Stream: Internet Engineering Task Force (IETF)
RFC: 9588
Category: Standards Track
Published: June 2024
ISSN: 2070-1721
Authors: N. McCallum, *Red Hat, Inc.*; S. Sorce, *Red Hat, Inc.*; R. Harwood, *Red Hat, Inc.*; G. Hudson, *MIT*

RFC 9588
Kerberos Simple Password-Authenticated Key Exchange (SPAKE) Pre-authentication

Abstract

This document defines a new pre-authentication mechanism for the Kerberos protocol. The mechanism uses a password-authenticated key exchange (PAKE) to prevent brute-force password attacks, and it may incorporate a second factor.

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 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9588.

Copyright Notice

Table of Contents

1. Introduction

The Kerberos protocol [RFC4120] commonly uses password-derived long-term keys to secure the initial authentication exchange between a Kerberos client and a Key Distribution Center (KDC). As noted in Section 10 of [RFC4120], an attacker can perform an offline dictionary attack against the password; this is performed either by initiating an authentication exchange to a principal for which the KDC does not require pre-authentication or after eavesdropping on a legitimate authentication exchange that uses encrypted timestamp pre-authentication (Section 5.2.7.2 of [RFC4120]).

This document defines a pre-authentication mechanism that authenticates using long-term keys but is resistant to offline dictionary attacks. The mechanism additionally enables the use of second-factor authentication without the need for a separately established secure channel, by leveraging the trust relationship established by the shared long-term key.

1.1. Properties of PAKE

Password-authenticated key exchange (PAKE) algorithms [RFC8125] provide several properties that defend against offline dictionary attacks and make them ideal for use in 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 machine-in-the-middle attack.

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

1.2. PAKE Algorithm Selection

The SPAKE algorithm (defined in [SPAKE]) works by encrypting the public keys of a Diffie-Hellman (DH) key exchange with a shared secret. SPAKE is selected for this pre-authentication mechanism for the following properties:

1. SPAKE's encryption method ensures that the result is a member of the underlying group, so it can be used with elliptic curve cryptography, which is believed to provide equivalent security levels to finite-field DH key exchange at much smaller key sizes.
2. It can compute the shared key after just one message from each party, minimizing the need for additional round trips and state.
3. It requires a small number of group operations; therefore, it can be implemented simply and efficiently.

1.3. PAKE and Two-Factor Authentication

Using PAKE in a pre-authentication mechanism also has another benefit when used as a component of 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. Encrypting this data using the long-term secret results in packets that are vulnerable to offline brute-force attacks on the password, using either an authentication tag or statistical properties of the 2FA credentials to determine whether a password guess is correct.

In "One-Time Password (OTP) Pre-Authentication" [RFC6560], this problem is mitigated using flexible authentication secure tunneling (FAST) (Section 5.4 of [RFC6113]), 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 make it difficult to enable FAST without manual configuration of client hosts. In contrast, SPAKE pre-authentication, 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 document departs from the advice given in Section 1 of [RFC6113], which states: "Mechanism designers should design FAST factors, instead of new pre-authentication mechanisms outside of FAST." However, the SPAKE 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')

In this mechanism, key verification happens implicitly by a successful decryption of the 2FA data or of a placeholder value when no second factor is required. This mechanism uses a tailored 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**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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

The terms "encryption type", "key generation seed length", and "random-to-key" are defined in [RFC3961].

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

[SPAKE] 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). For simplicity, this document refers to the algorithm as "SPAKE".

The terms "Abstract Syntax Notation One (ASN.1)" and "Distinguished Encoding Rules (DER)" are defined in [ITU-T.X680.2021] and [ITU-T.X690.2021], respectively.

When discussing operations within algebraic groups, this document uses additive notation (as described in Section 2.2 of [RFC6090]). Group elements are denoted with uppercase letters, while scalar multiplier values are denoted with lowercase letters.

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 that offers SPAKE pre-authentication as a stand-alone mechanism **MUST** supply a PA-ETYPE-INFO2 value containing a single ETYPE-INFO2-ENTRY, following the guidance in Section 2.1 of [RFC6113]. PA-ETYPE-INFO2 is specified in Section 5.2.7.5 of [RFC4120].

3.2. Cookie Support

KDCs that implement SPAKE pre-authentication **MUST** have some secure mechanism for retaining state between authentication service requests (AS-REQs). For stateless KDC implementations, this method will most commonly be an encrypted PA-FX-COOKIE. Clients that implement SPAKE pre-authentication **MUST** support PA-FX-COOKIE, as described in Section 5.2 of [RFC6113].

3.3. More Pre-authentication Data Required

Both KDCs and clients that implement SPAKE pre-authentication **MUST** support the use of KDC_ERR_MORE_PREAUTH_DATA_REQUIRED, as described in Section 5.2 of [RFC6113].

4. SPAKE Pre-authentication Message Protocol

This mechanism uses the reply key and provides the client authentication and strengthening reply key pre-authentication facilities (Section 3 of [RFC6113]). 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 mechanism negotiates a choice of group for the SPAKE algorithm. Groups are defined in the "Kerberos SPAKE Groups" registry created by this document (see Section 12.2). Each group definition specifies an associated hash function, which will be used for transcript protection and key derivation. Clients and KDCs **MUST** implement the edwards25519 group, but they **MAY** choose not to offer or accept it by default.

The subsections that follow 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 151

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 that requires pre-authentication and supports SPAKE **SHOULD** (unless configuration indicates otherwise) reply with a KDC_ERR_PREAUTH_REQUIRED error, with METHOD-DATA containing an empty PA-SPAKE PA-DATA element (possibly in addition to other PA-DATA elements). 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 wants 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 "Kerberos SPAKE Groups" registry (see Section 12.2). The group's sequence is ordered from the most preferred group to the least preferred group.

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

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. Otherwise, the KDC **MUST** respond with a KDC_ERR_PREAUTH_FAILED error.

The group selection determines the group order, which shall be a large prime p multiplied by a small cofactor h (possibly 1), a generator P of a prime-order subgroup, and two masking points M and N. The KDC selects a random integer x in the range 0 <= x < h*p, which **MUST** be divisible by h. The KDC computes a public key T=x*P+w*M, where w is computed from the initial reply key according to Section 5.

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.

```
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 "Kerberos SPAKE Groups" registry (see Section 12.2).

The pubkey field indicates the KDC's public key T, serialized according to 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 that 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. Note that this challenge is initially transmitted as unauthenticated plaintext; see Section 10.2.

The client and KDC will each initialize a transcript hash (Section 6) using the hash function associated with the chosen group and update it with the concatenation of the DER-encoded PA-SPAKE messages sent by the client and the KDC.

4.3. Third Pass

Upon receipt of the challenge message, the client observes which group was selected by the KDC and deserializes the KDC's public key T. The client selects a random integer y in the range 0 <= x < h*p, which **MUST** be divisible by h. The client computes a public key S=y*P+w*N, where w is computed from the initial reply key according to Section 5. The client computes a shared group element K=y*(T-w*M).

The client will then choose one of the second-factor types listed in the factors field of the challenge message and gather whatever data is required for the chosen second-factor type, possibly using the associated challenge data. Finally, the client will send 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 will update the transcript hash with the pubkey value and use the resulting hash for all encryption key derivations.

The pubkey field indicates the client's public key S, serialized according to Section 5.

The factor field indicates the client's chosen second-factor data. The key for this field is K'[1] (specified in Section 7). The kvno field of the EncryptedData sequence is omitted. The key usage number for the encryption is KEY_USAGE_SPAKE. The plaintext inside the EncryptedData is an encoding of the SPAKESecondFactor. Once decoded, the SPAKESecondFactor provides 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 that was not listed in the factors field of the challenge message.

When the KDC receives the response message from the client, it deserializes the client's public key S, and computes the shared group element K=x*(S-w*N). The KDC derives K'[1] and decrypts 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 **MUST** respond with a KDC_ERR_PREAUTH_FAILED error.

If validation of the second factor requires further round trips, the KDC **MUST** reply to the client with a KDC_ERR_MORE_PREAUTH_DATA_REQUIRED error containing a PA-SPAKE PA-DATA element using the encdata choice. The kvno field of the EncryptedData sequence is omitted. The key for the EncryptedData value is K'[2] (specified in Section 7), and the key usage number is

KEY_USAGE_SPAKE. The plaintext 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 data specific to the second-factor type.

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 element using the encdata choice or abort the AS exchange.

The key for client-originated encdata messages in subsequent passes is K'[3] (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. The strengthening of the reply key is accomplished by the client and KDC replacing 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 that 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. In this case, no SPAKESupport message is sent by the client, so the first update to the transcript hash contains only the KDC's optimistic challenge. If the KDC's chosen group is not supported by the client, the client **MUST** continue to the second pass. In this case, both the client and KDC **MUST** reinitialize the transcript hash for the client's support message. 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. If the KDC accepts the support message and generates a challenge, it **MUST** include a PA-ETYPE-INFO2 value within the METHOD-DATA of the KDC_ERR_MORE_PREAUTH_DATA_REQUIRED error response, as the client may not otherwise be able to compute the initial reply key. If the KDC cannot continue with SPAKE (either because the initial reply key type is incompatible with SPAKE or because it does not support any of the client's groups) but can offer other pre-authentication mechanisms, the KDC **MUST** respond with

a KDC_ERR_PREAUTH_FAILED error containing METHOD-DATA for the available mechanisms. A client supporting this optimization **MUST** continue after a KDC_ERR_PREAUTH_FAILED error as described in Section 2 of [RFC6113]. KDCs **MUST** support this optimization.

5. SPAKE Parameters and Conversions

Group elements are converted to and from octet strings using the serialization method defined in the "Kerberos SPAKE Groups" registry (see Section 12.2).

The SPAKE algorithm requires constants M and N for each group. These constants are defined in the "Kerberos SPAKE Groups" registry (see Section 12.2).

The SPAKE algorithm requires a shared secret input w to be used as a scalar multiplier. This value **MUST** be produced from the initial reply key as follows:

1. Determine the length of the multiplier octet string as defined in the "Kerberos SPAKE Groups" registry (see Section 12.2).
2. Compose a pepper string by concatenating the string "SPAKEsecret" and the group number as a big-endian four-byte two's complement binary number.
3. Produce an octet string of the required length using PRF+(K, pepper), where K is the initial reply key and PRF+ is as defined in Section 5.1 of [RFC6113].
4. Convert the octet string to a multiplier scalar using the multiplier conversion method defined in the "Kerberos SPAKE Groups" registry (see Section 12.2).

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 (see Section 10.4).

6. Transcript Hash

The transcript hash is an octet string of length equal to the output length of the hash function associated with the selected group. All bits are set to zero in the initial value.

When the transcript hash is updated with an octet string input, the new value is the hash function computed over the concatenation of the old value and the input.

In the normal message flow or with the second optimization described in Section 4.6, the transcript hash is:

1. updated with the concatenation of the client's support message and the KDC's challenge, then
2. updated a second time with the client's pubkey value.

Therefore, it 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. Once the transcript hash is finalized, it is used without change for all key derivations (Section 7). In particular, encrypted second-factor messages are not included in the transcript hash.

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

If the first optimization is used unsuccessfully (i.e., the client does not accept the KDC's selected group), the transcript hash is computed as in the normal message flow, without including the KDC's optimistic challenge.

7. Key Derivation

Implementations **MUST NOT** use the shared group element (denoted by K) directly for any cryptographic operation. Instead, the SPAKE result is used to derive keys K'[n] (defined in this section).

First, compute the hash function associated with the selected group over the concatenation of the following values:

- The fixed string "SPAKEkey".
- The group number as a big-endian four-byte two's complement binary number.
- The encryption type of the initial reply key as a big-endian four-byte two's complement binary number.
- The PRF+ output used to compute the initial secret input w (as specified in Section 5).
- The SPAKE result K, converted to an octet string (as specified in Section 5).
- The transcript hash.
- 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, and unsigned binary number.
- A single-byte block counter with the initial value 0x01.

If the hash output is too small for the encryption type's key generation seed length, the block counter value is incremented and the hash function is recomputed to produce as many blocks as are required. The result is truncated to the key generation seed length, and the random-to-key function is used to produce an intermediate key with the same encryption type as the initial reply key.

The key K'[n] has the same encryption type as the initial reply key, and has the value KRB-FX-CF2(initial-reply-key, intermediate-key, "SPAKE", "keyderiv"), where KRB-FX-CF2 is defined in Section 5.1 of [RFC6113].

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. Hint for Authentication Sets

If a KDC offers SPAKE pre-authentication as part of an authentication set (Section 5.3 of [RFC6113]), it **SHOULD** provide a pa-hint value containing the DER encoding of the ASN.1 type PA-SPAKE-HINT. This helps the client determine whether SPAKE pre-authentication is likely to succeed if the authentication set is chosen.

```
PA-SPAKE-HINT ::= SEQUENCE {
    groups      [0] SEQUENCE (SIZE(1..MAX)) OF Int32,
    factors     [1] SEQUENCE (SIZE(1..MAX)) OF SPAKESecondFactor
}
```

The groups field indicates the KDC's supported groups. The factors field indicates the KDC's supported second factors. The KDC **MAY** omit the data field of values in the factors list.

A KDC **MUST NOT** include a PA-SPAKE-HINT message directly in a pa-value field; hints must only be provided within authentication sets. A KDC **SHOULD** include a hint if SPAKE pre-authentication is offered as the second or later element of an authentication set.

The PA-SPAKE-HINT message is not part of the transcript, and it does not replace any part of the SPAKE message flow.

10. Security Considerations

10.1. SPAKE Computations

The deserialized public keys S and T **MUST** be verified to be elements of the group to prevent invalid curve attacks. It is not necessary to verify that they are members of the prime-order subgroup; the computation of K by both parties involves a multiplication by the cofactor h.

The aforementioned cofactor multiplication is accomplished by choosing private scalars x and y, which are divisible by the cofactor. If the client or KDC chooses a scalar that might not be divisible by the cofactor, an attacker might be able to coerce values of K that are not members of the prime-order subgroup and deduce a limited amount of information about w from the order of K.

The scalars x and y **MUST** be chosen uniformly. They **MUST NOT** be reused for different initial reply keys. If an x or y value is reused for pre-authentications involving two different initial reply keys, an attacker who observes both authentications and knows one of the initial reply keys can conduct an offline dictionary attack to recover the other one.

The M and N values for a group **MUST NOT** have known discrete logs. An attacker who knows the discrete log of M or N can perform an offline dictionary attack on passwords. Therefore, it is important to demonstrate that the M and N values for each group were computed without multiplying a known value by the generator P.

10.2. Unauthenticated Plaintext

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 hash 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. In particular, second-factor data in the challenge should not be included in user prompts: it could be modified by an attacker to contain misleading or offensive information.

Unless FAST is used, the factors field of a challenge message is visible to an attacker, who can use it to determine whether a second factor is required for the client.

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 the challenge to turn the client into an encryption or signing oracle for the second-factor credentials, unless the attacker knows the client's long-term key.

Unless FAST is used, any PA-SPAKE-HINT messages are unauthenticated and are not protected by the transcript hash if they are included when SPAKE is advertised in authentication sets. Since hints do not replace any part of the message flow, manipulation of hint messages can only affect the client's decision to use or not use an authentication set, which could more easily be accomplished by removing authentication sets entirely.

10.3. Side Channels

An implementation of the SPAKE 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 the KDC processes a response message, whether or not it 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.

Both the size of the EncryptedData and the number of EncryptedData messages used for second-factor data (including the factor field of the SPAKEResponse message and messages using the encdata PA-SPAKE choice) 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.

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.

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 only a minor risk in most commonly used groups, but it is a more serious 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.

10.4. KDC State

A stateless KDC implementation generally must use a PA-FX-COOKIE value to remember its private scalar value x and the transcript hash. 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, perhaps by splicing them into different SPAKE exchanges. The KDC **SHOULD** limit the time window of replays using a timestamp, and it **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.

The SPAKE 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. 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.

10.5. Dictionary Attacks

Although the SPAKE 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 the encrypted timestamp pre-authentication mechanism. Client implementations **SHOULD** provide a configuration option to enable or disable the encrypted timestamp mechanism on a per-realm basis to mitigate this attack.

If the user enters the wrong password, the client might fall back to the encrypted timestamp mechanism after receiving a KDC_ERR_PREAUTH_FAILED error from the KDC, if the encrypted timestamp mechanism is offered by the KDC and not disabled by client configuration. This fallback will enable a passive attacker to mount an offline dictionary attack against the incorrect password, which may be similar to the correct password. Client implementations **SHOULD** assume that the encrypted timestamp and encrypted challenge mechanisms are unlikely to succeed if SPAKE pre-authentication fails in the second pass and SF-NONE was used.

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

10.6. Brute-Force Attacks

The selected group's resistance to offline brute-force attacks may not correspond to the size of the reply key. For performance reasons, a KDC **MAY** select a group whose brute-force work factor is less than the reply key length. 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.

10.7. Denial-of-Service 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.

10.8. Reflection Attacks

The encdata choice of PA-SPAKE can be used in either direction; the factor-specific plaintext does not necessarily indicate a direction. However, each encdata message is encrypted using a derived key K'[n], with client-originated messages using only odd values of n and KDC-originated messages using only even values. Therefore, an attempted reflection attack would result in a failed decryption.

10.9. Reply Key Encryption Type

This mechanism does not upgrade the encryption type of the initial reply key and relies on that encryption type for confidentiality, integrity, and pseudorandom functions. If the client long-term key uses a weak encryption type, an attacker might be able to subvert the exchange, and the replaced reply key will also be of the same weak encryption type.

10.10. KDC Authentication

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 preexisting KDC authentication provided by the KDC-REP enc-part still applies.

11. Assigned Constants

The following key usage values are assigned for this mechanism:

KEY_USAGE_SPAKE 65

12. IANA Considerations

IANA has assigned the following number for PA-SPAKE in the "Pre-authentication and Typed Data" registry:

Type	Value	Reference
PA-SPAKE	151	RFC 9588

Table 1

This document establishes two registries (see Sections 12.1 and 12.2) with the following procedure, in accordance with [RFC8126]:

Registry entries are to be evaluated using the Specification Required method. All specifications must be published prior to entry inclusion in the registry. Once published, they can be submitted directly to the krb5-spake-review@ietf.org mailing list, where there will be a three-week-long review period by designated experts.

The designated experts ensure that the specification is publicly available. They may provide additional in-depth reviews, but their approval should not be taken as endorsement of the specification.

Prior to the end of the review period, the designated experts must approve or deny the request. This decision is conveyed to both IANA and the submitter. Since the mailing list archives are not public, it should include both a reasonably detailed explanation in the case of a denial as well as whether the request can be resubmitted.

IANA must only accept registry updates from the designated experts and should direct all requests for registration to the review mailing list.

12.1. Kerberos Second-Factor Types

This section specifies the "Kerberos Second-Factor Types" registry, which records the number, name, and reference for each second-factor protocol.

12.1.1. Registration Template

ID Number: A value that uniquely identifies this entry. It is a signed integer in the 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. Values should be assigned in increasing order.

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

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

12.1.2. Initial Registry Contents

ID Number: 0
Name: Reserved
Reference: RFC 9588

ID Number: 1
Name: SF-NONE
Reference: RFC 9588

12.2. Kerberos SPAKE Groups

This section specifies the "Kerberos SPAKE Groups" registry, which records the number, name, specification, serialization, multiplier length, multiplier conversion, SPAKE M and N constants, and associated hash function for each SPAKE Group.

12.2.1. Registration Template

ID Number: A value that uniquely identifies this entry. It is a signed integer in the 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: A brief, unique, human-readable name for this entry.

Specification: A reference to the definition of the group parameters and operations.

Serialization: A reference to the definition of the method used to serialize and deserialize group elements.

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

Multiplier Conversion: A 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.

Hash Function: The group's associated hash function.

12.2.2. Initial Registry Contents

12.2.2.1. Edwards 25519

ID Number: 1
Name: edwards25519
Specification: Section 4.1 of [RFC7748] (edwards25519)
Serialization: Section 3.1 of [RFC8032]
Multiplier Length: 32
Multiplier Conversion: Section 3.1 of [RFC8032]
SPAKE M Constant: d048032c6ea0b6d697ddc2e86bda85a33adac920f1bf18e1b0c6d166a5cecdaf
SPAKE N Constant: d3bfb518f44f3430f29d0c92af503865a1ed3281dc69b35dd868ba85f886c4ab
Hash function: SHA-256 [RFC6234]

12.2.2.2. P-256

ID Number: 2
Name: P-256
Specification: Section 2.4.2 of [SEC2]
Serialization: Section 2.3.3 of [SEC1] (compressed format)
Multiplier Length: 32
Multiplier Conversion: Section 2.3.8 of [SEC1]
SPAKE M Constant: 02886e2f97ace46e55ba9dd7242579f2993b64e16ef3dcab95afd497333d8fa12f
SPAKE N Constant:
03d8bbd6c639c62937b04d997f38c3770719c629d7014d49a24b4f98baa1292b49
Hash function: SHA-256 [RFC6234]

12.2.2.3. P-384

ID Number: 3
Name: P-384
Specification: Section 2.5.1 of [SEC2]
Serialization: Section 2.3.3 of [SEC1] (compressed format)
Multiplier Length: 48
Multiplier Conversion: Section 2.3.8 of [SEC1]
SPAKE M Constant:
030ff0895ae5ebf6187080a82d82b42e2765e3b2f8749c7e05eba366434b363d3dc36f15314739074
d2eb8613fceec2853
SPAKE N Constant:
02c72cf2e390853a1c1c4ad816a62fd15824f56078918f43f922ca21518f9c543bb252c5490214cf9aa
3f0baab4b665c10
Hash function: SHA-384 [RFC6234]

12.2.2.4. P-521

ID Number: 4
Name: P-521
Specification: Section 2.6.1 of [SEC2]
Serialization: Section 2.3.3 of [SEC1] (compressed format)
Multiplier Length: 48
Multiplier Conversion: Section 2.3.8 of [SEC1]
SPAKE M Constant:
02003f06f38131b2ba2600791e82488e8d20ab889af753a41806c5db18d37d85608cfae06b82e4a72
cd744c719193562a653ea1f119eef9356907edc9b56979962d7aa
SPAKE N Constant:
0200c7924b9ec017f3094562894336a53c50167ba8c5963876880542bc669e494b2532d76c5b53df
b349fdf69154b9e0048c58a42e8ed04cef052a3bc349d95575cd25
Hash function: SHA-512 [RFC6234]

13. References

13.1. Normative References

[ITU-T.X680.2021] ITU-T, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2021, February 2021.

[ITU-T.X690.2021] ITU-T, "Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1:2021, February 2021.

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

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, DOI 10.17487/RFC3961, February 2005, <https://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, <https://www.rfc-editor.org/info/rfc4120>.

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

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[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.

13.2. Informative References

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <https://www.rfc-editor.org/info/rfc6090>.

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

[RFC8125] Schmidt, J., "Requirements for Password-Authenticated Key Agreement (PAKE) Schemes", RFC 8125, DOI 10.17487/RFC8125, April 2017, <https://www.rfc-editor.org/info/rfc8125>.

[SPAKE] Abdalla, M. and D. Pointcheval, "Simple Password-Based Encrypted Key Exchange Protocols", CT-RSA 2005, Lecture Notes in Computer Science, Volume 3376, pages 191-208, Springer, DOI 10.1007/978-3-540-30574-3_14, February 2005, <https://doi.org/10.1007/978-3-540-30574-3_14>.

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,
    ...
}

PA-SPAKE-HINT ::= SEQUENCE {
    groups      [0] SEQUENCE (SIZE(1..MAX)) OF Int32,
    factors     [1] SEQUENCE (SIZE(1..MAX)) OF SPAKESecondFactor
}

END
```

Appendix B. SPAKE M and N Value Selection

The M and N values for the initial contents of the SPAKE group registry were generated using the following Python snippet, which assumes an elliptic curve implementation following the interface of Edwards25519Point.stdbase() and Edwards448Point.stdbase() in Appendix A of [RFC8032]:

```
def iterhash(seed, n):
    h = seed
    for i in range(n):
        h = hashlib.sha256(h).digest()
    return h

def bighash(seed, start, sz):
    n = -(-sz // 32)
    hashes = [iterhash(seed, i) for i in range(start, start + n)]
    return b''.join(hashes)[:sz]

def canon_pointstr(ecname, s):
    if ecname == 'edwards25519':
        return s
    elif ecname == 'edwards448':
        return s[:-1] + bytes([s[-1] & 0x80])
    else:
        return bytes([(s[0] & 1) | 2]) + s[1:]

def gen_point(seed, ecname, ec):
    for i in range(1, 1000):
        hval = bighash(seed, i, len(ec.encode()))
        pointstr = canon_pointstr(ecname, hval)
        try:
            p = ec.decode(pointstr)
            if p != ec.zero_elem() and p * p.l() == ec.zero_elem():
                return pointstr, i
        except Exception:
            pass
```

The initial seed strings are as follows:

- For group 1 M: edwards25519 point generation seed (M)
- For group 1 N: edwards25519 point generation seed (N)
- For group 2 M: 1.2.840.10045.3.1.7 point generation seed (M)
- For group 2 N: 1.2.840.10045.3.1.7 point generation seed (N)
- For group 3 M: 1.3.132.0.34 point generation seed (M)
- For group 3 N: 1.3.132.0.34 point generation seed (N)
- For group 4 M: 1.3.132.0.35 point generation seed (M)
- For group 4 N: 1.3.132.0.35 point generation seed (N)

Appendix C. Test Vectors

For the following text vectors:

- The key is the string-to-key of "password" with the salt "ATHENA.MIT.EDUraeburn" for the designated initial reply key encryption type.
- x and y were chosen randomly within the order of the designated group, then multiplied by the cofactor.

- The SPAKESupport message contains only the designated group's number.
- The SPAKEChallenge message offers only the SF-NONE second-factor type.
- The KDC-REQ-BODY message does not contain KDC options, but does contain the client principal name "raeburn@ATHENA.MIT.EDU", the server principal name "krbtgt/ATHENA.MIT.EDU", the realm "ATHENA.MIT.EDU", the till field "19700101000000Z", the nonce zero, and an etype list containing only the designated encryption type.

```
des3-cbc-sha1 edwards25519
key: 850bb51358548cd05e86768c313e3bfef7511937dcf72c3e
w (PRF+ output): 686d84730cb8679ae95416c6567c6a63
                 f2c9cef124f7a3371ae81e11cad42a37
w (reduced multiplier): a1f1a25cbd8e3092667e2fddba8ecd24
                        f2c9cef124f7a3371ae81e11cad42a07
x: 201012d07bfd48ddfa33c4aac4fb1e229fb0d043cfe65ebfb14399091c71a723
y: 500b294797b8b042aca1bedc0f5931a4f52c537b3608b2d05cc8a2372f439f25
X: ec274df1920dc0f690c8741b7941272337454441610l6ef950ad75c51db58c3e
Y: d90974f1c42dac1cd4454561ac2d49af762f2ac87bf02436d461e7b661b43028
T: 18f511e750c97b592acd30db7d9e5fca660389102e6bf610c1bfbed4616c8362
S: 5d10705e0d1e43d5dbf30240ccfbde4a0230c70d4c79147ab0b317edad2f8ae7
K: 25bde0d875f0feb5755f45ba5e857889d916ecf7476f116aa31dc3e037ec4292
SPAKESupport: a0093007a0053003020101
SPAKEChallenge: a1363034a003020101a122042018f511e750c97b592acd30
                db7d9e5fca660389102e6bf610c1bfbed4616c8362a20930
                073005a003020101
Transcript hash after challenge: 22bb2271e34d329d52073c70b1d11879
                                 73181f0bc7614266bb79ee80d3335175
Final transcript hash after pubkey: eaaa08807d0616026ff51c849efbf35b
                                    a0ce3c5300e7d486da46351b13d4605b
KDC-REQ-BODY: 3075a00703050000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020110
K'[0]: baf12fae7cd958cbf1a29bfbc71f89ce49e03e295d89dafd
K'[1]: 64f73dd9c41908206bcec1f719026b574f9d13463d7a2520
K'[2]: 0454520b086b152c455829e6baeff78a61dfe9e3d04a895d
K'[3]: 4a92260b25e3ef94c125d5c24c3e5bced5b37976e67f25c4

rc4-hmac edwards25519
key: 8846f7eaee8fb117ad06bdd830b7586c
w (PRF+ output): 7c86659d29cf2b2ea93bfe79c3cefb88
                 50e82215b3ea6fcd896561d48048f49c
w (reduced multiplier): 2713c1583c53861520b849bfef0525cd
                        4fe82215b3ea6fcd896561d48048f40c
x: c8a62e7b626f44cad807b2d695450697e020d230a738c5cd5691cc781dce8754
y: 18fe7c1512708c7fd06db270361f04593775bc634ceaf45347e5c11c38aae017
X: b0bcbbdd25aa031f4608d0442dd4924be7731d49c089a8301859d77343ffb567
Y: 7d1ab8aeda1a2b1f9eab8d11c0fda60b616005d0f37d1224c5f12b8649f579a5
T: 7db465f1c08c64983a19f560bce966fe5306c4b447f70a5bca14612a92da1d63
S: 38f8d4568090148ebc9fd17c241b4cc2769505a7ca6f3f7104417b72b5b5cf54
K: 03e75edd2cd7e7677642dd68736e91700953ac55dc650e3c2a1b3b4acdb800f8
SPAKESupport: a0093007a0053003020101
SPAKEChallenge: a1363034a003020101a12204207db465f1c08c64983a19f5
                60bce966fe5306c4b447f70a5bca14612a92da1d63a20930
                073005a003020101
```

```
Transcript hash after challenge: 3cde9ed9b562a09d816885b6c225f733
                                 6d9e2674bb4df903dfc894d963a2af42
Final transcript hash after pubkey: f4b208458017de6ef7f6a307d47d87db
                                    6c2af1d291b726860f68bc08bfef440a
KDC-REQ-BODY: 3075a00703050000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020117
K'[0]: 770b720c82384cbb693e85411eedecba
K'[1]: 621deec88e2865837c4d3462bb50a1d5
K'[2]: 1cc8f6333b9fa3b42662fd9914fbd5bb
K'[3]: edb4032b7fc3806d5211a534dcbc390c

aes128-cts-hmac-sha1-96 edwards25519
key: fca822951813fb252154c883f5ee1cf4
w (PRF+ output): 0d591b197b667e083c2f5f98ac891d3c
                 9f99e710e464e62f1fb7c9b67936f3eb
w (reduced multiplier): 17c2a9030afb7c37839bd4ae7fdfeb17
                        9e99e710e464e62f1fb7c9b67936f30b
x: 50be049a5a570fa1459fb9f666e6fd80602e4e87790a0e567f12438a2c96c138
y: b877afe8612b406d96be85bd9f19d423e95be96c0e1e0b5824127195c3ed5917
X: e73a443c678913eb4a0cad5cbd3086cf82f65a5a91b611e01e949f5c52efd6dd
Y: 473c5b44ed2be9cb50afe1762b535b3930530489816ea6bd962622cccf39f6e8
T: 9e9311d985c1355e022d7c3c694ad8d6f7ad6d647b68a90b0fe46992818002da
S: fbe08f7f96cd5d4139e7c9eccb95e79b8ace41e270a60198c007df18525b628e
K: c2f7f99997c585e6b686ceb62db42f17cc70932def3bb4cf009e36f22ea5473d
SPAKESupport: a0093007a0053003020101
SPAKEChallenge: a1363034a003020101a12204209e9311d985c1355e022d7c
                3c694ad8d6f7ad6d647b68a90b0fe46992818002daa20930
                073005a003020101
Transcript hash after challenge: 4512310282c01b39dd9aebd0cc2a5e53
                                 2ed077a6c11d4c973c4593d525078797
Final transcript hash after pubkey: 951285f107c87f0169b9c918a1f51f60
                                    cb1a75b9f8bb799a99f53d03add94b5f
KDC-REQ-BODY: 3075a00703050000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020111
K'[0]: 548022d58a7c47eae8c49dccf6baa407
K'[1]: b2c9ba0e13fc8ab3a9d96b51b601cf4a
K'[2]: 69f0ee5fdb6c237e7fcd38d9f87df1bd
K'[3]: 78f91e2240b5ee528a5cc8d7cbebfba5

aes256-cts-hmac-sha1-96 edwards25519
key: 01b897121d933ab44b47eb5494db15e50eb74530dbdae9b634d65020ff5d88c1
w (PRF+ output): e902341590a1b4bb4d606a1c643cccb3
                 f2108f1b6aa97b381012b9400c9e3f4e
w (reduced multiplier): 35b35ca126156b5bf4ec8b90e9545060
                        f2108f1b6aa97b381012b9400c9e3f0e
x: 88c6c0a4f0241ef217c9788f02c32d00b72e4310748cd8fb5f94717607e6417d
y: 88b859df58ef5c69bacdfe681c582754eaab09a74dc29cff50b328613c232f55
X: 23c48eaff2721051946313840723b38f563c59b92043d6ffd752f95781af0327
Y: 3d51486ec1d9be69bc45386bb675c013db87fd0488f6a9cacf6b43e8c81a0641
T: 6f301aacae1220e91be42868c163c5009aeea1e9d9e28afcfc339cda5e7105b5
S: 9e2cc32908fc46273279ec75354b4aeafa70c3d99a4d507175ed70d80b255dda
K: cf57f58f6e60169d2ecc8f20bb923a8e4c16e5bc95b9e64b5dc870da7026321b
```

```
SPAKESupport: a0093007a0053003020101
SPAKEChallenge: a1363034a003020101a12204206f301aacae1220e91be428
                68c163c5009aeea1e9d9e28afcfc339cda5e7105b5a20930
                073005a003020101
Transcript hash after challenge: 23a5e72eb4dedd1ca860f43736c458f0
                                 775c3bb1370a26af8a9374d521d70ec9
Final transcript hash after pubkey: 1c605649d4658b58cbe79a5faf227acc
                                    16c355c58b7dade022f90c158fe5ed8e
KDC-REQ-BODY: 3075a0070305000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020112
K'[0]: a9bfa71c95c575756f922871524b6528
       8b3f695573ccc0633e87449568210c23
K'[1]: 1865a9ee1ef0640ec28ac007391cac62
       4c42639c714767a974e99aa10003015f
K'[2]: e57781513fefdb978e374e156b0da0c1
       a08148f5eb26b8e157ac3c077e28bf49
K'[3]: 008e6487293c3cc9fabbbcdd8b392d6d
       cb88222317fd7fe52d12fbc44fa047f1

aes256-cts-hmac-sha1-96 P-256
key: 01b897121d933ab44b47eb5494db15e50eb74530dbdae9b634d65020ff5d88c1
w (PRF+ output): eb2984af18703f94dd5288b8596cd369
                 88d0d4e83bfb2b44de14d0e95e2090bd
w (reduced multiplier): eb2984af18703f94dd5288b8596cd369
                        88d0d4e83bfb2b44de14d0e95e2090bd
x: 935ddd725129fb7c6288e1a5cc45782198a6416d1775336d71eacd0549a3e80e
y: e07405eb215663abc1f254b8adc0da7a16febaa011af923d79fdef7c42930b33
X: 03bc802165aea7dbd98cc155056249fe0a37a9c203a7c0f7e872d5bf687bd105e2
Y: 0340b8d91ce3852d0a12ae1f3e82c791fc86df6b346006431e968a1b869af7c735
T: 024f62078ceb53840d02612195494d0d0d88de21feeb81187c71cbf3d01e71788d
S: 021d07dc31266fc7cfd904ce2632111a169b7ec730e5f74a7e79700f86638e13c8
K: 0268489d7a9983f2fde69c6e6a1307e9d252259264f5f2dfc32f58cca19671e79b
SPAKESupport: a0093007a0053003020102
SPAKEChallenge: a1373035a003020102a1230421024f62078ceb53840d0261
                2195494d0d0d88de21feeb81187c71cbf3d01e71788da209
                30073005a003020101
Transcript hash after challenge: 0a142afca77c2e92b066572a90389eac
                                 40a6b1f1ed8b534d342591c0e7727e00
Final transcript hash after pubkey: 20ad3c1a9a90fc037d1963a1c4bfb15a
                                    b4484d7b6cf07b12d24984f14652de60
KDC-REQ-BODY: 3075a0070305000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020112
K'[0]: 7d3b906f7be49932db22cd3463f032d0
       6c9c078be4b1d076d201fc6e61ef531e
K'[1]: 17d74e36f8993841fbb7feb12fa4f011
       243d3ae4d2ace55b39379294bbc4db2c
K'[2]: d192c9044081a2aa6a97a6c69e2724e8
       e5671c2c9ce073dd439cdbaf96d7dab0
K'[3]: 41e5bad6b67f12c53ce0e2720dd6a988
       7f877bf9463c2d5209c74c36f8d776b7

aes256-cts-hmac-sha1-96 P-384
```

```
key: 01b897121d933ab44b47eb5494db15e50eb74530dbdae9b634d65020ff5d88c1
w (PRF+ output): 0304cfc55151c6bbe889653db96dbfe0ba4acafc024c1e88
                 40cb3a486f6d80c16e1b8974016aa4b7fa43042a9b3825b1
w (reduced multiplier): 0304cfc55151c6bbe889653db96dbfe0
                        ba4acafc024c1e8840cb3a486f6d80c1
                        6e1b8974016aa4b7fa43042a9b3825b1
x: f323ca74d344749096fd35d0adf20806e521460637176e84d977e9933c49d76f
   cfc6e62585940927468ff53d864a7a50
y: 5b7c709acb175a5afb82860deabca8d0b341facdff0ac0f1a425799aa905d750
   7e1ea9c573581a81467437419466e472
X: 0211e3334f117b76635dd802d4022f601680a1fd066a56606b7f246493a10351
   7797b81789b225bd5bb1d9ae1da2962250
Y: 0383dfa413496e5e7599fc8c6430f8d6910d37cf326d81421bc92c0939b555c4
   ca2ef6a993f6d3db8cb7407655ef60866e
T: 02a1524603ef14f184696f854229d3397507a66c63f841ba748451056be07879
   ac298912387b1c5cdff6381c264701be57
S: 020d5adfdb92bc377041cf5837412574c5d13e0f4739208a4f0c859a0a302bc6
   a533440a245b9d97a0d34af5016a20053d
K: 0264aa8c61da9600dfb0beb5e46550d63740e4ef29e73f1a30d543eb43c25499
   037ad16538586552761b093cf0e37c703a
SPAKESupport: a0093007a0053003020103
SPAKEChallenge: a1473045a003020103a133043102a1524603ef14f184696f
                854229d3397507a66c63f841ba748451056be07879ac2989
                12387b1c5cdff6381c264701be57a20930073005a0030201
                01
Transcript hash after challenge: 4d4095d9f94552e15015881a3f2cf458
                                 1be83217cf7ad830d2f051dba3ec8caa
                                 6e354eaa85738d7035317ac557f8c294
Final transcript hash after pubkey: 5ac0d99ef9e5a73998797fe64f074673
                                    e3952dec4c7d1aacce8b75f64d2b0276
                                    a901cb8539b4e8ed69e4db0ce805b47b
KDC-REQ-BODY: 3075a00703050000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020112
K'[0]: b917d37c16dd1d8567fbe379f64e1ee3
       6ca3fd127aa4e60f97e4afa3d9e56d91
K'[1]: 93d40079dab229b9c79366829f4e7e72
       82e6a4b943ac7bac69922d516673f49a
K'[2]: bfc4f16f12f683e71589f9a888e23287
       5ef293ac9793db6c919567cd7b94bcd4
K'[3]: 3630e2b5b99938e7506733141e8ec344
       166f6407e5fc2ef107c156e764d1bc20

aes256-cts-hmac-sha1-96 P-521
key: 01b897121d933ab44b47eb5494db15e50eb74530dbdae9b634d65020ff5d88c1
w (PRF+ output): de3a095a2b2386eff3eb15b735398da1caf95bc8425665d8
                 2370aff58b0471f34a57bccddf1ebf0a2965b58a93ee5b45
                 e85d1a5435d1c8c83662999722d542831f9a
w (reduced multiplier): 003a095a2b2386eff3eb15b735398da1
                        caf95bc8425665d82370aff58b0471f3
                        4cce63791cfed967f0c94c16054b3e17
                        03133681bece1e05219f5426bc944b0f
                        bfb3
x: 017c38701a14b490b6081dfc83524562be7fbb42e0b20426465e3e37952d30bc
   ab0ed857010255d44936a1515607964a870c7c879b741d878f9f9cdf5a865306
   f3f5
```

```
y: 003e2e2950656fa231e959acdd984d125e7fa59cec98126cbc8f3888447911eb
   cd49428a1c22d5fdb76a19fbeb1d9edfa3da6cf55b158b53031d05d51433ade9
   b2b4
X: 03003e95272223b210b48cfd908b956a36add04a7ff443511432f94ddd87e064
   1d680ba3b3d532c21fa6046192f6bfae7af81c4b803aa154e12459d1428f8f2f
   56e9f2
Y: 030064916687960df496557ecab08298bf075429eca268c6dabbae24e258d568
   c62841664dc8ecf545369f573ea84548faa22f118128c0a87e1d47315afabb77
   3bb082
T: 02017d3de19a3ec53d0174905665ef37947d142535102cd9809c0dfbd0dfe007
   353d54cf406ce2a59950f2bb540df6fbe75f8bbbef811c9ba06cc275adbd9675
   6696ec
S: 02004d142d87477841f6ba053c8f651f3395ad264b7405ca5911fb9a55abd454
   fef658a5f9ed97d1efac68764e9092fa15b9e0050880d78e95fd03abf5931791
   6822b5
K: 03007c303f62f09282cc849490805bd4457a6793a832cbeb55df427db6a31e99
   b055d5dc99756d24d47b70ad8b6015b0fb8742a718462ed423b90fa3fe631ac1
   3fa916
SPAKESupport: a0093007a0053003020104
SPAKEChallenge: a1593057a003020104a145044302017d3de19a3ec53d0174
                905665ef37947d142535102cd9809c0dfbd0dfe007353d54
                cf406ce2a59950f2bb540df6fbe75f8bbbef811c9ba06cc2
                75adbd96756696eca20930073005a003020101
Transcript hash after challenge: 554405860f8a80944228f1fa2466411d
                                 cf236162aa385e1289131b39e1fd59f2
                                 5e58b4c281ff059c28dc20f5803b87c6
                                 7571ce64cea01b39a21819d1ef1cdc7f
Final transcript hash after pubkey: 8d6a89ae4d80cc4e47b6f4e48ea3e579
                                    19cc69598d0d3dc7c8bd49b6f1db1409
                                    ca0312944cd964e213aba98537041102
                                    237cff5b331e5347a0673869b412302e
KDC-REQ-BODY: 3075a00703050000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020112
K'[0]: 1eb3d10bee8fab483adcd3eb38f3ebf1
       f4feb8db96ecc035f563cf2e1115d276
K'[1]: 482b92781ce57f49176e4c94153cc622
       fe247a7dbe931d1478315f856f085890
K'[2]: a2c215126dd3df280aab5a27e1e0fb7e
       594192cbff8d6d8e1b6f1818d9bb8fac
K'[3]: cc06603de984324013a01f888de6d43b
       410a4da2dea53509f30e433c352fb668

aes256-cts-hmac-sha1-96 edwards25519, accepted optimistic challenge
key: 01b897121d933ab44b47eb5494db15e50eb74530dbdae9b634d65020ff5d88c1
w (PRF+ output): e902341590a1b4bb4d606a1c643cccb3
                 f2108f1b6aa97b381012b9400c9e3f4e
w (reduced multiplier): 35b35ca126156b5bf4ec8b90e9545060
                        f2108f1b6aa97b381012b9400c9e3f0e
x: 70937207344cafbc53c8a55070e399c584cbafce00b836980dd4e7e74fad2a64
y: 785d6801a2490df028903ac6449b105f2ff0db895b252953cdc2076649526103
X: 13841224ea50438c1d9457159d05f2b7cd9d05daf154888eeed223e79008b47c
Y: d01fc81d5ce20d6ea0939a6bb3e40ccd049f821baaf95e323a3657309ef75d61
T: 83523b35f1565006cbfc4f159885467c2fb9bc6fe23d36cb1da43d199f1a3118
S: 2a8f70f46cee9030700037b77f22cec7970dcc238e3e066d9d726baf183992c6
K: d3c5e4266aa6d1b2873a97ce8af91c7e4d7a7ac456acced7908d34c561ad8fa6
```

```
SPAKEChallenge: a1363034a003020101a122042083523b35f1565006cbfc4f
                159885467c2fb9bc6fe23d36cb1da43d199f1a3118a20930
                073005a003020101
Transcript hash after challenge: 0332da8ba3095ccd127c51740cb905ba
                                 c76e90725e769570b9d8338e6d62a7f2
Final transcript hash after pubkey: 26f07f9f8965307434d11ea855461d41
                                    e0cbabcc0a1bab48ecee0c6c1a4292b7
KDC-REQ-BODY: 3075a00703050000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020112
K'[0]: 4569ec08b5de5c3cc19d941725913ace
       8d74524b521a341dc746acd5c3784d92
K'[1]: 0d96ce1a4ac0f2e280a0cfc31742b064
       61d83d04ae45433db2d80478dd882a4c
K'[2]: 58018c19315a1ba5d5bb9813b58029f0
       aec18a6f9ca59e0847de1c60bc25945c
K'[3]: ed7e9bffd68c54d86fb19cd3c03f317f
       88a71ad9a5e94c28581d93fc4ec72b6a

aes256-cts-hmac-sha1-96 P-521, rejected edwards25519 challenge
key: 01b897121d933ab44b47eb5494db15e50eb74530dbdae9b634d65020ff5d88c1
w (PRF+ output): de3a095a2b2386eff3eb15b735398da1caf95bc8425665d8
                 2370aff58b0471f34a57bccddf1ebf0a2965b58a93ee5b45
                 e85d1a5435d1c8c83662999722d542831f9a
w (reduced multiplier): 003a095a2b2386eff3eb15b735398da1
                        caf95bc8425665d82370aff58b0471f3
                        4cce63791cfed967f0c94c16054b3e17
                        03133681bece1e05219f5426bc944b0f
                        bfb3
x: 01687b59051bf40048d7c31d5a973d792fa12284b7a447e7f5938b5885ca0bb2
   c3f0bd30291a55fea08e143e2e04bdd7d19b753c7c99032f06cab0d9c2aa8f83
   7ef7
y: 01ded675ebf74fe30c9a53710f577e9cf84f09f6048fe245a4600004884cc167
   733f9a9e43108fb83babe8754cd37cbd7025e28bc9ff870f084c7244f536285e
   25b4
X: 03001bed88af987101ef52db5b8876f6287eb49a72163876c2cf99deb94f4c74
   9bfd118f0f400833cc8daad81971fe40498e6075d8ba0a2acfac35eb9ec8530e
   e0edd5
Y: 02007bd3bf214200795ea449852976f241c9f50f445f78ff2714fffe42983f25
   cd9c9094ba3f9d7adadd6c251e9dc0991fc8210547e7769336a0ac406878fb94
   be2f1f
T: 02014cb2e5b592ece5990f0ef30d308c061de1598bc4272b4a6599bed466fd15
   21693642abcf4dbe36ce1a2d13967de45f6c4f8d0fa8e14428bf03fb96ef5f1e
   d3e645
S: 02016c64995e804416f748fd5fa3aa678cbc7cbb596a4f523132dc8af7ce84e5
   41f484a2c74808c6b21dcf7775baefa6753398425becc7b838b210ac5daa0cb0
   b710e2
K: 0200997f4848ae2e7a98c23d14ac662030743ab37fccc2a45f1c721114f40bcc
   80fe6ec6aba49868f8aea1aa994d50e81b86d3e4d3c1130c8695b68907c673d9
   e5886a
Optimistic SPAKEChallenge: a1363034a003020102a122042047ca8c
                           24c3a4a70b6eca228322529dadcfa85c
                           f58faceecf5d5c02907b9e2deba20930
                           073005a003020101
SPAKESupport: a0093007a0053003020104
SPAKEChallenge: a1593057a003020104a145044302014cb2e5b592ece5990f
```

```
                0ef30d308c061de1598bc4272b4a6599bed466fd15216936
                42abcf4dbe36ce1a2d13967de45f6c4f8d0fa8e14428bf03
                fb96ef5f1ed3e645a20930073005a003020101
Transcript hash after challenge: cb925b8baeae5e2867ab5b10ae1c941c
                                 4ff4b58a4812c1f7bd1c862ad480a8e1
                                 c6fcd5e88d846a2045e385841c91a75a
                                 d2035f0ff692717608e2a5a90842eff2
Final transcript hash after pubkey: d0efed5e3e2c39c26034756d92a66fec
                                    3082ad793d0197f3f89ad36026f146a3
                                    996e548aa3fc49e2e82f8cac5d132c50
                                    5aa475b39e7be79cded22c26c41aa777
KDC-REQ-BODY: 3075a00703050000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020112
K'[0]: 631fcc8596e7f40e59045950d72aa0b7
       bac2810a07b767050e983841cf3a2d4c
K'[1]: 881464920117074dbc67155a8f3341d1
       121ef65f78ea0380bfa81a134c1c47b1
K'[2]: 377b72ac3af2caad582d73ae4682fd56
       b531ee56706200dd6c38c42b8219837a
K'[3]: 35ad8e4d580ed3f0d15ad928329773c0
       81bd19f9a56363f3a5f77c7e66108c26
```

There are currently no encryption types with a seed size large enough to require multiple hash blocks during key derivation with any of the assigned hash functions. To exercise this possibility, the following test vector illustrates what keys would be derived if there were a copy of the edwards25519 group with group number -1 and associated hash function SHA-1:

```
AES256 edwards25519 SHA-1 group number -1
key: 01b897121d933ab44b47eb5494db15e50eb74530dbdae9b634d65020ff5d88c1
w (PRF+ output): 26da6b118cee6fa5ea795ed32d61490d
                 82b2f11102312f3f2fc04fb01c93df91
w (reduced multiplier): d166c7cc9e72ca8c61f6a9185a987251
                        81b2f11102312f3f2fc04fb01c93df01
x: 606c1b668008ed78fe2eee942e8f08007f3f1dcbef66d37fd69033525bda2030
y: 10fc4e0bb1a902e58f632a1ea0bceb366360ac985f46996d956a02572bfcf050
X: 389621509665abad35eaab26eab3a0f593c7b4380562aa5513c1140fd78ce048
Y: de3ed05986eeac518958b566f9bad065b321402025cd188f3d198dc55c6d6b8d
T: 2289a4f3c613e6e1df95e94aaa3c127dc062b9fceec3f9b62378dc729d61d0e3
S: f9a7fa352930dedb422d567700bfcd39ba221e7f9ac3e6b36f2b63b68b88642c
K: 6f61d6b18323b6c3ddaac7c56712845335384f095d3e116f69feb926a04f1340
SPAKESupport: a0093007a00530030201ff
SPAKEChallenge: a1363034a0030201ffa12204202289a4f3c613e6e1df95e9
                4aaa3c127dc062b9fceec3f9b62378dc729d61d0e3a20930
                073005a003020101
Transcript hash after challenge: f5c051eb75290f92142c
                                 bbe80557ec2c85902c94
Final transcript hash after pubkey: 9e26a3b148400c8f9cb8
                                    545331e4e7dcab399cc0
KDC-REQ-BODY: 3075a00703050000000000a1143012a003020101a10b3009
              1b077261656275726ea2101b0e415448454e412e4d49542e
              454455a3233021a003020102a11a30181b066b7262746774
              1b0e415448454e412e4d49542e454455a511180f31393730
              303130313030303030305aa703020100a8053003020112
K'[0]: 40bceb51bba474fd29ae65950022b704
       17b80d973fa8d8d6cd39833ff89964ad
K'[1]: c29a7315453dc1cce938fa12a9e5c0db
       2894b2194da14c9cd4f7bc3a6a37223d
K'[2]: f261984dba3c230fad99d324f871514e
       5aad670e44f00daef3264870b0851c25
K'[3]: d24b2b46bab7c4d1790017d9116a7eeb
       ca88b0562a5ad8989c826cb7dab715c7
```

Acknowledgements

Nico Williams (Cryptonector)

Taylor Yu (MIT)

Authors' Addresses

Nathaniel McCallum
Red Hat, Inc.
Email: nathaniel@mccallum.life

Simo Sorce
Red Hat, Inc.
Email: ssorce@redhat.com

Robbie Harwood
Red Hat, Inc.
Email: rharwood@pm.me

Greg Hudson
MIT
Email: ghudson@mit.edu