Internet Engineering Task Force (IETF)
Request for Comments: 6560
Category: Standards Track
ISSN: 2070-1721

G. Richards
RSA, The Security Division of EMC

April 2012

One-Time Password (OTP) Pre-Authentication

Abstract

The Kerberos protocol provides a framework authenticating a client using the exchange of pre-authentication data. This document describes the use of this framework to carry out One-Time Password (OTP) authentication.

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/rfc6560.

Table of Contents

1. Introduction

1.1. Scope

This document describes a Flexible Authentication Secure Tunneling (FAST) [RFC6113] factor that allows One-Time Password (OTP) values to be used in the Kerberos V5 [RFC4120] pre-authentication in a manner that does not require use of the user's Kerberos password. The system is designed to work with different types of OTP algorithms such as time-based OTPs [RFC2808], counter-based tokens [RFC4226] and challenge-response systems such as [RFC2289]. It is also designed to work with tokens that are electronically connected to the user's computer via means such as a USB interface.

This FAST factor provides the following facilities (as defined in [RFC6113]): client-authentication, replacing-reply-key, and KDC-authentication. It does not provide the strengthening-reply-key facility.

This proposal is partially based upon previous work on integrating single-use authentication mechanisms into Kerberos [HORENEZ004].

1.2. Overall Design

This proposal supports four- and two-pass variants. In the four-pass system, the client sends the Key Distribution Center (KDC) an initial AS-REQ, and the KDC responds with a KRB-ERROR containing pre-authentication data that includes a random nonce. The client then encrypts the nonce and returns it to the KDC in a second AS-REQ. Finally, the KDC returns the AS-REP. In the two-pass variant, the client encrypts a timestamp rather than a nonce from the KDC, and the encrypted data is sent to the KDC in the initial AS-REQ. The two-pass system can be used in cases where the client can determine in advance that OTP pre-authentication is supported by the KDC, which OTP key should be used and the encryption parameters required by the KDC.

In both systems, in order to create the message sent to the KDC, the client must generate the OTP value and two keys: the classic Reply Key used to decrypt the KDC's reply and a key to encrypt the data sent to the KDC. In most cases, the OTP value will be used in the key generation, but in order to support algorithms where the KDC cannot obtain the value (e.g., [RFC2289]), the system supports the option of including the OTP value in the request along with the encrypted nonce. In addition, in order to support situations where the KDC is unable to obtain the plaintext OTP value, the system also supports the use of hashed OTP values in the key derivation.

The pre-authentication data sent from the client to the KDC is sent within the encrypted data provided by the FAST pre-authentication data type of the AS-REQ. The KDC then obtains the OTP value, generates the same keys, and verifies the pre-authentication data by decrypting the nonce. If the verification succeeds, then it confirms knowledge of the Reply Key by using it to encrypt data in the AS-REP.

1.3. Conventions 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 assumes familiarity with the Kerberos pre-authentication framework [RFC6113] and so freely uses terminology and notation from that document.

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

2. Usage Overview

2.1. OTP Mechanism Support

As described above, this document describes a generic system for supporting different OTP mechanisms in Kerberos pre-authentication. To ensure interoperability, all implementations of this specification SHOULD provide a mechanism (e.g., a provider interface) to add or remove support for a particular OTP mechanism.

2.2. Pre-Authentication

The approach uses pre-authentication data in AS-REQ, AS-REP, and KRB-ERROR messages.

In the four-pass system, the client begins by sending an initial AS-REQ to the KDC that may contain pre-authentication data such as the standard Kerberos password data. The KDC will then determine, in an implementation dependent fashion, whether OTP authentication is required and if it is, it will respond with a KRB-ERROR message containing a PA-OTP-CHALLENGE (see Section 4.1) in the PA-DATA.

The PA-OTP-CHALLENGE will contain a KDC-generated nonce, a list of hash algorithm identifiers, and an iteration count if hashed OTP values are used (see Section 3.6) and OPTIONAL information on how the OTP should be generated by the client. The client will then generate the OTP value and two keys: a Client Key to encrypt the KDC's nonce and a Reply Key used to decrypt the KDC's reply.

As described in Section 5.4.1 of [RFC6113], the FAST system uses an Armor Key to set up an encrypted tunnel for use by FAST factors. As described in Section 3.6 of this document, the Client Key and Reply Key will be generated from the Armor Key and the OTP value, unless the OTP algorithm does not allow the KDC to obtain the OTP value. If hash algorithm identifiers were included in the PA-OTP-CHALLENGE, then the client will use the hash of the OTP value rather than the plaintext value in the key generation. Both keys will have the same encryption type as the Armor Key.

The generated Client Key will be used to encrypt the nonce received from the KDC. The encrypted value along with optional information on how the OTP was generated are then sent to the KDC in a PA-OTP-REQUEST (see Section 4.2) encrypted within the armored-data of a PA-FX-FAST-REQUEST PA-DATA element of a second AS-REQ.

In the two-pass system, the client sends the PA-OTP-REQUEST in the initial AS-REQ instead of sending it in response to a PA-OTP-CHALLENGE returned by the KDC. Since no challenge is received from the KDC, the client includes an encrypted timestamp in the request rather than the encrypted KDC nonce.

In both cases, on receipt of a PA-OTP-REQUEST, the KDC generates the keys in the same way as the client, and uses the generated Client Key to verify the pre-authentication by decrypting the encrypted data sent by the client (either nonce or timestamp). If the validation succeeds, then the KDC will authenticate itself to the client and confirm that the Reply Key has been updated by using the generated Reply Key in the AS-REP response.

2.3. PIN Change

Most OTP tokens involve the use of a Personal Identification Number (PIN) in the generation of the OTP value. This PIN value will be combined with the value generated by the token to produce the final OTP value that will be used in this protocol.

If, following successful validation of a PA-OTP-REQUEST in an AS-REQ, the KDC determines that the user's PIN has expired and needs to change, then it SHOULD respond with a KRB-ERROR of type KDC_ERR_PIN_EXPIRED. It MAY include formatting information on the PIN in a PA-OTP-PIN-CHANGE (see Section 4.3) encrypted within the armored-data of the PA-FX-FAST-REPLY PA-DATA element.

```
KDC_ERR_PIN_EXPIRED          96
```

If the PIN change is to be handled by a PIN-change service, then it is assumed that authentication to that service will succeed if the PIN has expired.

If the user's PIN has not expired but has been changed, then the KDC MAY return the new value to the client in a PA-OTP-PIN-CHANGE encrypted within the armored-data of the PA-FX-FAST-REPLY PA-DATA element of the AS-REP. Similarly, if a PIN change is not required, then the KDC MAY return a PA-OTP-PIN-CHANGE to inform the client of the current PIN's expiration time.

2.4. Resynchronization

It is possible with time- and event-based tokens that the OTP server will lose synchronization with the current token state. For example, event-based tokens may drift since the counter on the token is incremented every time the token is used, but the counter on the server is only incremented on an authentication. Similarly, the clocks on time-based tokens may drift.

Methods to recover from this type of situation are OTP algorithm-specific but may involve the client sending a sequence of OTP values to allow the server to further validate the correct position in its search window (see Section 7.4 of [RFC4226] for an example).

If, when processing a PA-OTP-REQUEST, the pre-authentication validation fails for this reason, then the KDC MAY return a KRB-ERROR message. The KRB-ERROR message MAY contain a PA-OTP-CHALLENGE in the PA-DATA with a single otp-tokenInfo representing the token used in the initial authentication attempt but with the "nextOTP" flag set. If this flag is set, then the client SHOULD re-try the authentication using an OTP value generated using the token in the "state" after that used in the failed authentication attempt, for example, using the next time interval or counter value.

3. Pre-Authentication Protocol Details

3.1. Initial Client Request

In the four-pass mode, the client begins by sending an initial AS-REQ, possibly containing other pre-authentication data. If the KDC determines that OTP-based pre-authentication is required and the request does not contain a PA-OTP-REQUEST, then it will respond as described in Section 3.2.

If the client has all the necessary information, it MAY use the two-pass system by constructing a PA-OTP-REQUEST as described in Section 3.3 and including it in the initial request.

3.2. KDC Challenge

If the user is required to authenticate using an OTP, then the KDC SHALL respond to the initial AS-REQ with a KRB-ERROR (as described in Section 2.2 of [RFC6113]), with a PA-OTP-CHALLENGE contained within the enc-fast-rep of the armored-data of a PA-FX-FAST-REPLY encrypted under the current Armor Key as described in [RFC6113].

If the OTP mechanism is to be carried out as an individual mechanism, then the PA-OTP-CHALLENGE SHALL be carried within the padata of the KrbFastResponse. Alternatively, if the OTP mechanism is required as part of an authentication set, then the PA-OTP-CHALLENGE SHALL be carried within a PA-AUTHENTICATION-SET-ELEM as described in Section 5.3 of [RFC6113].

The PA-OTP-CHALLENGE SHALL contain a nonce value to be returned encrypted in the client's PA-OTP-REQUEST. This nonce string MUST contain a randomly chosen component at least as long as the Armor Key length (see [RFC4086] for an in-depth discussion of randomness). In order to allow it to maintain any state necessary to verify the returned nonce, the KDC SHOULD use the mechanism described in Section 5.2 of [RFC6113].

The KDC MAY use the otp-service field to assist the client in locating the OTP token to be used by identifying the purpose of the authentication. For example, the otp-service field could assist a user in identifying the token to be used when a user has multiple OTP tokens that are used for different purposes. If the token is a connected device, then these values SHOULD be an exact octet-level match for the values present on the target token.

The KDC SHALL include a sequence of one or more otp-tokenInfo elements containing information on the token or tokens that the user can use for the authentication and how the OTP value is to be generated using those tokens. If a single otp-tokenInfo element is included, then only a single token is acceptable by the KDC, and any OTP value generated by the client MUST be generated according to the information contained within that element. If more than one otp-tokenInfo element is included, then the OTP value MUST be generated according to the information contained within one of those elements.

The KDC MAY include the otp-vendor field in an otp-tokenInfo to identify the vendor of the token that can be used in the authentication request in order to assist the client in locating that token.

If the KDC is able to obtain the OTP values for the token, then the OTP value SHOULD be used in the key generation as described in Section 3.6; therefore, the KDC SHOULD set the "must-encrypt-nonce" flag in the otp-tokenInfo. If the KDC is unable to obtain the OTP values for the token, then the "must-encrypt-nonce" flag MUST NOT be set. If the flag is not set, then the OTP value will be returned by the client in the otp-value field of the PA-OTP-REQUEST and so, if returning of OTP values in this way does not conform to KDC policy, then the KDC SHOULD NOT include the otp-tokenInfo for that token in the PA-OTP-CHALLENGE.

If the KDC requires that hashed OTPs be used in the key generation as described in Section 3.6 (for example, it is only able to obtain hashed OTP values for the token), then it MUST include the supported hash algorithms in order of preference in the supportedHashAlg of the otp-KeyInfo and the minimum value of the iteration count in the iterationCount element.

Since the OTP mechanism described in this document is replacing the Reply Key, the classic shared-key system cannot be relied upon to allow the client to verify the KDC. Therefore, as described in Section 3.4 of [RFC6113], some other mechanism must be provided to support this. If the OTP value is used in the Reply Key generation, then the client and KDC have a shared key and KDC-authentication is provided by the KDC using the Reply Key generated from the OTP value. However, if the OTP value is sent in the otp-value element of the PA-OTP-REQUEST, then there is no such shared key and the OTP mechanism does not provide KDC-authentication. Therefore, if the OTP mechanism is not being used in an environment where KDC-authentication is being provided by other means (e.g., by the use of a host-key-based Armor Key), then the KDC MUST NOT include any otp-tokenInfo elements in the PA-OTP-CHALLENGE that do not have the "must-encrypt-nonce" flag set.

If the OTP for a token is to be generated using a server-generated challenge, then the value of the challenge SHALL be included in the otp-challenge field of the otp-tokenInfo for that token. If the token is a connected device and the OTP is to be generated by combining the challenge with the token's current state (e.g., time), then the "combine" flag SHALL be set within the otp-tokenInfo containing the challenge.

If the KDC can determine which OTP token key (the seed value on the token used to generate the OTP) is to be used, then the otp-tokenID field MAY be included in the otp-tokenInfo to pass that value to the client.

The otp-algID field MAY be included in an otp-tokenInfo to identify the algorithm that should be used in the OTP calculation for that token. For example, it could be used when a user has been issued with multiple tokens that support different algorithms.

If the KDC can determine that an OTP token that can be used by the user does not require the client to collect a PIN, then it SHOULD set the "do-not-collect-pin" flag in the otp-tokenInfo representing that token. If the KDC can determine that the token requires the client to collect a PIN, then it SHOULD set the "collect-pin" flag. If the KDC is unable to determine whether or not the client should collect a PIN, then the "collect-pin" and "do-not-collect-pin" flags MUST NOT be set.

If the KDC requires the PIN of an OTP token to be returned to it separately, then it SHOULD set the "separate-pin-required" flag in the otp-KeyInfo representing that token.

If the KDC requires that the OTPs generated by the token have a Luhn check digit appended, as defined in [ISOIEC7812], then it MUST set the "check-digit" flag. This flag only applies if the format of the OTP is decimal; therefore, the otp-format field, if present, MUST have the value of "decimal".

Finally, in order to support connected tokens that can generate OTP values of varying lengths or formats, the KDC MAY include the desired otp-length and format of the OTP in the otp-length and otp-format fields of an otp-tokenInfo.

3.3. Client Response

The client response SHALL be sent to the KDC as a PA-OTP-REQUEST included within the enc-fast-req of the armored-data within a PA-FX-FAST-REQUEST encrypted under the current Armor Key as described in [RFC6113].

In order to generate its response, the client MUST generate an OTP value. If the PA-OTP-CHALLENGE contained one or more otp-tokenInfo elements, then the OTP value MUST be based on the information contained within one of those elements.

The otp-service, otp-vendor, otp-tokenID, otp-length, otp-format, and otp-algID elements of the PA-OTP-CHALLENGE are provided by the KDC to assist the client in locating the correct token to use, but the use of the above fields will depend on the type of token.

If the token is a disconnected device, then the values of otp-service and otp-vendor MAY be displayed to the user in order to help the user select the correct token, and the values of otp-algID, otp-tokenID, otp-length, and otp-format MAY be ignored.

If the token is a connected device, then these values, if present, SHOULD be used by the client to locate the correct token. When the token is connected, clients MUST support matching based on a binary comparison of the otp-vendor and otp-service strings when comparing the values against those present on the token. Clients MAY have other comparisons including normalization insensitive comparisons to try and find the right token. The values of otp-vendor and otp-service MAY be displayed to prompt the user if the correct token is not found.

If the "nextOTP" flag is set in the otp-tokenInfo from the PA-OTP-CHALLENGE, then the OTP value MUST be generated from the next token state rather than that used in the previous PA-OTP-REQUEST for that token. The "nextOTP" flag MUST also be set in the new PA-OTP-REQUEST.

If the "collect-pin" flag is set, then the token requires a PIN to be collected by the client. If the "do-not-collect-pin" flag is set in the otp-tokenInfo from the PA-OTP-CHALLENGE, then the token represented by the otp-tokenInfo does not require a PIN to be collected by the client as part of the OTP value. If neither of the "collect-pin" nor "do-not-collect-pin" flags are set, then PIN requirements of the token are unspecified. If both flags are set, then the client SHALL regard the request as invalid.

If the "separate-pin-required" flag is set, then any PIN collected by the client MUST be included as a UTF-8 string in the otp-pin of the PA-OTP-REQUEST.

If the token is a connected device, then how the PIN is used to generate the OTP value will depend on the type of device. However, if the token is a disconnected device, then it will depend on the "separate-pin-required" flag. If the flag is not set, then the OTP value MUST be generated by appending the PIN with the value from the token entered by the user and, if the flag is set, then the OTP value MUST be the value from the token.

The clients SHOULD NOT normalize the PIN value or any OTP value collected from the user or returned by a connected token in any way.

If the "check-digit" flag is set, then any OTP values SHOULD be decimal and have a Luhn check digit appended [ISOIEC7812]. If the token is disconnected, then the Client MAY ignore this flag; if the token is connected, then the Client MUST enforce it. The Client MUST regard the request as invalid, if otp-format is present and set to any value other than "decimal".

If an otp-challenge is present in the otp-tokenInfo selected by the client from the PA-OTP-CHALLENGE, then the OTP value for the token MUST be generated based on a challenge, if the token is capable of accepting a challenge. The client MAY ignore the provided challenge if and only if the token is not capable of including a challenge in the OTP calculation.

If the "combine" flag is not set in the otp-tokenInfo of the PA-OTP-CHALLENGE, then the OTP SHALL be calculated based only the challenge and not the internal state (e.g., time or counter) of the token. If the "combine" flag is set, then the OTP SHALL be calculated using both the internal state and the provided challenge, if that value is obtainable by the client. If the flag is set but otp-challenge is not present, then the client SHALL regard the request as invalid.

If token is a connected device, then the use of the challenge will depend on the type of device but will involve passing the challenge and the value of the "combine" flag in a token-specific manner to the token, along with a PIN if collected and the values of otp-length and otp-format if specified, in order to obtain the OTP value. If the token is disconnected, then the challenge MUST be displayed to the user and the value of the "combine" flag MAY be ignored by the client.

If the OTP value was generated using a challenge that was not sent by the KDC, then the challenge SHALL be included in the otp-challenge of the PA-OTP-REQUEST. If the OTP was generated by combining a challenge (either received from the KDC or generated by the client) with the token state, then the "combine" flag SHALL be set in the PA-OTP-REQUEST.

If the "must-encrypt-nonce" flag is set in the otp-tokenInfo, then the OTP value MUST be used to generate the Client Key and Reply Key as described in Section 3.6 and MUST NOT be included in the otp-value field of the PA-OTP-REQUEST. If the flag is not set, then the OTP value MUST be included in the otp-value field of the PA-OTP-REQUEST and MUST NOT be used in the key derivation. In this case, the Client Key and Reply Key SHALL be the same as the Armor Key as described in Section 3.6; so, if the returning of OTP values in this way does not conform to local policy on the client (for example, if KDC-Authentication is required and is not being provided by other means), then it SHOULD NOT use the token for authentication.

If the supportedHashAlg and iterationCount elements are included in the otp-tokenInfo, then the client MUST use hashed OTP values in the generation of the Reply Key and Client Key as described in Section 3.6. The client MUST select the first algorithm from the list that it supports and the AlgorithmIdentifer [RFC5280] selected MUST be placed in the hashAlg element of the PA-OTP-REQUEST. However, if none of the algorithm identifiers conform to local policy restrictions, then the authentication attempt MUST NOT proceed using that token. If the value of iterationCount does not conform to local policy on the client, then the client MAY use a larger value, but MUST NOT use a lower value. The value of the iteration count used by the client MUST be returned in the PA-OTP-REQUEST sent to the KDC.

If hashed OTP values are used, then the nonce generated by the client MUST be as long as the longest key length of the symmetric key types that it supports and MUST be chosen randomly (see [RFC4086]). The nonce MUST be included in the PA-OTP-REQUEST, along with the hash algorithm and iteration count used in the nonce, hashAlg, and iterationCount fields of the PA-OTP-REQUEST. These fields MUST NOT be included if hashed OTP values were not used. It is RECOMMENDED that the iteration count used by the client be chosen in such a way that it is computationally infeasible/unattractive for an attacker to brute-force search for the given OTP.

The PA-OTP-REQUEST returned by the client SHOULD include information on the generated OTP value reported by the OTP token when available to the client. The otp-time and otp-counter fields of the PA-OTP-REQUEST SHOULD be used to return the time and counter values used by the token if available to the client. The otp-format field MAY be used to report the format of the generated OTP. This field SHOULD be used if a token can generate OTP values in multiple formats. The otp-algID field SHOULD be used by the client to report the algorithm used in the OTP calculation, and the otp-tokenID SHOULD be used to report the identifier of the OTP token key used if the information is known to the client.

If the PA-OTP-REQUEST is being sent in response to a PA-OTP-CHALLENGE that contained an otp-vendor field in the selected otp-tokenInfo, then the otp-vendor field of the PA-OTP-REQUEST MUST be set to the same value. If no otp-vendor field was provided by the KDC, then the field SHOULD be set to the vendor identifier of the token if known to the client.

The generated Client Key is used by the client to encrypt data to be included in the encData of the PA-OTP-REQUEST to allow the KDC to authenticate the user. The key usage for this encryption is KEY_USAGE_OTP_REQUEST.

- If the PA-OTP-REQUEST is being generated in response to a PA-OTP-CHALLENGE returned by the KDC, then the client SHALL encrypt a PA-OTP-ENC-REQUEST containing the value of nonce from the PA-OTP-CHALLENGE using the same encryption type as the Armor Key.

- If the PA-OTP-REQUEST is not in response to a PA-OTP-CHALLENGE, then the client SHALL encrypt a PA-ENC-TS-ENC containing the current time as in the encrypted timestamp pre-authentication mechanism [RFC4120].

If the client is working in two-pass mode and so, is not responding to an initial KDC challenge, then the values of the iteration count and hash algorithms cannot be obtained from that challenge. The client SHOULD use any values obtained from a previous PA-OTP-CHALLENGE or, if no values are available, it MAY use initial configured values.

3.4. Verifying the Pre-Authentication Data

The KDC validates the pre-authentication data by generating the Client Key and Reply Key in the same way as the client and using the generated Client Key to decrypt the value of encData from the PA-OTP-REQUEST. The generated Reply Key is used to encrypt data in the AS-REP.

If the otp-value field is included in the PA-OTP-REQUEST, then the KDC MUST use that value unless the OTP method is required to support KDC-authentication (see Section 3.2). If the otp-value is not included in the PA-OTP-REQUEST, then the KDC will need to generate or obtain the OTP value.

If the otp-pin field is present in the PA-OTP-REQUEST, then the PIN value has to be value provided by the client. The KDC SHOULD SASLPrep (Stringprep Profile for User Names and Passwords) [RFC4013] the value in lookup mode before comparison.

It should be noted that it is anticipated that, as improved string comparison technologies are standardized, the processing done by the KDC will change, but efforts will be made to maintain as much compatibility with SASLprep as possible.

If the otp-challenge field is present, then the OTP was calculated using that challenge. If the "combine" flag is also set, then the OTP was calculated using the challenge and the token's current state.

It is RECOMMENDED that the KDC act upon the values of otp-time, otp-counter, otp-format, otp-algID, and otp-tokenID if they are present in the PA-OTP-REQUEST. If the KDC receives a request containing these values, but cannot act upon them, then they MAY be ignored.

The KDC generates the Client Key and Reply Key as described in Section 3.6 from the OTP value using the nonce, hash algorithm, and iteration count if present in the PA-OTP-REQUEST. The KDC MUST fail the request with KDC_ERR_INVALID_HASH_ALG, if the KDC requires hashed OTP values and the hashAlg field was not present in the PA-OTP-REQUEST or if the value of this field does not conform to local KDC policy. Similarly, the KDC MUST fail the request with KDC_ERR_INVALID_ITERATION_COUNT, if the value of the iterationCount included in the PA-OTP-REQUEST does not conform to local KDC policy or is less than that specified in the PA-OTP-CHALLENGE. In addition, the KDC MUST fail the authentication request with KDC_ERR_PIN_REQUIRED, if it requires a separate PIN to the OTP value and an otp-pin was not included in the PA-OTP-REQUEST. The above error codes are defined as follows:

KDC_ERR_INVALID_HASH_ALG	94
KDC_ERR_INVALID_ITERATION_COUNT	95
KDC_ERR_PIN_REQUIRED	97

The generated Client Key is then used to decrypt the encData from the PA-OTP-REQUEST. If the client response was sent as a result of a PA-OTP-CHALLENGE, then the decrypted data will be a PA-OTP-ENC-REQUEST and the client authentication MUST fail with KDC_ERR_PREAUTH_FAILED if the nonce value from the PA-OTP-ENC-REQUEST is not the same as the nonce value sent in the PA-OTP-CHALLENGE. If the response was not sent as a result of a PA-OTP-CHALLENGE, then the decrypted value will be a PA-ENC-TS-ENC, and the authentication process will be the same as with classic encrypted timestamp pre-authentication [RFC4120].

The KDC MUST fail the request with KDC_ERR_ETYPE_NOSUPP, if the encryption type used by the client in the encData does not conform to KDC policy.

If authentication fails due to the hash algorithm, iteration count, or encryption type used by the client, then the KDC SHOULD return a PA-OTP-CHALLENGE with the required values in the error response. If the authentication fails due to the token state on the server is no longer being synchronized with the token used, then the KDC MAY return a PA-OTP-CHALLENGE with the "nextOTP" flag set as described in Section 2.4.

If, during the authentication process, the KDC determines that the user's PIN has been changed, then it SHOULD include a PA-OTP-PIN-CHANGE in the response, as described in Section 2.3, containing the new PIN value. The KDC MAY also include the new PIN's expiration time and the expiration time of the OTP account within the last-req field of the PA-OTP-PIN-CHANGE. (These fields can be used by the KDC to handle cases where the account related to the user's OTP token has a different expiration time to the user's Kerberos account.) If the KDC determines that the user's PIN or OTP account are about to expire, it MAY return a PA-OTP-PIN-CHANGE with that information. Finally, if the KDC determines that the user's PIN has expired, then it SHOULD return a KRB-ERROR of type KDC_ERR_PIN_EXPIRED as described in Section 2.3

3.5. Confirming the Reply Key Change

If the pre-authentication data was successfully verified, then, in order to support mutual authentication, the KDC SHALL respond to the client's PA-OTP-REQUEST by using the generated Reply Key to encrypt the data in the AS-REP. The client then uses its generated Reply Key to decrypt the encrypted data and MUST NOT continue with the authentication process, if decryption is not successful.

3.6. Reply Key Generation

In order to authenticate the user, the client and KDC need to generate two encryption keys:

- The Client Key to be used by the client to encrypt and by the KDC to decrypt the encData in the PA-OTP-REQUEST.

- The Reply Key to be used in the standard manner by the KDC to encrypt data in the AS-REP.

  The method used to generate the two keys will depend on the OTP algorithm.

- If the OTP value is included in the otp-value of the PA-OTP-REQUEST, then the two keys SHALL be the same as the Armor Key (defined in [RFC6113]).

- If the OTP value is not included in the otp-value of the PA-OTP-REQUEST, then the two keys SHALL be derived from the Armor Key and the OTP value as described below.

If the OTP value is not included in the PA-OTP-REQUEST, then the Reply Key and Client Key SHALL be generated using the KRB-FX-CF2 algorithm from [RFC6113] as follows:

```
Client Key = KRB-FX-CF2(K1, K2, O1, O2)
Reply Key = KRB-FX-CF2(K1, K2, O3, O4)
```

The octet string parameters, O1, O2, O3, and O4 shall be the ASCII string "OTPComb1", "OTPComb2", "OTPComb3", and "OTPComb4" as shown below:

```
{0x4f, 0x54, 0x50, 0x43, 0x6f, 0x6d, 0x62, 0x31}
{0x4f, 0x54, 0x50, 0x43, 0x6f, 0x6d, 0x62, 0x32}
{0x4f, 0x54, 0x50, 0x43, 0x6f, 0x6d, 0x62, 0x33}
{0x4f, 0x54, 0x50, 0x43, 0x6f, 0x6d, 0x62, 0x34}
```

The first input key, K1, SHALL be the Armor Key and so, as described in Section 5.1 of [RFC6113], the enctypes of the generated Client Key and Reply Key will be the same as the enctype of Armor Key. The second input key, K2, shall be derived from the OTP value using string-to-key (defined in [RFC3961]) as described below.

If the hash of the OTP value is to be used, then K2 SHALL be derived as follows:

- An initial hash value, H, is generated:

```
H = hash(realm|nonce|OTP)
```

Where:

* "|" denotes concatenation.
* hash is the hash algorithm selected by the client.
* realm is the name of the server's realm as carried in the realm field of the AS-REQ (not including the tag and length from the DER encoding).
* nonce is the value of the random nonce value generated by the client and carried in the nonce field of the PA-OTP-REQUEST (not including the tag and length from the DER encoding).
* If the OTP format is decimal, hexadecimal, or alphanumeric, then OTP is the value of the OTP generated as described in Section 3.3 with SASLprep [RFC4013] applied in lookup mode; otherwise, it is the unnormalized OTP value.

- The initial hash value is then hashed iterationCount-1 times to produce a final hash value, H' (where iterationCount is the value from the PA-OTP-REQUEST).

```
H' = hash(hash(...(iterationCount-1 times)...(H)))
```

- The value of K2 is then derived from the Base64 [RFC2045] encoding of this final hash value.

```
K2 = string-to-key(Base64(H')|"Krb-preAuth")
```

If the hash value is not used, then K2 SHALL be derived from the base64 encoding of the OTP value.

```
K2 = string-to-key(Base64(OTP)|"Krb-preAuth")
```

The enctype used for string-to-key SHALL be that of the Armor Key and the salt and any additional parameters for string-to-key MAY be provided by the KDC in the PA-OTP-CHALLENGE. If the salt and string-to-key parameters are not provided, then the default values defined for the particular enctype SHALL be used.

If the strengthen-key is present in KrbFastResponse, then it is combined with the Reply Key to generate the final AS-REQ as described in [RFC6113]. The strengthen-key does not influence the Client Key.

4. OTP Kerberos Message Types

4.1. PA-OTP-CHALLENGE

The padata-type PA-OTP-CHALLENGE is returned by the KDC to the client in the enc-fast-rep of a PA-FX-FAST-REPLY in the PA-DATA of a KRB-ERROR when OTP pre-authentication is required. The corresponding padata-value field contains the Distinguished Encoding Rules (DER) [X.680] and [X.690] encoding of a PA-OTP-CHALLENGE containing a server-generated nonce and information for the client on how to generate the OTP.

```
PA-OTP-CHALLENGE     141

PA-OTP-CHALLENGE ::= SEQUENCE {
  nonce             [0] OCTET STRING,
  otp-service       [1] UTF8String                OPTIONAL,
  otp-tokenInfo     [2] SEQUENCE (SIZE(1..MAX)) OF
                                              OTP-TOKENINFO,
  salt              [3] KerberosString            OPTIONAL,
  s2kparams         [4] OCTET STRING              OPTIONAL,
  ...
```

```
}

OTP-TOKENINFO ::= SEQUENCE {
  flags             [0] OTPFlags,
  otp-vendor        [1] UTF8String                OPTIONAL,
  otp-challenge     [2] OCTET STRING (SIZE(1..MAX))
                                                  OPTIONAL,
  otp-length        [3] Int32                     OPTIONAL,
  otp-format        [4] OTPFormat                 OPTIONAL,
  otp-tokenID       [5] OCTET STRING              OPTIONAL,
  otp-algID         [6] AnyURI                    OPTIONAL,
  supportedHashAlg [7] SEQUENCE OF AlgorithmIdentifier
                                                  OPTIONAL,
  iterationCount    [8] Int32                     OPTIONAL,
  ...
}

OTPFormat ::= INTEGER {
  decimal(0),
  hexadecimal(1),
  alphanumeric(2),
  binary(3),
  base64(4)
}

OTPFlags ::= KerberosFlags
-- reserved(0),
-- nextOTP(1),
-- combine(2),
-- collect-pin(3),
-- do-not-collect-pin(4),
-- must-encrypt-nonce (5),
-- separate-pin-required (6),
-- check-digit (7)
```

nonce
: A KDC-supplied nonce value to be encrypted by the client in the PA-OTP-REQUEST. This nonce string MUST contain a randomly chosen component at least as long as the Armor Key length.

otp-service
: Use of this field is OPTIONAL, but MAY be used by the KDC to assist the client to locate the appropriate OTP tokens to be used. For example, this field could be used when a user has multiple OTP tokens for different purposes.

otp-tokenInfo
This element MUST be included, and it is a sequence of one or more OTP-TOKENINFO objects containing information on the token or tokens that the user can use for the authentication and how the OTP value is to be generated using those tokens. If a single OTP-TOKENINFO object is included, then only a single token is acceptable by the KDC and any OTP value generated by the client MUST be generated according to the information contained within that element. If more than one OTP-TOKENINFO object is included, then the OTP value MUST be generated according to the information contained within one of those objects.

flags
If the "nextOTP" flag is set, then the OTP SHALL be based on the next token "state" rather than the one used in the previous authentication. As an example, for a time-based token, this means the next time slot and for an event-based token, this could mean the next counter value. If the "nextOTP" flag is set, then there MUST only be a single otp-tokenInfo element in the PA-OTP-CHALLENGE.

The "combine" flag controls how the challenge included in otp-challenge shall be used. If the flag is set, then OTP SHALL be calculated using the challenge from otp-challenge and the internal token state (e.g., time or counter). If the "combine" flag is not set, then the OTP SHALL be calculated based only on the challenge. If the flag is set and otp-challenge is not present, then the request SHALL be regarded as invalid.

If the "do-not-collect-pin" flag is set, then the token represented by the current otp-tokenInfo does not require a PIN to be collected as part of the OTP. If the "collect-pin" flag is set, then the token requires a PIN. If neither flag is set, then whether or not a PIN is required is unspecified. The flags are mutually exclusive and so both flags MUST NOT be set, or the client MUST regard the request as invalid.

If the "must-encrypt-nonce" flag is set, then the OTP value MUST NOT be included in the otp-value field of the PA-OTP-REQUEST, but instead the OTP value MUST be used in the generation of the Reply Key and Client Key as described in Section 3.6.

If the "separate-pin-required" flag is set, then the PIN collected by the client SHOULD NOT be used in the generation of the OTP value and SHOULD be returned in the otp-pin field of the PA-OTP-REQUEST.

The "check-digit" flag controls whether or not the OTP values generated by the token need to include a Luhn check digit [ISOIEC7812]. If the token is disconnected, then the Client MAY ignore this flag; if this flag is set and the token is connected, then the OTP MUST be a decimal with a check digit appended.

otp-vendor
Use of this field is OPTIONAL, but MAY be used by the KDC to identify the vendor of the OTP token to be used.

otp-challenge
The otp-challenge is used by the KDC to send a challenge value for use in the OTP calculation. The challenge is an OPTIONAL octet string that SHOULD be uniquely generated for each request in which it is present. When the challenge is not present, the OTP will be calculated on the current token state only. The client MAY ignore a provided challenge if and only if the OTP token the client is interacting with is not capable of including a challenge in the OTP calculation. In this case, KDC policies will determine whether or not to accept a provided OTP value.

otp-length
Use of this field is OPTIONAL, but MAY be used by the KDC to specify the desired length of the generated OTP. For example, this field could be used when a token is capable of producing OTP values of different lengths. If the format of the OTP is 'decimal', 'hexidecimal', or 'alphanumeric', then this value indicates the desired length in digits/characters; if the OTP format is 'binary', then this value indicates the desired length in octets; and if the OTP format is 'base64', then this value indicates the desired length of the unencoded OTP value in octets.

otp-format
Use of this field is OPTIONAL, but MAY be used by the KDC to specify the desired format of the generated OTP value. For example, this field could be used when a token is capable of producing OTP values of different formats.

otp-tokenID
Use of this field is OPTIONAL, but MAY be used by the KDC to identify which token key should be used for the authentication. For example, this field could be used when a user has been issued multiple token keys by the same server.

otp-algID
Use of this field is OPTIONAL, but MAY be used by the KDC to identify the algorithm to use when generating the OTP. The value of this field MUST be a URI [RFC3986] and SHOULD be obtained from the Portable Symmetric Key Container (PSKC) algorithm registry [RFC6030].

supportedHashAlg
If present, then a hash of the OTP value MUST be used in the key derivation rather than the plain text value. Each AlgorithmIdentifier identifies a hash algorithm that is supported by the KDC in decreasing order of preference. The client MUST select the first algorithm from the list that it supports. Support for SHA-256 by both the client and KDC is REQUIRED. The AlgorithmIdentifier selected by the client MUST be placed in the hashAlg element of the PA-OTP-REQUEST.

iterationCount
The minimum value of the iteration count to be used by the client when hashing the OTP value. This value MUST be present if supportedHashAlg is present and otherwise MUST NOT be present. If the value of this element does not conform to local policy on the client, then the client MAY use a larger value but MUST NOT use a lower value. The value of the iteration count used by the client MUST be returned in the PA-OTP-REQUEST sent to the KDC.

salt
The salt value to be used in string-to-key when used to calculate the keys as described in Section 3.6.

s2kparams
Any additional parameters required by string-to-key as described in Section 3.6.

4.2. PA-OTP-REQUEST

The padata-type PA-OTP-REQUEST is sent by the client to the KDC in the KrbFastReq padata of a PA-FX-FAST-REQUEST that is included in the PA-DATA of an AS-REQ. The corresponding padata-value field contains the DER encoding of a PA-OTP-REQUEST.

The message contains pre-authentication data encrypted by the client using the generated Client Key and optional information on how the OTP was generated. It may also, optionally, contain the generated OTP value.

```
PA-OTP-REQUEST     142

PA-OTP-REQUEST ::= SEQUENCE {
  flags          [0]  OTPFlags,
  nonce          [1]  OCTET STRING                 OPTIONAL,
  encData        [2]  EncryptedData,
                      -- PA-OTP-ENC-REQUEST or PA-ENC-TS-ENC
                      -- Key usage of KEY_USAGE_OTP_REQUEST
  hashAlg        [3]  AlgorithmIdentifier          OPTIONAL,
  iterationCount [4]  Int32                        OPTIONAL,
  otp-value      [5]  OCTET STRING                 OPTIONAL,
  otp-pin        [6]  UTF8String                   OPTIONAL,
  otp-challenge  [7]  OCTET STRING (SIZE(1..MAX)) OPTIONAL,
  otp-time       [8]  KerberosTime                 OPTIONAL,
  otp-counter    [9]  OCTET STRING                 OPTIONAL,
  otp-format     [10] OTPFormat                    OPTIONAL,
  otp-tokenID    [11] OCTET STRING                 OPTIONAL,
  otp-algID      [12] AnyURI                       OPTIONAL,
  otp-vendor     [13] UTF8String                   OPTIONAL,
  ...
}

KEY_USAGE_OTP_REQUEST  45

PA-OTP-ENC-REQUEST ::= SEQUENCE {
   nonce     [0] OCTET STRING,
   ...
}
```

flags
: This field MUST be present.

  If the "nextOTP" flag is set, then the OTP was calculated based on the next token "state" rather than the current one. This flag MUST be set if and only if it was set in a corresponding PA-OTP-CHALLENGE.

  If the "combine" flag is set, then the OTP was calculated based on a challenge and the token state.

  No other OTPFlag bits are applicable and MUST be ignored by the KDC.

nonce
: This field MUST be present if a hashed OTP value was used as input to string-to-key (see Section 3.6) and MUST NOT be present otherwise. If present, it MUST contain the nonce value generated by the client and used in the generation of hashed OTP values as

described in Section 3.6. This nonce string MUST be as long as the longest key length of the symmetric key types that the client supports and MUST be chosen randomly.

encData
This field MUST be present and MUST contain the pre-authentication data encrypted under the Client Key with a key usage of KEY_USAGE_OTP_REQUEST. If the PA-OTP-REQUEST is sent as a result of a PA-OTP-CHALLENGE, then this MUST contain a PA-OTP-ENC-REQUEST with the nonce from the PA-OTP-CHALLENGE. If no challenge was received, then this MUST contain a PA-ENC-TS-ENC.

hashAlg
This field MUST be present if a hashed OTP value was used as input to string-to-key (see Section 3.6) and MUST NOT be present otherwise. If present, it MUST contain the AlgorithmIdentifier of the hash algorithm used. If the PA-OTP-REQUEST is sent as a result of a PA-OTP-CHALLENGE, then the AlgorithmIdentifer MUST be the first one supported by the client from the supportedHashAlg of the PA-OTP-CHALLENGE.

iterationCount
This field MUST be present if a hashed OTP value was used as input to string-to-key (see Section 3.6) and MUST NOT be present otherwise. If present, it MUST contain the iteration count used when hashing the OTP value. If the PA-OTP-REQUEST is sent as a result of a PA-OTP-CHALLENGE, then the value MUST NOT be less that specified in the PA-OTP-CHALLENGE.

otp-value
The generated OTP value. This value MUST NOT be present if the "must-encrypt-nonce" flag was set in the PA-OTP-CHALLENGE.

otp-pin
The OTP PIN value entered by the user. This value MUST NOT be present unless the "separate-pin-required" flag was set in the PA-OTP-CHALLENGE.

otp-challenge
Value used by the client in the OTP calculation. It MUST be sent to the KDC if and only if the value would otherwise be unknown to the KDC (for example, the token- or client-modified or generated challenge).

otp-time
This field MAY be included by the client to carry the time value as reported by the OTP token. Use of this element is OPTIONAL, but it MAY be used by a client to simplify the OTP calculations carried out by the KDC. It is RECOMMENDED that the KDC act upon this value if it is present in the request and it is capable of using it in the generation of the OTP value.

otp-counter
This field MAY be included by the client to carry the token counter value, as reported by the OTP token. Use of this element is OPTIONAL, but it MAY be used by a client to simplify the OTP calculations carried out by the KDC. It is RECOMMENDED that the KDC act upon this value if it is present in the request and it is capable of using it in the generation of the OTP value.

otp-format
This field MAY be used by the client to send the format of the generated OTP as reported by the OTP token. Use of this element is OPTIONAL, but it MAY be used by the client to simplify the OTP calculations carried out by the KDC. It is RECOMMENDED that the KDC act upon this value, if it is present in the request and it is capable of using it in the generation of the OTP value.

otp-tokenID
This field MAY be used by the client to send the identifier of the token key used, as reported by the OTP token. Use of this field is OPTIONAL, but MAY be used by the client to simplify the authentication process by identifying a particular token key associated with the user. It is RECOMMENDED that the KDC act upon this value, if it is present in the request and it is capable of using it in the generation of the OTP value.

otp-algID
This field MAY be used by the client to send the identifier of the OTP algorithm used, as reported by the OTP token. Use of this element is OPTIONAL, but it MAY be used by the client to simplify the OTP calculations carried out by the KDC. It is RECOMMENDED that the KDC act upon this value, if it is present in the request and it is capable of using it in the generation of the OTP value. The value of this field MUST be a URI [RFC3986] and SHOULD be obtained from the PSKC algorithm registry [RFC6030].

otp-vendor
If the PA-OTP-REQUEST is being sent in response to a PA-OTP-CHALLENGE that contained an otp-vendor field in the selected otp-tokenInfo, then this field MUST be set to the same value; otherwise, this field SHOULD be set to the vendor identifier of the token, if known to the client. It is RECOMMENDED that the KDC act upon this value if it is present in the request and it is capable of using it in the generation of the OTP value.

4.3. PA-OTP-PIN-CHANGE

The padata-type PA-OTP-PIN-CHANGE is returned by the KDC in the enc-fast-rep of a PA-FX-FAST-REPLY in the AS-REP if the user must change their PIN, if the user's PIN has been changed, or to notify the user of the PIN's expiry time.

The corresponding padata-value field contains the DER encoding of a PA-OTP-PIN-CHANGE.

```
PA-OTP-PIN-CHANGE     144

PA-OTP-PIN-CHANGE ::= SEQUENCE {
  flags     [0] PinFlags,
  pin       [1] UTF8String OPTIONAL,
  minLength [2] INTEGER    OPTIONAL,
  maxLength [3] INTEGER    OPTIONAL,
  last-req  [4] LastReq    OPTIONAL,
  format    [5] OTPFormat  OPTIONAL,
  ...
}

PinFlags ::= KerberosFlags
  -- reserved(0),
  -- systemSetPin(1),
  -- mandatory(2)
```

flags
The "systemSetPin" flag is used to indicate the type of PIN change that is taking place. If the flag is set, then the user's PIN has been changed for the user by the system. If the flag is not set, then the user's PIN needs to be changed by the user.

If the "systemSetPin" flag is not set and the "mandatory" flag is set, then user PIN change is required before the next authentication using the current OTP token. If the "mandatory" flag is not set, then the user PIN change is optional. If the

"systemSetPin" flag is set, then the "mandatory" flag has no meaning and SHOULD be ignored by the client.

pin
The pin field is used to carry a new PIN value. If the "systemSetPin" flag is set, then the pin field is used to carry the new PIN value set for the user and MUST be present. If the "systemSetPin" flag is not set, then use of this field is OPTIONAL and MAY be used to carry a system-generated PIN that MAY be used by the user when changing the PIN.

minLength and maxLength
Use of the minLength and maxLength fields is OPTIONAL. If the "systemSetPin" flag is not set, then these fields MAY be included to pass restrictions on the size of the user-selected PIN.

last-req
Use of the last-req field (as defined in Section 5.4.2 of [RFC4120])) is OPTIONAL, but MAY be included with an lr-type of 6 to carry PIN expiration information.

* If the "systemSetPin" flag is set, then the expiration time MUST be that of the new system-set PIN.

* If the "systemSetPin" flag is not set, then the expiration time MUST be that of the current PIN of the token used in the authentication.

The element MAY also be included with an lr-type of 7 to indicate when the OTP account will expire.

format
The format element MAY be included by the KDC to carry format restrictions on the new PIN.

* If the "systemSetPin" flag is set, then the element MUST describe the format of the new system-generated PIN.

* If the "systemSetPin" flag is not set, then the element MUST describe restrictions on any new user-generated PIN.

5. IANA Considerations

The OTP algorithm identifier URIs used as otp-algID values in the PA-OTP-CHALLENGE described in Section 4.1 and the PA-OTP-REQUEST described in Section 4.2 have been registered in the "Algorithm URI Registry and Related PSKC Profiles" registry [RFC6030].

The following pre-authentication types are defined in this document:

PA-OTP-CHALLENGE	141
PA-OTP-REQUEST	142
PA-OTP-PIN-CHANGE	144

Note that PA-OTP-CONFIRM (143) has been marked as OBSOLETE per this document.

These values are currently registered in a registry created by [RFC6113], but the entries have been updated to refer to this document.

The following error codes and key usage values are defined in this document:

KDC_ERR_INVALID_HASH_ALG	94
KDC_ERR_INVALID_ITERATION_COUNT	95
KDC_ERR_PIN_EXPIRED	96
KDC_ERR_PIN_REQUIRED	97
KEY_USAGE_OTP_REQUEST	45

These values are currently not managed by IANA and have not been accounted for. There is currently work in progress [LHA10] to define IANA registries and a registration process for these values.

6. Security Considerations

6.1. Man-in-the-Middle Attacks

In the system described in this document, the OTP pre-authentication protocol is tunneled within the FAST Armor channel provided by the pre-authentication framework. As described in [ASNINY02], tunneled protocols are potentially vulnerable to man-in-the-middle (MITM) attacks if the outer tunnel is compromised, and it is generally considered good practice in such cases to bind the inner encryption to the outer tunnel.

In order to mitigate against such attacks, the proposed system uses the outer Armor Key in the derivation of the inner Client and Reply keys and so achieves crypto-binding to the outer channel.

As described in Section 5.4 of [RFC6113], FAST can use an anonymous Ticket-Granting Ticket (TGT) obtained using anonymous Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) [RFC6112] [RFC4556] as the Armor Key. However, the current anonymous PKINIT proposal is open to MITM attacks since the attacker

can choose a session key such that the session key between the MITM and the real KDC is the same as the session key between the client and the MITM.

As described in Section 3.6, if the OTP value is not being sent to the KDC, then the Armor Key is used along with the OTP value in the generation of the Client Key and Reply Key. If the Armor Key is known, then the only entropy remaining in the key generation is provided by the OTP value. If the OTP algorithm requires that the OTP value be sent to the KDC, then it is sent encrypted within the tunnel provided by the FAST Armor and so, is exposed to the attacker if the attacker has the Armor Key.

Therefore, unless the identity of the KDC has been verified, anonymous PKINIT SHALL NOT be used with OTP algorithms that require the OTP value to be sent to the KDC. In addition, the security considerations should be carefully considered before anonymous PKINIT is used with other algorithms such as those with short OTP values.

Careful consideration should also be made if host key armor is used to provide the KDC-authentication facility with OTP algorithms where the OTP value is sent within the otp-value field of the PA-OTP-REQUEST since compromised host keys would allow an attacker to impersonate the KDC.

6.2. Reflection

The four-pass system described above is a challenge-response protocol, and such protocols are potentially vulnerable to reflection attacks. No such attacks are known at this point, but to help mitigate against such attacks, the system uses different keys to encrypt the client and server nonces.

6.3. Denial-of-Service Attacks

The protocol supports the use of an iteration count in the generation of the Client and Reply keys, and the client can send the number of iterations used as part of the PA-OTP-REQUEST. This could open the KDC up to a denial-of-service attack if a large value for the iteration count was specified by the attacker. It is therefore, particularly important that, as described in Section 3.4, the KDC reject any authentication requests where the iteration count is above a maximum value specified by local policy.

6.4. Replay

In the four-pass version of this protocol, the client encrypts a KDC-generated nonce, so replay can be detected by the KDC. The two-pass version of the protocol does not involve a server nonce; the client instead encrypts a timestamp, and therefore is not protected from replay in this way, but it will instead require some other mechanism, such as an OTP-server-based system or a timestamp-based replay cache on the KDC.

As described in Section 5.2 of [RFC6113], a client cannot be certain that it will use the same KDC for all messages in a conversation. Therefore, the client cannot assume that the PA-OTP-REQUEST will be sent to the same KDC that issued the PA-OTP-CHALLENGE. In order to support this, a KDC implementing this protocol requires a means of sharing session state. However, doing this does introduce the possibility of a replay attack where the same response is sent to multiple KDCs.

In the case of time- or event-based tokens or server-generated challenges, protection against replay may be provided by the OTP server being used if that server is capable of keeping track of the last used value. This protection therefore relies upon the assumption that the OTP server being used in this protocol is either not redundant or involves a commit protocol to synchronize between replicas. If this does not hold for an OTP server being used, then the system may be vulnerable to replay attacks.

However, OTP servers may not be able to detect replay of OTPs generated using only a client-generated challenge; since, the KDC would not be able to detect replay in two-pass mode, it is recommended that the use of OTPs generated from only a client-generated challenge (that is, not in combination with some other factor such as time) should not be supported in two-pass mode.

6.5. Brute-Force Attack

A compromised or hostile KDC may be able to obtain the OTP value used by the client via a brute-force attack. If the OTP value is short, then the KDC could iterate over the possible OTP values until a Client Key is generated that can decrypt the encData sent in the PA-OTP-REQUEST.

As described in Section 3.6, an iteration count can be used in the generation of the Client Key and the value of the iteration count can be controlled by local client policy. Use of this iteration count can make it computationally infeasible/unattractive for an attacker to brute-force search for the given OTP within the lifetime of that OTP.

If PINs contain international characters, similar looking or similar functioning characters may be mapped together. For example, the combined and decomposed forms of accented characters will typically be treated the same. Users who attempt to exploit artifacts of international characters to improve the strength of their PINs may experience false positives in the sense that PINs they intended to be distinct are not actually distinct. This decision was made in order to improve usability across the widest variety of input methods. Users can choose other methods to add strength to PINs.

6.6. FAST Facilities

The secret used to generate the OTP is known only to the client and the KDC, so successful decryption of the encrypted nonce by the KDC authenticates the user. If the OTP value is used in the Reply Key generation, then successful decryption of the encrypted nonce by the client proves that the expected KDC replied. The Reply Key is replaced by either a key generated from the OTP and Armor Key or by the Armor Key. This FAST factor therefore, provides the following facilities: client-authentication, replacing-reply-key, and, depending on the OTP algorithm, KDC-authentication.

7. Acknowledgments

Many significant contributions were made to this document by RSA employees, but special thanks go to Magnus Nystrom, John Linn, Richard Zhang, Piers Bowness, Robert Philpott, Robert Polansky, and Boris Khoutorski.

Many valuable suggestions were also made by members of the Kerberos Working Group, but special thanks go to Larry Zhu, Jeffrey Hutzelman, Tim Alsop, Henry Hotz, Nicolas Williams, Sam Hartman, Frank Cusak, Simon Josefsson, Greg Hudson, and Linus Nordberg.

I would also like to thank Tim Alsop and Srinivas Cheruku of CyberSafe for many valuable review comments.

8. References

8.1. Normative References

[ISOIEC7812] ISO, "ISO/IEC 7812-1:2006 Identification cards -- Identification of issuers -- Part 1: Numbering system", October 2006, <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39698>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[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.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 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.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

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

[RFC6113] Hartman, S. and L. Zhu, "A Generalized Framework for Kerberos Pre-Authentication", RFC 6113, April 2011.

[X.680] ITU-T, "Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.", July 2002.

[X.690] ITU-T, "Recommendation X.690 (2008) | ISO/IEC 8825-1:2008, X.690 : Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", December 2008.

8.2. Informative References

[ASNINY02] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication Protocols", Cryptology ePrint Archive Report 2002/163, November 2002.

[HORENEZ004] Horstein, K., Renard, K., Neuman, C., and G. Zorn, "Integrating Single-use Authentication Mechanisms with Kerberos", Work in Progress, July 2004.

[LHA10] Hornquist Astrand, L., "Kerberos number registry to IANA", Work in Progress, March 2010.

[RFC2289] Haller, N., Metz, C., Nesser, P., and M. Straw, "A One-Time Password System", STD 61, RFC 2289, February 1998.

[RFC2808] Nystrom, M., "The SecurID(r) SASL Mechanism", RFC 2808, April 2000.

[RFC4226] M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., and O. Ranen, "HOTP: An HMAC-Based One-Time Password Algorithm", RFC 4226, December 2005.

[RFC6030] Hoyer, P., Pei, M., and S. Machani, "Portable Symmetric Key Container (PSKC)", RFC 6030, October 2010.

Appendix A. ASN.1 Module

```
   OTPKerberos
   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

   IMPORTS

          KerberosTime, KerberosFlags, EncryptionKey, Int32,
          EncryptedData, LastReq, KerberosString
          FROM KerberosV5Spec2 {iso(1) identified-organization(3)
                                dod(6) internet(1) security(5)
                                kerberosV5(2) modules(4) krb5spec2(2)}
                                -- as defined in RFC 4120.
          AlgorithmIdentifier
          FROM PKIX1Explicit88 { iso (1) identified-organization (3)
                                 dod (6) internet (1)
                                 security (5) mechanisms (5) pkix (7)
                                 id-mod (0) id-pkix1-explicit (18) };
                                 -- As defined in RFC 5280.

          PA-OTP-CHALLENGE ::= SEQUENCE {
            nonce             [0] OCTET STRING,
            otp-service       [1] UTF8String                OPTIONAL,
            otp-tokenInfo     [2] SEQUENCE (SIZE(1..MAX)) OF
                                                  OTP-TOKENINFO,
            salt              [3] KerberosString            OPTIONAL,
            s2kparams         [4] OCTET STRING              OPTIONAL,
            ...
          }

          OTP-TOKENINFO ::= SEQUENCE {
            flags             [0] OTPFlags,
            otp-vendor        [1] UTF8String                OPTIONAL,
            otp-challenge     [2] OCTET STRING (SIZE(1..MAX))
                                                            OPTIONAL,
            otp-length        [3] Int32                     OPTIONAL,
            otp-format        [4] OTPFormat                 OPTIONAL,
            otp-tokenID       [5] OCTET STRING              OPTIONAL,
            otp-algID         [6] AnyURI                    OPTIONAL,
            supportedHashAlg [7] SEQUENCE OF AlgorithmIdentifier
                                                            OPTIONAL,
            iterationCount    [8] Int32                     OPTIONAL,
            ...
          }

          OTPFormat ::= INTEGER {
            decimal(0),
```

```
  hexadecimal(1),
  alphanumeric(2),
  binary(3),
  base64(4)
}

OTPFlags ::= KerberosFlags
-- reserved(0),
-- nextOTP(1),
-- combine(2),
-- collect-pin(3),
-- do-not-collect-pin(4),
-- must-encrypt-nonce (5),
-- separate-pin-required (6),
-- check-digit (7)

PA-OTP-REQUEST ::= SEQUENCE {
  flags          [0]  OTPFlags,
  nonce          [1]  OCTET STRING                OPTIONAL,
  encData        [2]  EncryptedData,
                      -- PA-OTP-ENC-REQUEST or PA-ENC-TS-ENC
                      -- Key usage of KEY_USAGE_OTP_REQUEST
  hashAlg        [3]  AlgorithmIdentifier         OPTIONAL,
  iterationCount [4]  Int32                       OPTIONAL,
  otp-value      [5]  OCTET STRING                OPTIONAL,
  otp-pin        [6]  UTF8String                  OPTIONAL,
  otp-challenge  [7]  OCTET STRING (SIZE(1..MAX)) OPTIONAL,
  otp-time       [8]  KerberosTime                OPTIONAL,
  otp-counter    [9]  OCTET STRING                OPTIONAL,
  otp-format     [10] OTPFormat                   OPTIONAL,
  otp-tokenID    [11] OCTET STRING                OPTIONAL,
  otp-algID      [12] AnyURI                      OPTIONAL,
  otp-vendor     [13] UTF8String                  OPTIONAL,
  ...
}

PA-OTP-ENC-REQUEST ::= SEQUENCE {
  nonce     [0] OCTET STRING,
  ...
}

PA-OTP-PIN-CHANGE ::= SEQUENCE {
  flags     [0] PinFlags,
  pin       [1] UTF8String OPTIONAL,
  minLength [2] INTEGER    OPTIONAL,
  maxLength [3] INTEGER    OPTIONAL,
  last-req  [4] LastReq    OPTIONAL,
  format    [5] OTPFormat  OPTIONAL,
```

```
        ...
    }

    PinFlags ::= KerberosFlags
    -- reserved(0),
    -- systemSetPin(1),
    -- mandatory(2)

    AnyURI ::= UTF8String
       (CONSTRAINED BY {
       -- MUST be a valid URI in accordance with IETF RFC 2396
       })

END
```

Appendix B. Examples of OTP Pre-Authentication Exchanges

This section is non-normative.

B.1. Four-Pass Authentication

In this mode, the client sends an initial AS-REQ to the KDC that does not contain a PA-OTP-REQUEST and the KDC responds with a KRB-ERROR containing a PA-OTP-CHALLENGE.

In this example, the user has been issued with a connected, time-based token, and the KDC requires hashed OTP values in the key generation with SHA-384 as the preferred hash algorithm and a minimum of 1024 iterations. The local policy on the client supports SHA-256 and requires 100,000 iterations of the hash of the OTP value.

The basic sequence of steps involved is as follows:

1. The client obtains the user name of the user.

2. The client sends an initial AS-REQ to the KDC that does not contain a PA-OTP-REQUEST.

3. The KDC determines that the user identified by the AS-REQ requires OTP authentication.

4. The KDC constructs a PA-OTP-CHALLENGE as follows:

   nonce
      A randomly generated value.

   otp-service
      A string that can be used by the client to assist the user in locating the correct token.

   otp-tokenInfo
      Information about how the OTP should be generated from the token.

   flags
      must-encrypt-nonce and collect-pin bits set

   supportedHashAlg
      AlgorithmIdentifiers for SHA-384, SHA-256, and SHA-1

   iterationCount
      1024

5. The KDC returns a KRB-ERROR with an error code of KDC_ERR_PREAUTH_REQUIRED and the PA-OTP-CHALLENGE in the e-data.

6. The client displays the value of otp-service and prompts the user to connect the token.

7. The client collects a PIN from the user.

8. The client obtains the current OTP value from the token using the PIN and records the time as reported by the token.

9. The client generates the Client Key and Reply Key as described in Section 3.6 using SHA-256 from the list of algorithms sent by the KDC, the iteration count of 100,000 as required by local policy, and a randomly generated nonce.

10. The client constructs a PA-OTP-REQUEST as follows:

    flags
       0

    nonce
       The randomly generated value.

    encData
       An EncryptedData containing a PA-OTP-ENC-REQUEST encrypted under the Client Key with a key usage of KEY_USAGE_OTP_REQUEST and the encryption type of the Armor Key. The PA-OTP-ENC-REQUEST contains the nonce from the PA-OTP-CHALLENGE.

    hashAlg
       SHA-256

    iterationCount
       100,000

    otp-time
       The time used in the OTP calculation as reported by the OTP token.

11. The client encrypts the PA-OTP-REQUEST within the enc-fast-req of a PA-FX-FAST-REQUEST.

12. The client sends an AS-REQ to the KDC containing the PA-FX-FAST-REQUEST within the padata.

13. The KDC validates the padata in the PA-OTP-REQUEST by performing the following steps:

    * Generates the Client Key and Reply Key from the OTP value for the user identified in the AS-REQ, using an iteration count of 100,000, a hash algorithm of SHA-256, and the nonce as specified in the PA-OTP-REQUEST.

    * Uses the generated Client Key to decrypt the PA-OTP-ENC-REQUEST in the encData of the PA-OTP-REQUEST.

    * Authenticates the user by comparing the nonce value from the decrypted PA-OTP-ENC-REQUEST to that sent in the corresponding PA-OTP-CHALLENGE.

14. The KDC constructs a TGT for the user.

15. The KDC returns an AS-REP to the client, encrypted using the Reply Key, containing the TGT and padata with the PA-FX-FAST-REPLY.

16. The client authenticates the KDC and verifies the Reply Key change. The client uses the generated Reply Key to decrypt the encrypted data in the AS-REP.

B.2. Two-Pass Authentication

In this mode, the client includes a PA-OTP-REQUEST within a PA-FX-FAST-REQUEST padata of the initial AS-REQ sent to the KDC.

In this example, the user has been issued a hand-held token, so, none of the OTP generation parameters (otp-length, etc.) are included in the PA-OTP-REQUEST. The KDC does not require hashed OTP values in the key generation.

It is assumed that the client has been configured with the following information or has obtained it from a previous PA-OTP-CHALLENGE.

- The OTP value must not be carried in the otp-value.
- The hashed OTP values are not required.

The basic sequence of steps involved is as follows:

1. The client obtains the user name and OTP value from the user.

2. The client generates the Client Key and Reply Key using unhashed OTP values as described in Section 3.6.

3. The client constructs a PA-OTP-REQUEST as follows:

   flags
      0

   encData
      An EncryptedData containing a PA-ENC-TS-ENC encrypted under the Client Key with a key usage of KEY_USAGE_OTP_REQUEST and an encryption type of the Armor Key. The PA-ENC-TS-ENC contains the current client time.

4. The client encrypts the PA-OTP-REQUEST within the enc-fast-req of a PA-FX-FAST-REQUEST.

5. The client sends an AS-REQ to the KDC containing the PA-FX-FAST-REQUEST within the padata.

6. The KDC validates the padata by performing the following steps:

   * Generates the Client Key and Reply Key from the unhashed OTP value for the user identified in the AS-REQ.

   * Uses the generated Client Key to decrypt the PA-ENC-TS-ENC in the encData of the PA-OTP-REQUEST.

   * Authenticates the user using the timestamp in the standard manner.

7. The KDC constructs a TGT for the user.

8. The KDC returns an AS-REP to the client, encrypted using the Reply Key, containing the TGT and padata with the PA-FX-FAST-REPLY.

9. The client authenticates the KDC and verifies the key change. The client uses the generated Reply Key to decrypt the encrypted data in the AS-REP.

B.3. PIN Change

This exchange follows from the point where the KDC receives the PA-OTP-REQUEST from the client in the examples in Appendix B.1 and Appendix B.2. During the validation of the pre-authentication data (whether encrypted nonce or encrypted timestamp), the KDC determines that the user's PIN has expired and so, must be changed. The KDC therefore, includes a PA-OTP-PIN-CHANGE in the AS-REP.

In this example, the KDC does not generate PIN values for the user but requires that the user generate a new PIN that is between 4 and 8 characters in length. The actual PIN change is handled by a PIN change service.

The basic sequence of steps involved is as follows:

1. The client constructs and sends a PA-OTP-REQUEST to the KDC as described in the previous examples.

2. The KDC validates the pre-authentication data and authenticates the user as in the previous examples but determines that the user's PIN has expired.

3. The KDC constructs a PA-OTP-PIN-CHANGE as follows:

   flags
      0
   minLength
      4
   maxLength
      8

4. The KDC encrypts the PA-OTP-PIN-CHANGE within the enc-fast-rep of a PA-FX-FAST-REPLY.

5. The KDC returns a KRB-ERROR to the client of type KDC_ERR_PIN_EXPIRED with padata containing the PA-FX-FAST-REPLY.

6. The client authenticates to the PIN change service and changes the user's PIN.

7. The client sends a second AS-REQ to the KDC containing a PA-OTP-REQUEST constructed using the new PIN.

8. The KDC responds with an AS-REP containing a TGT.

B.4. Resynchronization

This exchange follows from the point where the KDC receives the PA-OTP-REQUEST from the client. During the validation of the pre-authentication data (whether encrypted nonce or encrypted timestamp), the KDC determines that the local record of the token's state needs to be resynchronized with the token. The KDC therefore, includes a KRB-ERROR containing a PA-OTP-CHALLENGE with the "nextOTP" flag set.

The sequence of steps below follows is a variation of the four pass examples shown in Appendix B.1 but the same process would also work in the two-pass case.

1. The client constructs and sends a PA-OTP-REQUEST to the KDC as described in the previous examples.

2. The KDC validates the pre-authentication data and authenticates the user as in the previous examples, but determines that user's token requires resynchronizing.

3. KDC constructs a PA-OTP-CHALLENGE as follows:

   nonce
      A randomly generated value.

   otp-service
      Set to a string that can be used by the client to assist the user in locating the correct token.

   otp-tokenInfo
      Information about how the OTP should be generated from the token.

   flags
      must-encrypt-nonce, collect-pin, and nextOTP bits set

   supportedHashAlg
      AlgorithmIdentifiers for SHA-256 and SHA-1

   iterationCount
      1024

4. KDC returns a KRB-ERROR with an error code of KDC_ERR_PREAUTH_REQUIRED and the PA-OTP-CHALLENGE in the e-data.

5. The client obtains the next OTP value from the token and records the time as reported by the token.

6. The client generates the Client Key and Reply Key as described in Section 3.6 using SHA-256 from the list of algorithms sent by the KDC, the iteration count of 100,000 as required by local policy, and a randomly generated nonce.

7. The client constructs a PA-OTP-REQUEST as follows:

   flags
      nextOTP bit set

   nonce
      The randomly generated value.

   encData
      An EncryptedData containing a PA-OTP-ENC-REQUEST encrypted under the Client Key with a key usage of KEY_USAGE_OTP_REQUEST and the encryption type of the Armor Key. The PA-OTP-ENC-REQUEST contains the nonce from the PA-OTP-CHALLENGE.

   hashAlg
      SHA-256

   iterationCount
      100,000

   otp-time
      The time used in the OTP calculation as reported by the OTP token.

8. The client encrypts the PA-OTP-REQUEST within the enc-fast-req of a PA-FX-FAST-REQUEST.

9. The client sends an AS-REQ to the KDC containing the PA-FX-FAST-REQUEST within the padata.

10. The authentication process now proceeds as with the classic sequence.

Author's Address

Gareth Richards
RSA, The Security Division of EMC
RSA House
Western Road
Bracknell, Berkshire RG12 1RT
UK

EMail: gareth.richards@rsa.com