributes. The NONCE attribute value MUST be valid. The response MAY include a MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute, using the

previous NONCE to calculate it. Servers can revoke nonces in order to provide additional security. See Section 5.4 of [RFC7616] for guidelines.

If these checks pass, the server continues to process the request. Any response generated by the server MUST include the MESSAGE-INTEGRITY-SHA256 attribute, computed using the username and password utilized to authenticate the request, unless the request was processed as though PASSWORD-ALGORITHM was MD5 (because the request contained neither PASSWORD-ALGORITHMS nor PASSWORD-ALGORITHM). In that case, the MESSAGE-INTEGRITY attribute MUST be used instead of the MESSAGE-INTEGRITY-SHA256 attribute, and the REALM, NONCE, USERNAME, and USERHASH attributes SHOULD NOT be included.

9.2.5. Receiving a Response

If the response is an error response with an error code of 401 (Unauthenticated) or 438 (Stale Nonce), the client MUST test if the NONCE attribute value starts with the "nonce cookie". If so and the "nonce cookie" has the STUN Security Feature "Password algorithms" bit set to 1 but no PASSWORD-ALGORITHMS attribute is present, then the client MUST NOT retry the request with a new transaction.

If the response is an error response with an error code of 401 (Unauthenticated), the client SHOULD retry the request with a new transaction. This request MUST contain a USERNAME or a USERHASH, determined by the client as the appropriate username for the REALM from the error response. If the "nonce cookie" is present and has the STUN Security Feature "Username anonymity" bit set to 1, then the USERHASH attribute MUST be used; else, the USERNAME attribute MUST be used. The request MUST contain the REALM, copied from the error response. The request MUST contain the NONCE, copied from the error response. If the response contains a PASSWORD-ALGORITHMS attribute, the request MUST contain the PASSWORD-ALGORITHMS attribute with the same content. If the response contains a PASSWORD-ALGORITHMS attribute, and this attribute contains at least one algorithm that is supported by the client, then the request MUST contain a PASSWORD-ALGORITHM attribute with the first algorithm supported on the list. If the response contains a PASSWORD-ALGORITHMS attribute, and this attribute does not contain any algorithm that is supported by the client, then the client MUST NOT retry the request with a new transaction. The client MUST NOT perform this retry if it is not changing the USERNAME, USERHASH, REALM, or its associated password from the previous attempt.

If the response is an error response with an error code of 438 (Stale Nonce), the client MUST retry the request, using the new NONCE attribute supplied in the 438 (Stale Nonce) response. This retry MUST also include either the USERNAME or USERHASH, the REALM, and either the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute.

For all other responses, if the NONCE attribute starts with the "nonce cookie" with the STUN Security Feature "Password algorithms" bit set to 1 but PASSWORD-ALGORITHMS is not present, the response MUST be ignored.

If the response is an error response with an error code of 400 (Bad Request) and does not contain either the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute, then the response MUST be discarded, as if it were never received. This means that retransmits, if applicable, will continue.

> Note: In this case, the 400 response will never reach the application, resulting in a timeout.

The client looks for the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute in the response (either success or failure). If present, the client computes the message integrity over the response as defined in Sections 14.5 or 14.6, using the same password it utilized for the request. If the resulting value matches the contents of the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute, the response is considered authenticated. If the value does not match, or if both MESSAGE-INTEGRITY and MESSAGE-INTEGRITY-SHA256 are absent, the processing depends on the request being sent over a reliable or an unreliable transport.

If the request was sent over an unreliable transport, the response MUST be discarded, as if it had never been received. This means that retransmits, if applicable, will continue. If all the responses received are discarded, then instead of signaling a timeout after ending the transaction, the layer MUST signal that the integrity protection was violated.

If the request was sent over a reliable transport, the response MUST be discarded, and the layer MUST immediately end the transaction and signal that the integrity protection was violated.

If the response contains a PASSWORD-ALGORITHMS attribute, all the subsequent requests MUST be authenticated using MESSAGE-INTEGRITY-SHA256 only.

10. ALTERNATE-SERVER Mechanism

This section describes a mechanism in STUN that allows a server to redirect a client to another server. This extension is optional, and a usage must define if and when this extension is used. The ALTERNATE-SERVER attribute carries an IP address.

A server using this extension redirects a client to another server by replying to a request message with an error response message with an error code of 300 (Try Alternate). The server MUST include at least one ALTERNATE-SERVER attribute in the error response, which MUST contain an IP address of the same address family as the source IP address of the request message. The server SHOULD include an additional ALTERNATE-SERVER attribute, after the mandatory one, that contains an IP address of the address family other than the source IP address of the request message. The error response message MAY be authenticated; however, there are use cases for ALTERNATE-SERVER where authentication of the response is not possible or practical. If the transaction uses TLS or DTLS, if the transaction is authenticated by a MESSAGE-INTEGRITY-SHA256 attribute, and if the server wants to redirect to a server that uses a different certificate, then it MUST include an ALTERNATE-DOMAIN attribute containing the name inside the subjectAltName of that certificate. This series of conditions on the MESSAGE-INTEGRITY-SHA256 attribute indicates that the transaction is authenticated and that the client implements this specification and therefore can process the ALTERNATE-DOMAIN attribute.

A client using this extension handles a 300 (Try Alternate) error code as follows. The client looks for an ALTERNATE-SERVER attribute in the error response. If one is found, then the client considers the current transaction as failed and reattempts the request with the server specified in the attribute, using the same transport protocol used for the previous request. That request, if authenticated, MUST utilize the same credentials that the client would have used in the request to the server that performed the redirection. If the transport protocol uses TLS or DTLS, then the client looks for an ALTERNATE-DOMAIN attribute. If the attribute is found, the domain MUST be used to validate the certificate using the recommendations in [RFC6125]. The certificate MUST contain an identifier of type DNS-ID or CN-ID (eventually with wildcards) but not of type SRV-ID or URI-ID. If the attribute is not found, the same domain that was used for the original request MUST be used to validate the certificate. If the client has been redirected to a server to which it has already sent this request within the last five minutes, it MUST ignore the redirection and consider the transaction to have failed. This prevents infinite ping-ponging between servers in case of redirection loops.

11. Backwards Compatibility with RFC 3489

In addition to the backward compatibility already described in Section 12 of [RFC5389], DTLS MUST NOT be used with [RFC3489] (referred to as "classic STUN"). Any STUN request or indication without the magic cookie (see Section 6 of [RFC5389]) over DTLS MUST be considered invalid: all requests MUST generate a 500 (Server Error) error response, and indications MUST be ignored.

12. Basic Server Behavior

This section defines the behavior of a basic, stand-alone STUN server.

Historically, "classic STUN" [RFC3489] only defined the behavior of a server that was providing clients with server reflexive transport addresses by receiving and replying to STUN Binding requests. [RFC5389] redefined the protocol as an extensible framework, and the server functionality became the sole STUN Usage defined in that document. This STUN Usage is also known as "Basic STUN Server".

The STUN server MUST support the Binding method. It SHOULD NOT utilize the short-term or long-term credential mechanism. This is because the work involved in authenticating the request is more than the work in simply processing it. It SHOULD NOT utilize the ALTERNATE-SERVER mechanism for the same reason. It MUST support UDP and TCP. It MAY support STUN over TCP/TLS or STUN over UDP/DTLS; however, DTLS and TLS provide minimal security benefits in this basic mode of operation. It does not require a keep-alive mechanism because a TCP or TLS-over-TCP connection is closed after the end of the Binding transaction. It MAY utilize the FINGERPRINT mechanism but MUST NOT require it. Since the stand-alone server only runs STUN, FINGERPRINT provides no benefit. Requiring it would break compatibility with RFC 3489, and such compatibility is desirable in a stand-alone server. Stand-alone STUN servers SHOULD support backwards compatibility with clients using [RFC3489], as described in Section 11.

It is RECOMMENDED that administrators of STUN servers provide DNS entries for those servers as described in Section 8. If both A and AAAA resource records are returned, then the client can simultaneously send STUN Binding requests to the IPv4 and IPv6 addresses (as specified in [RFC8305]), as the Binding request is idempotent. Note that the MAPPED-ADDRESS or XOR-MAPPED-ADDRESS attributes that are returned will not necessarily match the address family of the server address used.

A basic STUN server is not a solution for NAT traversal by itself. However, it can be utilized as part of a solution through STUN Usages. This is discussed further in Section 13.

13. STUN Usages

STUN by itself is not a solution to the NAT traversal problem. Rather, STUN defines a tool that can be used inside a larger solution. The term "STUN Usage" is used for any solution that uses STUN as a component.

A STUN Usage defines how STUN is actually utilized -- when to send requests, what to do with the responses, and which optional procedures defined here (or in an extension to STUN) are to be used. A usage also defines:

- Which STUN methods are used.
- What transports are used. If DTLS-over-UDP is used, then implementing the denial-of-service countermeasure described in Section 4.2.1 of [RFC6347] is mandatory.
- What authentication and message-integrity mechanisms are used.
- The considerations around manual vs. automatic key derivation for the integrity mechanism, as discussed in [RFC4107].
- What mechanisms are used to distinguish STUN messages from other messages. When STUN is run over TCP or TLS-over-TCP, a framing mechanism may be required.
- How a STUN client determines the IP address and port of the STUN server.
- How simultaneous use of IPv4 and IPv6 addresses (Happy Eyeballs [RFC8305]) works with non-idempotent transactions when both address families are found for the STUN server.
- Whether backwards compatibility to RFC 3489 is required.
- What optional attributes defined here (such as FINGERPRINT and ALTERNATE-SERVER) or in other extensions are required.
- If MESSAGE-INTEGRITY-SHA256 truncation is permitted, and the limits permitted for truncation.
- The keep-alive mechanism if STUN is run over TCP or TLS-over-TCP.

- If anycast addresses can be used for the server in case 1) TCP or TLS-over-TCP or 2) authentication is used.

In addition, any STUN Usage must consider the security implications of using STUN in that usage. A number of attacks against STUN are known (see the Security Considerations section in this document), and any usage must consider how these attacks can be thwarted or mitigated.

Finally, a usage must consider whether its usage of STUN is an example of the Unilateral Self-Address Fixing approach to NAT traversal and, if so, address the questions raised in RFC 3424 [RFC3424].

14. STUN Attributes

After the STUN header are zero or more attributes. Each attribute MUST be TLV encoded, with a 16-bit type, 16-bit length, and value. Each STUN attribute MUST end on a 32-bit boundary. As mentioned above, all fields in an attribute are transmitted most significant bit first.

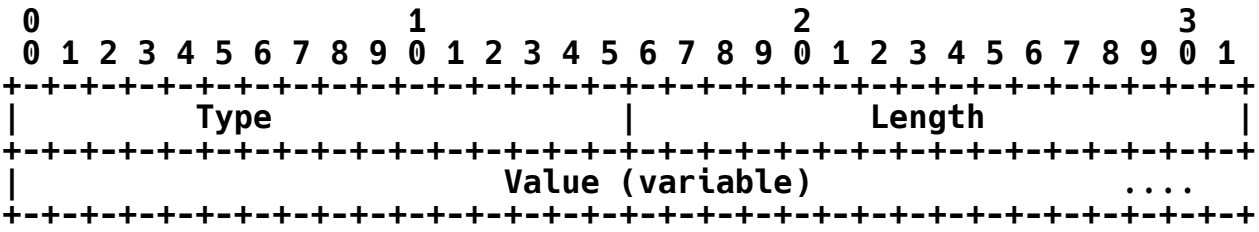

Figure 4: Format of STUN Attributes

The value in the Length field MUST contain the length of the Value part of the attribute, prior to padding, measured in bytes. Since STUN aligns attributes on 32-bit boundaries, attributes whose content is not a multiple of 4 bytes are padded with 1, 2, or 3 bytes of padding so that its value contains a multiple of 4 bytes. The padding bits MUST be set to zero on sending and MUST be ignored by the receiver.

Any attribute type MAY appear more than once in a STUN message. Unless specified otherwise, the order of appearance is significant: only the first occurrence needs to be processed by a receiver, and any duplicates MAY be ignored by a receiver.

To allow future revisions of this specification to add new attributes if needed, the attribute space is divided into two ranges. Attributes with type values between 0x0000 and 0x7FFF are

comprehension-required attributes, which means that the STUN agent cannot successfully process the message unless it understands the attribute. Attributes with type values between 0x8000 and 0xFFFF are comprehension-optional attributes, which means that those attributes can be ignored by the STUN agent if it does not understand them.

The set of STUN attribute types is maintained by IANA. The initial set defined by this specification is found in Section 18.3.

The rest of this section describes the format of the various attributes defined in this specification.

14.1. MAPPED-ADDRESS

The MAPPED-ADDRESS attribute indicates a reflexive transport address of the client. It consists of an 8-bit address family and a 16-bit port, followed by a fixed-length value representing the IP address. If the address family is IPv4, the address MUST be 32 bits. If the address family is IPv6, the address MUST be 128 bits. All fields must be in network byte order.

The format of the MAPPED-ADDRESS attribute is:

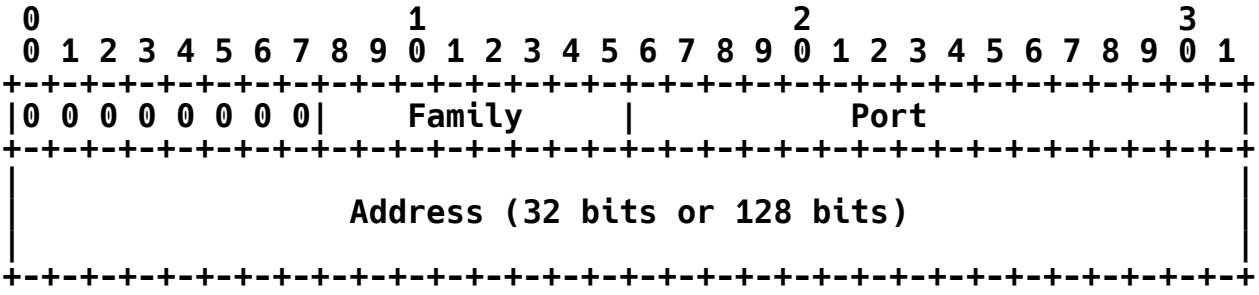

Figure 5: Format of MAPPED-ADDRESS Attribute

The address family can take on the following values:

0x01:IPv4
0x02:IPv6

The first 8 bits of the MAPPED-ADDRESS MUST be set to 0 and MUST be ignored by receivers. These bits are present for aligning parameters on natural 32-bit boundaries.

This attribute is used only by servers for achieving backwards compatibility with [RFC3489] clients.

14.2. XOR-MAPPED-ADDRESS

The XOR-MAPPED-ADDRESS attribute is identical to the MAPPED-ADDRESS attribute, except that the reflexive transport address is obfuscated through the XOR function.

The format of the XOR-MAPPED-ADDRESS is:

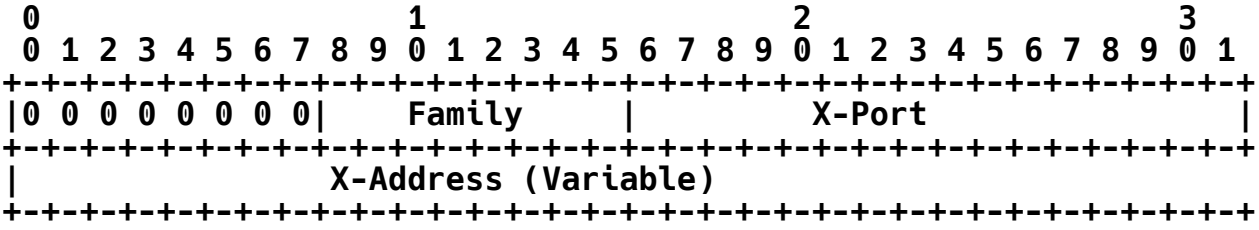

Figure 6: Format of XOR-MAPPED-ADDRESS Attribute

The Family field represents the IP address family and is encoded identically to the Family field in MAPPED-ADDRESS.

X-Port is computed by XOR'ing the mapped port with the most significant 16 bits of the magic cookie. If the IP address family is IPv4, X-Address is computed by XOR'ing the mapped IP address with the magic cookie. If the IP address family is IPv6, X-Address is computed by XOR'ing the mapped IP address with the concatenation of the magic cookie and the 96-bit transaction ID. In all cases, the XOR operation works on its inputs in network byte order (that is, the order they will be encoded in the message).

The rules for encoding and processing the first 8 bits of the attribute's value, the rules for handling multiple occurrences of the attribute, and the rules for processing address families are the same as for MAPPED-ADDRESS.

Note: XOR-MAPPED-ADDRESS and MAPPED-ADDRESS differ only in their encoding of the transport address. The former encodes the transport address by XOR'ing it with the magic cookie. The latter encodes it directly in binary. RFC 3489 originally specified only MAPPED-ADDRESS. However, deployment experience found that some NATs rewrite the 32-bit binary payloads containing the NAT's public IP address, such as STUN's MAPPED-ADDRESS attribute, in the well-meaning but misguided attempt to provide a generic Application Layer Gateway (ALG) function. Such behavior interferes with the operation of STUN and also causes failure of STUN's message-integrity checking.

14.3. USERNAME

The USERNAME attribute is used for message integrity. It identifies the username and password combination used in the message-integrity check.

The value of USERNAME is a variable-length value containing the authentication username. It MUST contain a UTF-8-encoded [RFC3629] sequence of fewer than 509 bytes and MUST have been processed using the OpaqueString profile [RFC8265]. A compliant implementation MUST be able to parse a UTF-8-encoded sequence of 763 or fewer octets to be compatible with [RFC5389].

> Note: [RFC5389] mistakenly referenced the definition of UTF-8 in [RFC2279]. [RFC2279] assumed up to 6 octets per characters encoded. [RFC2279] was replaced by [RFC3629], which allows only 4 octets per character encoded, consistent with changes made in Unicode 2.0 and ISO/IEC 10646.
>
> Note: This specification uses the OpaqueString profile instead of the UsernameCasePreserved profile for username string processing in order to improve compatibility with deployed password stores. Many password databases used for HTTP and SIP Digest authentication store the MD5 hash of username:realm:password instead of storing a plain text password. In [RFC3489], STUN authentication was designed to be compatible with these existing databases to the extent possible, which like SIP and HTTP performed no pre-processing of usernames and passwords other than prohibiting non-space ASCII control characters. The next revision of the STUN specification, [RFC5389], used the SASLprep [RFC4013] stringprep [RFC3454] profile to pre-process usernames and passwords. SASLprep uses Unicode Normalization Form KC (Compatibility Decomposition, followed by Canonical Composition) [UAX15] and prohibits various control, space, and non-text, deprecated, or inappropriate codepoints. The PRECIS framework [RFC8264] obsoletes stringprep. PRECIS handling of usernames and passwords [RFC8265] uses Unicode Normalization Form C (Canonical Decomposition, followed by Canonical Composition). While there are specific cases where different username strings under HTTP Digest could be mapped to a single STUN username processed with OpaqueString, these cases are extremely unlikely and easy to detect and correct. With a UsernameCasePreserved profile, it would be more likely that valid usernames under HTTP Digest would not match their processed forms (specifically usernames containing bidirectional text and compatibility forms). Operators are free to further restrict the allowed codepoints in usernames to avoid problematic characters.

14.4. USERHASH

The USERHASH attribute is used as a replacement for the USERNAME attribute when username anonymity is supported.

The value of USERHASH has a fixed length of 32 bytes. The username MUST have been processed using the OpaqueString profile [RFC8265], and the realm MUST have been processed using the OpaqueString profile [RFC8265] before hashing.

The following is the operation that the client will perform to hash the username:

userhash = SHA-256(OpaqueString(username) ":" OpaqueString(realm))

14.5. MESSAGE-INTEGRITY

The MESSAGE-INTEGRITY attribute contains an HMAC-SHA1 [RFC2104] of the STUN message. The MESSAGE-INTEGRITY attribute can be present in any STUN message type. Since it uses the SHA-1 hash, the HMAC will be 20 bytes.

The key for the HMAC depends on which credential mechanism is in use. Section 9.1.1 defines the key for the short-term credential mechanism, and Section 9.2.2 defines the key for the long-term credential mechanism. Other credential mechanisms MUST define the key that is used for the HMAC.

The text used as input to HMAC is the STUN message, up to and including the attribute preceding the MESSAGE-INTEGRITY attribute. The Length field of the STUN message header is adjusted to point to the end of the MESSAGE-INTEGRITY attribute. The value of the MESSAGE-INTEGRITY attribute is set to a dummy value.

Once the computation is performed, the value of the MESSAGE-INTEGRITY attribute is filled in, and the value of the length in the STUN header is set to its correct value -- the length of the entire message. Similarly, when validating the MESSAGE-INTEGRITY, the Length field in the STUN header must be adjusted to point to the end of the MESSAGE-INTEGRITY attribute prior to calculating the HMAC over the STUN message, up to and including the attribute preceding the MESSAGE-INTEGRITY attribute. Such adjustment is necessary when attributes, such as FINGERPRINT and MESSAGE-INTEGRITY-SHA256, appear after MESSAGE-INTEGRITY. See also [RFC5769] for examples of such calculations.

14.6. MESSAGE-INTEGRITY-SHA256

The MESSAGE-INTEGRITY-SHA256 attribute contains an HMAC-SHA256 [RFC2104] of the STUN message. The MESSAGE-INTEGRITY-SHA256 attribute can be present in any STUN message type. The MESSAGE-INTEGRITY-SHA256 attribute contains an initial portion of the HMAC-SHA-256 [RFC2104] of the STUN message. The value will be at most 32 bytes, but it MUST be at least 16 bytes and MUST be a multiple of 4 bytes. The value must be the full 32 bytes unless the STUN Usage explicitly specifies that truncation is allowed. STUN Usages may specify a minimum length longer than 16 bytes.

The key for the HMAC depends on which credential mechanism is in use. Section 9.1.1 defines the key for the short-term credential mechanism, and Section 9.2.2 defines the key for the long-term credential mechanism. Other credential mechanism MUST define the key that is used for the HMAC.

The text used as input to HMAC is the STUN message, up to and including the attribute preceding the MESSAGE-INTEGRITY-SHA256 attribute. The Length field of the STUN message header is adjusted to point to the end of the MESSAGE-INTEGRITY-SHA256 attribute. The value of the MESSAGE-INTEGRITY-SHA256 attribute is set to a dummy value.

Once the computation is performed, the value of the MESSAGE-INTEGRITY-SHA256 attribute is filled in, and the value of the length in the STUN header is set to its correct value -- the length of the entire message. Similarly, when validating the MESSAGE-INTEGRITY-SHA256, the Length field in the STUN header must be adjusted to point to the end of the MESSAGE-INTEGRITY-SHA256 attribute prior to calculating the HMAC over the STUN message, up to and including the attribute preceding the MESSAGE-INTEGRITY-SHA256 attribute. Such adjustment is necessary when attributes, such as FINGERPRINT, appear after MESSAGE-INTEGRITY-SHA256. See also Appendix B.1 for examples of such calculations.

14.7. FINGERPRINT

The FINGERPRINT attribute MAY be present in all STUN messages.

The value of the attribute is computed as the CRC-32 of the STUN message up to (but excluding) the FINGERPRINT attribute itself, XOR'ed with the 32-bit value 0x5354554e. (The XOR operation ensures that the FINGERPRINT test will not report a false positive on a packet containing a CRC-32 generated by an application protocol.) The 32-bit CRC is the one defined in ITU V.42 [ITU.V42.2002], which

has a generator polynomial of x^32 + x^26 + x^23 + x^22 + x^16 + x^12 + x^11 + x^10 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1. See the sample code for the CRC-32 in Section 8 of [RFC1952].

When present, the FINGERPRINT attribute MUST be the last attribute in the message and thus will appear after MESSAGE-INTEGRITY and MESSAGE-INTEGRITY-SHA256.

The FINGERPRINT attribute can aid in distinguishing STUN packets from packets of other protocols. See Section 7.

As with MESSAGE-INTEGRITY and MESSAGE-INTEGRITY-SHA256, the CRC used in the FINGERPRINT attribute covers the Length field from the STUN message header. Therefore, prior to computation of the CRC, this value must be correct and include the CRC attribute as part of the message length. When using the FINGERPRINT attribute in a message, the attribute is first placed into the message with a dummy value; then, the CRC is computed, and the value of the attribute is updated. If the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute is also present, then it must be present with the correct message-integrity value before the CRC is computed, since the CRC is done over the value of the MESSAGE-INTEGRITY and MESSAGE-INTEGRITY-SHA256 attributes as well.

14.8. ERROR-CODE

The ERROR-CODE attribute is used in error response messages. It contains a numeric error code value in the range of 300 to 699 plus a textual reason phrase encoded in UTF-8 [RFC3629]; it is also consistent in its code assignments and semantics with SIP [RFC3261] and HTTP [RFC7231]. The reason phrase is meant for diagnostic purposes and can be anything appropriate for the error code. Recommended reason phrases for the defined error codes are included in the IANA registry for error codes. The reason phrase MUST be a UTF-8-encoded [RFC3629] sequence of fewer than 128 characters (which can be as long as 509 bytes when encoding them or 763 bytes when decoding them).

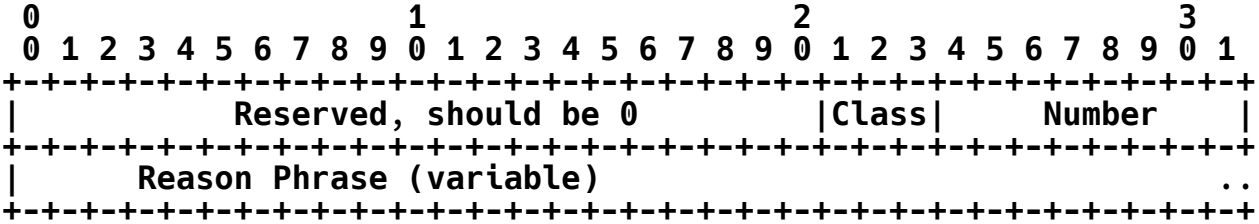

Figure 7: Format of ERROR-CODE Attribute

To facilitate processing, the class of the error code (the hundreds digit) is encoded separately from the rest of the code, as shown in Figure 7.

The Reserved bits SHOULD be 0 and are for alignment on 32-bit boundaries. Receivers MUST ignore these bits. The Class represents the hundreds digit of the error code. The value MUST be between 3 and 6. The Number represents the binary encoding of the error code modulo 100, and its value MUST be between 0 and 99.

The following error codes, along with their recommended reason phrases, are defined:

300 Try Alternate: The client should contact an alternate server for this request. This error response MUST only be sent if the request included either a USERNAME or USERHASH attribute and a valid MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute; otherwise, it MUST NOT be sent and error code 400 (Bad Request) is suggested. This error response MUST be protected with the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute, and receivers MUST validate the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 of this response before redirecting themselves to an alternate server.

Note: Failure to generate and validate message integrity for a 300 response allows an on-path attacker to falsify a 300 response thus causing subsequent STUN messages to be sent to a victim.

400 Bad Request: The request was malformed. The client SHOULD NOT retry the request without modification from the previous attempt. The server may not be able to generate a valid MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 for this error, so the client MUST NOT expect a valid MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute on this response.

401 Unauthenticated: The request did not contain the correct credentials to proceed. The client should retry the request with proper credentials.

420 Unknown Attribute: The server received a STUN packet containing a comprehension-required attribute that it did not understand. The server MUST put this unknown attribute in the UNKNOWN-ATTRIBUTE attribute of its error response.

438 Stale Nonce: The NONCE used by the client was no longer valid. The client should retry, using the NONCE provided in the response.

500 Server Error: The server has suffered a temporary error. The client should try again.

14.9. REALM

The REALM attribute may be present in requests and responses. It contains text that meets the grammar for "realm-value" as described in [RFC3261] but without the double quotes and their surrounding whitespace. That is, it is an unquoted realm-value (and is therefore a sequence of qdtext or quoted-pair). It MUST be a UTF-8-encoded [RFC3629] sequence of fewer than 128 characters (which can be as long as 509 bytes when encoding them and as long as 763 bytes when decoding them) and MUST have been processed using the OpaqueString profile [RFC8265].

Presence of the REALM attribute in a request indicates that long-term credentials are being used for authentication. Presence in certain error responses indicates that the server wishes the client to use a long-term credential in that realm for authentication.

14.10. NONCE

The NONCE attribute may be present in requests and responses. It contains a sequence of qdtext or quoted-pair, which are defined in [RFC3261]. Note that this means that the NONCE attribute will not contain the actual surrounding quote characters. The NONCE attribute MUST be fewer than 128 characters (which can be as long as 509 bytes when encoding them and a long as 763 bytes when decoding them). See Section 5.4 of [RFC7616] for guidance on selection of nonce values in a server.

14.11. PASSWORD-ALGORITHMS

The PASSWORD-ALGORITHMS attribute may be present in requests and responses. It contains the list of algorithms that the server can use to derive the long-term password.

The set of known algorithms is maintained by IANA. The initial set defined by this specification is found in Section 18.5.

The attribute contains a list of algorithm numbers and variable length parameters. The algorithm number is a 16-bit value as defined in Section 18.5. The parameters start with the length (prior to padding) of the parameters as a 16-bit value, followed by the parameters that are specific to each algorithm. The parameters are padded to a 32-bit boundary, in the same manner as an attribute.

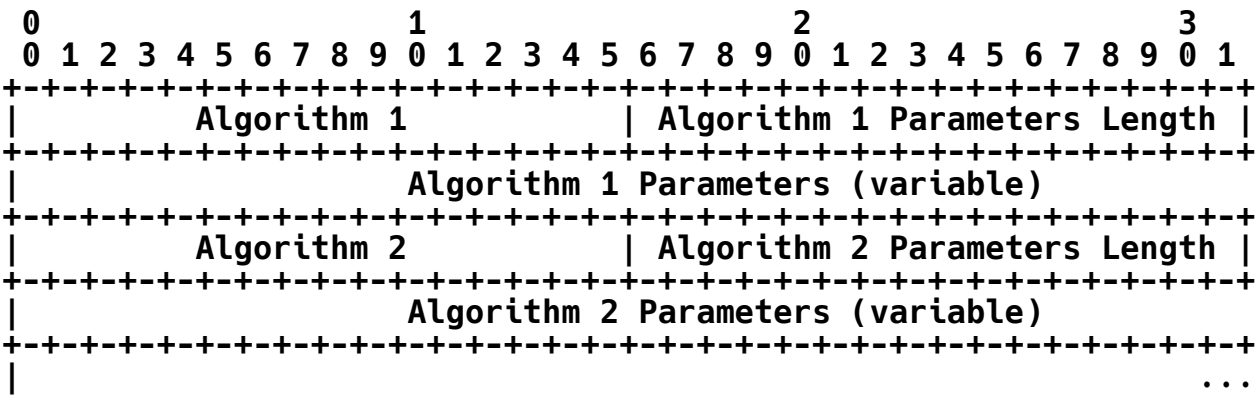

Figure 8: Format of PASSWORD-ALGORITHMS Attribute

14.12. PASSWORD-ALGORITHM

The PASSWORD-ALGORITHM attribute is present only in requests. It contains the algorithm that the server must use to derive a key from the long-term password.

The set of known algorithms is maintained by IANA. The initial set defined by this specification is found in Section 18.5.

The attribute contains an algorithm number and variable length parameters. The algorithm number is a 16-bit value as defined in Section 18.5. The parameters starts with the length (prior to padding) of the parameters as a 16-bit value, followed by the parameters that are specific to the algorithm. The parameters are padded to a 32-bit boundary, in the same manner as an attribute. Similarly, the padding bits MUST be set to zero on sending and MUST be ignored by the receiver.

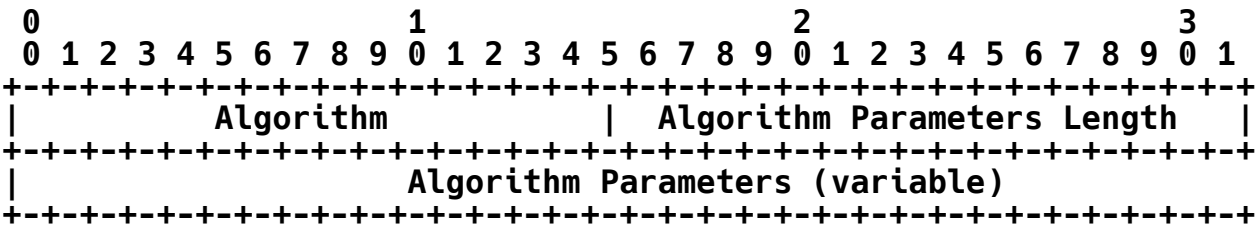

Figure 9: Format of PASSWORD-ALGORITHM Attribute

14.13. UNKNOWN-ATTRIBUTES

The UNKNOWN-ATTRIBUTES attribute is present only in an error response when the response code in the ERROR-CODE attribute is 420 (Unknown Attribute).

The attribute contains a list of 16-bit values, each of which represents an attribute type that was not understood by the server.

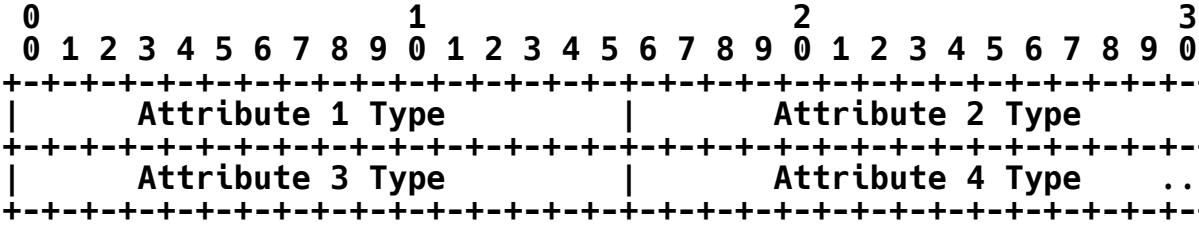

Figure 10: Format of UNKNOWN-ATTRIBUTES Attribute

Note: In [RFC3489], this field was padded to 32 by duplicating the last attribute. In this version of the specification, the normal padding rules for attributes are used instead.

14.14. SOFTWARE

The SOFTWARE attribute contains a textual description of the software being used by the agent sending the message. It is used by clients and servers. Its value SHOULD include manufacturer and version number. The attribute has no impact on operation of the protocol and serves only as a tool for diagnostic and debugging purposes. The value of SOFTWARE is variable length. It MUST be a UTF-8-encoded [RFC3629] sequence of fewer than 128 characters (which can be as long as 509 when encoding them and as long as 763 bytes when decoding them).

14.15. ALTERNATE-SERVER

The alternate server represents an alternate transport address identifying a different STUN server that the STUN client should try.

It is encoded in the same way as MAPPED-ADDRESS and thus refers to a single server by IP address.

14.16. ALTERNATE-DOMAIN

The alternate domain represents the domain name that is used to verify the IP address in the ALTERNATE-SERVER attribute when the transport protocol uses TLS or DTLS.

The value of ALTERNATE-DOMAIN is variable length. It MUST be a valid DNS name [RFC1123] (including A-labels [RFC5890]) of 255 or fewer ASCII characters.

15. Operational Considerations

STUN MAY be used with anycast addresses, but only with UDP and in STUN Usages where authentication is not used.

16. Security Considerations

Implementations and deployments of a STUN Usage using TLS or DTLS MUST follow the recommendations in [BCP195].

Implementations and deployments of a STUN Usage using the long-term credential mechanism (Section 9.2) MUST follow the recommendations in Section 5 of [RFC7616].

16.1. Attacks against the Protocol

16.1.1. Outside Attacks

An attacker can try to modify STUN messages in transit, in order to cause a failure in STUN operation. These attacks are detected for both requests and responses through the message-integrity mechanism, using either a short-term or long-term credential. Of course, once detected, the manipulated packets will be dropped, causing the STUN transaction to effectively fail. This attack is possible only by an on-path attacker.

An attacker that can observe, but not modify, STUN messages in-transit (for example, an attacker present on a shared access medium, such as Wi-Fi) can see a STUN request and then immediately send a STUN response, typically an error response, in order to disrupt STUN processing. This attack is also prevented for messages that utilize MESSAGE-INTEGRITY. However, some error responses, those related to authentication in particular, cannot be protected by MESSAGE-INTEGRITY. When STUN itself is run over a secure transport protocol (e.g., TLS), these attacks are completely mitigated.

Depending on the STUN Usage, these attacks may be of minimal consequence and thus do not require message integrity to mitigate. For example, when STUN is used to a basic STUN server to discover a server reflexive candidate for usage with ICE, authentication and message integrity are not required since these attacks are detected during the connectivity check phase. The connectivity checks themselves, however, require protection for proper operation of ICE overall. As described in Section 13, STUN Usages describe when authentication and message integrity are needed.

Since STUN uses the HMAC of a shared secret for authentication and integrity protection, it is subject to offline dictionary attacks. When authentication is utilized, it SHOULD be with a strong password that is not readily subject to offline dictionary attacks. Protection of the channel itself, using TLS or DTLS, mitigates these attacks.

STUN supports both MESSAGE-INTEGRITY and MESSAGE-INTEGRITY-SHA256, which makes STUN subject to bid-down attacks by an on-path attacker. An attacker could strip the MESSAGE-INTEGRITY-SHA256 attribute, leaving only the MESSAGE-INTEGRITY attribute and thus exploiting a potential vulnerability. Protection of the channel itself, using TLS or DTLS, mitigates these attacks. Timely removal of the support of MESSAGE-INTEGRITY in a future version of STUN is necessary.

Note: The use of SHA-256 for password hashing does not meet modern standards, which are aimed at slowing down exhaustive password searches by providing a relatively slow minimum time to compute the hash. Although better algorithms such as Argon2 [Argon2] are available, SHA-256 was chosen for consistency with [RFC7616].

16.1.2. Inside Attacks

A rogue client may try to launch a DoS attack against a server by sending it a large number of STUN requests. Fortunately, STUN requests can be processed statelessly by a server, making such attacks hard to launch effectively.

A rogue client may use a STUN server as a reflector, sending it requests with a falsified source IP address and port. In such a case, the response would be delivered to that source IP and port. There is no amplification of the number of packets with this attack (the STUN server sends one packet for each packet sent by the client), though there is a small increase in the amount of data, since STUN responses are typically larger than requests. This attack is mitigated by ingress source address filtering.

Revealing the specific software version of the agent through the SOFTWARE attribute might allow them to become more vulnerable to attacks against software that is known to contain security holes. Implementers SHOULD make usage of the SOFTWARE attribute a configurable option.

16.1.3. Bid-Down Attacks

This document adds the possibility of selecting different algorithms to protect the confidentiality of the passwords stored on the server side when using the long-term credential mechanism while still

ensuring compatibility with MD5, which was the algorithm used in [RFC5389]. This selection works by having the server send to the client the list of algorithms supported in a PASSWORD-ALGORITHMS attribute and having the client send back a PASSWORD-ALGORITHM attribute containing the algorithm selected.

Because the PASSWORD-ALGORITHMS attribute has to be sent in an unauthenticated response, an on-path attacker wanting to exploit an eventual vulnerability in MD5 can just strip the PASSWORD-ALGORITHMS attribute from the unprotected response, thus making the server subsequently act as if the client was implementing the version of this protocol defined in [RFC5389].

To protect against this attack and other similar bid-down attacks, the nonce is enriched with a set of security bits that indicates which security features are in use. In the case of the selection of the password algorithm, the matching bit is set in the nonce returned by the server in the same response that contains the PASSWORD-ALGORITHMS attribute. Because the nonce used in subsequent authenticated transactions is verified by the server to be identical to what was originally sent, it cannot be modified by an on-path attacker. Additionally, the client is mandated to copy the received PASSWORD-ALGORITHMS attribute in the next authenticated transaction to that server.

An on-path attack that removes the PASSWORD-ALGORITHMS will be detected because the client will not be able to send it back to the server in the next authenticated transaction. The client will detect that attack because the security bit is set but the matching attribute is missing; this will end the session. A client using an older version of this protocol will not send the PASSWORD-ALGORITHMS back but can only use MD5 anyway, so the attack is inconsequential.

The on-path attack may also try to remove the security bit together with the PASSWORD-ALGORITHMS attribute, but the server will discover that when the next authenticated transaction contains an invalid nonce.

An on-path attack that removes some algorithms from the PASSWORD-ALGORITHMS attribute will be equally defeated because that attribute will be different from the original one when the server verifies it in the subsequent authenticated transaction.

Note that the bid-down protection mechanism introduced in this document is inherently limited by the fact that it is not possible to detect an attack until the server receives the second request after the 401 (Unauthenticated) response.

SHA-256 was chosen as the new default for password hashing for its compatibility with [RFC7616], but because SHA-256 (like MD5) is a comparatively fast algorithm, it does little to deter brute-force attacks. Specifically, this means that if the user has a weak password, an attacker that captures a single exchange can use a brute-force attack to learn the user's password and then potentially impersonate the user to the server and to other servers where the same password was used. Note that such an attacker can impersonate the user to the server itself without any brute-force attack.

A stronger (which is to say, slower) algorithm, like Argon2 [Argon2], would help both of these cases; however, in the first case, it would only help after the database entry for this user is updated to exclusively use that stronger mechanism.

The bid-down defenses in this protocol prevent an attacker from forcing the client and server to complete a handshake using weaker algorithms than they jointly support, but only if the weakest joint algorithm is strong enough that it cannot be compromised by a brute-force attack. However, this does not defend against many attacks on those algorithms; specifically, an on-path attacker might perform a bid-down attack on a client that supports both Argon2 [Argon2] and SHA-256 for password hashing and use that to collect a MESSAGE-INTEGRITY-SHA256 value that it can then use for an offline brute-force attack. This would be detected when the server receives the second request, but that does not prevent the attacker from obtaining the MESSAGE-INTEGRITY-SHA256 value.

Similarly, an attack against the USERHASH mechanism will not succeed in establishing a session as the server will detect that the feature was discarded on path, but the client would still have been convinced to send its username in the clear in the USERNAME attribute, thus disclosing it to the attacker.

Finally, when the bid-down protection mechanism is employed for a future upgrade of the HMAC algorithm used to protect messages, it will offer only a limited protection if the current HMAC algorithm is already compromised.

16.2. Attacks Affecting the Usage

This section lists attacks that might be launched against a usage of STUN. Each STUN Usage must consider whether these attacks are applicable to it and, if so, discuss countermeasures.

Most of the attacks in this section revolve around an attacker modifying the reflexive address learned by a STUN client through a Binding request/response transaction. Since the usage of the

reflexive address is a function of the usage, the applicability and remediation of these attacks are usage-specific. In common situations, modification of the reflexive address by an on-path attacker is easy to do. Consider, for example, the common situation where STUN is run directly over UDP. In this case, an on-path attacker can modify the source IP address of the Binding request before it arrives at the STUN server. The STUN server will then return this IP address in the XOR-MAPPED-ADDRESS attribute to the client and send the response back to that (falsified) IP address and port. If the attacker can also intercept this response, it can direct it back towards the client. Protecting against this attack by using a message-integrity check is impossible, since a message-integrity value cannot cover the source IP address and the intervening NAT must be able to modify this value. Instead, one solution to prevent the attacks listed below is for the client to verify the reflexive address learned, as is done in ICE [RFC8445].

Other usages may use other means to prevent these attacks.

16.2.1. Attack I: Distributed DoS (DDoS) against a Target

In this attack, the attacker provides one or more clients with the same faked reflexive address that points to the intended target. This will trick the STUN clients into thinking that their reflexive addresses are equal to that of the target. If the clients hand out that reflexive address in order to receive traffic on it (for example, in SIP messages), the traffic will instead be sent to the target. This attack can provide substantial amplification, especially when used with clients that are using STUN to enable multimedia applications. However, it can only be launched against targets for which packets from the STUN server to the target pass through the attacker, limiting the cases in which it is possible.

16.2.2. Attack II: Silencing a Client

In this attack, the attacker provides a STUN client with a faked reflexive address. The reflexive address it provides is a transport address that routes to nowhere. As a result, the client won't receive any of the packets it expects to receive when it hands out the reflexive address. This exploitation is not very interesting for the attacker. It impacts a single client, which is frequently not the desired target. Moreover, any attacker that can mount the attack could also deny service to the client by other means, such as preventing the client from receiving any response from the STUN server, or even a DHCP server. As with the attack described in Section 16.2.1, this attack is only possible when the attacker is on path for packets sent from the STUN server towards this unused IP address.

16.2.3. Attack III: Assuming the Identity of a Client

This attack is similar to attack II. However, the faked reflexive address points to the attacker itself. This allows the attacker to receive traffic that was destined for the client.

16.2.4. Attack IV: Eavesdropping

In this attack, the attacker forces the client to use a reflexive address that routes to itself. It then forwards any packets it receives to the client. This attack allows the attacker to observe all packets sent to the client. However, in order to launch the attack, the attacker must have already been able to observe packets from the client to the STUN server. In most cases (such as when the attack is launched from an access network), this means that the attacker could already observe packets sent to the client. This attack is, as a result, only useful for observing traffic by attackers on the path from the client to the STUN server, but not generally on the path of packets being routed towards the client.

Note that this attack can be trivially launched by the STUN server itself, so users of STUN servers should have the same level of trust in the users of STUN servers as any other node that can insert itself into the communication flow.

16.3. Hash Agility Plan

This specification uses HMAC-SHA256 for computation of the message integrity, sometimes in combination with HMAC-SHA1. If, at a later time, HMAC-SHA256 is found to be compromised, the following remedy should be applied:

- Both a new message-integrity attribute and a new STUN Security Feature bit will be allocated in a Standards Track document. The new message-integrity attribute will have its value computed using a new hash. The STUN Security Feature bit will be used to simultaneously 1) signal to a STUN client using the long-term credential mechanism that this server supports this new hash algorithm and 2) prevent bid-down attacks on the new message-integrity attribute.

- STUN clients and servers using the short-term credential mechanism will need to update the external mechanism that they use to signal what message-integrity attributes are in use.

The bid-down protection mechanism described in this document is new and thus cannot currently protect against a bid-down attack that lowers the strength of the hash algorithm to HMAC-SHA1. This is why,

after a transition period, a new document updating this one will assign a new STUN Security Feature bit for deprecating HMAC-SHA1. When used, this bit will signal that HMAC-SHA1 is deprecated and should no longer be used.

Similarly, if HMAC-SHA256 is found to be compromised, a new userhash attribute and a new STUN Security Feature bit will be allocated in a Standards Track document. The new userhash attribute will have its value computed using a new hash. The STUN Security Feature bit will be used to simultaneously 1) signal to a STUN client using the long-term credential mechanism that this server supports this new hash algorithm for the userhash attribute and 2) prevent bid-down attacks on the new userhash attribute.

17. IAB Considerations

The IAB has studied the problem of Unilateral Self-Address Fixing (UNSAF), which is the general process by which a client attempts to determine its address in another realm on the other side of a NAT through a collaborative protocol reflection mechanism [RFC3424]. STUN can be used to perform this function using a Binding request/response transaction if one agent is behind a NAT and the other is on the public side of the NAT.

The IAB has suggested that protocols developed for this purpose document a specific set of considerations. Because some STUN Usages provide UNSAF functions (such as ICE [RFC8445]) and others do not (such as SIP Outbound [RFC5626]), answers to these considerations need to be addressed by the usages themselves.

18. IANA Considerations

18.1. STUN Security Features Registry

A STUN Security Feature set defines 24 bits as flags.

IANA has created a new registry containing the STUN Security Features that are protected by the bid-down attack prevention mechanism described in Section 9.2.1.

The initial STUN Security Features are:

Bit 0: Password algorithms
Bit 1: Username anonymity
Bit 2-23: Unassigned

Bits are assigned starting from the most significant side of the bit set, so Bit 0 is the leftmost bit and Bit 23 is the rightmost bit.

New Security Features are assigned by Standards Action [RFC8126].

18.2. STUN Methods Registry

A STUN method is a hex number in the range 0x000-0x0FF. The encoding of a STUN method into a STUN message is described in Section 5.

STUN methods in the range 0x000-0x07F are assigned by IETF Review [RFC8126]. STUN methods in the range 0x080-0x0FF are assigned by Expert Review [RFC8126]. The responsibility of the expert is to verify that the selected codepoint(s) is not in use and that the request is not for an abnormally large number of codepoints. Technical review of the extension itself is outside the scope of the designated expert responsibility.

IANA has updated the name for method 0x002 as described below as well as updated the reference from RFC 5389 to RFC 8489 for the following STUN methods:

0x000: Reserved
0x001: Binding
0x002: Reserved; was SharedSecret prior to [RFC5389]

18.3. STUN Attributes Registry

A STUN attribute type is a hex number in the range 0x0000-0xFFFF. STUN attribute types in the range 0x0000-0x7FFF are considered comprehension-required; STUN attribute types in the range 0x8000-0xFFFF are considered comprehension-optional. A STUN agent handles unknown comprehension-required and comprehension-optional attributes differently.

STUN attribute types in the first half of the comprehension-required range (0x0000-0x3FFF) and in the first half of the comprehension-optional range (0x8000-0xBFFF) are assigned by IETF Review [RFC8126]. STUN attribute types in the second half of the comprehension-required range (0x4000-0x7FFF) and in the second half of the comprehension-optional range (0xC000-0xFFFF) are assigned by Expert Review [RFC8126]. The responsibility of the expert is to verify that the selected codepoint(s) are not in use and that the request is not for an abnormally large number of codepoints. Technical review of the extension itself is outside the scope of the designated expert responsibility.

18.3.1. Updated Attributes

IANA has updated the names for attributes 0x0002, 0x0004, 0x0005, 0x0007, and 0x000B as well as updated the reference from RFC 5389 to RFC 8489 for each the following STUN methods.

In addition, [RFC5389] introduced a mistake in the name of attribute 0x0003; [RFC5389] called it CHANGE-ADDRESS when it was actually previously called CHANGE-REQUEST. Thus, IANA has updated the description for 0x0003 to read "Reserved; was CHANGE-REQUEST prior to [RFC5389]".

Comprehension-required range (0x0000-0x7FFF):
0x0000: Reserved
0x0001: MAPPED-ADDRESS
0x0002: Reserved; was RESPONSE-ADDRESS prior to [RFC5389]
0x0003: Reserved; was CHANGE-REQUEST prior to [RFC5389]
0x0004: Reserved; was SOURCE-ADDRESS prior to [RFC5389]
0x0005: Reserved; was CHANGED-ADDRESS prior to [RFC5389]
0x0006: USERNAME
0x0007: Reserved; was PASSWORD prior to [RFC5389]
0x0008: MESSAGE-INTEGRITY
0x0009: ERROR-CODE
0x000A: UNKNOWN-ATTRIBUTES
0x000B: Reserved; was REFLECTED-FROM prior to [RFC5389]
0x0014: REALM
0x0015: NONCE
0x0020: XOR-MAPPED-ADDRESS

Comprehension-optional range (0x8000-0xFFFF)
0x8022: SOFTWARE
0x8023: ALTERNATE-SERVER
0x8028: FINGERPRINT

18.3.2. New Attributes

IANA has added the following attribute to the "STUN Attributes" registry:

Comprehension-required range (0x0000-0x7FFF):
0x001C: MESSAGE-INTEGRITY-SHA256
0x001D: PASSWORD-ALGORITHM
0x001E: USERHASH

Comprehension-optional range (0x8000-0xFFFF)
0x8002: PASSWORD-ALGORITHMS
0x8003: ALTERNATE-DOMAIN

18.4. STUN Error Codes Registry

A STUN error code is a number in the range 0-699. STUN error codes are accompanied by a textual reason phrase in UTF-8 [RFC3629] that is intended only for human consumption and can be anything appropriate; this document proposes only suggested values.

STUN error codes are consistent in codepoint assignments and semantics with SIP [RFC3261] and HTTP [RFC7231].

New STUN error codes are assigned based on IETF Review [RFC8126]. The specification must carefully consider how clients that do not understand this error code will process it before granting the request. See the rules in Section 6.3.4.

IANA has updated the reference from RFC 5389 to RFC 8489 for the error codes defined in Section 14.8.

IANA has changed the name of the 401 error code from "Unauthorized" to "Unauthenticated".

18.5. STUN Password Algorithms Registry

IANA has created a new registry titled "STUN Password Algorithms".

A password algorithm is a hex number in the range 0x0000-0xFFFF.

The initial contents of the "Password Algorithm" registry are as follows:

0x0000: Reserved
0x0001: MD5
0x0002: SHA-256
0x0003-0xFFFF: Unassigned

Password algorithms in the first half of the range (0x0000-0x7FFF) are assigned by IETF Review [RFC8126]. Password algorithms in the second half of the range (0x8000-0xFFFF) are assigned by Expert Review [RFC8126].

18.5.1. Password Algorithms

18.5.1.1. MD5

This password algorithm is taken from [RFC1321].

The key length is 16 bytes, and the parameters value is empty.

Note: This algorithm MUST only be used for compatibility with legacy systems.

```
key = MD5(username ":" OpaqueString(realm)
  ":" OpaqueString(password))
```

18.5.1.2. SHA-256

This password algorithm is taken from [RFC7616].

The key length is 32 bytes, and the parameters value is empty.

```
key = SHA-256(username ":" OpaqueString(realm)
  ":" OpaqueString(password))
```

18.6. STUN UDP and TCP Port Numbers

IANA has updated the reference from RFC 5389 to RFC 8489 for the following ports in the "Service Name and Transport Protocol Port Number Registry".

```
stun   3478/tcp   Session Traversal Utilities for NAT (STUN) port
stun   3478/udp   Session Traversal Utilities for NAT (STUN) port
stuns  5349/tcp   Session Traversal Utilities for NAT (STUN) port
```

19. Changes since RFC 5389

This specification obsoletes [RFC5389]. This specification differs from RFC 5389 in the following ways:

- Added support for DTLS-over-UDP [RFC6347].
- Made clear that the RTO is considered stale if there are no transactions with the server.
- Aligned the RTO calculation with [RFC6298].
- Updated the ciphersuites for TLS.
- Added support for STUN URI [RFC7064].

- Added support for SHA256 message integrity.
- Updated the Preparation, Enforcement, and Comparison of Internationalized Strings (PRECIS) support to [RFC8265].
- Added protocol and registry to choose the password encryption algorithm.
- Added support for anonymous username.
- Added protocol and registry for preventing bid-down attacks.
- Specified that sharing a NONCE is no longer permitted.
- Added the possibility of using a domain name in the alternate server mechanism.
- Added more C snippets.
- Added test vector.

20. References

20.1. Normative References

[ITU.V42.2002] International Telecommunication Union, "Error-correcting procedures for DCEs using asynchronous-to-synchronous conversion", ITU-T Recommendation V.42, March 2002.

[KARN87] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", SIGCOMM '87, Proceedings of the ACM workshop on Frontiers in computer communications technology, Pages 2-7, DOI 10.1145/55483.55484, August 1987.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <https://www.rfc-editor.org/info/rfc1321>.

[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>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <https://www.rfc-editor.org/info/rfc2782>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[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>.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <https://www.rfc-editor.org/info/rfc5890>.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/info/rfc6151>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC7064] Nandakumar, S., Salgueiro, G., Jones, P., and M. Petit-Huguenin, "URI Scheme for the Session Traversal Utilities for NAT (STUN) Protocol", RFC 7064, DOI 10.17487/RFC7064, November 2013, <https://www.rfc-editor.org/info/rfc7064>.

[RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, DOI 10.17487/RFC7350, August 2014, <https://www.rfc-editor.org/info/rfc7350>.

[RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, <https://www.rfc-editor.org/info/rfc7616>.

[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>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[RFC8265] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", RFC 8265, DOI 10.17487/RFC8265, October 2017, <https://www.rfc-editor.org/info/rfc8265>.

[RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>.

20.2. Informative References

[Argon2] Biryukov, A., Dinu, D., Khovratovich, D., and S. Josefsson, "The memory-hard Argon2 password hash and proof-of-work function", Work in Progress, draft-irtf-cfrg-argon2-09, November 2019.

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, May 2015, <https://www.rfc-editor.org/info/bcp195>.

[RFC1952] Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, DOI 10.17487/RFC1952, May 1996, <https://www.rfc-editor.org/info/rfc1952>.

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, DOI 10.17487/RFC2279, January 1998, <https://www.rfc-editor.org/info/rfc2279>.

[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, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[RFC3424] Daigle, L., Ed. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", RFC 3424, DOI 10.17487/RFC3424, November 2002, <https://www.rfc-editor.org/info/rfc3424>.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, DOI 10.17487/RFC3454, December 2002, <https://www.rfc-editor.org/info/rfc3454>.

[RFC3489] Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, DOI 10.17487/RFC3489, March 2003, <https://www.rfc-editor.org/info/rfc3489>.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, DOI 10.17487/RFC4013, February 2005, <https://www.rfc-editor.org/info/rfc4013>.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, DOI 10.17487/RFC4107, June 2005, <https://www.rfc-editor.org/info/rfc4107>.

[RFC5090] Sterman, B., Sadolevsky, D., Schwartz, D., Williams, D., and W. Beck, "RADIUS Extension for Digest Authentication", RFC 5090, DOI 10.17487/RFC5090, February 2008, <https://www.rfc-editor.org/info/rfc5090>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <https://www.rfc-editor.org/info/rfc5389>.

[RFC5626] Jennings, C., Ed., Mahy, R., Ed., and F. Audet, Ed., "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, DOI 10.17487/RFC5626, October 2009, <https://www.rfc-editor.org/info/rfc5626>.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, DOI 10.17487/RFC5766, April 2010, <https://www.rfc-editor.org/info/rfc5766>.

[RFC5769] Denis-Courmont, R., "Test Vectors for Session Traversal Utilities for NAT (STUN)", RFC 5769, DOI 10.17487/RFC5769, April 2010, <https://www.rfc-editor.org/info/rfc5769>.

[RFC5780] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN)", RFC 5780, DOI 10.17487/RFC5780, May 2010, <https://www.rfc-editor.org/info/rfc5780>.

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, DOI 10.17487/RFC6544, March 2012, <https://www.rfc-editor.org/info/rfc6544>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[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>.

[RFC8264] Saint-Andre, P. and M. Blanchet, "PRECIS Framework: Preparation, Enforcement, and Comparison of Internationalized Strings in Application Protocols", RFC 8264, DOI 10.17487/RFC8264, October 2017, <https://www.rfc-editor.org/info/rfc8264>.

[RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/info/rfc8445>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[STUN-PMTUD] Petit-Huguenin, M., Salgueiro, G., and F. Garrido, "Packetization Layer Path MTU Discovery (PLMTUD) For UDP Transports Using Session Traversal Utilities for NAT (STUN)", Work in Progress, draft-ietf-tram-stun-pmtud-15, December 2019.

[UAX15] Unicode Standard Annex #15, "Unicode Normalization Forms", edited by Mark Davis and Ken Whistler. An integral part of The Unicode Standard, <http://unicode.org/reports/tr15/>.

Appendix A. C Snippet to Determine STUN Message Types

Given a 16-bit STUN message type value in host byte order in msg_type parameter, below are C macros to determine the STUN message types:

```
<CODE BEGINS>
#define IS_REQUEST(msg_type)       (((msg_type) & 0x0110) == 0x0000)
#define IS_INDICATION(msg_type)    (((msg_type) & 0x0110) == 0x0010)
#define IS_SUCCESS_RESP(msg_type)  (((msg_type) & 0x0110) == 0x0100)
#define IS_ERR_RESP(msg_type)      (((msg_type) & 0x0110) == 0x0110)
<CODE ENDS>
```

A function to convert method and class into a message type:

```
<CODE BEGINS>
int type(int method, int cls) {
  return (method & 0x1F80) << 2 | (method & 0x0070) << 1
    | (method & 0x000F) | (cls & 0x0002) << 7
    | (cls & 0x0001) << 4;
  }
<CODE ENDS>
```

A function to extract the method from the message type:

```
<CODE BEGINS>
int method(int type) {
  return (type & 0x3E00) >> 2 | (type & 0x00E0) >> 1
    | (type & 0x000F);
  }
<CODE ENDS>
```

A function to extract the class from the message type:

```
<CODE BEGINS>
int cls(int type) {
  return (type & 0x0100) >> 7 | (type & 0x0010) >> 4;
  }
<CODE ENDS>
```

Appendix B. Test Vectors

This section augments the list of test vectors defined in [RFC5769] with MESSAGE-INTEGRITY-SHA256. All the formats and definitions listed in Section 2 of [RFC5769] apply here.

B.1. Sample Request with Long-Term Authentication with MESSAGE-INTEGRITY-SHA256 and USERHASH

This request uses the following parameters:

Username: "<U+30DE><U+30C8><U+30EA><U+30C3><U+30AF><U+30B9>" (without quotes) unaffected by OpaqueString [RFC8265] processing

Password: "The<U+00AD>M<U+00AA>tr<U+2168>" and "TheMatrIX" (without quotes) respectively before and after OpaqueString [RFC8265] processing

Nonce: "obMatJos2AAACf//499k954d6OL34oL9FSTvy64sA" (without quotes)

Realm: "example.org" (without quotes)

```
      00 01 00 9c       Request type and message length
      21 12 a4 42       Magic cookie
      78 ad 34 33    }
      c6 ad 72 c0    }  Transaction ID
      29 da 41 2e    }
      00 1e 00 20       USERHASH attribute header
      4a 3c f3 8f    }
      ef 69 92 bd    }
      a9 52 c6 78    }
      04 17 da 0f    }  Userhash value (32 bytes)
      24 81 94 15    }
      56 9e 60 b2    }
      05 c4 6e 41    }
      40 7f 17 04    }
      00 15 00 29       NONCE attribute header
      6f 62 4d 61    }
      74 4a 6f 73    }
      32 41 41 41    }
      43 66 2f 2f    }
      34 39 39 6b    }  Nonce value and padding (3 bytes)
      39 35 34 64    }
      36 4f 4c 33    }
      34 6f 4c 39    }
      46 53 54 76    }
      79 36 34 73    }
      41 00 00 00    }
```

```
        00 14 00 0b      REALM attribute header
        65 78 61 6d   }
        70 6c 65 2e   }  Realm value (11 bytes) and padding (1 byte)
        6f 72 67 00   }
        00 1c 00 20      MESSAGE-INTEGRITY-SHA256 attribute header
        e4 68 6c 8f   }
        0e de b5 90   }
        13 e0 70 90   }
        01 0a 93 ef   }  HMAC-SHA256 value
        cc bc cc 54   }
        4c 0a 45 d9   }
        f8 30 aa 6d   }
        6f 73 5a 01   }
```

Acknowledgements

Thanks to Michael Tuexen, Tirumaleswar Reddy, Oleg Moskalenko, Simon Perreault, Benjamin Schwartz, Rifaat Shekh-Yusef, Alan Johnston, Jonathan Lennox, Brandon Williams, Olle Johansson, Martin Thomson, Mihaly Meszaros, Tolga Asveren, Noriyuki Torii, Spencer Dawkins, Dale Worley, Matthew Miller, Peter Saint-Andre, Julien Elie, Mirja Kuehlewind, Eric Rescorla, Ben Campbell, Adam Roach, Alexey Melnikov, and Benjamin Kaduk for the comments, suggestions, and questions that helped improve this document.

The Acknowledgements section of RFC 5389 appeared as follows:

The authors would like to thank Cedric Aoun, Pete Cordell, Cullen Jennings, Bob Penfield, Xavier Marjou, Magnus Westerlund, Miguel Garcia, Bruce Lowekamp, and Chris Sullivan for their comments, and Baruch Sterman and Alan Hawrylyshen for initial implementations. Thanks for Leslie Daigle, Allison Mankin, Eric Rescorla, and Henning Schulzrinne for IESG and IAB input on this work.

Contributors

Christian Huitema and Joel Weinberger were original coauthors of RFC 3489.

Authors' Addresses

Marc Petit-Huguenin
Impedance Mismatch

Email: marc@petit-huguenin.org

Gonzalo Salgueiro
Cisco
7200-12 Kit Creek Road
Research Triangle Park, NC 27709
United States of America

Email: gsalguei@cisco.com

Jonathan Rosenberg
Five9
Edison, NJ
United States of America

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net

Dan Wing
Citrix Systems, Inc.
United States of America

Email: dwing-ietf@fuggles.com

Rohan Mahy
Unaffiliated

Email: rohan.ietf@gmail.com

Philip Matthews
Nokia
600 March Road
Ottawa, Ontario K2K 2T6
Canada

Phone: 613-784-3139
Email: philip_matthews@magma.ca