Independent Submission
Request for Comments: 8236
Category: Informational
ISSN: 2070-1721

F. Hao, Ed.
Newcastle University (UK)

September 2017

J-PAKE: Password-Authenticated Key Exchange by Juggling

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 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 a candidate 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 http://www.rfc-editor.org/info/rfc8236.

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 Encrypted Key Exchange (EKE), was proposed by Steven Bellovin and Michael Merrit in 1992 [BM92]. Other well-known PAKE protocols include Simple Password Exponential Key Exchange (SPEKE) by David Jablon in 1996 [Jab96] and Secure Remote Password (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 [RFC8235]. Third, J-PAKE 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. Unlike SPAKE2 [AP05] and SESPAKE [SOAA15], it does not require a trusted setup (i.e., the so-called common reference model) to define a pair of generators whose discrete logarithm must be unknown. Finally, J-PAKE has been used in real-world applications at a relatively large scale, e.g., Firefox sync [MOZILLA], Pale moon sync [PALEMOON], and Google Nest products [ABM15]. It has been included into widely distributed open source libraries such as OpenSSL [BOINC], Network Security Services (NSS) [MOZILLA_NSS], and the Bouncy Castle [BOUNCY]. Since 2015, J-PAKE has been included in Thread [THREAD] as a standard key agreement mechanism for IoT (Internet of Things) applications, and also included in ISO/IEC 11770-4:2017 [ISO.11770-4].

1.1. Requirements Language

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.

1.2. Notation

The following notation is used in this document:

- Alice: the assumed identity of the prover in the protocol
- Bob: the assumed identity of the verifier in the protocol
- s: a low-entropy secret shared between Alice and Bob
- a | b: a divides b
- a || b: concatenation of a and b
- [a, b]: the interval of integers between and including a and b
- H: a secure cryptographic hash function
- p: a large prime
- q: a large prime divisor of p-1, i.e., q | p-1
- Zp*: a multiplicative group of integers modulo p

- Gq: a subgroup of Zp* with prime order q
- g: a generator of Gq
- g^d: g raised to the power of d
- a mod b: a modulo b
- Fp: a finite field of p elements, where p is a prime
- E(Fp): an elliptic curve defined over Fp
- G: a generator of the subgroup over E(Fp) with prime order n
- n: the order of G
- h: the cofactor of the subgroup generated by G, which is equal to the order of the elliptic curve divided by n
- P x [b]: multiplication of a point P with a scalar b over E(Fp)
- KDF(a): Key Derivation Function with input a
- MAC(MacKey, MacData): MAC 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 may use the same group parameters as DSA [FIPS186-4]. Let p and q be two large primes such that q | p-1. Let Gq denote a subgroup of Zp* with prime order q. 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), which can be hard-wired in the software code. They can also use the method in NIST FIPS 186-4, Appendix A [FIPS186-4] to generate (p, q, g). Here, DSA group parameters are used only as an example. Other multiplicative groups suitable for cryptography can also be used for the implementation, e.g., groups defined in [RFC4419]. A group setting that provides 128-bit security or above is recommended. The security proof of J-PAKE depends on the Decisional Diffie-Hellman (DDH) problem being intractable in the considered group.

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 equal to 0 modulo q.

2.2. Two-Round Key Exchange

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

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

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

In this round, the sender must send zero knowledge proofs to demonstrate the knowledge of the ephemeral private keys. A suitable technique is to use the Schnorr NIZK proof [RFC8235]. As an example, suppose one wishes to prove the knowledge of the exponent for D = g^d mod p. The generated Schnorr NIZK proof will contain: {UserID, V = g^v mod p, r = v - d * 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 || D || 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 [RFC8235].

When this round finishes, Alice verifies the received ZKPs as specified in [RFC8235] and also checks that g4 != 1 mod p. Similarly, Bob verifies the received ZKPs and also checks that g2 != 1 mod p. If any of these checks fails, this session should be aborted.

Round 2:

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

- Bob -> Alice: B = (g1*g2*g3)^(x4*s) mod p and a ZKP 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 shall explicitly check if these inequalities hold, and abort the session in case such a check fails.

When the second round finishes, Alice and Bob verify the received ZKPs. If the verification fails, the session is aborted. Otherwise, the two parties compute the common key material as follows:

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

- 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.

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 [RFC8235]. For Alice, she needs 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 may use the same EC parameters as ECDSA [FIPS186-4]. The FIPS 186-4 standard [FIPS186-4] defines three types of curves suitable for ECDSA: pseudorandom curves over prime fields, pseudorandom curves over binary fields, and special curves over binary fields called Koblitz curves or anomalous binary curves. All these curves that are suitable for ECDSA can also be used to implement J-PAKE. However, for illustration purposes, only curves over prime fields are described in this document. Typically, such curves include NIST P-256, P-384, and P-521. When choosing a curve, a level of 128-bit security or above is recommended. Let E(Fp) be an elliptic curve defined over a finite field Fp, where p is a large prime. Let G be a generator for the subgroup over E(Fp) of prime order n. Here, the NIST curves are used only as an example. Other secure curves such as Curve25519 are also suitable for implementation. The security proof of J-PAKE relies on the assumption that the DDH problem is intractable in the considered group.

As before, let s denote the shared secret between Alice and Bob. The value of s falls within [1, n-1]. In particular, note that s MUST not be equal to 0 mod n.

3.2. Two-Round Key Exchange

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

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

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

When this round finishes, Alice and Bob verify the received ZKPs as specified in [RFC8235]. As an example, to prove the knowledge of the discrete logarithm of D = G x [d] with respect to the base point G, the ZKP contains: {UserID, V = G x [v], r = v - d * c mod n}, where UserID is the unique identifier for the prover, v is a number chosen uniformly at random from [1, n-1] and c = H(G || V || D || UserID).

The verifier shall check the prover's UserID is a valid identity and is different from its own identity. If the verification of the ZKP fails, the session is aborted.

Round 2:

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

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

When the second round finishes, Alice and Bob verify the received ZKPs. The ZKPs are computed in the same way as in the previous round except that the generator is different. For Alice, the new generator is G1 + G3 + G4; for Bob, it is G1 + G2 + G3. Alice and Bob shall check that these new generators are not points at infinity. If any of these checks fails, the session is aborted. Otherwise, the two parties compute the common key material as follows:

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

- 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.

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 [RFC8235]. 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, ZKPs 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, ZKPs for x3, x4, and x4*s.

3. Alice -> Bob: A = (g1*g3*g4)^(x2*s) mod p and a ZKP 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. The key confirmation is also implicit [Stinson06]. The two parties may use the derived key straight away to start secure communication by encrypting messages in an authenticated mode. Only the party with the same derived session key will be able to decrypt and read those messages.

For achieving explicit authentication, an additional key confirmation procedure should be performed. This provides explicit assurance that the other party has actually derived the same key. In this case, the key confirmation is explicit [Stinson06].

In J-PAKE, explicit key confirmation is recommended whenever the network bandwidth allows it. It has the benefit of providing explicit and immediate confirmation if the two parties have derived the same key and hence are authenticated to each other. This allows a practical implementation of J-PAKE to effectively detect online dictionary attacks (if any), and stop them accordingly by setting a threshold for the consecutively failed connection attempts.

To achieve explicit key confirmation, there are several methods available. They are generically applicable to all key exchange protocols, not just J-PAKE. In general, it is recommended that a different key from the session key be used for key confirmation -- say, 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. (However, this perceived advantage is actually subtle and only theoretical.) Two explicit 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 above procedure requires two rounds instead of one, because the second message depends on the first. If both parties attempt to send the first message at the same time without an agreed order, they cannot tell if the message that they receive is a genuine challenge or a replayed message, and consequently may enter a deadlock.

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.

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

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

In the EC setting, the key confirmation works basically the same.

- Alice -> Bob: MacTagAlice = MAC(k', "KC_1_U" || Alice || Bob || G1 || G2 || G3 || G4)

- Bob -> Alice: MacTagBob = MAC(k', "KC_1_U" || Bob || Alice || G3 || G4 || G1 || G2)

The second method assumes an additional secure MAC function (e.g., one may use 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. Offline dictionary attack resistance: It does not leak any information that allows a passive/active attacker to perform offline 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. Online dictionary attack resistance: It limits an active attacker to test only one password per protocol execution.

First, a PAKE protocol must resist offline 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 offline 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 online 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 minimize the effect of the online attack. 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 shall be thwarted accordingly. It is recommended that the false authentication counter be handled in such a way that any error (which causes the session to fail during the key exchange or key confirmation) leads to incrementing the false authentication counter.

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 online 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 D26 draft of IEEE P1363.2 and the ISO/IEC 11770-4:2006 standard. As a result, the specification of SPEKE in ISO/IEC 11770-4:2006 has been revised to address the identified problems.

7. IANA Considerations

This document does not require any IANA actions.

8. References

8.1. Normative References

[ABM15] Abdalla, M., Benhamouda, F., and P. MacKenzie, "Security of the J-PAKE Password-Authenticated Key Exchange Protocol", 2015 IEEE Symposium on Security and Privacy, DOI 10.1109/sp.2015.41, May 2015.

[BM92] Bellovin, S. and M. Merrit, "Encrypted Key Exchange: Password-based Protocols Secure against Dictionary Attacks", IEEE Symposium on Security and Privacy, DOI 10.1109/risp.1992.213269, May 1992.

[HR08] Hao, F. and P. Ryan, "Password Authenticated Key Exchange by Juggling", Lecture Notes in Computer Science, pp. 159-171, from 16th Security Protocols Workshop (SPW '08), DOI 10.1007/978-3-642-22137-8_23, 2011.

[HR10] Hao, F. and P. Ryan, "J-PAKE: Authenticated Key Exchange Without PKI", Transactions on Computational Science XI, pp. 192-206, DOI 10.1007/978-3-642-17697-5_10, 2010.

[HS14] Hao, F. and S. Shahandashti, "The SPEKE Protocol Revisited", Security Standardisation Research, pp. 26-38, DOI 10.1007/978-3-319-14054-4_2, December 2014.

[Jab96] Jablon, D., "Strong Password-Only Authenticated Key Exchange", ACM SIGCOMM Computer Communication Review, Vol. 26, pp. 5-26, DOI 10.1145/242896.242897, October 1996.

[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>.

[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, <https://www.rfc-editor.org/info/rfc5054>.

[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>.

[RFC8235] Hao, F., Ed., "Schnorr Non-interactive Zero Knowledge Proof", RFC 8235, DOI 10.17487/RFC8235, September 2017, <https://www.rfc-editor.org/info/rfc8235>.

[SEC1] "Standards for Efficient Cryptography. SEC 1: Elliptic Curve Cryptography", SECG SEC1-v2, May 2009, <http://www.secg.org/sec1-v2.pdf>.

[Stinson06] Stinson, D., "Cryptography: Theory and Practice", 3rd Edition, CRC, 2006.

[Wu98] Wu, T., "The Secure Remote Password Protocol", Internet Society Symposium on Network and Distributed System Security, March 1998.

8.2. Informative References

[AP05] Abdalla, M. and D. Pointcheval, "Simple Password-Based Encrypted Key Exchange Protocols", Topics in Cryptology CT-RSA, DOI 10.1007/978-3-540-30574-3_14, 2005.

[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>.

[BOINC] BOINC, "Index of /android-boinc/libssl/crypto/jpake", February 2011, <http://boinc.berkeley.edu/android-boinc/libssl/crypto/jpake/>.

[BOUNCY] Bouncy Castle Cryptography Library, "org.bouncycastle.crypto.agreement.jpake (Bouncy Castle Library 1.57 API Specification)", May 2017, <https://www.bouncycastle.org/docs/docs1.5on/org/bouncycastle/crypto/agreement/jpake/package-summary.html>.

[FIPS186-4] National Institute of Standards and Technology, "Digital Signature Standard (DSS)", FIPS PUB 186-4, DOI 10.6028/NIST.FIPS.186-4, July 2013, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[Hao10] Hao, F., "On Small Subgroup Non-Confinement Attacks", IEEE Conference on Computer and Information Technology, DOI 10.1109/CIT.2010.187, 2010.

[ISO.11770-4] ISO/IEC, "Information technology -- Security techniques -- Key management -- Part 4: Mechanisms based on weak secrets", (under development), July 2017, <https://www.iso.org/standard/67933.html>.

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

[MOZILLA] Mozilla Wiki, "Services/KeyExchange", August 2011, <https://wiki.mozilla.org/index.php?title=Services/KeyExchange&oldid=343704>.

[MOZILLA_NSS]
Mozilla Central, "jpake.c - DXR", August 2016, <https://dxr.mozilla.org/mozilla-central/source/security/nss/lib/freebl/jpake.c>.

[PALEMOON] Moonchild Productions, "Pale Moon Sync", <https://www.palemoon.org/sync/>.

[RFC4419] Friedl, M., Provos, N., and W. Simpson, "Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol", RFC 4419, DOI 10.17487/RFC4419, March 2006, <https://www.rfc-editor.org/info/rfc4419>.

[SOAA15] Smyshlyaev, S., Oshkin, I., Alekseev, E., and L. Ahmetzyanova, "On the Security of One Password Authenticated Key Exchange Protocol", 2015, <http://eprint.iacr.org/2015/1237.pdf>.

[THREAD] Thread, "Thread Commissioning", White Paper, July 2015, <https://portal.threadgroup.org/DesktopModules/Inventures_Document/FileDownload.aspx?ContentID=658>.

[Zha04] Zhang, M., "Analysis of the SPEKE Password-Authenticated Key Exchange Protocol", IEEE Communications Letters, Vol. 8, pp. 63-65, DOI 10.1109/lcomm.2003.822506, January 2004.

Acknowledgements

The editor would like to thank Dylan Clarke, Siamak Shahandashti, Robert Cragie, Stanislav Smyshlyaev, and Russ Housley for many useful comments. This work is supported by EPSRC First Grant (EP/J011541/1) and ERC Starting Grant (No. 306994).

Author’s Address

Feng Hao (editor)
Newcastle University (UK)
Urban Sciences Building, School of Computing, Newcastle University
Newcastle Upon Tyne
United Kingdom

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