Internet Engineering Task Force (IETF)
Request for Comments: 6113
Updates: 4120
Category: Standards Track
ISSN: 2070-1721

S. Hartman
Painless Security
L. Zhu
Microsoft Corporation

April 2011

A Generalized Framework for Kerberos Pre-Authentication

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 facility called pre-authentication. Pre-authentication mechanisms can use this facility to extend the Kerberos protocol and prove the identity of a principal.

This document describes a more formal model for this facility. 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 key distribution center with a reply key strengthening mechanism; this secure channel can be used to protect the authentication exchange and 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.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6113.

Table of Contents

1. Introduction

The core Kerberos specification [RFC4120] treats pre-authentication data (padata) as an opaque typed hole in the messages to the key distribution center (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, tasks such as combining a key with the long-term secrets 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 plug-in architectures for pre-authentication; such architectures are likely to support mechanisms that follow this framework plus commonly used extensions. This framework also facilitates combining multiple pre-authentication mechanisms, each of which may represent an authentication factor, into a single multi-factor pre-authentication mechanism.

One of these common tools is the flexible authentication secure tunneling (FAST) padata type. FAST provides a protected channel between the client and the key distribution center (KDC), and it can optionally deliver key material used to strengthen the 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 5.4. 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.

1.1. 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].

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.

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 client's Authentication Service (AS) 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 client's AS. In the Ticket Granting Service (TGS) exchange, a conversation consists of the request message and the reply message. The term conversation is defined here for both AS and TGS for convenience of discussion. See Section 5.2 for specific rules on the extent of a conversation in the AS-REQ case. Prior to this framework, implementations needed to use implementation-specific heuristics to determine the extent of a conversation.

If the KDC reply in an AS exchange is verified, the KDC is authenticated by the client. In this document, verification of the KDC reply is used as a synonym of authentication of the KDC.

1.2. Conformance Requirements

This section summarizes the mandatory-to-implement subset of this specification as a convenience to implementors. The actual requirements and their context are stated in the body of the document.

Clients conforming to this specification MUST support the padata defined in Section 5.2.

Conforming implementations MUST support Kerberos FAST padata (Section 5.4). Conforming implementations MUST implement the FX_FAST_ARMOR_AP_REQUEST armor type.

Conforming implementations MUST support the encrypted challenge FAST factor (Section 5.4.6).

2. Model for Pre-Authentication

When a Kerberos client wishes to obtain a ticket, it sends an initial Authentication Service (AS) request to the KDC. If pre-authentication is required but not being used, then the KDC will respond with a KDC_ERR_PREAUTH_REQUIRED error [RFC4120]. 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 5.2.

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.

2.1. 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 origin of the KDC reply can be verified by the client (i.e., whether the KDC is authenticated to the client)

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 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 are 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. It is useful to keep track of this information 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 5.5.

Initially, the reply key is not 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 3.3 that the reply key SHOULD NOT be replaced after it is used.

Initially, the reply key is not replaced. If a mechanism implements the Replace Reply Key facility discussed in Section 3.3, 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 as 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 a conversation, the KDC reply is presumed to be verified using the client's long-term key. It should be noted that in this document, verifying the KDC reply means authenticating the KDC, and these phrases are used interchangeably. 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.

In this specification, KDC verification/authentication refers to the level of authentication of the KDC to the client provided by RFC 4120. There is a stronger form of KDC verification that, while sometimes important in Kerberos deployments, is not addressed in this specification: 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 machine 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.

2.2. 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_REQUIRED (defined in Section 5.2) for pre-authentication configurations that use multi-round-trip mechanisms; see Section 2.4 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, offered alone, or both. For each mechanism that is offered alone (even if it is also offered in an authentication set), the KDC includes the pre-authentication type ID

of the mechanism in the padata sequence returned in the KDC_ERR_PREAUTH_REQUIRED error. Mechanisms that are only offered as part of an authentication set are not directly represented in the padata sequence returned in the KDC_ERR_PREAUTH_REQUIRED error, although they are represented in the PA-AUTHENTICATION-SET sequence.

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 the KDC needs to expose cipher text encrypted in a weak key before the client has proven knowledge of that key, and where pre-authentication is desirable.

2.3. 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 guess values for the information it would normally receive from that error response or use cached information obtained in prior interactions with the KDC.

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 5.2. Clients SHOULD choose one authentication set or mechanism that could lead to authenticating the user and ignore other such mechanisms. However, this rule does not affect the processing of padata unrelated to this framework; clients SHOULD process such padata normally. 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 padata, it MUST NOT process the padata, allow the padata to affect the pre-authentication state, or respond to the padata.

For each instance of padata the client chooses to process, the client processes the padata and modifies the pre-authentication state as required by that mechanism.

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.

2.4. 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. If a PA-FX-COOKIE pre-authentication data item is present, it is processed first; see Section 5.2 for a definition. It then processes the padata in the request. As mentioned in Section 2.3, the KDC MAY ignore padata that are inappropriate for the configuration and MUST ignore padata of an unknown type. The KDC MUST NOT ignore padata of types used in previous messages. For example, if a KDC issues a KDC_ERR_PREAUTH_REQUIRED error including padata of type x, then the KDC cannot ignore padata of type x received in an AS-REQ message from the client.

At this point, the KDC decides whether it will issue an 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_REQUIRED 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. 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 are generated, the error response is sent. Typically, the errors with the code KDC_ERR_MORE_PREAUTH_DATA_REQUIRED in a conversation will include KDC state, as discussed in Section 5.2.

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.

3. 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. It is often easier to construct a secure, simple solution by combining multiple facilities in a single mechanism 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 3.2 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 2.1 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.

3.1. 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 Public Key Cryptography for Initial Authentication in Kerberos (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.

3.2. 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 5.1.

Typically, a mechanism implementing this facility will know that the other side of the exchange supports the facility before the reply key is changed. For example, a mechanism might need to learn the certificate for a KDC before encrypting a new key in the public key belonging to that certificate. However, 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. When mechanisms include such a container, the hint provided for use in authentication sets (as defined in Section 5.3) MUST contain a sequence of inner mechanisms along with hints for those mechanisms. 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.

3.3. Replace 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. This facility MUST NOT be used if there can be a man-in-the-middle between the client and the KDC. 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 the case 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 3.2, if appropriate.

3.4. 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.

4. 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 5.3).

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 considerations 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 alteration. 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 5.2, 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.

4.1. Protecting Requests/Responses

Mechanism designers SHOULD protect cleartext 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 than 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.

Mechanisms MUST guarantee that by the end of a successful authentication exchange, both the client and the KDC have verified all the plaintext sent by the other party. If there is more than one round trip in the exchange, mechanisms MUST additionally guarantee that no individual messages were reordered or replayed from a previous exchange. Strategies for accomplishing this include using message authentication codes (MACs) to protect the plaintext as it is sent including some form of nonce or cookie to allow for the chaining of state from one message to the next or exchanging a MAC of the entire conversation after a key is established.

Mechanism designers need to provide a strategy for updating cryptographic algorithms, such as defining a new pre-authentication type for each algorithm or taking advantage of the client's list of supported RFC 3961 encryption types to indicate the client's support for cryptographic algorithms.

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 incrementally deploy a fix specific to a particular algorithm thus avoid a multi-year standardization and deployment cycle, when real attacks do arise against that algorithm.

Note that data used by FAST factors (defined in Section 5.4) is encrypted in a protected channel; thus, they do not share the un-authenticated-text issues with mechanisms designed as full-blown pre-authentication mechanisms.

5. 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.

5.1. 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. An 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 passphrases.

```
KRB-FX-CF1(UTF-8 string, UTF-8 string) -> (UTF-8 string)
KRB-FX-CF1(x, y) := 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:

```
    KRB-FX-CF2(protocol key, protocol key, octet string,
              octet string)  ->  (protocol key)

    PRF+(K1, pepper1) -> octet-string-1
    PRF+(K2, pepper2) -> octet-string-2
    KRB-FX-CF2(K1, K2, pepper1, pepper2) :=
           random-to-key(octet-string-1 ^ octet-string-2)
```

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

```
 PRF+(protocol key, octet string) -> (octet string)

 PRF+(key, shared-info) := pseudo-random( key,  1 || shared-info ) ||
                pseudo-random( key, 2 || shared-info ) ||
                pseudo-random( key, 3 || 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. Unless otherwise specified, the resulting enctype of KRB-FX-CF2 is the enctype of k1. The pseudo-random() operation is the RFC 3961 pseudo-random() operation for the corresponding input key; the random-to-key() operation is the RFC 3961 random-to-key operation for the resulting key.

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.

5.2. Managing States for the KDC

Kerberos KDCs are stateless in that 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          90
```

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 padata type is defined in this section to facilitate state management in the AS exchange. These padata are 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.

```
PA-FX-COOKIE                     133
    -- Stateless cookie that is not tied to a specific KDC.
```

The corresponding padata-value field [RFC4120] contains an opaque token that will be echoed by the client in its response to an error from the KDC.

The cookie token is generated by the KDC and transmitted in a PA-FX-COOKIE pre-authentication data item of a KRB-ERROR message. 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. As a result, it is not generally possible to mix KDC implementations from different vendors in the same realm. 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. Implementations need to consider replay both of an entire conversation and of messages within a conversation when designing what information is stored in a cookie and how pre-authentication mechanisms are implemented.

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_REQUIRED error with a PA-FX-COOKIE to identify the conversation with the client, according to Section 2.2. The cookie is not expected to stay constant for a conversation: the KDC is expected to generate a new cookie for each message.

KDC_ERR_MORE_PREAUTH_DATA_REQUIRED 91

A client MAY throw away the state associated with a conversation and begin a new conversation by discarding its state and not including a cookie in the first message of a conversation. KDCs that comply with this specification MUST include a cookie in a response when the client can continue the conversation. In particular, a KDC MUST include a cookie in a KDC_ERR_PREAUTH_REQUIRED or KDC_ERR_MORE_PREAUTH_DATA_REQUIRED. KDCs SHOULD include a cookie in errors containing additional information allowing a client to retry. One reasonable strategy for meeting these requirements is to always include a cookie in KDC errors.

A KDC MAY indicate that it is terminating a conversation by not including a cookie in a response. When FAST is used, clients can assume that the absence of a cookie means that the KDC is ending the conversation. Similarly, if a cookie is seen at all during a conversation, clients MAY assume that the absence of a cookie in a future message means that the KDC is ending the conversation. Clients also need to deal with KDCs, prior to this specification, that do not include cookies; if neither cookies nor FAST are used in a conversation, the absence of a cookie is not a strong indication that the KDC is terminating the conversation.

5.3. 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.

PA-AUTHENTICATION-SET 134

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 OPTIONAL,
    pa-value     [2] OCTET STRING OPTIONAL,
    ...
}
```

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 that 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. The pa-value element of the PA-AUTHENTICATION-SET-ELEM sequence is included to carry the first padata-value from the KDC to the client. If the client chooses this authentication set, then the client MUST process this pa-value. The pa-value element MUST be absent for all but the first entry in the authentication set. Clients MUST ignore the pa-value for the second and following entries in the authentication set.

If the client chooses an authentication set, then its first AS-REQ message MUST contain a PA-AUTH-SET-SELECTED padata element. This element contains the encoding of the PA-AUTHENTICATION-SET sequence received from the KDC corresponding to the authentication set that is chosen. The client MUST use the same octet values received from the KDC; it cannot re-encode the sequence. This allows KDCs to use bit-wise comparison to identify the selected authentication set. Permitting bit-wise comparison may limit the ability to use certain pre-authentication mechanisms that generate a dynamic challenge in an authentication set with optimistic selection of an authentication set. As with other optimistic pre-authentication failures, the KDC MAY return KDC_ERR_PREAUTH_FAILED with a new list of pre-authentication mechanisms (including authentication sets) if optimistic pre-authentication fails. The PA-AUTH-SET-SELECTED padata element MUST come before any padata elements from the authentication set in the padata sequence in the AS-REQ message. The client MAY

cache authentication sets from prior messages and use them to construct an optimistic initial AS-REQ. If the KDC receives a PA-AUTH-SET-SELECTED padata element that does not correspond to an authentication set that it would offer, then the KDC returns the KDC_ERR_PREAUTH_BAD_AUTHENTICATION_SET error. The e-data in this error contains a sequence of padata just as for the KDC_ERR_PREAUTH_REQUIRED error.

PA-AUTH-SET-SELECTED	135
KDC_ERR_PREAUTH_BAD_AUTHENTICATION_SET	92

The PA-AUTHENTICATION-SET appears only in the first message from the KDC to the client. In particular, the client MAY fail if the authentication mechanism sets change as the conversation progresses. Clients MAY assume that the hints provided in the authentication set contain enough information that the client knows what user interface elements need to be displayed during the entire authentication conversation. Exceptional circumstances, such as expired passwords or expired accounts, may require that additional user interface be displayed. Mechanism designers need to carefully consider the design of their hints so that the client has this information. This way, clients can construct necessary dialogue boxes or wizards based on the authentication set and can present a coherent user interface. Current standards for user interfaces do not provide an acceptable experience when the client has to ask additional questions later in the conversation.

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 5.2, MUST be sent by the KDC. One reason for this requirement is so that the conversation can continue if the conversation involves multiple KDCs. KDCs MUST support clients that do not include a cookie because they optimistically choose an authentication set, although they MAY always return a KDC_ERR_PREAUTH_BAD_AUTHENTICATION_SET and include a cookie in that message. Clients that support PA-AUTHENTICATION-SET MUST support PA-FX-COOKIE.

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_REQUIRED as defined

in Section 5.2 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 are no padata, the e-data field is absent in the KRB-ERROR message.

If the client sends the last message for a given mechanism, then the KDC sends the first message for the next mechanism. If the next mechanism does not start with a KDC-side challenge, then the KDC includes a padata item with the appropriate pa-type and an empty pa-data.

If the KDC sends the last message for a particular mechanism, the KDC also includes the first padata for the next mechanism.

5.4. 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 attacks. When combined with encrypted challenge, 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 and 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 4.

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 [RFC6112] [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.

Any FAST armor scheme MUST provide a fresh armor key for each conversation. Clients and KDCs can assume that if a message is encrypted and integrity protected with a given armor key, then it is part of the conversation using that armor key.

All KDCs in a realm MUST support FAST if FAST is offered by any KDC as a pre-authentication mechanism.

5.4.1. 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 is an OCTET STRING that 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          1
```

The FX_FAST_ARMOR_AP_REQUEST armor is based on Kerberos tickets.

Conforming implementations MUST implement the FX_FAST_ARMOR_AP_REQUEST armor type. If a FAST KDC receives an unknown armor type it MUST respond with KDC_ERR_PREAUTH_FAILED.

An armor type may be appropriate for use in armoring AS requests, armoring TGS requests, or both. TGS armor types MUST authenticate the client to the KDC, typically by binding the TGT sub-session key

to the armor key. As discussed below, it is desirable for AS armor types to authenticate the KDC to the client, but this is not required.

FAST implementations MUST maintain state about whether the armor mechanism authenticates the KDC. If it does not, then a FAST factor that authenticates the KDC MUST be used if the reply key is replaced.

5.4.1.1. 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 defined by the following function:

```
armor_key = KRB-FX-CF2( subkey, ticket_session_key,
            "subkeyarmor", "ticketarmor" )
```

The 'ticket_session_key' is the session key from the ticket in the ap-req. The 'subkey' is the ap-req subkey. This construction guarantees that both the KDC (through the session key) and the client (through the subkey) contribute to the armor key.

The server name field of the armor ticket MUST identify the TGS of the target realm. Here are three common 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 an authentication path to the client's realm, the host machine obtains a TGT by 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 RFC 4120, but the KDC has an asymmetric signing key whose binding with the expected KDC can be verified by the client, the client can use anonymous PKINIT [RFC6112] [RFC4556] to authenticate the KDC and obtain an anonymous TGT as the armor ticket. The armor ticket can also be a cross-realm 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.

If anonymous PKINIT is used to obtain the armor ticket, the KDC cannot know whether its signing key can be verified by the client; hence, the KDC MUST be marked as unverified from the KDC's point of view while the client could be able to authenticate the KDC by verifying the KDC's signing key is bound with the expected KDC. The client needs to carefully consider the risk and benefit tradeoffs associated with active attacks before exposing cipher text encrypted using the user's long-term secrets when the armor does not authenticate the KDC.

The TGS MUST reject a request if there is an AD-fx-fast-armor (71) element in the authenticator of the pa-tgs-req padata or if the ticket in the authenticator of a pa-tgs-req contains the AD-fx-fast-armor authorization data element. These tickets and authenticators MAY be used as FAST armor tickets but not to obtain a ticket via the TGS. This authorization data is used in a system where the encryption of the user's pre-authentication data is performed in an unprivileged user process. A privileged process can provide to the user process a host ticket, an authenticator for use with that ticket, and the sub-session key contained in the authenticator. In order for the host process to ensure that the host ticket is not accidentally or intentionally misused, (i.e., the user process might use the host ticket to authenticate as the host), it MUST include a critical authorization data element of the type AD-fx-fast-armor when providing the authenticator or in the enc-authorization-data field of the TGS request used to obtain the TGT. The corresponding ad-data field of the AD-fx-fast-armor element is empty.

This armor type is only valid for AS requests; implicit armor, described below in TGS processing, is the only supported way to establish an armor key for the TGS at this time.

5.4.2. 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. As with all pre-authentication types, the KDC SHOULD advertise PA-FX-FAST in a PREAUTH_REQUIRED error. KDCs MUST send the advertisement of PA-FX-FAST with an empty pa-value. Clients MUST ignore the pa-value of PA-FX-FAST in an initial PREAUTH_REQUIRED error. FAST is not expected to be used in an authentication set: clients will typically use FAST padata if available and this decision should not depend on what other pre-authentication methods are available. As such, no pa-hint is defined for FAST at this time.

```
PA-FX-FAST                          136
    -- Padata type for Kerberos FAST

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.
    req-checksum [1] Checksum,
        -- For AS, contains the checksum performed over the type
        -- KDC-REQ-BODY for the req-body field of the KDC-REQ
        -- structure;
        -- For TGS, contains the checksum performed over the type
        -- AP-REQ in the PA-TGS-REQ padata.
        -- 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_REQ_CHKSUM.
    enc-fast-req [2] EncryptedData, -- KrbFastReq --
        -- The encryption key is the armor key, and the key usage
        -- number is KEY_USAGE_FAST_ENC.
    ...
}

KEY_USAGE_FAST_REQ_CHKSUM           50
KEY_USAGE_FAST_ENC                  51
```

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_ENC.

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.

- There are two possibilities for armor for a TGS request. If the ticket presented in the PA-TGS-REQ authenticator is a TGT, then the client SHOULD NOT include the armor field in the Krbfastreq and a subkey MUST be included in the PA-TGS-REQ authenticator. In this case, the armor key is the same armor key that would be computed if the TGS-REQ authenticator was used in an FX_FAST_ARMOR_AP_REQUEST armor. Clients MAY present a non-TGT in the PA-TGS-REQ authenticator and omit the armor field, in which case the armor key is the same that would be computed if the authenticator were used in an FX_FAST_ARMOR_AP_REQUEST armor. This is the only case where a ticket other than a TGT can be used to establish an armor key; even though the armor key is computed the same as an FX_FAST_ARMOR_AP_REQUEST, a non-TGT cannot be used as an armor ticket in FX_FAST_ARMOR_AP_REQUEST. Alternatively, a client MAY use an armor type defined in the future for use with the TGS request.

The req-checksum field contains a checksum computed differently for AS and TGS. For an AS-REQ, it is performed over the type KDC-REQ-BODY for the req-body field of the KDC-REQ structure of the containing message; for a TGS-REQ, it is performed over the type AP-REQ in the PA-TGS-REQ padata of the TGS request. The checksum key is the armor key, and the checksum type is the required checksum type for the enctype of the armor key per [RFC3961]. This checksum MUST be a keyed checksum and it is included in order to bind the FAST padata to the outer request. A KDC that implements FAST will ignore the outer request, but including a checksum is relatively cheap and may prevent confusing behavior.

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].
        -- This req-body field is preferred over the outer field
        -- in the KDC request.
     ...
}
```

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),
    -- hide-client-names(1),
```

Bits	Name	Description
0	RESERVED	Reserved for future expansion of this field.
1	hide-client-names	Requesting the KDC to hide client names in the KDC response, as described next in this section.
16	kdc-follow-referrals	reserved [REFERRALS].

Bits 1 through 15 inclusive (with bit 1 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, and 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 ignore unknown non-critical options.

```
KDC_ERR_UNKNOWN_CRITICAL_FAST_OPTIONS   93
```

The hide-client-names Option

The Kerberos response defined in [RFC4120] contains the client identity in cleartext. This makes traffic analysis straightforward. The hide-client-names option is designed to complicate traffic analysis. If the hide-client-names option is set, the KDC implementing PA-FX-FAST MUST identify the client as the anonymous principal [RFC6112] 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. A conforming KDC ignores the client principal name in the outer KDC-REQ-BODY field, and identifies the client using the cname and crealm fields in the req-body field of the KrbFastReq structure.

The kdc-follow-referrals Option

This option is reserved for [REFERRALS].

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. If the KDC supports the PA-FX-FAST-REQUEST padata, unless otherwise specified, the client MUST place any padata that is otherwise in the outer KDC request body into this field. In a TGS request, PA-TGS-REQ padata is not included in this field and it is present in the outer KDC request body.

The KDC-REQ-BODY in the FAST structure is used in preference to the KDC-REQ-BODY outside of the FAST pre-authentication. The outer KDC-REQ-BODY structure SHOULD be filled in for backwards compatibility with KDCs that do not support FAST. A conforming KDC ignores the outer KDC-REQ-BODY field in the KDC request. Pre-authentication data methods such as [RFC4556] that include a checksum of the KDC-REQ-BODY should checksum the KDC-REQ-BODY in the FAST structure.

In a TGS request, a client MAY include the AD-fx-fast-used authdata either in the pa-tgs-req authenticator or in the authorization data in the pa-tgs-req ticket. If the KDC receives this authorization data but does not find a FAST padata, then it MUST return KRB_APP_ERR_MODIFIED.

5.4.3. 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 5.4.4.

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.

```
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.
    ...
}
KEY_USAGE_FAST_REP                52
```

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 MUST support a local policy that rejects the response if PA-FX-FAST-REPLY is not included in 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:

```
    KrbFastResponse ::= SEQUENCE {
        padata         [0] SEQUENCE OF PA-DATA,
            -- padata typed holes.
        strengthen-key [1] EncryptionKey OPTIONAL,
            -- This, if present, strengthens the reply key for AS and
            -- TGS. MUST be present for TGS.
            -- MUST be absent in KRB-ERROR.
        finished       [2] KrbFastFinished OPTIONAL,
            -- Present in AS or TGS reply; absent otherwise.
        nonce          [3] UInt32,
            -- Nonce from the client request.
        ...
}
```

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. Unless otherwise specified, the KDC MUST include any padata that are otherwise in the outer KDC-REP or KDC-ERROR structure into this field. The padata field in the KDC reply structure outside of the PA-FX-FAST-REPLY structure typically includes only the PA-FX-FAST-REPLY padata.

The strengthen-key field provides a mechanism for the KDC to strengthen the reply key. If set, the strengthen-key value MUST be randomly generated to have the same etype as that of the reply key before being strengthened, and then the reply key is strengthened after all padata items are processed. Let padata-reply-key be the reply key after padata processing.

```
reply-key = KRB-FX-CF2(strengthen-key, padata-reply-key,
                       "strengthenkey", "replykey")
```

The strengthen-key field MAY be set in an AS reply; it MUST be set in a TGS reply; it must be absent in an error reply. The strengthen key is required in a TGS reply so that an attacker cannot remove the FAST PADATA from a TGS reply, causing the KDC to appear not to support FAST.

The finished field contains a KrbFastFinished structure. It is filled by the KDC in the final message in the conversation. This field is present in an AS-REP or a TGS-REP when a ticket is returned, and it is not present in an error reply.

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.
    crealm          [2] Realm,
    cname           [3] PrincipalName,
        -- Contains the client realm and the client name.
    ticket-checksum [4] Checksum,
        -- checksum of the ticket in the KDC-REP using the armor
        -- and the key usage is KEY_USAGE_FAST_FINISH.
        -- The checksum type is the required checksum type
        -- of the armor key.
    ...
}
KEY_USAGE_FAST_FINISHED           53
```

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. The client need not confirm that the timestamp returned is within allowable clock skew: the armor key guarantees that the reply is fresh. The client MAY trust the timestamp returned.

The cname and crealm fields identify the authenticated client. If facilities described in [REFERRALS] are used, the authenticated client may differ from the client in the FAST request.

The ticket-checksum is a checksum of the issued ticket. The checksum key is the armor key, 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.

When FAST padata is included, the PA-FX-COOKIE padata as defined in Section 5.2 MUST be included in the padata sequence in the KrbFastResponse sequence if the KDC expects at least one more message from the client in order to complete the authentication.

The nonce field in the KrbFastResponse contains the value of the nonce field in the KDC-REQ of the corresponding client request and it binds the KDC response with the client request. The client MUST verify that this nonce value in the reply matches with that of the request and reject the KDC reply otherwise. To prevent the response from one message in a conversation from being replayed to a request in another message, clients SHOULD use a new nonce for each message in a conversation.

5.4.4. 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] in the KrbFastResponse structure.

The KDC MUST also include a PA-FX-ERROR padata item in the KRBFastResponse structure. The padata-value element of this sequence is the ASN.1 DER encoding of the type KRB-ERROR. The e-data field MUST be absent in the PA-FX-ERROR padata. All other fields should be the same as the outer KRB-ERROR. The client ignores the outer error and uses the combination of the padata in the KRBFastResponse and the error information in the PA-FX-ERROR.

PA-FX-ERROR	137

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. However, the client SHOULD process the KDC errors as the result of the KDC's inability to accept the AP_REQ armor and potentially retry another request with a different armor when applicable. 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, that information should be transmitted in a pa-data element within the KRBFastResponse

structure. The padata-type is the same as the data-type would be in the typed data element and the padata-value is the same as the data-value. As discussed in Section 7, data-types and padata-types are drawn from the same namespace. 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 [RFC4556].

5.4.5. Outer and Inner Requests

Typically, a client will know that FAST is being used before a request containing PA-FX-FAST is sent. So, the outer AS request typically only includes one pa-data item: PA-FX-FAST. The client MAY include additional pa-data, but the KDC MUST ignore the outer request body and any padata besides PA-FX-FAST if and only if PA-FX-FAST is processed. In the case of the TGS request, the outer request should include PA-FX-FAST and PA-TGS-REQ.

When an AS generates a response, all padata besides PA-FX-FAST should be included in PA-FX-FAST. The client MUST ignore other padata outside of PA-FX-FAST.

5.4.6. The Encrypted Challenge FAST Factor

The encrypted challenge FAST factor authenticates a client using the client's long-term key. This factor works similarly to the encrypted timestamp pre-authentication option described in [RFC4120]. The word "challenge" is used instead of "timestamp" because while the timestamp is used as an initial challenge, if the KDC and client do not have synchronized time, then the KDC can provide updated time to the client to use as a challenge. The client encrypts a structure containing a timestamp in the challenge key. The challenge key used by the client to send a message to the KDC is KRB-FX-CF2(armor_key,long_term_key, "clientchallengearmor", "challengelongterm"). The challenge key used by the KDC encrypting to the client is KRB-FX-CF2(armor_key, long_term_key, "kdcchallengearmor", "challengelongterm"). Because the armor key is fresh and random, the challenge key is fresh and random. The only purpose of the timestamp is to limit the validity of the authentication so that a request cannot be replayed. A client MAY base the timestamp on the KDC time in a KDC error and need not maintain accurate time synchronization itself. If a client bases its time on an untrusted source, an attacker may trick the client into producing an authentication request that is valid at some future time. The attacker may be able to use this authentication request to make it appear that a client has authenticated at that future time. If ticket-based armor is used, then the lifetime of the ticket will limit the window in which an attacker can make the client appear to

have authenticated. For many situations, the ability of an attacker to cause a client to appear to have authenticated is not a significant concern; the ability to avoid requiring time synchronization on clients is more valuable.

The client sends a padata of type PA-ENCRYPTED-CHALLENGE. The corresponding padata-value contains the DER encoding of ASN.1 type EncryptedChallenge.

```
EncryptedChallenge ::= EncryptedData
        -- Encrypted PA-ENC-TS-ENC, encrypted in the challenge key
        -- using key usage KEY_USAGE_ENC_CHALLENGE_CLIENT for the
        -- client and KEY_USAGE_ENC_CHALLENGE_KDC for the KDC.

PA-ENCRYPTED-CHALLENGE          138
KEY_USAGE_ENC_CHALLENGE_CLIENT  54
KEY_USAGE_ENC_CHALLENGE_KDC     55
```

The client includes some timestamp reasonably close to the KDC's current time and encrypts it in the challenge key in a PA-ENC-TS-ENC structure (see Section 5.2.7.2 in RFC 4120). Clients MAY use the current time; doing so prevents the exposure where an attacker can cause a client to appear to authenticate in the future. The client sends the request including this factor.

On receiving an AS-REQ containing the PA-ENCRYPTED-CHALLENGE FAST factor, the KDC decrypts the timestamp. If the decryption fails the KDC SHOULD return KDC_ERR_PREAUTH_FAILED, including PA-ETYPE-INFO2 in the KRBFastResponse in the error. The KDC confirms that the timestamp falls within its current clock skew returning KRB_APP_ERR_SKEW if not. The KDC then SHOULD check to see if the encrypted challenge is a replay. The KDC MUST NOT consider two encrypted challenges replays simply because the timestamps are the same; to be a replay, the ciphertext MUST be identical. Allowing clients to reuse timestamps avoids requiring that clients maintain state about which timestamps have been used.

If the KDC accepts the encrypted challenge, it MUST include a padata element of type PA-ENCRYPTED-CHALLENGE. The KDC encrypts its current time in the challenge key. The KDC MUST strengthen the reply key before issuing a ticket. The client MUST check that the timestamp decrypts properly. The client MAY check that the timestamp is within the window of acceptable clock skew for the client. The client MUST NOT require that the timestamp be identical to the timestamp in the issued credentials or the returned message.

The encrypted challenge FAST factor provides the following facilities: Client Authentication and KDC Authentication. This FAST factor also takes advantage of the FAST facility to strengthen the reply key. It does not provide the Replace Reply Key facility. The Security Considerations section of this document provides an explanation why the security requirements are met.

The encrypted challenge FAST factor can be useful in an authentication set. No pa-hint is defined because the only information needed by this mechanism is information contained in the PA-ETYPE-INFO2 pre-authentication data. KDCs are already required to send PA-ETYPE-INFO2. If KDCs were not required to send PA-ETYPE-INFO2 then that information would need to be part of a hint for encrypted challenge.

Conforming implementations MUST support the encrypted challenge FAST factor.

5.5. 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        70
```

The corresponding ad-data field contains the DER encoding of the pre-authentication data set as defined in Section 5.3. 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. Unless otherwise specified, the hint and value fields of the members of this sequence MUST be empty. In order to permit mechanisms to carry additional information about strength in these fields in the future, clients and application servers MUST ignore non-empty hint and value fields for mechanisms unless the implementation is updated with the interpretation of these fields for a given pre-authentication mechanism in this authorization element.

The AD-authentication-strength element MUST be included in the AD-KDC-ISSUED container so that the KDC integrity protects its contents. This element can be ignored if it is unknown to the receiver.

6. Assigned Constants

The pre-authentication framework and FAST involve using a number of Kerberos protocol constants. This section lists protocol constants first introduced in this specification drawn from registries not managed by IANA. Many of these registries would best be managed by IANA; that is a known issue that is out of scope for this document. The constants described in this section have been accounted for and will appear in the next revision of the Kerberos core specification or in a document creating IANA registries.

Section 7 creates IANA registries for a different set of constants used by the extensions described in this document.

6.1. New Errors

KDC_ERR_PREAUTH_EXPIRED	90
KDC_ERR_MORE_PREAUTH_DATA_REQUIRED	91
KDC_ERR_PREAUTH_BAD_AUTHENTICATION_SET	92
KDC_ERR_UNKNOWN_CRITICAL_FAST_OPTIONS	93

6.2. Key Usage Numbers

KEY_USAGE_FAST_REQ_CHKSUM	50
KEY_USAGE_FAST_ENC	51
KEY_USAGE_FAST_REP	52
KEY_USAGE_FAST_FINISHED	53
KEY_USAGE_ENC_CHALLENGE_CLIENT	54
KEY_USAGE_ENC_CHALLENGE_KDC	55

6.3. Authorization Data Elements

AD-authentication-strength	70
AD-fx-fast-armor	71
AD-fx-fast-used	72

6.4. New PA-DATA Types

PA-FX-COOKIE	133
PA-AUTHENTICATION-SET	134
PA-AUTH-SET-SELECTED	135
PA-FX-FAST	136
PA-FX-ERROR	137
PA-ENCRYPTED-CHALLENGE	138

7. IANA Considerations

This document creates a number of IANA registries. These registries are all located under Kerberos Parameters on http://www.iana.org. See [RFC5226] for descriptions of the registration policies used in this section.

7.1. Pre-Authentication and Typed Data

RFC 4120 defines pre-authentication data, which can be included in a KDC request or response in order to authenticate the client or extend the protocol. In addition, it defines Typed-Data, which is an extension mechanism for errors. Both pre-authentication data and typed data are carried as a 32-bit signed integer along with an octet string. The encoding of typed data and pre-authentication data is slightly different. However, the types for pre-authentication data and typed-data are drawn from the same namespace. By convention, registrations starting with TD- are typed data and registrations starting with PA- are pre-authentication data. It is important that these data types be drawn from the same namespace, because some errors where it would be desirable to include typed data require the e-data field to be formatted as pre-authentication data.

When Kerberos FAST is used, pre-authentication data encoding is always used.

There is one apparently conflicting registration between typed data and pre-authentication data. PA-GET-FROM-TYPED-DATA and TD-PADATA are both assigned the value 22. However, this registration is simply a mechanism to include an element of the other encoding. The use of both should be deprecated.

This document creates a registry for pre-authentication and typed data. The registration procedures are as follows. Expert review for pre-authentication mechanisms designed to authenticate users, KDCs, or establish the reply key. The expert first determines that the purpose of the method is to authenticate clients, KDCs, or to establish the reply key. If so, expert review is appropriate. The expert evaluates the security and interoperability of the specification.

IETF review is required if the expert believes that the pre-authentication method is broader than these three areas. Pre-authentication methods that change the Kerberos state machine or otherwise make significant changes to the Kerberos protocol should be Standards Track RFCs. A concern that a particular method needs to be a Standards Track RFC may be raised as an objection during IETF review.

Several of the registrations indicated below were made at a time when the Kerberos protocol was less mature and do not meet the current requirements for this registry. These registrations are included in order to accurately document what is known about the use of these protocol code points and to avoid conflicts.

Type	Value	Reference
PA-TGS-REQ	1	[RFC4120]
PA-ENC-TIMESTAMP	2	[RFC4120]
PA-PW-SALT	3	[RFC4120]
[reserved]	4	[RFC6113]
PA-ENC-UNIX-TIME	5	(deprecated) [RFC4120]
PA-SANDIA-SECUREID	6	[RFC4120]
PA-SESAME	7	[RFC4120]
PA-OSF-DCE	8	[RFC4120]
PA-CYBERSAFE-SECUREID	9	[RFC4120]
PA-AFS3-SALT	10	[RFC4120] [RFC3961]
PA-ETYPE-INFO	11	[RFC4120]
PA-SAM-CHALLENGE	12	[KRB-WG.SAM]
PA-SAM-RESPONSE	13	[KRB-WG.SAM]
PA-PK-AS-REQ_OLD	14	[PK-INIT-1999]
PA-PK-AS-REP_OLD	15	[PK-INIT-1999]
PA-PK-AS-REQ	16	[RFC4556]
PA-PK-AS-REP	17	[RFC4556]
PA-PK-OCSP-RESPONSE	18	[RFC4557]
PA-ETYPE-INFO2	19	[RFC4120]
PA-USE-SPECIFIED-KVNO	20	[RFC4120]
PA-SVR-REFERRAL-INFO	20	[REFERRALS]
PA-SAM-REDIRECT	21	[KRB-WG.SAM]
PA-GET-FROM-TYPED-DATA	22	(embedded in typed data) [RFC4120]
TD-PADATA	22	(embeds padata) [RFC4120]
PA-SAM-ETYPE-INFO	23	(sam/otp) [KRB-WG.SAM]
PA-ALT-PRINC	24	(crawdad@fnal.gov) [HW-AUTH]
PA-SERVER-REFERRAL	25	[REFERRALS]
PA-SAM-CHALLENGE2	30	(kenh@pobox.com) [KRB-WG.SAM]
PA-SAM-RESPONSE2	31	(kenh@pobox.com) [KRB-WG.SAM]
PA-EXTRA-TGT	41	Reserved extra TGT [RFC6113]
TD-PKINIT-CMS-CERTIFICATES	101	CertificateSet from CMS
TD-KRB-PRINCIPAL	102	PrincipalName
TD-KRB-REALM	103	Realm
TD-TRUSTED-CERTIFIERS	104	[RFC4556]
TD-CERTIFICATE-INDEX	105	[RFC4556]
TD-APP-DEFINED-ERROR	106	Application specific [RFC6113]
TD-REQ-NONCE	107	INTEGER [RFC6113]
TD-REQ-SEQ	108	INTEGER [RFC6113]
TD_DH_PARAMETERS	109	[RFC4556]
TD-CMS-DIGEST-ALGORITHMS	111	[ALG-AGILITY]

TD-CERT-DIGEST-ALGORITHMS	112	[ALG-AGILITY]
PA-PAC-REQUEST	128	[MS-KILE]
PA-FOR_USER	129	[MS-KILE]
PA-FOR-X509-USER	130	[MS-KILE]
PA-FOR-CHECK_DUPS	131	[MS-KILE]
PA-AS-CHECKSUM	132	[MS-KILE]
PA-FX-COOKIE	133	[RFC6113]
PA-AUTHENTICATION-SET	134	[RFC6113]
PA-AUTH-SET-SELECTED	135	[RFC6113]
PA-FX-FAST	136	[RFC6113]
PA-FX-ERROR	137	[RFC6113]
PA-ENCRYPTED-CHALLENGE	138	[RFC6113]
PA-OTP-CHALLENGE	141	(gareth.richards@rsa.com) [OTP-PREAUTH]
PA-OTP-REQUEST	142	(gareth.richards@rsa.com) [OTP-PREAUTH]
PA-OTP-CONFIRM	143	(gareth.richards@rsa.com) [OTP-PREAUTH]
PA-OTP-PIN-CHANGE	144	(gareth.richards@rsa.com) [OTP-PREAUTH]
PA-EPAK-AS-REQ	145	(sshock@gmail.com) [RFC6113]
PA-EPAK-AS-REP	146	(sshock@gmail.com) [RFC6113]
PA_PKINIT_KX	147	[RFC6112]
PA_PKU2U_NAME	148	[PKU2U]
PA-SUPPORTED-ETYPES	165	[MS-KILE]
PA-EXTENDED_ERROR	166	[MS-KILE]

7.2. Fast Armor Types

FAST armor types are defined in Section 5.4.1. A FAST armor type is a signed 32-bit integer. FAST armor types are assigned by standards action.

Type	Name	Description
0		Reserved.
1	FX_FAST_ARMOR_AP_REQUEST	Ticket armor using an ap-req.

7.3. FAST Options

A FAST request includes a set of bit flags to indicate additional options. Bits 0-15 are critical; other bits are non-critical. Assigning bits greater than 31 may require special support in implementations. Assignment of FAST options requires standards action.

Type	Name	Description
0	RESERVED	Reserved for future expansion of this field.
1	hide-client-names	Requesting the KDC to hide client names in the KDC response
16	kdc-follow-referrals	Reserved.

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 denial of service using this option, KDCs MAY restrict the number of simultaneous active requests with this option for any given client principal.

Regarding the facilities provided by the Encrypted Challenge FAST factor, the challenge key is derived from the client secrets and because the client secrets are known only to the client and the KDC, the verification of the EncryptedChallenge structure proves the client's identity, the verification of the EncryptedChallenge structure in the KDC reply proves that the expected KDC responded. Therefore, the Encrypted Challenge FAST factor as a pre-authentication mechanism offers the following facilities: Client Authentication and KDC Authentication. There is no un-authenticated cleartext introduced by the Encrypted Challenge FAST factor.

FAST provides an encrypted tunnel over which pre-authentication conversations can take place. In addition, FAST optionally authenticates the KDC to the client. It is the responsibility of FAST factors to authenticate the client to the KDC. Care MUST be taken to design FAST factors such that they are bound to the conversation. If this is not done, a man-in-the-middle may be able to cut&paste a FAST factor from one conversation to another. The easiest way to do this is to bind each FAST factor to the armor key that is guaranteed to be unique for each conversation.

The anonymous PKINIT mode for obtaining an armor ticket does not always authenticate the KDC to the client before the conversation begins. Tracking the KDC verified state guarantees that by the end of the conversation, the client has authenticated the KDC. However, FAST factor designers need to consider the implications of using their factor when the KDC has not yet been authenticated. If this proves problematic in an environment, then the particular FAST factor should not be used with anonymous PKINIT.

Existing pre-authentication mechanisms are believed to be at least as secure when used with FAST as they are when used outside of FAST.

One part of this security is making sure that when pre-authentication methods checksum the request, they checksum the inner request rather than the outer request. If the mechanism checksummed the outer request, a man-in-the-middle could observe it outside a FAST tunnel and then cut&paste it into a FAST exchange where the inner rather than outer request would be used to select attributes of the issued ticket. Such attacks would typically invalidate auditing information or create a situation where the client and KDC disagree about what ticket is issued. However, such attacks are unlikely to allow an attacker who would not be able to authenticate as a principal to do so. Even so, FAST is believed to defend against these attacks in existing legacy mechanism. However, since there is no standard for how legacy mechanisms bind the request to the pre-authentication or provide integrity protection, security analysis can be difficult. In some cases, FAST may significantly improve the integrity protection of legacy mechanisms.

The security of the TGS exchange depends on authenticating the client to the KDC. In the AS exchange, this is done using pre-authentication data or FAST factors. In the TGS exchange, this is done by presenting a TGT and by using the session (or sub-session) key in constructing the request. Because FAST uses a request body in the inner request, encrypted in the armor key, rather than the request body in the outer request, it is critical that establishing the armor key be tied to the authentication of the client to the KDC. If this is not done, an attacker could manipulate the options requested in the TGS request, for example, requesting a ticket with different validity or addresses. The easiest way to bind the armor key to the authentication of the client to the KDC is for the armor key to depend on the sub-session key of the TGT. This is done with the implicit TGS armor supported by this specification. Future armor types designed for use with the TGS MUST either bind their armor keys to the TGT or provide another mechanism to authenticate the client to the KDC.

9. Acknowledgements

Sam Hartman would like to thank the MIT Kerberos Consortium for its funding of his time on this project.

Several suggestions from Jeffrey Hutzelman based on early revisions of this documents 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 [CROSS].

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

Srinivas Cheruku and Greg Hudson provided valuable review comments.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[RFC4556] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, June 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC6112] Zhu, L., Leach, P., and S. Hartman "Anonymity Support for Kerberos", RFC 6112, April 2011.

10.2. Informative References

[ALG-AGILITY] Astrand, L. and L. Zhu, "PK-INIT algorithm agility", Work in Progress, August 2008.

[CROSS] Sakane, S., Zrelli, S., and M. Ishiyama , "Problem statement on the cross-realm operation of Kerberos in a specific system", Work in Progress, July 2007.

[EKE] Bellovin, S. and M. Merritt, "Augmented Encrypted Key Exchange: A Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise, Proceedings of the 1st ACM Conference on Computer and Communications Security, ACM Press.", November 1993.

[HW-AUTH] Crawford, M., "Passwordless Initial Authentication to Kerberos by Hardware Preauthentication", Work in Progress, October 2006.

[IEEE1363.2] IEEE, "IEEE P1363.2: Password-Based Public-Key Cryptography", 2004.

[KRB-WG.SAM] Hornstein, K., Renard, K., Neuman, C., and G. Zorn, "Integrating Single-use Authentication Mechanisms with Kerberos", Work in Progress, July 2004.

[MS-KILE] Microsoft, "Kerberos Protocol Extensions", <http://msdn.microsoft.com/en-us/library/cc206927.aspx>.

[OTP-PREAUTH] Richards, G., "OTP Pre-authentication", Work in Progress, February 2011.

[PK-INIT-1999] Tung, B., Neuman, C., Hur, M., Medvinsky, A., Medvinsky, S., Wray, J., and J. Trostle, "Public Key Cryptography for Initial Authentication in Kerberos", Work in Progress, July 1999.

[PKU2U] Zhu, L., Altman, J., and N. Williams, "Public Key Cryptography Based User-to-User Authentication - (PKU2U)", Work in Progress, November 2008.

[REFERRALS] Hartman, S., Ed., Raeburn, K., and L. Zhu, "Kerberos Principal Name Canonicalization and KDC-Generated Cross-Realm Referrals", Work in Progress, March 2011.

[RFC4557] Zhu, L., Jaganathan, K., and N. Williams, "Online Certificate Status Protocol (OCSP) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4557, June 2006.

Appendix A. Test Vectors for KRB-FX-CF2

This informative appendix presents test vectors for the KRB-FX-CF2 function. Test vectors are presented for several encryption types. In all cases, the first key (k1) is the result of string-to-key("key1", "key1", default_parameters) and the second key (k2) is the result of string-to-key("key2", "key2", default_parameters). Both keys are of the same enctype. The presented test vector is the hexadecimal encoding of the key produced by KRB-FX-CF2(k1, k2, "a", "b"). The peppers are one-octet ASCII strings.

In performing interoperability testing, there was significant ambiguity surrounding [RFC3961] pseudo-random operations. These test vectors assume that the AES pseudo-random operation is aes-ecb(trunc128(sha-1(input))) where trunc128 truncates its input to 128 bits. The 3DES pseudo-random operation is assumed to be des3-cbc(trunc128(sha-1(input))). The DES pseudo-random operation is assumed to be des-cbc(md5(input)). As specified in RFC 4757, the RC4 pseudo-random operation is hmac-sha1(input).

Interoperability testing also demonstrated ambiguity surrounding the DES random-to-key operation. The random-to-key operation is assumed to be distribute 56 bits into high-7-bits of 8 octets and generate parity.

These test vectors were produced with revision 22359 of the MIT Kerberos sources. The AES 256 and AES 128 test vectors have been confirmed by multiple other implementors. The RC4 test vectors have been confirmed by one other implementor. The DES and triple DES test vectors have not been confirmed.

```
aes 128 (enctype 17): 97df97e4b798b29eb31ed7280287a92a
AES256 (enctype 18): 4d6ca4e629785c1f01baf55e2e548566
                     b9617ae3a96868c337cb93b5e72b1c7b
DES (enctype 1): 43bae3738c9467e6
3DES (enctype 16): e58f9eb643862c13ad38e529313462a7f73e62834fe54a01
RC4 (enctype 23): 24d7f6b6bae4e5c00d2082c5ebab3672
```

Appendix B. 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,
     Microseconds, KerberosFlags, UInt32
          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-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 OPTIONAL,
    pa-value     [2] OCTET STRING OPTIONAL,
    ...
}

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.
    req-checksum [1] Checksum,
        -- For AS, contains the checksum performed over the type
        -- KDC-REQ-BODY for the req-body field of the KDC-REQ
        -- structure;
        -- For TGS, contains the checksum performed over the type
```

```
            -- AP-REQ in the PA-TGS-REQ padata.
            -- 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_REQ_CHKSUM.
        enc-fast-req [2] EncryptedData, -- KrbFastReq --
            -- 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].
            -- This req-body field is preferred over the outer field
            -- in the KDC request.
         ...
    }

    FastOptions ::= KerberosFlags
        -- reserved(0),
        -- hide-client-names(1),
        -- kdc-follow-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.
        strengthen-key [1] EncryptionKey OPTIONAL,
            -- This, if present, strengthens the reply key for AS and
            -- TGS.  MUST be present for TGS
            -- MUST be absent in KRB-ERROR.
```

```
        finished       [2] KrbFastFinished OPTIONAL,
            -- Present in AS or TGS reply; absent otherwise.
        nonce          [3] UInt32,
            -- Nonce from the client request.
        ...
    }

    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.
        ticket-checksum [4] Checksum,
            -- checksum of the ticket in the KDC-REP  using the armor
            -- and the key usage is KEY_USAGE_FAST_FINISH.
            -- The checksum type is the required checksum type
            -- of the armor key.
        ...
    }

    EncryptedChallenge ::= EncryptedData
            -- Encrypted PA-ENC-TS-ENC, encrypted in the challenge key
            -- using key usage KEY_USAGE_ENC_CHALLENGE_CLIENT for the
            -- client and KEY_USAGE_ENC_CHALLENGE_KDC for the KDC.
    END
```

Authors' Addresses

Sam Hartman
Painless Security

EMail: hartmans-ietf@mit.edu

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

EMail: larry.zhu@microsoft.com