Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 8, 2019

W. Ladd
UC Berkeley
B. Kaduk, Ed.
Akamai

November 4, 2018

SPAKE2, a PAKE
draft-irtf-cfrg-spake2-07

Abstract

This document describes SPAKE2, a means for two parties that share a password to derive a strong shared key with no risk of disclosing the password. This method is compatible with any group, is computationally efficient, and has a security proof.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 8, 2019.

Table of Contents

1. Introduction

This document describes SPAKE2, a means for two parties that share a password to derive a strong shared key with no risk of disclosing the password. This password-based key exchange protocol is compatible with any group (requiring only a scheme to map a random input of fixed length per group to a random group element), is computationally efficient, and has a security proof. Predetermined parameters for a selection of commonly used groups are also provided for use by other protocols.

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

3.1. Setup

Let G be a group in which the Diffie-Hellman (DH) problem is hard of order p*h, with p a big prime and h a cofactor. We denote the operations in the group additively. Let H be a hash function from arbitrary strings to bit strings of a fixed length. Common choices for H are SHA256 or SHA512 [RFC6234]. We assume there is a representation of elements of G as byte strings: common choices would be SEC1 compressed [SEC1] for elliptic curve groups or big endian integers of a fixed (per-group) length for prime field DH.

|| 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.

We fix two elements M and N as defined in the table in this document for common groups, as well as a generator G of the group. G is specified in the document defining the group, and so we do not repeat it here.

Let A and B be two parties. We will assume that A and B also have digital representations of the parties' identities such as MAC addresses or other names (hostnames, usernames, etc). We assume they share an integer w; typically w will be the hash of a user-supplied password, truncated and taken mod p. Protocols using this specification must define the method used to compute w: it may be necessary to carry out various forms of normalization of the password before hashing. [RFC8265] The hashing algorithm SHOULD be designed to slow down brute-force attackers.

We present two protocols below. Note that it is insecure to use the same password with both protocols; passwords MUST NOT be used for both SPAKE2 and SPAKE2+.

3.2. SPAKE2

A picks x randomly and uniformly from the integers in [0,ph) divisible by h, and calculates X=x*G and T=w*M+X, then transmits T to B.

B selects y randomly and uniformly from the integers in [0,p*h), divisible by h and calculates Y=y*G, S=w*N+Y, then transmits S to A.

Both A and B calculate a group element K. A calculates it as x(S-wN), while B calculates it as y(T-w*M). A knows S because it has received it, and likewise B knows T.

This K is a shared value, but the scheme as described is not secure. K MUST be combined with the values transmitted and received via a hash function to prevent man-in-the-middle attackers from being able to insert themselves into the exchange. Higher-level protocols SHOULD prescribe a method for incorporating a "transcript" of the exchanged values and endpoint identity information into the shared secret. One such approach would be to compute a K' as H(len(A) || A || len(B) || B || len(S) || S || len(T) || T || len(K) || K || len(w) || w) and use K' as the key.

3.3. SPAKE2+

This protocol appears in [TDH]. We use the same setup as for SPAKE2, except that we have two secrets, w0 and w1, derived by hashing the password with the identities of the two participants. B stores L=w1*g and w0.

When executing SPAKE2+, A selects x uniformly at random from the numbers in the range [0, p*h) divisible by h, and lets X=x*G+w0*M, then transmits X to B. B selects y uniformly at random from the numbers in [0, p*h) divisible by h, then computes Y=y*G+w0*N, and transmits it to Alice.

A computes Z as x(Y-w0*N), and V as w1(Y-w0*N). B computes Z as y(X-w0*M) and V as y*L. Both share Z and V as common keys. It is essential that both Z and V be used in combination with the transcript to derive the keying material. For higher-level protocols without sufficient transcript hashing, let K' be H(len(A) || A || len(B) || B || len(X) || X || len(Y) || Y || len(Z) || Z || len(V) || V || len(w0) || w0) and use K' as the established key.

4. Table of points for common groups

For each curve in the table below, 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-512. 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 [RFC8032] curves 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 the [RFC8032] curves 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 of constructing a byte string of the appropriate length, formatting it as required for the curve, and checking if it is a valid point of the correct order, is repeated until a valid element is found.

These bytestrings are compressed points as in [SEC1] for curves from [SEC1].

For P256:

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 P384:

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 P521:

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)

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

```
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
```

5. Security Considerations

A security proof of SPAKE2 for prime order groups is found in [REF]. Note that the choice of M and N is critical for the security proof. The generation method specified in this document is designed to eliminate concerns related to knowing discrete logs of M and N.

SPAKE2+ appears in [TDH] along with a path to a proof that server compromise does not lead to password compromise under the DH assumption (though the corresponding model excludes precomputation attacks).

There is no key-confirmation as this is a one-round protocol. It is expected that a protocol using this key exchange mechanism will provide key confirmation separately if desired.

Elements received from a peer MUST be checked for group membership: failure to properly validate group elements can lead to attacks. In particular it is essential to verify that received points are valid compressions of points on an elliptic curve when using elliptic curves. It is not necessary to validate membership in the prime order subgroup: the multiplication by cofactors eliminates the potential for mebership in a small-order subgroup.

The choices of random numbers MUST BE uniform. Note that to pick a random multiple of h in [0, p*h) one can pick a random integer in [0, p) and multiply by h. Ephemeral values MUST NOT be reused; such reuse permits dictionary attacks on the password.

SPAKE2 does not support augmentation. As a result, the server has to store a password equivalent. This is considered a significant drawback, and so SPAKE2+ also appears in this document.

As specified, the shared secret K is not suitable for direct use as a shared key. It MUST be passed to a hash function along with the public values used to derive it and the identities of the participating parties in order to avoid attacks. In protocols which do not perform this separately, the value denoted K' MUST be used instead of K.

6. IANA Considerations

No IANA action is required.

7. Acknowledgments

Special thanks to Nathaniel McCallum and Greg Hudson for generation of test vectors. Thanks to Mike Hamburg for advice on how to deal with cofactors. Greg Hudson also suggested the addition of warnings on the reuse of x and y. Thanks to Fedor Brunner, Adam Langley, and the members of the CFRG for comments and advice. Trevor Perrin informed me of SPAKE2+.

8. References

8.1. Normative References

[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>.

[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>.

[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>.

[SEC1] SEC, "STANDARDS FOR EFFICIENT CRYPTOGRAPHY, "SEC 1: Elliptic Curve Cryptography", version 2.0", May 2009.

8.2. Informative References

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

Appears in A. Menezes, editor. Topics in Cryptography-CT-RSA 2005, Volume 3376 of Lecture Notes in Computer

Science, pages 191-208, San Francisco, CA, US. Springer-Verlag, Berlin, Germany.

[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>.

[TDH] Cash, D., Kiltz, E., and V. Shoup, "The Twin-Diffie Hellman Problem and Applications", 2008.

EUROCRYPT 2008. Volume 4965 of Lecture notes in Computer Science, pages 127-145. Springer-Verlag, Berlin, Germany.

Authors' Addresses

Watson Ladd
UC Berkeley

Email: watsonbladd@gmail.com

Benjamin Kaduk (editor)
Akamai Technologies

Email: kaduk@mit.edu