Improving the ISO/IEC 11770 standard for key management techniques

Cas Cremers and Marko Horvat

University of Oxford

Abstract. We provide the first systematic analysis of the ISO/IEC 11770 standard for key management techniques [18, 19], which describes a set of key exchange, key authentication, and key transport protocols. We analyse the claimed security properties, as well as additional modern requirements on key management protocols, for 30 protocols and their variants. Our formal, tool-supported analysis of the protocols uncovers several incorrect claims in the standard. We provide concrete suggestions for improving the standard.

1 Introduction

The International Organisation for Standardisation (ISO) develops and promotes international standards, which include a wide variety of security mechanisms. Many large vendors aim to support ISO standards, for example because they are mandated by oversight bodies [15] or to prevent trade barriers. Hence, it is critical that the ISO standards for security mechanisms are thoroughly scrutinised. However, most previous analyses of the ISO security standards have been very limited in scope, e.g., [10, 16, 23, 24]. One exception is the analysis of Basin et al. of the ISO/IEC 9798 standard for entity authentication [4] in 2012. Their analysis uncovered a series of issues that led to an updated version of the 9798 standard.

In this paper we focus on the ISO/IEC 11770 standard for key management protocols, in particular on parts 2 and 3 of this standard. In the most recent version as of June 2014, these two parts together describe 30 base protocols for key exchange, key agreement, and key transport. Many of the standard's protocols are based on protocols such as Diffie-Hellman, variants of MQV, and the TLS handshake. For many of the protocols, at least two variants are described. Thus, analysing these two parts is a significant undertaking.

In positive contrast to other security protocol standards [5], the ISO/IEC 11770 standard explicitly specifies security properties for each of its protocols. Two of these properties are structural properties, i.e., key control and replay detection. Additionally, there are four security properties that relate to active adversaries, namely key authentication, key confirmation, entity authentication, and forward secrecy.

In this work, we use tool-supported formal methods to determine if the protocols indeed satisfy their claimed properties. Additionally, we analyse the protocols for modern key exchange security properties, such as resilience against Key Compromise Impersonation (KCI) and Unknown Key Share (UKS) attacks.

Contributions We perform the first comprehensive analysis of parts 2 and 3 of the ISO/IEC 11770 standard. Our analysis uncovers multiple previously unreported errors and weaknesses. For each of the discovered issues, we provide concrete recommendations for improving the standard.

Our protocol models and tools used are available for download from `http://www.cs.ox.ac.uk/people/cas.cremers/scyther/iso11770/`.

Acknowledgements This work builds on, and extends abstract protocol models originally developed for an earlier analysis of ISO/IEC 11770 by Lara Schmid [25].

Overview In Section 2 we give some background on ISO/IEC 11770 and illustrate some of its protocols. We describe our analysis approach in Section 3 and present the results in Section 4. We provide concrete recommendations for improving the standard in Section 5, discuss related work in Section 6, and conclude in Section 7.

2 Background on ISO/IEC 11770

The ISO/IEC 11770 standard describes key management techniques. According to the standard, the purpose of key management is to provide procedures for handling cryptographic keying material to be used in symmetric or asymmetric mechanisms. Effectively, the standard describes a large number of key agreement, key transport, and key exchange protocols. We will therefore use the terms *mechanism* and *protocol* interchangeably.

The standard is currently divided into five parts. Part 1 was originally released in 1996 and has been updated over the years. It describes the context and framework. Parts 2 and 3 describe mechanisms based on symmetric and asymmetric techniques. Part 4 describes mechanisms based on weak secrets, such as password-based key exchange protocols. Part 5 describes group key management mechanisms. A part 6 on key derivation functions is currently under development.

2.1 Protocols

In this work we focus on part 2 [18] and part 3 [19] of the ISO/IEC 11770 standard. Part 2 describes 13 key establishment mechanisms. Part 3 describes 11 key agreement mechanisms and 6 key transport mechanisms. Many of these 30 mechanisms in parts 2 and 3 have optional message parts and message flows, giving rise to a large number of variants.

Additionally, the mechanisms produce keying material that must be used with a key derivation function to form a key for use in further messages. The standard does not specify a single key derivation function; instead it gives examples of various possible key derivation functions. Thus, using a single mechanism with different key derivation functions can be regarded as multiple variants of the same base mechanism. As we will see in Section 4.3, the choice of a key derivation function can influence the security of a mechanism.

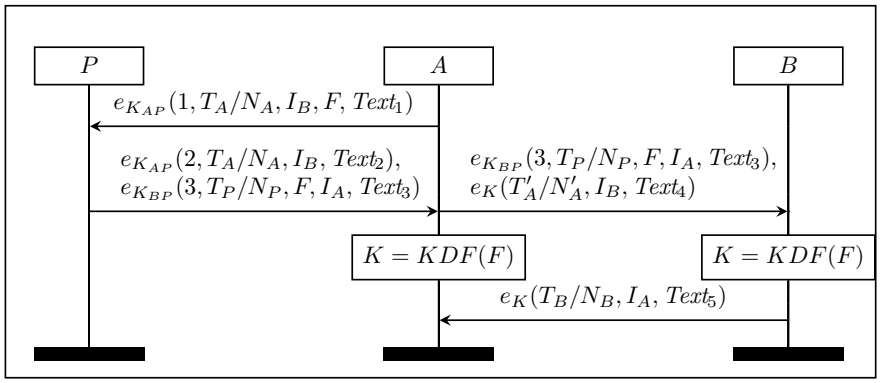

Fig. 1. Protocol 2-12 with optional fields

Naming conventions. We provide a unique name for each base mechanism in the considered parts of the standard. We refer to the thirteen key establishment mechanisms from part 2 as protocol 2-1, 2-2, ..., 2-13. We refer to the key agreement mechanisms as 3-KA-1, ..., 3-KA-11 and to the key transport mechanisms as 3-KT-1, ..., 3-KT-6.

We next describe two protocols from the standard. This enables us to introduce notation and provide an indication of the protocols contained in the standard.

Key Exchange Mechanism 12 (2-12). We give an example of a protocol described in part 2 [18], referenced in the standard in Section 7.2 as Key Establishment Mechanism 12. The protocol is stated to be derived from, but not fully compatible with, the four-pass mutual authentication mechanism specified in ISO/IEC 9798-2 [17]. The protocol has several variants. For this example, we consider the variant with all optional parts included, depicted using a Message Sequence Chart (MSC) in Figure 1.

In the figure, T_A/N_A is either a time stamp T_A or sequence number N_A of A. I_A and I_B respectively identify entities A and B. $e_K(m)$ denotes the encryption of the message m with the key K. The protocol assumes that entities A and B respectively share long-term symmetric keys K_{AP} and K_{BP} with a trusted third party P. $Text_1$ through $Text_5$ are text fields whose contents are not specified by the standard. F denotes keying material.

The protocol proceeds as follows. When a party A wants to communicate with another party B, it contacts trusted third party P. A generates fresh keying material F and includes it in the message encrypted for P, who responds with two encrypted messages. They are respectively encrypted with K_{AP} and K_{BP}. Both encrypted messages are sent to A, who forwards the second encryption to B. B decrypts the message and obtains the keying material F. A and B now both use a key derivation function to compute the session key K from F. We are only considering the protocol variant with optional fields, so the protocol proceeds with two messages that allow both entities to confirm to the other entity that they have successfully computed the key.

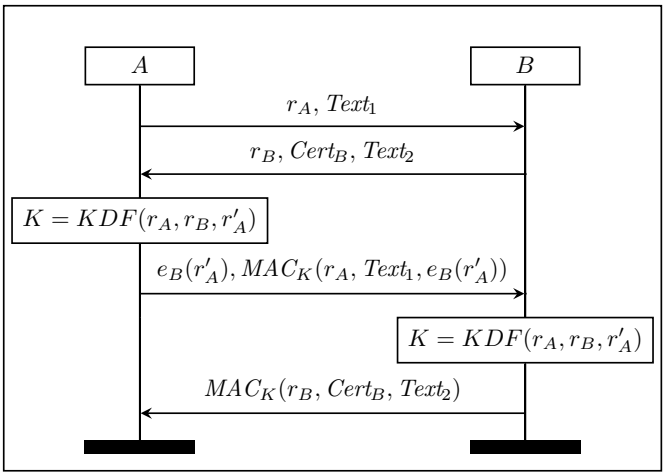

Fig. 2. Protocol 3-KA-11

For the key derivation function, we consider two extremes from the KDFs described in the standard: at the one end, some KDFs take as input only F, whereas others include additional parameters, such as the identities I_A and I_B.

Key Agreement Mechanism 11 (3-KA-11). Key Agreement Mechanism 11 from part 3, shown in Figure 2, establishes a key shared by entities A and B. First, A generates a random value r_A and sends it to B. B responds with his own random value r_B and his certificate. Upon receiving this message, A generates a new random value r'_A. r'_A is used with the other two random values to derive a session key K. Then r'_A is encrypted using B's public key, and sent to B along with a message authentication code (MAC) keyed with K that includes the earlier randomness r_A. B decrypts the message, computes K, and checks the MAC. B then responds with his own MAC of r_B and his certificate.

According to the standard, this protocol is derived from the TLS handshake protocol [14]. In particular, since only B uses his private key (to decrypt the message) and the random values are directly input to the key derivation function, the protocol resembles TLS's unilaterally authenticated RSA mode, where A corresponds to the client and B to the server. The random value r'_A in 3-KA-11 plays the same role as TLS's *pre-master secret* and the two text fields are used in TLS for the cipher suite negotiation.

2.2 Security properties and threat model of the standard

Most standards for security protocols do not specify threat models or intended security properties [5]. In this respect, ISO/IEC 11770 is an exception since it explicitly specifies a set of security properties, and states for each protocol which of these properties it satisfies. ISO/IEC 11770 defines the following properties [18,19]:

Implicit key authentication from entity A to entity B Assurance for entity B that A is the only other entity that can possibly be in possession of the correct key.

Explicit key authentication from entity A to entity B Assurance for entity B that A is the only other entity that is in possession of the correct key.

Key confirmation from entity A to entity B Assurance for entity B that entity A is in possession of the correct key.

Entity authentication of entity A to entity B Assurance of the identity of entity A to entity B.

Forward secrecy with respect to entity A Property that knowledge of entity A's long-term private key subsequent to a key agreement operation does not enable an opponent to recompute previously derived keys.

Forward secrecy with respect to both entity A and entity B Property that knowledge of entity A's long-term private key or knowledge of entity B's long-term private key subsequent to a key agreement operation does not enable an opponent to recompute previously derived keys.

Mutual forward secrecy Property that knowledge of both entity A's and entity B's long-term private keys subsequent to a key agreement operation does not enable an opponent to recompute previously derived keys.

For example, regarding the protocols described in the previous section, the standard claims the following: protocol 2-12 with optional parts satisfies mutual explicit key authentication, mutual key confirmation and mutual entity authentication, and protocol 3-KA-11 provides mutual explicit key authentication, mutual key confirmation, entity authentication to B and mutual forward secrecy.

The standard does not specify an explicit threat model. However, the security properties stated above are not claimed for all protocols. Because some protocols apparently do not meet the above properties, we can conclude that the adversary is considered to have at least the following capabilities:

Injecting network messages Entity authentication is claimed for some, but not all mechanisms. Entity authentication can only be effectively violated if the adversary is able to inject or tamper with network messages.

Eavesdropping on network messages If the adversary cannot eavesdrop on messages, we would need no complex key management mechanism, and could exploit simple authentication mechanisms.

Compromising long-term private keys For some protocols, perfect forward secrecy is claimed. The adversary can only violate perfect forward secrecy by compromising the long-term private keys of some entities.

3 Formally modelling the protocols and their properties

We analyse all 30 protocols specified in the standard, along with their described variants, by using formal methods. In particular, we use the Scyther framework [13] for the automatic symbolic analysis of security protocols. The Scyther tool [11] has built-in support for compromising adversaries [2], and is therefore especially suitable for analysing security notions that are common in the domain of protocols for key agreement, exchange, and transport.

3.1 Protocol specification

Within the Scyther framework, protocols are specified using so-called role scripts. A protocol can have any finite number of roles, and is run by entities who execute those roles. Entities may execute each role multiple times, and every role can be executed by any entity. We call each such role instance a session. We assume that, prior to protocol execution, every entity has generated or securely received a long-term asymmetric key pair consisting of a public and a private key, it has authentic and secret copies of all its long-term symmetric keys shared with other entities, and authentic copies of the public keys of all other entities.

Roles are specified as sequences of send, receive and claim events. Events have term parameters, where terms are constructed from role names, function names, variables, and constants. Receive events correspond to pattern matching on incoming messages, and may therefore contain nonces generated in previous send steps and variables to store incoming payloads. Send events can contain freshly generated nonces and variables that have been previously initialised in receive steps. We specify intended security properties using claim events.

For example, we give in Figure 3 the input for the Scyther tool to describe protocol 2-12 from Section 2.1. Send, receive and claim events are respectively specified with `send`, `recv`, and `claim`. Freshly generated nonces are declared with `fresh`, variables with `var`, user-defined types with `usertype`, hash functions with `hashfunction`. Every function, constant and variable can have a different type, such as `Nonce` or a user-defined type such as `Integer`, `KeyingMaterial`, or `String`—the types are used to restrict the pattern matching in the execution of a receive event. The keyword `macro` can be used to define shorthands.

3.2 Specifying security properties

We model the following properties from the standard: *key authentication*, *key confirmation*, *entity authentication*, and *forward secrecy*. Additionally, we model *key compromise impersonation (KCI)* and *unknown key share (UKS)* attacks.

Key authentication. According to the standard, both implicit and explicit key authentication require that if an entity A uses a protocol to establish a key K with entity B, then only A and B will learn the key. We model this by analysing the secrecy of K whilst allowing the adversary to impersonate any entity except for A and B. The possibility of impersonation is modelled by allowing the adversary to learn the long-term private key of any entity except for A and B. Additionally, explicit key authentication requires that entities in fact compute the key. We cover this requirement in our modelling of key confirmation.

Key confirmation. This and the following property correspond to authentication properties in Lowe's hierarchy [22]. Key confirmation from A to B corresponds to non-injective data agreement on the key, which we model with two claims: a Running claim in the A role and a Commit claim in the B role. If the Commit claim is executed, we require that the corresponding Running claim is executed as well: it must have the entities in reverse order, and the same

```
option "--partner-definition=2";

usertype KeyingMaterial;
usertype String;
usertype Integer;

hashfunction KDF;
const N1,N2,N3: Integer;

macro key = KDF(F);
macro sid = (A,B,key);

protocol 2-12-withOptional(A,B,P)
{
  role A
  {
    fresh TNA,TNA2: Nonce;
    fresh F: KeyingMaterial;
    fresh Text1,Text4: String;
    var Text2,Text5: String;
    var T: Ticket;
    var TNB: Nonce;

    claim(A,SID,sid);
    claim(A,Running,B,key);
    send_1(A,P,{N1,TNA,B,F,Text1}k(A,P));
    recv_2(P,A,{N2,TNA,B,Text2}k(A,P),T);
    send_3(A,B,T,{TNA2,B,Text4}key);
    recv_4(B,A,{TNB,A,Text5}key);

    claim(A,SKR,key);
    claim(A,Commit,B,key);
    claim(A,Alive,B);
  }
  role B
  {
    var TNP,TNA2: Nonce;
    var F: KeyingMaterial;
    var Text3,Text4: String;
    fresh TNB: Nonce;
    fresh Text5: String;

    recv_3(A,B,{N3,TNP,F,A,Text3}k(B,P),
                {TNA2,B,Text4}key);
    claim(B,SID,sid);
    claim(B,Running,A,key);
    send_4(B,A,{TNB,A,Text5}key);

    claim(B,SKR,key);
    claim(B,Commit,A,key);
    claim(B,Alive,A);
  }
  role P
  {
    var TNA: Nonce;
    var F: KeyingMaterial;
    var Text1: String;
    fresh Text2: String;
    fresh TNP: Nonce;
    fresh Text3: String;

    claim(P,SID,P);
    recv_1(A,P,{N1,TNA,B,F,Text1}k(A,P));
    send_2(P,A,{N2,TNA,B,Text2}k(A,P),
                {N3,TNP,F,A,Text3}k(B,P));
  }
}
```

Fig. 3. Scyther input file for 2-12 with confirmation messages and claimed properties.

contents (the entities are said to *agree* on the contents). It is called *non-injective* data agreement because replays are not considered.

Entity authentication. Entity authentication from A to B corresponds to aliveness [22]: an Alive claim of A is placed in the specification of role B. Whenever the claim is executed, the entity assumed to be performing the A role is required to have executed some event.

Forward secrecy. There are several definitions of forward secrecy in the literature, and it is not clear from the standard which property is intended. The *mutual forward secrecy (MFS)* notion from the standard seems to be closest to two common formal definitions. *Weak Perfect Forward Secrecy (wPFS)* [12, 20] requires that the adversary does not actively inject messages (and thus is passive) with respect to the session that he attacks. In contrast, *(strong) Perfect Forward Secrecy (PFS)* allows the adversary to actively interfere with the messages received by the session under attack. Scyther directly supports checking for both properties through its support of the LKRaftercorrect and LKRafter rules [2]. Our analysis revealed that the majority of protocols for which MFS is claimed in fact only achieve wPFS, and we therefore interpret MFS as wPFS.

Key compromise impersonation (KCI). Resilience to KCI attacks is a desirable property of key exchange protocols [6]. KCI attacks are attacks in which the adversary exploits his knowledge of the long-term private key of Alice to impersonate any entity in subsequent communication with Alice.

This property is modelled in Scyther by a session key secrecy claim of an entity whose long-term private keys the adversary is allowed to reveal. These attacks can be seen as a broader class than unilateral forward secrecy attacks because KCI attacks allow for dynamic usage of the compromised keys: the adversary can use them during protocol execution to inject messages or otherwise tamper with the communication.

Unknown key share (UKS). Unknown key share attacks are attacks in which only Alice and Bob know the session key K; however, Alice and Bob disagree on who they share K with [7]. For example, Alice correctly thinks K is shared with Bob, but Bob might think that K is shared with Charlie. Even though the adversary does not learn the key in such attacks, using the key is not sufficient to authenticate subsequent messages: if Alice sends a message encrypted with K or accompanied by a MAC keyed by K, Bob will assume that the message came from Charlie. Similarly, Bob will send messages intended for Charlie that will be received by Alice.

We model UKS attacks in the standard way, i.e., if the assumptions on the partner identities of the attacked session s do not match the assumptions of a session s', we allow the adversary to reveal the session key of s'. This causes UKS attacks to manifest as violations of secrecy of the session key computed by s. Note that false positives can also occur, where the revealed session key is used for more than computing the session key of s', e.g., for injecting messages.

We specify session identifiers (SIDs) manually in Scyther input files by including `option "--partner-definition=2"` and annotating each role with `SID` claims, in which the SID is specified for the role instance. For example, in Figure 3 we enable the manual specification of a partner session on line 1, define the session identifier on line 11 and insert it into role specifications on lines 24 and 45. When session key secrecy is analysed for a session s, and Scyther's SKR adversary rule (Session-Key Reveal) is enabled in the GUI or `--SKR=1` is provided as a command-line option, the adversary is able to obtain the session keys computed by any session whose identifier differs from that of s.

4 Results of the formal analysis

We analyse the protocol models described in the previous section with respect to their claimed properties, and afterwards consider KCI and UKS attacks.

4.1 Claimed properties

We give an overview of our results when analysing the protocols with respect to their claimed properties in Table 1. The contents of this table are directly taken from the tables in [18, 19], with the difference that we added notes and used red

and bold to mark incorrect statements. We classify the incorrect claims in the standard into five categories AT1...AT5, which we describe below.

Note that the table in [18] only has an (explicit) key authentication column with "yes" or "no" in the cells, but this information has to be combined with NOTE 2 [18], which states that all protocols in part 2 achieve implicit key authentication, and that "yes" should be interpreted as explicit key confirmation.

AT1**: entity authentication failures for 2-8, 2-9, 2-12, and 2-13.** We find several possible entity authentication failures for protocols in part 2 that are derived from protocols in an earlier version of the ISO/IEC 9798-2 standard for entity authentication [17].

These attacks are closely related to the attacks on the corresponding protocols from the 9798 standard as presented in [4]. The attacks work in all implementations where a single entity can perform not only the role of the trusted third party but also another role. In the attacks, the adversary can cause A to complete the protocol, apparently with B, even though B is not present. Thus, the attacks violate even the weakest form of entity authentication. We show an example of such an attack on protocol 2-12 in Appendix A.

Fixes for these protocols have been proposed in [4], which have been integrated into the ISO/IEC 9798 standard. As a result, these attacks no longer work on ISO/IEC 9798, but since no changes have been made to the derived protocols in ISO/IEC 11770, similar attacks are still possible on this standard.

AT2**: 3-KA-11 key authentication/confirmation failure for B.** According to the standard this mechanism (depicted in Figure 2) offers mutual explicit key authentication and mutual key confirmation. However, there is an attack on entity authentication on the B role that violates both of these claimed properties. In the attack, the adversary performs the A role, pretending to be Alice, and sends messages to Bob in the B role. Because executing the A role does not require the use of any long-term secrets, the adversary can simply claim to be anybody. The entity performing the B role therefore cannot obtain any authentication guarantees about its communication partner or about the secrecy of the key.

As said before, 3-KA-11 is derived from the unilaterally authenticated RSA mode of the TLS handshake [14]. In this mode of TLS, the server obtains no guarantee about whether the client is who he claims to be or not. The same issue occurs here for the B role of 3-KA-11.

AT3**: Failure of MFS for 3-KA-11.** Because protocol 3-KA-11 is derived from the RSA mode of TLS, it provides no forward secrecy. The adversary only needs to observe a regular session. If he afterwards obtains the long-term private key of B, he can decrypt $e_B(r'_A)$ and learn r'_A. Since r_A and r_B have been sent in plaintext, the adversary now has all the ingredients he needs to recompute the key K.

AT4**: Failure of key authentication for 2-11.** Depending on the implementation, it may be possible for an agent to misinterpret an agent identity as (random) keying material, for example if both are the same bit length. If

Table 1. Claimed properties Security properties claimed for the protocols in parts 2 and 3 of the standard. Our analysis revealed that some claims are incorrect, and we mark them using bold and red.

Mechanism in part 2	Key Authentication	Key Confirmation	Entity Authentication
2-1	implicit	no	no
2-2	implicit	no	no
2-3	explicit	no	A
2-4	explicit	no	A
2-5	explicit	no	A & B
2-6	explicit	no	A & B
2-7	implicit	no	no
2-8	**explicit** (AT1)	**opt.** (AT1)	**opt.** (AT1)
2-9	**explicit** (AT1)	**opt.** (AT1)	**opt.** (AT1)
2-10	explicit	no	no
2-11	**explicit** (AT4)	no	no
2-12	**explicit** (AT1)	**opt.** (AT1)	**opt.** (AT1)
2-13	**explicit** (AT1)	**opt.** (AT1)	**opt.** (AT1)

Mechanism in part 3	Implicit Key Authentication	Key Confirmation	Entity Authentication	Forward Secrecy
3-KA-1	A,B	no	no	no
3-KA-2	B	no	no	A
3-KA-3	A,B	B	A	A
3-KA-4	no	no	no	MFS
3-KA-5	A,B	opt	no	A,B
3-KA-6	A,B	opt	B	B
3-KA-7	A,B	A,B	A,B	MFS
3-KA-8	A,B	no	no	A
3-KA-9	A,B	no	no	MFS
3-KA-10	A,B	A,B	A,B	MFS
3-KA-11	**A,B** (AT2)	**A,B** (AT2)	B	**MFS** (AT3)
3-KT-1	B	no	no	A
3-KT-2	B	B	A	A
3-KT-3	B	B	A	A
3-KT-4	A	A	B	B
3-KT-5	A,B	(A),B	A,B	no
3-KT-6	A,B	**A,B** (AT5)	A,B	no

an implementation of 2-11 cannot tell the difference between these, it can be vulnerable to a type-flaw attack on key authentication.

The 2-11 protocol assumes pre-shared symmetric keys and a trusted third party P. In a regular execution of the protocol, A sends a request to P for a ticket to forward to B. The request is a triplet $(I_B, F, \mathit{Text}_1)$ encrypted with the key shared between A and P. P then returns a triplet $(F, I_A, \mathit{Text}_2)$ encrypted with the key shared between B and P, which A forwards to B.

The adversary Charlie can attack a session of Bob which assumes to be talking to Alice, even though Alice and Bob are not compromised. Charlie encrypts a message for the trusted third party Pete, requesting a key for Alice. However, instead of generating new keying material F, Charlie instead includes Bob's identity in the keying material field. Pete's response therefore is the triplet $(I_{\text{Alice}}, I_{\text{Charlie}}, \mathit{Text}_2)$ encrypted with $K_{\text{Pete,Bob}}$. Charlie re-sends this message to Pete. There is nothing in the standard that prevents Pete from accepting this message as a valid request. Now, Pete responds with the triplet $(I_{\text{Charlie}}, I_{\text{Alice}}, \mathit{Text}_2)$ encrypted with $K_{\text{Bob,Pete}}$. Upon receiving this message, Bob will assume that it is a valid message and that I_{Charlie} is secure keying material for communicating with Alice. The adversary can now compute the session key that Bob computes.

MSCs of the protocol and the attack are provided in the appendix.

AT5**: Failure of key confirmation for 3-KT-6.** There is a complex attack possible on some implementations of 3-KT-6 that meet three conditions. The 3-KT-6 protocol is a three-pass protocol that transfers two secret keys. After the exchange, a session key can be computed from either or both of these keys. We give the full attack description and the preconditions in the appendix.

4.2 Key Compromise Impersonation (KCI) results

All of the protocols in part 2 use symmetric cryptography and hashing only. Hence, they are necessarily vulnerable to KCI attacks, which is implied by the impossibility result from [3]. All the key transport and public key transport protocols from part 3 are KCI resilient.

The automatic analysis shows that four of the eleven key agreement protocols in part 3 are vulnerable to KCI attacks: 3-KA-1, 3-KA-3, 3-KA-6, and 3-KA-8. Mechanisms 3-KA-1 and 3-KA-3 are variants of the unsigned Diffie-Hellman protocol. Mechanism 3-KA-1 is the static Diffie-Hellman protocol, so as expected the session key is not secret when one of the static keys is known to the adversary. Similarly, 3-KA-3 is a one-pass Diffie-Hellman variant where A's ephemeral and B's static half keys are used: if the adversary gets B's static private key, he can use A's half key to infer the session key. In 3-KA-6, the fact that the input to the key derivation function is only protected by the private key of B allows an adversary who knows B's key to impersonate A in subsequent communications that are only protected with the established session key.

3-KA-8 is derived from one-pass MQV, and is described in Appendix C. The adversary can construct a message with his own injected randomness, send it to B, and use B's key to infer the session key.

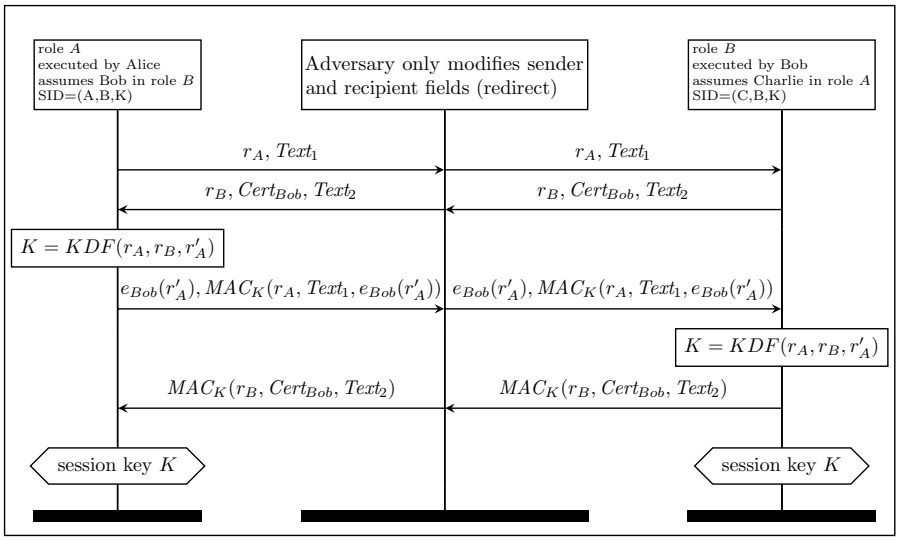

Fig. 4. Unknown key share attack on protocol 3-KA-11. Alice shares a key K with Bob as she expects, but Bob mistakenly assumes he is sharing K with Charlie.

Note that all (unilateral) forward secrecy attacks can be considered to be KCI attacks with late occurrences of long-term key compromise. The converse does not hold: forward secrecy does not imply KCI resilience in general, because KCI attacks may require knowledge of the long-term keys before the attacked session ends. In this specific set of protocols, no KCI attacks require early knowledge of the relevant long-term keys, because the long-term keys are only needed for session key computation. Hence, the protocols in the standard that satisfy forward secrecy are also KCI resilient.

Therefore, in order to obtain KCI resilience guarantees while preserving the required properties, we turn to the Forward Secrecy column in Table 1. We replace each protocol vulnerable to KCI attacks with one that achieves all the already satisfied security guarantees, plus forward secrecy with respect to both entities:

- 3-KA-1 can be replaced by 3-KA-5 (optionally, key confirmation can be enabled),
- 3-KA-3 and 3-KA-6 can be replaced by 3-KA-7 (if entity authentication is required) or 3-KA-5 (otherwise), and
- 3-KA-8 can be replaced by 3-KA-9.

4.3 Unknown Key Share (UKS) results

We used Scyther to analyse all protocols for which key authentication was claimed for UKS attacks. We found that two protocols are vulnerable to UKS for any

implementation, and that some implementations of several other protocols are vulnerable to UKS attacks.

We first explain the unknown key share attack on the 3-KA-11 protocol in detail. A graphical representation is given in Figure 4. In the attack, the adversary does not modify the content of any messages, but only changes the implicit sender/recipient fields. When Alice executes role A with intended partner Bob, she sends out her first message. The adversary modifies the sender field to "Charlie" and forwards the message to Bob. Bob assumes Charlie wants to communicate with him and starts to execute the B role, and sends the response message to Charlie. The adversary redirects this message to Alice. The protocol continues as usual except that the adversary continues to modify the sender fields and redirecting the responses. There is nothing in the messages that allows the entities to check each other's beliefs about the communication partner. In the end, Alice and Bob compute the same key K. Although the adversary does not know this key, Bob will believe that any subsequent messages he receives, which are encrypted or authenticated using K, are coming from Charlie, where in fact they come from Alice, leading to a serious authentication flaw [8, p. 139].

Although 3-KA-11 is derived from the TLS protocol, the TLS protocol is not vulnerable to unknown key share attacks. The reason for this is that the TLS protocol performs confirmation on all previously received messages, which in TLS contain the identities of the sender and recipient. This confirmation will fail if the parties have different views on their communication partners. In some sense, 3-KA-11 can be regarded as a stripped down version of the unilateral TLS-RSA handshake where security-relevant information (the identities of the participants) has been removed.

The second UKS attack is possible on the 2-10 protocol, which suffers from a role-mixup attack in which Alice and Bob both perform the A role and compute the same session key. This can lead to reflection attacks and misinterpretation attacks when the session key is later used to encrypt payloads. In implementations in which entities can perform multiple roles, protocols 2-2, 2-8, 2-9, 2-11, and 2-12 are also vulnerable to UKS attacks.

Fortunately, UKS attacks can be prevented by choosing a key derivation function that includes the identifiers (I_A and I_B) of the involved entities [7, 8]. For example, this is required by the NIST SP-800-56A key derivation [1], which is included in part 3 of the standard. We modelled the use of this KDF and used automated analysis to confirm that this prevents the UKS attacks. Intuitively, including the identities in the KDF ensures that entities that have different beliefs about their intended peers compute different keys, which prevents UKS attacks.

5 Recommendations

We provide three recommendations to improve the ISO/IEC 11770 standard.

1. Improving protocols to achieve the stated properties. Our first recommendation is to make small changes to the protocols to achieve their properties,

if possible. The most straightforward way is to adopt the recommendations made for ISO/IEC 9798 in [4, p. 14]. In particular, we require that

- no cryptographic data must be interchangeable, which can be enforced by including unique tags,
- when optional fields are not used, then they must be set to empty, and
- entities that perform the role of the TTP in the 2-8, 2-9, 2-12 and 2-13 protocols must not perform the A or B role.

Following these recommendations addresses all problems in Table 1 except for the problems of protocol 3-KA-11.

2. Using appropriate key derivation functions. Our second recommendation improves the security of the standard by preventing unknown key share attacks. If the input to the key derivation function includes the identities of the communicating parties, UKS is directly prevented. We therefore recommend making this an explicit requirement. For example, the key derivation function from NIST SP-800-56A [1], which is described in the standard, meets this requirement.

3. Addressing remaining issues with 3-KA-11. 3-KA-11 inherently does not offer perfect forward secrecy or mutual authentication. Switching to a protocol that does, such as mutually authenticated TLS-DHE_RSA, substantially changes the environmental assumptions, including the pre-distribution of keys.

A simpler solution is to adapt the statements made about the protocol. In particular, it should *not* be claimed in the overview table [19, p. 42] that 3-KA-11 achieves implicit key authentication for both entities, that it achieves key confirmation for both entities, or that it achieves MFS. Similarly, the running text [19, p. 26] should not claim that 3-KA-11 achieves mutual explicit key authentication.

6 Related work

In 1998, Horng and Hsu presented an attack on an early version of the 3-KT-6 protocol [16]. Their attack violated key confirmation and showed that the protocol did not offer any strong mutual authentication. In the same year, Mitchell and Yeun proposed a fix [24] that was later introduced in the standard. It essentially involves adding more identifiers to the messages, similar to Lowe's fix of the Needham-Schroeder protocol.

In 2004, Cheng and Comley presented two attacks on a previous version of the 2-12 protocol [10]. Their first attack is a replay attack. The second attack is a type flaw attack based on the possibility of interpreting an identity field as a fresh key. Cheng and Comley presented a fixed version of the protocol. Initially, protocol 2-12 was withdrawn from the standard, and it was later updated in 2008 with a new version that does not suffer from these attacks.

Mathuria and Sriram used Scyther to discover in 2008 [23] more complex type-flaw attacks on protocol 2-13 and on Cheng's and Comley's proposed fixed protocol. The attacks rely on the possibility that complex fields (concatenations,

encryptions) can be interpreted as atomic fields (random values, keys, identities) in some implementations.

In 2010, Chen and Mitchell [9] generalised some of the concepts occurring in this class of type-flaw attacks and presented countermeasures, some of which found their way into later versions of ISO standards.

7 Conclusions

In retrospect, though we found all attacks through automatic analysis, it is clear that some attacks should have been found by manual inspection. This holds especially for 3-KA-11, which is based on TLS's unilaterally authenticated RSA handshake: it is clear that this protocol cannot offer key authentication or confirmation for both parties, since only one party is authenticated.

One way in which standardisation bodies could be more proactive is by being aware of analysis of standards on which they build. For example, many protocols in ISO/IEC 11770 are mentioned to be derived from authentication protocols in ISO/IEC 9798. In 2012, the ISO/IEC 9798 standard was analysed, several problems were identified [4], and it was subsequently updated to fix the identified problems. However, it seems that no attempt was made to determine if the derived protocols inherited these problems. Our analysis shows that this was in fact the case, implying that the attacks on protocols from part 2 could have been identified earlier. Our analysis shows that applying the recommendations for ISO/IEC 9798 as described in [4] to ISO/IEC 11770 would have prevented all issues in Table 1, except for those on 3-KA-11.

The standard currently does not claim resilience to UKS or KCI attacks. One could consider identifying those protocols that achieve these properties or improve the others. For example, all UKS attacks that we found can easily be prevented, at negligible cost, by using key derivation functions that include the identities of the participants. We therefore recommend including the identities in the input to the KDF.

Compared to other security protocol standards, ISO standards have been less analysed in the academic literature. A possible reason for this difference is that people who are not members of the working groups can only access the standards by purchasing the final versions. One possible way to promote the external analysis of ISO standards is to publish early drafts of proposed changes or new standards. Parties that are interested in applying the standards will still need to purchase the final versions to ensure they comply. However, interested parties can freely analyse the designs from the early drafts, which may help identify and prevent problems before the standards are deployed.

References

1. E. Barker, D. Johnson, and M. Smid. NIST SP 800-56: Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised), 2007.

2. D. Basin and C. Cremers. Modeling and analyzing security in the presence of compromising adversaries. In *Computer Security - ESORICS 2010*, volume 6345 of *LNCS*, pages 340–356. Springer, 2010.
3. D. Basin, C. Cremers, and M. Horvat. Actor key compromise: Consequences and countermeasures. In *Proc. of the 27th IEEE Computer Security Foundations Symposium (CSF)*, 2014. To appear.
4. D. Basin, C. Cremers, and S. Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. *Journal of Computer Security*, 21(6):817–846, 2013.
5. D. Basin, C. Cremers, K. Miyazaki, S. Radomirovic, and D. Watanabe. Improving the security of cryptographic protocol standards. *IEEE Security & Privacy*, 2014.
6. S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *IMA Int. Conf.*, pages 30–45, 1997.
7. S. Blake-Wilson and A. Menezes. Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol, 1999.
8. C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Information Security and Cryptography. Springer, 2003.
9. L. Chen and C. J. Mitchell. Parsing ambiguities in authentication and key establishment protocols. *Int. J. Electron. Secur. Digit. Forensic*, 3(1):82–94, 2010.
10. Z. Cheng and R. Comley. Attacks on an ISO/IEC 11770-2 key establishment protocol. *I. J. Network Security*, 3(3):290–295, 2006.
11. C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proc. CAV*, volume 5123 of *LNCS*, pages 414–418. Springer, 2008. Available for download at `http://www.cs.ox.ac.uk/people/cas.cremers/scyther/index.html`.
12. C. Cremers and M. Feltz. Beyond eCK: perfect forward secrecy under actor compromise and ephemeral-key reveal. *Designs, Codes and Cryptography*, pages 1–36, 2013.
13. C. Cremers and S. Mauw. *Operational Semantics and Verification of Security Protocols*. Information Security and Cryptography. Springer, 2012.
14. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) protocol version 1.2. IETF RFC 5246, August 2008.
15. European Payments Council. Guidelines on algorithms usage and key management. Technical report, 2009. EPC342-08 Version 1.1.
16. G. Horng and C.-K. Hsu. Weakness in the Helsinki protocol. *Electronics Letters*, 34:354–355(1), February 1998.
17. International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-2:2008, Information technology – Security techniques – Entity Authentication – Part 2: Mechanisms using symmetric encipherment algorithms, 2008. Third edition.
18. International Organization for Standardization, Genève, Switzerland. ISO/IEC 11770-2:2008, Information technology – Security techniques – Key Management – Part 2: Mechanisms using Symmetric Techniques, 2009. Incorporating corrigendum September 2009.
19. International Organization for Standardization, Genève, Switzerland. ISO/IEC 11770-3:2008, Information technology – Security techniques – Key Management – Part 3: Mechanisms using Asymmetric Techniques, 2009. Incorporating corrigendum September 2009.
20. H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. Cryptology ePrint Archive, Report 2005/176, 2005. `http://eprint.iacr.org/`, retrieved on June 1, 2014.
21. L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28:119–134, 2003.

22. G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 31–44. IEEE, 1997.
23. A. Mathuria and G. Sriram. New attacks on ISO key establishment protocols. *IACR Cryptology ePrint Archive*, 2008:336, 2008.
24. C. J. Mitchell and C. Y. Yeun. Fixing a problem in the Helsinki protocol. *SIGOPS Oper. Syst. Rev.*, 32(4):21–24, Oct. 1998.
25. L. Schmid. Improving the ISO/IEC 11770 standard, 2013. Bachelor's thesis, ETH Zurich, Switzerland.

A Attack on protocol 2-12

The attack on entity authentication claimed for protocol 2-12 is depicted in Figure 5. It depends on the fact that the entity running role A does not check the contents of the message encrypted for entities running roles B and P. In fact, normally such a check is impossible because all three roles are run by different entities. Seeing the payload of that particular message would be the only way for Pete to detect that something is wrong: he could see that the message contains I_{Alice} where I_{Pete} should be. Pete then gladly confirms the session key to Bob in role B, who falsely thinks that Alice just confirmed it.

B AT5: 3-KT-6 attack

There are three preconditions for the attack, which will not be met by most implementations. However, there is nothing in the standard that ensures that they are not met.

The first precondition is that the implementation must not implement any of the optional text fields except for $Text_1$. Second, nonces must be acceptable values for the $Text_1$ field. Third, entities must be able to perform both the A and the B role of the protocol, which occurs in many implementations.

If an implementation meets these conditions, the adversary can attack an instance of the A role by exploiting three instances of the B role. We give a graphical representation in Figure 7 in the appendix. The adversary redirects each sent message into the first receive of a new instance of the B role, and swapping the entity assumptions for the next B instance. This is possible since entities can perform multiple roles, and enabled by the fact that the nonces in messages sent by instances of the B role can be accepted into the $Text_1$ field. After three instances of the B role, the final message is then rerouted back to the final receive of the A role. Consequently, there is no instance of B that agrees with the A instance on *both* of the private keys. Thus, when the session key is computed from both these keys, key confirmation fails for the A instance.

C Key Agreement Mechanism 8 (3-KA-8)

We next give an example from part 3 of the standard, referred to as Key Agreement Mechanism 8. This one-pass mechanism is derived from the one-pass variant of

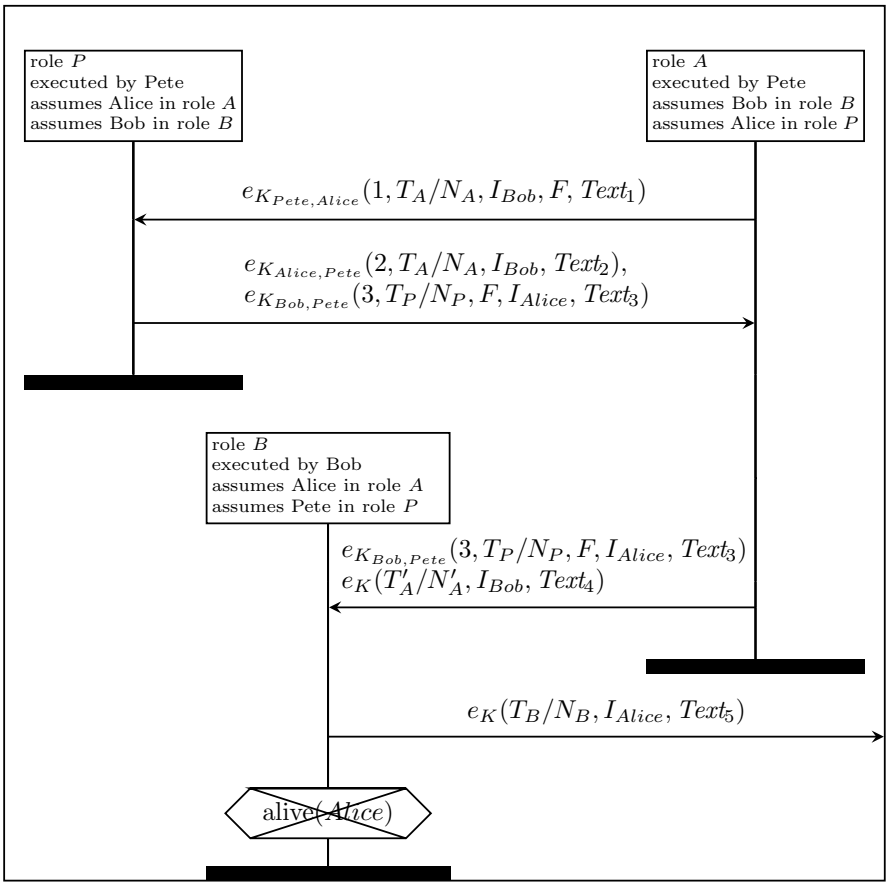

Fig. 5. Entity authentication attack on protocol 2-12 with optional fields

the MQV protocol [21]. It uses an elliptic curve agreed upon by entities A and B to establish a shared secret key. As shown in Figure 10, A generates an ephemeral secret and uses it to transmit to B a public value, modelled as a point on the elliptic curve. B then computes a fresh session key with his static private key and the public values of A (similarly, A uses her secrets and B's public values).

Formally, let H be an elliptic curve over a finite field $\mathbb{F}_q$ and $G \in H$ of prime order n. The parameters H, q, G and n are known to both entities. It is assumed that each entity X has a private key $h_X \in \mathbb{Z}_n^*$, a public key $P_X = h_X G$, and an authenticated copy of the other entity's public key. For the function π defined for every point P on H by $\pi(P) = (P_x \bmod 2^{\lceil \frac{\rho}{2} \rceil}) + 2^{\lceil \frac{\rho}{2} \rceil}$, where P_x is the x-coordinate of P and $\rho = \lceil \log_2 n \rceil$, the protocol is executed as follows: A randomly generates r_A in $\mathbb{Z}_n^*$, computes $r_A G$ and sends it to B. B computes the shared key as $K = (h_B + \pi(P_B)h_B)(r_A G + \pi(r_A G)P_A)$.

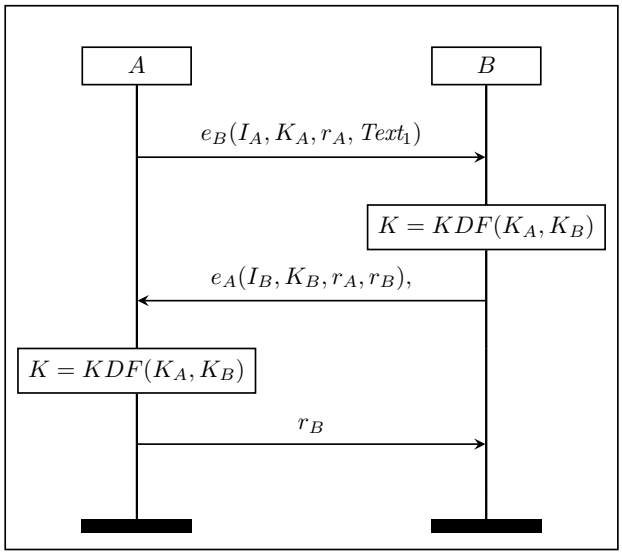

Fig. 6. Protocol 3-KT-6 combined key variant

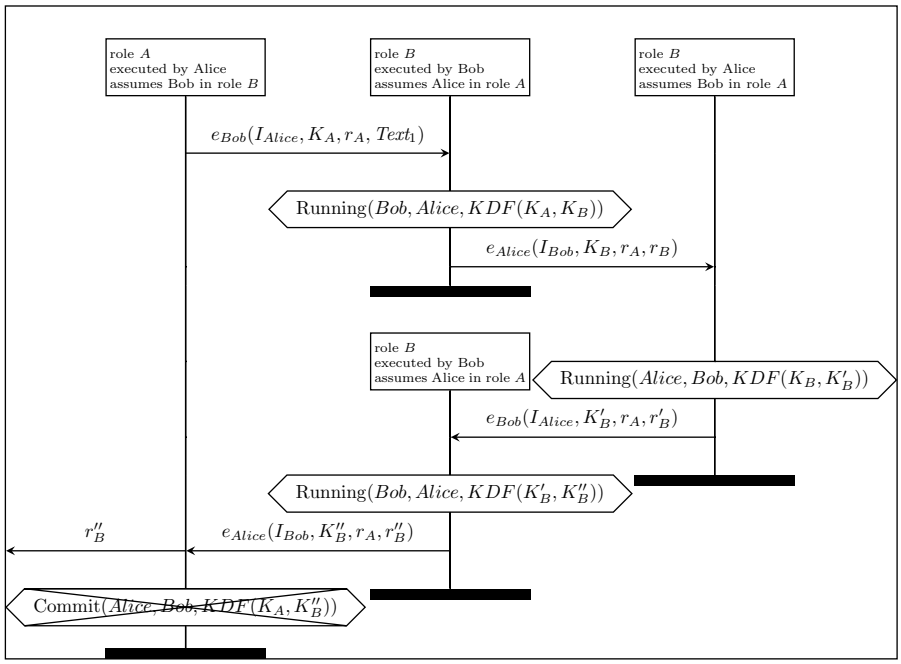

Fig. 7. 3-KT-6 combined key variant key confirmation attack

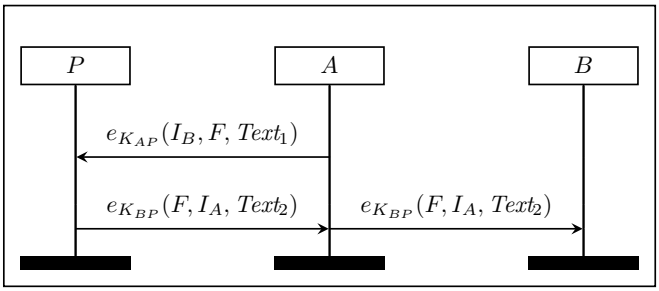

Fig. 8. Protocol 2-11

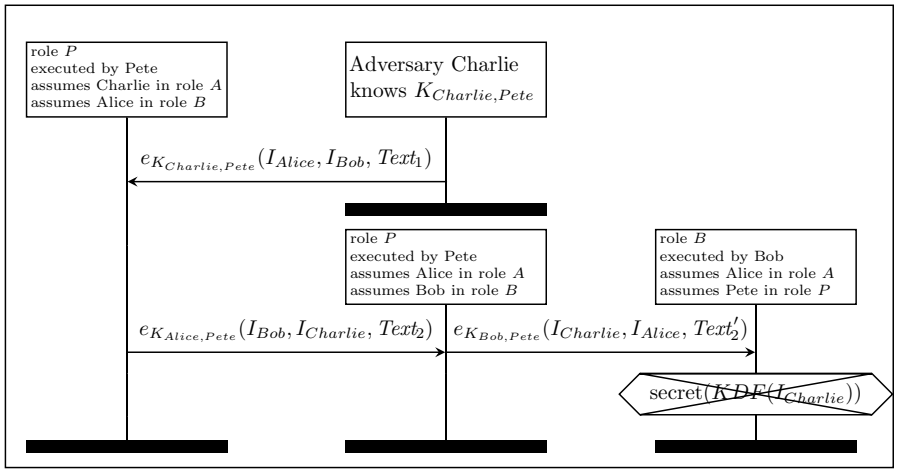

Fig. 9. Protocol 2-11 key authentication attack

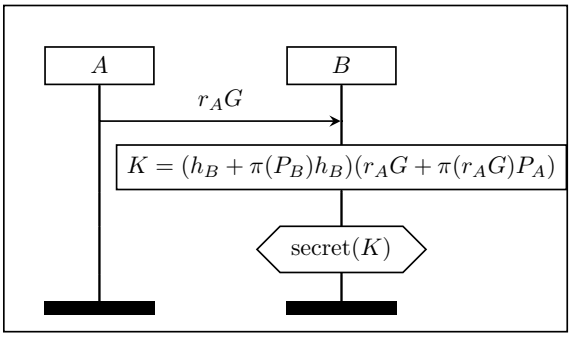

Fig. 10. Protocol 3-KA-8