Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: June 15, 2017

N. McCallum
S. Sorce
R. Harwood
Red Hat, Inc.
G. Hudson
MIT

December 12, 2016

SPAKE Pre-Authentication
draft-mccallum-kitten-krb-spake-preauth-01

Abstract

This document defines a new pre-authentication mechanism for the Kerberos protocol that uses a password authenticated key exchange. This document has three goals. First, increase the security of Kerberos pre-authentication exchanges by making offline brute-force attacks infeasible. Second, enable the use of secure second factor authentication without relying on FAST. This is achived using the existing trust relationship established by the shared first factor. Third, make Kerberos pre-authentication more resilient against time synchronization errors by removing the need to transfer an encrypted timestamp from the client.

Table of Contents

1. Introduction

The most widely deployed Kerberos pre-authentication method, PA-ENC-TIMESTAMP, encrypts a timestamp in the client principal's long-term secret. When a client uses this method, a passive attacker can perform an offline brute-force attack against the transferred ciphertext. When the client principal's long-term key is based on a password, especially a weak password, offline dictionary attacks can successfully recover the key.

1.1. Properties of PAKE

Password authenticated key exchange (PAKE) algorithms provide several properties which are useful to overcome this problem and make them ideal for use as a Kerberos pre-authentication mechanism.

1. Each side of the exchange contributes entropy.

2. Passive attackers cannot determine the shared key.

3. Active attackers cannot perform a man-in-the-middle attack.

4. Either side can store a password or password equivalent.

These properties of PAKE allow us to establish high-entropy encryption keys resistant to offline brute force attack, even when the passwords used are weak (low-entropy).

1.2. Which PAKE?

Diffie-Hellman Encrypted Key Exchange (DH-EKE) is the earliest widely deployed PAKE. It works by encrypting the public keys of a Diffie-Hellman key exchange with a shared secret. However, it requires both that unauthenticated encryption be used and that the public keys be indistinguishable from random data. This last requirement makes it impossible to use this form of PAKE with elliptic curve cryptography. For these reasons, DH-EKE is not a good fit.

Password authenticated key exchange by juggling (JPAKE) permits the use of elliptic curve cryptography. However, it too has drawbacks. First, the additional computation required for the algorithm makes it resource intensive for servers under load. Second, it requires an additional network round-trip, increasing latency and load on the network.

SPAKE is a variant of the technique used by DH-EKE which ensures that all public key encryption and decryption operations result in a member of the underlying group. This property allows SPAKE to be

used with elliptic curve cryptography, permitting the use of markedly smaller key sizes with equivalent security to a finite-field Diffie-Hellman key exchange. Additionally, SPAKE can complete the key exchange in just a single round-trip. These properties make SPAKE an ideal PAKE to use for Kerberos pre-authentication.

1.3. PAKE and Two-Factor Authentication

Using PAKE in a pre-authentication mechanism also has another benefit when coupled with two-factor authentication (2FA). 2FA methods often require the secure transfer of plaintext material to the KDC for verification. This includes one-time passwords, challenge/response signatures and biometric data. Attempting to encrypt this data using the long-term secret results in packets that are vulnerable to offline brute-force attack if either authenticated encryption is used or if the plaintext is distinguishable from random data. This is a problem that PAKE solves for first factor authentication. So a similar technique can be used with PAKE to encrypt second-factor data.

In the OTP pre-authentication [RFC6560] specification, this problem has been mitigated by using FAST, which uses a secondary trust relationship to create a secure encryption channel within which pre-authentication data can be sent. However, the requirement for a secondary trust relationship has proven to be cumbersome to deploy and often introduces third parties into the trust chain (such as certification authorities). These requirements lead to a scenario where FAST cannot be enabled by default without sufficient configuration. SPAKE pre-authentication, instead, can create a secure encryption channel implicitly, using the key exchange to negotiate a high-entropy encryption key. This key can then be used to securely encrypt 2FA plaintext data without the need for a secondary trust relationship. Further, if the second factor verifiers are sent at the same time as the first factor verifier, and the KDC is careful to prevent timing attacks, then an online brute-force attack cannot be used to attack the factors separately.

For these reasons, this draft departs from the advice given in Section 1 of RFC 6113 [RFC6113] which states that "Mechanism designers should design FAST factors, instead of new pre-authentication mechanisms outside of FAST." However, this pre-authentication mechanism does not intend to replace FAST, and may be used with it to further conceal the metadata of the Kerberos messages.

1.4. SPAKE Overview

The SPAKE algorithm can be broadly described in a series of four steps:

1. Calculation and exchange of the public key

2. Calculation of the shared secret (K)

3. Derivation of an encryption key (K')

4. Verification of the derived encryption key (K')

Higher level protocols must define their own verification step. In the case of this mechanism, verification happens implicitly by a successful decryption of the 2FA data.

This mechanism also provides its own method of deriving encryption keys from the calculated shared secret K, for several reasons: to fit within the framework of [RFC3961], to ensure negotiation integrity using a transcript hash, to derive different keys for each use, and to bind the KDC-REQ-BODY to the pre-authentication exchange.

2. Document Conventions

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 refers to numerous terms and protocol messages defined in [RFC4120].

The terms "encryption type", "required checksum mechanism", and "get_mic" are defined in [RFC3961].

The terms "FAST", "PA-FX-COOKIE", "KDC_ERR_PREAUTH_EXPIRED", "KDC_ERR_MORE_PREAUTH_DATA_REQUIRED", "pre-authentication facility", and "authentication set" are defined in [RFC6113].

The [SPAKE] paper defines SPAKE as a family of two key exchange algorithms differing only in derivation of the final key. This mechanism uses a derivation similar to the second algorithm (SPAKE2) with differences in detail. For simplicity, this document refers to the algorithm as "SPAKE". The normative reference for this algorithm is [I-D.irtf-cfrg-spake2].

The terms "ASN.1" and "DER" are defined in [CCITT.X680.2002] and [CCITT.X690.2002].

3. Prerequisites

3.1. PA-ETYPE-INFO2

This mechanism requires the initial KDC pre-authentication state to contain a singular reply key. Therefore, a KDC which offers SPAKE pre-authentication as a stand-alone mechanism MUST supply a PA-ETYPE-INFO2 value containing a single ETYPE-INFO2-ENTRY, as described in [RFC6113] section 2.1.

3.2. Cookie Support

KDCs which implement SPAKE pre-authentication MUST have some secure mechanism for retaining state between AS-REQs. For stateless KDC implementations, this method will most commonly be an encrypted PA-FX-COOKIE. Clients which implement SPAKE pre-authentication MUST support PA-FX-COOKIE.

3.3. More Pre-Authentication Data Required

Both KDCs and clients which implement SPAKE pre-authentication MUST support the use of KDC_ERR_MORE_PREAUTH_DATA_REQUIRED.

4. SPAKE Pre-Authentication Message Protocol

This mechanism uses the reply key and provides the Client Authentication and Strengthening Reply Key pre-authentication facilities. When the mechanism completes successfully, the client will have proved knowledge of the original reply key and possibly a second factor, and the reply key will be strengthened to a more uniform distribution based on the PAKE exchange. This mechanism also ensures the integrity of the KDC-REQ-BODY contents. This mechanism can be used in an authentication set; no pa-hint value is required or defined.

This section will describe the flow of messages when performing SPAKE pre-authentication. We will begin by explaining the most verbose version of the protocol which all implementations MUST support. Then we will describe several optional optimizations to reduce round-trips.

Mechanism messages are communicated using PA-DATA elements within the padata field of KDC-REQ messages or within the METHOD-DATA in the e-data field of KRB-ERROR messages. All PA-DATA elements for this mechanism MUST use the following padata-type:

PA-SPAKE TBD

The padata-value for all PA-SPAKE PA-DATA values MUST be empty or contain a DER encoding for the ASN.1 type PA-SPAKE.

```
PA-SPAKE ::= CHOICE {
    support     [0] SPAKESupport,
    challenge   [1] SPAKEChallenge,
    response    [2] SPAKEResponse,
    encdata     [3] EncryptedData,
    ...
}
```

4.1. First Pass

The SPAKE pre-authentication exchange begins when the client sends an initial authentication service request (AS-REQ) without pre-authentication data. Upon receipt of this AS-REQ, a KDC which requires pre-authentication and supports SPAKE SHOULD reply with a KDC_ERR_PREAUTH_REQUIRED error, with METHOD-DATA containing an empty PA-SPAKE PA-DATA element. This message indicates to the client that the KDC supports SPAKE pre-authentication.

4.2. Second Pass

Once the client knows that the KDC supports SPAKE pre-authentication and the client desires to use it, the client will generate a new AS-REQ message containing a PA-SPAKE PA-DATA element using the support choice. This message indicates to the KDC which groups the client prefers for the SPAKE operation. The group numbers are defined in the IANA "Kerberos SPAKE Groups" registry created by this document. The groups sequence is ordered from the most preferred group to the least preferred group.

```
SPAKESupport ::= SEQUENCE {
    groups      [0] SEQUENCE (SIZE(1..MAX)) OF Int32,
    ...
}
```

The client and KDC initialize a transcript checksum (Section 6) and update it with the DER-encoded PA-SPAKE message.

Upon receipt of the support message, the KDC will select a group. The KDC SHOULD choose a group from the groups provided by the support message. However, if the support message does not contain any group that is supported by the KDC, the KDC MAY select another group in hopes that the client might support it.

Once the KDC has selected a group, the KDC will reply to the client with a KDC_ERR_MORE_PREAUTH_DATA_REQUIRED error containing a PA-SPAKE

PA-DATA element using the challenge choice. The client and KDC update the transcript checksum with the DER-encoded PA-SPAKE message.

```
SPAKEChallenge ::= SEQUENCE {
    group       [0] Int32,
    pubkey      [1] OCTET STRING,
    factors     [2] SEQUENCE (SIZE(1..MAX)) OF SPAKESecondFactor,
    ...
}
```

The group field indicates the KDC-selected group used for all SPAKE calculations as defined in the IANA "Kerberos SPAKE Groups" registry created by this document.

The pubkey field indicates the KDC's public key generated using the M constant in the SPAKE algorithm, with inputs and conversions as specified in Section 5.

The factors field contains an unordered list of second factors which can be used to complete the authentication. Each second factor is represented by a SPAKESecondFactor.

```
SPAKESecondFactor ::= SEQUENCE {
    type        [0] Int32,
    data        [1] OCTET STRING OPTIONAL
}
```

The type field is a unique integer which identifies the second factor type. The factors field of SPAKEChallenge MUST NOT contain more than one SPAKESecondFactor with the same type value.

The data field contains optional challenge data. The contents in this field will depend upon the second factor type chosen.

4.3. Third Pass

Upon receipt of the challenge message, the client will complete its part of of the SPAKE process, resulting in the shared secret K.

Next, the client chooses one of the second factor types listed in the factors field of the challenge message and gathers whatever data is required for this second factor type; possibly using the challenge data for this second factor type. Finally, the client sends an AS-REQ containing a PA-SPAKE PA-DATA element using the response choice.

```
SPAKEResponse ::= SEQUENCE {
    pubkey      [0] OCTET STRING,
    factor      [1] EncryptedData, -- SPAKESecondFactor
    ...
}
```

The client and KDC update the transcript checksum with the pubkey value, and use the resulting checksum for all encryption key derivations.

The pubkey field indicates the client's public key generated using the N constant in the SPAKE algorithm, with inputs and conversions as specified in Section 5.

The factor field indicates the client's chosen second factor data. The key for this field is K'[1] as specified in Section 7. The key usage number for the encryption is KEY_USAGE_SPAKE_FACTOR. The plain text inside the EncryptedData is an encoding of SPAKESecondFactor. Once decoded, the SPAKESecondFactor contains the type of the second factor and any optional data used. The contents of the data field will depend on the second factor type chosen. The client MUST NOT send a response containing a second factor type which was not listed in the factors field of the challenge message.

When the KDC receives the response message from the client, it will use the pubkey to compute the SPAKE result, derive K'[1], and decrypt the factors field. If decryption is successful, the first factor is successfully validated. The KDC then validates the second factor. If either factor fails to validate, the KDC responds with an appropriate KRB-ERROR message.

If validation of the second factor requires further round-trips, the KDC MUST reply to the client with KDC_ERR_MORE_PREAUTH_DATA_REQUIRED containing a PA-SPAKE PA-DATA element using the encdata choice. The key for the EncryptedData value is K'[2] as specified in Section 7, and the key usage number is KEY_USAGE_SPAKE_FACTOR. The plain text of this message contains a DER-encoded SPAKESecondFactor message. As before, the type field of this message will contain the second factor type, and the data field will optionally contain second factor type specific data.

KEY_USAGE_SPAKE_FACTOR	TBD

4.4. Subsequent Passes

Any number of additional round trips may occur using the encdata choice. The contents of the plaintexts are specific to the second factor type. If a client receives a PA-SPAKE PA-DATA element using

the encdata choice from the KDC, it MUST reply with a subsequent AS-REQ with a PA-SPAKE PA-DATA using the encdata choice, or abort the AS exchange.

The key for client-originated encdata messages in subsequent passes is K'[3] as specified in Section 7 for the first subsequent pass, K'[5] for the second, and so on. The key for KDC-originated encdata messages is K'[4] for the first subsequent pass, K'[6] for the second, and so on.

4.5. Reply Key Strengthening

When the KDC has successfully validated both factors, the reply key is strengthened and the mechanism is complete. To strengthen the reply key, the client and KDC replace it with K'[0] as specified in Section 7. The KDC then replies with a KDC-REP message, or continues on to the next mechanism in the authentication set. There is no final PA-SPAKE PA-DATA message from the KDC to the client.

Reply key strengthening occurs only once at the end of the exchange. The client and KDC MUST use the initial reply key as the base key for all K'[n] derivations.

4.6. Optimizations

The full protocol has two possible optimizations.

First, the KDC MAY reply to the initial AS-REQ (containing no pre-authentication data) with a PA-SPAKE PA-DATA element using the challenge choice, instead of an empty padata-value. In this case, the KDC optimistically selects a group which the client may not support. If the group chosen by the challenge message is supported by the client, the client MUST skip to the third pass by issuing an AS-REQ with a PA-SPAKE message using the response choice. If the KDC's chosen group is not supported by the client, the client MUST initialize and update the transcript hash with the KDC's challenge message, and then continue to the second pass. Clients MUST support this optimization.

Second, clients MAY skip the first pass and send an AS-REQ with a PA-SPAKE PA-DATA element using the support choice. KDCs MUST support this optimization.

5. SPAKE Parameters and Conversions

Group elements are converted to octet strings for the SPAKEChallenge and SPAKEResponse pubkey fields and for key derivation using the

serialization method defined in the IANA "Kerberos SPAKE Groups" registry created by this document.

The SPAKE algorithm requires constants M and N for each group. These constants are defined in the IANA "Kerberos SPAKE Groups" registry created by this document.

The SPAKE algorithm requires a shared secret input w to be used as a scalar multiplier (see [I-D.irtf-cfrg-spake2] section 2). This value MUST be produced from the initial reply key as follows:

1. Determine the length of the multiplier octet string defined in the IANA "Kerberos SPAKE Groups" registry created by this document.

2. Produce an octet string of the above length using PRF+(K, "SPAKEsecret"), where K is the initial reply key and PRF+ is defined in [RFC6113] section 5.1.

3. Convert the octet array to a multiplier scalar using the multiplier conversion method defined in the IANA "Kerberos SPAKE Groups" registry created by this document.

The KDC chooses a secret scalar value x and the client chooses a secret scalar value y. As required by the SPAKE algorithm, these values are chosen randomly and uniformly. The KDC and client MUST NOT reuse x or y values for authentications involving different initial reply keys.

6. Transcript Checksum

The transcript checksum is an octet string of length equal to the output length of the required checksum type of the encryption type of the initial reply key. It initially contains all zero values.

When the transcript checksum is updated with an octet string input, the new value is the get_mic result computed over the concatenation of the old value and the input, for the required checksum type of the initial reply key's encryption type, using the initial reply key and the key usage number KEY_USAGE_SPAKE_TRANSCRIPT.

In the normal message flow or with the second optimization described in Section 4.6, the transcript checksum is first updated with the client's support message, then the KDC's challenge message, and finally with the client's pubkey value. It therefore incorporates the client's supported groups, the KDC's chosen group, the KDC's initial second-factor messages, and the client and KDC public values.

If the first optimization described in Section 4.6 is used successfully, the transcript checksum is updated only with the KDC's challenge message and the client's pubkey value.

If first optimization is used unsuccessfully (i.e. the client does not accept the KDC's selected group), the transcript checksum is updated with the KDC's optimistic challenge message, then with the client's support message, then the KDC's second challenge message, and finally with the client's pubkey value.

KEY_USAGE_SPAKE_TRANSCRIPT TBD

7. Key Derivation

Implementations MUST NOT use the SPAKE result (denoted by K in Section 2 of SPAKE [I-D.irtf-cfrg-spake2]) directly for any cryptographic operation. Instead, the SPAKE result is used to derive keys K'[n] as defined in this section. This method differs slightly from the method used to generate K' in Section 3 of SPAKE [I-D.irtf-cfrg-spake2].

A PRF+ input string is assembled by concatenating the following values:

- The fixed string "SPAKEKey".
- The SPAKE result K, converted to an octet string as specified in Section 5.
- The transcript checksum.
- The KDC-REQ-BODY encoding for the request being sent or responded to. Within a FAST channel, the inner KDC-REQ-BODY encoding MUST be used.
- The value n as a big-endian four-byte unsigned binary number.

The derived key K'[n] has the same encryption type as the initial reply key, and has the value random-to-key(PRF+(initial-reply-key, input-string)). PRF+ is defined in [RFC6113] section 5.1.

8. Second Factor Types

This document defines one second factor type:

SF-NONE 1

This second factor type indicates that no second factor is used. Whenever a SPAKESecondFactor is used with SF-NONE, the data field MUST be omitted. The SF-NONE second factor always successfully validates.

9. Security Considerations

All of the security considerations from SPAKE [I-D.irtf-cfrg-spake2] apply here as well.

This mechanism includes unauthenticated plaintext in the support and challenge messages. Beginning with the third pass, the integrity of this plaintext is ensured by incorporating the transcript checksum into the derivation of the final reply key and second factor encryption keys. Downgrade attacks on support and challenge messages will result in the client and KDC deriving different reply keys and EncryptedData keys. The KDC-REQ-BODY contents are also incorporated into key derivation, ensuring their integrity. The unauthenticated plaintext in the KDC-REP message is not protected by this mechanism.

Unless FAST is used, the factors field of a challenge message is not integrity-protected until the response is verified. Second factor types MUST account for this when specifying the semantics of the data field. Second factor data in the challenge should not be included in user prompts, as it could be modified by an attacker to contain misleading or offensive information.

Subsequent factor data, including the data in the response, are encrypted in a derivative of the shared secret K. Therefore, it is not possible to exploit the untrustworthiness of challenge to turn the client into an encryption or signing oracle, unless the attacker knows the client's long-term key.

An implementation of this pre-authentication mechanism can have the property of indistinguishability, meaning that an attacker who guesses a long-term key and a second factor value cannot determine whether one of the factors was correct unless both are correct. Indistinguishability is only maintained if the second factor can be validated solely based on the data in the response; the use of additional round trips will reveal to the attacker whether the long-term key is correct. Indistinguishability also requires that there are no side channels. When processing a response message, whether or not the KDC successfully decrypts the factor field, it must reply with the same error fields, take the same amount of time, and make the same observable communications to other servers.

Given that each EncryptedData will begin a new encryption context, a failure to properly derive the encryption keys will result in a situation where the risk of compromise is non-negligible.

Weak checksums present a risk to the transcript hash. Any etype with a checksum based on one of the following algorithms MUST NOT be used:

- CRC32
- MD4
- MD5

Both the size of the EncryptedData and the number of EncryptedData messages may reveal information about the second factor used in an authentication. Care should be taken to keep second factor messages as small and as few as possible.

A stateless KDC implementation generally must use a PA-FX-COOKIE value to remember its private scalar value x and the transcript checksum. The KDC MUST maintain confidentiality and integrity of the cookie value, perhaps by encrypting it in a key known only to the realm's KDCs. Cookie values may be replayed by attackers. The KDC SHOULD limit the time window of replays using a timestamp, and SHOULD prevent cookie values from being applied to other pre-authentication mechanisms or other client principals. Within the validity period of a cookie, an attacker can replay the final message of a pre-authentication exchange to any of the realm's KDCs and make it appear that the client has authenticated.

Any side channels in the creation of the shared secret input w, or in the multiplications wM and wN, could allow an attacker to recover the client long-term key. Implementations MUST take care to avoid side channels, particularly timing channels. Generation of the secret scalar values x and y need not take constant time, but the amount of time taken MUST NOT provide information about the resulting value.

If an x or y value is reused for pre-authentications involving two different client long-term keys, an attacker who observes both authentications and knows one of the long-term keys can conduct an offline dictionary attack to recover the other one.

This pre-authentication mechanism is not designed to provide forward secrecy. Nevertheless, some measure of forward secrecy may result depending on implementation choices. A passive attacker who determines the client long-term key after the exchange generally will not be able to recover the ticket session key; however, an attacker who also determines the PA-FX-COOKIE encryption key (if the KDC uses

an encrypted cookie) will be able to recover the ticket session key. The KDC can mitigate this risk by periodically rotating the cookie encryption key. If the KDC or client retains the x or y value for reuse with the same client long-term key, an attacker who recovers the x or y value and the long-term key will be able to recover the ticket session key.

Although this pre-authentication mechanism is designed to prevent an offline dictionary attack by an active attacker posing as the KDC, such an attacker can attempt to downgrade the client to encrypted timestamp. Client implementations SHOULD provide a configuration option to disable encrypted timestamp on a per-realm basis to mitigate this attack.

Like any other pre-authentication mechanism using the client long-term key, this pre-authentication mechanism does not prevent online password guessing attacks. The KDC is made aware of unsuccessful guesses, and can apply facilities such as password lockout to mitigate the risk of online attacks.

Elliptic curve group operations are more computationally expensive than secret-key operations. As a result, the use of this mechanism may affect the KDC's performance under normal load and its resistance to denial of service attacks.

The selected group's resistance to offline brute-force attacks may not correspond to the brute-force resistance of the secret key encryption type. For performance reasons, a KDC MAY select a group whose brute-force resistance is weaker than the secret key. A passive attacker who solves the group discrete logarithm problem after the exchange will be able to conduct an offline attack against the client long-term key. Although the use of password policies and costly, salted string-to-key functions may increase the cost of such an attack, the resulting cost will likely not be higher than the cost of solving the group discrete logarithm.

This mechanism does not directly provide the KDC Authentication pre-authentication facility, because it does not send a key confirmation from the KDC to the client. When used as a stand-alone mechanism, the traditional KDC authentication provided by the KDC-REP enc-part still applies.

The conversion of the scalar multiplier for the SPAKE w parameter may produce a multiplier that is larger than the order of the group. Some group implementations may be unable to handle such a multiplier. Others may silently accept such a multiplier, but proceed to perform multiplication that is not constant time. This is a minor risk in all known groups, but is a major risk for P-521 due to the extra

seven high bits in the input octet string. A common solution to this problem is achieved by reducing the multiplier modulo the group order, taking care to ensure constant time operation.

10. Assigned Constants

The following key usage values are assigned for this mechanism:

KEY_USAGE_SPAKE_TRANSCRIPT	TBD
KEY_USAGE_SPAKE_FACTOR	TBD

11. IANA Considerations

This document establishes two registries with the following procedure, in accordance with [RFC5226]:

Registry entries are to be evaluated using the Specification Required method. All specifications must be be published prior to entry inclusion in the registry. There will be a three-week review period by Designated Experts on the krb5-spake-review@ietf.org mailing list. Prior to the end of the review, the Designated Experts must approve or deny the request. This decision is to be conveyed to both the IANA and the list, and should include reasonably detailed explanation in the case of a denial as well as whether the request can be resubmitted.

11.1. Kerberos Second Factor Types

This section species the IANA "Kerberos Second Factor Types" registry. This registry records the number, name, implementation requirements and reference for each second factor protocol.

11.1.1. Registration Template

ID Number: This is a value that uniquely identifies this entry. It is a signed integer in range -2147483648 to 2147483647, inclusive. Positive values must be assigned only for algorithms specified in accordance with these rules for use with Kerberos and related protocols. Negative values should be used for private and experimental algorithms only. Zero is reserved and must not be assigned.

Name: Brief, unique, human-readable name for this algorithm.

Implementation Requirements: The second factor implementation requirements, which must be one of the words Required, Recommended, Optional, Deprecated, or Prohibited.

Reference: URI or otherwise unique identifier for where the details of this algorithm can be found. It should be as specific as reasonably possible.

11.1.2. Initial Registry Contents

- ID Number: 1
- Name: NONE
- Implementation Requirements: Required
- Reference: this draft.

11.2. Kerberos SPAKE Groups

This section specifies the IANA "Kerberos SPAKE Groups" registry. This registry records the number, name, implementation requirements, specification, serialization, multiplier length, multiplier conversion, SPAKE M constant and SPAKE N constant.

11.2.1. Registration Template

ID Number: This is a value that uniquely identifies this entry. It is a signed integer in range -2147483648 to 2147483647, inclusive. Positive values must be assigned only for algorithms specified in accordance with these rules for use with Kerberos and related protocols. Negative values should be used for private and experimental use only. Zero is reserved and must not be assigned. Values should be assigned in increasing order.

Name: Brief, unique, human readable name for this entry.

Implementation Requirements: The group implementation requirements, which must be one of the words Required, Recommended, Optional, Deprecated, or Prohibited.

Specification: Reference to the definition of the group parameters and operations.

Serialization: Reference to the definition of the method used to serialize group elements.

Multiplier Length: The length of the input octet string to multiplication operations.

Multiplier Conversion: Reference to the definition of the method used to convert an octet string to a multiplier scalar.

SPAKE M Constant: The serialized value of the SPAKE M constant in hexadecimal notation.

SPAKE N Constant: The serialized value of the SPAKE N constant in hexadecimal notation.

11.2.2. Initial Registry Contents

- ID Number: 1
- Name: P-256
- Implementation Requirements: Required
- Specification: [SEC2] section 2.4.2
- Serialization: [SEC1] section 2.3.3 (compressed).
- Multiplier Length: 32
- Multiplier Conversion: [SEC1] section 2.3.8.
- SPAKE M Constant:
  02886e2f97ace46e55ba9dd7242579f2993b64e16ef3dcab95afd497333d8fa12f
- SPAKE N Constant:
  03d8bbd6c639c62937b04d997f38c3770719c629d7014d49a24b4f98baa1292b49

- ID Number: 2
- Name: P-384
- Implementation Requirements: Optional
- Specification: [SEC2] section 2.5.1
- Serialization: [SEC1] section 2.3.3 (compressed).
- Multiplier Length: 48
- Multiplier Conversion: [SEC1] section 2.3.8.
- SPAKE M Constant:
  030ff0895ae5ebf6187080a82d82b42e2765e3b2f8749c7e05eba3664
  34b363d3dc36f15314739074d2eb8613fceec2853
- SPAKE N Constant:
  02c72cf2e390853a1c1c4ad816a62fd15824f56078918f43f922ca215
  18f9c543bb252c5490214cf9aa3f0baab4b665c10

- ID Number: 3
- Name: P-521
- Implementation Requirements: Optional
- Specification: [SEC2] section 2.6.1
- Serialization: [SEC1] section 2.3.3 (compressed).
- Multiplier Length: 66
- Multiplier Conversion: [SEC1] section 2.3.8.
- SPAKE M Constant:
  02003f06f38131b2ba2600791e82488e8d20ab889af753a41806c5db1
  8d37d85608cfae06b82e4a72cd744c719193562a653ea1f119eef9356907edc9b5
  6979962d7aa
- SPAKE N Constant:
  0200c7924b9ec017f3094562894336a53c50167ba8c5963876880542b
  c669e494b2532d76c5b53dfb349fdf69154b9e0048c58a42e8ed04cef052a3bc34
  9d95575cd25

12. References

12.1. Normative References

[CCITT.X680.2002]
International Telephone and Telegraph Consultative Committee, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", CCITT Recommendation X.680, July 2002.

[CCITT.X690.2002]
International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.

[I-D.irtf-cfrg-spake2]
Ladd, W., "SPAKE2, a PAKE", draft-irtf-cfrg-spake2-01 (work in progress), February 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, DOI 10.17487/RFC3961, February 2005, <http://www.rfc-editor.org/info/rfc3961>.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <http://www.rfc-editor.org/info/rfc4120>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6113] Hartman, S. and L. Zhu, "A Generalized Framework for Kerberos Pre-Authentication", RFC 6113, DOI 10.17487/RFC6113, April 2011, <http://www.rfc-editor.org/info/rfc6113>.

[SEC1] Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography", May 2009.

[SEC2] Standards for Efficient Cryptography Group, "SEC 2: Recommended Elliptic Curve Domain Parameters", January 2010.

12.2. Non-normative References

[RFC6560] Richards, G., "One-Time Password (OTP) Pre-Authentication", RFC 6560, DOI 10.17487/RFC6560, April 2012, <http://www.rfc-editor.org/info/rfc6560>.

[SPAKE] Abdalla, M. and D. Pointcheval, "Simple Password-Based Encrypted Key Exchange Protocols", February 2005.

Appendix A. ASN.1 Module

```
KerberosV5SPAKE {
        iso(1) identified-organization(3) dod(6) internet(1)
        security(5) kerberosV5(2) modules(4) spake(8)
} DEFINITIONS EXPLICIT TAGS ::= BEGIN

IMPORTS
    EncryptedData, Int32
      FROM KerberosV5Spec2 { iso(1) identified-organization(3)
        dod(6) internet(1) security(5) kerberosV5(2) modules(4)
        krb5spec2(2) };
        -- as defined in RFC 4120.

SPAKESupport ::= SEQUENCE {
    groups      [0] SEQUENCE (SIZE(1..MAX)) OF Int32,
    ...
}

SPAKEChallenge ::= SEQUENCE {
    group       [0] Int32,
    pubkey      [1] OCTET STRING,
    factors     [2] SEQUENCE (SIZE(1..MAX)) OF SPAKESecondFactor,
    ...
}

SPAKESecondFactor ::= SEQUENCE {
    type        [0] Int32,
    data        [1] OCTET STRING OPTIONAL
}

SPAKEResponse ::= SEQUENCE {
    pubkey      [0] OCTET STRING,
    factor      [1] EncryptedData, -- SPAKESecondFactor
    ...
}

PA-SPAKE ::= CHOICE {
    support     [0] SPAKESupport,
    challenge   [1] SPAKEChallenge,
    response    [2] SPAKEResponse,
    encdata     [3] EncryptedData,
    ...
}

END
```

Appendix B. Acknowledgements

Nico Williams (Cryptonector)
Tom Yu (MIT)

Authors' Addresses

Nathaniel McCallum
Red Hat, Inc.

EMail: npmccallum@redhat.com

Simo Sorce
Red Hat, Inc.

EMail: ssorce@redhat.com

Robbie Harwood
Red Hat, Inc.

EMail: rharwood@redhat.com

Greg Hudson
MIT

EMail: ghudson@mit.edu