Newcastle University

ePrints

Hao Feng, Metere Roberto, Shahandashti Siamak, Dong Changyu.
Analysing and Patching SPEKE in ISO/IEC.
***IEEE Transactions on Information Forensics and Security* 2018**
DOI: https://doi.org/10.1109/TIFS.2018.2832984

Copyright:

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

DOI link to article:

https://doi.org/10.1109/TIFS.2018.2832984

Date deposited:

11/05/2018

Newcastle University ePrints - eprint.ncl.ac.uk

Analysing and Patching SPEKE in ISO/IEC

Feng Hao, Roberto Metere, Siamak F. Shahandashti and Changyu Dong

***Abstract*—Simple Password Exponential Key Exchange (SPEKE) is a well-known Password Authenticated Key Exchange (PAKE) protocol that has been used in Blackberry phones for secure messaging and Entrust's TruePass end-to-end web products. It has also been included into international standards such as ISO/IEC 11770-4 and IEEE P1363.2. In this paper, we analyse the SPEKE protocol as specified in the ISO/IEC and IEEE standards. We identify that the protocol is vulnerable to two new attacks: an impersonation attack that allows an attacker to impersonate a user without knowing the password by launching two parallel sessions with the victim, and a key-malleability attack that allows a man-in-the-middle (MITM) to manipulate the session key without being detected by the end users. Both attacks have been acknowledged by the technical committee of ISO/IEC SC 27, and ISO/IEC 11770-4 revised as a result. We propose a patched SPEKE called P-SPEKE and present a formal analysis in the Applied Pi Calculus using ProVerif to show that the proposed patch prevents both attacks. The proposed patch has been included into the latest revision of ISO/IEC 11770-4 published in 2017.**

***Index Terms*—password-based authenticated key exchange, formal methods, key agreement**

I. Introduction

A password-authenticated key exchange (PAKE) protocol aims to establish a high-entropy session key for secure communication between two parties based on a low-entropy secret password known to both without relying on any external trusted parties. The idea of bootstrapping a high-entropy secret key based on a low-entropy secret password is counter-intuitive, and for a long time had been thought impossible until the seminal work by Bellovin and Merrit who proposed the first PAKE solution called Encrypted Key Exchange (EKE) [9]. Since then, research on PAKE has become a thriving field: many PAKE protocols have been proposed, and some have been included into international standards [22], [23].

However, the original EKE protocol was found to suffer from several limitations, of which the most significant one was the leakage about the password [26]. Motivated by addressing the limitations, Jablon proposed another PAKE solution called the simple password exponential key exchange (SPEKE) in 1996 [25]. SPEKE proves to be a more practical protocol than EKE since it does not have the same password leakage problem as in EKE. Although researchers raised concerns on some other aspects of SPEKE [31], [33] such as the possibility for an online attacker to test multiple passwords in one go, no major flaws have been reported. Over the years, SPEKE has been used in several commercial applications: for example, the secure messaging on Blackberry phones [11] and Entrust's TruePass end-to-end web products [16]. SPEKE has also been included into the international standards such as IEEE P1363.2 [22] and ISO/IEC 11770-4 [24].

Given the wide usage of SPEKE in practical applications and its inclusion in standards, we believe a thorough analysis of SPEKE is both necessary and important. In this paper, we revisit SPEKE and its variants specified in the original paper [25], the IEEE 1363.2 [22] and ISO/IEC 11770-4 [23] standards. We first observe that the original SPEKE protocol is subtly different from those defined in the standards. The difference has significant security implications, which are not explained in the standards.

During the review, we have identified several issues that have not been reported before. In particular, we find two new attacks on SPEKE: namely, an impersonation attack and a key-malleability attack. The first attack allows an attacker to impersonate a user without knowing the password by launching two parallel sessions with the victim. The second attack allows an attacker to manipulate the session key without being detected. To address the identified problems, we propose a patched SPEKE, called P-SPEKE, which prevents both attacks by including the user identities in the key derivation function without altering the symmetry of the original SPEKE protocol. We build a formal model in the Applied Pi Calculus using ProVerif and apply it to formally analyse P-SPEKE. Our analysis confirms that the proposed patch is immune to the attacks. Finally, we identify an efficiency problem with the key confirmation procedure specified in both the ISO/IEC and IEEE standards and accordingly propose an improved procedure.

Our contributions are summarized below.

- We discover two new attacks on SPEKE: an impersonation attack and a key-malleability attack. We explain how the attacks affect the SPEKE variants specified in the IEEE P1363.2 and ISO/IEC 11770-4 standards.
- We propose a patched SPEKE, called P-SPEKE, which prevents both attacks without altering the symmetry of the SPEKE protocol. Furthermore, we propose an improved key confirmation procedure, which is more round-efficient than the one defined in the standards.
- We build a formal model in the Applied Pi Calculus and verify the proposed patch by using ProVerif. Our formal analysis confirms that the proposed patch is immune against the identified attacks.

This paper extends the earlier conference paper [20] by adding a formal analysis of the patched SPEKE protocol, and details of how the proposed patch was accepted and included into the revision of ISO/IEC 11770-4. The two

Feng Hao, Roberto Metere and Changyu Dong are with School of Computing, Newcastle University, United Kingdom. E-mail: {feng.hao, r.metere2, changyu.dong}@ncl.ac.uk. Siamak F. Shahandashti is with the Department of Computer Science, University of York, United Kindom. Email: siamak.shahandashti@york.ac.uk. This work is supported by ERC Starting Grant, No. 306994. Dong is supported by EPSRC EP/M013561/2.

attacks and the efficiency issues, initially reported in [20], were discussed and acknowledged by the technical committee of ISO/IEC SC 27, Working Group 2. Accordingly, the ISO/IEC 11770-4 standard was revised. The latest revision ISO/IEC 11770-4:2017, incorporating our proposed patch and the improved key confirmation procedure, was formally published in November 2017 [24].

II. Background

A. Password Authenticated Key Exchange

Since the invention of the first PAKE solution in [9], many PAKE protocols have been proposed, among which only a few have been actually used in practice. Notable examples of PAKE that have been deployed in practical applications include EKE [9], SPEKE [25] and J-PAKE [19]. These three protocols happen to represent three different ways of constructing a PAKE. EKE works by using the shared password as a symmetric key to encrypt Diffie-Hellman key exchange items. Variants of EKE (e.g., SPAKE2 [5]) often differ only in how the symmetric cipher is instantiated. SPEKE works by using the shared password to derive a secret group generator for performing Diffie-Hellman key exchange. There are variants of SPEKE, such as Dragonfly [21] and PACE [10], which use different methods to derive the secret generator from the password. J-PAKE works by using the password to randomize the secret exponents in order to achieve a cancellation effect. A distinctive feature of J-PAKE as compared to the other two is its use of Zero Knowledge Proof (ZKP) to enforce participants to follow the protocol specification. By comparison, the use of ZKP is considered incompatible with the design of EKE and SPEKE.

A PAKE protocol serves to provide two functions: authentication and key exchange. The former is based on the knowledge of a password. If the two passwords match at both ends, a session key will be created for the subsequent secure communication. In the following, we review some common properties of a secure password authenticated key exchange protocols based on [9], [19], [25]; we also refer the reader to classic definitions of authentication from Lowe [28]. Formal treatments of PAKE, based on authenticated key exchange models proposed by Bellare and Rogaway in 1993 [8], can be found in [4], [7], [17], [27].

Correctness. In the setting of key-exchange protocols, the protocol is correct if it gives both authentication and key distribution in presence of honest parties [32]. This is a basic and necessary step in a formal model to prove that without influence of attackers, honest parties should always complete the protocol as expected.

Secrecy of the pre-shared password. This property requires that the execution of the protocol must not reveal any data that would allow an attacker to learn the password through off-line exhaustive search. If the attacker is directly engaging in the key exchange, he should be limited to guess only one password per protocol execution.

Implicit key authentication. Assume the key exchange protocol is run between Alice and Bob. The protocol is said to provide implicit key authentication if Alice is assured that no one other than Bob can compute the session key [30].

Explicit key authentication. Explicit authentication can only be achieved with a confirmation phase [30]. This property requires that the entities have actually computed the *same* key. It completes and strengthens the implicit key authentication; in fact, if the two participants are the sole entities who can learn the session key *and* they have actually computed the key, the successive communication shall be secure.

Weak and strong entity authentication. *Weak* or *strong* entity authentication respectively correspond to the *weak agreement* and *injective agreement* properties of Lowe [28]. A protocol achieves weak authentication if a participant believes she is speaking with another participant, and the other participant indeed started an authentication process with her. Even though this may seem a sufficient property for mutual authentication, it is not. In fact, nothing can be said about the problem where the party is tricked to communicate with some replayed session of the other party. With strong authentication, the additional property of agreeing with both the session and the session key is required. Strong entity authentication ensures that replay attacks and man-in-the-middle attacks are prevented.

Perfect forward secrecy. Perfect forward secrecy (PFS) ensures that the confidentiality of past session keys is preserved even when the long term secret, i.e., the password, is disclosed. This property implies that an attacker who knows the password still cannot learn the session key if he only *passively* eavesdrops the key exchange process.

B. The original SPEKE

The original specification of the SPEKE protocol in Jablon's 1996 paper [25] is as follows. Participants agreed on a group $\mathbb{G}$ of safe prime order $p = 2q+1$ where q is also a prime. The SPEKE protocol operates in the subgroup of $\mathbb{G}$ of prime order q where the discrete logarithm problem is assumed to be hard. Two remote parties, Alice and Bob, share a common secret password s from which they apply a function $f(\cdot)$ to compute the group generator: $g = f(s) = s^2 \bmod p$. Unless specified otherwise, all modular operations in the rest of the paper are performed with respect to the modulus p. We will omit "$\bmod p$" in the notation for simplicity.

The SPEKE protocol runs in two phases: the *key-exchange phase* and the *key-confirmation phase*, as illustrated in Figure 1. In the first phase, Alice chooses a secret value x uniformly at random in $\mathbb{Z}_q^\star = \{1, \ldots, q-1\}$, and sends g^x to Bob. Similarly, Bob chooses a secret value y uniformly at random in $\mathbb{Z}_q^\star$, and sends g^y to Alice. Upon receiving g^y, Alice verifies that its value is between 2 and $p-2$. This is to prevent the small subgroup confinement attack. Subsequently, Alice computes a session key $k = H((g^y)^x) = H(g^{xy})$ where H is a cryptographic hash function (used as a key derivation function here). Similarly Bob verifies g^x belongs to $\{2, \ldots, p-2\}$ and then

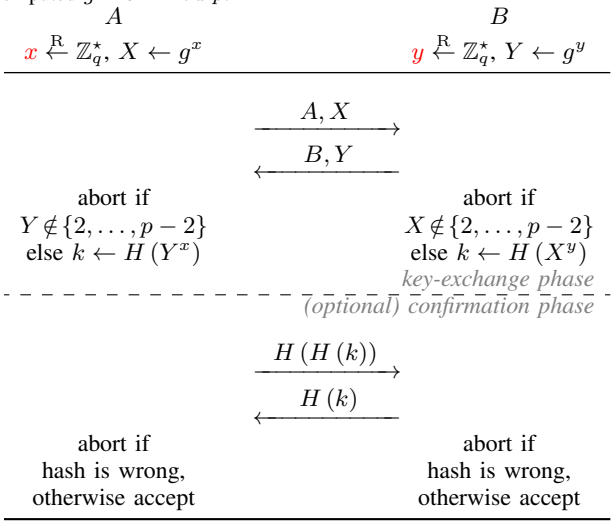
computes the same session key $k = H((g^x)^y) = H(g^{xy})$. The key-exchange phase is completely symmetric. The symmetry in the design helps simplify the security analysis and reduce the communication rounds especially in a mesh network.

The second phase serves to provide explicit assurance that both parties have actually derived the same session key. This is realized in the original SPEKE paper [25] as follows: one party sends $H(H(k))$ first and the other party replies with $H(k)$ later.

The above key confirmation method has two subtle issues. First, it is ambiguous which party should send $H(H(k))$ first. As we will explain, this ambiguity also carries over to the SPEKE specifications in the ISO/IEC and IEEE standards. Second, from a theoretical perspective, the direct use of the session key in the key confirmation process renders the session key no longer indistinguishable from random after the key confirmation is finished, hence breaking the session-key indistinguishability requirement in a formal model [8].

In the standards, the key confirmation phase is optional and it is left to the applications to decide whether it is added. With the absence of this phase, key confirmation will have to be deferred to the later secure communication stage where the session key is used to encrypt and decrypt messages (in the authenticated mode) and the decryption will only work if the session keys used at the two sides are equal.

Figure 1. Original SPEKE scheme. A and B share the password s and computed $g = s^2 \mod p$.



C. Previous attacks

In [33], Zhang proposed an exponential-equivalence attack on SPEKE. This attack exploits the fact that some passwords are exponentially related. For example, two different passwords s and s' may have the relation that $s' = s^r \mod p$ where r is an arbitrary integer ($r \neq 1$). By exploiting this relation, an active attacker can rule out two passwords in one go, and in the general case can rule out multiple passwords in one go if they are all exponentially related. This attack is especially problematic when the password is digits-only, e.g., a Personal Identification Numbers (PIN). As a countermeasure, Zhang proposed to hash the password before taking the square operation: in other words, redefining the password mapping function to $f(s) = (H(s))^2 \mod p$. The hashing of passwords makes it much harder for the attacker to find exponential equivalence among the hashed outputs. Zhang's attack is acknowledged in IEEE P1363.2 [22], which adds a hash function in SPEKE when deriving the base generator from the password.

Tang and Mitchell illustrated three attacks on the SPEKE protocol [31]. The first attack is essentially the same as Zhang's [33]: an active attacker is able to test multiple passwords in one execution of the protocol by exploiting the exponential equivalence of passwords. The authors suggest to hash the identities of the parties along with the password to get the generator, that is $g = H(s\|A\|B)$ where A and B are identities of two communicating parties. However, this countermeasure has the limitation that it breaks the symmetry of the protocol; instead of allowing the two parties to exchange messages simultaneously in one round, the two parties must first agree whose identity should be put first in the hash, which requires extra communication. The second attack is a *unilateral* Unknown Key-Share (UKS) attack. In this attack, the user is assumed to share the same password with more than one servers[1]. By replaying messages, the attacker may trick the user into believing that he is sharing a key with one server, but in fact he is sharing a key with a different server. To address the attack, they propose to include the server's identity into the computation of g. However, same as before, this countermeasure breaks the symmetry of the original protocol. The last attack they show is a scenario where two sessions are swapped. Here, the two parties run two concurrent sessions, and the attacker swaps the messages between the two sessions. At the end of the protocol, the parties will have shared two session keys, but they may get confused which message belongs to which session. They call this a generic vulnerability, which in this paper we call a *sessions swap* attack. To address this problem, they propose to include the "session identifier" into the computation of g, but the paper gives no details on the definition of the "session identifier".

D. Specification in standards

When SPEKE was included into the IEEE P1363.2 and ISO/IEC 11770-4 standards, the protocol was revised to prevent the exponential-equivalence attack reported in [33] and [31]. In the revised protocol, the password is hashed first before computing a secret generator. More specifically, the generator is obtained from $g = (H(s))^2 \mod p$ instead of $g = s^2 \mod p$ as in the original 1996 paper.

[1]We remark that it is unusual to assume a user shares the same password with multiple server in the security model for PAKE, as a server will be able to trivially impersonate another server. However, in practice, many users do reuse passwords across several accounts.

It is also worth noting that the key confirmation procedure of SPEKE defined in the standards is also different from that in the original SPEKE paper [25]. In IEEE P1363.2 [22] and in ISO/IEC 11770-4:2006 [23], the key confirmation is defined as follows.

$$\begin{array}{ccccc} \text{Alice} & \rightarrow & \text{Bob} & : & H(3\|g^x\|g^y\|g^{yx}\|g) \\ \text{Bob} & \rightarrow & \text{Alice} & : & H(4\|g^x\|g^y\|g^{xy}\|g) \end{array} \quad (1)$$

As explicitly stated in the ISO/IEC 11770-4 standard, there is no order in the above two steps. In the same standard, it is also stated that there is no order during the SPEKE exchange phase. We find the two statements contradictory: the fact that g^x comes before g^y in the definition of key confirmation implies there is an order during the key exchange phase.

We would like to highlight that the above issue was carried over from Jablon's original 1996 paper [25], which specifies that "Alice" sends the first confirmation message $H(H(k))$. Given the symmetric nature of the protocol, it is ambiguous which party is "Alice". This ambiguity was unquestioned at the time of standardization and consequently was inherited by the specifications in IEEE P1363.2 and ISO/IEC 11770-4:2006.

We presented the above issue to the ISO/IEC SC 27 technical committee. The issue was acknowledged and rectified in the latest revision ISO/IEC 11770-4:2017. We will explain the details of the change later.

III. New attacks

In this section, we present two new attacks that are not reported before: an impersonation attack and a key-malleability attack. We will first explain how the attacks work on the original SPEKE protocol [25] and then explain their applicability to the SPEKE variants defined in the IEEE and ISO/IEC standards [22], [23].

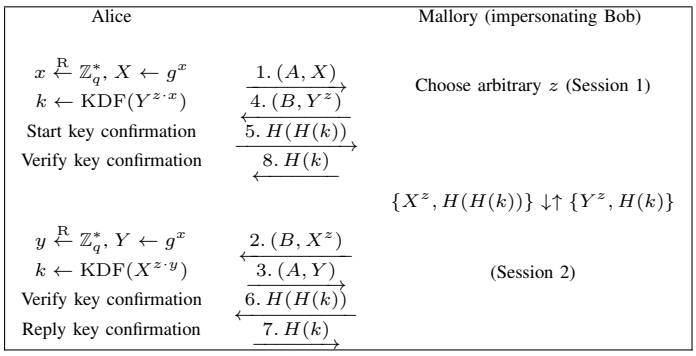
A. Impersonation attack

The first attack happens in the setting of parallel sessions: a user is engaged with another user in multiple sessions running in parallel. We illustrate the attack of Mallory who will be able to impersonate the user Bob to Alice, by launching parallel sessions with Alice to make Alice believe she is communicating with Bob, but actually Bob is not involved at all in the communication.

The attack is illustrated in Figure 2. Details of each step are explained below.

1) Alice chooses a secret exponent x and computes $X \leftarrow g^x$. She initiates the protocol by sending A, X to the insecure channel.
2) Mallory is in control of the channel and intercepts all the messages to Bob who never receives anything. So, Mallory receives the first message from Alice and generates an exponent z such that $X^z \in \{2, \ldots, p-2\}$[2]. Mallory, impersonating Bob, initiates a parallel SPEKE session with Alice by sending her B, X^z.
3) Alice follows the second session generating an exponent y and computing $Y \leftarrow g^y$. She sends A, Y to the insecure channel.
4) Mallory intercepts the message and raises it to the power of z (with overwhelming probability, Y^z will not be 1 or $p-1$). Then, Mallory sends back to Alice B, Y^z in the first session.
5) At this point, Alice computes the key $k = H\left((Y^z)^x\right) = H\left(g^{xyz}\right)$ for the first session, generates the key confirmation challenge $H(H(k))$, and sends it to Bob.
6) Mallory intercepts the challenge from the first session and relays it to Alice in the second session.
7) Following the protocol, Alice answers the challenge with $H\left(k\right)$.
8) Finally, Mallory intercepts Alice's answer in the second session and replays it in the first session to pass the key confirmation procedure.

Figure 2. Impersonation attack on SPEKE



At the end of the above attack, Alice authenticates Mallory as "Bob" in both sessions. However, Mallory does not know any secret password and the real "Bob" has never been involved in this key exchange. This indicates a serious flaw in the authentication procedure. We should note that in the above attack, we assume the initiator of the session is responsible for sending the first key confirmation message. This is allowed by the protocol since SPEKE specifications in both the IEEE and ISO/IEC standards permit the two parties to start the key confirmation in any order.

This attack can be regarded as a special instance of the Unknown Key-Share (UKS) attack [18]. Alice thinks she is communicating with "Bob", but actually she is communicating with another instance of herself. This confusion of identities in the key establishment can cause problems in some scenarios. For example, using the derived session key k in an authenticated mode, like AES-GCM, Alice may send an encrypted message to Bob: "Please pay Charlie 5 bitcoins". Mallory can intercept this message and (without knowing its content) relay back to Alice in the second session. Since the message is verified to be authentic from "Bob", Alice may follow the instruction (assume Alice is an automated program that follows the protocol). Thus,

[2]When $z = 1$ the work of Mallory reduces to simply relaying Alice's messages to herself in the other session, which may be detected if Alice checks for duplicate of messages.

although Alice's initial intention is to make Bob pay Charlie 5 bitcoins, she ends up paying Charlie instead.

B. Key-malleability attack

The second attack is a man-in-the-middle attack as shown in Figure 3. The attacker chooses an arbitrary z from $\{2, \ldots, q-2\}$, raises the intercepted item to the power of z and passes it on. The parties at the two ends are still able to derive the same session key $k = H(g^{xyz})$, but without being aware that the messages have been modified.

The fact that an attacker is able to manipulate the session key without being detected has significant implications on the theoretical analysis of the protocol. In the original SPEKE paper, the protocol has no security proofs; it is heuristically argued that the security of the session key in SPEKE depends on either the Computational Diffie-Hellman assumption (i.e., an attacker is unable to compute the session key) or the Decisional Diffie-Hellman assumption (i.e., an attacker is unable to distinguish the session key from random). The existence of such a key-malleability attack suggests that a clean reduction to CDH or DDH is not possible. As an example, z can be a result of an arbitrary function $f(\cdot)$ with the incepted inputs, i.e., $z = f(g^x, g^y)$. Because of the correlation of values on the exponent, standard CHD and DDH assumptions are not applicable since they require the secret values on the exponent be *independent*.

C. Discussion on standards

1) Explicit key confirmation: Recall from Section II-D that the SPEKE schemes specified in the standards differ from the original SPEKE paper in how the explicit key confirmation is defined. More specifically, the key confirmation procedure in IEEE P1363.2 and ISO/IEC 11770-4 includes additional data in the hash: i.e., key exchange items g^x and g^y. This change does not prevent the impersonation attack; the attacker is still able to relay the key confirmation string in one session to another parallel session to accomplish mutual authentication in both sessions. However, the key-malleability attack no longer works if the key confirmation method in IEEE 1363.2 or ISO/IEC 11770-4 is used. We should emphasize that the key confirmation method in both standards are marked as "optional". Hence, the key-malleability attack is still applicable to the *implicitly authenticated* version of the SPEKE in both standards.

2) Definition of shared secret: In the earlier conference version of the paper [20], we point out that the definition of the shared secret in ISO/IEC 11770-4:2006 [23] is ambiguous. The shared low-entropy secret, denoted π in that standard document [23], is defined as follows.

> *"A password-based octet string which is generally derived from a password or a hashed password, identifiers for one or more entities, an identifier of a communication session if more than one session might execute concurrently, and optionally includes a salt value and/or other data.*

The above definition seems to include the "identifiers for one or more entities" as part of the shared secret. If the entity identifiers were included, the impersonation attack would not work, but the key-malleability would still work. However, the standard does not provide any formula about π. It is not even clear if one or both entities' identifiers should be included, and if only one identifier is to be included, which one and how. Furthermore, the word "generally" weakens the rigour in the definition and makes it subject to potentially different interpretations.

By comparison, the definition of the shared secret in IEEE P1363.2 (D26) [22] is clearer. It is specified as follows:

> *"A password-based octet string, used for authentication. π is generally derived from a password or a hashed password, and may incorporate a salt value, identifiers for one or more parties, and/or other shared data."*

This definition clearly indicates that the incorporation of "a salt value, identifiers for one or more parties, and/or other shared data" is not mandatory (as indicated by the use of the word "may"). Based on the definition, it is clear that both attacks are applicable to the SPEKE scheme defined in IEEE P1363.2.

The issue about the ambiguity in the definition was acknowledge by ISO/IEC SC 27 after we first pointed it out in [20], and was rectified accordingly. In the latest revision in ISO/IEC 11770-4:2017, the definition of the shared secret has been revised to follow the same as in IEEE P1363.2 (D26) [22]. In this revision, the two reported attacks are addressed by making technical changes to the SPEKE specification, as we will explain in the next section.

IV. Solution

A. Patched SPEKE

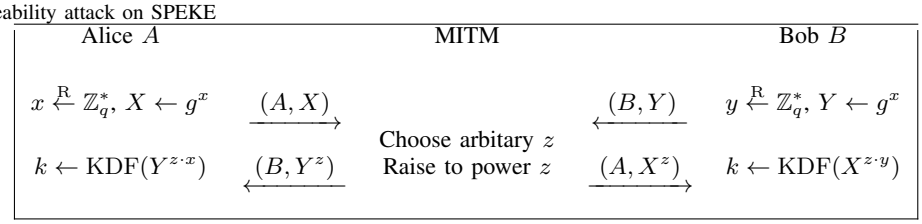
There are several reasons to explain the cause of the two attacks. First, there is no reliable method in SPEKE to prevent a sent message being relayed back to the sender. Second, there is no mechanism in the protocol to verify the integrity of the message, i.e., whether they have been altered during the transit. Third, no user identifiers are included in the key exchange process. It may be argued that all these issues can be addressed by using a Zero Knowledge Proof (ZKP) (as done in [19]). However, in SPEKE, the generator is a secret, which makes it incompatible with any existing ZKP construction. Since the use of ZKP is impossible in SPEKE, we need to address the attacks in a different way.

Our proposed patch is to redefine the session key computation. Assume Alice sends g^x and Bob sends g^y. The session key computation is defined below.

$$\begin{aligned} s_A &= H(A \| g^x) \\ s_B &= H(B \| g^y) \\ sID &= \max(s_A, s_B) \| \min(s_A, s_B) \\ k &= \mathrm{KDF}(sID \| g^{xy}) \end{aligned} \tag{2}$$

When the two users are engaged in multiple concurrent sessions, they need to ensure the identifiers are unique

Figure 3. Key-malleability attack on SPEKE



between these sessions. As an example, assume Alice and Bob launch several concurrent sessions. They may use "Alice" and "Bob" in the first session. When launching a second concurrent session, they should add an extension to make the entity identifier unique – for example, the entity identifiers may become "Alice (2)" and "Bob (2)" respectively in the second session, and so on. The use of the extension is to make the entity identifier distinguishable among multiple sessions running in parallel.

The new definition of the session-key computation function in Eq. 2 prevents both the impersonation and key-malleability attacks (as well as the *session swap* attack reported in [31]), which we will formally prove in the next section. The key confirmation remains "optional" as it is currently defined in the standards. Furthermore, this patch preserves the optimal one-round efficiency of the original SPEKE protocol.

An alternative patch, suggested in the earlier conference paper [20], is to refine the session key computation as follows.

$$\begin{aligned} M &= H\left(\min(A,B) \| \max(A,B)\right) \\ N &= H\left(\min(g^x, g^y) \| \max(g^x, g^y)\right) \\ k &= \mathrm{KDF}(M, N, g^{xy}) \end{aligned} \tag{3}$$

As we will formally analyze in Section V, the above solution also prevents the two attacks. However, the advantage of the solution in Eq. 2 is that the hash output has a fixed bit length, which makes it easier to implement the $\max$ and $\min$ function. The final patch, which has been included into the latest revision of ISO/IEC 11770-4 published in 2017, is summarized in Figure 4.

B. Improved key confirmation

As highlighted in Section II-D, neither of the key confirmation procedures defined in IEEE P1363.2 (D26) and ISO/IEC 11770-4 (2006) is symmetric. In both standards, they state that there is "no special ordering" of the key confirmation message. This implies that the messages can be sent simultaneously within one round. But in fact, these procedures require two rounds instead of one, because the second message depends on the first. This issue also applies to the key confirmation method in Jablon's original 1996 paper [25]. 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 enter a deadlock.

To address the above issue, we propose an improved key confirmation method which preserves the symmetry of the protocol and hence allows the key confirmation to be completed within one round. It works as follows.

$$\begin{array}{rcrcl} \text{Alice} & \rightarrow & \text{Bob} & : & H(A\|B\|g^x\|g^y\|g^{xy}\|g) \\ \text{Bob} & \rightarrow & \text{Alice} & : & H(B\|A\|g^y\|g^x\|g^{xy}\|g) \end{array} \tag{4}$$

An alternative solution, proposed in our earlier paper [20], is based on NIST SP 800-56A Revision 1 [6]. It works as follows.

$$\begin{array}{rcl} \text{Alice} \rightarrow \text{Bob} & : & \mathrm{MAC}(k_c, \text{"KC\_1\_U"} \| A \| B \| g^x \| g^y) \\ \text{Bob} \rightarrow \text{Alice} & : & \mathrm{MAC}(k_c, \text{"KC\_1\_U"}, \| B \| A \| g^y \| g^x) \end{array}$$

In the above method, MAC is a message authenticated code (MAC) algorithm, the string "KC_1_U" refers to unilateral key confirmation, and k_c is a MAC key. To allow the session key to remain indistinguishable from random even after the key confirmation phase, k_c should be derived differently from the session key, e.g., by adding a specific parameter to the key derivation function say $k_c = \mathrm{KDF}(g^{xy}, \text{"KC"})$. There is no dependence between the two flows, so Alice and Bob can send messages in one round. During the revision of ISO/IEC 11770-4, the hash based key confirmation method in Eq. 4 was preferred and was included into the latest standard since it requires minimum changes in the standard.

V. Formal analysis

In this section, we build a formal model using the Applied Pi Calculus, then we apply this model to formally analyse the proposed patch in comparison to existing SPEKE variants.

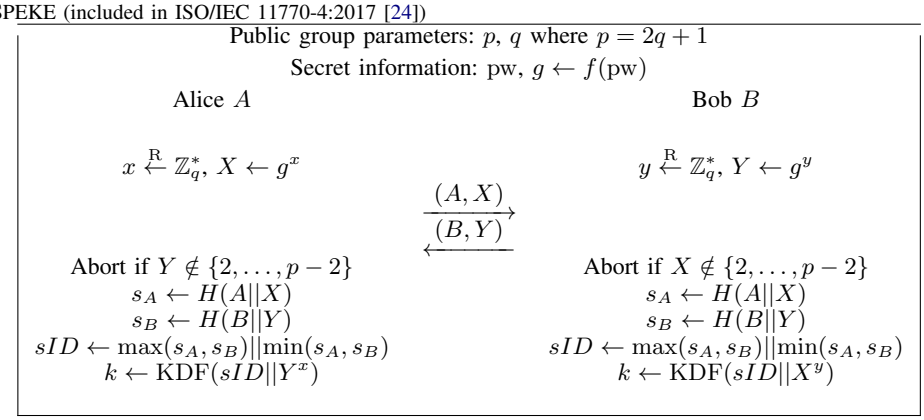
A. Reasoning in the Applied Pi Calculus in ProVerif

ProVerif [13] is a tool for reasoning in the symbolic model. It has proved successful in formally verifying dozens of protocols, and has been widely accepted by the community. ProVerif's input language is a dialect of the Applied Pi Calculus [2], [14], and we limit our informal illustration of the language to the subset that will be useful for describing our model.

The language is strongly typed, and arbitrary types can be declared. We use the following types for our model: $\mathcal{H}$ for the hosts, $\mathcal{S}$ for session IDs, $\mathcal{K}$ for session keys, then we have the order-q sub-group of $\mathbb{Z}_p^\star$ where $p = 2q + 1$ is a safe prime, and the elements in this sub-group can be expressed as g^x where g is a generator and x is from $\mathbb{Z}_q^\star$.

The main abstraction of the Pi Calculus is the process. A process describes the algorithm that an entity follows according to the specifications of a protocol scheme. They can (i) include variables, constants, functions, and (private)

Figure 4. Patched SPEKE (included in ISO/IEC 11770-4:2017 [24])



nonces (i.e. $\nu x.P$ restricts the value x to the process P) (ii) write to and read from any channel c, denoted by $\text{out}(c, \_)$ and $\text{in}(c, \_)$ respectively, (iii) insert and extract elements to and from any table t, $\text{insert } t(\_)$ and $get\ t(\_)$, and (iv) record events. Processes can be put in sequential or parallel execution with other processes including themselves, for unbounded number of times of replication. Such instruments allow for symbolic modelling of protocols.

To model security properties, the language offers some facility. The secrecy of names is verified in terms of unreachability and indistinguishability. The unreachability of the secret by the attacker determines whether the knowledge of the attacker can be augmented with such secret by using the inference rules determined with respect to the model. From the point of view of indistinguishability, the tool determines whether the attacker can distinguish between executions that use different secrets.

More sophisticated security properties, like entity authentication, bilateral unknown shared-key resilience, and others, can be formalised through events and correspondences [12]. Events must be explicitly included as extra lines into the processes, can take arguments, and will be recorded in the traces of execution. Correspondences are implications related to execution of events. By default the content of events is not accessible to the attacker, until the attacker is already aware of them or it will be by other rules. Moreover, the attacker is not directly capable of recording events, but it may induce processes to do so. Loosely speaking, an event can be thought as a piece of meta-semantics with respect to the purpose of the process itself. For example, the event $e(A, B)$ may be interpreted as "Alice believes of having started an authentication with Bob". Although the terminology "Alice believes…" can recall the BAN logic [3], and they undoubtedly share some sort of similarity at a high level, the concept of event is however different. In fact, its formalism has been built on top of a criticism to a lack of formality in the BAN logic [32]; in particular, an event may record something that cannot be interpreted as a *belief*. The generic proposition $e(a_1, \ldots, a_n)$ is used as a short notation to say that there exists (at least) a trace which recorded such event.

The reasoning engine of ProVerif will execute a *main* process and record traces of execution. At the same time, the attacker's knowledge and the tables, if any, are accordingly updated. Security properties are eventually checked by inspecting traces, tables, the attacker's knowledge, and, for equivalences, relations between traces and processes. We refer to the paper by Blanchet [13] for additional details.

B. Modelling the SPEKE protocol

We formally model the following variants of the SPEKE protocol in the Applied Pi Calculus [29]: the original Jablon's protocol [25], the ones in IEEE P1363.2:D26 [22] and ISO/IEC 11770-4:2006 [23], the earlier patch proposed by Hao and Shahandashti in 2014 [20], and the final patch described in this paper and included into ISO/IEC 11770-4:2017 [24], each in two modes:

- without explicit key confirmation,
- with explicit key confirmation as described in the respective documents.

It is worth noting that a meaningful key exchange process should always be completed with some form of key confirmation, let it be explicit or implicit. The explicit key confirmation is realized by executing the explicit key confirmation procedure, which requires extra rounds of communication. But the explicit key confirmation procedure is optional [22], [23]: without it, the key confirmation is deferred to the secure communication stage, and this is called *implicit* key confirmation [30]. However, the exact mechanisms for *implicit* key confirmation are not specified in [22], [23], [25], which makes it difficult to model SPEKE with implicit key confirmation. To address this issue, we assume the implicit key confirmation is realized in the secure communication stage by prepending the first encrypted message with an explicit key confirmation string as defined in the respective explicit key confirmation procedure. Thus, our formal model treats SPEKE with implicit and explicit key confirmations as essentially the same with the only difference being that the latter requires additional rounds of communication.

In the model, we formally specify the following:

The two parties. All variants of the SPEKE protocol involve two parties: the Initiator I and the Responder R.

They are modelled as two processes P_I and P_R. We use the initiator and the responder for the convenience of naming in our model. Essentially we assume that one party initiates the protocol by sending data in the first flow, and the other party responds by sending data in the second flow. Thus a one-round protocol is implemented in two flows. This does not change the security analysis of the protocol. Below we give the "vanilla" specification of the protocol. In the "vanilla" specification, we abstract out key reconstruction by a function symbol kdf, and the confirmation messages sent by the Initiator and the Responder are abstracted by the symbols kcf_I and kcf_R respectively. The actual specification of each variant has its own definitions of kdf, kcf_I and kcf_R to capture the differences between the variants.

Figure 5. The processes for the Initiator, P_I, and the Responder, P_R. In the above specification, the notation $= X$ means abort if the incoming value is not X.

$$
\begin{array}{l|l|l}
1 & P_I \leftarrow \mathrm{in}\,(c,(I,R))\,; & P_R \leftarrow \mathrm{in}\,(c,(I,R))\,; \\
2 & \quad \mathrm{get}\ t\,(=I,=R,g)\ \mathrm{in} & \quad \mathrm{get}\ t\,(=R,=I,g)\ \mathrm{in} \\
3 & \quad \nu x.\mathrm{let}\ X = g^{x}\ \mathrm{in} & \quad \nu y.\mathrm{let}\ Y = g^{y}\ \mathrm{in} \\
4 & \quad \mathrm{out}\,(c,(I,X))\,; & \quad \mathrm{out}\,(c,(R,Y))\,; \\
5 & \quad \mathrm{in}\,(c,(=R,Y))\,; & \quad \mathrm{in}\,(c,(=I,X))\,; \\
6 & \quad \mathrm{let}\ k = kdf\ \mathrm{in} & \quad \mathrm{let}\ k = kdf\ \mathrm{in} \\
7 & \quad \mathrm{out}\,(c,kcf_I)\,; & \quad \mathrm{in}\,(c,=kcf_I)\,; \\
8 & \quad \mathrm{in}\,(c,=kcf_R)\,; & \quad \mathrm{out}\,(c,kcf_R)\,; \\
9 & \quad \mathrm{out}\,(c,\mathrm{enc}\,(k,m))\,; & \quad \mathrm{out}\,(c,\mathrm{enc}\,(k,m))\,;
\end{array}
$$

As can be easily seen in Figure 5, the code inside the boxes is the part modelling the protocol scheme depicted in Figure 1, where the key reconstruction part is abstracted by the function symbol kdf, and the confirmation messages sent by the Initiator and the Responder are abstracted by the symbols kcf_I and kcf_R respectively, where we omit their arguments for simplicity. We highlight the symmetric nature of the protocol letting both processes to write to the channel simultaneously.

The other lines (outside the box) in Figure 5 serve to model the behaviour of the protocol and to verify security properties. In particular, the first line is to let the processes know the identities involved in the protocol; they read them from the channel c at the very beginning. The second line checks whether the password table contains a suitable password to communicate to the other party; otherwise, they abort. The last line is useful to verify the secrecy of the shared key k through the privacy of the message m, and the perfect forward secrecy. The details are deferred to Section V-C where we discuss the security properties.

The pre-shared password. A table t of passwords is filled with all secret group generators that would be calculated from the passwords, i.e., $g \in \mathcal{G}$ is the secret group generator for A and B. From the point of view of the symbolic protocol design, sharing a password and then computing the generator is the same as having directly shared the secret generator.

The main process. Informally, the main process P is an infinite repetition of the parallel execution of the Initiator's process P_I and the Responder's process P_R. Due to the symmetric nature of the SPEKE protocol, the naive implementation of the main process brings *false* attacks where the Initiator speaks to itself. To avoid this issue, we must explicitly support the session within the two parties. However, the session s is not private information, and we disclose it to the attacker by outputting it to the insecure channel c, i.e. $\mathrm{out}\,(c,s)$. At this point, we have the infinite repetition of the following process: $!\,(\nu s.\mathrm{out}\,(c,s)\,;(P_I|P_R))$. The two parties would never engage the protocol if they do not share the password. For this reason, we have an environment process P_P which is in charge of inserting shared passwords into a table that can be accessed by P_I and P_R, but not the attacker. In order to record events and verify correspondences, we also have a process P_A, which records the agreements between the parties through events.

Finally, the main process P that the tool checks has the following structure:

$$P \leftarrow (P_P|\,(!\,(\nu s.\mathrm{out}\,(c,s)\,;(P_I|P_R)))\,|!P_A)\,.$$

The process P_A collects information from two tables, one filled in by the Initiator and the other by the Responder. We emphasise that the protocol can be initiated by either of them and the two tables are put together recording a single event. For security properties that do not require tables to record events, the process will be simply **0**; otherwise, depending on the property to prove, the events e_S end e_C can be recorded, where e_S means that the involved parties in the protocol agree with the participants, the session, and the reconstructed session key at the end of the protocol, and e_C means that the involved parties in the protocol agree with the participants, the session, the secret group, the secret nonce, and the reconstructed session key at the end of the protocol.

C. Security properties

The security properties are modelled as follows.

1) Correctness: This property checks whether the protocol gives authentication and key distribution in presence of honest parties [32]. Even though this property is generally the easiest to prove, it should not be neglected when formally modelling a protocol, in order to avoid either logical or typographic errors. To check the correctness of the models, we need to reconstruct the session key kdf. Its implementation depends on the SPEKE variants.

Formally, for all the sessions and nonce exponents, we require that there exists at least a trace in which the event collecting private and shared values of the participants is recorded and is such that the two honest participants agree on their identities, the password, and the session key with

the right formula.

$$\forall s \in \mathcal{S}, x, y \in \mathbb{Z}_q^\star, g \in \mathbb{Z}_p^\star.\\ e_C(A, B, s, g, x, kdf\,(A, B, g^x, g^y, g^{xy}, s)\,,\\ A, B, s, g, y, kdf\,(A, B, g^x, g^y, g^{xy}, s))$$

where A and B are honest parties and g is the generator calculated from the shared password. If such an event is raised, then there exists a run of the protocol in which the two parties have authenticated each other and they have correctly computed the session key.

2) Secrecy of the pre-shared password: We proved the secrecy of the password through observational equivalence. Formally, if we call π_g the process describing the protocol where two honest parties A and B share the password g, and $\pi_{g'}$ the same protocol but with g' instead of g, then the observational equivalence $\pi_g \approx \pi_{g'}$ describes the property that any attacker cannot distinguish between the two runs of the protocol with probability (non-negligibly) better than a blind guess, and therefore no extra information about the secret password can be gained.

3) Implicit key authentication: Implicit key authentication is verified when only the two participants can reconstruct the session key. This concept is modelled by using the key to encrypt a secret message with deterministic encryption. We then check for observational equivalence of two runs of the processes P_I and P_R where in the last line (Figure 5) the message encrypted is provided by a choice, $\mathrm{out}\,(c, \mathrm{enc}\,(k, [m, m']))$. Similar to how we determine the secrecy of the password, if we call π_m the process describing the protocol where two honest parties A and B encrypt m, and $\pi_{m'}$ the same protocol but with $m' \neq m$, then the observational equivalence $\pi_m \approx \pi_{m'}$ is verified. If the observational equivalence holds and therefore the message m remains secret, it trivially follows that the shared key is at least as secret as m. In fact, the decryption function is public, and the reconstruction of the key will irredeemably compromise the secrecy of m.

4) Explicit key authentication: Explicit key authentication is verified when only the two participants can reconstruct the session key, and they actually do. It is therefore defined as implicit key authentication *and* an agreement on the computed key for the same session. Formally,

$$\forall h_1, h_2 \in \mathcal{H}, s \in \mathcal{S}, k, k' \in \mathcal{K}.\\ e_S\,(h_1, h_2, s, k, h_1, h_2, s, k') \Rightarrow k = k'$$

In other words, in a trace of execution the presence of the event e_S where the first and fifth arguments being equal (agreement on the initiator), the second and the sixth being equal (agreement on the responder), and the third and the seventh being equal (agreement on the session) implies that the fourth and the eighth are equal (equivalence of the reconstructed key). When this property is true, a protocol completed between two authenticated parties in the same session guarantees that the parties agree on the session key. This property, along with the implicit key authentication, gives explicit key authentication.

5) Weak and strong entity authentication: Weak entity authentication guarantees that two parties are indeed speaking to each other. Strong entity authentication requires agreement on other values than the mere entities. These values are supposed not to be injected, produced or inferred by an attacker. Those two properties share similarities in their formality. The events involved are 1) e_I to record that the initiator I believes that it has started a protocol with the responder R; 2) e_R to record that R believes that it has started a protocol with I; 3) e_{IR} to record that I believes that it speaks to R at the end of the protocol, and 4) e_{RI} to record that R believes that it speaks to I at the end of the protocol. The first and the third are recorded by the honest initiator, while the the second and the fourth by the honest responder. Mutual *weak* authentication is provided by the two following symmetric correspondences, one for each honest party:

$$\forall h_1, h_2 \in \mathcal{H}.\ e_{IR}\,(h_1, h_2) \Rightarrow e_R\,(h_1, h_2)\\ \forall h_1, h_2 \in \mathcal{H}.\ e_{RI}\,(h_1, h_2) \Rightarrow e_I\,(h_1, h_2)$$

And mutual strong authentication by the following:

$$\forall h_1, h_2 \in \mathcal{H}, s \in \mathcal{S}, k \in \mathcal{K}.\\ e_{IR}\,(h_1, h_2, s, k) \Rightarrow e_R\,(h_1, h_2, s, k)\\ \forall h_1, h_2 \in \mathcal{H}, s \in \mathcal{S}, k \in \mathcal{K}.\\ e_{RI}\,(h_1, h_2, s, k) \Rightarrow e_I\,(h_1, h_2, s, k)$$

where they agree also on the session and the exchanged session key. Agreeing on the session will prevent any replay attack from other sessions, even concurrent, while agreeing on the key will guarantee that no attacker can let two authenticated parties not to share the same key. However, a key-malleability attack is still possible even if the protocol can achieve strong entity authentication.

6) Perfect forward secrecy: Usually, key exchange protocols verify (or claim) the perfect forward secrecy (PFS) property. For the password authenticated key exchange protocols, this property means that if passwords are compromised, the *past* session keys derived from such passwords still remain secret. Hence, an adversary can only keep a record of past communication which has not been compromised. We can reformulate this concept as a *passive* adversary whom is given the password and eavesdrops (unbounded number of) executions of the protocol trying to reconstruct any of the session keys. In practice, to verify this property we disclose the secret generator g to the attacker, $\mathrm{out}\,(c, g)$, then we query the non-interference property on the encrypted message. Since the passive attacker can compute any decryption, the non-interference property captures the perfect forward secrecy, i.e., if the encrypted message cannot be reconstructed, it must be that any session key cannot be reconstructed either.

7) Bilateral UKS: Informally, a successful bilateral UKS attack makes two honest parties I and R believe that they share k with some other party [15]. To capture this attack,

we use the following correspondence:

$$\forall h_1,h_2,h_1',h_2' \in \mathcal{H}, s,s' \in \mathcal{S}, k \in \mathcal{K}.$$
$$e_S\left(h_1,h_2,s,k,h_1',h_2',s',k\right) \Rightarrow h_1 = h_1' \wedge h_2 = h_2'$$

If an initiator and a responder recorded the same key, then it must be that they agree on the entities. If we required that they should agree on the session too, then we could put $s = s'$ in logical AND with the two equivalences. On the contrary, if we wanted to force the tool to show bilateral UKS attacks in the same session, we could state $s = s'$ as a premise.

8) Impersonation attack: The impersonation attack is a problem that generally affects SPEKE protocols and an instance of such an attack has been shown in Section III-A. To formalise this attack, we build a model in which there exists only one honest party and the attacker. In this case, if the honest party ever shares a key with another party, then the other party must be the attacker, and the attacker must impersonate another honest party in order to run the protocol up to this point. In fact, all SPEKE variants without key confirmation phase are vulnerable to this attack.

To verify, we can check for every honest party, session and key, the event of authenticating the other party is not recorded in any trace (i.e. the adversary cannot establish a shared key with the honest party). Formally, we check the following property:

$$\forall h_1,h_2 \in \mathcal{H}, s \in \mathcal{S}, k \in \mathcal{K}.$$
$$\neg\left(e_{RI}\left(h_1,h_2,s,k\right) \vee e_{IR}\left(h_1,h_2,s,k\right)\right).$$

9) Sessions swap: A man-in-the-middle is able to perform the sessions swap attack if it can let a honest party in some session s share a key with another honest party in some other concurrent session s' and vice versa. This attack occurs in a key-exchange protocol where the key does not depend on the session. Formally, we say that for every two parties and for every key, the presence of an agreement on the Initiator, Responder, and session key must imply the equivalence of the sessions.

$$\forall h_1,h_2 \in \mathcal{H}, s,s' \in \mathcal{S}, k \in \mathcal{K}.$$
$$e_S\left(h_1,h_2,s,k,h_1,h_2,s',k\right) \Rightarrow s = s'.$$

10) Malleability: The malleability of the session key is an attack that affects many variants of the SPEKE protocols, and it was described in Section III-B. Capturing the malleability attack in ProVerif requires more efforts than other attacks, because it is based on an *extra* level of group exponentiation equality (three commutative exponents). This results in a larger search space when the reasoning engine checks the property, and ProVerif slows down considerably (taking minutes instead of milliseconds to verify the non-malleability property on a 3.2 GHz computer with 64 GB RAM running Linux). To detect malleability, we require the two honest parties to write into a table some values they agree with, plus their secret fresh exponents and the secret generator (the password). This way, when checking for correspondence, we can check whether the key is indeed what is expected with regard to the private inputs of the parties.

Formally, for every pair of parties, session, generator, two exponents and key, where the parties agree on the identities, the session, the generator and the key (they cannot agree on the other party's secret), then the key they agree on is computed equivalently to the formula provided by the protocol.

$$\forall h_1,h_2 \in \mathcal{H}, x,y \in \mathbb{Z}_q^\star, g \in \mathbb{Z}_p^\star, s \in \mathcal{S}, k \in \mathcal{K}.$$
$$e_C\left(h_1,h_2,s,h,x,k,h_1,h_2,s,g,y,k\right) \Rightarrow$$
$$k = kdf\left(a,b,g^x,g^y,g^{xy},s\right) \vee kdf\left(b,a,g^y,g^x,g^{xy},s\right).$$

Note the key k may have two different values depending on in the protocol who is the initiator and who is the responder.

VI. Summary of results

The ProVerif scripts that we created to model and verify the protocols are available at GitHub [1]. There are in total 54 scripts related to this paper, each for a different variant and a property. ProVerif will give one of the following four responses: (i) the property is true, (ii) the property is false, (iii) the property cannot be proved, and (iv) non-termination.

The results are summarised in Table I. The proposed patch (as well as the patch in [20]) improves the round efficiency over the previous SPEKE variants [22], [23], [25] by allowing the explicit key confirmation steps to be completed within one round. As a result, it requires only 2 rounds to finish the key exchange with explicit key confirmation as opposed to 3 rounds previously. All variants have the Implicit Key Authentication (IKA) property, confirming that the session key will not be learned by the attacker, and that the attacker cannot get confidential information by eavesdropping. This does not contradict the impersonation attack shown in Section III, since that attack works without the adversary learning the session key. However, that attack demonstrates that the adversary is able to manipulate the two parallel sessions to make them generate *identical* session keys. Consequently, the adversary is able to pass the explicit key confirmation by replaying messages. This is confirmed by our formal analysis that the original SPEKE [25], and the SPEKE in standards [22], [23] do not fulfil the explicit key authentication property. Also, the existence of the impersonation attack shows that these variants do not fulfil the weak/strong entity authentication which concerns assuring the identities of the entities involved in the key exchange protocol. The proposed patch prevents the Session Swap attack (SS), the UKS attack, and the Malleability (MAL) attack by making the session key depend on the session, the identities, and the transcript of the key exchange process. We emphasise that these security properties are verified *before* any key confirmation either implicit or explicit. To guarantee that the participants are mutually authenticated, the key confirmation becomes necessary. Such key confirmation must include all of the key points above, i.e., session, identities, and a transcript of the key exchange messages, so avoiding the above

Table I
SUMMARY OF RESULTS ON EFFICIENCY AND FORMAL VERIFICATION IN PROVERIF

Variants	RND	RND-E	IKA	EKA	WA	SA	IMP	SS	PFS	UKS	MAL
Jablon 1996 paper [25]	1	3	✓	×	×	×	×	×	✓	×	×
IEEE P1363.2:D26 [22]	1	3	✓	×	×	×	×	×	✓	×	✓
ISO/IEC 11770-4:2006 [23]	1	3	✓	×	×	×	×	×	✓	×	✓
Hao and Shahandashti [20]	1	2	✓	✓	✓	✓	✓	✓	✓	✓	✓
P-SPEKE (ISO/IEC 11770-4:2017)	1	2	✓	✓	✓	✓	✓	✓	✓	✓	✓

The results are grouped by variants with and without key confirmation phase (KC).
Legend. **Round efficiency**: without explicit key confirmation (RND), with explicit key confirmation (RND-E). **Security properties**: Implicit Key Authentication (IKA), Explicit Key Authentication (EKA), Weak Entity Authentication (WA), Strong Entity Authentication (SA), Impersonation resilience (IMP), Sessions Swap resilience (SS), Perfect Forward Secrecy (PFS), bilateral Unknown Key-Share resilience (UKS), and Malleability resilience (MAL). **Outcomes**: (✓) - verified, (×) - attacks found, (−) - not applicable.

mentioned attacks. Including only the identities allows to verify *weak* entity authentication only.

Our formal analysis using ProVerif confirms that our proposed patch prevents the two attacks as identified earlier. However, this analysis does not constitute a complete proof of security for SPEKE, as one might expect from formal authenticated key exchange models [4], [7], [8], [17], [27]. In particular, we have not proved that SPEKE is resistant against off-line dictionary attacks based on standard security assumptions such as DDH or CDH. We highlight that the original SPEKE was designed without a security proof. Retrospective efforts to prove the security of a protocol based on standard number theoretical assumptions may turn out to be very hard if not impossible. We leave further analysis of SPEKE to future work.

VII. CONCLUSIONS

The SPEKE protocol was firstly proposed by Jablon over two decades ago. Since then, it has been adopted by international standards, and built into smart phones and other products. We identified two weaknesses in the standardized SPEKE specification, which affect all implementations that follow the IEEE 1362.3 and ISO/IEC standards. Accordingly we proposed a patched SPEKE to address the identified issues. We formally modelled the discovered attacks against SPEKE and proved that the proposed patch was immune to these attacks. In addition, we contributed to improve the round efficiency of the protocol in the key confirmation phrase. Our proposed patch and the improved key confirmation procedure have been included into the latest revision ISO/IEC 11770-4 published in July 2017. However, the SPEKE specification in IEEE P1363.2 (which is currently not maintained) remains unfixed.

The problems in SPEKE identified in this paper have evaded 20 years cryptanalysis (informal and formal) by the security and standardization communities. The initial discovery of the two attacks on SPEKE was down to manual analysis, which was later formally verified by applying the ProVerif tool. The mechanised proofs that we produce are not only helpful for proving security properties of similar protocols, but also for preventing the same problems in the future. This shows that traditional human cryptanalysis, in conjunction with modern automated proof techniques, is useful in improving security protocols, especially those that have been included in international standards.

ACKNOWLEDGEMENTS

We thank Professor Liqun Chen for her invaluable advice and comments on revising SPEKE in ISO/IEC 11770-4.

REFERENCES

[1] Verification of SPEKE using ProVerif. https://github.com/nitrogl/speke-verification, 2018.
[2] M. Abadi, B. Blanchet, and C. Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *arXiv preprint arXiv:1609.03003*, 2016.
[3] M. Abadi and M. R. Tuttle. A semantics for a logic of authentication. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 201–216. ACM, 1991.
[4] M. Abdalla, F. Benhamouda, and P. MacKenzie. Security of the j-pake password-authenticated key exchange protocol. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 571–587. IEEE, 2015.
[5] M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA*, volume 3376, pages 191–208. Springer, 2005.
[6] E. Barker, L. Chen, A. Roginsky, and M. Smid. Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography. *NIST special publication*, 800:56A, 2013.
[7] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *international conference on the theory and applications of cryptographic techniques*, pages 139–155. Springer, 2000.
[8] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Annual international cryptology conference*, pages 232–249. Springer, 1993.
[9] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pages 72–84. IEEE, 1992.
[10] J. Bender, Ö. Dagdelen, M. Fischlin, and D. Kügler. The pacel aa protocol for machine readable travel documents, and its security. In *Financial Cryptography*, volume 7397, pages 344–358. Springer, 2012.
[11] BlackBerry Unlimited. *Security Note, BlackBerry 10 Devices*, 2016.
[12] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.
[13] B. Blanchet et al. An efficient cryptographic protocol verifier based on prolog rules. In *csfw*, volume 1, pages 82–96, 2001.
[14] B. Blanchet et al. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends® in Privacy and Security*, 1(1-2):1–135, 2016.
[15] L. Chen and Q. Tang. Bilateral unknown key-share attacks in key agreement protocols. *J. UCS*, 14(3):416–440, 2008.
[16] Entrust. *Entrust TruePass™Product Portfolio Strong Authentication, Digital Signatures and end-to-end encryption for the Web Portal.*, 2003.
[17] O. Goldreich and Y. Lindell. Session-key generation using human passwords only. In *Annual International Cryptology Conference*, pages 408–432. Springer, 2001.
[18] F. Hao. On robust key agreement based on public key authentication. In *Financial Cryptography*, volume 6052, pages 383–390. Springer, 2010.

[19] F. Hao and P. Y. Ryan. Password authenticated key exchange by juggling. In *International Workshop on Security Protocols*, pages 159–171. Springer, 2008.

[20] F. Hao and S. F. Shahandashti. The speke protocol revisited. *SSR*, 14:26–38, 2014.

[21] D. Harkins. Simultaneous authentication of equals: a secure, password-based key exchange for mesh networks. In *Second International Conference on Sensor Technologies and Applications, 2008 (SENSORCOMM'08)*, pages 839–844. IEEE, 2008.

[22] Standard specifications for password-based public-key cryptographic techniques. Standard, Institute of Electrical and Electronics Engineers, Inc., Sep 2006.

[23] Information technology - security techniques - key management - part 4: Mechanisms based on weak secrets. Standard, International Organization for Standardization, Geneva, CH, 2006.

[24] Information technology - security techniques - key management - part 4: Mechanisms based on weak secrets. Standard, International Organization for Standardization, Geneva, CH, 2017.

[25] D. P. Jablon. Strong password-only authenticated key exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, 1996.

[26] B. Jaspan. Dual-workfactor encrypted key exchange: Efficiently preventing password chaining and dictionary attacks. In *USENIX Security Symposium*, 1996.

[27] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 475–494. Springer, 2001.

[28] G. Lowe. A hierarchy of authentication specifications. In *Computer security foundations workshop, 1997. Proceedings., 10th*, pages 31–43. IEEE, 1997.

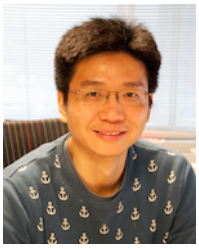
[29] M. D. Ryan and B. Smyth. Applied pi calculus. In V. Cortier and S. Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*. IOS Press, 2011.

[30] D. R. Stinson. *Cryptography: theory and practice*. CRC press, 2005.

[31] Q. Tang and C. Mitchell. On the security of some password-based key agreement schemes. *Computational Intelligence and Security*, pages 149–154, 2005.

[32] T. Y. Woo and S. S. Lam. A semantic model for authentication protocols. In *Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on*, pages 178–194. IEEE, 1993.

[33] M. Zhang. Analysis of the speke password-authenticated key exchange protocol. *IEEE Communications Letters*, 8(1):63–65, 2004.



Feng Hao received a PhD degree in Computer Science from the University of Cambridge in 2007. After working in security industry for several years, he joined the School of Computing, Newcastle University, as a lecturer (assistant professor) in 2010. He is currently a reader (associate professor) in Security Engineering. His research interests include applied cryptography, system security, and efficient computing algorithms.

Roberto Metere is a research student in Secure Computation at the School of Computing, Newcastle University. His research interests include theoretical and applied cryptography, data privacy, and machine code verification. His current research focuses on mechanising proofs in the computational model and the symbolic model for cryptographic protocol design in secure computation.

Siamak F. Shahandashti is a lecturer (assistant professor) at the University of York, UK. His research interests include applied cryptography, privacy-preserving protocols, electronic voting, and blockchain technology. He received his PhD from Wollongong University, Australia, and has been with Victoria University, Sydney, École Normale Supérieure, Paris, and Newcastle University, UK, before joining York.

Changyu Dong received a Ph.D. from Imperial College London. He is currently a senior lecturer at the School of Computing, Newcastle University. His research interests include applied cryptography, trust management, data privacy and security policies. His recent work focuses mostly on designing practical secure computation protocols. The application domains include e.g. secure cloud computing and privacy preserving data mining. He has over 30 publications in international journals and conferences.