Provably Repairing the ISO/IEC 9798 Standard for Entity Authentication*

David Basin, Cas Cremers, Simon Meier
Institute of Information Security
ETH Zurich, Switzerland

Abstract

We formally analyze the family of entity authentication protocols defined by the ISO/IEC 9798 standard and find numerous weaknesses, both old and new, including some that violate even the most basic authentication guarantees. We analyze the cause of these weaknesses, propose repaired versions of the protocols, and provide automated, machine-checked proofs of their correctness. From an engineering perspective, we propose two design principles for security protocols that suffice to prevent all the weaknesses. Moreover, we show how modern verification tools can be used for the falsification and certified verification of security standards. Based on our findings, the ISO working group responsible for the ISO/IEC 9798 standard has released an updated version of the standard.

1 Introduction

Entity authentication is a core building block for security in networked systems. In its simplest form, entity authentication boils down to establishing that a party's claimed identity corresponds to its real identity. In practice, stronger guarantees are usually required, such as mutual authentication, agreement among the participating parties on the identities of their peers, or authentication of transmitted data [27, 33].

The ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) jointly provide standards for Information Technology. Their standard 9798 specifies a family of entity authentication protocols. This standard is mandated by numerous other standards that require entity authentication as a building block. Examples include the Guidelines on Algorithms Usage and Key Management [13] by the European Committee for Banking Standards and the ITU-T multimedia standard H.235 [24].

Analysis of previous versions of the ISO/IEC 9798 standard has led to the discovery of several weaknesses [3, 8, 12]. The standard has been revised several

*A preliminary version of this paper appeared in Proceedings of the 1st Conference on Principles of Security and Trust (POST), 2012.

times to address weaknesses and ambiguities, with the latest updates before our analysis stemming from 2010. One may therefore expect that such a mature and pervasive standard is "bullet-proof" and that the protocols satisfy strong, practically relevant, authentication properties.

On request by CRYPTREC, the Cryptography Research and Evaluation Committee set up by the Japanese Government, we formally analyzed the protocols specified in Parts 1–4 of the ISO/IEC 9798 standard from 2010, using the Scyther tool [9]. To our surprise, we not only found that several previously reported weaknesses are still present in the standard, but we also found new weaknesses. In particular, many of the protocols guarantee only weak authentication properties and, under some circumstances, even no authentication at all. For the majority of implementations of the standard where only weak authentication is required, these weaknesses will not lead to security breaches. However, our findings clearly show that the guarantees provided by the protocols are much weaker than might be expected. Moreover, in some cases, additional assumptions are required to ensure even aliveness, the weakest possible form of authentication.

We analyze the shortcomings in the protocols' design and propose repairs. We justify the correctness of our fixes by providing machine-checked proofs of the repaired protocols. These proofs imply the absence of logical errors: the repaired protocols provide strong authentication properties in a Dolev-Yao model, even when multiple protocols from the standard are run in parallel using the same key infrastructure. Consequently, under the assumption of perfect cryptography, the repaired protocols guarantee strong authentication.

To generate the correctness proofs, we first extend the scyther-proof tool [31] to handle bidirectional keys and injective authentication properties. We then use the tool to generate proof scripts that are checked independently by the Isabelle/HOL theorem prover. As input, scyther-proof takes a description of a protocol and its properties and produces a proof in higher-order logic of the protocol's correctness. Both the generation of proof scripts and their verification by Isabelle/HOL are completely automatic. The scyther-proof tool and the protocol models that include the property specifications can be downloaded at [1].

From an engineering perspective, we observe that applying existing principles for constructing cryptographic protocols such as those of Abadi and Needham [2] would not have prevented most of the discovered weaknesses. We therefore additionally propose two design principles in the spirit of [2] whose application would have prevented all of the weaknesses.

Based on our analysis, the ISO/IEC working group responsible for the 9798 standard has released an updated version of the standard, incorporating our proposed fixes.

Summary of Contributions. First, we find previously unreported weaknesses in the 2010 version of the ISO/IEC 9798 standard. Second, we repair this practically relevant standard, and provide machine-checked proofs of the correctness of our repairs. Third, we propose two principles for engineering cryptographic protocols in the spirit of [2] that would have prevented the weaknesses. Finally, our work highlights how modern security protocol analysis tools can be used for falsification and machine-checked verification of security standards.

Organization. In Section 2, we describe the ISO/IEC 9798 standard. In Section 3, we model the protocols and model check them, discovering numerous weaknesses. In Section 4, we analyze the sources of these weaknesses and present two design principles that eliminate them. In Section 5, we explain how we automatically generate machine-checked correctness proofs for these repaired protocols. We describe related work in Section 6 and conclude in Section 7. In Appendix A, we describe our extensions of the SCYTHER-PROOF tool.

2 The ISO/IEC 9798 Standard

2.1 Overview

We give a brief overview of the standard, which specifies a family of entity authentication protocols. We consider here the first four parts of the standard. Part 1 is general and provides background for the other parts. The protocols are divided into three groups. Protocols using symmetric encryption are described in Part 2, those using digital signatures are described in Part 3, and those using cryptographic check functions such as MACs are described in Part 4.

Because the standard has been revised, we also take into account the technical corrigenda and amendments released prior to 2011. Our analysis covers the protocols specified by the following documents. For the first part of the standard, we cover ISO/IEC 9798-1:2010 [21]. For the second part, we cover ISO/IEC 9798-2:2008 [18] and Corrigendum 1 from 2010 [22]. For the third part, we cover ISO/IEC 9798-3:1998 [16], the corrigendum from 2009 [19], and the amendment from 2010 [23]. Finally, for the fourth part, our analysis covers ISO/IEC 9798-4:1999 [17] and the corrigendum from 2009 [20]. In this paper, we write “the standard” to refer to the above documents.

Table 1 lists the 17 associated protocols. For each cryptographic mechanism, there are unilateral and bilateral authentication variants. The number of messages and passes differs among the protocols as well as the communication structure. Some of the protocols also use a trusted third party (TTP).

Note that there is no consistent protocol naming scheme shared by the different parts of the ISO/IEC 9798 standard. The symmetric-key based protocols are referred to in [18] as “mechanism 1”, “mechanism 2”, etc., whereas the protocols in [16, 20, 23] are referred to by their informal name, e.g., “One-pass unilateral authentication”. In this paper we will refer to the protocols consistently by combining the document identifier, e.g., “9798-2” with a number n to identify the n-th protocol in that document. For protocols proposed in an amendment, we continue the numbering from the base document. Hence we refer to the first protocol in [23] as “9798-3-6”. The resulting identifiers are listed in Table 1.

Protocol	Description
Part 2: Symmetric-key Cryptography	
9798-2-1	One-pass unilateral authentication
9798-2-2	Two-pass unilateral authentication
9798-2-3	Two-pass mutual authentication
9798-2-4	Three-pass mutual authentication
9798-2-5	Four-pass with TTP
9798-2-6	Five-pass with TTP
Part 3: Digital Signatures	
9798-3-1	One-pass unilateral authentication
9798-3-2	Two-pass unilateral authentication
9798-3-3	Two-pass mutual authentication
9798-3-4	Three-pass mutual authentication
9798-3-5	Two-pass parallel mutual authentication
9798-3-6	Five-pass mutual authentication with TTP, initiated by A
9798-3-7	Five-pass mutual authentication with TTP, initiated by B
Part 4: Cryptographic Check Functions	
9798-4-1	One-pass unilateral authentication
9798-4-2	Two-pass unilateral authentication
9798-4-3	Two-pass mutual authentication
9798-4-4	Three-pass mutual authentication

Table 1: Protocols specified by Parts 1-4 of the standard.

Most of the protocols are parametrized by the following elements:

- All text fields included in the protocol specification are optional and their purpose is application-dependent.
- Many fields used to ensure uniqueness or freshness may be implemented either by sequence numbers, random numbers, or timestamps.
- Some protocols specify alternative message contents.
- Some identifier fields may be dropped, depending on implementation details.

2.2 Notation

We write $X \| Y$ to denote the concatenation of the bit strings X and Y. We write $\{\!|\, X \,|\!\}_k^{\mathsf{enc}}$ to denote the encryption of X with the symmetric key k and $\{\!|\, X \,|\!\}_k^{\mathsf{sign}}$ to denote the digital signature of X with the signature key k. The application of a cryptographic check function f, keyed with key k, to a message m, is denoted by $f_k(m)$.

In the standard, TVP denotes a Time-Variant Parameter, which may be a sequence number, a random number, or a timestamp. TN denotes a time stamp or sequence number. I_X denotes the identity of agent X. $Text_n$ refers to a text field. These fields are always optional and their use is not specified within the standard. We write K_{AB} to denote the long-term symmetric key shared by A and B. If the key is directional, we assume that A uses K_{AB} to communicate

$$\begin{array}{lll} 1. & A \rightarrow B: & TN_A \parallel Text_2 \parallel f_{K_{AB}}(TN_A \parallel I_B \parallel Text_1) \\ 2. & B \rightarrow A: & TN_B \parallel Text_4 \parallel f_{K_{AB}}(TN_B \parallel I_A \parallel Text_3) \end{array}$$

Figure 1: The 9798-4-3 two-pass mutual authentication protocol using a cryptographic check function.

$$\begin{array}{lll} 1. & A \rightarrow P: & TVP_A \parallel I_B \parallel Text_1 \\ 2. & P \rightarrow A: & Token_{PA} \\ 3. & A \rightarrow B: & Token_{AB} \\ 4. & B \rightarrow A: & Token_{BA} \end{array}$$

where

$$\begin{array}{lll} Token_{PA} & = & Text_4 \parallel \{\!| TVP_A \parallel kab \parallel I_B \parallel Text_3 |\!\}^{\mathsf{enc}}_{K_{AP}} \parallel \{\!| TN_P \parallel kab \parallel I_A \parallel Text_2 |\!\}^{\mathsf{enc}}_{K_{BP}} \\ Token_{AB} & = & Text_6 \parallel \{\!| TN_P \parallel kab \parallel I_A \parallel Text_2 |\!\}^{\mathsf{enc}}_{K_{BP}} \parallel \{\!| TN_A \parallel I_B \parallel Text_5 |\!\}^{\mathsf{enc}}_{kab} \\ Token_{BA} & = & Text_8 \parallel \{\!| TN_B \parallel I_A \parallel Text_7 |\!\}^{\mathsf{enc}}_{kab} \end{array}$$

Figure 2: The 9798-2-5 four pass protocol with TTP using symmetric encryption.

with B and that B uses a second key K_{BA} when sending messages to A. By convention, we use lower case strings for fresh session keys, like kab.

2.3 Protocol Examples

2.3.1 Example 1: 9798-4-3

The 9798-4-3 protocol is a two-pass mutual authentication protocol based on cryptographic check functions, e.g., message authentication codes. Its design, depicted in Figure 1, is similar to two related protocols based on symmetric key encryption (9798-2-3) and digital signatures (9798-3-3).

The initiator starts in role A and sends a message that consists of a time stamp or sequence number TN_A, concatenated with an optional text field and a cryptographic check value. This check value is computed by applying a cryptographic check function to the key shared between A and B and a string consisting of TN_A, B's identity, and optionally a text field $Text_1$. When B receives this message, he computes the cryptographic check himself and compares the result with the received check value. He then computes the response message in a similar way and sends it to A, who checks it.

2.3.2 Example 2: 9798-2-5

Figure 2 depicts the 9798-2-5 protocol, which is based on symmetric-key encryption and uses a Trusted Third Party. A first generates a time-variant parameter TVP_A (which must be non-repeating), and sends it with B's identity I_B and optionally a text field to the trusted party P. P then generates a fresh session key kab and computes $Token_{PA}$, which essentially consists of two encrypted copies of the key, using the long-term shared keys between P and A, and P and

B, respectively. Upon receiving $Token_{PA}$, A decrypts the first part to retrieve the session key, and uses the second part to construct $Token_{AB}$. Finally, B retrieves the session key from this message and sends its authentication message $Token_{BA}$ to A.

2.4 Optional Fields and Variants

There are variants for each protocol listed in Table 1. Each protocol contains *text fields*, whose purpose is not specified, and which may be omitted, giving rise to another protocol variant. As can be seen in the previous examples, some of these text fields are plaintext, whereas others are within the scope of cryptographic operations (i.e., signed, encrypted, or cryptographically checked). Note that the standard does not provide a rationale for choosing among these options.

In setups where symmetric keys are used, it is common that if Alice wants to communicate with Bob, she will use their shared key, which is the same key that Bob would use to communicate with Alice. Such keys are called *bidirectional*. Alternatively one can use *unidirectional* keys where each pair of agents shares two symmetric keys, one for each direction. In this case $K_{\text{Alice,Bob}}$ and $K_{\text{Bob,Alice}}$ are different. For some protocols that employ symmetric keys, the standard specifies that if unidirectional keys are used, some identity fields may be omitted from the encrypted (or checked) payload. This yields another variant.

The two protocols 9798-3-6 and 9798-3-7 both provide two options for the tokens included in their messages, giving rise to further variants. Note that in Section 5 we verify corrected versions of all 17 protocols in Table 1, taking all variants into account.

2.5 Threat Model and Security Properties

The ISO/IEC 9798 standard neither specifies a threat model nor defines the security properties that the protocols should satisfy. Instead, the introduction of ISO/IEC 9798-1 simply states that the protocols should satisfy mutual or unilateral authentication. Furthermore, the following attacks are mentioned as being relevant: man-in-the-middle attacks, replay attacks, reflection attacks, and forced-delay attacks. We note that the standard does not explicitly claim that any of the protocols are resilient against the above attacks.

3 Protocol Analysis

3.1 Background

We analyze the protocols in the standard with respect to three standard authentication properties: *aliveness*, *non-injective agreement*, and *injective agreement*, as defined in Lowe's hierarchy of authentication properties [27]. We recall the informal definitions from [27]. For the formal definitions we refer the reader to [1].

The weakest property we consider is *aliveness*, which states that an apparent communication partner has performed at least some protocol action.

Definition 1 (Aliveness [27])**.** *A protocol guarantees to an initiator A* aliveness *of another agent B if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol.*

Note that this definition does not require the responder to be aware that A is trying to authenticate him. Furthermore, most authentication protocols, including all the protocols in the ISO/IEC 9798 standard, exchange data between the agents, for example, time variant parameters and text fields. After a successful run of the protocol, we would like to be sure that the view of the two agents agrees on the exchanged data, i.e., it should not be the case that A assumes that $Text_1$ is `yes` whereas B assumes that $Text_1$ is `no`.

The notion of *non-injective agreement* implies aliveness and captures these additional requirements.

Definition 2 (Non-injective agreement [27])**.** *A protocol guarantees to an initiator A* non-injective agreement *with a responder B on a set of data items ds (where ds is a set of terms appearing in the protocol description) if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the terms in ds.*

Non-injective agreement is a strong form of authentication. However, it does not exclude replay attacks, where an attacker replays messages from a previous session to successfully complete the protocol any number of times with A. To prevent such attacks, we strengthen non-injective agreement to injective agreement, which additionally requires that if A successfully completes the protocol n times with B, then B has been running the protocol at least n times.

Definition 3 (Injective agreement [27])**.** *A protocol guarantees to an initiator A* agreement *with a responder B on a set of data items ds if, whenever A (acting as initiator) completes a run of the protocol, apparently with responder B, then B has previously been running the protocol, apparently with A, and B was acting as responder in his run, and the two agents agreed on the data values corresponding to all the terms in ds, and each such run of A corresponds to a* unique *run of B.*

To perform our analysis, we use two different analysis tools. In this section, we use the SCYTHER tool [9] to find attacks on the ISO/IEC 9798 protocols. In Section 5, we will use the related SCYTHER-PROOF tool [31] to generate machine-checked proofs of the corrected versions.

SCYTHER performs an automatic analysis of security protocols in a Dolev-Yao style model, for an unbounded number of instances. It is very efficient at both verification and falsification, in particular for authentication protocols such as those considered here.

Protocol	Role	Violated property	With	Data	Assumptions
9798-2-3	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-2-3	B	Non-injective agreement	A	TN_A, $Text_1$	
9798-2-3 (UDK)	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-2-3 (UDK)	B	Non-injective agreement	A	TN_A, $Text_1$	
9798-2-5	A	Aliveness	B		Alice-talks-to-Alice
9798-2-5	B	Aliveness	A		
9798-2-6	A	Aliveness	B		
9798-2-6	B	Aliveness	A		
9798-3-3	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-3-3	B	Non-injective agreement	A	TN_A, $Text_1$	
9798-3-7-1	A	Non-injective agreement	B	R_A, R_B, $Text_8$	Type-flaw
9798-4-3	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-4-3	B	Non-injective agreement	A	TN_A, $Text_1$	
9798-4-3 (UDK)	A	Non-injective agreement	B	TN_B, $Text_3$	
9798-4-3 (UDK)	B	Non-injective agreement	A	TN_A, $Text_1$	

Table 2: Overview of attacks found. (UDK) indicates the protocol variants where unidirectional keys are used.

3.2 Analysis results

Using Scyther, we performed protocol analysis with respect to aliveness and agreement. Our analysis reveals that the majority of the protocols in the standard ensure agreement on the exchanged data items. However, we also found attacks on five protocols and two protocol variants. These attacks fall into the following categories: role-mixup attacks, type flaw attacks, multiple-role TTP attacks, and reflection attacks. In all cases, when an agent finishes his role of the protocol, the protocol has not been executed as expected. This can lead the agent to proceed with false assumptions about the state of the other involved agents.

In Table 2 we list the attacks we found using Scyther. The table lists the protocols, the properties violated, and any additional assumptions required for the attacks. We have omitted in the table all attacks that are entailed by the attacks listed. For example, since 9798-2-5 does not satisfy aliveness from B's perspective, it also does not satisfy any stronger properties such as non-injective agreement. We now describe the classes of attacks in more detail.

3.3 Role-Mixup Attacks

Some protocols are vulnerable to a *role-mixup attack* in which an agent's assumptions about another agent's role are wrong. The two agreement properties require that when Alice finishes her role apparently with Bob, then Alice and Bob not only agree on the exchanged data, but additionally Alice can be sure that Bob was performing in the intended role. Protocols vulnerable to role-mixup attacks therefore violate agreement.

Figure 3 on the following page shows an example of a role-mixup attack on

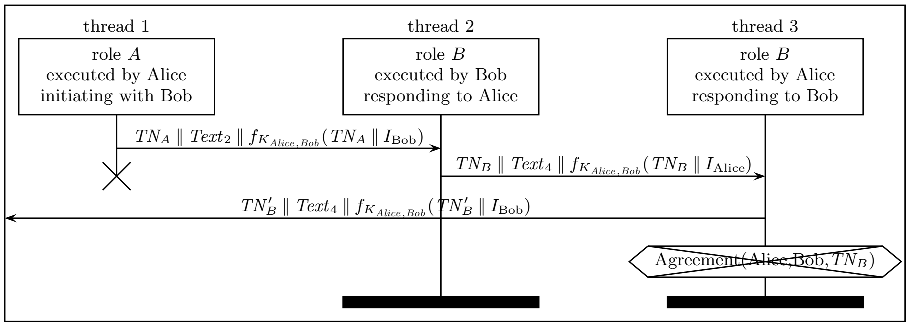
thread 1
thread 2
thread 3
role A executed by Alice initiating with Bob
role B executed by Bob responding to Alice
role B executed by Alice responding to Bob
$TN_A \parallel Text_2 \parallel f_{K_{Alice,Bob}}(TN_A \parallel I_{\text{Bob}})$
$TN_B \parallel Text_4 \parallel f_{K_{Alice,Bob}}(TN_B \parallel I_{\text{Alice}})$
$TN'_B \parallel Text_4 \parallel f_{K_{Alice,Bob}}(TN'_B \parallel I_{\text{Bob}})$
Agreement(Alice,Bob,TN_B)

Figure 3: Role-mixup attack on 9798-4-3: when Alice finishes thread 3 she wrongly assumes that Bob was performing the A role.

the 9798-4-3 protocol from Figure 1. Agents perform actions such as sending and receiving messages, resulting in message transmissions represented by horizontal arrows. Actions are executed within threads, represented by vertical lines. The box at the top of each thread denotes the parameters involved in the thread's creation. Claims of security properties are denoted by hexagons and a crossed-out hexagon denotes that the claimed property is violated.

In this attack, the adversary uses a message from Bob in role B (thread 2) to trick Alice in role B (thread 3) into thinking that Bob is executing role A and is trying to initiate a session with her. However, Bob (thread 2) is replying to a message from Alice in role A (thread 1), and is executing role B. The adversary thereby tricks Alice into thinking that Bob is in a different state than he actually is.

Additionally, when the optional text fields $Text_1$ and $Text_3$ are used, the role-mixup attack also violates the agreement property with respect to these fields: Alice will end the protocol believing that the optional field data she receives from Bob was intended as $Text_1$, whereas Bob actually sent this data in the $Text_3$ field. Depending on the use of these fields, this can constitute a serious security problem. Note that exploiting these attacks, as well as the other attacks described below, does not require "breaking" cryptography. Rather, the adversary exploits similarities among messages and the willingness of agents to engage in the protocol.

Summarizing, we found role-mixup attacks on the following protocols: 9798-2-3 with bidirectional or unidirectional keys, 9798-2-5, 9798-3-3, and 9798-4-3 with bidirectional or unidirectional keys.

3.4 Type Flaw Attacks

Some protocol implementations are vulnerable to *type flaw attacks* where data of one type is parsed as data of another type. Consider, for example, an implementation where agent names are encoded into bit-fields of length n, which

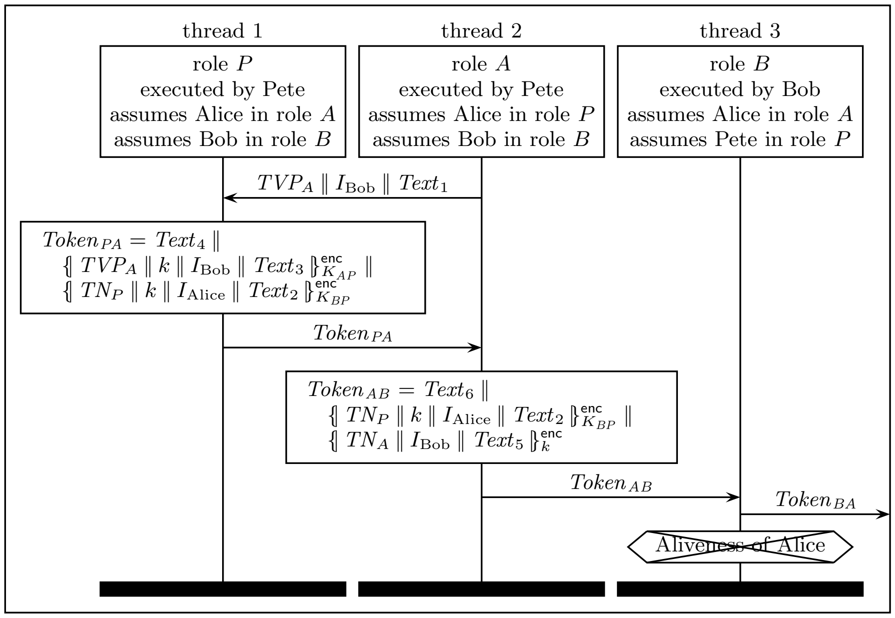
thread 1
thread 2
thread 3
role P executed by Pete assumes Alice in role A assumes Bob in role B
role A executed by Pete assumes Alice in role P assumes Bob in role B
role B executed by Bob assumes Alice in role A assumes Pete in role P
$TVP_A \parallel I_{\mathrm{Bob}} \parallel Text_1$
$Token_{PA} = Text_4 \parallel \{\!| TVP_A \parallel k \parallel I_{\mathrm{Bob}} \parallel Text_3 |\!\}^{\mathsf{enc}}_{K_{AP}} \parallel \{\!| TN_P \parallel k \parallel I_{\mathrm{Alice}} \parallel Text_2 |\!\}^{\mathsf{enc}}_{K_{BP}}$
$Token_{PA}$
$Token_{AB} = Text_6 \parallel \{\!| TN_P \parallel k \parallel I_{\mathrm{Alice}} \parallel Text_2 |\!\}^{\mathsf{enc}}_{K_{BP}} \parallel \{\!| TN_A \parallel I_{\mathrm{Bob}} \parallel Text_5 |\!\}^{\mathsf{enc}}_{k}$
$Token_{AB}$
$Token_{BA}$
~~Aliveness of Alice~~

Figure 4: Attack on the 9798-2-5 protocol where the trusted third party Pete performs both the P role and the A role. The assumptions of thread 1 and 3 agree. Bob wrongly concludes that Alice is alive.

is also the length of the bit-fields representing nonces. It may then happen that an agent who expects to receive a value that it has not seen before (such as a nonce generated in another role) accepts a bit string that was intended to represent an agent name.

Scyther finds such an attack on the 9798-3-7 protocol, also referred to as "Five pass authentication (initiated by B)" [23, p. 4]. In the attack, both (agent) Alice and (trusted party) Terence mistakenly accept the bit string corresponding to the agent name "Alice" as a nonce.

3.5 Attacks Involving TTPs that Perform Multiple Roles

Another class of attacks occurs when parties can perform both the role of the trusted third party and another role. This scenario is not excluded by the standard.

In Figure 4 we show an attack on 9798-2-5, from Figure 2. The attack closely follows a regular protocol execution. In particular, threads 1 and 3 perform the protocol as expected. The problem is thread 2. Threads 1 and 3 assume that the participating agents are Alice (in the A role), Bob (in the B role), and Pete (in the P role). From their point of view, Alice should be executing thread 2. Instead, thread 2 is executed by Pete, under the assumption that Alice

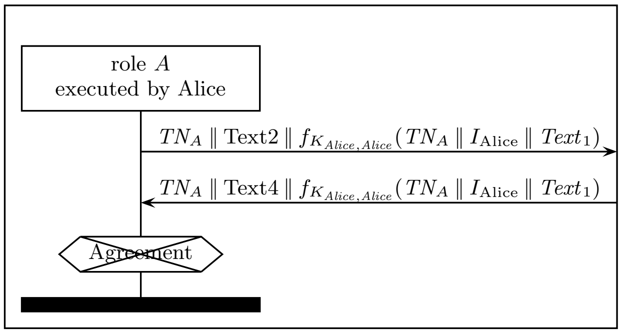
role A
executed by Alice
$TN_A \parallel \text{Text2} \parallel f_{K_{Alice,Alice}}(TN_A \parallel I_{\text{Alice}} \parallel Text_1)$
$TN_A \parallel \text{Text4} \parallel f_{K_{Alice,Alice}}(TN_A \parallel I_{\text{Alice}} \parallel Text_1)$
Agreement

Figure 5: Reflection attack on 9798-4-3.

is performing the P role. Thread 2 receives only a single message in the attack, which is $Token_{PA}$. Because the long-term keys are bidirectional, thread 2 cannot determine from the part of the message encrypted with K_{AP} that thread 1 has different assumptions. Thread 2 just forwards the other encrypted message part blindly to thread 3, as Thread 2 does not expect to be able to decrypt this part. Finally, thread 3 cannot detect the confusion between Alice and Pete, because the information in $Token_{AB}$ that was added by thread 2 only includes Bob's name.

This attack violates aliveness because Bob was apparently talking to Alice, but Alice never performed any action. Hence, in this case, the protocol fails to achieve even the weakest form of authentication.

Summarizing, we found attacks involving TTPs that perform multiple roles on the 9798-2-5 and 9798-2-6 protocol.

3.6 Reflection Attacks

Reflection attacks occur when agents may start sessions communicating with themselves, a so-called *Alice-talks-to-Alice* scenario. The feasibility and relevance of this scenario depends on the application and its internal checks. For example, it may be that Alice uses the same private key on different physical machines, and uses an Alice-talks-to-Alice scenario to communicate from one of her machines to another.

If an Alice-talks-to-Alice scenario is possible, some protocols are vulnerable to reflection attacks. The Message Sequence Chart in Figure 5 shows an example for the 9798-4-3 protocol from Figure 1. In this attack, the adversary (not depicted) reflects the time stamp (or nonce) and cryptographic check value from the message sent by Alice back to the same thread, while prepending the message Text4.

The attack violates both agreement properties, because even though the apparent partner (Alice) has performed an action, she did not perform the expected role. Furthermore, this attack violates one of the main requirements explicitly stated in the ISO/IEC 9798-1 introduction, namely absence of reflection attacks.

Summarizing, we found reflection attacks on the following protocols: 9798-2-3 with bidirectional or unidirectional keys, 9798-2-5, 9798-3-3, and 9798-4-3 with bidirectional or unidirectional keys.

4 Repairing the Protocols

4.1 Root Causes of the Problems

We identify two shortcomings in the design of the protocols, which together account for all of the weaknesses detected.

1) Cryptographic Message Elements May Be Accepted at Wrong Positions. In both the reflection and role-mixup attacks, the messages that are received in a particular step of a role were not intended to be received at that position. By design, the protocol messages are all similar in structure, making it impossible to determine at which point in the protocols the messages were intended to be received.

As a concrete example, consider the reflection attack in Figure 5. Here, the message sent in the protocol's first step can be accepted in the second step, even though this is not part of the intended message flow.

2) Underspecification of the Involved Identities and their Roles. As noted, the symmetric-key based protocols with a TTP, 9798-2-5 and 9798-2-6, do not explicitly state that entities performing the TTP role cannot perform other roles. Hence it is consistent with the standard for Alice to perform both the role of the TTP as well as role A or B. In these cases, the aliveness of the partner cannot be guaranteed, as explained in Section 3.5. The source of this problem is that one cannot infer from each message which identity is associated with which role.

For example, consider the first encrypted component from the third message in the 9798-2-5 protocol with bidirectional keys, in Figure 2.

$$\{\!| \, TN_P \parallel kab \parallel I_A \parallel Text_2 \, |\!\}^{\mathsf{enc}}_{K_{BP}}$$

This message implicitly includes the identities of the three involved agents: the identity of the agent performing the A role is explicitly included in the encryption, and the shared long-term key K_{BP} implicitly associates the message to the key shared between the agent performing the B and P roles. However, because the key is bidirectional, the recipient cannot determine which of the two agents (say, Bob and Pete) sharing the key performed which role: either Bob performed the B role and Pete the P role, or vice versa. Our attack exploits this ambiguity.

4.2 Associated Design Principles

To remedy these problems, we propose two principles for designing security protocols. These principles are in the spirit of Abadi and Needham's eleven principles for prudent engineering practice for cryptographic protocols [2].

Our first principle concerns tagging.

> **Principle: positional tagging.** Cryptographic message components should contain information that uniquely identifies their origin. In particular, the information should identify the protocol, the protocol variant, the message number, and the particular position within the message, from which the component was sent.

This is similar in spirit to Abadi and Needham's Principle 1, which states that *"Every message should say what it means: the interpretation of the message should depend only on its content. It should be possible to write down a straightforward English sentence describing the content — though if there is a suitable formalism available that is good too."* Our principle does not depend on the meaning of the message as intended by the protocol's designer. Instead, it is based solely on the structure of the protocol messages and their acceptance conditions.

Note that we consider protocols with optional fields to consist of multiple protocol variants. Thus, a message component where fields are omitted should contain information to uniquely determine which fields were omitted.

Our second principle is a stricter version of Abadi and Needham's Principle 3.

> **Principle: inclusion of identities and their roles.** Each cryptographic message component should include information about the identities of all the agents involved in the protocol run and their roles, unless there is a compelling reason to do otherwise.

A compelling reason to leave out identity information might be that *identity hiding* is a requirement, i.e., Alice wants to hide that she is communicating with Bob. However, such requirements can usually be met by suitably encrypting identity information.

Contrast this principle with the Abadi and Needham's Principle 3: *"If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message."* The original principle is only invoked when the identity is essential. Instead, we propose to always include information on all the identities as well as their roles. This principle would have prevented attacks on many protocols, including the attacks on the 9798-2-5 and 9798-2-6 protocols, as well as the Needham-Schroeder protocol [26].

For protocols with a fixed number of roles, this principle can be implemented by including an ordered sequence of the identities involved in each cryptographic message component, such that the role of an agent can be inferred from its position in the sequence.

Amendment 1:

The cryptographic data (encryptions, signatures, cryptographic check values) used at different places in the protocols must not be interchangeable. This may be enforced by including in each encryption/signature/CCF value the following two elements:

1. The object identifier as specified in Annex B [23, p. 6], in particular identifying the ISO standard, the part number, and the authentication mechanism.
2. For protocols that contain more than one cryptographic data element, each such element must contain a constant that uniquely identifies the position of the element within the protocol.

The recipient of a message must verify that the object identifier and the position identifiers are as expected. The cryptographic keys used by implementations of the ISO/IEC 9798 protocols must be distinct from the keys used by other protocols.

Amendment 2:

When optional fields, such as optional identities or optional text fields, are not used then they must be set to empty. In particular, the message encoding must ensure that the concatenation of a field and an empty optional field is uniquely parsed as a concatenation. This can be achieved by implementing optional fields as variable-length fields. If the optional field is not used, the length of the field is set to zero.

Amendment 3:

Entities that perform the role of the TTP in the 9798-2-5 and 9798-2-6 protocols must not perform the A or B role.

Figure 6: Proposed amendments to the ISO/IEC 9798 standard.

4.3 Proposed Modifications to the Standard

All the previously mentioned attacks on the ISO/IEC 9798 can be prevented by applying the previous two principles. Specifically, we propose three modifications to the ISO standard, shown in Figure 6. The first two directly follow from the principles and the third modification restricts the use of two protocols in the standard. Afterwards we give an example of a repaired protocol.

Note that in this section we only give informal arguments why our modifications prevent the attacks. In Section 5, we provide machine-checked proofs that this is the case.

4.3.1 Ensuring that Cryptographic Data Cannot Be Accepted at the Wrong Point

We factor the first principle (positional tagging) into two parts and propose two corresponding amendments to the standard. First, we explicitly include in each cryptographic message component constants that uniquely identify the protocol, the message number, and the position within the message. Second, we ensure that protocol variants can be uniquely determined from the messages.

In our first amendment, shown in Figure 6, we implement unique protocol identifiers by using an existing part of the standard: the object identifier from Annex B of the standard, which specifies an encoding of a unique protocol identifier. We also introduce a unique identifier for the position of the cryptographic component within the protocol.

Amendment 1 prevents all reflection attacks because messages sent in one step will no longer be accepted in another step. Furthermore, it prevents all role-mixup attacks, because the unique constants in the messages uniquely determine the sending role. The final part of Amendment 1, stating that cryptographic keys should not be used by other protocols, provides distinctness of cryptographic messages with respect to any other protocols.

Our second amendment, also shown in Figure 6, ensures that the protocol variant (determined by the omission of optional fields) can be uniquely determined from the messages. We implement this by requiring that the recipient of a message can uniquely determine which optional fields, if any, were omitted.

To see why protocols with omitted optional fields must be considered as protocol variants, consider the following example: Consider a protocol in which a message contains the sequence $X \parallel I_A \parallel \mathit{Text}$, where I_A is an identity field that may be dropped (e.g., with unidirectional keys) and *Text* is an optional text field. Then, it may be the case that in one protocol variant, an agent expects a message of the form $X \parallel I_A$, whereas the other implementation expects a message of the form $X \parallel \mathit{Text}$. The interaction between the two interpretations can result in attacks. For example, the text field is used to insert a fake agent identity, or an agent identity is wrongly assumed to be the content of the text field.

If we follow the second amendment in the above example, the expected messages correspond to $X \parallel I_A \parallel \perp$ and $X \parallel \perp \parallel \mathit{Text}$, respectively, where $\perp$ denotes the zero-length field. Because the ISO/IEC 9798 standard requires that concatenated fields can be uniquely decomposed into their constituent parts, misinterpretation of the fields is no longer possible.

Together, Amendments 1 and 2 implement our first principle.

4.3.2 Addressing Underspecification of the Role Played by Agents

Almost all the protocols in the ISO/IEC 9798 standard already adhere to our second principle: unique identification of the involved parties and their roles. However, all protocols in the standard conform to Abadi and Needham's third principle because the messages uniquely determine the identities of all involved parties.

$$\begin{array}{lll} 1. & A \rightarrow B: & TN_A \parallel Text_2 \parallel f_{K_{AB}}(\textbf{``9798-4-3 ccf1''} \parallel TN_A \parallel I_B \parallel \bot) \\ 2. & B \rightarrow A: & TN_B \parallel Text_4 \parallel f_{K_{AB}}(\textbf{``9798-4-3 ccf2''} \parallel TN_B \parallel I_A \parallel Text_3) \end{array}$$

Figure 7: Repaired version of the 9798-4-3 protocol with omitted $Text_1$ field.

There are two protocols in the standard that conform to Abadi and Needham's principle but not to our second principle: 9798-2-5 and 9798-2-6. For example, the messages of the 9798-2-5 protocol identify all parties involved by their association to the long-term keys. However they do not conform to our second principle because the roles of the involved identities cannot be uniquely determined from the messages. This is the underlying reason why, as currently specified, the 9798-2-5 and 9798-2-6 protocols do not guarantee the aliveness of the partner, as shown in Section 3.5.

This problem can be solved by applying our principle, i.e., including the identities of all three participants in each message, so that their roles can be uniquely determined. This is an acceptable solution and we have formally verified it using the method of Section 5. However, from our analysis with Scyther, we observe that the attacks require that the Trusted Third Party also performs other roles (A or B). Under the assumption that in actual applications a TTP will, by definition, not perform the A or B role, the protocols already provide strong authentication. Thus, an alternative solution is to leave the protocols unchanged and make this restriction explicit. This results in more streamlined protocols and also requires minimal changes to the standard. This is the proposal made in Amendment 3 in Figure 6. We have also verified this solution as described in Section 5.

4.3.3 Repaired Protocols

Applying our principles and proposed amendments to the standard, we obtain repaired versions of the protocols. As an example, we show the repaired version of the 9798-4-3 protocol with bidirectional keys in Figure 7. In this example, the $Text_1$ field is not used, and is therefore replaced by $\bot$. Each use of a cryptographic primitive (in this case the cryptographic check function) includes a constant that uniquely identifies the protocol (`9798-4-3`) as well as the position within the protocol specification (`ccf1` and `ccf2`).

5 Proving the Correctness of the Repaired Protocols

The principles and amendments proposed in the previous section are motivated by our analysis of the attacks and the protocol features that enable them. Consequently, the principles and amendments are designed to eliminate these undesired behaviors. Such principles are useful guides for protocol designers but their application does not strictly provide any security guarantees. In order to ensure that the repaired protocols actually have the intended strong authentication properties, we provide machine-checked correctness proofs.

5.1 Generating machine-checked correctness proofs

We use a version [1] of the SCYTHER-PROOF tool [31] to generate proofs of the authentication properties. Given a description of a protocol and its security properties, the tool generates a proof script that is afterwards automatically checked by the Isabelle/HOL theorem prover [34]. If the prover succeeds, then the protocol is verified with respect to a symbolic, Dolev-Yao model. Two extensions to the tool were required to verify our repaired protocols. We added support for bidirectional symmetric long-term keys and support for injective authentication properties. We describe these two extensions in Appendix A.

The proofs generated by SCYTHER-PROOF are based on a security protocol verification theory, which is formally derived in Isabelle/HOL from the formalization of a symbolic, Dolev-Yao model. This theory provides a sound way to perform finite case distinctions on the possible sources of messages that are known to the intruder in the context of a given protocol. We will illustrate how these case distinctions are used to prove security properties in Example 1 on page 18.

The tool searches for the proofs with the fewest number of these case distinctions. For example, in the proofs of our repaired protocols, two case distinctions are required on average to prove a security property. Therefore, the generated proof scripts are amenable to human inspection and understanding. To simplify the task of understanding how the proofs work and, hence, why the protocol is correct, the tool also generates proof outlines. These consist of a representation of the security property proven and a tree of case distinctions constituting the proof.

5.2 Parallel composition

We verify the properties of each protocol when composed in parallel with all other protocols that use the same cryptographic primitives and the same keys. Note that in the corresponding proofs, the case distinctions on the source of messages known to the intruder range over the roles of each protocol in the protocol group. Despite the substantial increase in the scope of these case distinctions, the proof structure of the composed protocols is the same as for the individual protocols, as all additional cases are always trivially discharged due to tagging: cryptographic components received by a thread of one protocol contain tags that do not match with the tags in messages produced by roles from other protocols.

The verification of the parallel composition of all protocols that use the same cryptographic primitives and the same keys implies that all protocols composed in parallel satisfy their security properties. This follows from the disjoint encryption theorem [14], which implies that protocols that do not share keying material can be safely composed.

```
Repaired version of 9798-4-3
protocol isoiec_9798_4_3_bdkey_repaired
{
  leak_A. A ->  : TNa
  leak_B. B ->  : TNb

  text_1.   -> A: Text1, Text2
       1. A -> B: A, B, TNa, Text2, Text1, h(('CCF', k[A,B]), ('isoiec_9798_4_3_ccf_1',
                  TNa, B, Text1))
  text_2.   -> B: Text3, Text4
       2. B -> A: B, A, TNb, Text4, Text3, h(('CCF', k[A,B]), ('isoiec_9798_4_3_ccf_2',
                  TNb, A, Text3))
}
properties (of isoiec_9798_4_3_bdkey_repaired)
  A_non_injective_agreement: niagree(A_2[A,B,TNb,Text3] -> B_2[A,B,TNb,Text3], {A, B})
  B_non_injective_agreement: niagree(B_1[A,B,TNa,Text1] -> A_1[A,B,TNa,Text1], {A, B})
```

Figure 8: Example of input provided to the SCYTHER-PROOF tool.

5.3 Details of the proven properties

In Tables 3, 4, and 5 we provide details of the authentication properties proven using SCYTHER-PROOF for each repaired protocol and its variants. For example, for the 9798-2-1 protocol with bidirectional keys, we have that if Bob successfully completes a thread of role B, apparently with Alice, then there must be a thread of Alice performing role A that agrees on the agent names, TN_A, and $Text_1$.

5.3.1 Non-injective agreement

For each repaired protocol, we use SCYTHER-PROOF to prove that it satisfies at least non-injective agreement on all data items within the scope of cryptographic operators in the presence of a Dolev-Yao intruder. Moreover, we prove that this holds even when all the protocols from the standard are executed in parallel using the same key infrastructure, provided that the set of bidirectional keys is disjoint from the set of unidirectional keys. As the content of text fields is underspecified in the standard, we assume that the intruder chooses their content immediately before they are sent. We model timestamps and sequence numbers by random numbers that are public and chosen at the beginning of the execution of a role.

Example 1. Figure 8 specifies our model of the repaired 9798-4-3 protocol with bidirectional keys in the input language of the SCYTHER-PROOF tool. The `leak_A` and the `leak_B` steps model that the timestamps (represented here as randomly generated numbers) are publicly known by leaking them to the intruder. We model that the contents of the `Text_1` through `Text_4` fields are chosen by the intruder by defining them as variables that receive their content from the network, and therefore from the intruder. We model the cryptographic check function by the hash function h.

Figure 9 shows the proof outline for non-injective agreement for the `A`-role of this protocol, which is automatically generated by our tool. We have taken minor liberties in its presentation to improve readability. In the figure, `#i` is a

```
 1  property (of isoiec_9798_4_3_bdkey_repaired)
 2    A_non_injective_agreement:
 3    "All #i.
 4      role(#i) = isoiec_9798_4_3_bdkey_repaired_A &
 5      step(#i, isoiec_9798_4_3_bdkey_repaired_A_2) &
 6      uncompromised( A#i, B#i )
 7      ==> (Ex #j. role(#j) = isoiec_9798_4_3_bdkey_repaired_B &
 8                  step(#j, isoiec_9798_4_3_bdkey_repaired_B_2) &
 9                  (A#j, B#j, TNb#j, Text3#j) = (A#i, B#i, TNb#i, Text3#i)) "
10  sources( h(('CCF', k[A#i,B#i]), ('isoiec_9798_4_3_ccf_2', TNb#i, A#i, Text3#i)) )
11    case fake
12    contradicts secrecy of k[A#i,B#i]
13  next
14    case (isoiec_9798_4_3_bdkey_B_2_repaired_hash #k)
15    tautology
16  qed
```

Figure 9: Example proof outline automatically produced by the SCYTHER-PROOF tool.

symbolic variable denoting some thread i and `A#i` is the value of the variable `A` in the thread i. Lines 3–9 state the security property: for each thread `#i` that executes the `A`-role and has executed its Step `2` with uncompromised (honest) agents `A#i` and `B#i`, there exists some thread `#j` that executed Step `2` of the `B`-role and thread `#j` agrees with thread `#i` on the values of `A`, `B`, `TNb`, and `Text3`.

The proof proceeds by observing that thread `#i` executed Step `2` of the `A`-role. Therefore, thread `#i` received the hash in line 10 from the network, and therefore the intruder knows this hash. A case distinction on the sources of this hash results in two cases: (1) the intruder could have constructed (faked) this hash by himself or (2) the intruder could have learned this hash from some thread `#k` that sent it in Step `2` of the `B`-role. There are no other cases because all other hashes have different tags. Case 1 is contradictory because the intruder does not know the long-term key shared between the two uncompromised agents `A#i` and `B#i`. In Case 2, the security property holds because we can instantiate thread `#j` in the conclusion with thread `#k`. Thread `#k` executed Step `2` of the `B`-role and agrees with thread `#i` on all desired values because they are included in the hash. □

5.3.2 Injective agreement

The protocols in the ISO/IEC 9798 standard are also designed to provide *injective* agreement. In this case, if an agent completes the A or B role n times, he can be sure that the agent performing the other roles has participated at least n times. In the terminology of the ISO/IEC standard, this concept is called *uniqueness* and it is used to rule out replay attacks. The standard uses three techniques to achieve uniqueness: random numbers, time stamps, and sequence numbers.

Random numbers About half of the protocols achieve uniqueness by using random numbers for challenge-response. The party that wants to ensure

Repaired protocol	Role	Property	With	Data
9798-2-1	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-2-1 (UDK)	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-2-2	B	Injective agreement	A	$A, B, R_B, Text_2$
9798-2-2 (UDK)	B	Injective agreement	A	$A, B, R_B, Text_2$
9798-2-3	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-2-3 (UDK)	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-2-4	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-2-4 (UDK)	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-2-5 (DR)	A	Injective agreement	B	$A, B, P, K_{AB}, TN_A, Text_5, TN_B, Text_7$
	B	Non-injective agreement	A	$A, B, P, K_{AB}, TN_A, Text_5$
	A	Injective agreement	P	$A, B, P, K_{AB}, TVP_A, Text_3$
	B	Non-injective agreement	P	$A, B, P, K_{AB}, TN_P, Text_2$
9798-2-5 (UDK)	A	Injective agreement	B	$A, B, P, K_{AB}, TN_A, Text_5, TN_B, Text_7$
	B	Non-injective agreement	A	$A, B, P, K_{AB}, TN_A, Text_5$
	A	Injective agreement	P	$A, B, P, K_{AB}, TVP_A, Text_3$
	B	Non-injective agreement	P	$A, B, P, K_{AB}, TN_P, Text_2$
9798-2-6 (DR)	A	Injective agreement	B	$A, B, P, K_{AB}, R'_A, R_B, Text_6, Text_8$
	B	Injective agreement	A	$A, B, P, K_{AB}, R'_A, R_B, Text_6$
	A	Injective agreement	P	$A, B, P, R_A, K_{AB}, Text_4$
	B	Injective agreement	P	$A, B, P, R_B, K_{AB}, Text_3$
9798-2-6 (UDK)	A	Injective agreement	B	$A, B, P, K_{AB}, R'_A, R_B, Text_6, Text_8$
	B	Injective agreement	A	$A, B, P, K_{AB}, R'_A, R_B, Text_6$
	A	Injective agreement	P	$A, B, P, R_A, K_{AB}, Text_4$
	B	Injective agreement	P	$A, B, P, R_B, K_{AB}, Text_3$

Table 3: Properties proven of the repaired protocols in Part 2 of the standard. (UDK) indicates the protocol variants where unidirectional keys are used. (DR) indicates the variants where participants that perform the A or B role cannot also perform the P role, in accordance with Amendment 3 in Figure 6.

uniqueness generates a fresh random number and sends this to the other party. The other party must include the random number in his response, thereby uniquely binding the response to the request.

For each protocol in the standard that uses random numbers, we use our extension of SCYTHER-PROOF described in Appendix A.2 to prove injective agreement for our repaired versions.

Time stamps and sequence numbers The other protocols use time stamps or sequence numbers to achieve uniqueness. This includes all one-pass protocols because including a challenge and a response requires at least two messages exchanges.

Each time a party runs a new instance of a role, it uses a new sequence number which differs from all numbers it previously used. Recipients of messages that include a sequence number are required to verify that the received number is different from all previously received sequence numbers. Similarly, time stamps are used to achieve uniqueness by only accepting them within a fixed time

Repaired protocol	Role	Property	With	Data
9798-3-1	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-3-2	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-3-3	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-3-4	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-3-5	A	Injective agreement	B	$A, B, R_A, R_B, Text_5$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_3, Text_5$
9798-3-6 (Opt. 1)	A	Injective agreement	B	$A, B, R_A, R_B, Text_2$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_8$
	A	Injective agreement	T	$B, T, R'_A, PK_B, Text_6$
	B	Injective agreement	T	$A, T, R_B, PK_A, Text_5$
9798-3-6 (Opt. 2)	A	Injective agreement	B	$A, B, R_A, R_B, Text_2$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_8$
	A	Injective agreement	T	$A, B, T, R'_A, R_B, PK_A, PK_B, Text_5$
	B	Injective agreement	T	$A, B, T, R'_A, R_B, PK_A, PK_B, Text_5$
9798-3-7 (Opt. 1)	A	Injective agreement	B	$A, B, R_A, R_B, Text_8$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_6$
	A	Injective agreement	T	$B, T, R'_A, PK_B, Text_4$
	B	Injective agreement	T	$A, T, R_B, PK_A, Text_3$
9798-3-7 (Opt. 2)	A	Injective agreement	B	$A, B, R_A, R_B, Text_8$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_6$
	A	Injective agreement	T	$A, B, T, R'_A, R_B, PK_A, PK_B, Text_3$
	B	Injective agreement	T	$A, B, T, R'_A, R_B, PK_A, PK_B, Text_3$

Table 4: Properties proven of the repaired protocols in Part 3 of the standard.

Repaired protocol	Role	Property	With	Data
9798-4-1	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-4-1 (UDK)	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-4-2	B	Injective agreement	A	$A, B, R_B, Text_2$
9798-4-2 (UDK)	B	Injective agreement	A	$A, B, R_B, Text_2$
9798-4-3	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-4-3 (UDK)	A	Non-injective agreement	B	$A, B, TN_B, Text_3$
	B	Non-injective agreement	A	$A, B, TN_A, Text_1$
9798-4-4	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$
9798-4-4 (UDK)	A	Injective agreement	B	$A, B, R_A, R_B, Text_2, Text_4$
	B	Injective agreement	A	$A, B, R_A, R_B, Text_2$

Table 5: Properties proven of the repaired protocols in Part 4 of the standard. (UDK) indicates the protocol variants where unidirectional keys are used.

window, and verifying that the time stamp has not already been received in the time window. For both mechanisms, the required checks can only be performed by a thread if it can access data of other threads of the same agent. In other words, checking uniqueness of sequence numbers or time stamps requires shared memory among the different threads of an agent.

In our protocol model, we can not precisely model the required uniqueness checks of sequence numbers and timestamps. The reason for this is that our protocol model assumes that an agent's threads do not share any state other than the agent's long-term keys. Hence we can not prove injective agreement for protocols that use time stamps or sequence numbers within our model. However, it is straightforward to see that if an agent verifies that the messages received in each thread are different from those received in previous threads (which is possible because the messages include either a time stamp or a sequence number), then injective authentication follows from non-injective authentication.

From our analysis we conclude that all repaired protocols in the standard that provably satisfy non-injective agreement, also satisfy injective agreement.

5.4 Performance

Our extensions of the SCYTHER-PROOF tool as well as the protocol models (including the property specifications) can be downloaded at [1]. Using a Core 2 Duo 2.20GHz laptop with 2GB RAM, the full proof script generation requires less than 20 seconds, and Isabelle's proof checking requires less than three hours.

6 Related Work

Previous Analyses of the ISO/IEC 9798 Protocols. Chen and Mitchell [8] reported attacks based on parsing ambiguities on protocols from several standards. They identify two types of ambiguities in parsing strings involving concatenation: (1) recipients wrongly parse an encrypted string after decryption and (2) recipients wrongly assume that a different combination of data fields was input to the digital signature or MAC that they are verifying. They show that such errors lead to attacks, and they propose modifications to the standards. Their analysis resulted in a technical corrigendum to the ISO/IEC 9798 standard [19, 20, 22].

Some of the protocols have been used as case studies for security protocol analysis tools. In [12], the Casper/FDR tool is used to discover weaknesses in six protocols from the ISO/IEC 9798 standard. The attacks discovered are similar to our reflection and role-mixup attacks. They additionally report so-called multiplicity attacks, but these are prevented by following the specification of the time-variant parameters in Part 1 of the standard. Contrary to our findings, their analysis reports "no attack" on the 9798-2-5 and 9798-2-6 protocols as they do not consider type-flaw attacks. A role-mixup attack on the 9798-3-3 protocol was also discovered by the SATMC tool [3]. Neither of these works suggested how to eliminate the detected weaknesses.

In [11], the authors verify the three-pass mutual authentication protocols that use symmetric encryption and digital signatures, i.e., 9798-2-4 and 9798-3-4. Their findings are consistent with our results.

Related Protocols. The SASL authentication mechanism from RFC 3163 [35] claims to be based on Part 3 of the ISO/IEC 9798 standard. However, the SASL protocol is designed differently than the ISO/IEC protocols and is vulnerable to a man-in-the-middle attack similar to Lowe's well-known attack on the Needham-Schroeder public-key protocol. Currently, the SASL protocol is not recommended for use (as noted in the RFC). The SASL protocol only provides authentication in the presence of a passive eavesdropping adversary, which can also be achieved using only plaintext messages.

In the academic literature on key exchange protocols, one finds references to a Diffie-Hellman-based key exchange protocol known as "ISO 9798-3". This protocol seems to be due to [7, p. 464-465], where a protocol is given that is similar in structure to the three-pass mutual authentication ISO/IEC 9798 protocol based on digital signatures, where each random number n is replaced by ephemeral public keys of the form g^x. However, in the actual ISO/IEC 9798 standard, no key exchange protocols are defined, and no protocols use Diffie-Hellman exponentiation.

7 Conclusions

Our findings show that great care must be taken when using current implementations of the ISO/IEC 9798 standard. Under the assumption that trusted third parties do not play other roles, the protocols guarantee a weak form of authentication, namely, aliveness. However, many of the protocols do not satisfy any stronger authentication properties, such as injective or non-injective agreement, which are needed in realistic applications. For example, when using these protocols one cannot assume that when a text field is encrypted with a key and was apparently sent by Bob, that Bob actually sent it, or that he was performing the intended role. In contrast, our repaired versions satisfy strong authentication properties and hence ensure not only aliveness but also injective agreement on the participating agents, their roles, the values of time-variant parameters, and the message fields that are cryptographically protected.

Based on our analysis of the standard's weaknesses, we have proposed amendments and provided machine-checked proofs of their correctness. Our proofs guarantee the absence of these weaknesses even in the case that all protocols from the standard are run in parallel using the same key infrastructure. The working group responsible for the ISO/IEC 9798 standard has released an updated version of the standard based on our analysis and proposed fixes.

Formal methods are slowly starting to have an impact in standardization bodies, e.g., [4–6, 15, 25, 29, 30]. We expect this trend to continue as governments and other organizations increasingly push for the use of formal methods for the development and evaluation of critical standards. For example, ISO/IEC JTC 1/SC 27 started the project "Verification of cryptographic protocols (ISO/IEC 29128)" in 2007 which is developing standards for certifying cryptographic protocols, where the highest evaluation levels require the use of formal, machine checked correctness proofs [28].

We believe that the approach we have taken here to analyze and provably repair the ISO/IEC 9798 standard can play an important role in future standardization efforts. Our approach supports standardization committees with both falsification, for analysis in the early phases of standardization, and verification, providing objective and verifiable security guarantees in the end phases.

References

[1] Source code of Scyther-Proof including the models of the repaired ISO/IEC 9798 protocols, March 2013. `http://hackage.haskell.org/package/scyther-proof-0.6.0.0`.

[2] M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *Software Engineering, IEEE Transactions on*, 22(1):6 –15, 1996.

[3] A. Armando and L. Compagna. SAT-based model-checking for security protocols analysis. *Int. J. Inf. Sec.*, 7(1):3–32, 2008.

[4] D. Basin, C. Cremers, and C. Meadows. Model checking security protocols. In E. Clarke, T. Henzinger, and H. Veith, editors, *Handbook of Model Checking*, chapter 24. Springer, 2013. To appear.

[5] K. Bhargavan, C. Fournet, R. Corin, and E. Zalinescu. Cryptographically verified implementations for TLS. In *ACM Conference on Computer and Communications Security*, pages 459–468. ACM, 2008.

[6] K. Bhargavan, C. Fournet, A. D. Gordon, and N. Swamy. Verified implementations of the information card federated identity-management protocol. In *ASIACCS*, pages 123–135. ACM, 2008.

[7] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *EUROCRYPT*, volume 2045 of *LNCS*, pages 453–474. Springer, 2001.

[8] L. Chen and C. J. Mitchell. Parsing ambiguities in authentication and key establishment protocols. *Int. J. Electron. Secur. Digit. Forensic*, 3:82–94, 2010.

[9] 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://people.inf.ethz.ch/cremersc/scyther/`.

[10] C. Cremers, S. Mauw, and E. de Vink. Injective synchronisation: an extension of the authentication hierarchy. *Theoretical Computer Science*, pages 139–161, 2006.

[11] A. Datta, A. Derek, J. Mitchell, and D. Pavlovic. Abstraction and refinement in protocol derivation. In *Proc. 17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 30–45. IEEE Comp. Soc., June 2004.

[12] B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR. In *Proc. of the Workshop on Formal Methods and Security Protocols*, 1999.

[13] European Payments Council. Guidelines on algorithms usage and key management. Technical report, 2009. EPC342-08 Version 1.1.

[14] J. D. Guttman and F. J. Thayer. Protocol independence through disjoint encryption. In *CSFW*, pages 24–34, 2000.

[15] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *Proc. of the 12th ACM conference on Computer and communications security*, CCS '05, pages 2–15, New York, NY, USA, 2005. ACM.

[16] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-3:1998, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using digital signature techniques, 1998. Second edition.

[17] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-4:1999, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using a cryptographic check function, 1999. Second edition.

[18] 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.

[19] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-3:1998/Cor.1:2009, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using digital signature techniques. Technical Corrigendum 1, 2009.

[20] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-4:1999/Cor.1:2009, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using a cryptographic check function. Technical Corrigendum 1, 2009.

[21] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-1:2010, Information technology – Security techniques – Entity Authentication – Part 1: General, 2010. Third edition.

[22] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-2:2008/Cor.1:2010, Information technology – Security techniques – Entity Authentication – Part 2: Mechanisms using symmetric encipherment algorithms. Technical Corrigendum 1, 2010.

[23] International Organization for Standardization, Genève, Switzerland. ISO/IEC 9798-3:1998/Amd.1:2010, Information technology – Security techniques – Entity Authentication – Part 3: Mechanisms using digital signature techniques. Amendment 1, 2010.

[24] ITU-T. Recommendation H.235 - Security and encryption for H-series (H.323 and other H.245-based) multimedia terminals, 2003.

[25] D. Kuhlman, R. Moriarty, T. Braskich, S. Emeott, and M. Tripunitara. A correctness proof of a mesh security architecture. In *Proc. of the 2008 21st IEEE Computer Security Foundations Symposium*, pages 315–330. IEEE Computer Society, 2008.

[26] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *TACAS'96*, volume 1055 of *LNCS*, pages 147–166. Springer, 1996.

[27] G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE Computer Security Foundations Workshop (CSFW)*, pages 31–44. IEEE, 1997.

[28] S. Matsuo, K. Miyazaki, A. Otsuka, and D. A. Basin. How to evaluate the security of real-life cryptographic protocols? - the cases of ISO/IEC 29128 and CRYPTREC. In *Financial Cryptography and Data Security, FC 2010 Workshops, RLCPS, WECSR, and WLC 2010, Spain, January 25-28, 2010, Revised Selected Papers*, volume 6054 of *LNCS*, pages 182–194. Springer, 2010.

[29] C. Meadows. Analysis of the Internet Key Exchange protocol using the NRL Protocol Analyzer. In *IEEE Symposium on Security and Privacy*, pages 216–231, 1999.

[30] C. Meadows, P. F. Syverson, and I. Cervesato. Formal specification and analysis of the Group Domain Of Interpretation Protocol using NPATRL and the NRL Protocol Analyzer. *Journal of Computer Security*, 12(6):893–931, 2004.

[31] S. Meier, C. Cremers, and D. Basin. Efficient construction of machine-checked symbolic protocol security proofs. *Journal of Computer Security*, 21(1):41–87, 2013.

[32] S. Meier, C. J. F. Cremers, and D. A. Basin. Strong invariants for the efficient construction of machine-checked protocol security proofs. In *CSF*, pages 231–245. IEEE Computer Society, 2010.

[33] A. Menezes, P. V. Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc., 5th edition, 2001.

[34] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[35] R. Zuccherato and M. Nystrom. RFC 3163: ISO/IEC 9798-3 Authentication SASL Mechanism, 2001. `http://www.rfc-editor.org/info/rfc3163`.

A Extensions of Scyther-Proof

In the following, we describe two extensions of SCYTHER-PROOF needed to verify our repaired versions of the ISO/IEC 9798 protocols. The first extension supports modeling protocols that use bidirectional keys. The second extension supports proving injective authentication properties between two parties.

A.1 Bidirectional Keys

The security protocol semantics underlying the SCYTHER-PROOF tool uses a message algebra that only supports unidirectional symmetric long-term keys. Therefore every pair of agents a and b shares two different unidirectional symmetric long-term keys K_{ab} and K_{ba}, one for each direction of communication.

To verify the protocol variants of the ISO/IEC 9798 standard that use bidirectional keys, we extended the message algebra with an additional constructor $K_{\{a_1,\ldots,a_n\}}$ that models a symmetric key shared between the set of the agents $\{a_1,\ldots,a_n\}$. We can therefore model the special case of a bidirectional key shared between two agents a and b as $K_{\{a,b\}}$. Analogously, we extended the message pattern algebra, which is used to specify the messages sent and received in a security protocol, with a constructor $K_{\{v_1,\ldots,v_n\}}$ for variables $v_1,\ldots,v_n$. A thread i executing a protocol's role interprets the message pattern $K_{\{v_1,\ldots,v_n\}}$ by instantiating the variables $v_1,\ldots,v_n$ and looking up the corresponding key in a table of (pre-)shared symmetric long-term keys. If this lookup fails, then the thread stops.

We now describe in more detail how the above behavior is modeled in the transition system formalizing our security-protocol semantics. A thread's variable assignments are stored in the system state as a function $\sigma : \mathit{Var} \times \mathit{TID} \to \mathit{Msg}$. The interpretation mapping a message pattern to its corresponding message in the context of a thread i and the variable store σ is modeled by the family of partial functions $\mathit{inst}_{\sigma,i} : \mathit{Pat} \nrightarrow \mathit{Msg}$. In the semantics presented in [31,32], these functions were total. However, we now use partial functions because the lookup of a shared symmetric long-term key always fails if one of the variables is not instantiated to an agent name. Formally, we extend the definition of the $\mathit{inst}_{\sigma,i}$ function family from [31] with the following case.

$$\mathit{inst}_{\sigma,i}(K_{\{v_1,\ldots,v_n\}}) = \begin{cases} K_{\{\sigma(v_1,i),\ldots,\sigma(v_n,i)\}} & \text{if } \{\sigma(v_1,i),\ldots,\sigma(v_n,i)\} \subseteq \mathit{Agent} \\ \bot & \text{otherwise} \end{cases}$$

We also change the SEND and RECV transition rules such that the thread stops if the interpretation of a message pattern fails.

SCYTHER-PROOF's proof generation algorithm is based on a symbolic backwards search. We added support to this algorithm for bidirectional symmetric long-term keys modeled using $K_{\{a,b\}}$ as described above. We extended its unification algorithm to solve equalities over symbolic bidirectional symmetric long-term keys of the form $K_{\{\sigma(a,i),\sigma(b,i)\}}$ for agent variables a and b and some thread i. The extended unification algorithm assumes a fixed term order $\prec$ and keeps all terms normalized with respect to the conditional rewriting rule

$$K_{\{a,b\}} \rightsquigarrow K_{\{b,a\}} \qquad \text{if } a \prec b \ .$$

If an equality of the form $c(t_1, \dots, t_n) = K_{\{a,b\}}$ for a constructor c other than $K_{\{.\}}$ is encountered, then no unifier exists. Unifications of equalities of the form $K_{\{a,b\}} = K_{\{x,y\}}$ are delayed. If these are the only remaining equalities, then the unification algorithm tries to exploit one of the following equivalences and continues unification.

$$\begin{array}{ll}
K_{\{a,b\}} = K_{\{x,b\}} \Leftrightarrow a = x & K_{\{a,b\}} = K_{\{b,y\}} \Leftrightarrow a = y \\
K_{\{a,b\}} = K_{\{x,a\}} \Leftrightarrow b = x & K_{\{a,b\}} = K_{\{a,y\}} \Leftrightarrow b = y \\
K_{\{a,a\}} = K_{\{x,y\}} \Leftrightarrow a = y \wedge x = y & \\
K_{\{a,b\}} = K_{\{x,x\}} \Leftrightarrow a = x \wedge b = x &
\end{array}$$

If there are remaining equalities for which none of the above equivalences applies, then a case split is performed using the equivalence

$$(K_{\{a,b\}} = K_{\{x,y\}}) \Leftrightarrow (a = x \wedge b = y) \vee (a = y \wedge b = x).$$

Note that verifying the repaired protocol variants with unidirectional keys and the ones with bidirectional keys makes little difference in SCYTHER-PROOF's performance since such (potentially expensive) case splits are rarely required. Most encryptions include the identity of at least one of the agents sharing the key in their encrypted message. Hence, the equality of the encrypted message fully determines the equality of the agents sharing the key.

A.2 Proving Injective Authentication Properties

In this section, we explain how we extend SCYTHER-PROOF with support for proving injective two-party authentication properties. We first define both injective and non-injective two-party authentication properties. Then, we state a theorem that allows us to reduce proving an injective two-party authentication property to proving the corresponding non-injective property and an injectivity condition. Finally, we explain how we integrate this theorem into SCYTHER-PROOF's proof generation algorithm.

We build on the security protocol model from [31]. In this model, the set *TID* denotes the set of all thread identifiers. The set of reachable states of a protocol P is given by *reachable*(P). Slightly generalizing the notion of authentication properties, we define a *non-injective two-party authentication property* of a protocol P to be a closed formula of the form

$$\forall q \in \mathit{reachable}(P).\ \forall i \in \mathit{TID}.\ \mathit{claim}_q(i) \Rightarrow \exists j \in \mathit{TID}.\ \mathit{partner}_q(i, j) \ .$$

Here, $\mathit{claim}_q(i)$ is a predicate formalizing for state q which threads claim that there exists a partner thread and $\mathit{partner}_q(i, j)$ formalizes for a state q when a thread j is a partner of a thread i. It is easy to see that classical authentication properties like non-injective agreement [27] or non-injective synchronization [10] can be represented as closed formulas of this form. The following example illustrates the use of the *claim* and *partner* predicates for defining a non-injective agreement property.

Example 2. The ISO/IEC 9798-2-2 protocol with unidirectional keys, denoted by `9798_2_2_udk`, satisfies non-injective agreement for the B role on the identities `A` and `B`, the exchanged nonce `RB`, and the `Text2` field, as shown in Table 3. In the security protocol model underlying SCYTHER-PROOF [31], we formalize this property by the formula

$$\forall q \in \mathit{reachable}(\texttt{9798\_2\_2\_udk}).\ \forall i \in \mathit{TID}.\ \mathit{claim}_q(i) \Rightarrow \exists j \in \mathit{TID}.\ \mathit{partner}_q(i,j)$$

where

$$\begin{aligned} \mathit{claim}_{(tr,th,\sigma)}(i) &\overset{\text{def}}{=} \mathit{role}_{th}(i) = B \wedge (i, B_2) \in \mathit{steps}(tr) \wedge \sigma(\texttt{A}, i) \notin \mathit{Compr} \\ \mathit{partner}_{(tr,th,\sigma)}(i,j) &\overset{\text{def}}{=} \mathit{role}_{th}(j) = A \wedge (i, A_2) \in \mathit{steps}(tr) \wedge \\ &\quad\ \sigma(\texttt{A}, i) = \sigma(\texttt{A}, j) \wedge \sigma(\texttt{B}, i) = \sigma(\texttt{B}, j) \wedge \\ &\quad\ \sigma(\texttt{Text2}, i) = \sigma(\texttt{Text2}, j) \wedge \texttt{RB} \sharp i = \sigma(\texttt{RB}, j)\ . \end{aligned}$$

Intuitively, the $\mathit{claim}_q(i)$ predicate formalizes that, in the state (tr, th, σ), some thread i completed an execution of the B role talking to an uncompromised agent $\sigma(\texttt{A}, i)$. That thread i completed the execution of the B role is formalized by the condition $(i, B_2) \in \mathit{steps}(tr)$, which denotes that thread i executed the second (i.e., the last) step of the B role. The $\mathit{partner}_q(i,j)$ predicate formalizes that, in the state (tr, th, σ), the thread j completed an execution of the A role and agrees with the thread i on the values of `A`, `B`, `Text2`, and `RB`. Note that the interpretation of `RB` differs between thread i and thread j. The expression $\texttt{RB} \sharp i$ denotes the nonce freshly generated by thread i. The expression $\sigma(\texttt{RB}, j)$ denotes the value of the variable `RB` in thread j, which stores the nonce that thread j extracted from the message that it received in its first step. The remaining values `A`, `B`, and `Text2` are interpreted as variables in both threads.

We can generate a machine-checked proof certifying that the `9798_2_2_udk` protocol satisfies this non-injective agreement property using the existing proof-generation algorithm of SCYTHER-PROOF. In the following, we show how to generate a machine-checked proof certifying that this protocol also satisfies the corresponding injective agreement property.

Given a function f and a set $A \subseteq \mathit{dom}(f)$, we say that f is *injective on* A, written $\mathit{inj}_A(f)$, iff $\forall x, y \in A.\ f(x) = f(y) \Rightarrow x = y$. We define an *injective two-party authentication property* to be a closed formula of the form

$$\begin{aligned} \forall q \in \mathit{reachable}(P).\ \exists f.\ &\mathit{inj}_{\{i \in \mathit{TID} \mid \mathit{claim}_q(i)\}}(f) \wedge \\ &\forall i \in \mathit{TID}.\ \mathit{claim}_q(i) \Rightarrow \mathit{partner}_q(i, f(i))\ . \end{aligned}$$

Note that we only require f to be injective on all thread identifiers i for which $\mathit{claim}_q(i)$ holds because f is only evaluated on these thread identifiers.

In practice, many protocols rely on a challenge-response mechanism to achieve injectivity: a thread i generates a nonce (the challenge) and sends it to its intended partner, which returns the nonce in a later message (the response). This response will only be accepted by thread i, since all other threads are waiting for responses containing a different challenge. The challenge-response mechanism thus binds every responding thread to a unique challenging thread.

In our formalization of authentication properties, the set of challenging threads of a state q is characterized by the *claim* predicate, while the associated responding threads are characterized by the *partner* predicate. The *partner* predicate uniquely binds responding threads to challenging threads if

$$\forall i_1, i_2, j \in \mathit{TID}.\ \mathit{partner}_q(i_1, j) \wedge \mathit{partner}_q(i_2, j) \Rightarrow i_1 = i_2\ .$$

We call this the *injectivity property.* For a *partner* predicate satisfying this property, non-injective authentication implies injective authentication. We formalize this by the following theorem, which we have proven in Isabelle/HOL.

Theorem 1 (Non-Injective to Injective Authentication)**.** *Assume that a protocol P satisfies the non-injective two-party authentication property*

$$\forall q \in \mathit{reachable}(P).\ \forall i \in \mathit{TID}.\ \mathit{claim}_q(i) \Rightarrow \exists j \in \mathit{TID}.\ \mathit{partner}_q(i, j)$$

for some definition of the predicates claim and partner. Then the corresponding injective two-party authentication property

$$\forall q \in \mathit{reachable}(P).\ \exists f.\ \mathit{inj}_{\{i \in \mathit{TID}.\ \mathit{claim}_q(i)\}}(f) \wedge \\ \forall i \in \mathit{TID}.\ \mathit{claim}_q(i) \Rightarrow \mathit{partner}_q(i, f(i))$$

also holds, provided that the partner predicate satisfies the injectivity property.

Before explaining the integration of Theorem 1 in the proof generation algorithm of SCYTHER-PROOF, we illustrate its use in a pen-and-paper proof.

Example 3. We continue Example 2, where we proved non-injective agreement for the B role of the `9798_2_2_udk` protocol. In this example, we additionally prove injective agreement for the B-role. We formalize injective agreement using the definitions of the *claim* and *partner* predicates from Example 2. The proof is as follows.

Non-injective agreement was previously established. Moreover, the *partner* predicate satisfies the injectivity property, as $\mathit{partner}_{(tr,th,\sigma)}(i, j)$ implies the equality $\mathtt{RB} \sharp i = \sigma(\mathtt{RB}, j)$, i.e., the threads i and j agree on the nonce $\mathtt{RB} \sharp i$ freshly generated by the thread i. This equality implies the injectivity property, as $\mathtt{RB} \sharp i_1 = \sigma(\mathtt{RB}, j)$ and $\mathtt{RB} \sharp i_2 = \sigma(\mathtt{RB}, j)$ imply $\mathtt{RB} \sharp i_1 = \sigma(\mathtt{RB}, j) = \mathtt{RB} \sharp i_2$, which in turn implies $i_1 = i_2$. By Theorem 1, we thus conclude that in the `9798_2_2_udk` protocol the B role injectively agrees with the A role on the values of `A`, `B`, `Text2`, and `RB`. □

In SCYTHER-PROOF's proof generation algorithm, we prove injective two-party authentication properties by checking whether both the injectivity property and the non-injective authentication property hold. If this is the case, we then generate a proof script that instantiates Theorem 1 appropriately. In this proof script, we use Isabelle/HOL's "auto" tactic to prove the injectivity property. To generate the proof for the non-injective authentication property, we use SCYTHER-PROOF's existing proof generation algorithm described in [31].

In the implementation of SCYTHER-PROOF, we check the injectivity property by exploiting that all two-party authentication properties handled by SCYTHER-PROOF are of the form

$$\forall q \in \mathit{reachable}(P).\ \forall i \in \mathit{TID}.\ (\textstyle\bigwedge_{A\in\Gamma} A) \Rightarrow \exists j \in \mathit{TID}.\ (\textstyle\bigwedge_{B\in\Delta} B),$$

where Γ, Δ are sets of *atoms* of the following form.

$$\begin{array}{lllll} \mathit{tid} = \mathit{tid}' & m = m' & \mathit{role}_{th}(\mathit{tid}) = R & \sigma(a, \mathit{tid}) \in \mathit{Compr} & \mathit{false} \\ e \prec_{tr} e' & (\mathit{tid}, s) \in \mathit{steps}(tr) & m \in \mathit{knows}(tr) & \sigma(a, \mathit{tid}) \notin \mathit{Compr} & \end{array}$$

See [31, Section IV-B] for the formal definitions of these constructs. To check the injectivity property, we can therefore check whether

$$(\textstyle\bigwedge_{B\in\Delta} B)[i \leftarrow i_1] \wedge (\textstyle\bigwedge_{B\in\Delta} B)[i \leftarrow i_2] \quad \text{implies} \quad i_1 = i_2\ .$$