Stream: Independent Submission
RFC: 9383
Category: Informational
Published: September 2023
ISSN: 2070-1721
Authors: T. Taubert C. A. Wood
Apple Inc.

RFC 9383
SPAKE2+, an Augmented Password-Authenticated Key Exchange (PAKE) Protocol

Abstract

This document describes SPAKE2+, a Password-Authenticated Key Exchange (PAKE) protocol run between two parties for deriving a strong shared key with no risk of disclosing the password. SPAKE2+ is an augmented PAKE protocol, as only one party has knowledge of the password. This method is simple to implement, compatible with any prime-order group, and computationally efficient.

This document was produced outside of the IETF and IRTF and represents the opinions of the authors. Publication of this document as an RFC in the Independent Submissions Stream does not imply endorsement of SPAKE2+ by the IETF or IRTF.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This is a contribution to the RFC Series, independently of any other RFC stream. The RFC Editor has chosen to publish this document at its discretion and makes no statement about its value for implementation or deployment. Documents approved for publication by the RFC Editor are not candidates for any level of Internet Standard; see 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/rfc9383.

Copyright Notice

Table of Contents

1. Introduction

This document describes SPAKE2+, a Password-Authenticated Key Exchange (PAKE) protocol run between two parties for deriving a strong shared key with no risk of disclosing the password. SPAKE2+ is an augmented PAKE protocol, as only one party makes direct use of the password during the execution of the protocol. The other party only needs a record corresponding to the first party's registration at the time of the protocol execution instead of the password. This record can be computed once, during an offline registration phase. The party using the password directly would typically be a client and would act as a Prover, while the other party would be a server and would act as a Verifier.

The protocol is augmented in the sense that it provides some resilience against the compromise or extraction of the registration record. The design of the protocol forces the adversary to recover the password from the record to successfully execute the protocol. Hence, this protocol can be advantageously combined with a salted Password Hashing Function to increase the cost of the recovery and slow down attacks. The record cannot be used directly to successfully run the protocol as a Prover, making this protocol more robust than balanced PAKEs, which don't benefit from Password Hashing Functions to the same extent.

This augmented property is especially valuable in scenarios where the execution of the protocol is constrained and the adversary cannot query the salt of the Password Hashing Function ahead of the attack. For example, a constraint may be when physical proximity through a local network is required or when a first authentication factor is required for initiation of the protocol.

This document has content split out from a related document, [RFC9382], which specifies SPAKE2. SPAKE2 is a symmetric PAKE protocol, where both parties have knowledge of the password. SPAKE2+ is the asymmetric or augmented version of SPAKE2, wherein only one party has knowledge of the password. SPAKE2+ is specified separately in this document because the use cases for symmetric and augmented PAKEs are different and therefore warrant different technical specifications. Neither SPAKE2 nor SPAKE2+ was selected as the result of the Crypto Forum Research Group (CFRG) PAKE selection competition. However, this password-based key exchange protocol appears in [TDH] and is proven secure in [SPAKE2P-Analysis]. It is compatible with any prime-order group and relies only on group operations, making it simple and computationally efficient. Thus, it was felt that publication was beneficial to make the protocol available for wider consideration.

This document was produced outside of the IETF and IRTF and represents the opinions of the authors. Publication of this document as an RFC in the Independent Submissions Stream does not imply endorsement of SPAKE2+ by the IETF or IRTF.

2. Requirements Notation

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.

3. Definition of SPAKE2+

Let G be a group in which the computational Diffie-Hellman (CDH) problem is hard. Suppose G has order p*h where p is a large prime; h will be called the cofactor. Let I be the unit element in G, e.g., the point at infinity if G is an elliptic curve group. We denote the operations in the group additively. We assume that there is a representation of elements of G as byte strings: common choices would be SEC 1 uncompressed or compressed [SEC1] for elliptic curve groups or big-endian integers of a fixed (per-group) length for prime field DH. We fix a generator P of the (large) prime-order subgroup of G. P is specified in the document defining the group, and so we do not repeat it here.

|| denotes concatenation of strings. We also let len(S) denote the length of a string in bytes, represented as an eight-byte little-endian number. Finally, let nil represent an empty string, i.e., len(nil) = 0.

KDF is a key derivation function that takes as input a salt, input keying material (IKM), info string, and derived key length L to derive a cryptographic key of length L. MAC is a Message Authentication Code algorithm that takes a secret key and message as input to produce an output. Let Hash be a hash function from arbitrary strings to bit strings of a fixed length. Common choices for Hash are SHA256 or SHA512 [RFC6234]. Section 4 specifies variants of KDF, MAC, and Hash suitable for use with the protocols contained herein.

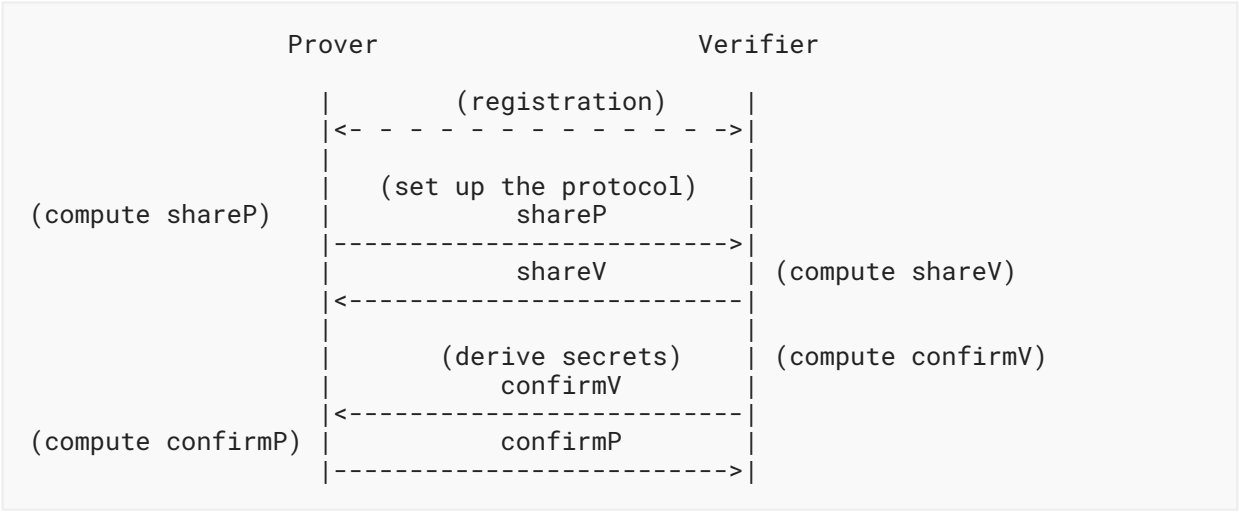
Let there be two parties, a Prover and a Verifier. Their identities, denoted as idProver and idVerifier, may also have digital representations such as Media Access Control addresses or other names (hostnames, usernames, etc.). The parties may share additional data (the context) separate from their identities, which they may want to include in the protocol transcript. One example of additional data is a list of supported protocol versions if SPAKE2+ were used in a higher-level protocol that negotiates the use of a particular PAKE. Another example is the inclusion of the application name. Including these data points would ensure that both parties agree upon the same set of supported protocols and therefore prevents downgrade and cross-protocol attacks. Specification of precise context values is out of scope for this document.

3.1. Protocol Overview

SPAKE2+ is a two-round protocol that establishes a shared secret with an additional round for key confirmation. Prior to invocation, both parties are provisioned with information such as the input password needed to run the protocol. The registration phase may include communicating identities, protocol version, and other parameters related to the registration record; see Section 3.2 for details.

During the first round, the Prover sends a public share, shareP, to the Verifier, which in turn responds with its own public share, shareV. Both parties then derive a shared secret used to produce encryption and authentication keys. The latter are used during the second round for key confirmation. (Section 3.4 details the key derivation and confirmation steps.) In particular, the Verifier sends a key confirmation message, confirmV, to the Prover, which in turn responds with its own key confirmation message, confirmP. (Note that shareV and confirmV **MAY** be sent in the same message.) Both parties **MUST NOT** consider the protocol complete prior to receipt and validation of these key confirmation messages.

A sample trace is shown below.



3.2. Offline Registration

The registration phase computes the values w0 and w1, as well as the registration record L=w1*P. w0 and w1 are derived by hashing the password pw with the identities of the two participants. w0 and the record L are then shared with the Verifier and stored as part of the registration record associated with the Prover. The Prover **SHOULD** derive w0 and w1 from the password before the protocol begins. Both w0 and w1 are derived using a function with range [0, p-1], which is modeled as a random oracle in [SPAKE2P-Analysis].

The registration phase also produces two random elements, M and N, in the prime-order subgroup of G. The algorithm for selecting M and N is defined in Appendix B. Importantly, this algorithm chooses M and N such that their discrete logs are not known. Precomputed values for

M and N are listed in Section 4 for each group. Applications **MAY** use different M and N values, provided they are computed, e.g., using different input seeds to the algorithm in Appendix B, as random elements for which the discrete log is unknown.

Applications using this specification **MUST** define the method used to compute w0 and w1. For example, it may be necessary to carry out various forms of normalization of the password before hashing [RFC8265]. This section contains requirements and default recommendations for computing w0 and w1.

The **RECOMMENDED** method for generating w0 and w1 is via a Password-Based Key Derivation Function (PBKDF), which is a function designed to slow down brute-force attackers. Brute-force resistance may be obtained through various computation hardness parameters such as memory or CPU cycles and are typically configurable. The scrypt [RFC7914] function and the Argon2id [RFC9106] function are common examples of PBKDFs. Absent an application-specific profile, **RECOMMENDED** parameters (N, r, p) for scrypt are (32768,8,1), and **RECOMMENDED** parameters for Argon2id are in Section 4 of [RFC9106].

Each half of the output of the PBKDF will be interpreted as an integer and reduced modulo p. To control bias, each half must be of length at least ceil(log2(p)) + k bits, with k >= 64. Reducing such integers mod p gives bias at most 2^-k for any p; this bias is negligible for any k >= 64.

The minimum total output length of the PBKDF then is 2 * (ceil(log2(p)) + k) bits. For example, given the prime order of the P-256 curve, the output of the PBKDF **SHOULD** be at least 640 bits or 80 bytes.

Given a PBKDF, password pw, and identities idProver and idVerifier, the **RECOMMENDED** method for computing w0 and w1 is as follows:

```
w0s || w1s = PBKDF(len(pw) || pw ||
                   len(idProver) || idProver ||
                   len(idVerifier) || idVerifier)
w0 = w0s mod p
w1 = w1s mod p
```

If an identity is unknown at the time of computing w0s or w1s, its length is given as zero and the identity itself is represented as an empty octet string. If both idProver and idVerifier are unknown, then their lengths are given as zero and both identities will be represented as empty octet strings. idProver and idVerifier are included in the transcript TT as part of the protocol flow.

3.3. Online Authentication

The online SPAKE2+ protocol runs between the Prover and Verifier to produce a single shared secret upon completion. To begin, the Prover selects x uniformly at random from the integers in [0, p-1], computes the public share shareP=X, and transmits it to the Verifier.

```
x <- [0, p-1]
X = x*P + w0*M
```

Upon receipt of X, the Verifier checks the received element for group membership and aborts if X is not in the large prime-order subgroup of G; see Section 6 for details. The Verifier then selects y uniformly at random from the integers in [0, p-1], computes the public share shareV=Y, and transmits it to the Prover. Upon receipt of Y, the Prover checks the received element for group membership and aborts if Y is not in the large prime-order subgroup of G.

```
y <- [0, p-1]
Y = y*P + w0*N
```

Both participants compute Z and V; Z and V are then shared as common values. The Prover computes:

```
Z = h*x*(Y - w0*N)
V = h*w1*(Y - w0*N)
```

The Verifier computes:

```
Z = h*y*(X - w0*M)
V = h*y*L
```

The multiplication by the cofactor h prevents small subgroup confinement attacks. All proofs of security hold even if the discrete log of the fixed group element N is known to the adversary. In particular, one **MAY** set N=I, i.e., set N to the unit element in G.

It is essential that both Z and V be used in combination with the transcript to derive the keying material. The protocol transcript encoding is shown below.

```
TT = len(Context) || Context
  || len(idProver) || idProver
  || len(idVerifier) || idVerifier
  || len(M) || M
  || len(N) || N
  || len(shareP) || shareP
  || len(shareV) || shareV
  || len(Z) || Z
  || len(V) || V
  || len(w0) || w0
```

Context is an application-specific customization string shared between both parties and **MUST** precede the remaining transcript. It might contain the name and version number of the higher-level protocol, or simply the name and version number of the application. The context **MAY** include additional data such as the chosen ciphersuite and PBKDF parameters like the iteration count or salt. The context and its length prefix **MAY** be omitted.

If an identity is absent, its length is given as zero and the identity itself is represented as an empty octet string. If both identities are absent, then their lengths are given as zero and both are represented as empty octet strings. In applications where identities are not implicit, idProver and idVerifier **SHOULD** always be non-empty. Otherwise, the protocol risks unknown key-share attacks (discussion of unknown key-share attacks in a specific protocol is given in [RFC8844]).

Upon completion of this protocol, both parties compute shared secrets K_main, K_shared, K_confirmP, and K_confirmV as specified in Section 3.4. The Verifier **MUST** send a key confirmation message, confirmV, to the Prover so both parties can confirm that they agree upon these shared secrets. After receipt and verification of the Verifier's confirmation message, the Prover **MUST** respond with its confirmation message. The Verifier **MUST NOT** send application data to the Prover until it has received and verified the confirmation message. Key confirmation verification requires recomputation of confirmP or confirmV and checking for equality against the data that was received.

3.4. Key Schedule and Key Confirmation

The protocol transcript TT, as defined in Section 3.3, is unique and secret to the participants. Both parties use TT to derive the shared symmetric secret K_main from the protocol. The length of K_main is equal to the length of the digest output, e.g., 256 bits for Hash() = SHA-256. The confirmation keys K_confirmP and K_confirmV, as well as the shared key K_shared, are derived from K_main.

```
K_main = Hash(TT)
K_confirmP || K_confirmV = KDF(nil, K_main, "ConfirmationKeys")
K_shared = KDF(nil, K_main, "SharedKey")
```

Neither K_main nor its derived confirmation keys are used for anything except key derivation and confirmation and **MUST** be discarded after the protocol execution. Applications **MAY** derive additional keys from K_shared as needed.

The length of each confirmation key is dependent on the MAC function of the chosen ciphersuite. For HMAC, the **RECOMMENDED** key length is equal to the output length of the digest output, e.g., 256 bits for Hash() = SHA-256. For CMAC-AES, each confirmation key **MUST** be of length k, where k is the chosen AES key size, e.g., 128 bits for CMAC-AES-128.

Both endpoints **MUST** employ a MAC that produces pseudorandom tags for key confirmation. K_confirmP and K_confirmV are symmetric keys used to compute tags confirmP and confirmV over the public key shares received from the other peer earlier.

```
confirmP = MAC(K_confirmP, shareV)
confirmV = MAC(K_confirmV, shareP)
```

Once key confirmation is complete, applications **MAY** use K_shared as an authenticated shared secret as needed. For example, applications **MAY** derive one or more keys and nonces from K_shared, for use with Authenticated Encryption with Associated Data (AEAD) and subsequent application data encryption.

4. Ciphersuites

This section documents SPAKE2+ ciphersuite configurations. A ciphersuite indicates a group, cryptographic hash algorithm, and pair of KDF and MAC functions, e.g., P256-SHA256-HKDF-HMAC-SHA256. This ciphersuite indicates a SPAKE2+ protocol instance over P-256 that uses SHA256 along with HKDF [RFC5869] and HMAC [RFC2104] for G, Hash, KDF, and MAC functions, respectively. Since the choice of PBKDF, its parameters for computing w0 and w1, and the distribution of w0 and w1 do not affect interoperability, the PBKDF is not included as part of the ciphersuite.

If no MAC algorithm is used in the key confirmation phase, its respective column in Table 1 can be ignored and the ciphersuite name will contain no MAC identifier.

G	Hash	KDF	MAC
P-256	SHA256 [RFC6234]	HKDF-SHA256 [RFC5869]	HMAC-SHA256 [RFC2104]
P-256	SHA512 [RFC6234]	HKDF-SHA512 [RFC5869]	HMAC-SHA512 [RFC2104]
P-384	SHA256 [RFC6234]	HKDF-SHA256 [RFC5869]	HMAC-SHA256 [RFC2104]
P-384	SHA512 [RFC6234]	HKDF-SHA512 [RFC5869]	HMAC-SHA512 [RFC2104]
P-521	SHA512 [RFC6234]	HKDF-SHA512 [RFC5869]	HMAC-SHA512 [RFC2104]
edwards25519	SHA256 [RFC6234]	HKDF-SHA256 [RFC5869]	HMAC-SHA256 [RFC2104]
edwards448	SHA512 [RFC6234]	HKDF-SHA512 [RFC5869]	HMAC-SHA512 [RFC2104]
P-256	SHA256 [RFC6234]	HKDF-SHA256 [RFC5869]	CMAC-AES-128 [RFC4493]
P-256	SHA512 [RFC6234]	HKDF-SHA512 [RFC5869]	CMAC-AES-128 [RFC4493]

Table 1

The following points represent permissible point generation seeds for the groups listed in Table 1, using the algorithm presented in Appendix B. These byte strings are compressed points as in [SEC1] for curves from [SEC1] and [RFC8032]. Note that these values are identical to those used in the companion SPAKE2 specification [RFC9382].

For P-256:

```
M =
02886e2f97ace46e55ba9dd7242579f2993b64e16ef3dcab95afd497333d8fa12f
seed: 1.2.840.10045.3.1.7 point generation seed (M)

N =
03d8bbd6c639c62937b04d997f38c3770719c629d7014d49a24b4f98baa1292b49
seed: 1.2.840.10045.3.1.7 point generation seed (N)
```

For P-384:

```
M =
030ff0895ae5ebf6187080a82d82b42e2765e3b2f8749c7e05eba366434b363d3dc
36f15314739074d2eb8613fceec2853
seed: 1.3.132.0.34 point generation seed (M)

N =
02c72cf2e390853a1c1c4ad816a62fd15824f56078918f43f922ca21518f9c543bb
252c5490214cf9aa3f0baab4b665c10
seed: 1.3.132.0.34 point generation seed (N)
```

For P-521:

```
M =
02003f06f38131b2ba2600791e82488e8d20ab889af753a41806c5db18d37d85608
cfae06b82e4a72cd744c719193562a653ea1f119eef9356907edc9b56979962d7aa
seed: 1.3.132.0.35 point generation seed (M)

N =
0200c7924b9ec017f3094562894336a53c50167ba8c5963876880542bc669e494b25
32d76c5b53dfb349fdf69154b9e0048c58a42e8ed04cef052a3bc349d95575cd25
seed: 1.3.132.0.35 point generation seed (N)
```

For edwards25519:

```
M =
d048032c6ea0b6d697ddc2e86bda85a33adac920f1bf18e1b0c6d166a5cecdaf
seed: edwards25519 point generation seed (M)

N =
d3bfb518f44f3430f29d0c92af503865a1ed3281dc69b35dd868ba85f886c4ab
seed: edwards25519 point generation seed (N)
```

For edwards448:

```
M =
b6221038a775ecd007a4e4dde39fd76ae91d3cf0cc92be8f0c2fa6d6b66f9a12
942f5a92646109152292464f3e63d354701c7848d9fc3b8880
seed: edwards448 point generation seed (M)

N =
6034c65b66e4cd7a49b0edec3e3c9ccc4588afd8cf324e29f0a84a072531c4db
f97ff9af195ed714a689251f08f8e06e2d1f24a0ffc0146600
seed: edwards448 point generation seed (N)
```

5. IANA Considerations

This document has no IANA actions.

6. Security Considerations

SPAKE2+ appears in [TDH] and is proven secure in [SPAKE2P-Analysis].

The ephemeral randomness used by the Prover and Verifier **MUST** be generated using a cryptographically secure Pseudorandom Number Generator (PRNG).

Elements received from a peer **MUST** be checked for group membership: failure to properly deserialize and validate group elements can lead to attacks. An endpoint **MUST** abort the protocol if any received public value is not a member of the large prime-order subgroup of G. Multiplication of a public value V by the cofactor h will yield the identity element I whenever V is an element of a small-order subgroup. Consequently, the Prover and Verifier **MUST** abort the protocol upon receiving any value V such that V*h = I. Failure to do so may lead to subgroup confinement attacks.

7. References

7.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[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>.

[RFC4493] Song, JH., Poovendran, R., Lee, J., and T. Iwata, "The AES-CMAC Algorithm", RFC 4493, DOI 10.17487/RFC4493, June 2006, <https://www.rfc-editor.org/info/rfc4493>.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/info/rfc5480>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[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>.

[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>.

[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>.

[RFC8265] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", RFC 8265, DOI 10.17487/RFC8265, October 2017, <https://www.rfc-editor.org/info/rfc8265>.

[RFC9382] Ladd, W., "SPAKE2, a Password-Authenticated Key Exchange", RFC 9382, DOI 10.17487/RFC9382, September 2023, <https://www.rfc-editor.org/info/rfc9382>.

[SEC1] Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography", version 2.0, May 2009, <https://secg.org/sec1-v2.pdf>.

[SPAKE2P-Analysis] Shoup, V., "Security analysis of SPAKE2+", March 2020, <https://eprint.iacr.org/2020/313.pdf>.

[TDH] Cash, D., Kiltz, E., and V. Shoup, "The Twin-Diffie Hellman Problem and Applications", EUROCRYPT 2008, Lecture Notes in Computer Science, Volume 4965, pages 127-145, Springer-Verlag, Berlin, Germany, DOI 10.1007/978-3-540-78967-3_8, April 2008, <https://doi.org/10.1007/978-3-540-78967-3_8>.

7.2. Informative References

[RFC7914] Percival, C. and S. Josefsson, "The scrypt Password-Based Key Derivation Function", RFC 7914, DOI 10.17487/RFC7914, August 2016, <https://www.rfc-editor.org/info/rfc7914>.

[RFC8844] Thomson, M. and E. Rescorla, "Unknown Key-Share Attacks on Uses of TLS with the Session Description Protocol (SDP)", RFC 8844, DOI 10.17487/RFC8844, January 2021, <https://www.rfc-editor.org/info/rfc8844>.

[RFC9106] Biryukov, A., Dinu, D., Khovratovich, D., and S. Josefsson, "Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications", RFC 9106, DOI 10.17487/RFC9106, September 2021, <https://www.rfc-editor.org/info/rfc9106>.

Appendix A. Protocol Flow

This section describes the flow of the SPAKE2+ protocol, including computations and mandatory checks performed by the Prover and Verifier. The constants M, N, P, p, and h are defined by the chosen ciphersuite.

A.1. Prover

The Prover implements two functions, ProverInit and ProverFinish, which are described below.

```
def ProverInit(w0):
   // Compute Prover key share
   x <- [0, p-1]
   X = x*P + w0*M
   return (x, X)

def ProverFinish(w0, w1, x, Y):
   if not_in_subgroup(Y):
      raise "invalid input"

   // Compute shared values
   Z = h*x*(Y - w0*N)
   V = h*w1*(Y - w0*N)

   return (Y, Z, V)
```

A.2. Verifier

The Verifier implements a single function, VerifierFinish, which is described below.

```
def VerifierFinish(w0, L, X):
   if not_in_subgroup(X):
      raise "invalid input"

   // Compute Verifier key share
   y <- [0, p-1]
   Y = y*P + w0*N

   // Compute shared values
   Z = h*y*(X - w0*M)
   V = h*y*L

   return (Z, V)
```

A.3. Transcript Computation

Both the Prover and the Verifier share the same function to compute the protocol transcript, ComputeTranscript, which is described below.

```
def ComputeTranscript(Context, idProver, idVerifier,
                      shareP, shareV, Z, V, w0):
   TT = len(Context) || Context
     || len(idProver) || idProver
     || len(idVerifier) || idVerifier
     || len(M) || M
     || len(N) || N
     || len(shareP) || shareP
     || len(shareV) || shareV
     || len(Z) || Z
     || len(V) || V
     || len(w0) || w0
```

A.4. Key Schedule Computation

Both the Prover and the Verifier share the same function to compute the key schedule, ComputeKeySchedule, which is described below.

```
def ComputeKeySchedule(TT):
   K_main = Hash(TT)
   K_confirmP || K_confirmV = KDF(nil, K_main, "ConfirmationKeys")
   K_shared = KDF(nil, K_main, "SharedKey")
   return K_confirmP, K_confirmV, K_shared
```

A.5. Protocol Run

A full SPAKE2+ protocol run initiated by the Prover will look as follows, where Transmit and Receive are shorthand for sending and receiving a message to the peer:

```
Prover(Context, idProver, idVerifier, w0, w1):
   (x, X) = ProverInit(w0)
   Transmit(X)
   Y = Receive()
   (Z, V) = ProverFinish(w0, w1, x, Y)
   TT = ComputeTranscript(Context, idProver, idVerifier, X, Y,
                          Z, V, w0)
   (K_confirmP, K_confirmV, K_shared) = ComputeKeySchedule(TT)
   expected_confirmV = MAC(K_confirmV, X)
   confirmV = Receive()
   if not_equal_constant_time(expected_confirmV, confirmV):
      raise "invalid confirmation message"

   confirmP = MAC(K_confirmP, Y)
   Transmit(confirmP)

   return K_shared

Verifier(Context, idProver, idVerifier, w0, L):
   X = Receive()
   (Y, Z, V) = VerifierFinish(w0, L, X)
   Transmit(Y)
   TT = ComputeTranscript(Context, idProver, idVerifier, X, Y,
                          Z, V, w0)
   (K_confirmP, K_confirmV, K_shared) = ComputeKeySchedule(TT)
   confirmV = MAC(K_confirmV, X)
   Transmit(confirmV)

   expected_confirmP = MAC(K_confirmP, Y)
   confirmP = Receive()
   if not_equal_constant_time(expected_confirmP, confirmP):
      raise "invalid confirmation message"

   return K_shared
```

Appendix B. Algorithm Used for Point Generation

This section describes the algorithm that was used to generate the points M and N in Table 1 (Section 4). This algorithm produces M and N such that they are indistinguishable from two random points in the prime-order subgroup of G, where the discrete log of these points is unknown. See [SPAKE2P-Analysis] for additional details on this requirement.

For each curve in Table 1, we construct a string using the curve OID from [RFC5480] (as an ASCII string) or its name, combined with the needed constant -- for instance, "1.3.132.0.35 point generation seed (M)" for P-521. This string is turned into a series of blocks by hashing with SHA256, and hashing that output again to generate the next 32 bytes, and so on. This pattern is repeated for each group and value, with the string modified appropriately.

A byte string of length equal to that of an encoded group element is constructed by concatenating as many blocks as are required, starting from the first block, and truncating to the desired length. The byte string is then formatted as required for the group. In the case of Weierstrass curves, we take the desired length as the length for representing a compressed point (Section

2.3.4 of [SEC1]) and use the low-order bit of the first byte as the sign bit. In order to obtain the correct format, the value of the first byte is set to 0x02 or 0x03 (clearing the first six bits and setting the seventh bit), leaving the sign bit as it was in the byte string constructed by concatenating hash blocks. For the curves described in [RFC8032], a different procedure is used. For edwards448, the 57-byte input has the least-significant 7 bits of the last byte set to zero, and for edwards25519, the 32-byte input is not modified. For both of the curves described in [RFC8032], the (modified) input is then interpreted as the representation of the group element. If this interpretation yields a valid group element with the correct order (p), the (modified) byte string is the output. Otherwise, the initial hash block is discarded and a new byte string constructed from the remaining hash blocks. The procedure for constructing a byte string of the appropriate length, formatting it as required for the curve, and checking to see if it is a valid point of the correct order is repeated until a valid element is found.

The following Python snippet generates the above points, assuming an elliptic curve implementation following the interface of Edwards25519Point.stdbase() and Edwards448Point.stdbase() in Appendix A of [RFC8032]:

```
def iterated_hash(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 = [iterated_hash(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
```

Appendix C. Test Vectors

This section contains various test vectors for SPAKE2+. (The choice of PBKDF is omitted, and values for w0 and w1 are provided directly.) All points are encoded using the uncompressed format, i.e., with a 0x04 octet prefix, specified in [SEC1]. idProver and idVerifier identity strings are provided in the protocol invocation.

```
[Context=b'SPAKE2+-P256-SHA256-HKDF-SHA256-HMAC-SHA256 Test Vectors
']
[idProver=b'client']
[idVerifier=b'server']
w0 = 0xbb8e1bbcf3c48f62c08db243652ae55d3e5586053fca77102994f23ad9549
1b3
w1 = 0x7e945f34d78785b8a3ef44d0df5a1a97d6b3b460409a345ca7830387a74b1
dba
L = 0x04eb7c9db3d9a9eb1f8adab81b5794c1f13ae3e225efbe91ea487425854c7f
c00f00bfedcbd09b2400142d40a14f2064ef31dfaa903b91d1faea7093d835966efd
x = 0xd1232c8e8693d02368976c174e2088851b8365d0d79a9eee709c6a05a2fad5
39
shareP = 0x04ef3bd051bf78a2234ec0df197f7828060fe9856503579bb17330090
42c15c0c1de127727f418b5966afadfdd95a6e4591d171056b333dab97a79c7193e3
41727
y = 0x717a72348a182085109c8d3917d6c43d59b224dc6a7fc4f0483232fa6516d8
b3
shareV = 0x04c0f65da0d11927bdf5d560c69e1d7d939a05b0e88291887d679fcad
ea75810fb5cc1ca7494db39e82ff2f50665255d76173e09986ab46742c798a9a6843
7b048
Z = 0x04bbfce7dd7f277819c8da21544afb7964705569bdf12fb92aa388059408d5
0091a0c5f1d3127f56813b5337f9e4e67e2ca633117a4fbd559946ab474356c41839
V = 0x0458bf27c6bca011c9ce1930e8984a797a3419797b936629a5a937cf2f11c8
b9514b82b993da8a46e664f23db7c01edc87faa530db01c2ee405230b18997f16b68
TT = 0x380000000000000005350414b45322b2d503235362d5348413235362d484b4
4462d5348413235362d484d41432d534841323536205465737420566563746f72730
60000000000000636c69656e74060000000000000736572766572410000000000
00004886e2f97ace46e55ba9dd7242579f2993b64e16ef3dcab95afd497333d8fa12
f5ff355163e43ce224e0b0e65ff02ac8e5c7be09419c785e0ca547d55a12e2d20410
000000000000004d8bbd6c639c62937b04d997f38c3770719c629d7014d49a24b4f9
8baa1292b4907d60aa6bfade45008a636337f5168c64d9bd36034808cd564490b1e6
56edbe7410000000000000004ef3bd051bf78a2234ec0df197f7828060fe98565035
79bb1733009042c15c0c1de127727f418b5966afadfdd95a6e4591d171056b333dab
97a79c7193e34172741000000000000004c0f65da0d11927bdf5d560c69e1d7d939
a05b0e88291887d679fcadea75810fb5cc1ca7494db39e82ff2f50665255d76173e0
9986ab46742c798a9a68437b048410000000000000004bbfce7dd7f277819c8da215
44afb7964705569bdf12fb92aa388059408d50091a0c5f1d3127f56813b5337f9e4e
67e2ca633117a4fbd559946ab474356c41839410000000000000000458bf27c6bca01
1c9ce1930e8984a797a3419797b936629a5a937cf2f11c8b9514b82b993da8a46e66
4f23db7c01edc87faa530db01c2ee405230b18997f16b68200000000000000bb8e1
bbcf3c48f62c08db243652ae55d3e5586053fca77102994f23ad95491b3
K_main = 0x4c59e1ccf2cfb961aa31bd9434478a1089b56cd11542f53d3576fb6c2
a438a29
K_confirmP = 0x871ae3f7b78445e34438fb284504240239031c39d80ac23eb5ab9
be5ad6db58a
K_confirmV = 0xccd53c7c1fa37b64a462b40db8be101cedcf838950162902054e6
44b400f1680
```

```
HMAC(K_confirmP, shareV) = 0x926cc713504b9b4d76c9162ded04b5493e89109
f6d89462cd33adc46fda27527
HMAC(K_confirmV, shareP) = 0x9747bcc4f8fe9f63defee53ac9b07876d907d55
047e6ff2def2e7529089d3e68
K_shared = 0x0c5f8ccd1413423a54f6c1fb26ff01534a87f893779c6e68666d772
bfd91f3e7
```

```
[Context=b'SPAKE2+-P256-SHA512-HKDF-SHA512-HMAC-SHA512 Test Vectors
']
[idProver=b'client']
[idVerifier=b'server']
w0 = 0x1cc5207d6e34b8f7828206fb64b86aa9c712bc952abf251bb9f5856b24d8c
8cc
w1 = 0x4279649e62532b01dc27d2ed39100ba350518fb969672061a01edce752d0e
672
L = 0x043a348ad475d2200d46df876f1eb2e136056da31dafff52cc7762bf3be84d
e0097c4e69b0b9321326af1f0af4a14561a9c7b640cb5afd6822d14cb34830fc4511
x = 0xb586ab83f175c1a2b56b6a1b6a283523f88a9befcf11e22efb48e2ee1fe69a
23
shareP = 0x04a7928c4b47f6b8657a5b8ebcb6f1bd266192e152fb9745a4180c946
57a2f323b4d50d536c0325cdb0ec42c9bd8db8d7af3ff6dc85edb4b5365375c62e09
def4a
y = 0xac1fb828f041782d452ea9cc00c3fa34a55fa8f7f98c04be45a3d607b092d4
41
shareV = 0x04498c29e37dbd53ebf8db76679901d90c6be3af57f46ac3025b32420
839f0489c6c3b6bf5ddc8ecbc3d7c83d0891ad814a00ad23eba13197c9d96a5b1027
5e35d
Z = 0x04a81e31be54283cee81bf7bdc877764b6b2ac6a399f1176380aac8a82172c
18051aa17dfcf438896ad253f53b52cd45ec2c7399488a919bcfcfecc0261cbf5284
V = 0x04de0a53f96cbe4abcd31c1e0a23ea6f169c162dc5a007393c8fcddd2abd5d
518bb2d9734b1d2dfce3fd916e991ab9dc3a2760d439c083eb39b65408857d2bb4aa
TT = 0x380000000000000005350414b45322b2d503235362d5348413531322d484b4
4462d5348413531322d484d41432d534841353132205465737420566563746f72730
60000000000000636c69656e74060000000000000073657276657241000000000000
00004886e2f97ace46e55ba9dd7242579f2993b64e16ef3dcab95afd497333d8fa12
f5ff355163e43ce224e0b0e65ff02ac8e5c7be09419c785e0ca547d55a12e2d20410
000000000000004d8bbd6c639c62937b04d997f38c3770719c629d7014d49a24b4f9
8baa1292b4907d60aa6bfade45008a636337f5168c64d9bd36034808cd564490b1e6
56edbe7410000000000000004a7928c4b47f6b8657a5b8ebcb6f1bd266192e152fb9
745a4180c94657a2f323b4d50d536c0325cdb0ec42c9bd8db8d7af3ff6dc85edb4b5
365375c62e09def4a41000000000000000004498c29e37dbd53ebf8db76679901d90c6
be3af57f46ac3025b32420839f0489c6c3b6bf5ddc8ecbc3d7c83d0891ad814a00ad
23eba13197c9d96a5b10275e35d410000000000000004a81e31be54283cee81bf7bd
c877764b6b2ac6a399f1176380aac8a82172c18051aa17dfcf438896ad253f53b52c
d45ec2c7399488a919bcfcfecc0261cbf52844100000000000000004de0a53f96cbe4
abcd31c1e0a23ea6f169c162dc5a007393c8fcddd2abd5d518bb2d9734b1d2dfce3f
d916e991ab9dc3a2760d439c083eb39b65408857d2bb4aa20000000000000001cc52
07d6e34b8f7828206fb64b86aa9c712bc952abf251bb9f5856b24d8c8cc
K_main = 0x527613439c279a375c116342a4216a8d92441d2fe1921dd1e60f140b2
855916ccac7db4dbf22bd56e344a8cd506d08949bde1e9d83c24d68ff4246458dc14
288
K_confirmP = 0x0aa129d7b82067c2a9607677c9c4fdedc1cd7cfed9ff72c54c0ae
bb2b1a8aa915b96834b2986725c6040852ceaafbb17d638a715198f795654eac89bf
0739878
K_confirmV = 0xa1f1038de30a8c12d43d06c27d362daa9699249e941faa2d5cbc5
9a9683bf42aed9537818245677fdb54b5274506542994f4a83455f6d7b3af5ec017f
```

```
aa58f61
HMAC(K_confirmP, shareV) = 0x6b2469b56cf8ac3f94a8d0b533380ea6b3d0f46
b3e12ee82550d49e129c2412728c9437a64ee5f80c8cdc5e8a30faa0a6deb8a52513
46ba81bb6fc955b2304fc
HMAC(K_confirmV, shareP) = 0x154174fc278a935e290b3352ba877e179fa9281
c0a76928faea703c72d383b267511a5cf084cb07147efece94e3cfd91944e7baab85
6858fbebc087167b0f409
K_shared = 0x11887659d9e002f34fa6cc270d33570f001b2a3fc0522b643c07327
d09a4a9f47aab85813d13c585b53adf5ac9de5707114848f3dc31a4045f69a2cc197
2b098
```

```
[Context=b'SPAKE2+-P384-SHA256-HKDF-SHA256-HMAC-SHA256 Test Vectors
']
[idProver=b'client']
[idVerifier=b'server']
w0 = 0x097a61cbb1cee72bb654be96d80f46e0e3531151003903b572fc193f23377
2c23c22228884a0d5447d0ab49a656ce1d2
w1 = 0x18772816140e6c3c3938a693c600b2191118a34c7956e1f1cd5b0d519b56e
a5858060966cfaf27679c9182129949e74f
L = 0x04f27dd5384d6b9beb4c5022c94b1978d632779e1d3abe458611e734a529d0
04e25053398e5dc9eeaa4ffa59743ca7ddbc0e7ce69155295cb2b846da83ee6a4449
0dd8e96bb0b0f6645281bfd978dd5f6836561ea0d8b2c045ff04cef2e5873d2c
x = 0x2f1bdbeda162ff2beba0293d3cd3ae95f663c53663378c7e18ee8f56a4a48b
00d31ce0ef43606548da485058f12e8e73
shareP = 0x049fb0404ca7ce71fb85d3aaa8fd05fa054affac996135bc245149be0
9571e43e2bf76e00d6d52ac452b8224f6b9da31420a4f5e214b377546daad4d61da5
ca0cfdea59a5a92ebdb6b42da5d14663b8d1f9eb97050139ab89788e0ada27b048fc
f
y = 0xbbcaf02404a16ed4fa73b183f703a8d969386f3d34f5e98b3a904e760512f1
1757f07dfcf87a2ada8fc6d028445bd53e
shareV = 0x0493b1c1f6a30eac4ac4a15711e44640bae3576787627ee2541104298
1e94b2e9604b9374f66bb247bc431759212ef3fa0a20c087863b89efb32219e1337c
e94be2175f8cb9fd50cf0b84772717fd063c52b69de1229a01ab840b55993287f32e
d
Z = 0x048cd880e5147e49b42b5754c1bc6d2091ad414789bc3b030f2d787ea480f3
e35d0fa0d02d0dd06fee7f242b702a2d984efd79c76d99ab35b99e359a205cea56bb
a8dd8f995c101a69a5157686d1cf6a7288d7cff2f2a9748db99b24f646ea7b37
V = 0x041c3c9cc38b03a06a49cf17cc5e7754cf1ccbbc6fffc0ddf1a6e23f57294a
25d96f7da5ce4ac0a617c78502f2f235a5fcf2f76a62385434ed2b6e95521b41eff3
c4ce93ecf8fb32005dd76335d0a7c78153257288d7fde1a22d404f5d73d068e2
TT = 0x380000000000000005350414b45322b2d503338342d5348413235362d484b4
4462d5348413235362d484d41432d534841323536205465737420566563746f72730
600000000000000636c69656e7406000000000000007365727665726100000000000
000040ff0895ae5ebf6187080a82d82b42e2765e3b2f8749c7e05eba366434b363d3
dc36f15314739074d2eb8613fceec285397592c55797cdd77c0715cb7df2150220a0
119866486af4234f390aad1f6addde5930909adc67a1fc0c99ba3d52dc5dd6100000
00000000004c72cf2e390853a1c1c4ad816a62fd15824f56078918f43f922ca21518
f9c543bb252c5490214cf9aa3f0baab4b665c10c38b7d7f4e7f320317cd717315a79
7c7e02933aef68b364cbf84ebc619bedbe21ff5c69ea0f1fed5d7e3200418073f406
100000000000000049fb0404ca7ce71fb85d3aaa8fd05fa054affac996135bc24514
9be09571e43e2bf76e00d6d52ac452b8224f6b9da31420a4f5e214b377546daad4d6
1da5ca0cfdea59a5a92ebdb6b42da5d14663b8d1f9eb97050139ab89788e0ada27b0
48fcf6100000000000000000493b1c1f6a30eac4ac4a15711e44640bae3576787627ee
25411042981e94b2e9604b9374f66bb247bc431759212ef3fa0a20c087863b89efb3
2219e1337ce94be2175f8cb9fd50cf0b84772717fd063c52b69de1229a01ab840b55
993287f32ed6100000000000000048cd880e5147e49b42b5754c1bc6d2091ad41478
```

```
9bc3b030f2d787ea480f3e35d0fa0d02d0dd06fee7f242b702a2d984efd79c76d99a
b35b99e359a205cea56bba8dd8f995c101a69a5157686d1cf6a7288d7cff2f2a9748
db99b24f646ea7b37610000000000000000041c3c9cc38b03a06a49cf17cc5e7754cf1
ccbbc6fffc0ddf1a6e23f57294a25d96f7da5ce4ac0a617c78502f2f235a5fcf2f76
a62385434ed2b6e95521b41eff3c4ce93ecf8fb32005dd76335d0a7c78153257288d
7fde1a22d404f5d73d068e2300000000000000000097a61cbb1cee72bb654be96d80f4
6e0e3531151003903b572fc193f233772c23c22228884a0d5447d0ab49a656ce1d2
K_main = 0x61370f8bf65e0df7e9a7b2c2289be1ee4b5dd6c21f4b85165730700c4
4ce30af
K_confirmP = 0x2c8940419d94e53d5d240801e702c4658531aa7a9f14ec75f0d67
f12fa84196c
K_confirmV = 0x8e74afe16c53a44590ad6bf43aa89324978b8f20014336675f618
387f99f3fdc
HMAC(K_confirmP, shareV) = 0x7ae825e242a5a1f86ad7db172c2c12fcb458b6a
2b1ddfc96b2b7cfd2eed5f7ab
HMAC(K_confirmV, shareP) = 0x1581062167d6a3d14493447cd170d408f6fdc58
e31225438db86214167426a7a
K_shared = 0x99758e838ae1a856589689fb55b6befe4e2382e6ebbeca1a6232a68
f9dc04c1a
```

```
[Context=b'SPAKE2+-P384-SHA512-HKDF-SHA512-HMAC-SHA512 Test Vectors
']
[idProver=b'client']
[idVerifier=b'server']
w0 = 0xb8d44a0982b88abe19b724d4bdafba8c90dc93130e0bf4f8062810992326d
a126fd01db53e40250ca33a3ff302044cb0
w1 = 0x2373e2071c3bb2a6d53ece57830d56f8080189816803c22375d6a4a514f9d
161b64d0f05b97735b98b348f9b33cc2e30
L = 0x049ca7217ff6456bb2e2bcf71b31d9b1e5ed6e0c9700936ae617e990cee87e
e1ce3a03629dd5532948c39b89f38b39f13c7f513c5b1ada00f6533a4a8b02b9cd04
e1b2a5db1f24ec5fe959198a19666037e04b768cc02e75ac9da0048736db6e5b
x = 0x5a835d52714f30d2ef539268b89df9558628400063dfa0e41eb979066f4caf
409bbf7aab3ddddea13f1b070a1827d3d4
shareP = 0x042f382eef464a2c9aecfdf4b81d25c4de2de113ba67405ce336c762c
69217ae7e27bda875144140d7536c4cc08b9b4dace5f872a6a2ed57f34042688ad3c
5d446c187dc0caf9cea812df3a4dd6fdbc64b9d7d7d7ff4bf6965abb06eeb108d55e
e
y = 0xc883ee5b08cf7ba122038c279459ab1a730f85f2d624a02732d519faab56a4
98e773a8dec6c447ed02d5c00303a18bc4
shareV = 0x04d72e11eee332305062454c0a058b8103a3304785d445510cd8d101e
9cb44cfb159cb7b72123abaf719ab1c42e0558c84c14b0886e8b446e4c880bff2f4b
291fafafc748cb4115824e66732bdeba7fae176388e228ab9d7546255994ca3fb5a5
2
Z = 0x043cb63f5fcb573cf3e2ee40bca5fbc1f00ff2554caab3790329184c45ed69
c39b2e1323bc13c8f821b844feb5921b1470e7b3f70bd10508e5de6db157305badf8
20fa28d68742d8287fb201383a8deec70d5bcf2a61498a481290ed8cc94ab3a0
V = 0x0468604d188f4da560ddaaece126abe40f5de255f8af093c7c3aff71f95d90
92804426127d73d46a817085e9095de6bcf30733a5124a98f567148efe92a7134994
0c7244623247d33a8b78cbc9a53cd45bb22430f318a635084d1840c905f236c8
TT = 0x380000000000000005350414b45322b2d503338342d5348413531322d484b4
4462d5348413531322d484d41432d534841353132205465737420566563746f72730
60000000000000636c69656e74060000000000000073657276657261000000000000
000040ff0895ae5ebf6187080a82d82b42e2765e3b2f8749c7e05eba366434b363d3
dc36f15314739074d2eb8613fceec285397592c55797cdd77c0715cb7df2150220a0
119866486af4234f390aad1f6addde5930909adc67a1fc0c99ba3d52dc5dd6100000
00000000004c72cf2e390853a1c1c4ad816a62fd15824f56078918f43f922ca21518
```

```
f9c543bb252c5490214cf9aa3f0baab4b665c10c38b7d7f4e7f320317cd717315a79
7c7e02933aef68b364cbf84ebc619bedbe21ff5c69ea0f1fed5d7e3200418073f406
1000000000000000042f382eef464a2c9aecfdf4b81d25c4de2de113ba67405ce336c
762c69217ae7e27bda875144140d7536c4cc08b9b4dace5f872a6a2ed57f34042688
ad3c5d446c187dc0caf9cea812df3a4dd6fdbc64b9d7d7d7ff4bf6965abb06eeb108
d55ee610000000000000004d72e11eee332305062454c0a058b8103a3304785d4455
10cd8d101e9cb44cfb159cb7b72123abaf719ab1c42e0558c84c14b0886e8b446e4c
880bff2f4b291fafafc748cb4115824e66732bdeba7fae176388e228ab9d75462559
94ca3fb5a52610000000000000000043cb63f5fcb573cf3e2ee40bca5fbc1f00ff2554
caab3790329184c45ed69c39b2e1323bc13c8f821b844feb5921b1470e7b3f70bd10
508e5de6db157305badf820fa28d68742d8287fb201383a8deec70d5bcf2a61498a4
81290ed8cc94ab3a0610000000000000000468604d188f4da560ddaaece126abe40f5
de255f8af093c7c3aff71f95d9092804426127d73d46a817085e9095de6bcf30733a
5124a98f567148efe92a71349940c7244623247d33a8b78cbc9a53cd45bb22430f31
8a635084d1840c905f236c83000000000000000b8d44a0982b88abe19b724d4bdafb
a8c90dc93130e0bf4f8062810992326da126fd01db53e40250ca33a3ff302044cb0
K_main = 0x571af2e9a0bf4b354cca18d713f8a84315a46c999ceb92ca6a88b8a6d
615795140862dbccd6fdc0abecc5956c43f8ab40343a22fc1b91752cb7c2737dab90
41e
K_confirmP = 0x6c8c7fc6becf3bc07f081b4f7f867bec76fd8eeddbd7968356723
bae701e04f35f800e647dfa013b2876958efe0ce68e7595ba46f1de0b17adfc02dfe
3f18a18
K_confirmV = 0x2d0c9702a0f5536bacddd596eb6ea365d17f176db30081b97b83e
05bb87e9a36c0565b7616251c93bc76c76fc5c3531a28db40779d986d4e7b71a24c4
3fbc731
HMAC(K_confirmP, shareV) = 0x7f806ae56ea3e49a8b16ffee528086489418913
641f529d50ff92aa456ad4648e522f9540b403bff6bd94ee1adc95c7d1b2666f7ba6
f9c10748bc7bfb4181d27
HMAC(K_confirmV, shareP) = 0x8daa262decb79cceda4421f4f8dacf22ec027c0
8e036f071beea563c8e00813a29807963ff9d7d6bbff48dd5bdcdd9ca9fd7ffc272b
162258d981913f7253dcb
K_shared = 0x31e0075a823b9269af5769d71ef3b2f5001cbfe044584fe8551124a
217dad078415630bf3eda16b5a38341d418a6d72b3960f818a0926f0de88784b59d6
a694b
```

```
[Context=b'SPAKE2+-P521-SHA512-HKDF-SHA512-HMAC-SHA512 Test Vectors
']
[idProver=b'client']
[idVerifier=b'server']
w0 = 0x009c79bcd7656716314fca5a6e2c5cda7ef86131399438e012a043051e863
f60b5aeb3c101731e1505e721580f48535a9b0456b231b9266ae6fff49ee90d25f72
f5f
w1 = 0x01632c15f51fcd916cd79e19075f8a69b72b0099922ad62ff8d540b469569
f0aa027047aed2b3f242ea0ac4288b4e4db6a4e5946d8ad32b42192c5aa66d9ef8e1
b33
L = 0x040135072d0fa36f9e80031294cef5c3c35b882a0efa2c66570d64a49f8bec
6c66435bf65bb7c7b2a3e7dece491e02b4d567e7087dbc32fe0fae8af417dcb50be6
d704012a194588b690e6d3db492656f72ddea01fc1c7fcec0f5d34a5af0102939f6f
deae39c20cff74fcdb7f09855f0fc9520d20b0520b0b096b8d42c7c3d68b4a66f751
x = 0x00b69e3bb15df82c9fa0057461334e2c66ab92fc9b8d3662eec81216ef5ddc
4a43f19e90dedaa2d72502f69673a115984ffcf88e03a9364b07102114c5602cd93c
69
shareP = 0x0400a14431edf6852ff5fe868f8683e16e9e0a45d9e27f9a96442285a
c6b161fc0bf267362a5ffb06f9cbd14b7a37e492146d77cae4c77812df00a91dbae0
9e27e1fac00ae019317ef9768548325bca35ce258e6206fe03c6338b2eb889d09d9f
11400a36cf6328a7e1f81c6c7a2af7ff1d9b5210768318f27e57b75b39b9fbfc7b37
```

```
a60ab
y = 0x0056d01c5246fbde964c0493934b7ece89eafd943eb27d357880a2a2202249
9e249528c5707b1afe8794c8a1d60ceedaeed96dd0dd904ea075f096c9fec5da7de4
96
shareV = 0x0401aa5af0f3027f63b7170572db5ff06dd1f3d6ea8ea771b26b434fb
bc6c9de7d80975131c9c2e94d30c0ed2d62449c4c1b7e95037a85ed7598e415a2591
26365e89500d0f2156b551b70416d719944736990f346f6f9ba4fbaf2f63e0987369
0bcf730582e0a7b03ffede50f5787b631d5021a94287f0a29a081b62b9f5a3bf393b
001b3
Z = 0x0401e3015bf2811891a518d342c63541294dc80e0ee210e8220a5b9cab010d
77945724ef1185d739a62847fdada9da9b1bca6b9fa173fa551185c6084c3db26d3a
f0ac01f9356d01beebebd5ff026ca19f9df5d614355f3498816ac20b63bc936eed82
8a7039d1e17dba740471d9afc0e0b4427d65b2d27a57a87e42300004e2b4620c23c9
V = 0x0401058b21ca71e4439281579d6df3b86ae874d70742fe8eae2de60e77e07e
6e1c31b9c277de36b38531f5b769e9e4030ba09258f510c83c5c21957610355ce920
1fe600672db35efd1d0903bc285d4e27e9fb4472c30f17118dfa028f182bc9361c6a
749f560e31b9c404624d24e68010f064101d4a1154e77be8f2105dbeb8b0349adb0e
TT = 0x380000000000000005350414b45322b2d503532312d5348413531322d484b4
4462d5348413531322d484d41432d534841353132205465737420566563746f72730
60000000000000636c69656e7406000000000000007365727665728500000000000
00004003f06f38131b2ba2600791e82488e8d20ab889af753a41806c5db18d37d856
08cfae06b82e4a72cd744c719193562a653ea1f119eef9356907edc9b56979962d7a
a01bdd179a3d547610892e9b96dea1eab10bdd7ac5ae0cf75aa0f853bfd185cf782f
894301998b11d1898ede2701dca37a2bb50b4f519c3d89a7d054b51fb84912192850
00000000000000400c7924b9ec017f3094562894336a53c50167ba8c596387688054
2bc669e494b2532d76c5b53dfb349fdf69154b9e0048c58a42e8ed04cef052a3bc34
9d95575cd2501c62bee650c9287a651bb75c7f39a2006873347b769840d261d17760
b107e29f091d556a82a2e4cde0c40b84b95b878db2489ef760206424b3fe7968aa8e
0b1f33485000000000000000400a14431edf6852ff5fe868f8683e16e9e0a45d9e27
f9a96442285ac6b161fc0bf267362a5ffb06f9cbd14b7a37e492146d77cae4c77812
df00a91dbae09e27e1fac00ae019317ef9768548325bca35ce258e6206fe03c6338b
2eb889d09d9f11400a36cf6328a7e1f81c6c7a2af7ff1d9b5210768318f27e57b75b
39b9fbfc7b37a60ab850000000000000000401aa5af0f3027f63b7170572db5ff06dd
1f3d6ea8ea771b26b434fbbc6c9de7d80975131c9c2e94d30c0ed2d62449c4c1b7e9
5037a85ed7598e415a259126365e89500d0f2156b551b70416d719944736990f346f
6f9ba4fbaf2f63e09873690bcf730582e0a7b03ffede50f5787b631d5021a94287f0
a29a081b62b9f5a3bf393b001b385000000000000000000401e3015bf2811891a518d34
2c63541294dc80e0ee210e8220a5b9cab010d77945724ef1185d739a62847fdada9d
a9b1bca6b9fa173fa551185c6084c3db26d3af0ac01f9356d01beebebd5ff026ca19
f9df5d614355f3498816ac20b63bc936eed828a7039d1e17dba740471d9afc0e0b44
27d65b2d27a57a87e42300004e2b4620c23c9850000000000000000401058b21ca71e
4439281579d6df3b86ae874d70742fe8eae2de60e77e07e6e1c31b9c277de36b3853
1f5b769e9e4030ba09258f510c83c5c21957610355ce9201fe600672db35efd1d090
3bc285d4e27e9fb4472c30f17118dfa028f182bc9361c6a749f560e31b9c404624d2
4e68010f064101d4a1154e77be8f2105dbeb8b0349adb0e420000000000000000009c7
9bcd7656716314fca5a6e2c5cda7ef86131399438e012a043051e863f60b5aeb3c10
1731e1505e721580f48535a9b0456b231b9266ae6fff49ee90d25f72f5f
K_main = 0xf672a73216568d20cc3433247bc43a3b875a421cbdba76cf1db8bfe57
2b658bf3f7a4ef8cc9ff1f6a2827ff7b19860454b775a4097009040f3b36b7420407
16e
K_confirmP = 0xa211c60ea8d4b3b294bd6ca9515663b77f3caac28af3658b34fe1
512f25077f2f64b8de426caa662b4cbbdc9c2f8f12347993c8d57fdf68c177732d7d
da7277b
K_confirmV = 0x0e9bf6b9a37339144cb32a78a872f50b10839f81eda6c09a827dd
bb158c47162bec274af920cdf809f162b98fa701efebada26cdfbeac408b5a35b052
d18f0c6
HMAC(K_confirmP, shareV) = 0xf0f5c903dfa42fe367659656a26058cd984b76a
8e91ae4d0fa4c13db149008e2ae57713fb230a627761174fefd263b9c10e9a4b6a37
```

```
46cde59c5943040c17133
HMAC(K_confirmV, shareP) = 0xa8f7ab43f3a800171d3a3fb26d742e1ed236c2d
5804ecd328f220a7d245cd2e3bfb6c0526983bff9229c94f70fe64ba9bb5a4d0dc10
afcda64a4c96d4c3d81ad
K_shared = 0xd1c170e4e55efacb9db8abad286293ebd1dcf24f13973427b9632bb
c323e42e447afca2aa7f74f2af3fb5f51684ec543db854b7002cde6799c330b032ba
8820a
```

```
[Context=b'SPAKE2+-P256-SHA256-HKDF-SHA256-CMAC-AES-128 Test Vector
s']
[idProver=b'client']
[idVerifier=b'server']
w0 = 0x9aad90c603cf16cec4ee40d81acd7a865130b28cc6d0664ae2e0f406aa47e
d61
w1 = 0x872be859cec1e78d191882bd9c2f032af018a25016813788fe8954bfffc58
c8e
L = 0x04d79a53698c5dd79e14b426e73b4a7f1b42469815fe24e8f53ce01579e902
eb198d59f05bc451c41826b88e3db5476a69e197fdf474c75b387f6d40361c3fda35
x = 0x9d39a3511a007a7d3fe6af5555cf60301bcda503f2bf6634b2caf9e4fd0743
a1
shareP = 0x04788218027ba4b17f7279ef0aef47a8733cf88b5bf65d6127ecadc78
b8a0f65b9001f7e54719fb63c072ddd1e1a4adfb376dde37ba1aa2082362b6c2ca14
a8e53
y = 0x9c3219841626325c68d89c22fb6c55611e3136442daa8b9b784db7242afff3
ed
shareV = 0x04c05953ea9d1cd6248b8c61becd7d55e46237526d8b1e23495ea7566
b7f6bc24b3da1cfb2e88a975fcfb5dc4e72b5cbea509b1cfdd1ef8f8195fa8bf2bd5
ca1e5
Z = 0x049444a17ad5909548a084fa182275a89a496ec6669bd08892aa9c64a512d4
0212147e6005bf1d510e3bbcfee8efc38243acaf4c5f2decffa009341b1e330b0442
V = 0x0457a8919af393e2da1de209a01fdda275eab0a682d8931b0e6ee1b9339794
63a25ccbcda1956a6a555706f0b062aa880617bd219d09391ad8576d3a73e9233f57
TT = 0x3900000000000000005350414b45322b2d503235362d5348413235362d484b4
4462d5348413235362d434d41432d4145532d313238205465737420566563746f727
3060000000000000000636c69656e74060000000000000007365727665724100000000000
0000004886e2f97ace46e55ba9dd7242579f2993b64e16ef3dcab95afd497333d8fa
12f5ff355163e43ce224e0b0e65ff02ac8e5c7be09419c785e0ca547d55a12e2d204
10000000000000004d8bbd6c639c62937b04d997f38c3770719c629d7014d49a24b4
f98baa1292b4907d60aa6bfade45008a636337f5168c64d9bd36034808cd564490b1
e656edbe7410000000000000000004788218027ba4b17f7279ef0aef47a8733cf88b5bf
65d6127ecadc78b8a0f65b9001f7e54719fb63c072ddd1e1a4adfb376dde37ba1aa2
082362b6c2ca14a8e5341000000000000000004c05953ea9d1cd6248b8c61becd7d55e
46237526d8b1e23495ea7566b7f6bc24b3da1cfb2e88a975fcfb5dc4e72b5cbea509
b1cfdd1ef8f8195fa8bf2bd5ca1e5410000000000000000049444a17ad5909548a084f
a182275a89a496ec6669bd08892aa9c64a512d40212147e6005bf1d510e3bbcfee8e
fc38243acaf4c5f2decffa009341b1e330b0442410000000000000000457a8919af39
3e2da1de209a01fdda275eab0a682d8931b0e6ee1b933979463a25ccbcda1956a6a5
55706f0b062aa880617bd219d09391ad8576d3a73e9233f57200000000000000009aa
d90c603cf16cec4ee40d81acd7a865130b28cc6d0664ae2e0f406aa47ed61
K_main = 0x6002da6b2740056f2836ac0316ae9e02e2b24c5c109883136e90ed868
b2fcf62
K_confirmP = 0x857d0db7f5e06385853bf4b8abd43b5a
K_confirmV = 0x268c75933332157118063550c6bfe846
CMAC(K_confirmP, shareV) = 0xd340bc94a03feafd14491e316514ca5f
CMAC(K_confirmV, shareP) = 0x2b42d0fe76bcf9ccc208d06d60082f96
```

```
K_shared = 0xe832094adfc028bf288e49ab902fc208b7eeff084f259da7613c047
9869d4fc9
```

```
[Context=b'SPAKE2+-P256-SHA512-HKDF-SHA512-CMAC-AES-128 Test Vector
s']
[idProver=b'client']
[idVerifier=b'server']
w0 = 0x56e0299ac95739b616a973276c1338e3651285345dde2f7faf74c25c0b50e
b90
w1 = 0x462fe5b522a17d3d35b27323113bdd252de9cbfdd6f264b35721bf59a9a74
f0b
L = 0x040540332ffec8a2faa8d17ae6da5973c11e078b8c10c89fd6af996726b802
3513eff2914c3ced64fbedd4e261438fb0ea6ef9fc1faef4ba1ead780636faac1bc1
x = 0x254dd22780eeb6af2464dd6a2bd026b46a34966d6933607f1be956314f74b0
ea
shareP = 0x049661cfdb0f7bd24b637f8d1d0f464c17f0b9c15129ea31156dcc581
da6c840240b275d72f28ea73a5c088c99d73896af24a5ae26e036eb2dedaf26e511a
24a48
y = 0x695beec24305fbd5660bc200228598e7c891fdf60a55df4bdd3a57debc3847
4a
shareV = 0x0461f580eb3eb4b2f412d5c07491f360ad6e4492d8f23e346f0ba999f
bbcb9715a3c2485c3b250a6672e6698da3c9a9725645f607ee90a9b1b34fd44b9df6
e551a
Z = 0x0406f77a4bca254219dc3eeca9989f377037407105540bfddc5bdeff3d27a8
7d68442e69d543a000077bd4c42e33930f890d29fb4be5e8dcc627f6811ace96c274
V = 0x0442952a531a2937e03808e74f6d65afbedb4cfb7fcf91991498f77db21b14
6f5c2249e727e374de03f32848465aba5c5ebfe6501d3537d09160c7f42e4b3f133d
TT = 0x3900000000000000005350414b45322b2d503235362d5348413531322d484b4
4462d5348413531322d434d41432d4145532d313238205465737420566563746f727
3060000000000000636c69656e74060000000000000073657276657241000000000
0000004886e2f97ace46e55ba9dd7242579f2993b64e16ef3dcab95afd497333d8fa
12f5ff355163e43ce224e0b0e65ff02ac8e5c7be09419c785e0ca547d55a12e2d204
10000000000000004d8bbd6c639c62937b04d997f38c3770719c629d7014d49a24b4
f98baa1292b4907d60aa6bfade45008a636337f5168c64d9bd36034808cd564490b1
e656edbe7410000000000000000049661cfdb0f7bd24b637f8d1d0f464c17f0b9c1512
9ea31156dcc581da6c840240b275d72f28ea73a5c088c99d73896af24a5ae26e036e
b2dedaf26e511a24a4841000000000000000461f580eb3eb4b2f412d5c07491f360a
d6e4492d8f23e346f0ba999fbbcb9715a3c2485c3b250a6672e6698da3c9a9725645
f607ee90a9b1b34fd44b9df6e551a4100000000000000000406f77a4bca254219dc3ee
ca9989f377037407105540bfddc5bdeff3d27a87d68442e69d543a000077bd4c42e3
3930f890d29fb4be5e8dcc627f6811ace96c27441000000000000000442952a531a2
937e03808e74f6d65afbedb4cfb7fcf91991498f77db21b146f5c2249e727e374de0
3f32848465aba5c5ebfe6501d3537d09160c7f42e4b3f133d200000000000000056e
0299ac95739b616a973276c1338e3651285345dde2f7faf74c25c0b50eb90
K_main = 0x111790ae23de3fc5bb43bdc1f63106461dbd8d86360adf056bf117164
8bfb231503853db2625275b7136b5a823dd5a94482514fce7f791c4daca2b21c7bde
756
K_confirmP = 0xb234d2e152a03168b76c6474d5322070
K_confirmV = 0x683d62024626fe0c5126ef4df58b88ee
CMAC(K_confirmP, shareV) = 0x0dc514d262e37470eb43e058e0d615f4
CMAC(K_confirmV, shareP) = 0xde076589efcd5d96c2ea6061d96772d9
K_shared = 0x488a34663d6be5e02590bb8e9ad9ad3e0f580dec41e8b99ed4ae4b7
34da49287638cac4c9f17fe3c3ae18dda0d6d7f14c17e4640d5a2aaab959efa0cbea
4e546
```

Acknowledgements

Thanks to Ben Kaduk and Watson Ladd, from whom this specification originally emanated.

Authors' Addresses

Tim Taubert
Apple Inc.
One Apple Park Way
Cupertino, California 95014
United States of America
Email: ttaubert@apple.com

Christopher A. Wood
Email: caw@heapingbits.net