INTERNET-DRAFT
draft-petke-serv-deity-protocol-00.txt
Expires: 15-May-97

G Brown
CompuServe

15 November 1996

Remote Passphrase Authentication
Part Four: Service-to-Deity Protocol

Status of this Memo

Abstract

Remote Passphrase Authentication provides a way to authenticate a user to a service by using a pass phrase over an insecure network, without revealing the pass phrase to eavesdroppers. In addition, the service need not know and does not learn the user's pass phrase, making this scheme useful in distributed environments where it would be difficult or inappropriate to trust a service with a pass phrase database or to allow the server to learn enough to masquerade as the user in a future authentication attempt.

This draft is part four of a four part series and explains the protocol between the service and the deity. Part one of this series (draft-petke-ext-intro-00.txt) provides an extended introduction to the problems of authentication over insecure networks. Part two (draft-petke-mech-00.txt) explains the RPA mechanism. Part three (draft-petke-http-auth-scheme-00.txt) explains how to incorporate the mechanism into HTTP.

This scheme was inspired by Dave Raggett's Mediated Digest Authentication paper.

Table of Contents

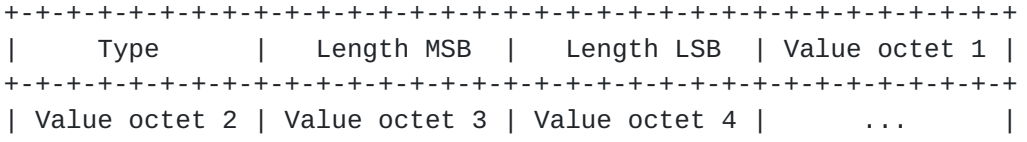
1. Introduction

The service sends a request to the authentication deity and receives a reply. The requests and replies may be packaged in UDP datagrams, or as byte streams over a TCP connection. The tradeoff is primarily that opening a TCP connection requires multiple round trip delays, where UDP doesn't; but TCP avoids the "I wonder whether it's actually running" issue.

How to find the deity is a service configuration issue. The service must know the IP addresses, TCP or UDP port numbers, etc., for the deities for a particular realm; it must also know its name and pass phrase in that realm.

2. Object formats

Every message is an object composed of other objects. Every object consists of a type-length-value encoded structure:

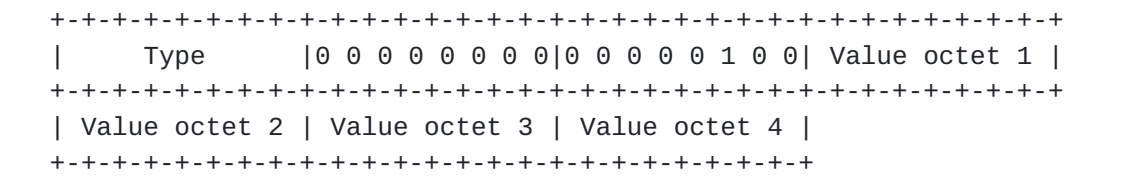


In this picture, each box represents one octet. Octets are transmitted in order from left to right, top to bottom.

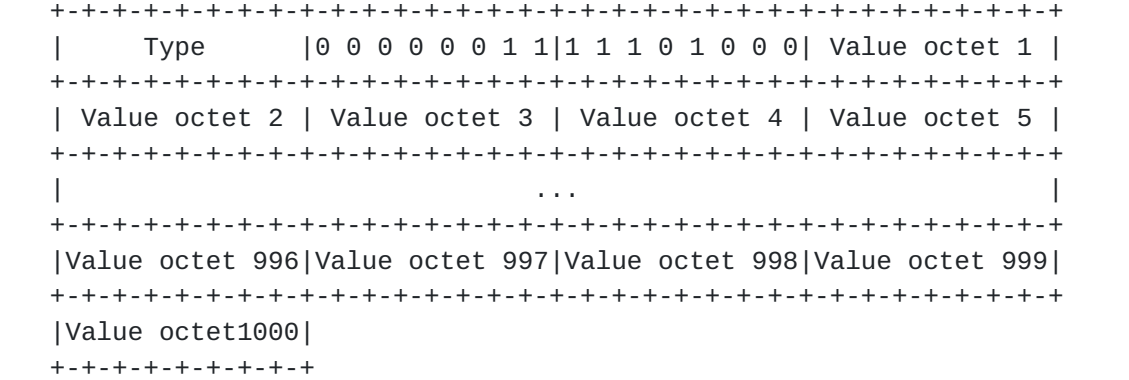
"Type" is a single octet that identifies the type of the object.

"Length" indicates the number of octets following the length field, as a 16-bit, big-endian value. The appropriate number of value octets--possibly none--follow the length field. Their meaning is determined by the type of the object; in some cases, the value octets contain a sequence of other objects.

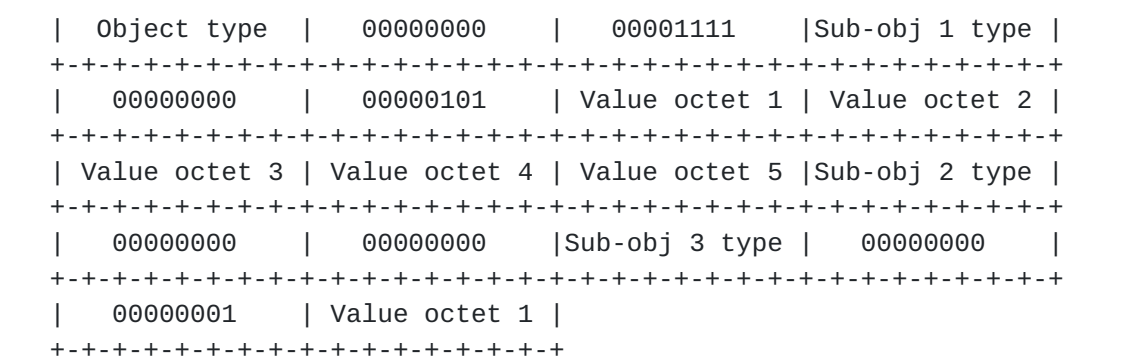
Here is an example of an object that contains 4 value octets:



And here is an example of an object that contains 1,000 value octets:



No padding or alignment is used; if an object contains sub-objects, they follow one another with no padding. For example, an object whose value consists of three sub-objects might look like this:



In this example, we have a single object whose value contains 15 octets. In this example, the value is a sequence of three objects, the first of which contains five octets, the second of which is zero length, and third of which contains one octet. The meaning of each object depends on its type; we'll describe all object types in detail after describing the message objects.

We'll sometimes use the term "sub-object" to refer to an object when it is a part of another object, but this is merely a matter of terminology. There is no difference in encoding nor in the meaning of the type field, regardless of whether the object is contained in some other object or not.

3. The blob

All messages may contain a "blob" that conveys information defined by a particular deity. The blob is used in three contexts.

* In a request, a service may use the blob to tell the deity the nature of the action for which authentication is being performed, if there's some reason to do so. In addition, the service might ask the deity for particular information about the username being authenticated, although, in the general case, the deity will already know what additional information to return to a particular service.

* In an affirmative response, the deity may return additional information about the username.

* In other responses, the blob might indicate something about the nature of the problem.

In general, different deities and services will have different information that's appropriate for inclusion in the blob, so it is difficult to conceive of a truly "standard" set of information. Thus, the definition of the blob's contents is left to each deity, but we define one format for inclusion of attribute/value pairs.

Should information in the blob be encrypted? That's a deity configuration issue. Beware, though, of naively encrypting the blob by XOR'ing it with the session key. That would reveal parts of the key because, in general, portions of the plaintext are known.

4. Message object types

There are six message object types, one for a request and five kinds of replies.

* Authentication request
* Authentication response, affirmative
* Authentication response, no service
* Authentication response, negative
* Authentication response, invalid service
* Authentication response, problem

The various response flavors indicate various conditions of the account as described below.

Remember, a message is simply an object that contains other objects. The message itself is encoded as a type, length, and value, as described above, where the value consists of the concatenation of the component objects of that message; each component object consists of its own type, length, and value. Unless stated to the contrary, all messages must contain exactly the objects indicated in the order shown. Optional components, such as the blob, may be omitted.

> [When appropriate, it is possible to add extensions, or make a sub-object optional, yet parse the containing object successfully. But in a security protocol, it is best to stick to well-defined formats, rather than adopting a "construct them any way you wish" attitude.]

Contents of the component objects are explained in more detail following the descriptions of the message objects.

4.1 Authentication request

An authentication request contains the following sub-objects.

> Request identifier
> Nr (Realm name)
> Ns (Service name)
> Nu (User name)
> Cu (Challenge from user)
> Cs (Challenge from service)
> Ts (Timestamp from service)
> Ru (Response from user)
> Blob (optional)
> Rs (Response from service)

The value contained in most of the sub-objects matches the value

described in part two of this specification
(draft-petke-mech-00.txt).

The request identifier contains arbitrary data that is not interpreted by the deity; it is simply echoed in a response to provide a way for the requesting service to match requests and responses.

The blob contains additional information about the request, and is described below. Usually, it will be omitted or null, i.e., zero-length.

Rs is calculated as MD5(Ps + Z + M + Ps), where M is the request shown above, octet by octet, from the type octet for the message object itself through the last length octet of the length field of the Rs object. Thus, it serves to protect the entire request, including its structure, length, etc., and is a different calculation from that shown in the authentication document.

4.2 Authentication response, affirmative

An affirmative response indicates that the username is recognized, and is indeed the user you're talking to.

Request identifier
Canonical Nu (User name, case corrected)
Kuss (Key obscured for service)
Kusu (Key obscured for user)
Au (Authentication value for user)
Blob (optional)
As (Authentication value for service)

The response contains the canonical username in the desired case; this is not the same object type as Nu in the request. In an environment that is not case sensitive, this is the preferred form of the name, which might differ from the name in the request.

The blob may contain additional information about the username; see below.

As is calculated as

MD5(Ps + Z + Ns + Nu + Nr + Kuss + Cs + Cu + Ts + Kus + M + Ps)

where M is the request shown above, octet by octet, from the type octet for the containing object through the last octet of the length field of the As object, inclusive. This serves to protect the entire request, and differs from the calculation in the authentication document by the addition of the message contents as shown. Note that the Nu mentioned as the third component in the formula is the originally specified username, not the altered-case version in the

response message.

Beware that an affirmative response does not necessarily mean that it is reasonable to provide service to the user. Often, there are criteria beyond a "yes" answer, which could mean anything from "it's a valid user" to "it's a valid user but not billable" to "it's an account that was signed up five minutes ago and we haven't had a chance to look at it yet."

Typically, the authentication deity applies criteria appropriate to the requesting service. For example, if the service doesn't want to allow "free" users, the authentication deity would be configured to return a no-service response for such a user. Alternatively, the deity could be configured to provide an affirmative response but include information in the blob that would permit the service to distinguish "free" from "paying" users and treat them differently.

4.3 Authentication response, no service

The no-service response is an indication that the user is whom he claims to be, but you should not provide service to him for one reason or another. For example, he might be a "free" user but your service is provided only to paying accounts; his billing choices might not include your service; or Customer Service might be waiting for him to provide a new credit card number.

The authentication deity's configuration for this particular service determines the criteria applied by the deity when making the decision to reply affirmative or no service.

Request identifier
Canonical Nu (User name, case corrected)
Kuss (Key obscured for service)
Kusu (Key obscured for user)
Au (Authentication value for user)
Blob (optional)
As (Authentication value for service)

The contents of this object are identical to those for an affirmative response, but the service would not normally use the keys or Au values. The blob might include information useful in distinguishing the reason for the no service response, if appropriate for this service.

4.4 Authentication response, negative

A negative response means the user is not who he says he is. Whether there is such a username, but that's not who you're talking to; there is such a username, but it is not an enabled account; or there is no

such username, is not specified.

Request identifier
Blob (optional)
As (Authentication value for service)

As is calculated as MD5(Ps + Z + M + Ps). The message may contain a blob if there is additional information about the problem, e.g., for logging, but it may be omitted.

4.5 Authentication response, invalid service

An invalid request response means the request could not be processed because you (the service) are not whom you claim to be, based on your apparently not knowing the service's pass phrase or based on any other kind of authentication checking done by the deity.

Request identifier
Blob (optional)

The blob, if present, contains information that allows the deity administrators to trace the problem. There is no As field, because there is no shared secret to authenticate the response. This presents some obvious denial of service issues.

4.6 Authentication response, problem

A "problem" response indicates that the request could not be processed for some reason. This could indicate a failure in the system, an unparsable request, or a request for a realm that isn't handled by this deity.

Request identifier
Blob (optional)
As (optional)

The blob may contain information that allows the deity administrators to trace the problem. As might or might not be present, depending on the nature of the problem, i.e., whether there is a known shared secret with the server; if present, it is calculated as MD5(Ps + Z + M + Ps).

5. Object types

The following types of objects are defined in this protocol. These object types apply to the messages themselves and objects contained in messages. These types do not apply to the contents of the blob.

[Numbers for the object type field are indicated for each type, but are not necessarily accurate in this draft of the document.]

Authentication request--type 1--The request to the authentication deity. Its contents consist of a sequence of other objects as described elsewhere in this document.

Authentication response, affirmative--type 2

Authentication response, no service--type 3

Authentication response, negative--type 4

Authentication response, invalid service--type 5

Authentication response, problem--type 6

Request identifier--type 128--A request must contain an identifier to assist in matching replies to requests. This identifier is opaque to the deity, and is simply echoed in the reply, so its value is defined only by the requesting entity. The value should, of course, be unique for each request, but it is otherwise meaningless. It may be of any length.

Realm name--type 129--The name of the realm in which the user's and service's identities exist. This is included in the request to allow a deity to serve more than one realm. The value consists of the name in Unicode, in big-endian order. There is no terminating null character, and the realm is generally treated as being case insensitive. For example, the realm aol.com might look like this:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   10000001    |   00000000    |   00001110    |   00000000    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   01100001    |   00000000    |   01101111    |   00000000    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   01101100    |   00000000    |   00101110    |   00000000    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   01100011    |   00000000    |   01101111    |   00000000    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   01101101    |
+-+-+-+-+-+-+-+-+
```

That's type 129, fourteen octets follow, and the big-endian Unicode representation of the seven characters aol.com.

Service name--type 130--The name of the service in big-endian Unicode.

User name--type 131--The name of the user in big-endian Unicode, e.g., gsb.

User challenge--type 132--The user's challenge, a sequence of random octets. The length is not bounded by the protocol, but the deity will impose length restrictions, e.g., a minimum and maximum length. All bit patterns are legal in the challenge.

Service challenge--type 133--The service's challenge, a sequence of random octets. The length is not bounded by the protocol, but the deity will impose length restrictions, e.g., a minimum and maximum length. All bit patterns are legal in the challenge.

Time stamp--type 134--The time stamp, containing 14 octets with the value specified in part two of this specification (draft-petke-mech-00.txt).

User's response--type 135--The user's response, containing 16 octets with the value specified in part two of this specification (draft-petke-mech-00.txt). This is a binary value, so any bit pattern is possible in this value.

Service's response--type 136--The service's response, calculated as described elsewhere in this document. This is a binary value, so any bit pattern is possible in this value.

Key obscured for user--type 137--The key for the user, containing 16 octets as described in part two of this specification (draft-petke-mech-00.txt). This is a binary value, so any bit pattern is possible in this value.

Key obscured for service--type 138--The key for the service, containing 16 octets as described in part two of this specification (draft-petke-mech-00.txt). This is a binary value, so any bit pattern is possible in this value.

Authentication proof for user--type 139--The authentication proof, Au, for the user, containing 16 octets as described in part two of this specification (draft-petke-mech-00.txt). This is a binary value, so any bit pattern is possible in this value.

Authentication proof for service--type 140--The authentication proof, As, for the service, containing 16 octets calculated as described elsewhere in this document (not as described in part two of this specification). This is a binary value, so any bit pattern is possible in this value.

Canonical user name--type 141--The username adjusted to canonical case, in big-endian Unicode.

Blob--type 142--Deity-specific request and response information.

6. The blob

The blob consists of a sequence of objects that contain information about the user's account, or indicate, by their presence or absence, some characteristic of the user's account. Note that the use of any particular object is a function of the deity's configuration for a particular service.

Consider, for example, a "free" account in an environment where services are normally provided for a price. There are three most-likely possibilities for how the deity would handle a free account when a particular service asks the deity to authenticate a user:

* If the account is free, return an affirmative response.

* If the account is free, return an affirmative response and include a "free" indicator in the blob.

* If the account is free, return a no-service response.

The first alternative would be appropriate for a service that should provide service to free users, when it's none of the service's business whether the user is paying or not.

The second alternative would be appropriate for a service that should provide service to free users, but should know that it's doing so, e.g., to provide a different class of service to free users than not-free users.

The third alternative would be appropriate for a service that should not provide service to free users. In that case, the deity might include a free indicator in the blob, to note the reason why.

It's difficult to conceive of a truly general, standard blob format; that's why we called it a "blob." Therefor, a service-deity pair may define the blob in any way they wish. However, we define one type of sub-object that contains textual attribute/value pairs, to provide a standard encoding for one common need.

Type 1--Attribute/value pairs

If the blob contains a type-1 object, that object is composed of a sequence of textual attribute/value pairs, where the value is optional: sometimes, the presence or absence of an attribute is significant, with no need for a corresponding value. An attribute consists of a sequence of Unicode characters in the syntax

```
attribute ["=" value]
```

i.e., the attribute name optionally followed by an '=' character (code 003D) and a value. All characters are taken from the Unicode character set and stored in big-endian byte order. The attribute name may consist of any characters except a null or equals sign; the value may consist of any characters except a null.

To include more than one attribute, use a null character (code 0000) as a separator. A null character following the last attribute is optional, and may be omitted.

```
L"free\0language=en"
```

is a C-syntax example of a pair of attributes, the first with no value.

Attribute names are meaningful only in the context of a particular service-deity pair. They may be case sensitive or not as appropriate.

> [Perhaps there should be two blocks of named attributes, one with standard-defined attributes and one with deity-specific attributes?]

7. Security Considerations

This entire document is about security.

8. Author's Address

Gary S. Brown
CompuServe Incorporated
5000 Britton Rd
P.O. Box 5000
Hilliard OH 43026-5000
USA

+1 614 723 1127
<gsb@csi.compuserve.com>