Research Report

Symbolic and Cryptographic Analysis of the Secure WS-ReliableMessaging Scenario

Michael Backes[1], Sebastian Mödersheim[2], Birgit Pfitzmann[1], and Luca Viganò[2]

[1]IBM Research GmbH
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

[2]Information Security Group
ETH Zentrum
8092 Zürich
Switzerland

IBM **Research**
Almaden · **Austin** · **Beijing** · **Delhi** · **Haifa** · **T.J. Watson** · **Tokyo** · **Zurich**

Symbolic and Cryptographic Analysis of the Secure WS-ReliableMessaging Scenario

Michael Backes[1], Sebastian Mödersheim[2], Birgit Pfitzmann[1], and Luca Viganò[2]

[1] IBM Zurich Research Lab, Switzerland
[2] Information Security Group, ETH Zurich, Switzerland

Abstract. Web services are an important series of industry standards for adding semantics to web-based and XML-based communication, in particular among enterprises. Like the entire series, the security standards and proposals are highly modular. Combinations of several standards are put together for testing as interoperability scenarios, and these scenarios are likely to evolve into industry best practices. In the terminology of security research, the interoperability scenarios correspond to security protocols. Hence, it is desirable to analyze them for security. In this paper, we analyze the security of the new Secure WS-ReliableMessaging Scenario, the first scenario to combine security elements with elements of another quality-of-service standard. We do this both symbolically and cryptographically. The results of both analyses are positive. The discussion of actual cryptographic primitives of web-services security is a novelty of independent interest in this paper.

1 Introduction

Web services are a series of standards that add higher-layer semantics and quality of service to web-based communication. They use XML as the basic format for all exchanged content and SOAP as the basis for message exchanges [19]. In principle, web services are independent of the underlying transport protocol; in practice, as the name suggests, typical web protocols are commonly used. An important principle of web services is modularity (see [28]). This principle was in particular applied to the design of quality-of-service features like security and message ordering. Thus, these features are addressed by a set of standards and pre-standard proposals that can, at least syntactically, be combined in a highly flexible way. It is well-known, however, that combinations of security elements have to be treated with care in the sense that many combinations may not yield the properties that one might expect. The equivalent to the classic notion of security protocols in the web-services space is interoperability profiles or scenarios. While primarily defined for interoperability testing, they are not unlikely to evolve into industry best practices for common cases. At the same time, they are at the level of concreteness where an analysis for well-known protocol security properties is possible.

In this paper, we present the first such analysis for an interoperability profile that combines features from the standards and proposals for security and another quality-of-service area, reliable messaging. It is the Secure WS-ReliableMessaging Scenario [25], which recently arose from the WS-ReliableMessaging and WS-SecureConversation Composability Interop Workshop held in April 2005.[1] It is based on the WS-Security standard [37] and the

---

[1] The title of [25] contains "scenarios" in the plural, but for our purposes the document defines one protocol and we thus use the singular.

recent standard proposals WS-ReliableMessaging [29] and WS-SecureConversation [33] with a few additional references to WS-Trust [34] and WS-Addressing [18].

We present two types of analyses:

1. an automated analysis based on a number of symbolic protocol analysis techniques under the assumption of perfect cryptography, and
2. an analysis closer to real cryptography based on explicit cryptographic assumptions on the underlying cryptographic algorithms used.

Both analyses refer to the properties that are already informally stated in WS-ReliableMessaging [29], where they are pointed out as desirable security properties in the context of reliable sending of messages. WS-ReliableMessaging does not address how these properties can be achieved but refers to a suitable combination with the techniques offered by the security-specific web services standards. The Secure WS-ReliableMessaging Scenario provides such a combination, and our analysis exemplifies that the properties can indeed be achieved by the techniques offered by existing web services standards.

Our first, symbolic analysis has been carried out by employing the AVISPA Tool [3], which is a push-button tool for the analysis of security-sensitive protocols and applications, under the assumption of perfect cryptography. The AVISPA Tool relies on a modular and expressive formal language for specifying protocols and their security properties, and integrates different back-ends that implement a variety of state-of-the-art automatic analysis approaches. For our analysis, we have employed OFMC [8] and CL-AtSe [44], which are the two more mature back-ends of the tool and which both perform protocol falsification and bounded verification by employing a number of symbolic techniques (where some basic techniques are common to the two back-ends while other ones are back-end specific).

The Secure WS-ReliableMessaging Scenario has a structure that is far more complex than standard security protocols. Hence, an important part of modeling the protocol in a way feasible for automated analysis has been the search for a way to restrict the number of permissible interleavings of sending and receiving events without excluding attacks, i.e., every attack on the original protocol should be possible also on the simplified version. Below, we will first explain how we have built such a specification, and then illustrate the goals that we have checked in our analysis. Roughly speaking, we have shown that a client and a service mutually authenticate each other on certain messages that they exchange when executing the protocol, and that these messages remain secret. These problems give rise to an infinite search space, so that automated tools need to make restrictions on some aspects of the problem in order to analyze it. We have considered different settings by imposing bounds on the number of possible parallel protocol sessions, on the number of message sequences that can be considered in each session, and on the number of payloads per message sequence. Neither OFMC nor CL-AtSe have reported any attacks for the settings we considered, and they have thus verified the Secure WS-ReliableMessaging Scenario with respect to the modeled security properties for these settings.

Our second analysis is manual (and thus more time-consuming, less flexible to protocol additions, and more prone to human error), but more realistic with respect to the cryptographic primitives. For instance, we show that we can treat the occurring key derivation via hash functions in the standard model of cryptography as pseudo-random functions if applied to certain pairs of arguments. For the other primitives, symmetric and asymmetric encryption as well as symmetric authentication and signatures, we can use standard definitions. We also discuss how close existing theorems on justifying symbolic analyses such as

2

Long-term keys:

pke_X, ske_X	Public and secret encryption key of $X \in \{C, S\}$.
pks_X, sks_X	Public and secret signature key of $X \in \{C, S\}$.
pks_{CA}	Public signature key of a certification authority CA.
$Cert_X$	Public key certificate of $X \in \{C, S\}$. We have $Cert_X = X, pke_X, pks_X,$ $\mathsf{Sig}_{CA}(X, pke_X, pks_X)$, where $\mathsf{Sig}_{CA}(\cdot)$ denotes a signature computed by the certification authority CA, valid with respect to pks_{CA}.

Cryptographic primitives:

$\mathsf{Enc}_X(\cdot)$	A public-key encryption scheme, denoting encryptions computed with public key pke_X for $X \in \{C, S\}$.
$\mathsf{Sig}_X(\cdot)$	A digital signature scheme, denoting signatures computed with secret key sks_X for $X \in \{C, S\}$.
$\mathsf{SymEnc}_k(\cdot)$	A symmetric encryption scheme, denoting encryptions computed with secret key k.
$\mathsf{Mac}_k(\cdot)$	A message authentication code, denoting MACs computed with secret key k.
$\mathsf{Hash}(\cdot)$	A hash function, e.g., SHA-256.

Fig. 1. Keys and cryptographic algorithms used in the Secure WS-ReliableMessaging Scenario.

our first one come to replacing a from-scratch cryptographic analysis such as our second one. Note, however, that the Secure WS-ReliableMessaging Scenario, like all other current communication security standards, does not prescribe that provably secure primitives in the cryptographic sense are used, in particular for the symmetric primitives. Thus, we cannot claim that we proved exactly the standard implementations under what became known as standard cryptographic assumptions such as the hardness of factoring. Our cryptographic analysis is modular, and some results can immediately be reused for other profiles, e.g., the analysis of the initial key exchange based mainly on WS-Trust and that of the key derivation using elements of WS-SecureConversation.

Both our analyses have positive results, i.e., they demonstrate that at the abstraction level of each analysis, the protocol is error-free. Note that our two analyses are complementary (in particular, neither of them is derived from the other), but we consider it interesting future work to investigate how to link the two kinds of analysis for web services in the style of previous proofs of soundness of Dolev-Yao models, e.g., see [1, 5–7, 21, 42].

Outline of the Paper. We start by describing the Secure WS-ReliableMessaging Scenario and the corresponding security properties in Section 2. Sections 3 and 4 contain the symbolic and the cryptographic analysis of the scenario, respectively. After reviewing further related work in Section 5, we give concluding remarks and an outlook on possible future extensions of this work in Section 6.

2 The Secure WS-ReliableMessaging Scenario

The Secure WS-ReliableMessaging Scenario is a two-party protocol initiated by a client C and run together with a service S. It consists of three phases starting with a key-exchange

Quantities occurring in the protocols:	
$ID_1, \ldots, ID_9$	Message IDs of the individual protocol messages.
ID_{sk}	ID of the symmetric master key sk that is established in the initial key exchange phase.
ID_{Seq}	Sequence ID denoting the sequence of exchanged messages.
N, N^*	Nonces used to compute the master key sk.
N_1, N_2	Nonces used to compute the authentication and encryption session keys sk_1 and sk_2.
m	Payload that should be reliably sent from C to S.
n	Natural number denoting an acknowledged message.
k, k'	Symmetric keys used within a hybrid encryption in the initial key exchange phase.
sk	Symmetric master key shared between C and S after the initial key exchange phase. Derived from N and N^* as $sk = \mathsf{Hash}(N, N^*)$.
sk_1, sk_2	Symmetric session keys for authentication and encryption shared between C and S after the start of the message sending. Derived from sk, N_1, and N_2 as $sk_i = \mathsf{Hash}(N_i, sk)$.

Fig. 2. Quantities used in the Secure WS-ReliableMessaging Scenario.

phase, followed by the message-sending phase which uses this key, and finished by a termination phase which cancels the validity of the exchanged keys.

In the following, we will use a straight font to denote cryptographic algorithms (Enc, Sig, etc.), a straight font with capital letters to denote protocol-specific constants (RST, RSTR, etc.), and an *italic* font to denote keys, identities, etc.

The key-exchange phase is based on public-key cryptography and hence requires a mechanism to authenticate the respective public keys. The profile assumes a certification authority CA, which has a secret key sks_{CA}. Its public counterpart, pks_{CA}, is known to both C and S. The certification authority certifies the public keys of party $X \in \{C, S\}$ by signing the triple (X, pke_X, pks_X) with its key sks_{CA}, where pke_X and pks_X denote X's public encryption key and X's signature verification key, respectively. Note that pks_{CA} must have been conveyed in an authenticated manner to both C and S, and that pks_{CA} must not give certificates with the name X of an honest party to any other party.

Figures 1 and 2 summarize the notation for the keys held by both parties, the cryptographic primitives we will be using, and the quantities involved in the protocol. For interoperability, the scenario uses specific cryptographic algorithms to implement the respective primitives — RSA-1.5 for public-key encryption, RSA-SHA1 for digital signatures, AES128-CBC for symmetric encryption, and HMAC-SHA1 for message authentication codes. In the cryptographic analysis that we carry out in Section 4, we do not fix specific algorithms but require that the used algorithms satisfy the respective security definitions under active attacks, e.g., indistinguishability under adaptive chosen-ciphertext attacks in the case of public-key encryption. Efficient schemes that satisfy these definitions exist under reasonable assumptions.

Composite Fields for Initial Key Exchange (Step 1-2):

$body_1$	$\mathsf{SymEnc}_k(\mathsf{RST}, S, N)$
$SigConf$	$\mathsf{Sig}_C(ID_1, S, \mathsf{RST}, C, body_1, Cert_C)$
$header_1$	$\mathsf{Enc}_S(k), \mathsf{SymEnc}_k(SigConf)$
$body_2$	$\mathsf{SymEnc}_{k'}(\mathsf{RSTR}, ID_{sk}, S, N^*)$
$header_2$	$\mathsf{Enc}_C(k'), \mathsf{SymEnc}_{k'}(SigConf),$ $\mathsf{SymEnc}_{k'}(\mathsf{Sig}_S(ID_2, C, \mathsf{RSTR}, ID_1, SigConf, body_2))$

Protocol Flows (Step 1-2 from WS-SecureConversation):

1. $\mathsf{RequestSecurityToken}$: $C \xrightarrow{\quad ID_1, S, \mathsf{RST}, C, Cert_C, header_1, body_1 \quad} S$

2. $\mathsf{RequestSecurityTokenResponse}$: $C \xleftarrow{\quad ID_2, C, \mathsf{RSTR}, ID_1, Cert_C, header_2, body_2 \quad} S$

Fig. 3. The Key-exchange Phase, implemented via WS-SecureConversation.
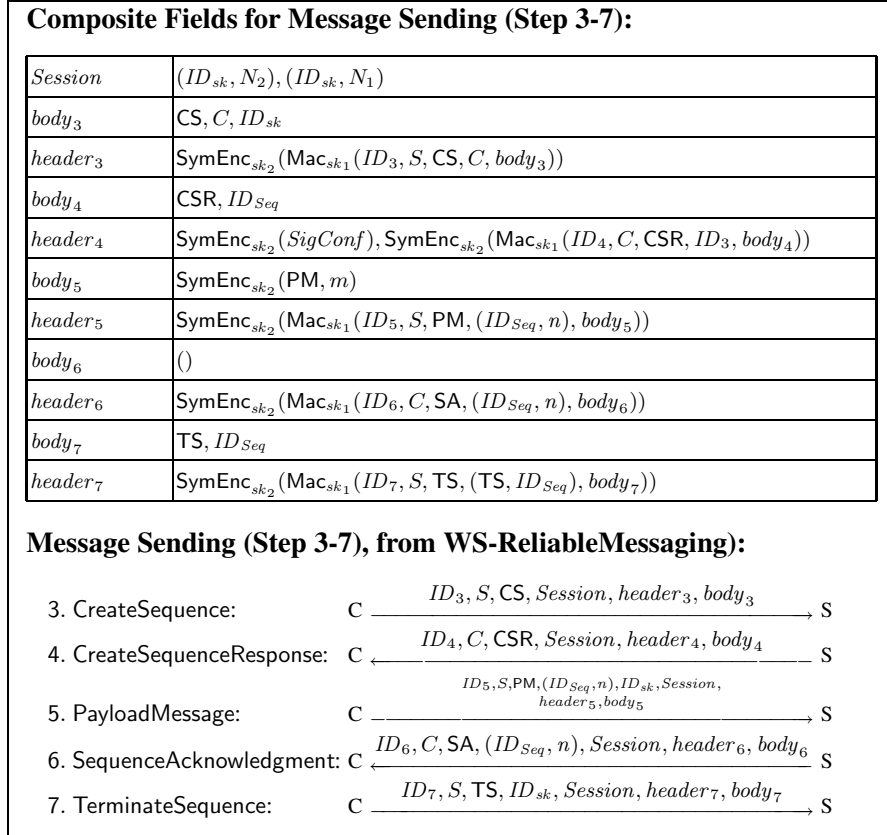
2.1 Description of the Protocol

Before the protocol begins, each party $X \in \{C, S\}$ has some starting information. Besides its own encryption and signature keys, the client starts with the signature verification key pks_{CA} of the certification authority CA, a certificate $Cert_C$ of its own public keys, and a certificate $Cert_S$ of the public keys of the service. The service starts only with the signature verification key pks_{CA} and with its own encryption and signature keys.

The protocol consists of nine steps. The first two steps constitute the key-exchange phase of the protocol between the client and the service and essentially rely on the functionalities offered by WS-SecureConversation; they are depicted in Figure 3. Similarly, the last two steps cancel the validity of this key again as depicted in Figure 5. Steps three to seven are depicted in Figure 4 and constitute the message-sending phase, which consists of the creation of a message sequence, the secure sending of a message m, and the closing of the sequence; each of these steps essentially relies on the functionalities offered by WS-ReliableMessaging. An illustrative prose description of the individual protocol steps based on Figure 3-5 is given in Appendix A.

The protocol is not simply a ping-pong protocol: after the key-exchange phase has been completed, the client is allowed to start multiple sessions of the message-sending phase in parallel and there are non-deterministic choices on the order of messages.

The necessary tests on the received messages follow the usual convention as described in [36], e.g., an honest receiver of a message checks that the decrypted plaintexts are of the correct format, that respective parts of the plaintext match corresponding parts sent unencrypted in the same message, and that the sender and receiver fields contain the expected values. We do not always mention this explicitly in the following.

Possible Protocol Extensions. We moreover sketch a possible extension of the interoperability scenario to reflect additional capabilities of the client and the service offered by the WS-ReliableMessaging standard. The standard additionally allows a client to request an unreceived acknowledgment of a previously sent message, and it allows a service to ask the client to re-send a message if it has not been received yet. This yields two additional steps which are depicted in Figure 6. Their prose description is given in Appendix A.

Composite Fields for Message Sending (Step 3-7):

$Session$	$(ID_{sk}, N_2), (ID_{sk}, N_1)$
$body_3$	CS, C, ID_{sk}
$header_3$	$\mathsf{SymEnc}_{sk_2}(\mathsf{Mac}_{sk_1}(ID_3, S, \mathsf{CS}, C, body_3))$
$body_4$	CSR, ID_{Seq}
$header_4$	$\mathsf{SymEnc}_{sk_2}(SigConf), \mathsf{SymEnc}_{sk_2}(\mathsf{Mac}_{sk_1}(ID_4, C, \mathsf{CSR}, ID_3, body_4))$
$body_5$	$\mathsf{SymEnc}_{sk_2}(\mathsf{PM}, m)$
$header_5$	$\mathsf{SymEnc}_{sk_2}(\mathsf{Mac}_{sk_1}(ID_5, S, \mathsf{PM}, (ID_{Seq}, n), body_5))$
$body_6$	$()$
$header_6$	$\mathsf{SymEnc}_{sk_2}(\mathsf{Mac}_{sk_1}(ID_6, C, \mathsf{SA}, (ID_{Seq}, n), body_6))$
$body_7$	TS, ID_{Seq}
$header_7$	$\mathsf{SymEnc}_{sk_2}(\mathsf{Mac}_{sk_1}(ID_7, S, \mathsf{TS}, (\mathsf{TS}, ID_{Seq}), body_7))$

Message Sending (Step 3-7), from WS-ReliableMessaging):

3. CreateSequence: $C \xrightarrow{\quad ID_3, S, \mathsf{CS}, Session, header_3, body_3 \quad} S$

4. CreateSequenceResponse: $C \xleftarrow{\quad ID_4, C, \mathsf{CSR}, Session, header_4, body_4 \quad} S$

5. PayloadMessage: $C \xrightarrow{\quad ID_5, S, \mathsf{PM}, (ID_{Seq}, n), ID_{sk}, Session, \; header_5, body_5 \quad} S$

6. SequenceAcknowledgment: $C \xleftarrow{\quad ID_6, C, \mathsf{SA}, (ID_{Seq}, n), Session, header_6, body_6 \quad} S$

7. TerminateSequence: $C \xrightarrow{\quad ID_7, S, \mathsf{TS}, ID_{sk}, Session, header_7, body_7 \quad} S$

Fig. 4. The Message-sending Phase, implemented via WS-ReliableMessaging.

2.2 Security Properties

We consider a range of reasonable security requirements for the parties involved; some of the requirements are explicitly mandated by the standards, others are optional and hold only under stronger assumptions on the underlying cryptographic primitives.

The following security properties are explicitly pointed out in WS-ReliableMessaging:

– *No Message Alteration*: Payloads contained in the 5. PayloadMessage in a session between an honest client and an honest service cannot be altered by an adversary.
– *No Message Disclosure*: Payloads contained in the 5. PayloadMessage in a session between an honest client and an honest service remain secret from the adversary.
– *Key Integrity and Confidentiality*: If an honest client and an honest service established a shared key sk after the first two steps of the protocol, both parties obtained the same key. Moreover, this key is secret from the adversary.
– *Authentication*: If an honest service accepts a payload m presumably from an honest client, then this honest client indeed sent this payload in the same session.

Accountability is also mentioned in WS-ReliableMessaging as one of the properties desirable in certain scenarios. As this scenario uses symmetric cryptography for the message authentication, accountability in the sense of non-repudiation is clearly not a goal of this

Composite Fields for Session Closure (Step 8-9):

$Session$	$(ID_{sk}, N_2), (ID_{sk}, N_1)$
$body_8$	CST, ID_{sk}
$header_8$	$\mathsf{SymEnc}_{sk_2}(\mathsf{Mac}_{sk_1}(ID_8, S, \mathsf{CST}, C, body_8))$
$body_9$	CSTR
$header_9$	$\mathsf{SymEnc}_{sk_2}(SigConf),$ $\mathsf{SymEnc}_{sk_2}(\mathsf{Mac}_{sk_1}(ID_9, C, \mathsf{CSTR}, (ID_{sk}, N_1), body_9))$

Protocol Flows (Step 8-9, from WS-SecureConversation):

8. CancelSecurityToken: $\quad$ C $\xrightarrow{\quad ID_8, S, \mathsf{CST}, C, ID_{sk}, Session, header_8, body_8 \quad}$ S

9. CancelSecurityTokenResp: C $\xleftarrow{\quad ID_9, C, \mathsf{CSTR}, Session, header_9, body_9 \quad}$ S

Fig. 5. The Termination Phase, implemented via WS-SecureConversation.

scenario. The potential real-life accountability of this scenario is formally captured on the protocol level by the message integrity property and otherwise given by non-protocol factors. We refer to Appendix C for additional useful properties that are not explicitly required by the standard as well as for a refinement of the aforementioned properties tailored to the Secure WS-ReliableMessaging Scenario.

3 Symbolic Security Analysis

The AVISPA Tool. We have carried out a symbolic analysis of the Secure WS-ReliableMessaging Scenario by employing the *AVISPA Tool* [3], which is a push-button tool for the automated validation, under the assumption of perfect cryptography and Dolev-Yao adversary [26], of industrial-scale Internet security-sensitive protocols and applications. A user interacts with the AVISPA Tool by specifying a *security problem* (a protocol paired with a security property that it is expected to achieve) in the *High-Level Protocol Specification Language HLPSL* [22], which is an expressive, modular, role-based, formal language that allows for the specification of control flow patterns, data structures, alternative adversary models, complex security properties, as well as different cryptographic operators and their algebraic properties. The AVISPA Tool automatically translates a user-defined security problem into an equivalent description of an infinite-state transition system that is then input to the back-ends of the AVISPA Tool. The back-ends search the transition system for states that represent attacks on the intended properties of the protocol.

The current version [3] of the tool integrates four back-ends that implement a variety of state-of-the-art automatic analysis techniques, ranging from *protocol falsification* (by finding an attack on the input protocol) to *abstraction-based verification* methods for infinite numbers of sessions. The back-ends are: the *On-the-fly Model-Checker OFMC*, the *Constraint-Logic-based Attack Searcher CL-AtSe*, the *SAT-based Model-Checker SATMC*, and the *TA4SP verifier*, which analyzes protocols by implementing tree automata based on automatic approximations. All the back-ends of the tool analyze protocols by considering the standard Dolev-Yao model of an active adversary that controls the network but cannot

7

Composite Fields for Protocol Extension:

$ID_{5.1}, ID_{5.1*}$	Message IDs of the additional protocol messages.
$body_{5.1}$	()
$header_{5.1}$	$\mathsf{SymEnc}_{sk_2}(\mathsf{Mac}_{sk_1}(ID_{5.1}, C, \mathsf{NAck}, (ID_{Seq}, n), body_{5.1}))$
$body_{5.1*}$	()
$header_{5.1*}$	$\mathsf{SymEnc}_{sk_2}(\mathsf{Mac}_{sk_1}(ID_{5.1*}, C, \mathsf{AR}, (ID_{Seq}, n), body_{5.1*}))$

Resend and Ack Inquiries (Between Step 5 and 6, from WS-ReliableMessaging):

5.1. NotAcknowledged: $C \xleftarrow{ID_{5.1}, C, \mathsf{NAck}, (ID_{Seq}, n), Session, header_{5.1}, body_{5.1}} S$

5.1*. AckRequested: $C \xrightarrow{ID_{5.1*}, C, \mathsf{AR}, (ID_{Seq}, n), Session, header_{5.1*}, body_{5.1*}} S$

Fig. 6. Extension of the Secure WS-ReliableMessaging Scenario with Resend Inquiries.

break cryptography; in particular, the adversary can intercept messages and analyze them if it possesses the respective keys for decryption, and it can generate messages from his knowledge and send them under any party's name. Upon termination, the AVISPA Tool outputs that the protocol was verified with respect to the specified security problem, that an attack was found, or that the available resources were exhausted.

For our analysis of the Secure WS-ReliableMessaging Scenario, we have employed OFMC [8] and CL-AtSe [44], which are the two more mature back-ends of the tool, with better scope and performance. OFMC and CL-AtSe both perform protocol falsification and bounded verification by employing a number of symbolic techniques. Some of these techniques are back-end specific, while other ones are common to the two back-ends, such as the *lazy intruder technique* to symbolically represent all the possible messages that the Dolev-Yao adversary can generate. These techniques enable both OFMC and CL-AtSe to handle protocols with complex message terms and in particular to model the Secure WS-ReliableMessaging Scenario in its full complexity, without having to simplify the messages that are exchanged.[2]

The Model. The back-ends of the AVISPA Tool have successfully validated (or found a number of new attacks on) security protocols such as those in the Clark/Jacob library [23], as well as Kerberos, IKE, SET, and other protocols proposed by standardization organizations such as the IETF, ITU, W3C, Oasis, IEEE, 3GPP, and OMA. Similar analyses have been carried out by other (semi-)automated tools such as [9, 16, 17, 27, 43].

The Secure WS-ReliableMessaging Scenario has a structure that is far more complex than that of standard security protocols. Nonetheless, thanks to its expressiveness, HLPSL allows us to completely model the protocol, i.e., to provide a formal specification of the complex interactions between the two honest parties, which we can model as two separate client and service *programs* that communicate over an insecure network controlled by a Dolev-Yao adversary. However, such a model is too complex for automated analysis as

---

[2] The complexity of the Scenario prevents the usage of SATMC or TA4SP. We hope to soon be able to report on the analysis with these back-ends as well; in particular, if analysis with TA4SP succeeded, then that would prove that the protocol is safe for secrecy goals for any number of sessions.

even for a limited number of sessions, the set of permissible interleavings of sending and receiving events is enormous. For instance, the messages sent by the client may arrive in any order at the service. Additionally, both the client and the service can send "administrative" messages, i.e., acknowledge messages, request the retransmission of messages, or request the acknowledgment of messages. An important part of modeling the protocol in a way feasible for automated analysis has thus been the search for a way to restrict the number of interleavings without excluding attacks, i.e., such that any attack on the original protocol is possible also on the simplified version.

We have performed a step-by-step simplification of the client and service programs, whereby we have showed that these simplifications do not exclude any attacks.[3] As we lack space to give the HLPSL specification here due to its complexity and the amount of explanation that would be necessary, we only sketch the main ideas behind our HLPSL specification. In particular, we briefly illustrate the simplifications we have carried out for the client program; the ones for the service program are similar, and more details can be found in Appendix B.

In order to simplify the client, note, firstly, that it is not a restriction if the client sends in one transition all the messages that it wishes to transmit via the 5. PayloadMessage step as soon as it has received the 4. CreateSequenceResponse message. Secondly, the client can neglect any requests of step 5.1. NotAcknowledged from the service to retransmit messages, since the Dolev-Yao adversary has seen all messages and can thus replay them to the service if this is necessary for an attack. Hence, we can consider a simplified client program that, having sent all its payload messages, simply waits for acknowledgment messages (6. SequenceAcknowledgment) or, after timing out, requests acknowledgment from the service (5.1*. AckRequested). Thirdly, since the Dolev-Yao adversary can intercept all responses from the service, it might deliberately make the client produce acknowledge request messages. Hence we can assume that the adversary can obtain acknowledge request messages of step 5.1*. AckRequested for every payload message. No attacks are therefore excluded if the client program sends with every 5. PayloadMessage also an 5.1*. AckRequested message.

These simplifications yield a client program that behaves as follows in every message sending phase: it sends all payload messages together with the corresponding requests for acknowledgment in one step, then waits until all messages are acknowledged, and finally sends a 7. TerminateSequence message.

Goals. Let us define the *security-relevant messages* of the Secure WS-ReliableMessaging Scenario to be the key-material (sk, sk_1, and sk_2) and all payloads transmitted with a 5. PayloadMessage. For our symbolic analysis, we have specified a number of *secrecy* and *authentication goals* (giving rise to different HLPSL security problems for the Scenario):

– secrecy of all security-relevant messages, and
– mutual authentication between client and service on all security-relevant messages.

We model these goals by labeling several transitions in the HLPSL specification with special events that express the meaning of the transition with respect to the goals of the protocol.

---

[3] The simplified (restricted) version of the protocol that we obtain in this way is only useful for the formal analysis, not for the practical deployment of the protocol: for instance, since a Dolev-Yao adversary can replay old messages arbitrarily if this is necessary to mount an attack, we can restrict the model to client programs that never retransmit old messages.

First, whenever a client c that believes to talk with service s creates a security-relevant message m, then it generates a *secret* event secret(m,{c,s}) expressing that m must remain secret between the parties in the specified set, in this case c and s. This allows us to define a *violation of secrecy* by a state of the transition system in which the adversary knows a message m for which a secrecy event has occurred with a set of parties to which the adversary does not belong. Second, we define violations of authentication by labeling the transitions with *witness* and *request* events. Whenever a party a that believes to talk with another party b first "handles" some security-relevant message m (i.e., either creates it or receives it for the first time), then it generates an event witness(a,b,id,m) where id is an identifier that uniquely determines the purpose of the message in the protocol. This witness event expresses that a uses message m for communication with b and for purpose id. The service s generates an event request(s,c,id,m) when it receives a payload m (supposedly) from the client c with index id. Similarly, if the client c receives the acknowledgement for the id-th payload (supposedly) from the service s, and if c has previously sent m as the id-th payload, then c generates the event request(c,s,id,m). Similar request events are generated for the authentication on the key-material. (Intuitively, request events express that a party begins to rely on the agreement with another party on the specified value.)

A *violation of authentication* is then defined as any of the two following situations. First, *weak authentication* is violated whenever there is a request(b,a,id,m) but no matching witness event witness(a,b,id,m), i.e., a party b believes a message m to come from a, but a has never sent m, at least not for this purpose. Second, *strong authentication* is violated whenever weak authentication is, or whenever a request event occurs more frequently than the corresponding witness event (i.e., by a kind of replay, the adversary made party b accept a message more often than it was actually said by a). Note that these goals are equivalent to Lowe's [41] notions of non-injective and injective agreement, respectively.

The security problems that we obtain by modeling these goals cover the main security properties stated for the Secure WS-ReliableMessaging Scenario in Section 2.2 as follows:

- secrecy of all security-relevant messages covers *no message disclosure* and *key confidentiality*,
- mutual authentication between client and service on all security-relevant messages covers *no message alteration*, *key integrity*, and *authentication*.

Bounds of the Analysis. The security problems associated with the Secure WS-ReliableMessaging Scenario give rise to an infinite search space, so that, in order to analyze this space, automated tools need to make some restrictions, i.e., to impose some bounds to consider relevant protocol execution and analysis settings. In the following, we will describe the restrictions that we imposed in our analysis with OFMC and CL-AtSe.

In general, there is no bound on the number of parties and sessions of the protocol that can be executed in parallel. While one can bound the number of parties, by the argumentations of [24] or by the symbolic session technique of OFMC [8], the problem of an unbounded number of sessions cannot be solved in general since it gives rise to undecidability. Moreover, there are two similar problems of unboundedness in the protocol: there is no bound on the number of payload messages to be exchanged or on the number of new message sequences that can be started, i.e., the protocol contains unbounded loops. All these problems give rise to an unbounded number of steps of honest parties, while both OFMC

and CL-AtSe currently require analysis settings with bounded numbers of steps of honest parties.

In general, there is also no bound on the complexity of messages that the adversary can generate. However, as we remarked above, both OFMC and CL-AtSe implement the lazy intruder technique, which uses a symbolic representation to avoid explicitly enumerating the possible messages that the Dolev-Yao adversary can generate, and which allows for an analysis without restricting this parameter of the problem.

We have therefore analyzed the protocol with OFMC and CL-AtSe under the following execution/analysis settings: there are at most three parallel protocol sessions, the client can start at most two message-sending sequences per protocol session, and there are at most three payload messages per message sequence. Neither OFMC nor CL-AtSe have reported any attacks on the protocol for these analysis settings. In particular, for three parallel sessions, both OFMC and CL-AtSe verified the protocol within three hours (while the verification of smaller settings required between few seconds to a minute).

4 Cryptographic Security Analysis

In this section, we complement the symbolic analysis of the security properties of the WS-ReliableMessaging Scenario from Section 3 by a cryptographic analysis. Thus we now analyze the security of the scenario in a cryptographic setting where the cryptographic primitives and the perfect cryptography assumption are replaced with actual cryptographic algorithms and the corresponding security notions that reason about probabilistic polynomial-time attackers. It is known that, even if the symbolic analysis is careful in distinguishing primitives like symmetric encryption and authentication, as both the analyzed scenario and the analysis in Section 3 do, and even if one assumes that an implementation is made with primitives secure according to the strictest usual cryptographic definitions, the results of such a symbolic analysis may not carry over to the real implementation. The most prominent example is that it cannot be avoided in general that the length of encrypted payload data, such as the values m in the PayloadMessage, leaks. Other problems that may occur in general scenarios are due to the probabilism of secure public-key encryption, key-stealing attacks, and manipulations of symmetric encryptions unless authenticated encryption [11, 10] is used in the implementation [5, 4]. Consequently, in a Dolev-Yao-style cryptographic library designed to be implemented based on arbitrary cryptographically secure primitives and to be usable in a secure way within arbitrary protocols with arbitrary security properties, both the abstraction and the realization must have certain idiosyncrasies. Hence, while it might be interesting to augment a tool like the AVISPA Tool by the idiosyncrasies of the Dolev-Yao style model of [5, 6, 4], and while implementing the primitives of WS-Security with the extended realizations from those papers (e.g., some additional tagging and randomization) might realize the goal of web service security to offer completely composable primitives also in a semantic sense, neither has been done yet. Other work on bridging the gap between symbolic and cryptographic security concentrated more on keeping very close to standard symbolic and real versions at the cost of generality. However, at present none of them covers the protocol class of Secure WS-ReliableMessaging, nor the security properties required. The seminal work [1] treats passive attacks only. Active attacks have been considered in this context in [42, 40, 21]. First, however, each of these papers treats only one cryptographic primitive, asymmetric encryption in [42, 21] and symmetric encryption in [40]. Secondly, [42] only treats integrity properties, while [40] only treats the secrecy

of fixed, protocol-internal messages and [21] only treats the secrecy of nonces, i.e., random values chosen within the protocol and not usable for operations (such as encrypting) in that protocol. It may be interesting future work to extend such results on restricted usage of cryptographic libraries to the typical usage in WS-Security protocols. Our following considerations can be seen as a step in this direction.

Given these shortcomings of the current methods for deducing the security in the cryptographic setting from a symbolic proof, we do not try to do that, but base our proof directly on existing cryptographic work that explored the security of encryption, signatures, and MACs when combined in specific ways. In the following, we assume that the public-key encryption system Enc be secure against adaptive-chosen ciphertext attacks (short IND-CCA2-secure), that the symmetric encryption scheme be secure under adaptive chosen-plaintext attacks (short IND-CPA-secure), and that the signature scheme Sig and the message authentication scheme Mac be secure against adaptive chosen-message attacks (short IND-CMA-secure). These are the commonly accepted security definitions of these primitives under active attacks so that we omit their rigorous definition. Primitives secure in this sense exist under reasonable assumptions.

Furthermore, we have to require that the hash function Hash used to compute the secret key sk based on two secret nonces does not degenerate the randomness induced by the nonces. This would be clear if we worked in the random oracle model; however, the specific setting of the scenario allows us to work in the standard model with a sufficient condition being that Hash, when applied to pairs, is a pseudo-random function in its first argument.

We obtain the following theorem (whose proof is postponed to Appendix D for the sake of space), in which we assume that the Secure WS-ReliableMessaging Scenario is run as a stand-alone protocol. This is not necessarily realistic for a web-services implementation; then our approach may have to take policies into account as in [14].

Theorem 1. *(Cryptographic Security of Secure WS-Reliable Messaging Scenario)* If Enc is IND-CCA2-secure, if Sig is IND-CMA-secure, if SymEnc is IND-CPA-secure, and if Hash, when applied to pairs, is a pseudo-random function in its first argument, then *key integrity and key confidentiality* are cryptographically fulfilled for the scenario, i.e., if the protocol is run with a probabilistic polynomial-time adversary, the keys are authentic with overwhelming probability, and the keys are indistinguishable from fresh random keys given the view of the adversary. If additionally Mac is IND-CMA-secure, then *message integrity* and *no message disclosure* as well as all optional properties listed in Section C are cryptographically fulfilled. □

5   Further Related Literature

Work is currently underway on scaling-up formal analysis methods and tools to web services security protocols, e.g., [12–15, 30], although none of these works performs a cryptographic protocol analysis. In particular, the TulaFale tool [15] compiles descriptions of XML/SOAP-based security protocols and properties into the applied pi-calculus and then employs the ProVerif tool [16]. We considered employing also TulaFale for the automatic symbolic analysis of Secure WS-ReliableMessaging, but its input language would first need to be extended to express all the constructs of the profile, and we thus leave this analysis and the comparison with our own symbolic analysis as future work. Recent work has

also considered the automated analysis of XML-based web services: [38] presents a formal analysis of an encoding of the original XML messages into standard security protocol notation, showing that this encoding is without loss of attacks. Based on this encoding, the Casper/FDR tool can then check security properties for an unbounded number of sessions thanks to the employed data independence technique (which is similar to the abstraction techniques in TA4SP). The considered protocol, however, is simpler than the Secure WS-ReliableMessaging Scenario (e.g. no open-ended exchange of payload messages) and its analysis with Casper/FDR required simplifications of the message terms. It is thus not clear if the method of [38] could also work on complex protocols such as the one considered in this paper.

Another type of analysis of a web services security protocol is that of an interoperability profile of WS-Federation in [32]. The analyzed profile [35] is a passive requestor profile, i.e., the user is represented only by a browser. The emphasis therefore lies on treating a browser in a protocol security proof. The analysis is by hand, and as only signatures and secure channels occur as cryptographic primitives, there is not much discussion of detailed properties of the cryptographic primitives in web services.

6 Conclusion and Outlook

We have given a symbolic and a cryptographic analysis of the security of the new Secure WS-ReliableMessaging Scenario, which constitutes the first web services scenario to combine security elements with elements of another quality-of-service standard. The results of both analyses are positive, i.e., they are proofs as far as the techniques faithfully represent the standards; these restrictions concern the cryptographic primitives and in the symbolic case the analysis settings. Our symbolic analysis is a further step in the use of formal proof tools for the validation of security protocols and web services under the perfect cryptography assumption. Our cryptographic analysis constitutes an important first step to reason about the security of web services in the more realistic setting where the perfect cryptography assumption is replaced by the complexity-theoretic definitions of cryptography. Some of the cryptographic results are of more general applicability in web-service security than for the specific settings analyzed here.

As future work on the symbolic side, we have begun considering additional symbolic analysis settings, as well as employing abstraction techniques for carrying out unbounded verification. To this end, it would be particularly interesting not only to employ AVISPA's TA4SP, but also to investigate the relationships and possible complementarity of our analysis with an analysis carried out by TulaFale/ProVerif, especially since the model-checkers that we used implement different techniques than those of ProVerif (which combines symbolic representations based on first-order logic and abstractions). Moreover, it would be of great help to be able to exploit the automatic compilation provided by TulaFale and we will thus investigate how to do so for the AVISPA Tool. We believe that the work of [31] will be helpful here, as it provides a preliminary translation procedure from protocol descriptions in HLPSL to descriptions in the applied pi calculus, which thus allows one to apply the ProVerif tool to some existing HLPSL protocol specifications.

On the cryptographic side, it would be interesting to see in which respect one can weaken the security requirements imposed on the cryptographic primitives without invalidating the security properties. Furthermore, we intend to apply our techniques to other profiles and scenarios and possibly even to a policy-based analysis similar to [14] on the symbolic side.

13

References

1. M. Abadi and P. Rogaway. Reconciling two views of cryptography: The computational soundness of formal encryption. In *Proc. 1st IFIP TCS*, LNCS 1872, pp. 3–22. Springer, 2000.
2. J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *Proc. EUROCRYPT 2002*, LNCS 2332, pp. 83–107. Springer, 2002.
3. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P.-C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *Proc. CAV'2005*, LNCS 3576, pp. 281–285. Springer-Verlag, 2005. URL: www.avispa-project.org.
4. M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE CSFW*, 2004.
5. M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations (extended abstract). In *Proc. 10th ACM CCS*, pp. 220–230, 2003. Full version in IACR Cryptology ePrint Archive 2003/015, Jan. 2003, http://eprint.iacr.org/.
6. M. Backes, B. Pfitzmann, and M. Waidner. Symmetric authentication within a simulatable cryptographic library. In *Proc. 8th ESORICS*, LNCS 2808, pp. 271–290. Springer, 2003.
7. M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In *Proc. 1st TCC*, LNCS 2951, pp. 336–354. Springer, 2004.
8. D. Basin, S. Mödersheim, and L. Viganò. OFMC: A Symbolic Model-Checker for Security Protocols. *International Journal of Information Security*, 4(3):181–208, 2005.
9. G. Bella, F. Massacci, and L. C. Paulson. Verifying the SET Purchase Protocols. *Journal of Automated Reasoning*, to appear.
10. M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *Proc. ASIACRYPT 2000*, LNCS 1976, pp. 531–545. Springer, 2000.
11. M. Bellare and P. Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient constructions. In *Proc. ASIACRYPT 2000*, LNCS 1976, pp. 317–330. Springer, 2000.
12. K. Bhargavan, R. Corin, C. Fournet, and A. Gordon. Secure sessions for web services. In *ACM Workshop on Secure Web Services (SWS)*. ACM Press, to appear, 2004.
13. K. Bhargavan, C. Fournet, and A. Gordon. A semantics for web service authentication. In *Proc. 31st POPL*, pp. 198–209, 2004.
14. K. Bhargavan, C. Fournet, and A. Gordon. Verifying policy-based security for web services. In *Proc. 11th ACM CCS*, pp. 268–277, 2004.
15. K. Bhargavan, C. Fournet, A. Gordon, and R. Pucella. TulaFale: A security tool for web servics. In *Proc. 2nd FMCO*, 2003. To appear in Springer LNCS, Revised Lectures, 2004.
16. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE CSFW*, pp. 82–96, 2001.
17. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Static validation of protocol narration. *Journal of Computer Security*, to appear.
18. D. Box, F. Curbera *et al.* Web Services Addressing (WS-Addressing), Aug. 2004.
19. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple object access protocol (SOAP) 1.1, May 2000.
20. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *Proc. 30th STOC*, pp. 209–218, 1998.
21. R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334, 2004.

22. Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron. A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In *Proc. Workshop on Specification and Automated Processing of Security Requirements (SAPS'04)*, pp. 193–205. Austrian Computer Society, 2004.

23. J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997.

24. H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proc. 12th ESOP*, vol. 2618 of *LNCS*, pp. 99–113. Springer, 2003.

25. D. Davis, C. Ferris, V. Gajjala, K. Gavrylyuk, M. Gudgin, C. Kaler, D. Langworthy, M. Moroney, A. Nadalin, J. Roots, T. Storey, T. Vishwanath, and D. Walter. Secure WS-ReliableMessaging scenarios, Apr. 2005. `ftp://www6.software.ibm.com/software/developer/library/ws-rmseconscenar%io.doc`.

26. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

27. B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR. In *Proc. Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

28. D. F. Ferguson, T. Storey, B. Lovering, and J. Shewchuk. Secure, reliable, transacted Web Services – architecture and composition, Oct. 2003. Available at `http://www-106.ibm.com/developerworks/webservices/library/ws-securtrans%/`.

29. C. Ferris, D. Langworthy *et al.* Web Services Reliable Messaging Protocol (WS-ReliableMessaging), Feb. 2005.

30. A. Gordon and R. Pucella. Validating a web service security abstraction by typing. In *Proc. 1st ACM Workshop on XML Security*, pp. 18–29, 2002.

31. A. Gotsman, F. Massacci, and M. Pistore. Towards an Independent Semantics and Verification Technology for the HLPSL Specification Language. *Electronic Notes in Theoretical Computer Science 135(1):59–77 (Proceedings of the Workshop on Automated Reasoning for Security Protocol Analysis, ARSPA 2005)*, 2005.

32. T. Groß and B. Pfitzmann. Proving a WS-Federation Passive Requestor profile. In *ACM Workshop on Secure Web Services (SWS)*. ACM Press, to appear, 2004.

33. M. Gudgin, A. Nadalin *et al.* Web Services Secure Conversation Language (WS-SecureConversation), Feb. 2005.

34. M. Gudgin, A. Nadalin *et al.* Web Services Trust Language (WS-Trust), Feb. 2005.

35. M. Hur, R. D. Johnson, A. Medvinsky, Y. Rouskov, J. Spellman, S. Weeden, and A. Nadalin. Passive Requestor Federation Interop Scenario, Version 0.4, Feb. 2004. `ftp://www6.software.ibm.com/software/developer/library/ws-fpscenario2.d%oc`.

36. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and verifying security protocols. In *Proc. LPAR 2000*, LNCS 1955, pp. 131–160. Springer, 2000.

37. C. Kaler *et al.* Web Services Security (WS-Security), version 1.0, Apr. 2002.

38. E. Kleiner and A. Roscoe. On the relationship of traditional and web services security protocols (extended abstract). Unpublished manusscript, available from `http://web.comlab.ox.ac.uk/oucl/work/eldar.kleiner/`, 2005.

39. H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL). In *Proc. CRYPTO 2001*, LNCS 2139, pp. 310–331. Springer, 2001.

40. P. Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proc. 25th IEEE Symposium on Security & Privacy*, pp. 71–85, 2004.

41. G. Lowe. A hierarchy of authentication specifications. In *Proc. 10th IEEE CSFW*, pp. 31–43, 1997.

42. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proc. 1st TCC*, LNCS 2951, pp. 133–151. Springer, 2004.

43. D. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9:47–74, 2001.

44. M. Turuani. *Sécurité des Protocoles Cryptographiques: Décidabilité et Complexité*. Phd, Université Henri Poincaré, Nancy, December 2003.

A Illustrative Description of the Individual Protocol Steps

In this section, we provide a flow-by-flow description of the Secure WS-ReliableMessaging Scenario, based on Figure 3-5 for the steps of the original protocol, and based on Figure 6 for the auxiliary protocol steps.

A.1 Steps of the Original Protocol

1. RequestSecurityToken : The client first generates a fresh symmetric encryption key k and a fresh nonce N and computes $body_1 = \mathsf{SymEnc}_k(\mathsf{RST}, S, N)$. It then computes a signature $SigConf$ over $(ID_1, S, \mathsf{RST}, C, body_1, Cert_C)$ using its key sks_C. (The name $SigConf$ reflects that this signature is called a signature confirmation in the scenario description since the service will include this signature in a subsequent protocol step to confirm that it successfully received the signature.) The client then computes $header_1 = \mathsf{Enc}_S(k), \mathsf{SymEnc}_k(SigConf)$, where the key pke_S used to compute Enc_S is part of $Cert_S$, and sends the 1. RequestSecurityToken message.

2. RequestSecurityTokenResponse : The service decrypts $\mathsf{Enc}_S(k)$ yielding k, then decrypts $\mathsf{SymEnc}_k(SigConf)$ yielding $SigConf$. The service extracts $Cert_C$ from $SigConf$, validates that $Cert_C$ is a valid certificate with respect to the public key pks_{CA} and verifies the validity of $SigConf$ with respect to the verification key pks_C contained in $Cert_C$. The service then decrypts $body_1$ and aborts if this is not possible or the resulting cleartext is not of the correct format (RST, S, N). The service then generates a fresh key k' and a fresh nonce N^*, sets $sk := \mathsf{Hash}(N, N^*)$, and selects a unique key ID ID_{sk} for sk. The service then computes $body_2$ and $header_2$ as shown in the protocol description, where the encryption Enc_C is computed with the public encryption key pke_C that is contained in the certificate $Cert_C$, and sends the 2. RequestSecurityTokenResponse message.

3. CreateSequence : The client decrypts $\mathsf{Enc}_C(k')$ yielding k'. It then decrypts the remaining encrypted message parts with respect to k', checks if the plaintexts are of the correct form, and checks if the new signature is valid with respect to the public key pks_S in the certificate $Cert_S$. If so, the client extracts the nonce N^* and the key identifier ID_{sk} and computes the shared master key as $sk := \mathsf{Hash}(N, N^*)$. The client may then open new message sequences using the master key sk by first selecting two fresh nonces N_1, N_2 and by computing keys $sk_i := \mathsf{Hash}(N_i, sk)$ for $i = 1, 2$. The client then computes $body_3$ and $header_3$ according to the protocol description and sends the 3. CreateSequence message.

4. CreateSequenceResponse : The service answers the sequence creation by first computing the keys sk_i from sk and $Session$ and by decrypting $header_3$ with key sk_2 and by checking the validity of the contained message authentication code with respect to sk_1. If the code is over a message of the correct form, the service computes $body_4$ and $header_4$ according to the protocol description and sends the 4. CreateSequenceResponse message.

5. PayloadMessage : Once the client receives a response that the sequence has been established, it can start to send payloads within that sequence. To uniquely identify a sequence, the client first selects a unique ID_{Seq} denoting the ID of the sequence. The WS-ReliableMessaging standard mandates that payloads be equipped with successive natural

numbers. The client can send the first payload m by constructing the message according to the protocol description with n instantiated to 1. Honest clients are furthermore required to be consistent in that they never send different payloads having the same sequence number, e.g., as a response to a message resend inquiry. In the original interoperability scenario, this message is called PingMessage because for concrete testing, a concrete payload, ping, was also chosen.

6. SequenceAcknowledgment : At any point in time after the service has sent the 4. CreateSequenceResponse message and before it receives a 7. TerminateSequence message (cf. below), the service is allowed to acknowledge receipt of already received payloads. This is modeled in the 6. SequenceAcknowledgment message. Here n denotes either a single natural number denoting an acknowledgment for the message with sequence number n, or a range of numbers indicating an acknowledgment of all messages whose sequence number is within this range.

7. TerminateSequence : If the client has received acknowledgments for all messages within a sequence, it terminates the sequence by sending the 7. TerminateSequence message to the service. This indicates to the service that the transmission of the respective messages within the considered sequence has been successfully completed.

8. CancelSecurityToken : After a sequence has been terminated, the client can furthermore ask to cancel the validity of the master key sk, i.e., it sends a message indicating to the service that the key sk and all its derivates should be no longer considered for future sessions. This is modeled using the 8. CancelSecurityToken message.

9. CancelSecurityTokenResponse : The service responds to the key cancellation by sending a 9. CancelSecurityTokenResponse message, indicating that the previous message has been successfully received and that the master key sk has been revoked.

A.2 Steps of the Extension of the Protocol

5.1. NotAcknowledged : After the service received a 3. CreateSequence message and before it receives a 7. TerminateSequence message, the service can ask the client to resend a message that it has not received yet. The corresponding message closely resembles the 5. PayloadMessage where the number n denotes the number of the message that should be re-sent. If a client receives a message of that form, it re-sends the corresponding 5. PayloadMessage for the considered number and message.

5.1^*. AckRequested : Similar to the previous flow, the client can request acknowledgment for a message it previously sent to the service and for which it did not get an acknowledgment. If the service receives such a request, it responds with a 6. SequenceAcknowledgment message if the corresponding message has been received, otherwise with a 5.1. NotAcknowledged message.

B More About the Symbolic Security Analysis

B.1 The AVISPA Tool.
The *AVISPA Tool* [3] is a push-button tool for the automated validation, under the assumption of perfect cryptography, of industrial-scale Internet security-sensitive protocols and

applications. It provides a modular and expressive formal language for specifying protocols and their security properties, and integrates different back-ends that implement a variety of state-of-the-art automatic analysis techniques, ranging from *protocol falsification* (by finding an attack on the input protocol) to *abstraction-based verification* methods for both finite and infinite numbers of sessions.

More specifically, a user interacts with the AVISPA Tool by specifying a *security problem* (a protocol paired with a security property that it is expected to achieve) in the *High-Level Protocol Specification Language HLPSL* [22], which is an expressive, modular, role-based, formal language that allows for the specification of control flow patterns, data structures, alternative adversary models, complex security properties, as well as different cryptographic operators and their algebraic properties. The AVISPA Tool automatically translates a user-defined security problem into an equivalent specification written in the rewrite-based formalism *Intermediate Format IF*. An IF specification describes an infinite-state transition system amenable to formal analysis: this specification is input to the back-ends of the AVISPA Tool, which implement a variety of techniques to search the corresponding infinite-state transition system for states that represent attacks on the intended properties of the protocol.

As we remarked above, for our analysis of the Secure WS-ReliableMessaging Scenario, we have employed the back-ends OFMC [8] and CL-AtSe [44], which employ a number of automated reasoning and simplification techniques that enable them to handle protocols with complex message terms and in particular to model the Secure WS-ReliableMessaging Scenario in its full complexity, without simplification of the messages that are exchanged. Although the complexity of the Scenario prevents the usage of the back-ends SATMC and TA4SP, we hope to soon be able to report on the analysis with these back-ends as well. In particular, if analysis with TA4SP succeeded, then that would prove that the protocol is safe for secrecy goals for any number of sessions (by over-approximation).

B.2   The Model.

The Secure WS-ReliableMessaging Scenario has a structure that is far more complex than standard security protocols. Nonetheless, thanks to its expressiveness, HLPSL allows us to completely model the protocol, i.e., to provide a formal specification of the complex interactions between the two honest parties, which we can model as two separate client and service *programs* that communicate over an insecure network controlled by a Dolev-Yao adversary.

However, such a model is too complex for automated analysis as even for a limited number of sessions, the set of permissible interleavings of sending and receiving events is enormous. For instance, the messages sent by the client may arrive in any order at the service. Additionally, both the client and the service can send "administrative" messages, i.e., acknowledge messages, request the retransmission of messages, or request the acknowledgment of messages. An important part of modeling the protocol in a way feasible for automated analysis has thus been the search for a way to restrict the number of interleavings without excluding attacks, i.e., such that any attack on the original protocol is possible also on the simplified version.

We perform a step-by-step simplification of the client and service programs, and we argue that these simplifications do not exclude any attacks. Note that the simplified (restricted) version of the protocol that we obtain in this way is only useful for the formal analysis, not

18

for the practical deployment of the protocol: for instance, since a Dolev-Yao adversary can replay old messages arbitrarily if this is necessary to mount an attack, we can restrict the model to client programs that never retransmit old messages.

The following discussion will allow us to sketch the main ideas behind our HLPSL specification. We do lack space, however, to give the HLPSL specification in full here due to its complexity and the amount of explanation that would be necessary. As a concrete example, we will give an excerpt of our specification in the next subsection.

Simplifying the Client. First, it is not a restriction if the client sends in one transition all the messages that it wishes to transmit via the 5. PayloadMessage step as soon as it has received the 4. CreateSequenceResponse message. Second, the client can neglect any requests of step 5.1. NotAcknowledged from the service to retransmit messages, since the Dolev-Yao adversary has seen all messages and can thus replay them to the service if this is necessary for an attack. Hence, we can consider a simplified client program that, having sent all its payload messages, simply waits for acknowledgment messages (6. SequenceAcknowledgment) or, after timing out, requests acknowledgment from the service (5.1*. AckRequested). Thirdly, since the Dolev-Yao adversary can intercept all responses from the service, it might deliberately make the client produce acknowledge request messages. Hence, we can assume that the adversary can obtain acknowledge request messages of step 5.1*. AckRequested for every payload message. No attacks are therefore excluded if the client program sends with every 5. PayloadMessage also an 5.1*. AckRequested message.

These simplifications yield a client program that behaves as follows in every message sending phase: it sends all payload messages together with the corresponding requests for acknowledgment in one step, then waits until all messages are acknowledged, and finally sends a 7. TerminateSequence message.

Simplifying the Service. The first simplification concerns the 5.1. NotAcknowledged message. Whenever the service receives 5.1*. AckRequested, but the corresponding payload message has not yet been received, it answers with 5.1. NotAcknowledged to make the client send the payload message again. An adversary can therefore obtain the 5.1. NotAcknowledged message for every payload message sent by the client since, by the client simplification described above, it also has the respective messages of step 5.1*. AckRequested. We thus let the service generate, and send to the adversary, all such 5.1. NotAcknowledged messages as soon as the service has received the first 5. PayloadMessage. This simplification is not a restriction, but a slight over-approximation of the situations that can occur. In general, it can happen that such an over-approximation introduces *false attacks*, i.e., ones that do not work in the original model, but this has not happened in the analysis of this protocol.

The second simplification concerns the 6. SequenceAcknowledgment message, which we deal with in a similar way. As soon as the service has received all 5. PayloadMessage messages up to an index i, the adversary can easily obtain from him the 6. SequenceAcknowledgment by sending the 5.1. AckRequested message (which, as remarked before, the adversary knows at this time). We thus can simplify the service to always directly send the sequence acknowledgment messages as soon as possible, i.e., as soon as all messages up to a certain index are received.

The third simplification concerns the reception of 5.1*. AckRequested messages by the service. The service can reply to such a request in two different ways: if it has already re-

ceived all the payload messages for which the acknowledgement is required, then it will send a 6. SequenceAcknowledgment message; otherwise, it will send a 5.1. NotAcknowledged message for all payload messages it has not yet received. However, all these possible reply messages by the service are already known to the adversary by the first and second simplification above. We thus do need to process such 5.1*. AckRequested messages from the service program.

Let us now consider the 7. TerminateSequence message, which should only be generated by an honest client when it has received an acknowledgment for the entire sequence. We cannot, however, directly derive a simplification of the service program from this, as we cannot exclude that the adversary for some reason can somehow generate such a message (possibly using messages from other protocol runs) and produce a confusion between service and client. We therefore declare as an additional goal of our analysis that the service can never receive a 7. TerminateSequence message before it has sent an acknowledgment of all the messages that the client has sent. This is a reasonable additional goal and, given this goal, we can safely simplify the service so that this 7. TerminateSequence message can only be received after acknowledgment of all messages.

B.3 HLPSL Specification

As we remarked above, thanks to the expressivity of HLPSL, we have been able to formalize a complete, generic HLPSL-specification of the Secure WS-ReliableMessaging Scenario parameterized over the number of messages exchanged. Such a specification is, however, too complex for automated analysis and we have thus devised the simplifications described in the previous subsection. By applying these simplifications to our generic HLPSL specification, we have obtained a specification which has some of these parameters as hard-wired constants of the description. In particular, we have unrolled several loops and compressed them into a single transition, e.g. that the client sends out all payload messages in one transition.

The resulting specification is over 10 pages long and we thus only briefly discuss an excerpt here (formal details about HLPSL, which we will not describe here, can be found in [3, 22]). Figure 7 shows one transition of the client role. This transition can fire when the client's state variable `State` is 2, which is the case when it has sent out a 3. CreateSequence message and is waiting for a 4. CreateSequenceResponse. The pattern of this expected message is described in the `Rcv` predicate: all those variables that are primed are learned during this transition (and any value is acceptable in these positions) while all unprimed variables represent values determined already in previous steps. (Note that all identifiers that start with a lower-case letter are constants.)

If the preconditions are satisfied, the client can start sending payload messages (according to the 5. PayloadMessage format), but, as explained above, in our simplified model, the client will now send out all his payload messages at once, together with all possible requests for acknowledgment of the 5.1 ∗ .AckRequested format. Since, in our analysis settings, we have limited the number of payload messages to 3, the client sends out 6 messages in this transition (the `Snd` predicates). Note that primed variables like `Msg_one`' on the right-hand side of the transition that did not appear on the left-hand side and that are not part of

```
3. State = 2 /\
   Rcv(ID4'.C.csr.IDsk.N2.IDsk.N1.{{ID1.S.rst.C.{rst.S.N}_K.
       {C.PkeC.PksC}_inv(PksCA)}_inv(PksC)}_SK2.
       {mac(SK1.ID4'.C.csr.ID3.csr.IDSeq')}_SK2.csr.IDSeq')
   =|>
   State'=3 /\

   Snd(ID5_one'.S.ping.IDSeq'.one.IDsk.IDsk.N2.IDsk.N1.
       {mac(SK1.ID5_one'.S.ping.IDSeq'.one.
       {ping.Msg_one'}_SK2)}_SK2.{ping.Msg_one'}_SK2) /\
   Snd(ID51star'.C.ar.IDSeq'.one.IDsk.N2.IDsk.N1.
       {mac(SK1.ID51star'.C.ar.IDSeq'.one)}_SK2) /\

   Snd(ID5_two'.S.ping.IDSeq'.two.IDsk.IDsk.N2.IDsk.N1.
       {mac(SK1.ID5_two'.S.ping.IDSeq'.two.
       {ping.Msg_two'}_SK2)}_SK2.{ping.Msg_two'}_SK2) /\
   Snd(ID51star'.C.ar.IDSeq'.two.IDsk.N2.IDsk.N1.
       {mac(SK1.ID51star'.C.ar.IDSeq'.two)}_SK2) /\

   Snd(ID5_three'.S.ping.IDSeq'.three.IDsk.IDsk.N2.IDsk.N1.
       {mac(SK1.ID5_three'.S.ping.IDSeq'.three.
       {ping.Msg_three'}_SK2)}_SK2.{ping.Msg_three'}_SK2) /\
   Snd(ID51star'.C.ar.IDSeq'.three.IDsk.N2.IDsk.N1.
       {mac(SK1.ID51star'.C.ar.IDSeq'.three)}_SK2)

   request(C,S,n1,N1) /\ request(C,S,n2,N2) /\
   witness(C,S,one,Msg_one')     /\ secret(Msg_one',{C,S}) /\
   witness(C,S,two,Msg_two')     /\ secret(Msg_two',{C,S}) /\
   witness(C,S,three,Msg_three') /\ secret(Msg_three',{C,S})
```

Fig. 7. An excerpt of the HLPSL specification: a transition of the client role.

an equation (like `State'=3`) represent freshly generated values that the adversary cannot predict.[4]

Finally, the transition generates several predicates relevant for the security properties. With the `request` predicates, the client "requires" correspondence with the service on the nonces of the initial request-security-token exchange. Also, for every payload message, the client "declares" his intention to use it as the corresponding payload message and that it is supposed to be a secret shared only with the service.

C   Additional Security Properties and Refined Property Description

In this section, we give a more precise definition of the security properties informally stated in Section 2.2. We capture these properties as follows, adapted to the scenario:

---

[4] In some applications of the Scenario, the payload messages may be (partially) predictable or at least guessable. The secrecy of payload message thus refers only to whether the Scenario itself may reveal such values to an adversary.

- We treat no message alteration and authentication together as the following property *message integrity*: Let an honest service S accept a 5. PayloadMessage with the parameters m (in $body_5$), ID_{Seq}, and n, and let C be the client identity seen by the service in this session. Then if C is honest, it sent a 5. PayloadMessage with parameters m, ID_{Seq}, and n.
- No message disclosure is slightly refined as follows in the symbolic version: Let an honest client send a 5. PayloadMessage with the parameter m, and let S denote the service identity as seen by the client in this session. Then if S is honest, only S learns m, assuming C and S do not reuse m. Cryptographically, we have to restrict this by allowing the adversary to learn the length of m, while we can relax the precondition on non-reuse of m, see Section 4.
- Key integrity and confidentiality are to some extent auxiliary properties used to show the preceding properties. We refine them as follows: If an honest client derives a key sk in Step 3 and sees S as the service identity, and if S is honest, then S has derived the same key in Step 2 of a local session. If an honest server derives a key sk in Step 2 and sees C as the client identity, if C is honest, and if S then accepts a CreateSequenceResponse in the same session, then C has derived the same key in Step 3 of a local session. Furthermore, in both cases, symbolically no other party learns the key. Cryptographically, the key is indistinguishable from a random value at the time Step 2 has been finished.

The following properties are not explicitly required by the standard but they are useful as well:

- *Message freshness*: If a service accepts a message as defined under message integrity, then the client sent the corresponding message not before the service sent its CreateSequenceResponse message.
- *Key freshness for client*: If a client derives sk in Step 3, then the service started the session in which it derived sk not before Step 1 of the client's session.
- *Correct Confirmation*: If a client receives a message that confirms receipt of a message by an honest service, the service has indeed received this message.
- *Liveness*: If an honest client sends a message to an honest service, and if the underlying network guarantees that all messages are eventually delivered, then the client will eventually receive a confirmation of the receipt of the message by the service.
- *Successful session termination*: If an honest client and an honest service successfully terminated a session, both will no longer use the shared session keys sk_1 and sk_2.
- *Successful cancellation of master key*: If an honest client and an honest service successfully canceled the validity of the key sk, both will no longer use the shared key sk.

D   Postponed Proofs

This section contains the proof of Theorem 1.

D.1   Existing Results on Nesting Encryption and Authentication

The security of the scenario crucially relies on a specific combination of public-key encryption and signatures in the key-exchange phase (respectively, the use of hybrid encryption

with signatures), and specific combinations of symmetric encryption and authentication in the message-sending and termination phase. Before we start with the actual proof of security, we briefly review existing security results for these combinations.

The key-exchange phase relies on a sign-then-encrypt construct $\mathsf{Enc}(\mathsf{Sig}(\cdot))$, if we ignore the hybrid encryption for the moment. It has been shown by An et al. that the sign-then-encrypt construct guarantees authenticity and secrecy of the contained message if the encryption and signature schemes satisfy the security definitions we outlined above, and if the respective public keys have been exchanged in an authenticated manner [2].

In the message-sending and the termination phase, each message contains an authenticate-then-encrypt construct $\mathsf{Symenc}(\mathsf{Mac}(\cdot))$; the $5.\,\mathsf{PayloadMessage}$ furthermore contains an encrypt-then-authenticate construct $\mathsf{Mac}(\mathsf{SymEnc}(\cdot))$, since $body_5 = \mathsf{SymEnc}(\mathsf{PM}, m)$ is contained in the MAC. Both combinations have been extensively studied in the literature, in particular since they constitute crucial parts of SSL and IPSec. In particular, Krawczyk has shown that under the security definitions we pointed out before, the encrypt-then-authenticate construct guarantees both authenticity and secrecy of the contained message, and that the authenticate-then-encrypt approach guarantees the authenticity of the contained message [39].[5] The results rely on the assumption that both parties share secret encryption and authentication keys which are indistinguishable from randomly chosen keys given the view of the adversary.

D.2 Analysis of the Key-Exchange Phase

Cryptographic Secrecy and Integrity of the Nonces N and N^* after Step 2 We first show that in an execution between an honest client and an honest service, the nonces N and N^* exchanged in Step 1 and 2 of the protocol are cryptographically secure immediately after Step 2, i.e., they are authentically distributed between C and S and indistinguishable from random bitstrings of the same length, given the view of an arbitrary cryptographic attacker. In the following, we assume that all nonces be of a length $\mathsf{nonce\_len}$, which is a polynomial in the security parameter k.

The security of the nonces essentially follows since we have a hybrid encryption of signatures with a fresh symmetric key in Step 1 and Step 2, combined with the results of Section D.1. More precisely, instead of standard hybrid encryption where each symmetric key is used for exactly one encryption, we have a construct $\mathsf{Enc}_X(k), \mathsf{SymEnc}_k(m_1'), \ldots, \mathsf{SymEnc}_k(m_t')$ for a constant t. Security nevertheless holds under our assumptions: k is indistinguishable from a random key given $\mathsf{Enc}_X(k)$, and symmetric encryption with random keys is known to be secure for multiple encryptions. Hence we can safely replace the hybrid encryption with public-key encryptions $\mathsf{Enc}'_X(m_1'), \ldots, \mathsf{Enc}'_X(m_t')$, where Enc' denotes another secure public-key encryption scheme.

We now apply this transformation to the first two steps of the key-exchange phase. Next we exploit the results on the sign-then-encrypt construct, cf. Section D.1. The certificates $Cert_C$ and $Cert_S$ ensure that the respective public keys are exchanged in an authenticated

---

[5] It was also shown there that secrecy is not guaranteed in the authenticate-then-encrypt construct in general, but that secrecy can be achieved if the construct additionally satisfies the definition of integrity of ciphertexts; this is the case for, e.g., a standard MAC combined with CBC based on an arbitrary block cipher.

manner, under our assumption that the public key of the certification authority is authentically distributed to C and S. Now we can deduce that the nonces N and N^* are authentically exchanged in Step 1 and 2 since they are contained in a sign-then-encrypt construct with correctly distributed public keys. The additional occurrence of N in $SigConf$ and $body_1$ and of N^* in $body_2$, which are not contained in an sign-then-encrypt construct, are no problem for authenticity since they only impose additional tests upon message parsing. To establish the secrecy of N and N^*, note that the sign-then-encrypt construct guarantees that N and N^* remain cryptographically secret in $header_1$ and $header_2$. However, N and N^* occur as well in $body_1$ and $body_2$ (and in $SigConf$, but there only as $body_1$ again), respectively, so that we have to show that no information about N and N^* leaks from these body elements. This however is an easy consequence of the IND-CCA2 security of the public-key encryption system as it particularly ensures secrecy if the same key is used polynomially many times.

Secrecy and Integrity of the Key sk after Step 2 Although we now know that the pair of N and N^* has been authentically exchanged and that it is indistinguishable from a pair of random bitstrings of the same length, we cannot conclude in general that $sk :=$ $\mathsf{Hash}(N, N^*)$ is indistinguishable from a random bitstring of the same length if we only make normal cryptographic assumptions about the hash function.[6] An easy way to show the desired randomness would be to work in the random oracle model and to treat the hash function as a random oracle. However, that would introduce a non-justifiable idealization [20]. Fortunately, we do not need that, but can remain in the standard model of cryptography: We only require that the hash function, when applied to pairs, be a pseudorandom function in its first argument. We define that a key sk_ψ for the pseudorandom functions is chosen as a random string of length $\mathsf{nonce\_len}(k)$ and that the pseudorandom function works on messages of length $\mathsf{nonce\_len}(k)$ and, given a key sk_ψ and a message m, outputs $r := \mathsf{Hash}(m, sk\_\psi)$.

The definition of a pseudo-random function is that an adversary without access to sk_ψ, but given certain pairwise different messages $m_1, \ldots, m_n$, cannot distinguish the sequence of outputs $r_i := \mathsf{Hash}(m_i, sk\_\psi)$ of this function from a series of random values. In our case, sk_ψ, being the nonce N^*, is not completely random and secret. However, given the indistinguishability result of Section D.2, standard cryptographic arguments show that the pseudo-randomness of the values r_i still holds with N^* as the key, if no other information becomes known about N^* later. Indeed the nonce N^* is never reused in Steps 3-9 or outside this protocol.

The integrity of sk follows directly from that of N and N^* because it is the same deterministic function of them for both parties.

D.3 Analysis of the Payload-Sending Phase and the Termination Phase

A similar derivation process as for sk is made for the keys $sk_i = \mathsf{Hash}(N_i, sk)$ for $i = 1, 2$, where the same sk may be used for multiple such derivations, each time with new nonces N_1 and N_2. In the following, let $Keys(sk)$ denote the set of key-pairs (sk_1, sk_2) that C has derived from sk. Let further $Keys(sk)[1]$ and $Keys(sk)[2]$ denote the set of authentication

---

[6] As an example, let Hash denote a hash function that is one-way and collision-free for a concrete strong variant, and define $\mathsf{Hash}^*(m) := 2 \cdot \mathsf{Hash}(m)$. It is easy to show that Hash^* is secure iff Hash is secure but the hashes are always even and hence distinguishable from randomly chosen elements of the same length.

keys and the set of encryption keys contained in $Keys(sk)$, respectively. As we have shown that sk is indistinguishable from a random value after Steps 1 and 2, and as sk is not used in Steps 3-9 except in these derivations, the same arguments as in Section D.2 show that each $(sk_1, sk_2) \in Keys(sk)$ is indistinguishable from a pair of random values of this length for the adversary.

Authenticity of these keys is not so easy to show as the client only sends the nonces N_i in the unprotected message part $Session$. Assume that an adversary replaces one or both of them by a different value N_i^*. Then the service obtains a corresponding key sk_i^* that is indistinguishable from a random value for the adversary (by the same argument as for the correct nonce N_i). We first show that if $sk_1^* \notin Keys(sk)[1]$ or if $sk_2^* \notin Keys(sk)[2]$ the probability is negligible that decrypting $header_3$ containing ID_{sk} with key sk_2^* and testing the validity of the resulting plaintext with key sk_1^* succeeds for the service.

If $sk_1^* \notin Keys(sk)[1]$ then this would immediately contradict the security of the symmetric authentication scheme, which can be shown by an easy reduction where the new adversary on the pure authentication key chooses its own key sk_1^*. If $sk_2^* \notin Keys(sk)[2]$ we construct an adversary A^* against the pure encryption scheme. It constructs a message μ as C would do in Step 3 and computes a MAC of μ with a random key sk_1. It hands this MAC and an equally long string of zeros to the encryption oracle as two messages that it tries to distinguish. When the encryption oracle returns a ciphertext c of either the MAC or the zeroes, then A^* decrypts c with a random key sk_2. If the resulting plaintext π is a correct MAC for μ, then A^* decides that the MAC was decrypted. One can easily see that π is a correct MAC in this scenario with random keys with essentially the same probability as in the attack by A if the encryption oracle encrypted the MAC. In contrast, if it were a correct MAC with not negligible probability if the encryption oracle encrypted the zeroes, one could again break the MACs. Hence this leads to a not negligible distinguishing probability for A^*.

Now assume that an honest service S accepts a 5. PayloadMessage with the parameters m (in $body_5$), ID_{Seq}, and n, and let C be the client identity seen by the service in this session. We have shown in the previous step that m was authenticated using a key sk_1^* and encrypted with a key sk_2^* such that $sk_1^* \in Keys(sk)[1]$ and $sk_2^* \in Keys(sk)[2]$. Since ID_{Seq} has been freshly generated by the service after Step 3 and since sk_1^* is unknown to the adversary, we conclude that this MAC must have been newly created by either C or S; otherwise, we could perform a reduction against the message authentication scheme again. Since the service S does not create messages of the format of 5. PayloadMessage, this message must have been created by C and hence after C's completion of Step 4. The encryption was performed with the unique key sk_2' that satisfies $(sk_1^*, sk_2') \in Keys(sk)$. We finally show that $sk_2' = sk_2^*$, i.e., if the message of Step 5 was generated using the key sk_2' for a message m, then using the key sk_2^* instead must not give a message m', i.e., we have to have $\mathsf{SymEnc}_{sk_2'}(\mathsf{PM}, m) = \mathsf{SymEnc}_{sk_2^*}(\mathsf{PM}, m')$ and at the same time $\mathsf{SymEnc}_{sk_2'}(\mathsf{Mac}_{sk_1^*}(ID_5, S, \mathsf{PM}, (ID_{seq}, n), \mathsf{SymEnc}_{sk_2'}(\mathsf{PM}, m))) = \mathsf{SymEnc}_{sk_2^*}(\mathsf{Mac}_{sk_1^*}(ID_5, S, \mathsf{PM}, (ID_{seq}, n), \mathsf{SymEnc}_{sk_2^*}(\mathsf{PM}, m')))$ under the assumption that ID_{seq} is fresh. This means that two independent random encryption keys behave identically on large parts of the plaintext unchanged, which gives rise to a straightforward reduction against the security of the encryption scheme: The adversary gives the encryption oracle a message (PM, m) and an equally long string of zeros. It then decrypts the obtained ciphertext with a random key sk_2^*. If the resulting plaintext π is of the form (PM, m') for

some m', then the adversary decides that the first message was encrypted. Similar to the above proof, one can easily see that π is of this format with essentially the same probability as in the attack by A which hence leads to a not negligible distinguishing probability for the adversary attacking the encryption scheme.

For authentication, note that $body_5$ is contained within an authenticate-then-encrypt construct. This guarantees the authenticity of $body_5$ and hence of m since we already showed that the keys sk_1 and sk_2 are shared secretly and authentically between C and S. The additional occurrence of m in $body_5$ outside of $header_5$ is no problem for authenticity since it only imposes additional tests upon message parsing.

For the secrecy of m, we cannot simply argue in the same way since the authenticate-then-encrypt construct does not necessarily guarantee secrecy. However, there is a nested encrypt-then-authenticate construct in $header_5$, since $body_5$ is contained in the MAC. Although this does not precisely match the standard construct $\mathsf{Mac}(\mathsf{SymEnc}(\cdot))$ but looks like $\mathsf{Mac}(ID_3, S, \mathsf{PM}, (ID_{Seq}, n), \mathsf{SymEnc}(\cdot))$, it is easy to see that this does not violate the secrecy guarantee for the encrypted message as long as it does not additionally occur in the MAC outside the encryption. Hence the message m contained in $header_5$ is secret. We finally have to consider $body_5$, which contains m and which is transmitted outside of $header_5$. We have shown that sk_2 is secret from the adversary. Hence the secrecy of m follows from the security definition of symmetric encryption.

We do not formally address the optional properties of message and key freshness, correct confirmation, and liveness as defined in Appendix C, but they are easily derivable from the proofs above or follow exactly the same proof pattern, respectively.