A Generalized Framework for Kerberos Pre-Authentication
draft-ietf-krb-wg-preauth-framework-06

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on January 9, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

Kerberos is a protocol for verifying the identity of principals (e.g., a workstation user or a network server) on an open network. The Kerberos protocol provides a mechanism called pre-authentication for proving the identity of a principal and for better protecting the long-term secret of the principal.

This document describes a model for Kerberos pre-authentication
mechanisms. The model describes what state in the Kerberos request a pre-authentication mechanism is likely to change. It also describes how multiple pre-authentication mechanisms used in the same request will interact.

This document also provides common tools needed by multiple pre-authentication mechanisms. One of these tools is a secure channel between the client and the KDC with a reply key delivery mechanism; this secure channel can be used to protect the authentication exchange thus eliminate offline dictionary attacks. With these tools, it is relatively straightforward to chain multiple authentication mechanisms, utilize a different key management system, or support a new key agreement algorithm.
Table of Contents

1. Introduction ................................................. 4
2. Conventions and Terminology Used in This Document ............... 5
3. Model for Pre-Authentication ................................. 5
   3.1. Information Managed by the Pre-authentication Model ........ 6
   3.2. Initial Pre-authentication Required Error ..................... 8
   3.3. Client to KDC ........................................... 9
   3.4. KDC to Client ........................................... 10
4. Pre-Authentication Facilities ................................ 10
   4.1. Client-authentication Facility ............................. 12
   4.2. Strengthening-reply-key Facility ............................. 12
   4.3. Replacing-reply-key Facility ............................... 13
   4.4. KDC-authentication Facility ................................. 14
5. Requirements for Pre-Authentication Mechanisms .................. 14
6. Tools for Use in Pre-Authentication Mechanisms .................. 15
   6.1. Combining Keys ........................................... 15
   6.2. Protecting Requests/Responses ............................... 16
   6.3. Managing States for the KDC ................................. 17
   6.4. Pre-authentication Set ..................................... 19
   6.5. Definition of Kerberos FAST Padata ......................... 21
      6.5.1. FAST Armors .......................................... 22
      6.5.2. FAST Request ......................................... 23
      6.5.3. FAST Response ......................................... 27
      6.5.4. Authenticated Kerberos Error Messages using Kerberos FAST ................................... 29
      6.5.5. The Authenticated Timestamp FAST Factor ................. 30
   6.6. Authentication Strength Indication .......................... 32
7. IANA Considerations .......................................... 33
8. Security Considerations ....................................... 33
9. Acknowledgements ............................................. 34
10. References .................................................. 34
    10.1. Normative References ..................................... 34
    10.2. Informative References ................................... 34
11. Appendix A. ASN.1 module ..................................... 35
12. Authors’ Addresses ........................................... 38
13. Intellectual Property and Copyright Statements ................ 39
1. Introduction

The core Kerberos specification [RFC4120] treats pre-authentication data as an opaque typed hole in the messages to the KDC that may influence the reply key used to encrypt the KDC reply. This generality has been useful: pre-authentication data is used for a variety of extensions to the protocol, many outside the expectations of the initial designers. However, this generality makes designing more common types of pre-authentication mechanisms difficult. Each mechanism needs to specify how it interacts with other mechanisms. Also, problems like combining a key with the long-term secret or proving the identity of the user are common to multiple mechanisms. Where there are generally well-accepted solutions to these problems, it is desirable to standardize one of these solutions so mechanisms can avoid duplication of work. In other cases, a modular approach to these problems is appropriate. The modular approach will allow new and better solutions to common pre-authentication problems to be used by existing mechanisms as they are developed.

This document specifies a framework for Kerberos pre-authentication mechanisms. It defines the common set of functions that pre-authentication mechanisms perform as well as how these functions affect the state of the request and reply. In addition several common tools needed by pre-authentication mechanisms are provided. Unlike [RFC3961], this framework is not complete--it does not describe all the inputs and outputs for the pre-authentication mechanisms. Pre-Authentication mechanism designers should try to be consistent with this framework because doing so will make their mechanisms easier to implement. Kerberos implementations are likely to have plugin architectures for pre-authentication; such architectures are likely to support mechanisms that follow this framework plus commonly used extensions.

One of these common tools is the flexible authentication secure tunneling (FAST) padata type. FAST provides a protected channel between the client and the KDC, and it can optionally deliver a reply key within the protected channel. Based on FAST, pre-authentication mechanisms can extend Kerberos with ease, to support, for example, password authenticated key exchange (PAKE) protocols with zero knowledge password proof (ZKPP) [EKE] [IEEE1363.2]. Any pre-authentication mechanism can be encapsulated in the FAST messages as defined in Section 6.5. A pre-authentication type carried within FAST is called a FAST factor. Creating a FAST factor is the easiest path to create a new pre-authentication mechanism. FAST factors are significantly easier to analyze from a security standpoint than other pre-authentication mechanisms.

Mechanism designers should design FAST factors, instead of new pre-
authentication mechanisms outside of FAST.

2. Conventions and Terminology Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The word padata is used as a shorthand for pre-authentication data.

A conversation is the set of all authentication messages exchanged between the client and the KDCs in order to authenticate the client principal. A conversation as defined here consists of all messages that are necessary to complete the authentication between the client and the KDC.

Lastly, this document should be read only after reading the documents describing the Kerberos cryptography framework [RFC3961] and the core Kerberos protocol [RFC4120]. This document may freely use terminology and notation from these documents without reference or further explanation.

3. Model for Pre-Authentication

When a Kerberos client wishes to obtain a ticket using the authentication server, it sends an initial Authentication Service (AS) request. If pre-authentication is required but not being used, then the KDC will respond with a KDC_ERR_PREAUTH_REQUIRED error. Alternatively, if the client knows what pre-authentication to use, it MAY optimize away a round-trip and send an initial request with padata included in the initial request. If the client includes the padata computed using the wrong pre-authentication mechanism or incorrect keys, the KDC MAY return KDC_ERR_PREAUTH_FAILED with no indication of what padata should have been included. In that case, the client MUST retry with no padata and examine the error data of the KDC_ERR_PREAUTH_REQUIRED error. If the KDC includes pre-authentication information in the accompanying error data of KDC_ERR_PREAUTH_FAILED, the client SHOULD process the error data, and then retry.

The conventional KDC maintains no state between two requests; subsequent requests may even be processed by a different KDC. On the other hand, the client treats a series of exchanges with KDCs as a single conversation. Each exchange accumulates state and hopefully brings the client closer to a successful authentication.
These models for state management are in apparent conflict. For many of the simpler pre-authentication scenarios, the client uses one round trip to find out what mechanisms the KDC supports. Then the next request contains sufficient pre-authentication for the KDC to be able to return a successful reply. For these simple scenarios, the client only sends one request with pre-authentication data and so the conversation is trivial. For more complex conversations, the KDC needs to provide the client with a cookie to include in future requests to capture the current state of the authentication session. Handling of multiple round-trip mechanisms is discussed in Section 6.3.

This framework specifies the behavior of Kerberos pre-authentication mechanisms used to identify users or to modify the reply key used to encrypt the KDC reply. The PA-DATA typed hole may be used to carry extensions to Kerberos that have nothing to do with proving the identity of the user or establishing a reply key. Such extensions are outside the scope of this framework. However mechanisms that do accomplish these goals should follow this framework.

This framework specifies the minimum state that a Kerberos implementation needs to maintain while handling a request in order to process pre-authentication. It also specifies how Kerberos implementations process the padata at each step of the AS request process.

3.1. Information Managed by the Pre-authentication Model

The following information is maintained by the client and KDC as each request is being processed:

- The reply key used to encrypt the KDC reply
- How strongly the identity of the client has been authenticated
- Whether the reply key has been used in this conversation
- Whether the reply key has been replaced in this conversation
- Whether the contents of the KDC reply can be verified by the client principal

Conceptually, the reply key is initially the long-term key of the principal. However, principals can have multiple long-term keys because of support for multiple encryption types, salts and string2key parameters. As described in Section 5.2.7.5 of the Kerberos protocol [RFC4120], the KDC sends PA-ETYPE-INFO2 to notify...
the client what types of keys are available. Thus in full
generality, the reply key in the pre-authentication model is actually
a set of keys. At the beginning of a request, it is initialized to
the set of long-term keys advertised in the PA-ETYPE-INFO2 element on
the KDC. If multiple reply keys are available, the client chooses
which one to use. Thus the client does not need to treat the reply
key as a set. At the beginning of a request, the client picks a
reply key to use.

KDC implementations MAY choose to offer only one key in the PA-ETYPE-
INFO2 element. Since the KDC already knows the client’s list of
supported enctypes from the request, no interoperability problems are
created by choosing a single possible reply key. This way, the KDC
implementation avoids the complexity of treating the reply key as a
set.

When the padata in the request is verified by the KDC, then the
client is known to have that key, therefore the KDC SHOULD pick the
same key as the reply key.

At the beginning of handling a message on both the client and the
KDC, the client’s identity is not authenticated. A mechanism may
indicate that it has successfully authenticated the client’s
identity. This information is useful to keep track of on the client
in order to know what pre-authentication mechanisms should be used.
The KDC needs to keep track of whether the client is authenticated
because the primary purpose of pre-authentication is to authenticate
the client identity before issuing a ticket. The handling of
authentication strength using various authentication mechanisms is
discussed in Section 6.6.

Initially the reply key has not been used. A pre-authentication
mechanism that uses the reply key to encrypt or checksum some data in
the generation of new keys MUST indicate that the reply key is used.
This state is maintained by the client and the KDC to enforce the
security requirement stated in Section 4.3 that the reply key cannot
be replaced after it is used.

Initially the reply key has not been replaced. If a mechanism
implements the Replace Reply Key facility discussed in Section 4.3,
then the state MUST be updated to indicate that the reply key has
been replaced. Once the reply key has been replaced, knowledge of
the reply key is insufficient to authenticate the client. The reply
key is marked replaced in exactly the same situations as the KDC
reply is marked as not being verified to the client principal.
However, while mechanisms can verify the KDC reply to the client,
once the reply key is replaced, then the reply key remains replaced
for the remainder of the conversation.
Without pre-authentication, the client knows that the KDC reply is authentic and has not been modified because it is encrypted in a long-term key of the client. Only the KDC and the client know that key. So at the start of handling any message the KDC reply is presumed to be verified using the client principal’s long-term key. Any pre-authentication mechanism that sets a new reply key not based on the principal’s long-term secret MUST either verify the KDC reply some other way or indicate that the reply is not verified. If a mechanism indicates that the reply is not verified then the client implementation MUST return an error unless a subsequent mechanism verifies the reply. The KDC needs to track this state so it can avoid generating a reply that is not verified.

The typical Kerberos request does not provide a way for the client machine to know that it is talking to the correct KDC. Someone who can inject packets into the network between the client machine and the KDC and who knows the password that the user will give to the client machine can generate a KDC reply that will decrypt properly. So, if the client machine needs to authenticate that the user is in fact the named principal, then the client machine needs to do a TGS request for itself as a service. Some pre-authentication mechanisms may provide a way for the client to authenticate the KDC. Examples of this include signing the reply that can be verified using a well-known public key or providing a ticket for the client machine as a service.

3.2. Initial Pre-authentication Required Error

Typically a client starts a conversation by sending an initial request with no pre-authentication. If the KDC requires pre-authentication, then it returns a KDC_ERR_PREAUTH_REQUIRED message. After the first reply with the KDC_ERR_PREAUTH_REQUIRED error code, the KDC returns the error code KDC_ERR_MORE_PREAUTH_DATA_NEEDED (defined in Section 6.3) for pre-authentication configurations that use multi-round-trip mechanisms; see Section 3.4 for details of that case.

The KDC needs to choose which mechanisms to offer the client. The client needs to be able to choose what mechanisms to use from the first message. For example consider the KDC that will accept mechanism A followed by mechanism B or alternatively the single mechanism C. A client that supports A and C needs to know that it should not bother trying A.

Mechanisms can either be sufficient on their own or can be part of an authentication set---a group of mechanisms that all need to successfully complete in order to authenticate a client. Some mechanisms may only be useful in authentication sets; others may be
useful alone or in authentication sets. For the second group of mechanisms, KDC policy dictates whether the mechanism will be part of an authentication set or offered alone. For each mechanism that is offered alone, the KDC includes the pre-authentication type ID of the mechanism in the padata sequence returned in the KDC_ERR_PREAUTH_REQUIRED error.

The KDC SHOULD NOT send data that is encrypted in the long-term password-based key of the principal. Doing so has the same security exposures as the Kerberos protocol without pre-authentication. There are few situations where pre-authentication is desirable and where the KDC needs to expose cipher text encrypted in a weak key before the client has proven knowledge of that key.

3.3. Client to KDC

This description assumes that a client has already received a KDC_ERR_PREAUTH_REQUIRED from the KDC. If the client performs optimistic pre-authentication then the client needs to optimistically guess values for the information it would normally receive from that error response.

The client starts by initializing the pre-authentication state as specified. It then processes the padata in the KDC_ERR_PREAUTH_REQUIRED.

When processing the response to the KDC_ERR_PREAUTH_REQUIRED, the client MAY ignore any padata it chooses unless doing so violates a specification to which the client conforms. Clients conforming to this specification MUST NOT ignore the padata defined in Section 6.3. Clients SHOULD process padata unrelated to this framework or other means of authenticating the user. Clients SHOULD choose one authentication set or mechanism that could lead to authenticating the user and ignore the rest. Since the list of mechanisms offered by the KDC is in the decreasing preference order, clients typically choose the first mechanism or authentication set that the client can usefully perform. If a client chooses to ignore a padata it MUST NOT process the padata, allow the padata to affect the pre-authentication state, nor respond to the padata.

For each padata the client chooses to process, the client processes the padata and modifies the pre-authentication state as required by that mechanism. Padata are processed in the order received from the KDC.

After processing the padata in the KDC error, the client generates a new request. It processes the pre-authentication mechanisms in the order in which they will appear in the next request, updating the
state as appropriate. The request is sent when it is complete.

3.4. KDC to Client

When a KDC receives an AS request from a client, it needs to determine whether it will respond with an error or an AS reply. There are many causes for an error to be generated that have nothing to do with pre-authentication; they are discussed in the core Kerberos specification.

From the standpoint of evaluating the pre-authentication, the KDC first starts by initializing the pre-authentication state. It then processes the padata in the request. As mentioned in Section 3.3, the KDC MAY ignore padata that is inappropriate for the configuration and MUST ignore padata of an unknown type.

At this point the KDC decides whether it will issue a pre-authentication required error or a reply. Typically a KDC will issue a reply if the client’s identity has been authenticated to a sufficient degree.

In the case of a KDC_ERR_MORE_PREAUTH_DATA_NEEDED error, the KDC first starts by initializing the pre-authentication state. Then it processes any padata in the client’s request in the order provided by the client. Mechanisms that are not understood by the KDC are ignored. Mechanisms that are inappropriate for the client principal or the request SHOULD also be ignored. Next, it generates padata for the error response, modifying the pre-authentication state appropriately as each mechanism is processed. The KDC chooses the order in which it will generate padata (and thus the order of padata in the response), but it needs to modify the pre-authentication state consistently with the choice of order. For example, if some mechanism establishes an authenticated client identity, then the subsequent mechanisms in the generated response receive this state as input. After the padata is generated, the error response is sent. Typically the errors with the code KDC_ERR_MORE_PREAUTH_DATA_NEEDED in a conversation will include KDC state as discussed in Section 6.3.

To generate a final reply, the KDC generates the padata modifying the pre-authentication state as necessary. Then it generates the final response, encrypting it in the current pre-authentication reply key.

4. Pre-Authentication Facilities

Pre-Authentication mechanisms can be thought of as providing various conceptual facilities. This serves two useful purposes. First,
mechanism authors can choose only to solve one specific small problem. It is often useful for a mechanism designed to offer key management not to directly provide client authentication but instead to allow one or more other mechanisms to handle this need. Secondly, thinking about the abstract services that a mechanism provides yields a minimum set of security requirements that all mechanisms providing that facility must meet. These security requirements are not complete; mechanisms will have additional security requirements based on the specific protocol they employ.

A mechanism is not constrained to only offering one of these facilities. While such mechanisms can be designed and are sometimes useful, many pre-authentication mechanisms implement several facilities. By combining multiple facilities in a single mechanism, it is often easier to construct a secure, simple solution than by solving the problem in full generality. Even when mechanisms provide multiple facilities, they need to meet the security requirements for all the facilities they provide. If the FAST factor approach is used, it is likely that one or a small number of facilities can be provided by a single mechanism without complicating the security analysis.

According to Kerberos extensibility rules (Section 1.5 of the Kerberos specification [RFC4120]), an extension MUST NOT change the semantics of a message unless a recipient is known to understand that extension. Because a client does not know that the KDC supports a particular pre-authentication mechanism when it sends an initial request, a pre-authentication mechanism MUST NOT change the semantics of the request in a way that will break a KDC that does not understand that mechanism. Similarly, KDCs MUST NOT send messages to clients that affect the core semantics unless the client has indicated support for the message.

The only state in this model that would break the interpretation of a message is changing the expected reply key. If one mechanism changed the reply key and a later mechanism used that reply key, then a KDC that interpreted the second mechanism but not the first would fail to interpret the request correctly. In order to avoid this problem, extensions that change core semantics are typically divided into two parts. The first part proposes a change to the core semantic—for example proposes a new reply key. The second part acknowledges that the extension is understood and that the change takes effect. Section 4.2 discusses how to design mechanisms that modify the reply key to be split into a proposal and acceptance without requiring additional round trips to use the new reply key in subsequent pre-authentication. Other changes in the state described in Section 3.1 can safely be ignored by a KDC that does not understand a mechanism. Mechanisms that modify the behavior of the request outside the scope
of this framework need to carefully consider the Kerberos
extensibility rules to avoid similar problems.

4.1. Client-authentication Facility

The client authentication facility proves the identity of a user to
the KDC before a ticket is issued. Examples of mechanisms
implementing this facility include the encrypted timestamp facility
defined in Section 5.2.7.2 of the Kerberos specification [RFC4120].
Mechanisms that provide this facility are expected to mark the client
as authenticated.

Mechanisms implementing this facility SHOULD require the client to
prove knowledge of the reply key before transmitting a successful KDC
reply. Otherwise, an attacker can intercept the pre-authentication
exchange and get a reply to attack. One way of proving the client
knows the reply key is to implement the Replace Reply Key facility
along with this facility. The PKINIT mechanism [RFC4556] implements
Client Authentication alongside Replace Reply Key.

If the reply key has been replaced, then mechanisms such as
encrypted-timestamp that rely on knowledge of the reply key to
authenticate the client MUST NOT be used.

4.2. Strengthening-reply-key Facility

Particularly, when dealing with keys based on passwords, it is
desirable to increase the strength of the key by adding additional
secrets to it. Examples of sources of additional secrets include the
results of a Diffie-Hellman key exchange or key bits from the output
of a smart card [KRB-WG.SAM]. Typically these additional secrets can
be first combined with the existing reply key and then converted to a
protocol key using tools defined in Section 6.1.

If a mechanism implementing this facility wishes to modify the reply
key before knowing that the other party in the exchange supports the
mechanism, it proposes modifying the reply key. The other party then
includes a message indicating that the proposal is accepted if it is
understood and meets policy. In many cases it is desirable to use
the new reply key for client authentication and for other facilities.
Waiting for the other party to accept the proposal and actually
modify the reply key state would add an additional round trip to the
exchange. Instead, mechanism designers are encouraged to include a
typed hole for additional padata in the message that proposes the
reply key change. The padata included in the typed hole are
generated assuming the new reply key. If the other party accepts the
proposal, then these padata are considered as an inner level. As
with the outer level, one authentication set or mechanism is
typically chosen for client authentication, along with auxiliary mechanisms such as KDC cookies, and other mechanisms are ignored. Containers like this need more thought. For example if you are constructing an authentication set do you expect to use a strengthen reply key mechanism in conjunction with something else, do you include the something else in the hint of the strengthen mechanism or as its own entry. It’s easier to configure and express the authentication set as its own entry. However if you do that the composition of the mechanisms looks in practice than it appears in the authentication set.]} The party generating the proposal can determine whether the padata were processed based on whether the proposal for the reply key is accepted.

The specific formats of the proposal message, including where padata are included is a matter for the mechanism specification. Similarly, the format of the message accepting the proposal is mechanism-specific.

Mechanisms implementing this facility and including a typed hole for additional padata MUST checksum that padata using a keyed checksum or encrypt the padata. This requirement protects against modification of the contents of the typed hole. By modifying these contents an attacker might be able to choose which mechanism is used to authenticate the client, or to convince a party to provide text encrypted in a key that the attacker had manipulated. It is important that mechanisms strengthen the reply key enough that using it to checksum padata is appropriate.

4.3. Replacing-reply-key Facility

The Replace Reply Key facility replaces the key in which a successful AS reply will be encrypted. This facility can only be used in cases where knowledge of the reply key is not used to authenticate the client. The new reply key MUST be communicated to the client and the KDC in a secure manner. Mechanisms implementing this facility MUST mark the reply key as replaced in the pre-authentication state. Mechanisms implementing this facility MUST either provide a mechanism to verify the KDC reply to the client or mark the reply as unverified in the pre-authentication state. Mechanisms implementing this facility SHOULD NOT be used if a previous mechanism has used the reply key.

As with the strengthening-reply-key facility, Kerberos extensibility rules require that the reply key not be changed unless both sides of the exchange understand the extension. In the case of this facility it will likely be more common for both sides to know that the facility is available by the time that the new key is available to be used. However, mechanism designers can use a container for padata in
a proposal message as discussed in Section 4.2 if appropriate.

4.4. KDC-authentication Facility

This facility verifies that the reply comes from the expected KDC. In traditional Kerberos, the KDC and the client share a key, so if the KDC reply can be decrypted then the client knows that a trusted KDC responded. Note that the client machine cannot trust the client unless the machine is presented with a service ticket for it (typically the machine can retrieve this ticket by itself). However, if the reply key is replaced, some mechanism is required to verify the KDC. Pre-authentication mechanisms providing this facility allow a client to determine that the expected KDC has responded even after the reply key is replaced. They mark the pre-authentication state as having been verified.

5. Requirements for Pre-Authentication Mechanisms

This section lists requirements for specifications of pre-authentication mechanisms.

For each message in the pre-authentication mechanism, the specification describes the pa-type value to be used and the contents of the message. The processing of the message by the sender and recipient is also specified. This specification needs to include all modifications to the pre-authentication state.

Generally mechanisms have a message that can be sent in the error data of the KDC_ERR_PREAUTH_REQUIRED error message or in an authentication set. If the client needs information such as trusted certificate authorities in order to determine if it can use the mechanism, then this information should be in that message. In addition, such mechanisms should also define a pa-hint to be included in authentication sets. Often, the same information included in the padata-value is appropriate to include in the pa-hint (as defined in Section 6.4).

In order to ease security analysis the mechanism specification should describe what facilities from this document are offered by the mechanism. For each facility, the security consideration section of the mechanism specification should show that the security requirements of that facility are met. This requirement is applicable to any FAST factor that provides authentication information.

Significant problems have resulted in the specification of Kerberos protocols because much of the KDC exchange is not protected against
authentication. The security considerations section should discuss unauthenticated plaintext attacks. It should either show that plaintext is protected or discuss what harm an attacker could do by modifying the plaintext. It is generally acceptable for an attacker to be able to cause the protocol negotiation to fail by modifying plaintext. More significant attacks should be evaluated carefully.

As discussed in Section 6.3, there is no guarantee that a client will use the same KDCs for all messages in a conversation. The mechanism specification needs to show why the mechanism is secure in this situation. The hardest problem to deal with, especially for challenge/response mechanisms is to make sure that the same response cannot be replayed against two KDCs while allowing the client to talk to any KDC.

6. Tools for Use in Pre-Authentication Mechanisms

This section describes common tools needed by multiple pre-authentication mechanisms. By using these tools mechanism designers can use a modular approach to specify mechanism details and ease security analysis.

6.1. Combining Keys

Frequently a weak key needs to be combined with a stronger key before use. For example, passwords are typically limited in size and insufficiently random, therefore it is desirable to increase the strength of the keys based on passwords by adding additional secrets. Additional source of secrecy may come from hardware tokens.

This section provides standard ways to combine two keys into one.

KRB-FX-CF1() is defined to combine two pass-phrases.

\[
\text{KRB-FX-CF1(UTF-8 string, UTF-8 string)} \rightarrow (\text{UTF-8 string})
\]
\[
\text{KRB-FX-CF1(x, y)} \rightarrow x \| y
\]

Where \(\|\) denotes concatenation. The strength of the final key is roughly the total strength of the individual keys being combined assuming that the string_to_key() function [RFC3961] uses all its input evenly.

An example usage of KRB-FX-CF1() is when a device provides random but short passwords, the password is often combined with a personal identification number (PIN). The password and the PIN can be combined using KRB-FX-CF1().
KRB-FX-CF2() combines two protocol keys based on the pseudo-random() function defined in [RFC3961].

Given two input keys, K1 and K2, where K1 and K2 can be of two different enctypes, the output key of KRB-FX-CF2(), K3, is derived as follows:

\[
\text{KRB-FX-CF2}(\text{protocol key, protocol key, octet string, octet string}) \rightarrow \text{(protocol key)}
\]

\[
\text{PRF}^+ (K1, \text{pepper}_1) \rightarrow \text{octet-string-1}
\]

\[
\text{PRF}^+ (K2, \text{pepper}_2) \rightarrow \text{octet-string-2}
\]

\[
\text{KRB-FX-CF2}(K1, K2, \text{pepper}_1, \text{pepper}_2) \rightarrow \text{random-to-key(octet-string-1} \ ^ \text{octet-string-2})
\]

Where ^ denotes the exclusive-OR operation. PRF+() is defined as follows:

\[
\text{PRF}^+ (\text{protocol key, octet string}) \rightarrow \text{(octet string)}
\]

\[
\text{PRF}^+ (\text{key, shared-info}) \rightarrow \text{pseudo-random} (\text{key, 1} || \text{shared-info}) || \text{pseudo-random} (\text{key, 2} || \text{shared-info}) || \text{pseudo-random} (\text{key, 3} || \text{shared-info}) || ... 
\]

Here the counter value 1, 2, 3 and so on are encoded as a one-octet integer. The pseudo-random() operation is specified by the enctype of the protocol key. PRF+() uses the counter to generate enough bits as needed by the random-to-key() [RFC3961] function for the encryption type specified for the resulting key; unneeded bits are removed from the tail.

Mechanism designers MUST specify the values for the input parameter pepper1 and pepper2 when combining two keys using KRB-FX-CF2(). The pepper1 and pepper2 MUST be distinct so that if the two keys being combined are the same, the resulting key is not a trivial key.

### 6.2. Protecting Requests/Responses

Mechanism designers SHOULD protect clear text portions of pre-authentication data. Various denial of service attacks and downgrade attacks against Kerberos are possible unless plaintexts are somehow protected against modification. An early design goal of Kerberos Version 5 [RFC4120] was to avoid encrypting more of the authentication exchange that was required. (Version 4 doubly-encrypted the encrypted part of a ticket in a KDC reply, for example.) This minimization of encryption reduces the load on the KDC and busy servers. Also, during the initial design of Version 5, the existence of legal restrictions on the export of cryptography...
made it desirable to minimize of the number of uses of encryption in the protocol. Unfortunately, performing this minimization created numerous instances of unauthenticated security-relevant plaintext fields.

If there is more than one roundtrip for an authentication exchange, mechanism designers need to allow either the client or the KDC to provide a checksum of all the messages exchanged on the wire in the conversation, and the checksum is then verified by the receiver.

New mechanisms MUST NOT be hard-wired to use a specific algorithm.

Primitives defined in [RFC3961] are RECOMMENDED for integrity protection and confidentiality. Mechanisms based on these primitives are crypto-agile as the result of using [RFC3961] along with [RFC4120]. The advantage afforded by crypto-agility is the ability to avoid a multi-year standardization and deployment cycle to fix a problem that is specific to a particular algorithm, when real attacks do arise against that algorithm.

Note that data used by FAST factors (defined in Section 6.5) is encrypted in a protected channel, thus they do not share the unauthenticated-text issues with mechanisms designed as full-blown pre-authentication mechanisms.

6.3. Managing States for the KDC

Kerberos KDCs are stateless. There is no requirement that clients will choose the same KDC for the second request in a conversation. Proxies or other intermediate nodes may also influence KDC selection. So, each request from a client to a KDC must include sufficient information that the KDC can regenerate any needed state. This is accomplished by giving the client a potentially long opaque cookie in responses to include in future requests in the same conversation. The KDC MAY respond that a conversation is too old and needs to restart by responding with a KDC_ERR_PREAUTH_EXPIRED error.

KDC_ERR_PREAUTH_EXPIRED TBA

When a client receives this error, the client SHOULD abort the existing conversation, and restart a new one.

An example, where more than one message from the client is needed, is when the client is authenticated based on a challenge-response scheme. In that case, the KDC needs to keep track of the challenge issued for a client authentication request.

The PA-FX-COOKIE pdata type is defined in this section to facilitate
state management. This padata is sent by the KDC when the KDC requires state for a future transaction. The client includes this opaque token in the next message in the conversation. The token may be relatively large; clients MUST be prepared for tokens somewhat larger than the size of all messages in a conversation.

\begin{verbatim}
PA_FX_COOKIE TBA
   -- Stateless cookie that is not tied to a specific KDC.
\end{verbatim}

The corresponding padata-value field \cite{RFC4120} contains the Distinguished Encoding Rules (DER) \cite{X60} \cite{X690} encoding of the following Abstract Syntax Notation One (ASN.1) type PA-FX-COOKIE:

\begin{verbatim}
PA-FX-COOKIE ::= SEQUENCE {
   conversationId [0] OCTET STRING,
      -- Contains the identifier of this conversation. This field
      -- must contain the same value for all the messages
      -- within the same conversation.
   enc-binding-key [1] EncryptedData OPTIONAL,
      -- EncryptionKey --
      -- This field is present when and only when a FAST
      -- padata as defined in Section 6.5 is included.
      -- The encrypted data, when decrypted, contains an
      -- EncryptionKey structure.
      -- This encryption key is encrypted using the armor key
      -- (defined in Section 6.5.1), and the key usage for the
      -- encryption is KEY_USAGE_FAST_BINDING_KEY.
      -- Present only once in a conversation.
   cookie [2] OCTET STRING OPTIONAL,
      -- Opaque data, for use to associate all the messages in
      -- a single conversation between the client and the KDC.
      -- This is generated by the KDC and the client MUST copy
      -- the exact cookie encapsulated in a PA_FX_COOKIE data
      -- element into the next message of the same conversation.
   ...}
\end{verbatim}

\begin{verbatim}
KEY_USAGE_FAST_BINDING_KEY TBA
\end{verbatim}

The conversationId field contains a sufficiently-long rand number that uniquely identifies the conversation. If a PA_FX_COOKIE padata is present in one message, a PA_FX_COOKIE structure MUST be present in all subsequent messages of the same conversation between the client and the KDC, with the same conversationId value.

The enc-binding-key field is present when and only when a FAST padata (defined in Section 6.5) is included. The enc-binding-key field is present only once in a conversation. It MUST be ignored if it is present in a subsequent message of the same conversation. The
encrypted data, when decrypted, contains an EncryptionKey structure that is called the binding key. The binding key is encrypted using the armor key (defined in Section 6.5.1), and the key usage for the encryption is KEY_USAGE_FAST_BINDING_KEY.

If a Kerberos FAST padata as defined in Section 6.5 is included in one message, it MUST be included in all subsequent messages of the same conversation.

When FAST padata as defined Section 6.5 is included, the PA-FX-COOKIE padata MUST be included.

The cookie token is generated by the KDC and the client MUST copy the exact cookie encapsulated in a PA_FX_COOKIE data element into the next message of the same conversation. The content of the cookie field is a local matter of the KDC. However the KDC MUST construct the cookie token in such a manner that a malicious client cannot subvert the authentication process by manipulating the token. The KDC implementation needs to consider expiration of tokens, key rollover and other security issues in token design. The content of the cookie field is likely specific to the pre-authentication mechanisms used to authenticate the client. If a client authentication response can be replayed to multiple KDCs via the PA_FX_COOKIE mechanism, an expiration in the cookie is RECOMMENDED to prevent the response being presented indefinitely.

If at least one more message for a mechanism or a mechanism set is expected by the KDC, the KDC returns a KDC_ERR_MORE_PREAUTH_DATA_NEEDED error with a PA_FX_COOKIE to identify the conversation with the client according to Section 6.5.4.

6.4. Pre-authentication Set

If all mechanisms in a group need to successfully complete in order to authenticate a client, the client and the KDC SHOULD use the PA_AUTHENTICATION_SET padata element.

A PA_AUTHENTICATION_SET padata element contains the ASN.1 DER encoding of the PA-AUTHENTICATION-SET structure:
PA-AUTHENTICATION-SET ::= SEQUENCE OF PA-AUTHENTICATION-SET-ELEM

PA-AUTHENTICATION-SET-ELEM ::= SEQUENCE {
    pa-type     [0] Int32,
        -- same as padata-type.
    pa-hint     [1] OCTET STRING,
        -- hint data.
    ...
}  

The pa-type field of the PA-AUTHENTICATION-SET-ELEM structure contains the corresponding value of padata-type in PA-DATA [RFC4120]. Associated with the pa-type is a pa-hint, which is an octet-string specified by the pre-authentication mechanism. This hint may provide information for the client which helps it determine whether the mechanism can be used. For example a public-key mechanism might include the certificate authorities it trusts in the hint info. Most mechanisms today do not specify hint info; if a mechanism does not specify hint info the KDC MUST NOT send a hint for that mechanism. To allow future revisions of mechanism specifications to add hint info, clients MUST ignore hint info received for mechanisms that the client believes do not support hint info. If a member of the pre-authentication mechanism set that requires a challenge, a separate padata that carries the challenge SHOULD be included along with the pre-authentication set padata.

The PA-AUTHENTICATION-SET appears only in the first message from the KDC to the client. In particular, the client should not be prepared for the future authentication mechanisms to change as the conversation progresses. [[anchor9: I think this is correct; we should discuss and if the WG agrees the text should reflect this.]]

When indicating which sets of pre-authentication mechanisms are supported, the KDC includes a PA-AUTHENTICATION-SET padata element for each pre-authentication mechanism set.

The client sends the padata-value for the first mechanism it picks in the pre-authentication set, when the first mechanism completes, the client and the KDC will proceed with the second mechanism, and so on until all mechanisms complete successfully. The PA_FX_COOKIE as defined in Section 6.3 MUST be sent by the KDC along with the first message that contains a PA-AUTHENTICATION-SET, in order to keep track of KDC states.

Before the authentication succeeds and a ticket is returned, the message that the client sends is an AS_REQ and the message that the KDC sends is a KRB-ERROR message. The error code in the KRB-ERROR message from the KDC is KDC_ERR_MORE_PREAUTH_DATA_NEEDED as defined
in Section 6.3 and the accompanying e-data contains the DER encoding of ASN.1 type METHOD-DATA. The KDC includes the padata elements in the METHOD-DATA. If there is no padata, the e-data field is absent in the KRB-ERROR message.

If one mechanism completes on the client side, and the client expects the KDC to send the next padata for the next pre-authentication mechanism before the authentication succeeds, the client sends an AS_REQ with a padata of type PA_FX_HEARTBEAT.

    PA_FX_HEARTBEAT                    TBA

The padata-value for the PA_FX_HEARTBEAT is empty.

If one mechanism completes on the KDC side, and the KDC expects the client to send the next padata for the next pre-authentication mechanism before the authentication succeeds, the KDC sends a KRB-ERROR message with the code KDC_ERR_MORE_PREAUTH_DATA_NEEDED and includes a padata of type PA_FX_HEARTBEAT.

6.5. Definition of Kerberos FAST Padata

As described in [RFC4120], Kerberos is vulnerable to offline dictionary attacks. An attacker can request an AS-REP and try various passwords to see if they can decrypt the resulting ticket. RFC 4120 provides the encrypted timestamp pre-authentication method that ameliorates the situation somewhat by requiring that an attacker observe a successful authentication. However stronger security is desired in many environments. The Kerberos FAST pre-authentication padata defined in this section provides a tool to significantly reduce vulnerability to offline dictionary attack. When combined with encrypted timestamp, FAST requires an attacker to mount a successful man-in-the-middle attack to observe ciphertext. When combined with host keys, FAST can even protect against active attacks. FAST also provides solutions to common problems for pre-authentication mechanisms such as binding of the request and the reply, freshness guarantee of the authentication. FAST itself, however, does not authenticate the client or the KDC, instead, it provides a typed hole to allow pre-authentication data be tunneled. A pre-authentication data element used within FAST is called a FAST factor. A FAST factor captures the minimal work required for
extending Kerberos to support a new pre-authentication scheme.

A FAST factor MUST NOT be used outside of FAST unless its specification explicitly allows so. The typed holes in FAST messages can also be used as generic holes for other padata that are not intended to prove the client’s identity, or establish the reply key.

New pre-authentication mechanisms SHOULD be designed as FAST factors, instead of full-blown pre-authentication mechanisms.

FAST factors that are pre-authentication mechanisms MUST meet the requirements in Section 5.

FAST employs an armoring scheme. The armor can be a Ticket Granting Ticket (TGT) obtained by the client’s machine using the host keys to pre-authenticate with the KDC, or an anonymous TGT obtained based on anonymous PKINIT [KRB-ANON] [RFC4556].

The rest of this section describes the types of armors and the syntax of the messages used by FAST. Conforming implementations MUST support Kerberos FAST padata.

6.5.1. FAST Armors

An armor key is used to encrypt pre-authentication data in the FAST request and the response. The KrbFastArmor structure is defined to identify the armor key. This structure contains the following two fields: the armor-type identifies the type of armors, and the armor-value as an OCTET STRING contains the description of the armor scheme and the armor key.

KrbFastArmor ::= SEQUENCE {
  armor-type   [0] Int32,
    -- Type of the armor.
  armor-value  [1] OCTET STRING,
    -- Value of the armor.
  ...
}

The value of the armor key is a matter of the armor type specification. Only one armor type is defined in this document.

FX_FAST_ARMOR_AP_REQUEST TBA

The FX_FAST_ARMOR_AP_REQUEST armor is based on Kerberos tickets.

Conforming implementations MUST implement the FX_FAST_ARMOR_AP_REQUEST armor type.
6.5.1.1. Ticket-based Armors

This is a ticket-based armoring scheme. The armor-type is FX_FAST_ARMOR_AP_REQUEST, the armor-value contains an ASN.1 DER encoded AP-REQ. The ticket in the AP-REQ is called an armor ticket or an armor TGT. The subkey field in the AP-REQ MUST be present. The armor key is the subkey in the AP-REQ authenticator.

The server name field of the armor ticket MUST identify the TGS of the target realm. Here are three ways in the decreasing preference order how an armor TGT SHOULD be obtained:

1. If the client is authenticating from a host machine whose Kerberos realm has a trust path to the client’s realm, the host machine obtains a TGT by pre-authenticating intiitally the realm of the host machine using the host keys. If the client’s realm is different than the realm of the local host, the machine then obtains a cross-realm TGT to the client’s realm as the armor ticket. Otherwise, the host’s primary TGT is the armor ticket.

2. If the client’s host machine cannot obtain a host ticket strictly based on RFC4120, but the KDC has an asymmetric signing key that the client can verify the binding between the public key of the signing key and the expected KDC, the client can use anonymous PKINIT [KRB-ANON] [RFC4556] to authenticate the KDC and obtain an anonymous TGT as the armor ticket. The armor key can be a cross-team TGT obtained based on the initial primary TGT obtained using anonymous PKINIT with KDC authentication.

3. Otherwise, the client uses anonymous PKINIT to get an anonymous TGT without KDC authentication and that TGT is the armor ticket. Note that this mode of operation is vulnerable to man-in-the-middle attacks at the time of obtaining the initial anonymous armor TGT. The armor key can be a cross-team TGT obtained based on the initial primary TGT obtained using anonymous PKINIT without KDC authentication.

Because the KDC does not know if the client is able to trust the ticket it has, the KDC MUST initialize the pre-authentication state to an unverified KDC.

6.5.2. FAST Request

A padata type PA_FX_FAST is defined for the Kerberos FAST pre-authentication padata. The corresponding padata-value field [RFC4120] contains the DER encoding of the ASN.1 type PA-FX-FAST-REQUEST.
The PA-FX-FAST-REQUEST structure contains a KrbFastArmoredReq type. The KrbFastArmoredReq encapsulates the encrypted padata.

The enc-fast-req field contains an encrypted KrbFastReq structure. The armor key is used to encrypt the KrbFastReq structure, and the key usage number for that encryption is KEY_USAGE_FAST_ARMOR.

The armor key is selected as follows:

- In an AS request, the armor field in the KrbFastArmoredReq structure MUST be present and the armor key is identified according to the specification of the armor type.

- In a TGS request, the armor field in the KrbFastArmoredReq structure MUST NOT be present and the subkey in the AP-REQ authenticator in the PA-TGS-REQ PA-DATA MUST be present.
case, the armor key is that subkey in the AP-REQ authenticator.

The req-checksum field contains a checksum that is performed over the
type KDC-REQ-BODY for the req-body field of the KDC-REQ [RFC4120]
structure of the containing message. The checksum key is the armor
key, and the checksum type is the required checksum type for the
tenotype of the armor key per [RFC3961]. [[anchor12: Is this checksum
still needed if we include a full kdc-req-body]]

The KrbFastReq structure contains the following information:

KrbFastReq ::= SEQUENCE {
  fast-options [0] FastOptions,
    -- Additional options.
  padata       [1] SEQUENCE OF PA-DATA,
    -- padata typed holes.
  req-body     [2] KDC-REQ-BODY,
    -- Contains the KDC request body as defined in Section
    -- 5.4.1 of [RFC4120]. The req-body field in the KDC-REQ
    -- structure [RFC4120] MUST be ignored.
    -- The client name and realm in the KDC-REQ [RFC4120]
    -- MUST NOT be present for AS-REQ and TGS-REQ when
    -- Kerberos FAST padata is included in the request.
    ...  
}

[[anchor13: See mailing list discussion about whether client name
absent is correct.]]

The fast-options field indicates various options that are to modify
the behavior of the KDC. The following options are defined:

FastOptions ::= KerberosFlags
  -- reserved(0),
  -- anonymous(1),
  -- kdc-referrals(16)

<table>
<thead>
<tr>
<th>Bits</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>RESERVED</td>
<td>Reserved for future expansion of this field.</td>
</tr>
<tr>
<td>1</td>
<td>anonymous</td>
<td>Requesting the KDC to hide client names in the KDC response, as described next in this section.</td>
</tr>
<tr>
<td>16</td>
<td>kdc-referrals</td>
<td>Requesting the KDC to follow referrals, as described next in this section.</td>
</tr>
</tbody>
</table>

Bits 1 through 15 (with bit 2 and bit 15 included) are critical
options. If the KDC does not support a critical option, it MUST fail the request with KDC_ERR_UNKNOWN_CRITICAL_FAST_OPTIONS (there is no accompanying e-data defined in this document for this error code). Bit 16 and onward (with bit 16 included) are non-critical options. KDCs conforming to this specification ignores unknown non-critical options.

KDC_ERR_UNKNOWN_FAST_OPTIONS TBA

The anonymous Option

The Kerberos response defined in [RFC4120] contains the client identity in clear text. This makes traffic analysis straightforward. The anonymous option is designed to complicate traffic analysis. If the anonymous option is set, the KDC implementing PA_FX_FAST MUST identify the client as the anonymous principal in the KDC reply and the error response. Hence this option is set by the client if it wishes to conceal the client identity in the KDC response.

The kdc-referrals Option

The Kerberos client described in [RFC4120] has to request referral TGTs along the authentication path in order to get a service ticket for the target service. The Kerberos client described in the [REFERRALS] need to contact the AS specified in the error response in order to complete client referrals. The kdc-referrals option is designed to minimize the number of messages that need to be processed by the client. This option is useful when, for example, the client may contact the KDC via a satellite link that has high network latency, or the client has limited computational capabilities. If the kdc-referrals option is set, the KDC that honors this option acts as the client to follow AS referrals and TGS referrals [REFERRALS], and return the service ticket to the named server principal in the client request using the reply key expected by the client. The kdc-referrals option can be implemented when the KDC knows the reply key. The KDC can ignore kdc-referrals option when it does not understand it or it does not allow this option based on local policy. The client SHOULD be able to process the KDC responses when this option is not honored by the KDC.

The padata field contains a list of PA-DATA structures as described in Section 5.2.7 of [RFC4120]. These PA-DATA structures can contain FAST factors. They can also be used as generic typed-holes to contain data not intended for proving the client’s identity or establishing a reply key, but for protocol extensibility.
The KDC-REQ-BODY in the FAST structure is used in preference to the KDC-REQ-BODY outside of the FAST pre-authentication. This outer structure SHOULD be filled in for backwards compatibility with KDCs that do not support FAST. The client MAY fill in the cname and crealm fields in the kdc-req-body in the KrbFastReq structure and leave the cname field and the crealm field in KDC-REQ absent, in order to conceal the client’s identity in the AS-REQ. [[anchor14: Absent is probably wrong. Presumably we want a name similar to the anonymous principal name.]]

6.5.3. FAST Response

The KDC that supports the PA_FX_FAST padata MUST include a PA_FX_FAST padata element in the KDC reply. In the case of an error, the PA_FX_FAST padata is included in the KDC responses according to Section 6.5.4.

The corresponding padata-value field [RFC4120] for the PA_FX_FAST in the KDC response contains the DER encoding of the ASN.1 type PA-FX-FAST-REPLY.

\[
\text{PA-FX-FAST-REPLY ::= CHOICE { }
\text{\hspace{1em}}\text{armored-data [0] KrbFastArmoredRep, }
\text{\hspace{1em}}\ldots
\text{}}
\]

\[
\text{KrbFastArmoredRep ::= SEQUENCE { }
\text{\hspace{1em}}\text{\text{enc-fast-rep [0] EncryptedData, -- KrbFastResponse -- }
\text{\hspace{1em}}\text{-- The encryption key is the armor key in the request, and }
\text{\hspace{1em}}\text{-- the key usage number is KEY_USAGE_FAST_REP. }
\text{\hspace{1em}}\ldots
\text{}}
\]

\[\text{KEY_USAGE_FAST_REP TBA}\]

The PA-FX-FAST-REPLY structure contains a KrbFastArmoredRep structure. The KrbFastArmoredRep structure encapsulates the padata in the KDC reply in the encrypted form. The KrbFastResponse is encrypted with the armor key used in the corresponding request, and the key usage number is KEY_USAGE_FAST_REP.

The Kerberos client who does not receive a PA-FX-FAST-REPLY in the KDC response MUST support a local policy that rejects the response. Clients MAY also support policies that fall back to other mechanisms or that do not use pre-authentication when FAST is unavailable. It is important to consider the potential downgrade attacks when deploying such a policy.

The KrbFastResponse structure contains the following information:
The padata field in the KrbFastResponse structure contains a list of PA-DATA structures as described in Section 5.2.7 of [RFC4120]. These PA-DATA structures are used to carry data advancing the exchange specific for the FAST factors. They can also be used as generic typed-holes for protocol extensibility.

The rep-key field, if present, contains the reply key that is used to encrypted the KDC reply. The rep-key field MUST be absent in the case where an error occurs. The enctype of the rep-key is the strongest mutually supported by the KDC and the client.

The finished field contains a KrbFastFinished structure. It is filled by the KDC in the final message in the conversation; it MUST be absent otherwise. In other words, this field can only be present in an AS-REP or a TGS-REP when a ticket is returned.

The KrbFastFinished structure contains the following information:

KrbFastFinished ::= SEQUENCE {
    timestamp   [0] KerberosTime,
    usec        [1] Microseconds,
    -- timestamp and usec represent the time on the KDC when
    -- the reply was generated.
    realm       [2] Realm,
    cname       [3] PrincipalName,
    -- Contains the client realm and the client name.
    checksum    [4] Checksum,
    -- Checksum performed over all the messages in the
    -- conversation, except the containing message.
    -- The checksum key is the binding key as defined in
    -- Section 6.3, and the checksum type is the required
    -- checksum type of the binding key.
    ...
}

...
The timestamp and usec fields represent the time on the KDC when the reply ticket was generated, these fields have the same semantics as the corresponding-identically-named fields in Section 5.6.1 of [RFC4120]. The client MUST use the KDC’s time in these fields thereafter when using the returned ticket. Note that the KDC’s time in AS-REP may not match the authtime in the reply ticket if the kdc-referrals option is requested and honored by the KDC.

The cname and crealm fields identify the authenticated client.

The checksum field contains a checksum of all the messages in the conversation prior to the containing message (the containing message is excluded). The checksum key is the binding key as defined in Section 6.3, and the checksum type is the required checksum type of the enctype of that key, and the key usage number is KEY_USAGE_FAST_FINISHED. Examples would be good here; what all goes into the checksum?

When FAST padata is included, the PA-FX-COOKIE padata as defined in Section 6.3 MUST also be included if the KDC expects at least one more message from the client in order to complete the authentication.

6.5.4.Authenticated Kerberos Error Messages using Kerberos FAST

If the Kerberos FAST padata was included in the request, unless otherwise specified, the e-data field of the KRB-ERROR message [RFC4120] contains the ASN.1 DER encoding of the type METHOD-DATA [RFC4120] and a PA_FX_FAST is included in the METHOD-DATA. The KDC MUST include all the padata elements such as PA-ETYPE-INFO2 and padata elements that indicate acceptable pre-authentication mechanisms [RFC4120] and in the KrbFastResponse structure.

If the Kerberos FAST padata is included in the request but not included in the error reply, it is a matter of the local policy on the client to accept the information in the error message without integrity protection. The Kerberos client MAY process an error message without a PA-FX-FAST-REPLY, if that is only intended to return better error information to the application, typically for trouble-shooting purposes.

In the cases where the e-data field of the KRB-ERROR message is expected to carry a TYPED-DATA [RFC4120] element, the PA_FX_TYPED_DATA padata is included in the KrbFastResponse structure to encapsulate the TYPED-DATA [RFC4120] elements. For example, the TD_TRUSTED_CERTIFIERS structure is expected to be in the KRB-ERROR message when the error code is KDC_ERR_CANT_VERIFY_CERTIFICATE.
The corresponding padata-value for the PA_FX_TYPED_DATA padata type contains the DER encoding of the ASN.1 type TYPED-DATA [RFC4120].

6.5.5.  The Authenticated Timestamp FAST Factor

The encrypted time stamp [RFC4120] padata can be used as a FAST factor to authenticate the client and it does not expose the cipher text derived using the client’s long term keys. However this FAST factor is not risk-free from current intellectual property claims as of the time of this writing. To provide a clean replacement FAST factor that closely matches the encrypted timestamp FAST factor, the authenticated timestamp pre-authentication is introduced in this section.

The authenticated timestamp FAST factor authenticates a client by means of computing a checksum over a time-stamped structure using the client’s long term keys. The padata-type is PA_AUTHENTICATED_TIMESTAMP and the corresponding padata-value contains the DER encoding of ASN.1 type AuthenticatedTimestamp.
AuthenticatedTimestampToBeSigned ::= SEQUENCE {
  timestamp   [0] PA-ENC-TS-ENC,
    -- Contains the timestamp field of the corresponding
    -- AuthenticatedTimestamp structure.
  req-body    [1] KDC-REQ-BODY OPTIONAL,
    -- MUST contain the req-body field of the KDC-REQ
    -- structure in the containing AS-REQ for the client
    -- request.
    -- MUST be Absent for the KDC reply.
  ...
}

AuthenticatedTimestamp ::= SEQUENCE {
  timestamp   [0] PA-ENC-TS-ENC,
    -- Filled out according to Section 5.2.7.2 of [RFC4120]
    -- Contains the client’s current time for the client,
    -- and the KDC’s current time for the KDC.
  checksum    [1] CheckSum,
    -- The checksum is performed over the type
    -- AuthenticatedTimestampToBeSigned and the key usage is
    -- KEY_USAGE_AUTHENTICATED_TS_CLIENT for the client and
    -- KEY_USAGE_AUTHENTICATED_TS_KDC for the KDC
    ...
}

KEY_USAGE_AUTHENTICATED_TS_CLIENT  TBA
KEY_USAGE_AUTHENTICATED_TS_KDC     TBA

The client fills out the AuthenticatedTimestamp structure as follows:

- The timestamp field in the AuthenticatedTimestamp structure is
  filled out with the client’s current time according to Section
  5.2.7.2 of [RFC4120].

- The checksum field in the AuthenticatedTimestamp structure is
  performed over the type AuthenticatedTimestampToBeSigned. The
  checksum key is one of the client’s long term keys. The key usage
  for the checksum operation is KEY_USAGE_AUTHENTICATED_TS_CLIENT.
  The checksum type is the required checksum type for the strongest
  enctype mutually supported by the client and the KDC.

- Within the AuthenticatedTimestampToBeSigned structure, the
  timestamp field contains the timestamp field of the corresponding
  AuthenticatedTimestamp structure, and the req-body field MUST
  contain the req-body field of the KDC-REQ structure in the
  containing AS-REQ.

Upon receipt of the PA_AUTHENTICATED_TIMESTAMP FAST factor, the KDC
MUST process the padata in a way similar to that of the encrypted timestamp padata. The KDC MUST verify the checksum in the AuthenticatedTimestamp structure and the timestamp is within the window of acceptable clock skew for the KDC.

When the authenticated timestamp FAST factor is accepted by the KDC, the KDC MUST include a PA_AUTHENTICATED_TIMESTAMP as a FAST factor in a successful KDC reply and it MUST include the rep-key field as defined in Section 6.5.3.

The KDC fills out the AuthenticatedTimestamp structure as follows:

- The timestamp field in the AuthenticatedTimestamp structure is filled out with the KDC’s current time according to Section 5.2.7.2 of [RFC4120].

- The checksum field in the AuthenticatedTimestamp structure is performed over the type AuthenticatedTimestampToBeSigned. The checksum key is the reply key picked from the client’s long term keys according to [RFC4120]. The key usage for the checksum operation is KEY_USAGE_AUTHENTICATED_TS_KDC. The checksum type is the required checksum type for the checksum key.

- Within the AuthenticatedTimestampToBeSigned structure, the timestamp field contains the timestamp field of the corresponding AuthenticatedTimestamp structure, and the req-body field MUST be absent.

Upon receipt of the PA_AUTHENTICATED_TIMESTAMP FAST factor in the KDC reply, the client MUST verify the checksum in the AuthenticatedTimestamp structure and the timestamp is within the window of acceptable clock skew for the client. The successful verification of the PA_AUTHENTICATED_TIMESTAMP padata authenticates the KDC.

The authenticated timestamp FAST factor provides the following facilities: client-authentication, replacing-reply-key, KDC-authentication. It does not provide the strengthening-reply-key facility. The security considerations section of this document provides an explanation why the security requirements are met.

Conforming implementations MUST support the authenticated timestamp FAST factor.

6.6. Authentication Strength Indication

Implementations that have pre-authentication mechanisms offering significantly different strengths of client authentication MAY choose
to keep track of the strength of the authentication used as an input into policy decisions. For example, some principals might require strong pre-authentication, while less sensitive principals can use relatively weak forms of pre-authentication like encrypted timestamp.

An AuthorizationData data type AD-Authentication-Strength is defined for this purpose.

AD-authentication-strength TBA

The corresponding ad-data field contains the DER encoding of the pre-authentication data set as defined in Section 6.4. This set contains all the pre-authentication mechanisms that were used to authenticate the client. If only one pre-authentication mechanism was used to authenticate the client, the pre-authentication set contains one element.

The AD-authentication-strength element MUST be included in the AD-IF-RELEVANT, thus it can be ignored if it is unknown to the receiver.

7. IANA Considerations

This document defines several new pa-data types, key usages and error codes. In addition it would be good to track which pa-data items are only to be used as FAST factors.

8. Security Considerations

The kdc-referrals option in the Kerberos FAST padata requests the KDC to act as the client to follow referrals. This can overload the KDC. To limit the damages of denied of service using this option, KDCs MAY restrict the number of simultaneous active requests with this option for any given client principal.

Because the client secrets are known only to the client and the KDC, the verification of the authenticated timestamp proves the client’s identity, the verification of the authenticated timestamp in the KDC reply proves that the expected KDC responded. The encrypted reply key is contained in the rep-key in the PA-FX-FAST-REPLY. Therefore, the authenticated timestamp FAST factor as a pre-authentication mechanism offers the following facilities: client-authentication, replacing-reply-key, KDC-authentication. There is no un-authenticated clear text introduced by the authenticated timestamp FAST factor.
9.  Acknowledgements

Several suggestions from Jeffery Hutzman based on early revisions of this document led to significant improvements of this document.

The proposal to ask one KDC to chase down the referrals and return the final ticket is based on requirements in [ID.CROSS].

Joel Webber had a proposal for a mechanism similar to FAST that created a protected tunnel for Kerberos pre-authentication.

10.  References

10.1.  Normative References


10.2.  Informative References


[KRB-WG.SAM]
Appendix A.  ASN.1 module

KerberosPreauthFramework {
    iso(1) identified-organization(3) dod(6) internet(1)
    security(5) kerberosV5(2) modules(4) preauth-framework(3)
} DEFINITIONS EXPLICIT TAGS ::= BEGIN

IMPORTS
    KerberosTime, PrincipalName, Realm, EncryptionKey, Checksum,
    Int32, EncryptedData, PA-ENC-TS-ENC, PA-DATA, KDC-REQ-BODY
FROM KerberosV5Spec2 { iso(1) identified-organization(3)
    dod(6) internet(1) security(5) kerberosV5(2)
    modules(4) krb5spec2(2) };  
-- as defined in RFC 4120.

PA-FX-COOKIE ::= SEQUENCE {
    conversationId  [0] OCTET STRING,
        -- Contains the identifier of this conversation. This field
        -- must contain the same value for all the messages
        -- within the same conversation.
    enc-binding-key [1] EncryptedData OPTIONAL,
        -- EncryptionKey --
        -- This field is present when and only when a FAST
        -- padata as defined in Section 6.5 is included.
        -- The encrypted data, when decrypted, contains an
        -- EncryptionKey structure.
        -- This encryption key is encrypted using the armor key
        -- (defined in Section 6.5.1), and the key usage for the
        -- encryption is KEY_USAGE_FAST_BINDING_KEY.
    cookie          [2] OCTET STRING OPTIONAL,
        -- Opaque data, for use to associate all the messages in
        -- a single conversation between the client and the KDC.
        -- This is generated by the KDC and the client MUST copy
        -- the exact cookie encapsulated in a PA_FX_COOKIE data
        -- element into the next message of the same conversation.
..."}
KrbFastArmor ::= SEQUENCE {
  armor-type  [0] Int32,
  -- Type of the armor.
  armor-value [1] OCTET STRING,
  -- Value of the armor.
  ...
}

PA-FX-FAST-REQUEST ::= CHOICE {
  armored-data [0] KrbFastArmoredReq,
  ...
}

KrbFastArmoredReq ::= SEQUENCE {
  armor        [0] KrbFastArmor OPTIONAL,
  -- Contains the armor that identifies the armor key.
  -- MUST be present in AS-REQ.
  -- MUST be absent in TGS-REQ.
  req-checksum [1] Checksum,
  -- Checksum performed over the type KDC-REQ-BODY for
  -- the req-body field of the KDC-REQ structure defined in
  -- [RFC4120]
  -- The checksum key is the armor key, the checksum
  -- type is the required checksum type for the enctype of
  -- the armor key, and the key usage number is
  -- KEY_USAGE_FAST_REA_CHKSUM.
  -- The encryption key is the armor key, and the key usage
  -- number is KEY_USAGE_FAST_ENC.
  ...
}

KrbFastReq ::= SEQUENCE {
  fast-options [0] FastOptions,
  -- Additional options.
  padata       [1] SEQUENCE OF PA-DATA,
  -- padata typed holes.
  req-body     [2] KDC-REQ-BODY,
  -- Contains the KDC request body as defined in Section
  -- 5.4.1 of [RFC4120]. The req-body field in the KDC-REQ
  -- structure [RFC4120] MUST be ignored.
  -- The client name and realm in the KDC-REQ [RFC4120]
-- MUST NOT be present for AS-REQ and TGS-REQ when
-- Kerberos FAST padata is included in the request.
...
}

FastOptions ::= KerberosFlags
   -- reserved(0),
   -- anonymous(1),
   -- kdc-referrals(16)

PA-FX-FAST-REPLY ::= CHOICE {
   armored-data [0] KrbFastArmoredRep,
   ...}

KrbFastArmoredRep ::= SEQUENCE {
   enc-fast-rep [0] EncryptedData, -- KrbFastResponse --
   -- The encryption key is the armor key in the request, and
   -- the key usage number is KEY_USAGE_FAST_REP.
   ...}

KrbFastResponse ::= SEQUENCE {
   padata      [0] SEQUENCE OF PA-DATA,
   -- padata typed holes.
   rep-key     [1] EncryptionKey OPTIONAL,
   -- This, if present, replaces the reply key for AS and TGS.
   -- MUST be absent in KRB-ERROR.
   finished    [2] KrbFastFinished OPTIONAL,
   -- MUST be present if the client is authenticated,
   -- absent otherwise.
   -- Typically this is present if and only if the containing
   -- message is the last one in a conversation.
   ...}

KrbFastFinished ::= SEQUENCE {
   timestamp   [0] KerberosTime,
   usec        [1] Microseconds,
   -- timestamp and usec represent the time on the KDC when
   -- the reply was generated.
   crealm      [2] Realm,
   cname       [3] PrincipalName,
   -- Contains the client realm and the client name.
   checksum    [4] Checksum,
   -- Checksum performed over all the messages in the
   -- conversation, except the containing message.
   -- The checksum key is the binding key as defined in
   ...}
-- Section 6.3, and the checksum type is the required
-- checksum type of the binding key.
...

AuthenticatedTimestampToBeSigned ::= SEQUENCE {
  timestamp   [0] PA-ENC-TS-ENC,
    -- Contains the timestamp field of the corresponding
    -- AuthenticatedTimestamp structure.
  req-body    [1] KDC-REQ-BODY OPTIONAL,
    -- MUST contain the req-body field of the KDC-REQ
    -- structure in the containing AS-REQ for the client
    -- request.
    -- MUST be Absent for the KDC reply.
...

AuthenticatedTimestamp ::= SEQUENCE {
  timestamp   [0] PA-ENC-TS-ENC,
    -- Filled out according to Section 5.2.7.2 of [RFC4120].
    -- Contains the client’s current time for the client,
    -- and the KDC’s current time for the KDC.
  checksum    [1] CheckSum,
    -- The checksum is performed over the type
    -- AuthenticatedTimestampToBeSigned and the key usage is
    -- KEY_USAGE_AUTHENTICATED_TS_CLIENT for the client and
    -- KEY_USAGE_AUTHENTICATED_TS_KDC for the KDC
...

END

Authors’ Addresses

Larry Zhu
Microsoft Corporation
One Microsoft Way
Redmond, WA  98052
US

Email: lzhu@microsoft.com

Sam hartman
MIT

Email: hartmans@mit.edu