J-PAKE: Password Authenticated Key Exchange by Juggling
draft-hao-jpake-04

Abstract

   This document specifies a Password Authenticated Key Exchange by
   Juggling (J-PAKE) protocol. This protocol allows the establishment
   of a secure end-to-end communication channel between two remote
   parties over an insecure network solely based on a shared password,
   without requiring a Public Key Infrastructure (PKI) or any trusted
   third party.

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 http://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 January 7, 2017.

Copyright Notice

the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction

Password-Authenticated Key Exchange (PAKE) is a technique that aims
to establish secure communication between two remote parties solely
based on their shared password, without relying on a Public Key
Infrastructure or any trusted third party [BM92]. The first PAKE
protocol, called EKE, was proposed by Steven Bellovin and Michael
Merrit in 1992 [BM92]. Other well-known PAKE protocols include SPEKE
(by David Jablon in 1996) [Jab96] and SRP (by Tom Wu in 1998) [Wu98].
SRP has been revised several times to address reported security and
efficiency issues. In particular, the version 6 of SRP, commonly
known as SRP-6, is specified in [RFC5054].

This document specifies a PAKE protocol called Password Authenticated
Key Exchange by Juggling (J-PAKE), which was designed by Feng Hao and
Peter Ryan in 2008 [HR08].

There are a few factors that may be considered in favor of J-PAKE.
First, J-PAKE has security proofs, while equivalent proofs are
lacking in EKE, SPEKE and SRP-6. Second, J-PAKE follows a completely
different design approach from all other PAKE protocols, and is built

upon a well-established Zero Knowledge Proof (ZKP) primitive: Schnorr NIZK proof [I-D-Schnorr]. Third, J-PAKE is efficient. It adopts novel engineering techniques to optimize the use of ZKP so that overall the protocol is sufficiently efficient for practical use. Fourth, J-PAKE is designed to work generically in both the finite field and elliptic curve settings (i.e., DSA and ECDSA-like groups respectively). Unlike SPEKE, it does not require any extra primitive to hash passwords onto a designated elliptic curve. Finally, J-PAKE has been used in real-world applications at a relatively large scale, e.g., Firefox sync [1], Pale moon sync [2] and Google Nest products [ABM15]; it has been included into widely distributed open source libraries such as OpenSSL [3], Network Security Services (NSS) [4] and the Bouncy Castle [5]; since 2015, it has been included into Thread [6] as a standard key agreement mechanism for IoT (Internet of Things) applications; and currently J-PAKE is being standardized by ISO/IEC 11770-4 [7].

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Notations

The following notations are used in this document:

o Alice: the assumed identity of the prover in the protocol

o Bob: the assumed identity of the verifier in the protocol

o s: a low-entropy secret shared between Alice and Bob

o a || b: concatenation of a and b

o H: a secure cryptographic hash function

o p: a large prime

o q: a large prime divisor of p-1, i.e., q | p-1

o Zp*: a multiplicative group of integers modulo p

o Gq: a subgroup of Zp* with prime order q

o g: a generator of Gq

o g^x: g raised to the power of x

o a mod b: a modulo b

o Fq: a finite field of q elements where q is a prime

o E(Fq): an elliptic curve defined over Fq

o G: a generator of the subgroup over E(Fq) with prime order n

o n: the order of G

o h: the cofactor of the subgroup generated by G, as defined by h
   = |E(Fq)|/n

o P x [b]: multiplication of a point P with a scalar b over E(Fq)

o P.x: the x coordinate of a point P over E(Fq)

o KDF(a): Key Derivation Function with input a

o HMAC(MacKey, MacData): HMAC function with MacKey as the key and
   MacData as the input data

2. J-PAKE over Finite Field

2.1. Protocol Setup

When implemented over a finite field, J-PAKE uses essentially the
same group parameters as DSA. Choose two large primes, p and q, such
that q | p-1. Let Gq denote a subgroup of Zp* with prime order q, in
which the Decisional Diffie-Hellman problem (DDH) is intractable.
Let g be a generator for Gq. Any non-identity element in Gq can be a
generator. The two communicating parties, Alice and Bob, both agree
on (p, q, g).

Let s be a secret value derived from a low-entropy password shared
between Alice and Bob. The value of s is required to fall within the
range of [1, q-1]. (Note that s must not be 0 for any non-empty
secret.) This range is defined as a necessary condition in [HR08]
for proving the "on-line dictionary attack resistance", since s, s+q,
s+2q, ..., are all considered equivalent values as far as the
protocol specification is concerned. In a practical implementation,
one may obtain s by taking a cryptographic hash of the password and
wrapping the result with respect to modulo q. Alternatively, one may
simply treat the password as an octet string and convert the string
to an integer modulo q by following the method defined in section
2.3.8 of [SEC1]. In either case, one must ensure s is not 0.

2.2. Two-Round Key Exchange

Round 1: Alice selects x1 uniformly at random from [0, q-1] and x2
from [1, q-1]. Similarly, Bob selects x3 uniformly at random from
[0, q-1] and x4 from [1, q-1].

o Alice -> Bob: g1 = g^x1 mod p, g2 = g^x2 mod p and knowledge
   proofs for x1 and x2

o Bob -> Alice: g3 = g^x3 mod p, g4 = g^x4 mod p and knowledge
   proofs for x3 and x4

In this round, the sender must demonstrate the knowledge of the
ephemeral private keys. A suitable technique is to use the Schnorr
NIZK proof [I-D-Schnorr]. [[Q1:: The reference is an accompanying
internet draft submission to IETF and it needs to be updated once it
is accepted by IETF. --Hao]] As an example, suppose one wishes to
prove the knowledge of the exponent for X = g^x mod p. The generated
Schnorr NIZK proof will contain: {UserID, V = g^v mod p, r = v - x *
c mod q} where UserID is the unique identifier for the prover, v is a
number chosen uniformly at random from [0, q-1] and c = H(g || V ||
X || UserID). The "uniqueness" of UserID is defined from the user's
perspective -- for example, if Alice communicates with several
parties, she shall associate a unique identity with each party. Upon
receiving a Schnorr NIZK proof, Alice shall check the prover's UserID
is a valid identity and is different from her own identity. During
the key exchange process using J-PAKE, each party shall ensure that
the other party has been consistently using the same identity
throughout the protocol execution. Details about the Schnorr NIZK
proof, including the generation and the verification procedures, can
be found in [I-D-Schnorr].

When this round finishes, Alice verifies the received knowledge
proofs as specified in [I-D-Schnorr] and also checks that g4 != 1 mod
p. Similarly, Bob verifies the received knowledge proofs and also
checks that g2 != 1 mod p.

Round 2:

o Alice -> Bob: A = (g1*g3*g4)^(x2*s) mod p and a knowledge proof
   for x2*s

o Bob -> Alice: B = (g1*g2*g3)^(x4*s) mod p and a knowledge proof
   for x4*s

In this round, the Schnorr NIZK proof is computed in the same way as
in the previous round except that the generator is different. For
Alice, the generator used is (g1*g3*g4) instead of g; for Bob, the

generator is (g1*g2*g3) instead of g. Since any non-identity element
in Gq can be used as a generator, Alice and Bob just need to ensure
g1*g3*g4 != 1 mod p and g1*g2*g3 != 1 mod p. With overwhelming
probability, these inequalities are statistically guaranteed even
when the user is communicating with an adversary (i.e., in an active
attack). Nonetheless, for absolute guarantee, the receiving party
may want to explicitly check if these inequalities hold, and the cost
of doing that is negligible.

When the second round finishes, Alice and Bob verify the received
knowledge proofs and then compute the key material as follows:

o Alice computes Ka = (B/g4^(x2*s))^x2 mod p

o Bob computes Kb = (A/g2^(x4*s))^x4 mod p

Here Ka = Kb = g^((x1+x3)*x2*x4*s) mod p. Let K denote the same key
material held by both parties. Using K as input, Alice and Bob then
apply a Key Derivation Function (KDF) to derive a common session key
k. If the subsequent secure communication uses a symmetric cipher in
an authenticated mode (say AES-GCM), then one key is sufficient,
i.e., k = KDF(K). Otherwise, the session key should comprise an
encryption key (for confidentiality) and a MAC key (for integrity),
i.e., k = k_enc || k_mac, where k_enc = KDF(K || "JPAKE_ENC") and
k_mac = KDF(K || "JPAKE_MAC"). The exact choice of the KDF is left
to specific applications to define. (In many cases, the KDF may
simply be a cryptographic hash function, e.g., SHA-256.)

2.3. Computational Cost

The computational cost is estimated based on counting the number of
modular exponentiations since they are the predominant cost factors.
Note that it takes one exponentiation to generate a Schnorr NIZK
proof and two to verify it [I-D-Schnorr]. For Alice, she has to
perform 8 exponentiations in the first round, 4 in the second round,
and 2 in the final computation of the session key. Hence, that is 14
modular exponentiations in total. Based on the symmetry, the
computational cost for Bob is exactly the same.

3. J-PAKE over Elliptic Curve

3.1. Protocol Setup

The J-PAKE protocol works basically the same in the elliptic curve
(EC) setting, except that the underlying multiplicative group over a
finite field is replaced by an additive group over an elliptic curve.
Nonetheless, the EC version of J-PAKE is specified here for
completeness.

When implemented over an elliptic curve, J-PAKE uses essentially the same EC parameters as ECDSA. Let E(Fq) be an elliptic curve defined over a finite field Fq where q is a large prime. Let G be a generator for the subgroup over E(Fq) of prime order n.

As before, let s denote the shared secret between Alice and Bob. The value of s is required to fall within [1, n-1].

3.2. Two-Round Key Exchange

Round 1: Alice selects x1 and x2 uniformly at random from [1, n-1]. Similarly, Bob selects x3 and x4 uniformly at random from [1, n-1].

o Alice -> Bob: G1 = G x [x1], G2 = G x [x2] and knowledge proofs
   for x1 and x2

o Bob -> Alice: G3 = G x [x3], G4 = G x [x4] and knowledge proofs
   for x3 and x4

When this round finishes, Alice and Bob verify the received knowledge proofs as specified in [I-D-Schnorr].

Round 2:

o Alice -> Bob: A = (G1 + G3 + G4) x [x2*s] and a knowledge proof
   for x2*s

o Bob -> Alice: B = (G1 + G2 + G3) x [x4*s] and a knowledge proof
   for x4*s

When the second round finishes, Alice and Bob verify the received knowledge proofs and then compute the key material as follows:

o Alice computes Ka = (B - (G4 x [x2*s])) x [x2]

o Bob computes Kb = (A - (G2 x [x4*s])) x [x4]

Here Ka = Kb = G x [(x1+x3)*(x2*x4*s)]. Let K denote the same key material held by both parties. Using K as input, Alice and Bob then apply a Key Derivation Function (KDF) to derive a common session key k. Note that K is a point on E(Fq), consisting of the x and y coordinates. In practice, it is sufficient to use only the x coordinate as the input to KDF to derive the session key. The x coordinate, which is a field element in Fq, can be converted to an octet string, by following the method defined in section 2.3.3 in [SEC1].

[3.3](). Computational Cost

In the EC setting, the computational cost of J-PAKE is estimated
based on counting the number of scalar multiplications over the
elliptic curve. Note that it takes one multiplication to generate a
Schnorr NIZK proof and one to verify it [[I-D-Schnorr]()]. For Alice,
she has to perform 6 multiplications in the first round, 3 in the
second round, and 2 in the final computation of the session key.
Hence, that is 11 multiplications in total. Based on the symmetry,
the computational cost for Bob is exactly the same.

[4](). Three-Pass Variant

The two-round J-PAKE protocol is completely symmetric, which
significantly simplifies the security analysis. In practice, one
party normally initiates the communication and the other party
responds. In that case, the protocol will be completed in three
passes instead of two rounds. The two-round J-PAKE protocol can be
trivially changed to three passes without losing security. Take the
finite field setting as an example and assume Alice initiates the key
exchange. The three-pass variant works as follows:

1. Alice -> Bob: $g1 = g^{x1} \mod p$, $g2 = g^{x2} \mod p$, knowledge proofs
    for x1 and x2.

2. Bob -> Alice: $g3 = g^{x3} \mod p$, $g4 = g^{x4} \mod p$, $B =
    (g1*g2*g3)^{(x4*s)} \mod p$, knowledge proofs for x3, x4, and x4*s.

3. Alice -> Bob: $A = (g1*g3*g4)^{(x2*s)} \mod p$ and a knowledge proof
    for x2*s.

Both parties compute the session keys in exactly the same way as
before.

[5](). Key Confirmation

The two-round J-PAKE protocol (or the three-pass variant) provides
cryptographic guarantee that only the authenticated party who used
the same password at the other end is able to compute the same
session key. So far the authentication is only implicit.

For achieving explicit authentication, an additional key confirmation
procedure shall be performed. This is to ensure that both parties
have actually obtained the same session key.

There are several explicit key confirmation methods available. They
are generically applicable to all key exchange protocols, not just
J-PAKE. In general, it is recommended to use a different key from

the session key for key confirmation, say using k' = KDF(K ||
"JPAKE_KC"). The advantage of using a different key for key
confirmation is that the session key remains indistinguishable from
random after the key confirmation process (although this perceived
advantage is actually subtle and only theoretical). Two key-
confirmation methods are presented here.

The first method is based on the one used in the SPEKE protocol
[Jab96]. Suppose Alice initiates the key confirmation. Alice sends
to Bob H(H(k')), which Bob will verify. If the verification is
successful, Bob sends back to Alice H(k'), which Alice will verify.
This key confirmation procedure needs to be completed in two rounds,
as shown below.

1. Alice -> Bob: H(H(k'))

2. Bob -> Alice: H(k')

The second method is based on the unilateral key confirmation scheme
specified in NIST SP 800-56A Revision 1 [BJS07]. Alice and Bob send
to each other a MAC tag, which they will verify accordingly. This
key confirmation procedure can be completed in one round.

In the finite field setting it works as follows.

o Alice -> Bob: MacTagAlice = HMAC(k', "KC_1_U" || Alice || Bob ||
   g1 || g2 || g3 || g4)

o Bob -> Alice: MacTagBob = HMAC(k', "KC_1_U" || Bob || Alice ||
   g3 || g4 || g1 || g2)

In the EC setting it works basically the same. Let G1.x, G2.x, G3.x
and G4.x be the x coordinates of G1, G2, G3 and G4 respectively. It
is sufficient (and simpler) to include only the x coordinates in the
HMAC function. Hence, the key confirmation works as follows.

o Alice -> Bob: MacTagAlice = HMAC(k', "KC_1_U" || Alice || Bob ||
   G1.x || G2.x || G3.x || G4.x)

o Bob -> Alice: MacTagBob = HMAC(k', "KC_1_U" || Bob || Alice ||
   G3.x || G4.x || G1.x || G2.x)

The second method assumes an additional secure MAC function (HMAC)
and is slightly more complex than the first method. However, it can
be completed within one round and it preserves the overall symmetry
of the protocol implementation. For this reason, the second method
is recommended.

[6](#). **Security Considerations**

   A PAKE protocol is designed to provide two functions in one protocol
   execution. The first one is to provide zero-knowledge authentication
   of a password. It is called "zero knowledge" because at the end of
   the protocol, the two communicating parties will learn nothing more
   than one bit information: whether the passwords supplied at two ends
   are equal. Therefore, a PAKE protocol is naturally resistant against
   phishing attacks. The second function is to provide session key
   establishment if the two passwords are equal. The session key will
   be used to protect the confidentiality and integrity of the
   subsequent communication.

   More concretely, a secure PAKE protocol shall satisfy the following
   security requirements [[HR10](#)].

   1. Off-line dictionary attack resistance: It does not leak any
       information that allows a passive/active attacker to perform off-
       line exhaustive search of the password.

   2. Forward secrecy: It produces session keys that remain secure even
       when the password is later disclosed.

   3. Known-key security: It prevents a disclosed session key from
       affecting the security of other sessions.

   4. On-line dictionary attack resistance: It limits an active
       attacker to test only one password per protocol execution.

   First, a PAKE protocol must resist off-line dictionary attacks. A
   password is inherently weak. Typically, it has only about 20-30 bits
   entropy. This level of security is subject to exhaustive search.
   Therefore, in the PAKE protocol, the communication must not reveal
   any data that allows an attacker to learn the password through off-
   line exhaustive search.

   Second, a PAKE protocol must provide forward secrecy. The key
   exchange is authenticated based on a shared password. However, there
   is no guarantee on the long-term secrecy of the password. A secure
   PAKE scheme shall protect past session keys even when the password is
   later disclosed. This property also implies that if an attacker
   knows the password but only passively observes the key exchange, he
   cannot learn the session key.

   Third, a PAKE protocol must provide known key security. A session
   key lasts throughout the session. An exposed session key must not
   cause any global impact on the system, affecting the security of
   other sessions.

Finally, a PAKE protocol must resist on-line dictionary attacks. If the attacker is directly engaging in the key exchange, there is no way to prevent such an attacker trying a random guess of the password. However, a secure PAKE scheme should mitigate the effect of the on-line attack to the minimum. In the best case, the attacker can only guess exactly one password per impersonation attempt. Consecutively failed attempts can be easily detected and the subsequent attempts can be thwarted accordingly.

It has been proven in [HR10] that J-PAKE satisfies all of the four requirements based on the assumptions that the Decisional Diffie-Hellman problem is intractable and the underlying Schnorr NIZK proof is secure. An independent study that proves security of J-PAKE in a model with algebraic adversaries and random oracles can be found in [ABM15]. By comparison, it has been known that EKE has the problem of leaking partial information about the password to a passive attacker, hence not satisfying the first requirement [Jas96]. For SPEKE and SRP-6, an attacker may be able to test more than one password in one on-line dictionary attack (see [Zha04] and [Hao10]), hence they do not satisfy the fourth requirement in the strict theoretical sense. Furthermore, SPEKE is found vulnerable to an impersonation attack and a key-malleability attack [HS14]. These two attacks affect the SPEKE protocol specified in Jablon's original 1996 paper [Jab96] as well in the latest IEEE P1363.2 standard draft D26 and the latest published ISO/IEC 11770-4:2006 standard. As a result, the specification of SPEKE in ISO/IEC 11770-4 is being revised to address the identified problems.

7. IANA Considerations

This document has no actions for IANA.

8. Acknowledgements

9. References

9.1. 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,
           <http://www.rfc-editor.org/info/rfc2119>.

[RFC5054]   Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin,
            "Using the Secure Remote Password (SRP) Protocol for TLS
            Authentication", RFC 5054, DOI 10.17487/RFC5054, November
            2007, <http://www.rfc-editor.org/info/rfc5054>.

[SEC1]   "Standards for Efficient Cryptography. SEC 1: Elliptic
            Curve Cryptography", SECG SEC1-v2, May 2004,
            <http://www.secg.org/sec1-v2.pdf>.

[ABM15]   Abdalla, M., Benhamouda, F., and P. MacKenzie, "Security
            of the J-PAKE Password-Authenticated Key Exchange
            Protocol", IEEE Symposium on Security and Privacy, May
            2015.

[BM92]   Bellovin, S. and M. Merrit, "Encrypted Key Exchange:
            Password-based Protocols Secure against Dictionary
            Attacks", IEEE Symposium on Security and Privacy, May
            1992.

[HR08]   Hao, F. and P. Ryan, "Password Authenticated Key Exchange
            by Juggling", 16th Workshop on Security Protocols
            (SPW'08), May 2008.

[HR10]   Hao, F. and P. Ryan, "J-PAKE: Authenticated Key Exchange
            Without PKI", Springer Transactions on Computational
            Science XI, 2010.

[HS14]   Hao, F. and S. Shahandashti, "The SPEKE Protocol
            Revisited", Security Standardisation Research, December
            2014.

[Jab96]   Jablon, D., "Strong Password-Only Authenticated Key
            Exchange", ACM Computer Communications Review, October
            1996.

[Wu98]   Wu, T., "The Secure Remote Password protocol", Symposimum
            on Network and Distributed System Security, March 1998.

[I-D-Schnorr]
            Hao, F., "Schnorr NIZK proof: Non-interactive Zero
            Knowledge Proof for Discrete Logarithm", Internet Draft
            submitted to IETF, 2013.

9.2. Informative References

[BJS07]   Barker, E., Johnson, D., and M. Smid, "Recommendation for
           Pair-Wise Key Establishment Schemes Using Discrete
           Logarithm Cryptography (Revised)", NIST Special
           Publication 800-56A, March 2007,
           <http://csrc.nist.gov/publications/nistpubs/800-56A/
           SP800-56A_Revision1_Mar08-2007.pdf>.

[Jas96]   Jaspan, B., "Dual-Workfactor Encrypted Key Exchange:
           Efficiently Preventing Password Chaining and Dictionary
           Attacks", USENIX Symphosium on Security, July 1996.

[Zha04]   Zhang, M., "Analysis of The SPEKE Password-Authenticated
           Key Exchange Protocol", IEEE Communications Letters,
           January 2004.

[Hao10]   Hao, F., "On Small Subgroup Non-Confinement Attacks",
            IEEE conference on Computer and Information Technology,
           2010.

9.3. URIs

[1] https://wiki.mozilla.org/Services/Sync/SyncKey/J-PAKE

[2] https://www.palemoon.org/sync/

[3] http://boinc.berkeley.edu/android-boinc/libssl/crypto/jpake/

[4] https://dxr.mozilla.org/mozilla-
    central/source/security/nss/lib/freebl/jpake.c

[5] https://www.bouncycastle.org/docs/docs1.5on/org/bouncycastle/cryp
    to/agreement/jpake/package-summary.html

[6] http://threadgroup.org/Portals/0/documents/whitepapers/
    Thread%20Commissioning%20white%20paper_v2_public.pdf

[7] http://www.iso.org/iso/home/store/catalogue_tc/
    catalogue_detail.htm?csnumber=67933

Author's Address

Feng Hao (editor)
Newcastle University (UK)
Claremont Tower, School of Computing Science, Newcastle University
Newcastle Upon Tyne
United Kingdom

Phone: +44 (0)191-208-6384
EMail: feng.hao@ncl.ac.uk