

Formal analysis of modern security protocols in current standards



Marko Horvat
Kellogg College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michaelmas 2015

*To Dubravka and my parents,
Darija and Berislav*

Acknowledgements

First of all, I would like to thank my supervisor at both ETH Zurich and the University of Oxford, Prof. Cas Cremers, for unlimited patience and helpful advice throughout my research. I would also like to thank my former co-supervisor, Prof. David Basin, for his guidance and the opportunity to do research at ETH Zurich, and my MSc mentor Prof. Mladen Vuković for providing me with interesting problems to think about during my studies at the University of Zagreb.

Next, I thank all my friends and colleagues who studied with me in Zagreb, Zurich, and Oxford. I thank Dr. Vedran Čačić who got me interested in all things formal and logical in the first place, and Filip Nikšić for sharing limitless back-and-forth reasoning over a pint. I am also indebted to Dr. Simon Meier and Dr. Benedikt Schmidt for helping me with my first steps in approaching security and understanding their amazing Tamarin prover, and Dr. Michèle Feltz for some great discussions concerning our joint project. I thank Sam Scott and Thyla van der Merwe for our fruitful collaboration on TLS 1.3, and their readiness to double-check the corresponding part of this thesis. Another big thank you goes to Esma Balkir, Astrid Erber, Dr. Ivan Flechais, Mobeen Iqbal, and Andrew Paverd for helping me get my thoughts in order by constructively commenting on slides for my talks, and to Katriel Cohn-Gordon, Martin Dehnel-Wild, Luke Garratt, and Dr. Ralf Sasse for diligently proof-reading parts of my thesis.

Last, but by no means least, I thank those already mentioned and Ognjen Marić, Anna Niedbala, Grgur Petric Maretić, Dr. Joel Reardon, Martin Strohmeier, and Prof. Frank Wood for all the enjoyable discussions and laughter-filled moments we shared. Everyone I left out will have to excuse me for not filling the rest of this thesis with names alone, which I gladly would have done if I could.

Abstract

While research has been done in the past on evaluating standardised security protocols, most notably TLS, there is still room for improvement. Modern security protocols need to be rigorously and thoroughly analysed, ideally before they are widely deployed, so as to minimise the impact of often creative, powerful adversaries. We explore the potential vulnerabilities of modern security protocols specified in current standards, including TLS 1.2, TLS 1.3, and SSH. We introduce and formalise the threat of Actor Key Compromise (AKC), and show how this threat can and cannot be avoided in the protocol design stage. We find AKC-related and other serious security flaws in protocols from the ISO/IEC 11770 standard, find realistic exploits, and harden the protocols to ensure strong security properties. Based on our work, the ISO/IEC 11770 working group is releasing an updated version of the standard that incorporates our suggested improvements. We analyse the unilaterally and mutually authenticated modes of the TLS 1.3 Handshake and Record protocols according to revision 06 of their specification draft. We verify session key secrecy and perfect forward secrecy in both modes with respect to a powerful symbolic attacker and an unbounded number of threads. Subsequently, we model and verify the standard authenticated key exchange requirements in revision 10. We analyse a proposal for its extension and uncover a flaw in it, which directly impacts the draft of revision 11.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Our approach	3
1.3	Contributions	4
1.4	Thesis overview	6
2	Background	7
2.1	Notation	7
2.2	Scyther framework	8
2.2.1	Protocol specification in Scyther	8
2.2.2	Scyther execution model and security properties	13
2.3	Tamarin framework	18
2.3.1	Brief overview of the Tamarin framework	18
3	Actor Key Compromise	20
3.1	Introduction	20
3.2	Formalising actor key compromise	24
3.3	Achieving AKCS by transformation	33
3.3.1	Achieving AKCS of secrecy	34
3.3.2	Achieving AKCS of agreement	37
3.4	Impossibility results	39
3.5	Case studies	42
3.5.1	Needham-Schroeder-Lowe	43
3.5.2	ITU-T X.509 family	44
3.5.3	SSH	46
3.5.4	Mutually authenticated TLS	47
3.5.5	Unilateral TLS combined with authorisation protocols	49
3.6	Conclusions	50

4	Improving the ISO/IEC 11770 standard	52
4.1	Introduction	52
4.2	Background on ISO/IEC 11770	54
4.2.1	Protocols	54
4.2.2	Security properties and threat model of the standard	57
4.3	Formally modelling the protocols and properties	59
4.3.1	Protocol specification	59
4.3.2	Specifying security properties	60
4.4	Results of the formal analysis	63
4.4.1	Main analysis results	63
4.4.2	Implications for properties claimed in the standard	65
4.4.3	Key Compromise Impersonation (KCI) results	71
4.4.4	Unknown Key Share (UKS) results	73
4.5	Recommendations	75
4.6	Conclusions	77
5	Formal analysis of TLS 1.3	79
5.1	Introduction	79
5.2	TLS 1.3, rev 06 overview	81
5.2.1	TLS 1.3 design rationale	81
5.2.2	TLS 1.3, rev 06 technical details	84
5.3	Modelling the protocols in TLS 1.3, rev 06	87
5.4	Security properties and threat model	91
5.5	Formal analysis of TLS 1.3, rev 06	93
5.6	Formal analysis of TLS 1.3, rev 10	95
5.6.1	Differences between rev 06 and rev 10	95
5.6.2	State machine and Tamarin model changes	96
5.6.3	Analysis and proof automation	99
5.7	Post-handshake client authentication	101
5.7.1	Attack on client authentication in PSK mode	101
5.7.2	Underlying cause and mitigation	103
5.8	Conclusions	104
6	Related work	106
6.1	Actor Key Compromise	106
6.2	ISO/IEC 11770	108
6.3	TLS 1.3	109

7	Conclusions	111
8	Future work	114
	Bibliography	116
	Index	127

List of Figures

2.1	Example protocol 1.	13
2.2	Example protocol 2.	13
2.3	Operational semantics for protocol (Π, type)	15
3.1	Challenge-response protocol example	21
3.2	Secrecy and agreement example	22
3.3	Needham-Schroeder-Lowe protocol	22
3.4	Attack on Needham-Schroeder-Lowe protocol	23
3.5	Protocol and attack showing difficulties in defining AKC attack notion	25
3.6	KCIR as AKCS of secrecy and non-injective agreement	30
3.7	Transforming Π for secrecy of m	34
3.8	Transforming Π for non-injective agreement on n	38
3.9	NSL-AKC protocol	43
3.10	ITU-T X.509 three-message protocol	44
3.11	AKCS version of ITU-T X.509 three-message protocol	44
3.12	ITU-T X.509 one-message protocol	45
3.13	AKCS version of ITU-T X.509 one-message protocol	45
3.14	SSH Transport Layer protocol	46
3.15	Abstract depiction of mutually authenticated TLS-RSA	47
3.16	Abstract unilateral TLS with client authentication over TLS channel	49
4.1	Key management protocol example	53
4.2	Protocol 2-12 with optional parts	56
4.3	Protocol 3-KA-11	57
4.4	Scyther input file for 2-12 with confirmation messages and claimed properties.	60
4.5	Entity authentication attack on protocol 2-12 with optional parts	68
4.6	Protocol 2-11	69
4.7	Protocol 2-11 key authentication attack	69

4.8	Protocol 3-KT-6 combined key variant with $Text_1$ optional field . . .	70
4.9	3-KT-6 combined key variant with $Text_1$ optional field key confirmation attack	71
4.10	Protocol 3-KA-8	73
4.11	Unknown key share attack on protocol 3-KA-11	74
4.12	Protocol 2-10	75
4.13	Protocol 2-10 UKS attack	75
5.1	Full handshake	84
5.2	Full handshake with mismatched parameters	86
5.3	Resumed handshake	86
5.4	Simplified client and server state machines for TLS 1.3, rev 06	88
5.5	Simplified Mealy state machine for client in our TLS 1.3, rev 06 model	89
5.6	Rule C.1 in our TLS 1.3, rev 06 model	90
5.7	Simplified client and server state machines for TLS 1.3, rev 10	97
5.8	Tamarin model excerpt with lemmas from Table 5.1	99
5.9	Client impersonation attack on TLS 1.3 rev 10 if delayed client authen- tication allowed in PSK mode	102

Chapter 1

Introduction

In this thesis, we analyse widely used security protocols, some of which constitute critical parts of the Internet infrastructure. We motivate our work in Section 1.1 and provide the details of our approach in Section 1.2. We explain our contributions in Section 1.3 and give an overview of the thesis in Section 1.4.

1.1 Motivation

Communication is one of the essential catalysts of civilisation. The especially important modern context of communicating applications often requires the exchanged data to be reliable, secret, authentic, anonymous, etc. Depending on their purpose, these applications may have pre-existing long-term keys they could use to protect their data, but the keys may need updating due to e.g. their inadequate size or possible compromise.

In order to ensure the usage of up-to-date cryptographic material, any number of communicating applications may decide to set up a shared *session key*. The session key then becomes a foundation for providing the applications with all kinds of useful security guarantees, some of which were mentioned above. This is due to the fact that a symmetric encryption algorithm can be used with the session key to symmetrically encrypt the data to be transmitted. However, in order for this to work, the key needs to be both *properly established* and *properly used*. For example, if the key was transmitted in plaintext, then someone who had been successfully eavesdropping on the communication would have discovered its value, and thus be able to decrypt messages encrypted with it. More interestingly, application data might also leak if the key used to encrypt it is prone to cryptanalysis, e.g. it is a supposedly random value generated with a large bias.

The standard method of establishing a secret session key involves running a *key exchange protocol*, for instance the TLS Handshake Protocol [28], or one of the key management protocols in the ISO/IEC 11770 standard [46, 47]. Protocols in general are distributed algorithms, and key exchange protocols are a type of protocol designed expressly for the purpose of transporting or agreeing on a session key in order to facilitate secure communication. A session key established by executing a key exchange protocol can be used in symmetric-key message transmission protocols, such as the TLS Record Protocol [28].

Key exchange protocols usually rely on two sources of secret information to bootstrap a shared secret: freshly generated random values, and long-term private keys. Real-world attackers sometimes manage to improve their chances of breaking a protocol by obtaining some of these pieces of information. For example, a pseudo-random number generator implementation used for fresh values might be predictable [10, 33, 38, 75]. Long-term keys can be obtained through coercion [81], implementation bugs [77], cryptanalysis [98], etc. State-of-the-art protocols are designed to resist the possibility of converting these vulnerabilities, perhaps even some combinations thereof, into actual attacks.

Some of the important security properties we are looking for in modern key exchange protocols are key secrecy, key agreement, Perfect Forward Secrecy (PFS) [29], Unknown Key Share (UKS) resilience [14], and Key Compromise Impersonation (KCI) resilience [13, 51]. PFS means that the compromise of long-term keys of all protocol participants does not allow an attacker to compute any session keys which were established prior to the compromise. UKS resilience means that all protocol participants know with whom they share their previously established session keys. KCI resilience means that the attacker can never use someone’s long-term private key to impersonate others in communication with the key’s owner. Here, impersonation is usually modelled as an attack on session key secrecy because this invariably leads to future misauthentication of application data protected only with the session key. Note that the attacker can always use compromised long-term keys to impersonate their owner in this sense. Our first research question explores a much more general scenario than KCI.

Research question 1: What security guarantees can an entity expect if its own long-term keys are compromised, and how can security protocols help achieve these guarantees?

The second thing that needs to be understood well in key exchange is proper key usage. Even if a session key was established using a KCI resilient key exchange protocol, it may be the case that the key leaks when it is later used in a second protocol. An attacker who could not impersonate protocol participants during the key exchange might be able to perform some sort of impersonation in the second protocol. Moreover, perhaps there is a property other than session key secrecy that can be violated in the same way. This possibility ties in with our second research question.

Research question 2: How do advanced adversaries, i.e. ones who can compromise all kinds of short-term and long-term secret values, impact security protocols in modern security standards, and how can we use formal approaches to improve them?

1.2 Our approach

In order to answer our research questions, we analyse all the protocols mentioned so far with respect to symbolic attackers who can eavesdrop on and actively tamper with messages transmitted over a network, compromise various kinds of secret information, and use this and publicly available information to infer new messages.

In a symbolic model, messages are represented by terms in a term algebra, and inferring new ones is precisely determined by a set of simple deduction rules (cf. Section 2.2.1). The rules are defined in such a way that encryption, decryption, signing, and signature verification can only be done with the corresponding keys. This view was introduced by Dolev and Yao [30], and is called *the perfect cryptography assumption*. Note that this assumption is much stronger than the standard indistinguishability assumptions that are common in cryptography. It represents a shift in focus where cryptographic primitives are assumed to be flawless, and matters such as the potentially very involved communication of unboundedly many concurrent threads come under scrutiny. This allows us to quickly obtain insightful results, usually aided by automated verification, but as with any model, realistic attacks may be missed. The reader can refer to [22] for more information.

In our analyses, we use two complementary approaches: manual proofs and automated protocol analysis. In Chapter 3, we manually prove strong, sweeping statements that later guide our custom, efficient fixes to protocols which are flagged as flawed by our automated protocol analysis. We use two state-of-the-art symbolic verification tools for security protocols: Scyther [23] and the Tamarin prover [69, 70, 94, 95]. Scyther is a model checker that allows for push-button verification of classes of protocols with respect to built-in properties and very powerful attacker models, and

features basic support for Diffie-Hellman-style protocols. The Tamarin prover uses multiset-rewriting techniques to allow for finer control of properties, attacker models, and Diffie-Hellman group operations. However, Tamarin is best suited to the in-depth analysis of single protocols due to the additional effort required to specify the protocol, and sometimes the adversarial capabilities. Hence, Scyther is our tool of choice when we explore *Actor Key Compromise* (AKC) in simpler protocols in Chapter 3, and for scripting the verification of batches of protocols in the ISO/IEC 11770 standard for key management, which we look at in Chapter 4. We use Tamarin to verify the more intricate protocols, such as the various versions of the TLS Handshake and Record Protocols in Chapter 3 and Chapter 5.

The formal models underlying these tools are both symbolic, but very different. We describe them in Chapter 2 in formal support of our analyses. Note that neither of the two models, or any model equivalent to them or more expressive, such as probabilistic polynomial-time Turing machines, can avoid the fact that some of the discussed properties are in general undecidable. This follows from the Post Correspondence Problem [32].

Despite this theoretical barrier, in practice we will be able to analyse very complex protocols. For instance, loops are a critical feature of the TLS 1.3 protocol, which we formally analyse in Chapter 5. Different revisions of this protocol specify retries due to parameter mismatches and different kinds of reconnection handshakes. The verification process therefore includes not just time-consuming, manually guided proofs for such more complicated protocols, but also coming up with the right definitions and creative lemmas in the best framework for the job.

1.3 Contributions

Our contributions include both theoretical and practical advances in the area of modern security protocols in current security standards. Here we briefly outline the most important of these contributions.

We provide the first systematic analysis of the consequences of compromising an entity’s secret key with respect to that entity’s security requirements, and propose countermeasures. We introduce the notion of Actor Key Compromise (AKC) and define the related notions of AKC security, resilience, and attack. Our definitions are independent of the choice of protocol, adversary, and property. We show that key compromise impersonation resilience is a specific instance of AKC security. We also provide constructive results showing how some AKC vulnerabilities can be avoided

in protocol design by using asymmetric encryption and signatures. We then prove impossibility results showing that a large class of authentication properties cannot be achieved under AKC by protocols that only use symmetric cryptography and hashing. When looking at AKC in practice, we analyse a set of protocols, including TLS and SSH, for their resilience against AKC. We find attacks on several protocols, including AKC attacks on mutually authenticated TLS-RSA as well as the combinations of unilateral TLS-RSA with authorisation protocols such as Apache’s `mod_auth_basic`, OAuth, and SAML. For mutually authenticated TLS, we implement and carry out our AKC attack against a fully patched Apache web server running TLS 1.2. We provide and verify concrete fixes for the vulnerable protocols.

We perform the first comprehensive analysis of Parts 2 and 3 of the ISO/IEC 11770 standard. The standard lacks a precise threat model, so we reverse-engineer one by considering the discussed security properties and what combinations of adversarial capabilities can satisfy them in some protocols while they falsify them in others. Our analysis against this threat model uncovers multiple previously unreported errors and weaknesses, including entity authentication failures, key confirmation failures, and forward secrecy failures. Additionally, we analyse vulnerabilities that lead to Unknown Key Share attacks and Key Compromise Impersonation attacks. For each of the discovered issues, we provide concrete recommendations for improving the standard. We explain that many of the discovered issues could have been prevented if the recent recommendations for protocols in related standards, in particular ISO/IEC 9798, had been applied to their counterparts in ISO/IEC 11770.

We model and analyse revision 06 of the specification draft of the TLS 1.3 Handshake and Record Protocols. We find that the protocols establish session key secrecy and Perfect Forward Secrecy in both the unilaterally and mutually authenticated modes, even when all possible interleavings of unboundedly many resumptions and application data transmissions are allowed, along with unboundedly many retries. The threat model captures a powerful, active adversary who can perform a range of kinds of key reveal, such as Session-Key Reveal, Actor Key Compromise, and Diffie-Hellman Exponent Reveal. We build on our protocol model to facilitate our analysis of rev 10, where we verify standard authenticated key exchange requirements and find a potential vulnerability in one of the proposals for its extension. More specifically, there is a client impersonation attack if unrestricted, post-handshake client authentication is added to rev 10.

All the above contributions will be discussed in detail in the remainder of the thesis. We explain its structure in the following section, which mostly follows our past

and forthcoming publications:

1. D. Basin, C. Cremers, and M. Horvat. Actor key compromise: Consequences and countermeasures. In *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF)*, pages 244–258, 2014.
2. C. Cremers and M. Horvat. Improving the ISO/IEC 11770 standard for key management techniques. In *Proceedings of the 1st International Conference on Security Standardisation Research (SSR)*, volume 8893 of Lecture Notes in Computer Science, pages 215–235. Springer, 2014.
3. C. Cremers and M. Horvat. Improving the ISO/IEC 11770 standard for key management techniques. *International Journal of Information Security (IJIS)*, doi: 10.1007/s10207-015-0306-9, 2015.
4. C. Cremers, M. Horvat, S. Scott, and T. van der Merwe. Automated Analysis and Verification of TLS 1.3: 0-RTT, Resumption and Delayed Authentication. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (S&P, Oakland)*, 2016 (accepted for publication).

1.4 Thesis overview

After detailing in Chapter 2 the formal frameworks underlying the tools we use in our analysis, we divide the rest of the thesis into three main parts: Chapters 3, 4, and 5. Chapter 3 explores the notion of Actor Key Compromise and its impact on modern security protocols, mostly by employing manual proofs and Scyther [23]. It additionally requires the Tamarin prover [95] for its advanced support of Diffie-Hellman. This part is based on a joint publication [7] with Basin and Cremers. Chapter 4 deals with improving the ISO/IEC 11770 standard for key management techniques, and is based on a joint publication [26] with Cremers. The tool used in the analysis is Scyther. Chapter 5 comprises our analysis of revision 06 of the TLS 1.3 specification [84] using Tamarin to deal with Diffie-Hellman and loops. It also details our work with Cremers, Scott, and van der Merwe on revision 10. We review related work in Chapter 6, discuss our conclusions in Chapter 7, and outline our planned future work in Chapter 8.

Chapter 2

Background

In this section, we provide background information on the symbolic frameworks underlying Scyther [23] and Tamarin [69, 70, 94, 95], the tools that we use throughout this thesis. The perfect cryptography assumption internal to both these tools is explained in Section 1.2. We occasionally refine the framework definitions for higher precision in our theoretical results, for example by adding fine-grained types to the Scyther execution model.

Note that our treatment of the Scyther framework in this chapter is fairly detailed because we use it to prove theoretical results in Chapter 3. These results can then be applied to whole classes of protocols supported by the framework in Chapter 4. In contrast to this approach, we present the Tamarin framework in just enough detail to explain how we use it to verify protocols in Chapter 3 and Chapter 5. We refer the reader to [23, 69, 70, 94, 95] for details, e.g. the specific verification algorithms employed by the respective tools.

2.1 Notation

Before we describe our formal frameworks, we give the notation we use for functions and sequences. For f a partial function from X to Y , we write $f : X \rightarrowtail Y$. If f is also a (total) function, i.e. it is defined for each element of X , we write $f : X \rightarrow Y$. For every partial function f , we denote its domain by $\text{dom}(f)$, its range by $f(\text{dom}(f))$, and $f \cup f'$ for the union of two partial functions with disjoint domains. We write $f[a \mapsto b]$ to denote an update of f , i.e. f' where $f'(a) = b$ and for all $x \in \text{dom}(f) \setminus \{a\}$, $f'(x) = f(x)$. Let S^* denote the set of all finite sequences of elements from S . We write $\langle s_1, \dots, s_n \rangle$ to denote the sequence of s_1 to s_n and define $\text{last}(\langle s_1, \dots, s_n \rangle) = s_n$. For $s = \langle s_1, \dots, s_n \rangle$, we write $e \in s$ if there exists an $i \in \{1, \dots, n\}$ such that $e = s_i$. Finally, we write $s.s'$ for the concatenation of the sequences s and s' .

2.2 Scyther framework

We extensively use Scyther in Chapter 3 and Chapter 4. For this reason, we explain the formal details of its underlying framework that are sufficient to understand our theoretical and practical results. We refer the reader to [23] for a complete description of the theory behind Scyther.

2.2.1 Protocol specification in Scyther

Our symbolic framework is based on [6,27], where the main building blocks for protocol specifications are roles. A protocol can have any finite number of roles, and is run by agents who execute those roles. Agents may execute each role multiple times, and every role can be executed by any agent. Concrete role instances that occur during protocol execution are called runs¹. While roles are built out of role events, runs consist of the instantiated counterparts of role events, which are called run events. Role and run events contain role and run terms, respectively. We assume a trusted initial key distribution of public/private key pairs, and of a shared symmetric key for each pair of agents. Such assumptions are studied in e.g. [15].

We assume the existence of pairwise disjoint, infinite sets *Role*, *Agent*, *Fresh*, *RID*, *Var*, and *Func* of roles, agent names, freshly generated terms (nonces, coin flips, etc.), run identifiers, variables, and function names (hash functions, constants, etc.). We also assume that there are infinitely many function names of arity 0 and infinitely many function names of arity 1. We denote by $\text{Const} \subseteq \text{Func}$ the set of all constants, i.e. the function names of arity 0. We assume that *RID* contains two distinguished run identifiers, *Test* and *rid_A*, which are respectively used to identify the *test run* and the *adversary run*. In the computational setting, the test run is the run under attack. We refer to it to define the adversary’s capabilities to perform different types of key reveal queries. Such queries can be executed by the adversary run, which is an auxiliary run that represents adversarial interference—it is not a concrete instance of any role.

In the following definition, we define the terms we use when specifying and executing protocols in our framework.

¹Runs are sometimes also called threads and sessions. We will use the former in later chapters, but we distinguish the latter from runs/threads when discussing the TLS protocol.

Definition 1 (Terms).

$$\begin{aligned}
\text{Term} ::= & \text{Role} \mid \text{Agent} \mid \text{Fresh} \mid \text{Fresh}^{\#\text{RID}} \mid \text{Var} \\
& \mid \text{k}(\text{Role}, \text{Role}) \mid \text{pk}(\text{Role}) \mid \text{sk}(\text{Role}) \\
& \mid \text{k}(\text{Agent}, \text{Agent}) \mid \text{pk}(\text{Agent}) \mid \text{sk}(\text{Agent}) \\
& \mid (\text{Term}, \text{Term}) \mid \{\text{Term}\}_{\text{Term}} \mid \text{Func}(\text{Term}^n)
\end{aligned}$$

We denote the symmetric long-term secret key shared between X and X' by $\text{k}(X, X')$, $\text{pk}(X)$ denotes X 's asymmetric long-term public key, and $\text{sk}(X)$ denotes the corresponding secret key. Pairing is not associative, and we write (a, b, c) to denote $((a, b), c)$. With $\{t_1\}_{t_2}$, we denote a signature, an asymmetric encryption, or a symmetric encryption of t_1 with t_2 , which depends on whether t_2 is an asymmetric secret key, an asymmetric public key, or any other term, respectively. The superscript n in $\text{Func}(\text{Term}^n)$ denotes that the arity $n \in \{0, 1\}$ depends on the function name. We omit the brackets if $n = 0$.

We define two subterm relations: the *syntactic* subterm relation, which does not take into account the position of a subterm within a term, and the *accessible* subterm relation, which only identifies potentially retrievable subterms, given knowledge of the appropriate keys.

Definition 2 (Syntactic subterm relation). *The syntactic subterm relation $\sqsubseteq$ is the reflexive, transitive closure of the smallest relation $\sqsubseteq$ on terms such that, for all $t_1, t_2 \in \text{Term}$, X_1, X_2 both from either Role or Agent, $j \in \{1, 2\}$, and $f \in \text{Func}$ of arity 1:*

$$\begin{aligned}
X_j &\sqsubseteq \text{k}(X_1, X_2) & X_1 &\sqsubseteq \text{pk}(X_1) & X_1 &\sqsubseteq \text{sk}(X_1) \\
t_j &\sqsubseteq (t_1, t_2) & t_j &\sqsubseteq \{t_1\}_{t_2} & t_1 &\sqsubseteq f(t_1)
\end{aligned}$$

Definition 3 (Accessible subterm relation). *The accessible subterm relation $\sqsubseteq_{\text{acc}}$ is the reflexive, transitive closure of the smallest relation $\sqsubseteq_{\text{acc}}$ on terms where for all $t_1, t_2 \in \text{Term}$, $t_1 \sqsubseteq_{\text{acc}} (t_1, t_2)$, $t_2 \sqsubseteq_{\text{acc}} (t_1, t_2)$, and $t_1 \sqsubseteq_{\text{acc}} \{t_1\}_{t_2}$.*

We extend both subterm relations to sets: for a term t and a set S , $t \sqsubseteq S$ ($t \sqsubseteq_{\text{acc}} S$) means that there exists a term $t' \in S$ such that $t \sqsubseteq t'$ ($t \sqsubseteq_{\text{acc}} t'$). We write $\text{vars}(t)$ for $\{x \in \text{Var} : x \sqsubseteq t\}$. We say that a term t is *ground* if $\text{vars}(t) = \emptyset$. We assume the existence of an inverse function on terms such that, for all $t \in \text{Term}$, if $t = \text{pk}(t')$ or $t = \text{sk}(t')$ for some t' , then $t^{-1} = \text{sk}(t')$ or $t^{-1} = \text{pk}(t')$, respectively; otherwise, $t^{-1} = t$.

We define *substitutions* as partial functions in the usual way, except that they are defined on $\text{Role} \cup \text{Var}$. We require that every substitution $\sigma : \text{Role} \cup \text{Var} \rightarrow \text{Term}$ satisfies $\sigma(\text{Role}) \subseteq \text{Agent}$. For finite substitutions σ such that $\text{dom}(\sigma) = \{x_1, \dots, x_n\}$ and $\sigma(x_i) = t_i$ for all $i \in \{1, \dots, n\}$, we write $\sigma = [t_1, \dots, t_n / x_1, \dots, x_n]$. If σ is infinite, we may write $\sigma = [t_1, t_2, \dots / x_1, x_2, \dots]$ to emphasise which terms σ assigns to x_1, x_2 , etc. When applying a (total) substitution to terms, we extend the substitution to an endomorphism on Term in the standard way. For $t, t' \in \text{Term}$ such that $t' \not\sqsubseteq t$, we use $\{t/t'\}$ to denote the function that replaces every occurrence of the subterm t' with the term t . We call such functions *replacements*.

We now define which terms can be inferred from a given set of terms, and refer the reader to Section 1.2 for the standard motivation behind this definition. We denote by $\vdash$ the smallest relation such that, for all $n \in \{0, 1\}$, $f \in \text{Func}$ of arity n , and $S \cup \{t_1, t_2\} \subseteq \text{Term}$, $\vdash$ respects the following rules:

$$\begin{array}{c}
\frac{}{S \vdash t_1} (t_1 \in S) \\
\\
\frac{S \vdash t_1 \quad S \vdash t_2}{S \vdash (t_1, t_2)} \quad \frac{S \vdash t_1 \quad S \vdash t_2}{S \vdash \{t_1\}_{t_2}} \quad \frac{}{S \vdash f}^{(n=0)} \quad \frac{S \vdash t_1}{S \vdash f(t_1)}^{(n=1)} \\
\\
\frac{S \vdash (t_1, t_2)}{S \vdash t_1} \quad \frac{S \vdash (t_1, t_2)}{S \vdash t_2} \quad \frac{S \vdash \{t_1\}_{t_2} \quad S \vdash t_2^{-1}}{S \vdash t_1}
\end{array}$$

We call the rules in the second row *composition rules* and those in the third row *decomposition rules*. If a term t can be derived from a set S in finitely many applications of the above rules with every branch of the derivation closed, we write $S \vdash t$ and say that S *infers* t . We call such a derivation a $\vdash$ -*derivation tree for t from S* . If there exists a $\vdash$ -derivation tree for t from S of height at most n , we write $S \vdash_n t$.

We define the set of *role terms* RoleTerm as the set of terms that have no subterms in $\text{Agent} \cup \{n^{\#rid} : n \in \text{Fresh}, rid \in \text{RID}\}$. We define the set of *run terms* RunTerm as the set of terms that have no subterms in $\text{Role} \cup \text{Fresh}$. We denote the set of all ground run terms with RunTerm_0 . We call $\text{Role} \cup \text{Agent} \cup \{x, x^{\#rid} : x \in \text{Fresh}, rid \in \text{RID}\} \cup \text{Var} \cup \text{Const}$ the set of *atomic terms*.

Next we define role and run events. We assume two infinite, disjoint sets are given, both pairwise disjoint from Term , Func , and RID . The two sets are Claim , the set of all claim names, which we use to specify claims of various security properties, and Label , which we use to label events. We additionally assume that $\{\text{alive}, \text{commit}, \text{running}, \text{secret}\} \subseteq \text{Claim}$.

Definition 4 (Role and run events). For all $R \in \text{Role}$ and $a \in \text{Agent}$, we define:

$$\begin{aligned} \text{RoleEvent}_R &:: \text{send}_{\text{Label}}(R, \text{Role}, \text{RoleTerm}) \\ &\quad | \text{recv}_{\text{Label}}(\text{Role}, R, \text{RoleTerm}) \\ &\quad | \text{claim}_{\text{Label}}(R, \text{Claim}[, \text{Role}][, \text{RoleTerm}]) \\ \text{RunEvent}_a &:: \text{send}_{\text{Label}}(a, \text{Agent}, \text{RunTerm}) \\ &\quad | \text{recv}_{\text{Label}}(\text{Agent}, a, \text{RunTerm}) \\ &\quad | \text{claim}_{\text{Label}}(a, \text{Claim}[, \text{Agent}][, \text{RunTerm}]) \end{aligned}$$

We also let $\text{RoleEvent} = \bigcup_{R \in \text{Role}} \text{RoleEvent}_R$ and $\text{RunEvent} = \bigcup_{a \in \text{Agent}} \text{RunEvent}_a$.

For example, the event $\text{send}_{l_0}(\text{Alice}, \text{Bob}, \{n^{\#rid}\}_{\text{pk}(\text{Bob})})$ signifies that Alice sends Bob a nonce $n^{\#rid}$, which is generated in the run rid and encrypted with Bob's public key.

Two other events can occur during protocol execution: **create** and **LKR**. They respectively denote creating a run of some protocol role and revealing the long-term keys of an agent to the adversary. We denote the set $\text{RunEvent} \cup \{\text{create}(R) : R \in \text{Role}\} \cup \{\text{LKR}(a) : a \in \text{Agent}\}$ with TraceEvent , and the set $\text{RoleEvent} \cup \text{TraceEvent}$ of all *events* with Event . We homomorphically extend all replacements and substitutions to events and sequences of terms and events. For every $\text{ev} \in \{\text{send}, \text{recv}, \text{claim}\}$, $l \in \text{Label}$, and event $e = \text{ev}_l(\cdot)$, we let $\text{evtype}(e) = \text{ev}$ and $\text{label}(e) = l$. We call ev the *event type* of e , and l the *label* of e . Additionally, if e is of the form $\text{ev}_l(\cdot, \cdot, m)$ or $\text{ev}_l(\cdot, \cdot, \cdot, m)$, where $m \in \text{Term}$, we write $\text{cont}(e) = m$ and call m the *contents* of e .

We require that every sequence of role events that occurs in a protocol specification satisfies some well-formedness conditions. In order to state these conditions, for all $S \in \{\text{Role}, \text{Agent}\}$, we define the set $\text{LTK}(x)$ of all long-term secret keys of $x \in S$ as $\{\text{sk}(x)\} \cup \bigcup_{x' \in S} \{\text{k}(x, x'), \text{k}(x', x)\}$. We also define the operator $\downarrow$ that selects the set of terms that are the contents of events of a particular type: for all $e \in \text{Event}$, $s \in \text{Event}^*$, and $\text{ev} \in \{\text{send}, \text{recv}, \text{claim}\}$, let $\langle \rangle \downarrow \text{ev} = \emptyset$ and

$$(\langle e \rangle.s) \downarrow \text{ev} = \begin{cases} \{\text{cont}(e)\} \cup (s \downarrow \text{ev}), & \text{if } \text{evtype}(e) = \text{ev}, \\ s \downarrow \text{ev}, & \text{otherwise.} \end{cases}$$

For all $s \in \text{Event}^*$ and $l \in \text{Label}$ such that l occurs in s , i.e. there exists $e \in s$ where $\text{label}(e) = l$, we define $\text{upto}(s, l)$ as the prefix of s , up to and including the first event labelled l .

Definition 5 (Well-formed sequence of role events for R). Let $R \in \text{Role}$. A sequence $s \in \text{RoleEvent}_R^*$ is well-formed for R if:

- all event labels in s are unique,
- for all $x \in \text{Var}$, if $e \in s$ is the first event in s such that $x \sqsubseteq \text{cont}(e)$, then $\text{evtype}(e) = \text{recv}$ and $x \sqsubseteq_{\text{acc}} \text{cont}(e)$ (every variable occurring in an event must be initialised in an accessible position in a **recv**), and
- for all $l \in \text{Label}$, $R' \in \text{Role}$, and $t \in \text{RoleTerm}$ such that $\text{send}_l(R, R', t) \in s$, we have that $\text{LTK}(R) \cup \{S, \text{pk}(S) : S \in \text{Role}\} \cup \{n \in \text{Fresh} : n \sqsubseteq (\text{upto}(s, l) \downarrow \text{send})\} \cup (\text{upto}(s, l) \downarrow \text{recv}) \vdash t$ (role R must be able to construct the contents of each of its **send** events).

When a message is received during protocol execution (defined in the next section), some terms may be stored in variables. A *type function* formalises which terms can be stored in which variable. A **recv** step can be executed only if each variable stores a term of its type.

Definition 6 (Protocol). Let $\Pi : \text{Role} \rightarrow \text{RoleEvent}^*$ be a partial function and $\text{type} : \text{Var} \rightarrow \mathcal{P}(\text{RunTerm}_0)$ a function. If for all $R \in \text{dom}(\Pi)$, $\Pi(R)$ is well-formed for R , we say that (Π, type) is a protocol.

Note that while there is no finite bound on the number of roles in our protocols, we are usually interested in one or two roles that are most relevant to the security property we are considering. In Sections 3.2, 3.3, and 3.4 of this chapter, we depict only these roles in message sequence charts.

We introduce two protocols that we use as running examples in Figures 2.1 and 2.2. Both protocols are two-role protocols, depicted as message sequence charts. In the first one, R sends to R' its identity and a freshly generated nonce n , encrypted with the public key of R' . In the second protocol, a signed hash of the payload is additionally transmitted. In both protocols, role R' claims the secrecy of the nonce n upon successful completion, i.e. that the adversary cannot infer it.

Let us assume that the protocol from Figure 2.1 is a key transport protocol, and that n is a fresh session key. If n is secret, R' normally has the following guarantee: any messages symmetrically encrypted with n that R' later receives are secret and are sent by R . If the adversary, however, knows $\text{sk}(R')$, it can learn n and use it to encrypt any message for R' .

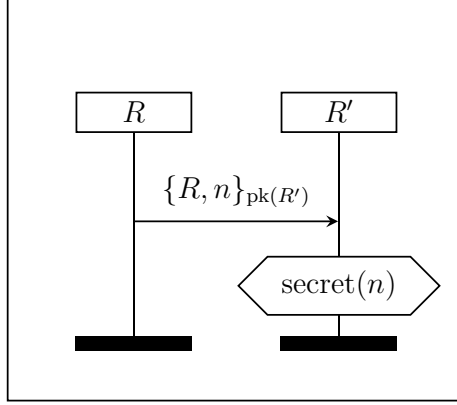


Figure 2.1: Example protocol 1.

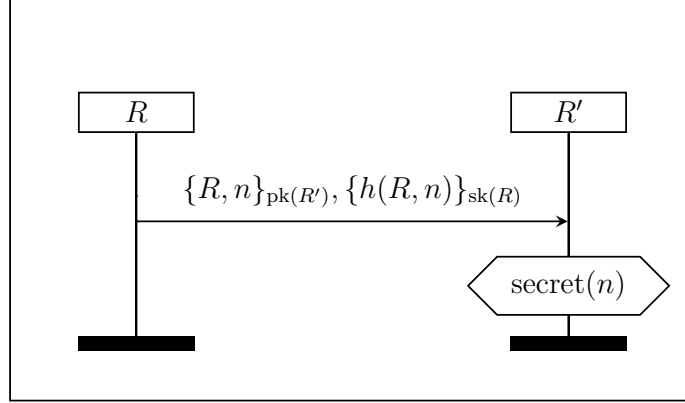


Figure 2.2: Example protocol 2.

Example 7. One possible formal specification of the protocol in Figure 2.1 is as follows: let $R, R' \in \text{Role}$, $\{1, 2\} \subseteq \text{Label}$, $n \in \text{Fresh}$, $x_n \in \text{Var}$, and define

$$\Pi(x) = \begin{cases} \langle \text{send}_1(R, R', \{R, n\}_{\text{pk}(R')}) \rangle, & \text{if } x = R, \\ \langle \text{recv}_1(R, R', \{R, x_n\}_{\text{pk}(R')}) \rangle, \\ \langle \text{claim}_2(R', \text{secret}, x_n) \rangle, & \text{if } x = R'. \end{cases}$$

Moreover, let type assign RunTerm_0 to every variable.

We can extend any type function to Term by assigning to each term the set of all its possible instantiations: for all $R \in \text{Role}$, $n \in \text{Fresh}$, and $y \in \text{Agent} \cup \text{Fresh}^{\#\text{RID}}$, we define $\text{type}(R) = \text{Agent}$, $\text{type}(n) = \{n^{\#\text{rid}} : \text{rid} \in \text{RID}\}$, and $\text{type}(y) = \{y\}$. We homomorphically extend type to Term .

2.2.2 Scyther execution model and security properties

We model protocol execution as a transition system. The set of all states of our system is $\text{State} = \text{Trace} \times \mathcal{P}(\text{RunTerm}) \times (\text{RID} \rightarrow \text{RunEvent}^*) \times (\text{RID} \rightarrow (\text{Role} \cup \text{Var}) \rightarrow$

RunTerm), where $\text{Trace} = (\text{RID} \times \text{TraceEvent})^*$ represents all possible execution histories or *traces*. Every execution state $s = (tr_s, AK_s, th_s, \sigma_s)$ consists of (1) a trace tr_s , (2) the adversary knowledge AK_s , (3) a partial function th_s mapping the run identifiers of initiated runs to sequences of run events, and (4) the role and variable instantiations σ_s of initiated runs. To keep the notation compact, we write $\sigma_s(rid)$ as $\sigma_{s,rid}$.

To define initial states, for each $rid \in \text{RID}$, we define a replacement $(\cdot)^{\#rid}$ to distinguish between local, freshly generated terms of each run by assigning unique names to the terms. The replacement $(\cdot)^{\#rid}$ takes $n \in \text{Fresh}$ and assigns $n^{\#rid} \in \text{Fresh}^{\text{RID}}$ to it: $(\cdot)^{\#rid} = \bigcup_{n \in \text{Fresh}} \{n^{\#rid} / n\}$. We will homomorphically apply this function to terms, events, and sequences of terms and events. For every $S \subseteq \text{Term}$ and type function type, we define the set $GS(S, \text{type})$ of all *ground substitutions on S respecting type* as the set of all substitutions $\tau : S \rightarrow \text{RunTerm}_0$ such that, for all $x \in S$, $\tau(x) \in \text{type}(x)$.

Definition 8 (Initial states). *Let (Π, type) be a protocol. For all $R \in \text{dom}(\Pi)$, the set of initial states $IS(\Pi, \text{type}, R)$ is defined as*

$$\bigcup_{\tau \in GS(\text{Role} \cup \text{Var}, \text{type})} \{(\langle \rangle, AK_0, \text{Test} \mapsto \tau(\Pi(R)^{\#Test}), \text{Test} \mapsto \tau)\},$$

where $AK_0 = \{a, \text{pk}(a) : a \in \text{Agent}\} \cup \{n^{\#rid_A} : n \in \text{Fresh}\}$ is the initial adversary knowledge.

Example 9. We consider the following initial state for the protocol in Example 7, where no events have been executed yet, but the test run has been completely predetermined:

$$\begin{aligned} &(\langle \rangle, AK_0, \text{Test} \mapsto \langle \text{recv}_1(\text{Alice}, \text{Bob}, \{\text{Alice}, n^{\#rid}\}_{\text{pk}(\text{Bob})}), \\ &\text{claim}_2(\text{Bob}, \text{secret}, n^{\#rid}) \rangle, \\ &\text{Test} \mapsto [\text{Alice}, \text{Bob}, n^{\#rid}, \dots / R, R', x_n, \dots]) \end{aligned}$$

The operational semantics of a protocol (Π, type) is defined by a transition system that combines the execution-model rules from Figure 2.3(a) with a set of adversary-compromise rules or *capabilities* [6], chosen from those in Figure 2.3(b). We identify each adversary with the set of its capabilities. This allows us to say what an adversary *can* do, but also what it *cannot* do. We normally omit the subscripted parameters from the rule names.

$$\begin{array}{c}
\frac{R \in \text{dom}(\Pi) \quad \text{rid} \notin (\text{dom}(th) \cup \{\text{rid}_A, \text{Test}\}) \quad \tau : \text{Role} \rightarrow \text{Agent}}{(tr, AK, th, \sigma) \longrightarrow (tr. \langle (rid, \text{create}(R)) \rangle, AK, th[\text{rid} \mapsto \tau(\Pi(R)^{\#rid})], \sigma[\text{rid} \mapsto \tau])} [\text{create}_\Pi] \\
\\
\frac{th(\text{rid}) = \langle \text{send}_l(a, b, m) \rangle. seq}{(tr, AK, th, \sigma) \longrightarrow (tr. \langle (rid, \text{send}_l(a, b, m)) \rangle, AK \cup \{m\}, th[\text{rid} \mapsto seq], \sigma)} [\text{send}] \\
\\
\frac{th(\text{rid}) = \langle \text{recv}_l(a, b, pt) \rangle. seq \quad \tau \in GS(\text{vars}(pt), \text{type}) \quad AK \vdash \tau(pt)}{(tr, AK, th, \sigma) \longrightarrow (tr. \langle (rid, \text{recv}_l(a, b, \tau(pt))) \rangle, AK, th[\text{rid} \mapsto \tau(seq)], \sigma[\text{rid} \mapsto \sigma_{rid} \cup \tau])} [\text{recv}_{\text{type}}] \\
\\
\frac{th(\text{rid}) = \langle e \rangle. seq \quad \text{evtype}(e) = \text{claim}}{(tr, AK, th, \sigma) \longrightarrow (tr. \langle (rid, e) \rangle, AK, th[\text{rid} \mapsto seq], \sigma)} [\text{claim}] \\
\\
\text{(a) Execution-model rules} \\
\\
\frac{a = \sigma_{\text{Test}}(R) \quad a \notin \{\sigma_{\text{Test}}(R') : R' \in \text{dom}(\Pi) \setminus \{R\}\}}{(tr, AK, th, \sigma) \longrightarrow (tr. \langle (rid_A, \text{LKR}(a)) \rangle, AK \cup \text{LTK}(a), th, \sigma)} [\text{LKR}_{\text{actor}\Pi, R}] \\
\\
\frac{a \notin \{\sigma_{\text{Test}}(R) : R \in \text{dom}(\Pi)\}}{(tr, AK, th, \sigma) \longrightarrow (tr. \langle (rid_A, \text{LKR}(a)) \rangle, AK \cup \text{LTK}(a), th, \sigma)} [\text{LKR}_{\text{others}\Pi}] \\
\\
\text{(b) Adversary-compromise rules}
\end{array}$$

Figure 2.3: Operational semantics for protocol (Π, type)

The **create** rule starts a new run of a protocol role R . The rule is parameterised by the function Π . A fresh run identifier rid is assigned to the run, thereby distinguishing it from previously created runs, the adversary run, and the test run. The role names of $\Pi(R)$ are replaced with agent names by a substitution τ , which is saved in the state. The **send** rule sends a message m to the network, thereby adding it to the adversary knowledge. In contrast, the **recv** rule, which is parameterised by type, accepts messages from the network that match the pattern pt , where pt is a term that may contain variables. Each variable must be instantiated with an element of its type. The resulting substitution τ is applied to the remaining steps of rid and saved in the state. The **claim** rule is used simply to log the statements that runs make about the security properties they expect to hold. We explain the connection between claims and security properties in detail later in this section.

In both the adversary-compromise rules we define, the adversary run issues a long-term key reveal query on some agent a , which is marked in the trace with $(\text{rid}_A, \text{LKR}(a))$. The $\text{LKR}_{\text{actor}}$ rule allows the adversary to learn the long-term keys

of the agent executing the test run (also called *the actor*). The rule takes Π and R as parameters. The second premise of the rule is needed because we allow agents to communicate with themselves. Since $\text{LKR}_{\text{actor}}$ is the core component of Actor Key Compromise, we discuss it in detail in Chapter 3. The $\text{LKR}_{\text{others}}$ rule, which is parameterised by Π , formalises the standard Dolev-Yao adversary's capability to reveal the keys of any agent a that is not an intended partner (or *peer*) of the test run.

Note that all adversaries defined in this section are limited by the above two adversary-compromise rules. In other words, we are considering four different adversary models: $\emptyset$ or the empty adversary (who can still perform eavesdropping on and active tampering with network messages), $\{\text{LKR}_{\text{actor}}\}$, $\{\text{LKR}_{\text{others}}\}$, and $\{\text{LKR}_{\text{actor}}, \text{LKR}_{\text{others}}\}$.

Definition 10 (Transition relation and reachable states). *Let (Π, type) be a protocol, $R \in \text{dom}(\Pi)$ a role, and A an adversary. We define a transition relation $\rightarrow_{\Pi, \text{type}, R, A}$ from the execution-model rules in Figure 2.3(a) and the rules in A . For states s and s' , $s \rightarrow_{\Pi, \text{type}, R, A} s'$ iff there exists a rule in either the execution-model rules or A with the conclusion $s \rightarrow s'$ such that all of the premises hold. We define the set of reachable states $\text{RS}(\Pi, \text{type}, R, A)$ by $\{s \in \text{State} : (\exists s_0 \in \text{IS}(\Pi, \text{type}, R))(s_0 \rightarrow_{\Pi, \text{type}, R, A}^* s)\}$.*

Example 11. The state $(\langle (rid, \text{create}(R)) \rangle, AK_0, th, \sigma)$ where

$$th(x) = \begin{cases} \langle \text{send}_1(\text{Alice}, \text{Bob}, \{\text{Alice}, n^{\#rid}\}_{\text{pk}(\text{Bob})}) \rangle, & \text{if } x = rid, \\ \langle \text{recv}_1(\text{Alice}, \text{Bob}, \{\text{Alice}, n^{\#rid}\}_{\text{pk}(\text{Bob})}) \rangle, \\ \langle \text{claim}_2(\text{Bob}, \text{secret}, n^{\#rid}) \rangle, & \text{if } x = \text{Test}, \end{cases}$$

and

$$\sigma_x = \begin{cases} [\text{Alice}, \text{Bob}, \dots / R, R', \dots], & \text{if } x = rid, \\ [\text{Alice}, \text{Bob}, n^{\#rid}, \dots / R, R', x_n, \dots], & \text{if } x = \text{Test} \end{cases}$$

is reached from the initial state in Example 9 by a single application of the **create** rule, regardless of the adversary A . In this state, Alice is running the newly created run rid of the role R with whom she believes to be Bob in role R' . Alice has not yet executed any protocol steps in run rid , i.e. she has not sent her encrypted message.

We model security properties as reachability properties. To keep our definitions independent of the protocol, we use the claim events to declare that a protocol is meant to satisfy a certain property. We will define three security properties for role R : secrecy, aliveness, and non-injective data agreement [63]. Let us first introduce an auxiliary function: for all $rid \in \text{RID}$, $R' \in \text{Role}$, and reachable states s (for any instantiation of Π , type , R and A) such that $(rid, \text{create}(R')) \in tr_s$, we define $\text{role}_s(rid) = R'$, and we let $\text{role}_s(\text{Test}) = R$.

Definition 12 (Security claims and $\models$). Given a label $l \in \text{Label}$, roles $R, R' \in \text{Role}$, and $t \in \text{RoleTerm}$, we call every

$$\gamma \in \{\text{claim}_l(R, \text{secret}, t), \\ \text{claim}_l(R, \text{alive}, R'), \\ \text{claim}_l(R, \text{commit}, R', t)\}$$

a security claim for R . For all $s \in \text{State}$, let a, b, t_0 be respective instances of $R, R', t^{\#Test}$ assigned by $\sigma_{s, Test}$. With $s \models \gamma$, we denote that the following implication is true: if $(Test, \sigma_{s, Test}(\gamma^{\#Test})) \in tr_s$, then

- $AK_s \not\models \sigma_{s, Test}(t^{\#Test})$ for $\gamma = \text{claim}_l(R, \text{secret}, t)$
(secrecy of t_0 for a)
- $(\exists rid \in \text{RID})(\sigma_{s, rid}(\text{role}_s(rid)) = \sigma_{s, Test}(R'))$ for
 $\gamma = \text{claim}_l(R, \text{alive}, R')$ (aliveness of b for a)
- $(\exists rid \in \text{RID})(\text{role}_s(rid) = R' \wedge$
 $(rid, \sigma_{s, Test}(\text{claim}_l(R', \text{running}, R, t^{\#Test}))) \in tr_s)$
for $\gamma = \text{claim}_l(R, \text{commit}, R', t)$
(non-injective agreement for a with b on t_0)

Let (Π, type) be a protocol, $R \in \text{dom}(\Pi)$, A an adversary, and $\gamma \in \Pi(R)$ a security claim. By $(\Pi, \text{type}) \models_A \gamma$ we denote that for all $s \in \text{RS}(\Pi, \text{type}, R, A)$, $s \models \gamma$.

Note that we do not consider a running claim to be a security claim and simply use it to define non-injective agreement. Namely, if in a particular state a commit claim executed by $Test$ matches a running claim previously executed by rid , and rid was run in the correct role, then $Test$ and rid agree on the claims' contents.

Example 13. Let P be the protocol in Example 7 and let A be an adversary. We have $P \not\models_A \text{claim}_2(R', \text{secret}, x_n)$. If we change P to P' by adding $\text{claim}_3(R, \text{secret}, n)$ in any position to the R role, we have $P' \models_A \text{claim}_3(R, \text{secret}, n)$. We will prove a generalisation of this fact in Section 3.3.

For the statement $P \not\models_A \text{claim}_3(R', \text{secret}, x_n)$ in the above example, we say that x_n is not secret for R' in P with respect to A . We paraphrase the truth value of other security claims in a similar way.

2.3 Tamarin framework

In this section, we give a very brief overview the Tamarin framework that suffices to understand the results explained in Chapter 3 and Chapter 5 of this thesis. We refer the interested reader to [69, 70, 94, 95] for the details.

2.3.1 Brief overview of the Tamarin framework

Within the Tamarin framework, the local states of protocol participants, the generated fresh values, the knowledge of the adversary, and the messages on the network are all parts of the global state, which is formalised as a finite multiset of terms called *facts*.

Facts can be *produced* and *consumed*. Depending on the number of times that each fact can be consumed, it can be either *linear* or *persistent*. While linear facts model limited resources which cannot be consumed more times than they are produced, persistent facts model unlimited resources that, when once produced, can be consumed any number of times. Persistent facts are designated with a prefixed exclamation point to be clearly distinguishable from linear facts. For example, the built-in fact symbol **Fr** can be used to model a freshly generated x by the linear fact $\text{Fr}(x)$. Another built-in fact symbol is **!KU**. The persistent fact $\text{!KU}(x)$ models that x is known, and will remain known, to the adversary.

The semantics of the above facts as well as additional user-defined facts is specified by *multiset-rewriting rules*. Rules in Tamarin have a *left-hand side* (facts to be consumed by the rule execution), *actions* (observable events to be logged), and a *right-hand side* (facts to be produced). A rule can only be executed if its left-hand side contains facts that are available for consumption in the global state.

We introduce the way Tamarin rules are written by considering a simple rule:

```
rule Reveal_Ltk:
[ !Ltk(A, ltkA) ]--[ RevLtk(A) ]->[ Out(ltkA) ]
```

The **Reveal_Ltk** rule can be executed if an agent A registered a long-term public key whose corresponding secret key is ltkA . The execution of the rule models the adversarial compromise of A ; the key ltkA is sent out to the network and the compromise is logged in the observable trace. Another important rule that we will refer to in our TLS 1.3 analysis is the built-in **Fresh** rule:

```
rule Fresh:
[ ]--[ ]->[ Fr(x) ]
```

This rule can always be executed because it consumes no facts, and its execution produces a single linear $\mathbf{Fr}(\mathbf{x})$ fact without it being logged in the observable trace. Note that the **Fresh** rule has a special internal semantics: it is the only rule that can produce $\mathbf{Fr}$ facts, and no two produced $\mathbf{Fr}$ facts can ever match.

While a protocol and a threat model can be encoded as rules, the required security properties are specified as formulæ of a fragment of first-order logic with timepoints. Rule actions name timepoint-ordered observable events, and the Tamarin property specification language makes it possible to quantify over their parameters and timepoints.

Our first step in performing protocol verification using the Tamarin prover is to first abstractly model the protocol, attacker and security properties, and then encode them as direct inputs to the Tamarin prover. Typically, abstract state machines are first inferred from a protocol specification, and then refined to more precisely reflect the relevant updates to the global state. Encoding the refined model as an input to Tamarin includes a proper binding of variables in the rule inputs and outputs, and adding observable events as actions to express the required security properties. Then the model is iteratively corrected and refined as needed during the verification process, which is explained in detail on a practically relevant protocol in Chapter 5.

Chapter 3

Actor Key Compromise

Despite Alice’s best efforts, her long-term secret keys may be revealed to an adversary. Possible reasons include weakly generated keys, compromised key storage, subpoena, and coercion. However, Alice may still be able to communicate securely with other parties, depending on the protocol used. We call the associated property resilience against *Actor Key Compromise* (AKC). We formalise this property in a symbolic model and identify conditions under which it can and cannot be achieved. In case studies that include TLS and SSH, we find that many protocols are not resilient against AKC. We implement a concrete AKC attack on the mutually authenticated TLS protocol.

3.1 Introduction

Most of us use devices which run security protocols on our behalf on a daily basis. Examples include, but are not limited to, secure searches, e-shopping, remote login, and physical access. Particularly dangerous scenarios emerge when protocols which use no cryptographic mechanisms at all are used in lieu of security protocols, such as when the Dallas Semiconductor Corp. 1-Wire protocol [65] is used for physical access control over an iButton fob [66]. In this setup, the fob reader asks the fob for its identification number, which is then sent unencrypted by the fob. The identification number can then be recorded and replayed, which is sufficient to gain physical access [21].

When actual security protocols are used for physical access control, they usually provide unilateral authentication of a protocol participant by *securely* checking if it knows a particular secret value. A bare-bones example of such a unilaterally authenticated physical access protocol is the simple challenge-response protocol depicted in Figure 3.1. We write $\{n\}_K$ to denote the encryption of the fresh value n with the key

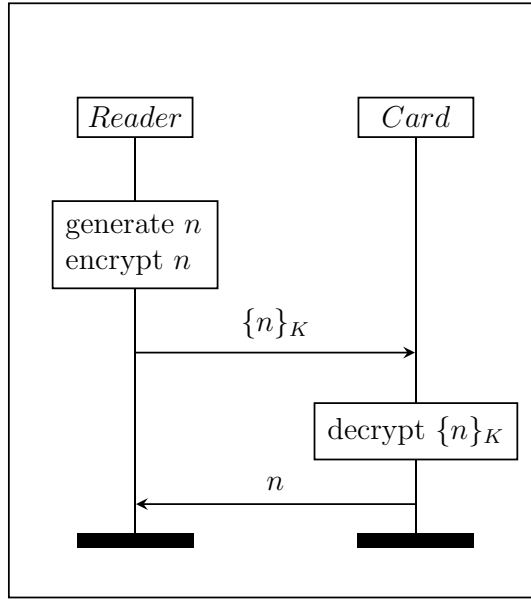


Figure 3.1: Challenge-response protocol example

K shared by the card and the reader. Here the reader knows¹ that the card is present if K is secret.

Unfortunately, long-term secrets may be compromised in this type of scenario [37], as well as more sinister ones. If a government agency obtains the long-term secret key of a service provider [77,81], it is clear that the agency can impersonate the service provider to its users. But can the agency also impersonate an arbitrary user to the service provider? Whether this is possible depends on the security protocol in question. In this chapter, we study the property that formalises this behaviour, which we call resilience against *Actor Key Compromise* (AKC).

To illustrate AKC, consider a setting with a public-key infrastructure: each party X has a long-term key pair for asymmetric encryption or signing, where $\text{pk}(X)$ denotes the public key and $\text{sk}(X)$ denotes the corresponding secret key. We write $h(m)$ for the hash of m . In this setting, Alice can use certain protocols to establish unilateral security guarantees. For example, Alice is guaranteed the secrecy of the nonce na and agreement on its value when sending it encrypted to Bob and receiving a hash of it as depicted in Figure 3.2. Here, as in many unilateral protocols, the security of the protocol relies only on the secrecy of Bob’s long-term secret key.

Most modern protocols offer bilateral guarantees, established through mutual authentication protocols or authenticated key exchange protocols. As a standard

¹This is a simplification for introductory purposes. Other things need to be assumed, such as bounded adversarial power, the correct implementation of pseudorandom number generators to avoid bad randomness [35], etc.

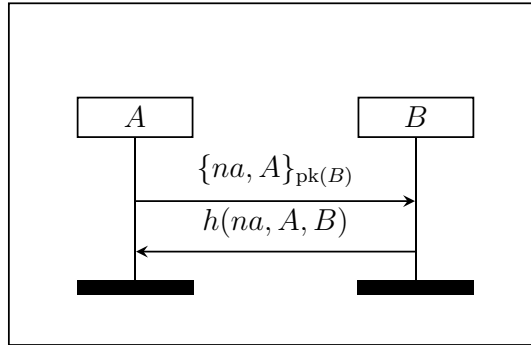


Figure 3.2: Secrecy and agreement example

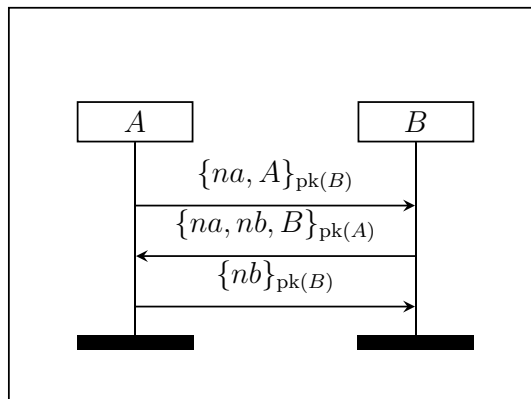


Figure 3.3: Needham-Schroeder-Lowe protocol

example, consider the Needham-Schroeder-Lowe (NSL) protocol [62] visualised in Figure 3.3.

Such bilateral protocols can be viewed as combining two unilateral protocols: if Alice’s long-term secret key is compromised, Bob’s half of the bilateral guarantees is lost because the adversary can impersonate Alice. But what about Alice’s half? Since Bob’s key is not compromised, Alice might expect to obtain the guarantees she would have when using an appropriate unilateral protocol.

It turns out that not every bilateral protocol has this property. For example, if Alice’s secret key is compromised in the Needham-Schroeder-Lowe protocol, she then no longer obtains secrecy of the nonce nb : the adversary can simply decrypt the second message, even without interfering with the order or content of messages passed between the participants. The adversary can also violate the participants’ agreement on nb , as in Figure 3.4.

We say that such protocols are vulnerable to AKC attacks: if the long-term secret key of a party (the actor) is compromised, the party can no longer obtain unilateral guarantees when communicating with another party (the peer) even when the peer’s

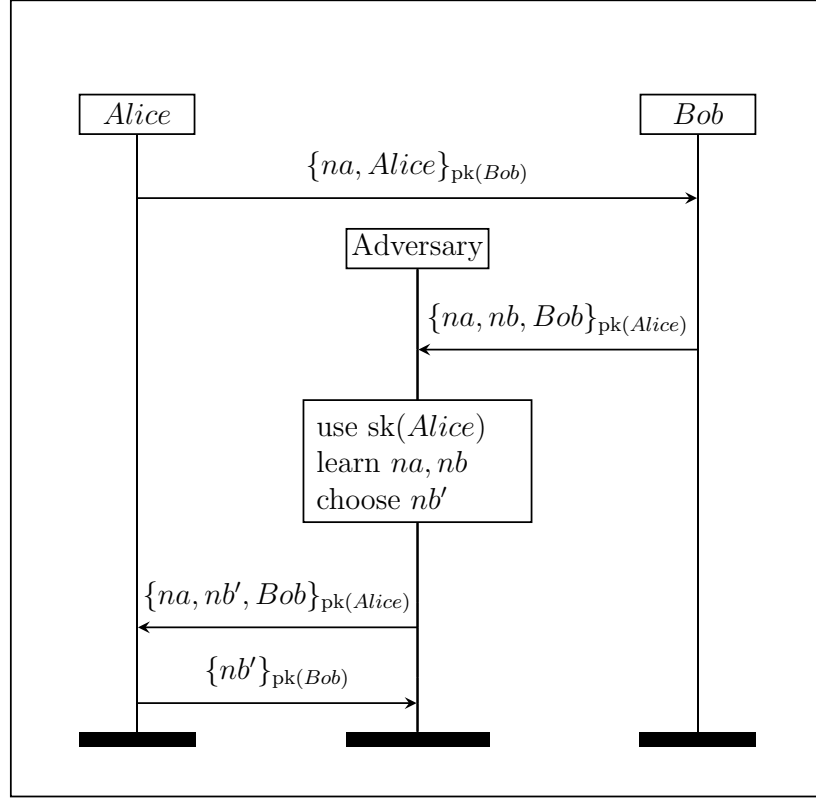


Figure 3.4: Attack on Needham-Schroeder-Lowe protocol

key is still secret. From the actor's local perspective, protocols that are vulnerable to AKC attacks offer weaker security guarantees than many unilateral protocols, because the vulnerable protocols only achieve the unilateral guarantees if *both* the long-term keys of the actor and the peer are secret.

This phenomenon has been largely ignored by the security protocol community. A notable exception is in the research literature on authenticated key exchange protocols, where a limited instance of this problem has been studied. Namely, there are so-called *Key Compromise Impersonation* (KCI) attacks [13, 51], where the actor's key is revealed and used by the adversary to impersonate another party communicating with the actor. Of course, one could also consider such an adversary when ensuring the non-repudiation of online payments, or for secrecy of votes in an e-voting protocol. We conclude that the core issue is limited neither to key exchange protocols nor to authentication. The loss of a party's long-term secret key may impact any security property of that party in any type of security protocol.

Overview. We use the Scyther framework to formalise actor key compromise in Section 3.2. We show how to achieve actor key compromise security by protocol

transformation in Section 3.3 and prove impossibility results in Section 3.4. We present case studies in Section 3.5 and draw conclusions in Section 3.6.

3.2 Formalising actor key compromise

In this section, we formalise the notions related to actor key compromise using the Scyther framework described in Chapter 2. This enables us to reason about the security guarantees of agents whose long-term secret keys have been compromised. We define these notions independently of the choice of protocol, adversary, and property.

We first give an informal definition of an AKC attack.

Definition 14 (Actor key compromise attack, informal). *We say that an attack on a security property of an agent A is an actor key compromise attack if the attack requires that the adversary obtains and uses a long-term private key of A .*

We now formally define which protocols we assume to be completely unaffected by AKC attacks and discuss whether our earlier example protocols fit the definition.

Definition 15 (Actor key compromise security). *Let (Π, type) be a protocol, $R \in \text{dom}(\Pi)$, A an adversary such that $\text{LKR}_{\text{actor}} \in A$, and $\gamma \in \Pi(R)$ a security claim. We say that γ is actor key compromise secure in (Π, type) with respect to A if $(\Pi, \text{type}) \models_A \gamma$.*

For example, the claimed secrecy of the nonce n in the protocols in Figures 2.1 and 2.2 is not AKC secure. Consider the trace of a regular execution of either protocol, where Alice performs R and Bob performs R' . We can extend this trace using the $\text{LKR}_{\text{actor}}$ rule on Bob, which is allowed since Bob is not an element of the set $\{\sigma_{\text{Test}}(R)\} = \{\text{Alice}\}$. The adversary can then use $\text{sk}(\text{Bob})$ to decrypt the nonce, thus violating the secrecy claim.

We would like to formally define the notion of AKC attack by extending the informal notion of KCI attack, which suggests that the actor's long-term keys are not only available to the adversary, but are integral in performing the attack. In the KCI literature, whenever a protocol execution is described as a KCI attack, it contains both the compromise of the actor's key and its illicit usage by the adversary. This is why attacks on AKC security of secrecy overapproximate KCI attacks.

Our first try at stating an AKC attack definition might be to say that an attack is AKC if the attacker compromises the actor's keys; this idea is unsuccessful because,

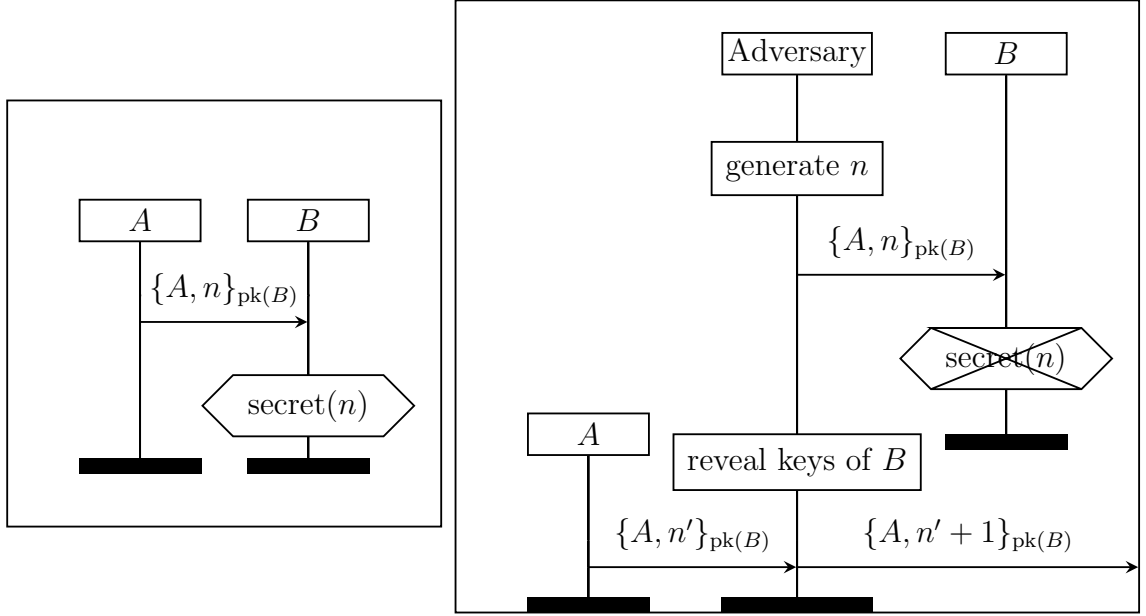


Figure 3.5: Protocol and attack showing difficulties in defining AKC attack notion

trivially, an instance of **LKR** as prescribed by the $\text{LKR}_{\text{actor}}$ rule can be added to most traces.

A natural second attempt involves requiring that the adversary uses the key after compromising it. However, the mentioned key usage can also unnecessarily be appended to attack traces. An example of this can be found in Figure 3.5. We see that the adversary performs an attack by encrypting for B a nonce that the adversary generates itself. It then allows A to send an encrypted nonce, which the adversary learns by compromising the keys of B and decrypting A 's message. The adversary then increments the nonce, reencrypts it, and sends it off, perhaps to another instance of the B role. Even if this attack cannot be performed without $\text{sk}(B)$, there is a prefix of the attack trace that is still an attack in which $\text{sk}(B)$ is not used at all.

Trying to fix the second attempt by saying that an AKC attack is one where the prefix up to the security claim that is being verified must require the use of the actor's key also does not work: in the previous example, the attacker can first talk to A , and then to B .

It seems that what we really want to build into the definition of AKC attack is that the compromised actor's key is not only used and/or required, but necessarily used in a clever way. Unfortunately, the concept of cleverness or lack thereof can be difficult to formalise [91]. This is why we settle on the following underapproximation: if an adversary cannot violate a property without $\text{LKR}_{\text{actor}}$, but can violate it when

using $\text{LKR}_{\text{actor}}$, only then do we call the attacks that arise AKC attacks. We term the absence of such attacks AKC resilience.

Definition 16 (AKC resilience and AKC attack). *Let (Π, type) be a protocol, $R \in \text{dom}(\Pi)$, A an adversary such that $\text{LKR}_{\text{actor}} \in A$, and $\gamma \in \Pi(R)$ a security claim. We say that γ is actor key compromise resilient in (Π, type) with respect to A if*

$$(\Pi, \text{type}) \models_{A \setminus \{\text{LKR}_{\text{actor}}\}} \gamma \implies (\Pi, \text{type}) \models_A \gamma.$$

Otherwise, each $s \in \text{RS}(\Pi, \text{type}, R, A)$ where $s \not\models \gamma$ is an actor key compromise attack by A on γ in (Π, type) .

Note that a claim γ is trivially AKC resilient if there is an attack on γ where the $\text{LKR}_{\text{actor}}$ capability remains unused. We say that a protocol is *AKC resilient* or *AKC secure* with respect to an adversary if the same is true for all of the security claims in the protocol. If there is an adversary who can perform an AKC attack on at least one of the claims, we say the protocol is *vulnerable to AKC attacks*.

We now revisit our examples. The KCI attack on the protocol in Figure 2.1, which is described just before Example 7, represents an attack on secrecy by $\{\text{LKR}_{\text{actor}}\}$. It does not however represent an AKC attack on secrecy by $\{\text{LKR}_{\text{actor}}\}$ because there already exist attacks on secrecy by the empty adversary, i.e. one with no capabilities beyond the execution-model rules. The adversary can generate and encrypt a nonce itself. Hence the protocol is trivially AKC resilient with respect to $\{\text{LKR}_{\text{actor}}\}$. We see that, in general, AKC resilience does not imply AKC security, which is the *absence* of attacks on a property by an adversary who has the $\text{LKR}_{\text{actor}}$ capability.

The protocol from Figure 2.2, however, is not AKC resilient with respect to $\{\text{LKR}_{\text{actor}}\}$. The empty adversary cannot generate the nonce itself because it cannot forge the signature. In fact, the nonce is secret with respect to the empty adversary. Therefore, decrypting a sent, encrypted nonce as in the KCI attack described just before Example 7 gives rise to an AKC attack by $\{\text{LKR}_{\text{actor}}\}$ on the secrecy of the nonce.

The reader might consider this to be an undesirable effect of the AKC attack definition. After all, the seemingly stronger protocol suddenly admits an AKC attack where the weaker one does not. Another issue one might raise with the definition is that one might expect it to be monotonic in the following sense: if an adversary cannot perform AKC attacks, then a weaker one also cannot perform AKC attacks. We show that this is not the case, i.e. that AKCR as defined is not a trace property.

Proposition 17. *There is a protocol $P = (\Pi, \text{type})$, role $R \in \text{dom}(\Pi)$, adversaries A and B where $\text{LKR}_{\text{actor}} \in A$ and $A \subseteq B$, and a security claim $\gamma \in \Pi(R)$ such that AKCR of γ with respect to B does not imply AKCR of γ with respect to A .*

Proof. We need to find P , R , A , B , and γ such that the implication

$$\text{if } P \models_{B \setminus \{\text{LKR}_{\text{actor}}\}} \gamma, \text{ then } P \models_B \gamma$$

is true and

$$\text{if } P \models_{A \setminus \{\text{LKR}_{\text{actor}}\}} \gamma, \text{ then } P \models_A \gamma$$

is false. Let $A = \{\text{LKR}_{\text{actor}}\}$ and $B = \{\text{LKR}_{\text{actor}}, \text{LKR}_{\text{others}}\}$. We are done if we can find P and γ such that $P \not\models_{\{\text{LKR}_{\text{others}}\}} \gamma$, $P \models_{\emptyset} \gamma$, and $P \not\models_{\{\text{LKR}_{\text{actor}}\}} \gamma$. A well-known example of such a protocol is the Needham-Schroeder Public Key protocol [74], where the type function assigns RunTerm_0 to every variable and γ is the claim of secrecy of one of the exchanged nonces from the point of view of the responder. While an adversary can learn neither nonce without revealing any long-term secret keys, it can use either $\text{LKR}_{\text{actor}}$ to get the keys to decrypt them, or use $\text{LKR}_{\text{others}}$ and perform the well-known man-in-the-middle attack [62]. $\square$

When AKC resilience is non-trivially satisfied, it coincides with AKC security (AKCS). This, along with the above discussion, is the reason we will primarily focus on AKCS: after all, for a given purpose and any pair of adversary and security property, we want to use a protocol where no attacks are possible, regardless if they are AKC-related. Another reason is that we can represent KCI resilience (KCIR) as an instance of AKCS.

KCIR as an instance of AKCS

KCIR key establishment protocols provide an important security guarantee to their users: even if the users' own long-term keys are compromised, they can still authenticate messages encrypted with established session keys. Although the notion of a KCI *attack* has remained informal and subjective, KCI *resilience* has been incorporated into different formal models based on a recurring idea: if an adversary capable of getting the actor's keys cannot perform *any* attack on session key secrecy in a key establishment protocol, then it cannot perform a KCI attack.

In the following proposition, we prove that session key secrecy during a key establishment handshake coincides with key agreement after a particular key confirmation step, even under AKC. The step requires a peer to apply any hash function unused in

the handshake to the key and the relevant identities, and send the hash value to the actor as depicted in Figure 3.6. This shows how the standard notion of KCIR can be recast in our terms as both AKCS of secrecy and AKCS of authentication.

We will need two lemmas in the proof, and an auxiliary lemma for each. One of the auxiliary lemmas states that accessible subterms of terms inferable from a set are either inferable themselves or accessible in the set.

Lemma 18 (Inference of accessible subterms). *Let $S \cup \{t\} \subseteq \text{Term}$ and $n \in \mathbb{N}_0$ such that $S \vdash_n t$. Then for all $t' \in \text{Term}$ such that $t' \sqsubseteq_{\text{acc}} t$, $S \vdash_n t'$ or $t' \sqsubseteq_{\text{acc}} S$.*

Proof. We prove the lemma by induction on n . Suppose $n = 0$. If $t \in \text{Const}$, the statement holds. Otherwise, we have $t \in S$, so $t' \sqsubseteq_{\text{acc}} t$ implies $t' \sqsubseteq_{\text{acc}} S$. Assuming that the statement holds for all natural numbers less than some n , we prove the statement for n . All decomposition cases immediately follow from the inductive hypothesis. The non-trivial composition cases are similar, so we only provide a proof for one. If the last step of a derivation of t from S of height n infers $S \vdash (t_1, t_2)$ from $S \vdash t_1$ and $S \vdash t_2$, then either $t' = t$ (and thus $S \vdash_n t'$), or it is the case that $t' \sqsubseteq_{\text{acc}} t_1$ or $t' \sqsubseteq_{\text{acc}} t_2$. Without loss of generality, assume the former. Then the inductive hypothesis gives us $S \vdash_{n-1} t'$ or $t' \sqsubseteq_{\text{acc}} S$. $\square$

The first of the lemmas actually invoked in the proof of the proposition helps us understand the options available to an adversary for inferring a given term: the term has been sent as an accessible subterm and the adversary managed to retrieve it, or the adversary composed it itself.

Lemma 19 (Composition lemma). *Let $S \cup \{t\} \subseteq \text{Term}$ and $S \vdash t$. Then $t \sqsubseteq_{\text{acc}} S$, or every minimal-height derivation of t from S ends in a composition rule.*

Proof. Assume $S \vdash t$. Then there is a derivation of t from S , so there is one of minimal height n . Suppose $n = 0$. The first case is $t \in \text{Const}$, which makes the single inference rule a composition rule. Otherwise, $t \in S$ implies $t \sqsubseteq_{\text{acc}} S$. Assume now that $n > 0$ and that some derivation of t from S of height n ends in a decomposition rule. The premises of that rule then imply that there is a term t' such that $S \vdash_{n-1} t'$ and $t \sqsubseteq_{\text{acc}} t'$. However, $S \not\vdash_{n-1} t$ by the minimality of n , so from Lemma 18 we conclude $t \sqsubseteq_{\text{acc}} S$. $\square$

Another auxiliary lemma states that atomic terms, symmetric long-term keys, and hashes are only useful in an inference when they occur as subterms in the inference.

Lemma 20 (Inference-irrelevant terms). *Let $S \cup \{t\} \subseteq \text{Term}$ where $t \not\sqsubseteq S \setminus \{t\}$, $t^{-1} = t$ and t has no proper accessible subterms. Let $t' \in \text{Term}$ such that $t \not\sqsubseteq t'$. Then $S \vdash t'$ if and only if $S \setminus \{t\} \vdash t'$.*

Proof. $\boxed{\Leftarrow}$ Trivial.

$\boxed{\Rightarrow}$ Let $t' \in \text{Term}$ such that $t \not\sqsubseteq t'$. We prove by induction on n that for all $n \in \mathbb{N}_0$, if $S \vdash_n t'$, then $S \setminus \{t\} \vdash_n t'$. First we consider the case of $n = 0$. For $t' \in \text{Const}$, the statement holds. Otherwise, we have $t' \in S$. From $t \not\sqsubseteq t'$ we get $t \neq t'$, so $t' \in S \setminus \{t\}$, i.e. $S \setminus \{t\} \vdash_0 t'$. Now we assume that up to some $n \in \mathbb{N}_0$, the statement holds and that a $\vdash$ -derivation tree for t' from S of height at most $n + 1$ is given. If it is at most n , we are done. Otherwise, we proceed with a case split on the last rule applied in that tree. If the last rule infers t' from t_1, t_2 by composition, assuming $t \sqsubseteq t_i$ for some $i \in \{1, 2\}$, we would have $t \sqsubseteq t'$, which is a contradiction. Therefore, we can apply the inductive hypothesis and infer each of t_1, t_2 from $S \setminus \{t\}$ with trees of height at most n each.

In the remaining cases, the last rule to derive t' is a decomposition rule:

- $S \vdash_n (t', t'')$ or $S \vdash_n (t'', t')$: Without loss of generality, we assume the former and do a case split on the accessibility of (t', t'') in S .
 - $(t', t'') \sqsubseteq_{\text{acc}} S$: The term t has no proper accessible subterms, so from $(t', t'') \sqsubseteq_{\text{acc}} S$ we get $(t', t'') \sqsubseteq_{\text{acc}} S \setminus \{t\}$. Suppose $t \sqsubseteq (t', t'')$. By transitivity of $\sqsubseteq$, we get $t \sqsubseteq S \setminus \{t\}$, which contradicts our assumptions. We can now apply the inductive hypothesis to infer $S \setminus \{t\} \vdash_n (t', t'')$. Hence $S \setminus \{t\} \vdash_{n+1} t'$.
 - $(t', t'') \not\sqsubseteq_{\text{acc}} S$: We can replace the derivation of (t', t'') with one of a minimal height, which is of height at most n . By Lemma 19, that derivation ends in a composition rule from t' and t'' . Therefore, $S \vdash_{n-1} t'$, so $t \not\sqsubseteq t'$ with the inductive hypothesis implies $S \setminus \{t\} \vdash_{n-1} t'$.
- $S \vdash_n \{t'\}_{t''}$ and $S \vdash_n t''^{-1}$:
 - $\{t'\}_{t''} \sqsubseteq_{\text{acc}} S$: As above, we conclude $t \not\sqsubseteq \{t'\}_{t''}$. The inductive hypothesis then gives us $S \setminus \{t\} \vdash_n \{t'\}_{t''}$. From $t \not\sqsubseteq \{t'\}_{t''}$ we also get $t \not\sqsubseteq t''$.

We have two cases: if $t''^{-1} = t''$, then $t \not\sqsubseteq t''^{-1}$ is immediate. Otherwise, if $t''^{-1} \neq t''$, from $t^{-1} = t$ we get $t \neq t''^{-1}$. Without loss of generality, let $u \in \text{Term}$ such that $t'' = \text{pk}(u)$ and $t''^{-1} = \text{sk}(u)$. From $t \not\sqsubseteq t''$ we get $t \not\sqsubseteq u$. Hence $t \neq t''^{-1}$ implies $t \not\sqsubseteq t''^{-1}$.

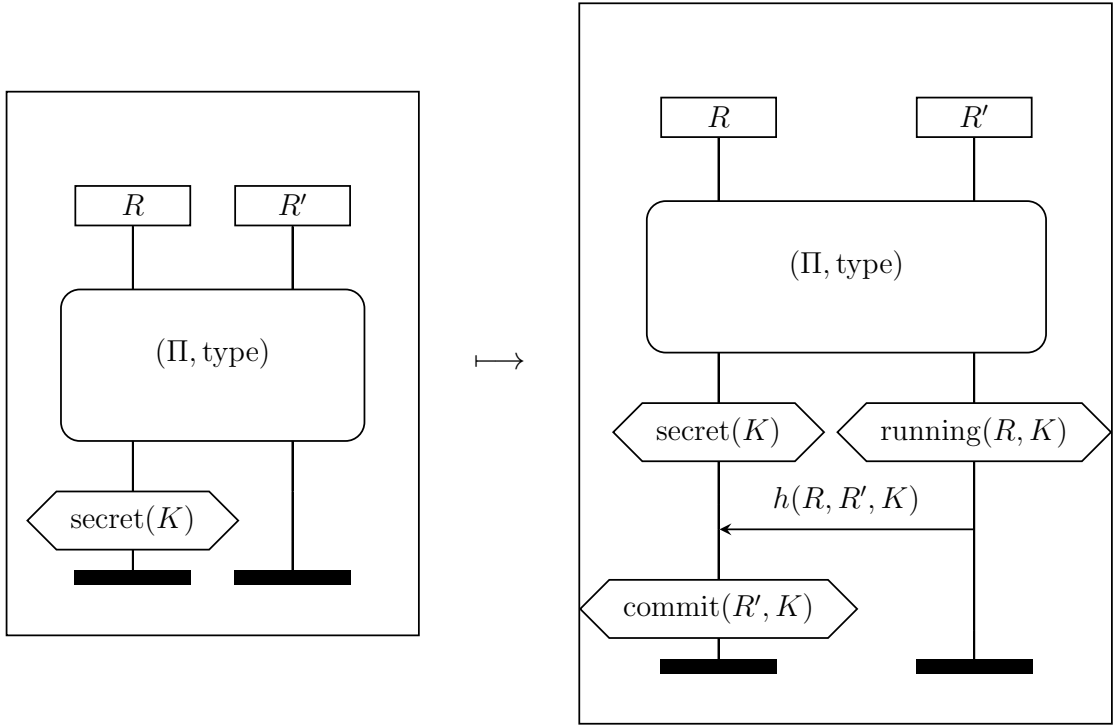


Figure 3.6: KCIR as AKCS of secrecy and non-injective agreement

In both cases we get $t \not\sqsubseteq t''^{-1}$, so we can apply the inductive hypothesis to infer $S \setminus \{t\} \vdash_n t''^{-1}$.

– $\{t'\}_{t''} \not\sqsubseteq_{\text{acc}} S$: As above. □

We can now prove Lemma 21, which is a generalisation of the previous lemma to finite sets. We can remove a finite number of terms by applying Lemma 20, provided that we do it in the right order.

Lemma 21 (Inference-irrelevant sets). *Let $S \cup T \subseteq \text{Term}$ where T is finite and for all $t \in T$, $t \not\sqsubseteq S \setminus T$, $t^{-1} = t$ and t has no proper accessible subterms. Let $t' \in \text{Term}$ such that, for all $t \in T$, $t \not\sqsubseteq t'$. Then $S \vdash t'$ if and only if $S \setminus T \vdash t'$.*

Proof. Assume $S \vdash t'$. Since T is finite and partially ordered by $\sqsubseteq$, it has a maximal element t . But then $t \not\sqsubseteq T \setminus \{t\}$, which with $t \not\sqsubseteq S \setminus T$ implies $t \not\sqsubseteq S \setminus \{t\}$. We can now apply Lemma 20 and get $S \setminus \{t\} \vdash t'$. The set T is finite, so by induction we get $S \setminus T \vdash t'$. □

We now define the transformation depicted in Figure 3.6, and prove that the adversary can violate agreement on the session key in the transformed protocol if and

only if it knows the key in the initial one. We will instantiate all the parameters just before using the transformation. We let:

$$KC(\Pi)(x) = \begin{cases} \Pi(R').\langle \text{claim}_{l_1}(R', \text{running}, R, K'), \\ \text{send}_{l_2}(R', R, h(R, R', K')) \rangle, & \text{if } x = R', \\ \Pi(R).\langle \text{recv}_{l_2}(R', R, h(R, R', K)), \\ \text{claim}_{l_1}(R, \text{commit}, R', K) \rangle, & \text{if } x = R, \\ \Pi(x), & \text{if } x \in \text{dom}(\Pi) \setminus \{R, R'\}. \end{cases}$$

The term K' corresponds to the view from R' on the key K ; usually, one of them is a variable and the other is a fresh value, but some protocols use compound terms. We want the transformation to work on any key establishment protocol, so we choose a symbol h that does not occur in the protocol specification. With the help of some typing assumptions, this prevents messages from the confirmation step from being accepted in the receive events of the handshake.

Proposition 22 (KCIR as an instance of AKCS). *Let (Π, type) be a protocol, $R, R' \in \text{dom}(\Pi)$ such that $R \neq R'$, and $\text{last}(\Pi(R)) = \text{claim}_{l_0}(R, \text{secret}, K)$ where $l_0 \in \text{Label}$ and $K, K' \in \text{RoleTerm}$. Let $l_1, l_2 \in \text{Label}$ and $h \in \text{Func}$ of arity 1 all be unused in Π , where $l_1 \neq l_2$ and h does not occur in the set $\text{type}(\text{Var} \cup \{K'\})$. Let A be an adversary. If $(KC(\Pi), \text{type})$ is a protocol, then*

$$\begin{aligned} (\Pi, \text{type}) \models_A \text{claim}_{l_0}(R, \text{secret}, K) &\iff \\ (KC(\Pi), \text{type}) \models_A \text{claim}_{l_1}(R, \text{commit}, R', K). \end{aligned}$$

Proof. $\Rightarrow$ We prove the contrapositive. Suppose that

$$(KC(\Pi), \text{type}) \not\models_A \text{claim}_{l_1}(R, \text{commit}, R', K).$$

Then there is a state $s \in \text{RS}(KC(\Pi), \text{type}, R, A)$ such that

$$(\text{Test}, \sigma_{s, \text{Test}}(\text{claim}_{l_1}(R, \text{commit}, R', K^{\# \text{Test}}))) \in tr_s$$

and there is no $rid \in \text{RID}$ where both $\text{role}_s(rid) = R'$ and

$$(rid, \sigma_{s, \text{Test}}(\text{claim}_{l_1}(R', \text{running}, R, K^{\# \text{Test}}))) \in tr_s.$$

For all $L \subseteq \text{Label}$, we define $\delta_L(\langle \rangle) = \langle \rangle$ and

$$\delta_L(\langle (rid, e) \rangle . u) = \begin{cases} \delta(u), & \text{if } \text{label}(e) \in L, \\ \langle (rid, e) \rangle . \delta(u), & \text{otherwise.} \end{cases}$$

We want to construct an attack $s' \in \text{RS}(\Pi, \text{type}, R, A)$ on the secrecy of K where $tr_{s'} = \delta_{\{l_1, l_2\}}(tr_s)$.

First we prove that h does not appear in $\delta_{\{l_1, l_2\}}(tr_s)$. Assume that the opposite is true, i.e. that for $T = \{h(t) : t \in \text{Term}\}$, there exist $\tau \in T$, $rid \in \text{RID}$ and $e' \in \text{RunEvent}$ such that $(rid, e') \in tr_s$, $\tau \sqsubseteq \text{cont}(e')$ and $\text{label}(e') \notin \{l_1, l_2\}$. Then either there exists $e'' \in \Pi(\text{role}_s(rid))$ such that $x \sqsubseteq_{\text{acc}} \text{cont}(e'')$ and $\tau \sqsubseteq \sigma_{s', rid}(x)$, or there is a $\tau' \in T$ that occurs in $\Pi(\text{role}_s(rid))$ and $\sigma_{s, rid}(\tau') = \tau$. Both cases contradict the assumptions on h .

The construction of s' proceeds inductively, by following $\delta_{\{l_1, l_2\}}(tr_s)$. For prefix length 0, we know the state

$$s'_0 = (\langle \rangle, AK_0, \text{Test} \mapsto \sigma_{s, \text{Test}}(\Pi(R)^{\# \text{Test}}), \text{Test} \mapsto \sigma_{s, \text{Test}})$$

is reachable. The only interesting case in the induction step involves checking if **recv** transitions are still enabled. Let s'_n be a state reached after n transitions from s'_0 and $e \in \text{RunEvent}$ with $\text{evtype}(e) = \text{recv}$ the next event. All we need to prove is $AK_{s'_n} \vdash \text{cont}(e)$. Since e is also an event in tr_s , let $s_n \in \text{RS}(KC(\Pi), \text{type}, R, A)$ be any state such that $s_n \rightarrow^* s$, just after e is executed. The only **send** events deleted by $\delta_{\{l_1, l_2\}}$ are the ones labelled l_2 . Hence there exists a finite set $T_\delta \subseteq T$ such that $AK_{s'_n} = AK_{s_n} \setminus T_\delta$. We know that $\text{cont}(e)$ does not contain h . Since for all $\tau \in T_\delta$, $\tau \not\sqsubseteq AK_{s'_n}$, we can apply Lemma 21 to get $AK_{s'_n} \vdash \text{cont}(e)$.

Let $\tau = \sigma_{s, \text{Test}}(h(R, R', K)^{\# \text{Test}})$ for the rest of this proof. We still need to prove $AK_{s'} \vdash \sigma_{s', \text{Test}}(K^{\# \text{Test}})$, where the discussed instance of K , i.e. $\sigma_{s', \text{Test}}(K^{\# \text{Test}}) = \sigma_{s, \text{Test}}(K^{\# \text{Test}})$, is a subterm of τ . To that end, we prove that no **send** event in tr_s contains τ as a syntactic subterm. Suppose $rid \in \text{RID}$ and $e' \in \text{RunEvent}$ such that $(rid, e') \in tr_s$, $\text{evtype}(e') = \text{send}$, and $\tau \sqsubseteq \text{cont}(e')$. Since $\tau \in T$, we know that $\text{label}(e') = l_2$ and $\text{role}_s(rid) = R'$. From $\tau \sqsubseteq \text{cont}(e')$, we have $\tau = \text{cont}(e')$ or $\tau \sqsubseteq \sigma_{s, rid}(K'^{\# rid})$. Because of the running claim which precedes the **send** _{l_2} role event in $KC(\Pi)$, $\tau = \text{cont}(e')$ would contradict our assumption that no such claim occurs in tr_s . However, $\tau \sqsubseteq \sigma_{s, rid}(K'^{\# rid})$ contradicts the typing assumptions on K' . Therefore, no **send** event in tr_s contains τ as a syntactic subterm, so $\tau \not\sqsubseteq AK_s$ holds.

From $AK_s \vdash \tau$ and Lemma 19, we get that the adversary constructed the hash τ itself, i.e. $AK_s \vdash \sigma_{s', \text{Test}}(K^{\# \text{Test}})$. We have $AK_{s'} = AK_s \setminus T'$, for some finite $T' \subseteq T$. For all $\tau' \in T'$, we have $\tau' \not\sqsubseteq \sigma_{s', \text{Test}}(K^{\# \text{Test}})$ and $\tau' \not\sqsubseteq AK_{s'}$ because h does not occur in $tr_{s'}$, so Lemma 21 gives us $AK_{s'} \vdash \sigma_{s', \text{Test}}(K^{\# \text{Test}})$.

$\boxed{\Leftarrow}$ Let $s \in \text{RS}(\Pi, \text{type}, R, A)$ such that

$$(\text{Test}, \sigma_{s, \text{Test}}(\text{claim}_{l_0}(R, \text{secret}, K^{\# \text{Test}}))) \in tr_s$$

and $AK_s \vdash \sigma_{s,Test}(K^{\#Test})$. We now construct $s' \in \text{RS}(KC(\Pi), \text{type}, R, A)$ such that

$$(Test, \sigma_{s',Test}(\text{claim}_{l_1}(R, \text{commit}, R', K)^{\#Test})) \in tr_{s'}$$

and there is no $rid \in \text{RID}$ such that $\text{role}_{s'}(rid) = R'$ and

$$(rid, \sigma_{s',Test}(\text{claim}_{l_1}(R', \text{running}, R, K)^{\#Test}))$$

is in $tr_{s'}$. We know $th_s(Test) = \langle \rangle$, because claim_{l_0} is the last step of the role. For all $rid \in \text{dom}(th_s)$, we define $th_{s'}(rid)$:

$$\begin{aligned} & th_s(rid). \sigma_{s,rid}(\langle \text{recv}_{l_2}(R', R, h(R, R', K)), \\ & \text{claim}_{l_1}(R, \text{commit}, R', K) \rangle^{\#rid}) \end{aligned}$$

if $\text{role}_s(rid) = R$ and $rid \neq Test$,

$$\begin{aligned} & th_s(rid). \sigma_{s,rid}(\langle \text{claim}_{l_1}(R', \text{running}, R, K'), \\ & \text{send}_{l_2}(R', R, h(R, R', K')) \rangle^{\#rid}) \end{aligned}$$

if $\text{role}_s(rid) = R'$, and $th_s(rid)$ otherwise. Then for

$$\begin{aligned} s' = & (tr_s. \sigma_{s,Test}(\langle (Test, \text{recv}_{l_2}(R', R, h(R, R', K))), \\ & (Test, \text{claim}_{l_1}(R, \text{commit}, R', K)) \rangle^{\#Test}), AK_s, th_{s'}, \sigma_s), \end{aligned}$$

we have $s' \in \text{RS}(KC(\Pi), \text{type}, R, A)$. Therefore,

$$(KC(\Pi), \text{type}) \not\models_A \text{claim}_{l_1}(R, \text{commit}, R', K). \quad \square$$

3.3 Achieving AKCS by transformation

In this section, we show how to avoid AKC vulnerabilities during the protocol design stage. We achieve this by exploiting unilateral protocols whose security only depends on the long-term secret keys of the peers. To ensure that such keys are unavailable to the adversary, we restrict ourselves to protocols where no role sends out accessible asymmetric long-term secret keys.

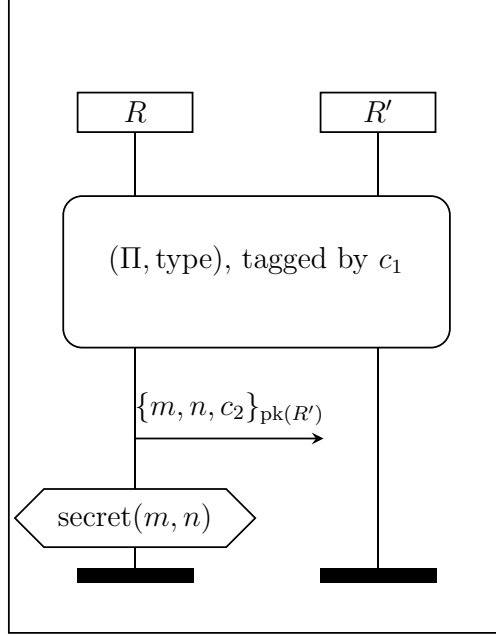


Figure 3.7: Transforming Π for secrecy of m

3.3.1 Achieving AKCS of secrecy

We first present a transformation that ensures the AKCS of secrecy. The transformation, shown in Figure 3.7, adds a single message exchange containing an asymmetrically encrypted message. Its payload is a nonce with any kind of additional data, and it stays secret due to a combination of typing assumptions and tagging. This combination prevents a rerouting of the message to the old part of the protocol.

Definition 23 (Tagging function τ_c). Let $c \in \text{Const}$. We define $\tau_c : \text{Term} \rightarrow \text{Term}$ for all t, t_1, t_2 and f of arity 1 with

$$\tau_c(t) = \begin{cases} t, & \text{if } t \text{ atomic or long-term key,} \\ (\tau_c(t_1), \tau_c(t_2)), & \text{if } t = (t_1, t_2), \\ \{\tau_c(t_1), c\}_{\tau_c(t_2)}, & \text{if } t = \{t_1\}_{t_2}, \\ f(\tau_c(t_1), c), & \text{if } t = f(t_1). \end{cases}$$

The following lemma tells us that if some non-constant atomic term is only accessible within an encryption, then it will remain accessible only within that same encryption in every inferable term, unless the decryption key is known.

Lemma 24. Let $S \cup \{t, t', K\} \subseteq \text{Term}$ and $\perp \in \text{Const}$ such that $t \notin \text{Const}$ is atomic, $t \not\sqsubseteq_{acc} \{\perp / \{t'\}_K\}(S)$, and $S \not\vdash K^{-1}$. Then for all t'' such that $S \vdash t''$, $t \not\sqsubseteq_{acc} \{\perp / \{t'\}_K\}(t'')$.

Proof. We prove the lemma by induction on the derivation height n for t'' from S . If $n = 0$, the statement holds. Assume $n \neq 0$ and fix any derivation of height n . The inductive hypothesis tells us that $t \not\sqsubseteq_{\text{acc}} \{\perp / \{t'\}_K\}(u)$ for all inferred terms u appearing in the premises of the last rule in the derivation.

We now consider all the possible cases for the last rule in the derivation. We know it cannot be a decryption with key K^{-1} due to $S \not\vdash K^{-1}$. For any other rule, $t \sqsubseteq_{\text{acc}} \{\perp / \{t'\}_K\}(t'')$ would imply $t \sqsubseteq_{\text{acc}} \{\perp / \{t'\}_K\}(u)$ for some u in the premises, which contradicts the inductive hypothesis. $\square$

We require that no role is instructed to send its asymmetric long-term secret key in an accessible position. Therefore, since no adversary can *reveal* asymmetric long-term secret keys of peers, they can also never *infer* those keys. The lemma in which we prove this statement must be restricted if more powerful adversaries such as the ones in [6] are allowed.

Lemma 25 (Peers' asymmetric secret keys not inferable). *Let (Π, type) be a protocol where no asymmetric long-term secret keys appear in accessible positions in send events. Let $R \in \text{dom}(\Pi)$, A an adversary, $s \in \text{RS}(\Pi, \text{type}, R, A)$ a state, and $a \in \{\sigma_{s, \text{Test}}(R') : R' \in \text{dom}(\Pi) \setminus \{R\}\}$ an agent. Then $AK_s \not\vdash \text{sk}(a)$ holds.*

Proof. We prove the statement by contradiction. Assume $AK_s \vdash \text{sk}(a)$. Then Lemma 19 implies $\text{sk}(a) \sqsubseteq_{\text{acc}} AK_s$, because no derivation of $\text{sk}(a)$ can end in a composition rule. Since $\text{sk}(a) \not\sqsubseteq_{\text{acc}} AK_0$, there exist $s', s'' \in \text{RS}(\Pi, \text{type}, R, A)$ where $s'' \rightarrow s' \rightarrow^* s$, $\text{sk}(a) \not\sqsubseteq_{\text{acc}} AK_{s''}$ and $\text{sk}(a) \sqsubseteq_{\text{acc}} AK_{s'}$. Therefore, for $\text{rid} \in \text{RID}$ and $e \in \text{Event}$ such that $\text{last}(tr_{s'}) = (\text{rid}, e)$, either $(\text{rid}, e) = (\text{rid}_A, \text{LKR}(a))$, which contradicts the fact that adversaries cannot reveal peers' keys, or (rid, e) is the first send of $\text{sk}(a)$ in an accessible position in tr_s .

For $e' \in \Pi(\text{role}_{s'}(\text{rid}))$ such that $\text{label}(e') = \text{label}(e)$, the existence of $t \in \text{RoleTerm}$ such that $\text{sk}(t) \sqsubseteq_{\text{acc}} \text{cont}(e')$ would contradict our assumption on (Π, type) . Hence, we conclude that there exists $x \in \text{Var}$ such that $x \sqsubseteq_{\text{acc}} \text{cont}(e')$ and $\text{sk}(a) \sqsubseteq_{\text{acc}} \sigma_{s', \text{rid}}(x)$.

Assume e'' is the role event of the **recv** where rid initialised the variable x , in s'' or a reachable state before s'' . Let $t = \text{cont}(\sigma_{s'', \text{rid}}(e''^{\# \text{rid}}))$. We have $AK_{s''} \vdash t$, $\text{sk}(a) \sqsubseteq_{\text{acc}} t$, and $\text{sk}(a) \not\sqsubseteq_{\text{acc}} AK_{s''}$. Lemma 18 then gives us $AK_{s''} \vdash \text{sk}(a)$. From Lemma 19 we deduce that there is a derivation of $\text{sk}(a)$ from $AK_{s''}$ that ends in a composition rule, which contradicts the definition of the $\vdash$ relation. $\square$

For the transformation in Figure 3.7, we use the function:

$$TS(\Pi)(x) = \begin{cases} \tau_{c_1}(\Pi(R)).\langle \mathbf{send}_l(R, R', \{m, n, c_2\}_{\text{pk}(R')}) \rangle, \\ \mathbf{claim}_{l'}(R, \text{secret}, (m, n)), & \text{if } x = R, \\ \tau_{c_1}(\Pi(x)), & \text{if } x \in \text{dom}(\Pi) \setminus \{R\}. \end{cases}$$

For the new type, we use the composition $\tau_{c_1} \circ \text{type}$, i.e. a tagged version of the old one, so that sent messages can still be received. The transformation TS will only be useful if the protocol designer ensures that R can indeed send the added message. In other words, $(TS(\Pi), \tau_{c_1} \circ \text{type})$ must be a protocol.

Proposition 26 (Secrecy by asymmetric encryption). *Let (Π, type) be a protocol and $R, R' \in \text{dom}(\Pi)$ where $R \neq R'$. Let $c_1, c_2 \in \text{Const}$, $l, l' \in \text{Label}$ and $n \in \text{Fresh}$ all be unused in Π such that $c_1 \neq c_2$ and $l \neq l'$. Let $m \in \text{RoleTerm}$ and let A be an adversary. If $(TS(\Pi), \tau_{c_1} \circ \text{type})$ is a protocol where no asymmetric long-term secret keys appear in accessible positions in $\mathbf{send}$ events, then*

$$(TS(\Pi), \tau_{c_1} \circ \text{type}) \models_A \mathbf{claim}_{l'}(R, \text{secret}, (m, n)).$$

Proof. Let $s \in \text{RS}(TS(\Pi), \tau_{c_1} \circ \text{type}, R, A)$ such that

$$(Test, \sigma_{s, Test}(\mathbf{claim}_{l'}(R, \text{secret}, (m^{\#Test}, n^{\#Test})))) \in tr_s.$$

We want to prove that $AK_s \not\vdash \sigma_{s, Test}(m^{\#Test}, n^{\#Test})$. First we use induction over the prefix length of tr_s to prove that $n^{\#Test}$ only appears accessible in $\mathbf{send}$ events in tr_s as a subterm of $\sigma_{s, Test}(\{m^{\#Test}, n^{\#Test}, c_2\}_{\text{pk}(R')})$. Let $s' \in \text{RS}(TS(\Pi), \tau_{c_1} \circ \text{type}, R, A)$, $rid \in \text{RID}$ and $e \in \text{Event}$ such that $s' \rightarrow^* s$, $(rid, e) = \text{last}(tr_{s'})$, $\text{evtype}(e) = \mathbf{send}$ and $n^{\#Test} \sqsubseteq_{\text{acc}} \text{cont}(e)$. If $rid = Test$, the statement is true, because the only time $Test$ sends $n^{\#Test}$ is when $n^{\#Test}$ is generated in the event labelled l .

Otherwise, if $rid \neq Test$, let $\perp \in \text{Const}$. From the inductive hypothesis, we know² that $n^{\#Test}$ only appears accessible inside $\sigma_{s, Test}(\{m^{\#Test}, n^{\#Test}, c_2\}_{\text{pk}(R')})$ in $AK_{s'} \setminus \{\text{cont}(e)\}$, i.e.

$$n^{\#Test} \not\sqsubseteq_{\text{acc}} \{\perp / \sigma_{s, Test}(\{m^{\#Test}, n^{\#Test}, c_2\}_{\text{pk}(R')})\}(AK_{s'} \setminus \{\text{cont}(e)\}).$$

However, from Lemma 25 we get $AK_s \not\vdash \text{sk}(\sigma_{s, Test}(R'))$. For that reason, Lemma 24 tells us that for all $t \in \text{Term}$ such that $AK_{s'} \setminus \{\text{cont}(e)\} \vdash t$,

$$n^{\#Test} \not\sqsubseteq_{\text{acc}} \{\perp / \sigma_{s, Test}(\{m^{\#Test}, n^{\#Test}, c_2\}_{\text{pk}(R')})\}(t),$$

²If $\text{cont}(e)$ was sent previously, then the inductive hypothesis additionally covers $\text{cont}(e)$, which simplifies the proof.

so $n^{\#Test}$ is accessible in **recv** events in $tr_{s'}$ only inside $\sigma_{s,Test}(\{m^{\#Test}, n^{\#Test}, c_2\}_{pk(R')})$.

Suppose $n^{\#Test}$ occurs accessible outside the term $\sigma_{s,Test}(\{m^{\#Test}, n^{\#Test}, c_2\}_{pk(R')})$ in $\text{cont}(e)$. But then there is a tagged **recv** step of Π where a variable is initialised with $n^{\#Test}$ outside of the encryption, i.e. there exist $\{t\}_{t'} \in \text{RoleTerm}$ and $x \in \text{Var}$ such that $\{t\}_{t'} \sqsubseteq_{\text{acc}} \Pi(\text{role}_s(\text{rid}))$, $x \sqsubseteq_{\text{acc}} t$, $n^{\#Test} \sqsubseteq_{\text{acc}} \sigma_{s,rid}(x)$, and

$$\sigma_{s,rid}(\tau_{c_1}((\{t\}_{t'})^{\#rid})) = \sigma_{s,Test}(\{m^{\#Test}, n^{\#Test}, c_2\}_{pk(R')}),$$

which contradicts $c_1 \neq c_2$. Hence, $n^{\#Test}$ is only accessible in the set AK_s as a subterm of the term $\sigma_{s,Test}(\{m^{\#Test}, n^{\#Test}, c_2\}_{pk(R')})$. From Lemma 24 we now conclude $AK_s \not\models n^{\#Test}$, which implies $AK_s \not\models \sigma_{s,Test}(m^{\#Test}, n^{\#Test})$. $\square$

3.3.2 Achieving AKCS of agreement

We now define a function TA that transforms a protocol into one that achieves non-injective agreement, as depicted in Figure 3.8. A message signed by a peer convinces the test run that at least one of the peer's runs in the correct role agrees on the message:

$$TA(\Pi)(x) = \begin{cases} \tau_{c_1}(\Pi(R')).\langle \text{claim}_l(R', \text{running}, R, n), \\ \text{send}_{l'}(R', R, \{R, n, c_2\}_{sk(R')}) \rangle, & \text{if } x = R', \\ \tau_{c_1}(\Pi(R)).\langle \text{recv}_{l'}(R', R, \{R, x_n, c_2\}_{sk(R')}), \\ \text{claim}_l(R, \text{commit}, R', x_n) \rangle, & \text{if } x = R, \\ \tau_{c_1}(\Pi(x)), & \text{if } x \in \text{dom}(\Pi) \setminus \{R, R'\}. \end{cases}$$

$$\text{type}'(x) = \begin{cases} (\tau_{c_1} \circ \text{type})(x), & \text{if } x \in \text{Var} \setminus \{x_n\}, \\ (\tau_{c_2} \circ \text{type})(x_n), & \text{if } x = x_n. \end{cases}$$

The proof uses the fact that the adversary cannot forge the signature in $(TA(\Pi), \text{type}')$, because by Lemma 25 it cannot get the required key. Through the signature, the peer confirms that it agrees on the identities and the data to the actor. Note that this time we do not need the assumption that the result of the transformation is a protocol in the proposition statement; this condition is trivially satisfied.

Proposition 27 (Agreement by signing). *Let (Π, type) be a protocol and $R, R' \in \text{dom}(\Pi)$ such that $R \neq R'$. Let $l, l' \in \text{Label}$, $n \in \text{Fresh}$, $x_n \in \text{Var}$ where $n \in \text{type}(x_n)$, and $c_1, c_2 \in \text{Const}$ all be different and unused in Π , and A an adversary. If no asymmetric long-term secret keys appear in accessible positions in **send** events in (Π, type) , then*

$$(TA(\Pi), \text{type}') \models_A \text{claim}_l(R, \text{commit}, R', x_n).$$

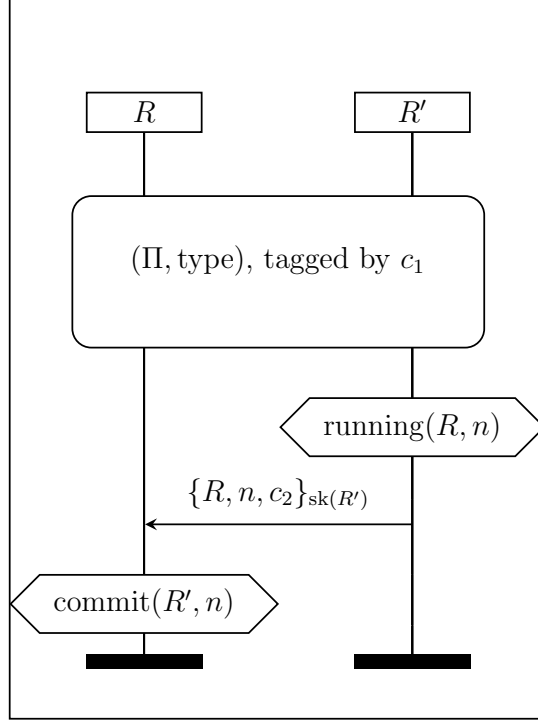


Figure 3.8: Transforming Π for non-injective agreement on n

Proof. Let $s \in \text{RS}(TA(\Pi), \text{type}', R, A)$ and

$$(\text{Test}, \sigma_{s, \text{Test}}(\text{claim}_l(R, \text{commit}, R', x_n))) \in tr_s.$$

We now prove that a run executed the corresponding running claim. Denote $t = \sigma_{s, \text{Test}}(\{R, x_n, c_2\}_{\text{sk}(R')})$. From the above, there exists an $s' \in \text{RS}(TA(\Pi), \text{type}', R, A)$ such that $s' \rightarrow^* s$ and

$$(\text{Test}, \text{recv}_{l'}(\sigma_{s, \text{Test}}(R'), \sigma_{s, \text{Test}}(R), t)) = \text{last}(tr_{s'}).$$

From this, we have $AK_{s'} \vdash t$. We know from Lemma 25 that $AK_{s'} \not\vdash \text{sk}(\sigma_{s, \text{Test}}(R'))$ holds. That means no derivation of t from $AK_{s'}$ can end in a composition step, so Lemma 19 now implies $t \sqsubseteq_{\text{acc}} AK_{s'}$. Thus, there exist $\text{rid} \in \text{RID}$ and $e \in \text{RunEvent}$ such that $\text{evtype}(e) = \text{send}$, $(\text{rid}, e) \in tr_{s'}$ and $t \sqsubseteq_{\text{acc}} \text{cont}(e)$.

Without loss of generality, let (rid, e) be the first **send** with the above properties in $tr_{s'}$. We know that for all $x \in \text{Var}$ accessible in $\text{upto}(TA(\Pi)(\text{role}_{s'}(\text{rid})), \text{label}(e))$, $t \not\sqsubseteq_{\text{acc}} \sigma_{s', \text{rid}}(x)$, i.e. t cannot occur in $\text{cont}(e)$ as an accessible subterm of any instantiated variable in an earlier **recv** step, since Lemma 18 would imply the adversary knew t even earlier, which would cause Lemma 25 and Lemma 19 to contradict the minimality of (rid, e) .

Suppose that $\text{label}(e) \neq l'$. Then e is an instance of a tagged step of Π , i.e. there is a unique $e' \in \text{RoleTerm}$ such that $\sigma_{s',rid}(e'^{\#rid}) = \text{cont}(e)$ and $\text{send}_{\text{label}(e)}(\cdot, \cdot, e') \in \tau_{c_1}(\Pi(\text{role}_{s'}(rid)))$. From the previous paragraph, we know there must be a $\{t_0\}_{t_1} \in \text{RoleTerm}$ such that $\sigma_{s',rid}(\tau_{c_1}(\{t_0\}_{t_1})^{\#rid}) = t$. However, that implies $c_1 = c_2$, which contradicts our assumption. Hence, $\text{label}(e) = l'$, $\text{role}_{s'}(rid) = R'$, and

$$\sigma_{s',rid}(\text{claim}_l(R', \text{running}, R, n^{\#rid})) \in tr_{s'}.$$

Moreover, $t \sqsubseteq_{\text{acc}} \text{cont}(e)$ implies $t = \text{cont}(e)$, so we get $\sigma_{s,Test}(x_n) = n^{\#rid}$. $\square$

3.4 Impossibility results

It is conjectured in [16] that KCIR requires the use of asymmetric cryptography. We give a partial confirmation of this conjecture: under weak assumptions on the protocol specification and the adversary model, the use of just symmetric cryptography and hashing cannot ensure AKCR for a large class of security properties. This class includes, for example, all authentication properties in Lowe's hierarchy [63]. We prove the result for aliveness and generalise it to all stronger claims.

For the proof of our first impossibility result, we need a lemma that states that we can infer a term from all its subterms that are atomic or long-term keys.

Lemma 28 (Composition from atomic subterms). *Let $S \cup \{t\} \subseteq \text{Term}$. If for all $x \sqsubseteq t$ where x is atomic or x is a long-term key, $S \vdash x$, then $S \vdash t$.*

Proof. We prove the statement by structural induction on t . If t is atomic or a long-term key, since $t \sqsubseteq t$, the assumption gives us $S \vdash t$. If for some t_1, t_2 , the term t is equal to (t_1, t_2) , $\{t_1\}_{t_2}$ or $f(t_1)$, then the inductive hypothesis implies $S \vdash t_1$, along with $S \vdash t_2$ in the first two cases. We can then use the corresponding composition rule to get t . $\square$

Proposition 29 (Impossibility of aliveness). *Let (Π, type) be a protocol, $R, R' \in \text{dom}(\Pi)$ such that $R \neq R'$, $l \in \text{Label}$, and $\text{claim}_l(R, \text{alive}, R') \in \Pi(R)$. If for all $S, T \in \text{Role}$, $x \in \text{Var}$, and $n \in \text{Fresh}$,*

- $\text{pk}(S) \not\sqsubseteq \Pi(R)$ and $\text{sk}(S) \not\sqsubseteq \Pi(R)$,
- if $\text{k}(S, T) \sqsubseteq \Pi(R)$, then $S = R$ or $T = R$,
- there exists $n_x \in \text{Fresh}$ such that $n_x^{\#rid_A} \in \text{type}(x)$, and

- if $n \sqsubseteq \Pi(R)$, n in $\Pi(R)$ appears first in a **send**, in accessible positions only,

then $(\Pi, \text{type}) \not\models_{\{\text{LKR}_{\text{actor}}\}} \text{claim}_l(R, \text{alive}, R')$.

Proof. Let $a, b \in \text{Agent}$ such that $a \neq b$ and define the substitution τ as follows:

$$\tau(x) = \begin{cases} a, & \text{if } x = R, \\ b, & \text{if } x \in \text{Role} \setminus \{R\}, \\ n_x^{\#rid_A}, & \text{if } x \in \text{Var}. \end{cases}$$

By the third assumption in the proposition statement, $\tau \in GS(\text{Role} \cup \text{Var}, \text{type})$. If the length of $\Pi(R)$ is k for some $k \in \mathbb{N}$, we define:

$$\begin{aligned} seq &= \langle (Test, \tau(\Pi(R)^{\#Test})_1), \dots, (Test, \tau(\Pi(R)^{\#Test})_k) \rangle \\ s &= (\langle (rid_A, \text{LKR}(a)) \rangle.seq, AK_0 \cup \text{LTK}(a) \cup \\ &\quad (\tau(\Pi(R)^{\#Test}) \downarrow \text{send}), Test \rightarrow \langle \rangle, Test \mapsto \tau). \end{aligned}$$

The proof of reachability proceeds by induction on the prefix length of the sequence $\langle (rid_A, \text{LKR}(a)) \rangle.seq$. More specifically, we prove two statements within the induction:

- the adversary can infer all sent nonces just after they are first sent, and
- the adversary can infer the contents of all **recv** events just before they occur.

For lengths 0 and 1, we know that the states

$$\begin{aligned} s_0 &= (\langle \rangle, AK_0, Test \mapsto \tau(\Pi(R)^{\#Test}), Test \mapsto \tau) \text{ and } s_1 = \\ &(\langle (rid_A, \text{LKR}(a)) \rangle, AK_0 \cup \text{LTK}(a), Test \mapsto \tau(\Pi(R)^{\#Test}), Test \mapsto \tau) \end{aligned}$$

are reachable and that no nonces have yet been sent. Let $m \in \mathbb{N}$, s_m a state reached after m transitions from s_0 , and $e \in \text{RunEvent}$ the next event. Let K be any long-term key such that $K \sqsubseteq \text{cont}(e)$. We want to prove that $K \in AK_{s_m}$. Since τ assigned nonces local to rid_A to variables in $\Pi(R)$, we know that K does not occur inside a variable instance in $\text{cont}(e)$. Therefore, there is a t in $\Pi(R)$ of the form $k(\cdot, \cdot)$, $\text{pk}(\cdot)$ or $\text{sk}(\cdot)$ such that $K = \sigma_{s_m, Test}(t^{\#Test}) = \tau(t^{\#Test})$. The first assumption in the proposition statement now gives us that $K = k(c, d)$ for some $c, d \in \text{Agent}$, and the second one implies that $c = a$ or $d = a$. Hence $K \in \text{LTK}(a)$, which implies $K \in AK_{s_1}$. Since $AK_{s_1} \subseteq AK_{s_m}$, we get $K \in AK_{s_m}$.

Suppose now that $(Test, e)$ is the first **send** for $n^{\#Test}$ where $n \in \text{Fresh}$. By the inductive hypothesis, we know that for all $n'^{\#Test}$ sent strictly before $(Test, e)$ in tr_{s_m} , we have $AK_{s_m} \vdash n'^{\#Test}$. By the fourth assumption in the proposition statement,

we know all other nonces local to $Test$ that appear for the first time in $(Test, e)$, appear in $\text{cont}(e)$ in accessible positions only. Therefore, $AK_{s_m} \cup \{\text{cont}(e)\}$ infers all inaccessible atomic subterms and long-term keys in $\text{cont}(e)$. By induction on $t \sqsubseteq_{\text{acc}} \text{cont}(e)$, we prove $AK_{s_m} \cup \{\text{cont}(e)\} \vdash t$. The basis, where $t = \text{cont}(e)$, is trivial. Assume now that $t \neq \text{cont}(e)$, and let $\text{cont}(e) = \{t_1\}_{t_2}$ first. We want to conclude $AK_{s_m} \cup \{\text{cont}(e)\} \vdash t_2^{-1}$. We know that all atomic subterms and long-term keys in t_2 are inferable by $AK_{s_m} \cup \{\text{cont}(e)\}$, so by Lemma 28 we have $AK_{s_m} \cup \{\text{cont}(e)\} \vdash t_2$. By construction of seq , we have $t_2 = t_2^{-1}$, so $AK_{s_m} \cup \{\text{cont}(e)\} \vdash t_2^{-1}$. We can therefore apply the decryption rule to get $AK_{s_m} \cup \{\text{cont}(e)\} \vdash t_1$. Finally, if $\text{cont}(e) = (t_1, t_2)$, without loss of generality we can assume that $t \sqsubseteq_{\text{acc}} t_1$ and use the unpairing rule to get $AK_{s_m} \cup \{\text{cont}(e)\} \vdash t_1$. By proceeding inductively, we get $AK_{s_m} \cup \{\text{cont}(e)\} \vdash t$. In particular, we get $AK_{s_m} \cup \{\text{cont}(e)\} \vdash n^{\#Test}$, which proves the first statement.

Now assume $\text{evtype}(e) = \text{recv}$. We need to check that e is enabled, i.e. that $AK_{s_m} \vdash \text{cont}(e)$ holds. All nonces local to $Test$ to be received in $\text{cont}(e)$ have been sent previously by $Test$, so we know they are inferable by AK_{s_m} . Additionally, AK_{s_m} already contains all long-term keys in $\text{cont}(e)$, and AK_{s_0} contains all nonces local to rid_A and the names of all agents, so we can apply Lemma 28 to get $AK_{s_m} \vdash \text{cont}(e)$. $\square$

Definition 30 (At least as strong security claim). *Let γ and γ' be security claims for $R \in \text{Role}$. We say that γ is at least as strong as γ' if for all protocols (Π, type) such that $R \in \text{dom}(\Pi)$ and $\gamma, \gamma' \in \Pi(R)$, adversaries A , and reachable states $s \in \text{RS}(\Pi, \text{type}, R, A)$, whenever instances of both γ and γ' are executed by $Test$ in tr_s and $s \models \gamma$, then $s \models \gamma'$.*

For example, $\text{claim}_l(R, \text{commit}, R', t)$ is at least as strong as $\text{claim}_{l'}(R, \text{alive}, R')$.

Corollary 31 (Impossibility of authentication). *Under the Proposition 29 assumptions, for every security claim γ that is at least as strong as $\text{claim}_l(R, \text{alive}, R')$ and occurs before it in $\Pi(R)$, if $(\Pi, \text{type}) \models_{\emptyset} \gamma$, then γ is not AKC resilient in (Π, type) with respect to the $\{\text{LKR}_{\text{actor}}\}$ adversary.*

Proof. Let $\gamma' = \text{claim}_l(R, \text{alive}, R')$. By Proposition 29, we get $(\Pi, \text{type}) \not\models_{\{\text{LKR}_{\text{actor}}\}} \gamma'$. Therefore, we know there exists a state $s \in \text{RS}(\Pi, \text{type}, R, \{\text{LKR}_{\text{actor}}\})$ such that $Test$ executed an instance of γ' in tr_s , but $s \not\models \gamma'$. Since γ appears before γ' in $\Pi(R)$, we know $Test$ also executed an instance of γ in tr_s . Since γ is at least as strong as γ' , $s \models \gamma$. This implies $(\Pi, \text{type}) \not\models_{\{\text{LKR}_{\text{actor}}\}} \gamma$. Since $(\Pi, \text{type}) \models_{\emptyset} \gamma$ holds by assumption, we are done. $\square$

	Protocol	AKC attack	Section
1.	Needham-Schroeder-Lowe	Yes	3.5.1
2.	NSL-AKC	No	3.5.1
3.	ITU-T X.509 three-message	Yes	3.5.2
4.	ITU-T X.509 three-message fixed	No	3.5.2
5.	ITU-T X.509 one-message	Yes	3.5.2
6.	ITU-T X.509 one-message fixed	No	3.5.2
7.	SSH Transport Layer	No	3.5.3
8.	Mutual TLS-RSA	Yes	3.5.4
9.	Mutual TLS-DHE_RSA	No	3.5.4
10.	Unilateral TLS-RSA with mod_auth_basic	Yes	3.5.5

Table 3.1: Automatic analysis results

3.5 Case studies

We use Scyther [23] to analyse the Needham-Schroeder-Lowe (NSL) protocol, the ITU (International Telecommunication Union) Telecommunication Standardization Sector (ITU-T) X.509 family of protocols, the SSH Transport Layer protocol, and the TLS protocol. Our findings are summarised in Table 3.1 and include an AKC attack on mutually authenticated TLS-RSA, for which we provide a concrete implementation against an Apache web server running TLS 1.2. The protocol specifications needed to reproduce the tests in this section, and the scripts for the attack, are available at [25].

There are several ways to fix a vulnerable protocol, depending on its requirements and deployment status: (1) switch to a different mode of the protocol within its protocol suite, (2) use generic Propositions 26 and 27 verbatim, or (3) use a modified version of the transformations described in these propositions, perhaps to achieve slightly different security requirements, and prove the resulting protocol secure, e.g. with tool support. The approach listed as (2) is best suited for use in the protocol design stage. We give an application of (1) in the sections on TLS, and examples of (3) when discussing NSL and the ITU-T X.509 protocols.

Note that Proposition 29 tells us what kind of changes are insufficient to ensure AKCS: we cannot achieve authentication under actor key compromise by employing just hashing and symmetric keys. However, adding one or both to a protocol that utilises other mechanisms may suffice. With this in mind, we propose practical fixes for each of the vulnerable protocols.

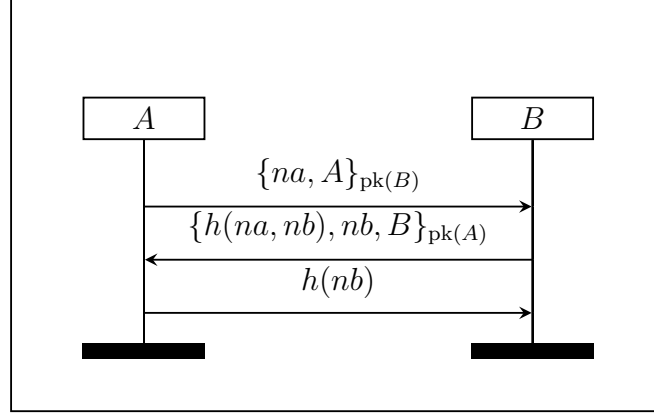


Figure 3.9: NSL-AKC protocol

3.5.1 Needham-Schroeder-Lowe

In the presence of a Dolev-Yao adversary, the Needham-Schroeder-Lowe protocol (cf. Figure 3.3) achieves mutual authentication and secrecy of both nonces [62]. We use Scyther to confirm this result. However, as explained in the introduction to this chapter, NSL is vulnerable to AKC attacks on non-injective agreement on the nonces for the initiator, and secrecy of both nonces for both roles. We can fix the AKC vulnerabilities by hashing the nonces in the response messages, which always prevents leaking one of the two nonces after the adversary reveals the actor's key.

However, the above is insufficient to achieve agreement on both nonces because the claim of an actor in the A role could be violated by the adversary replacing nb by nb' . We remedy this problem by linking the nonces with a hash in the second message. Additionally, there is now no need for encryption in the third message, so it can be removed for efficiency. We call the resulting protocol NSL-AKC. It is shown in Figure 3.9. This protocol is not vulnerable to the attack from the introduction because the adversary does not learn na from decrypting the second message. Therefore, it cannot produce the hash $h(na, nb')$, and agreement on both nonces is achieved even under AKC. In NSL-AKC, AKC leads to the adversary learning the peer's nonce nb . However, the combination of both nonces under a different hash, e.g. $h'(na, nb)$, can still act as a shared secret. We use Scyther to verify that NSL-AKC achieves *synchronisation* (a strong authentication property that implies agreement, cf. [27]) and secrecy of $h'(na, nb)$, with respect to $\{\text{LKR}_{\text{actor}}\}$ and for both roles.

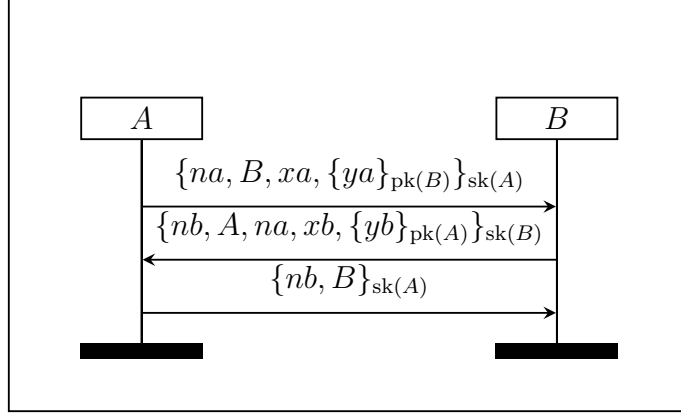


Figure 3.10: ITU-T X.509 three-message protocol

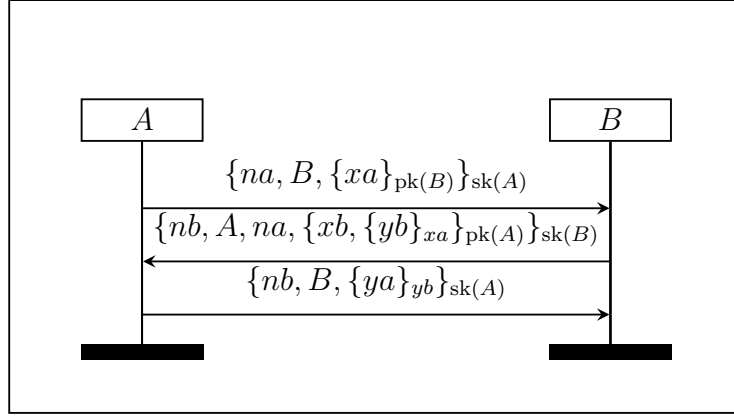


Figure 3.11: AKCS version of ITU-T X.509 three-message protocol

3.5.2 ITU-T X.509 family

We consider a family of protocols from the latest, 2012 recommendations for the ITU-T (formerly CCITT) X.509 standard [48]. The protocols are meant to enable secure and (mutually) authenticated access to a certificate directory. There are AKC attacks on the secrecy of y_b for the A role (likewise, y_a for the B role) in the ITU-T X.509 three-message protocol, which is depicted in Figure 3.10.

We would like to obtain secrecy of y_a for B under AKC. Encrypting y_a should work, as long as we encrypt it with something other than $\text{pk}(B)$. We normally have two sources of secrecy to choose from (long-term secret keys and freshly generated values), but under AKC we cannot depend on the actor's long-term secret keys. Therefore, we postpone the transmission of y_a to the third message and encrypt it with y_b , which is generated in the second message. The encryption with $\text{pk}(A)$ in message 2 makes y_b secret for B , analogous to the transformation in Proposition 26. We make x_a secret for B under AKC by encrypting it with $\text{pk}(B)$. We also encrypt y_b with x_a so that it

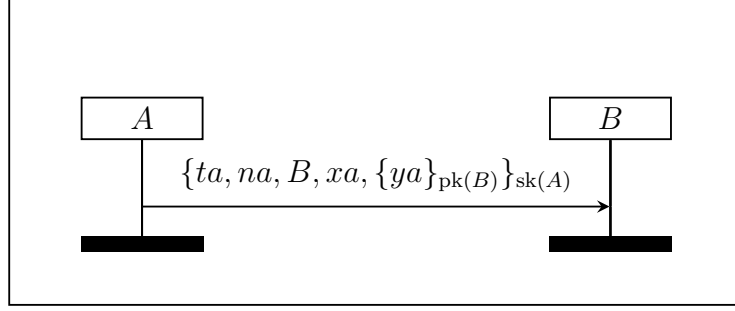


Figure 3.12: ITU-T X.509 one-message protocol

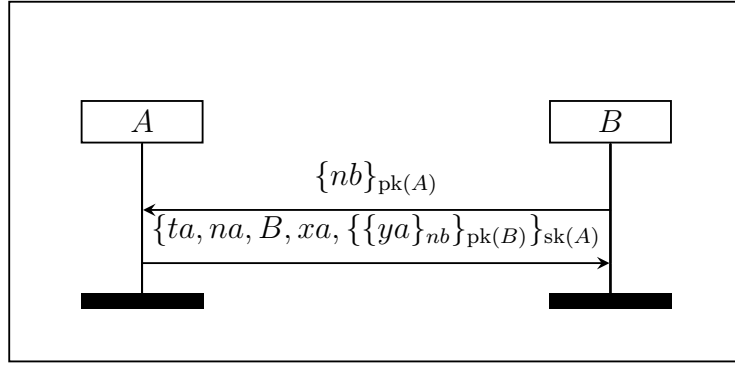


Figure 3.13: AKCS version of ITU-T X.509 one-message protocol

is secret for A . We put xb inside of the encryption with $\text{pk}(A)$ to achieve agreement on xb for B , which leads to synchronisation [27] under AKC for both roles. The AKC secure version of the protocol, shown in Figure 3.11, is successfully verified by Scyther.

The ITU-T X.509 one-message protocol shown in Figure 3.12 is also vulnerable to an AKC attack. Even though it is unilateral, its properties depend on the keys of both parties. There is an AKC attack on the secrecy of ya for the responder, similarly to the three-message protocol.

We would ideally like to prevent the attack without introducing additional message passes, i.e. by transforming the single transmitted message into some message m . Unfortunately, the message m clearly cannot include fresh values generated by the responder. Therefore, the AKC-capable adversary can infer ya from m in the same way as the responder.

Although we cannot avoid adding a message pass to fix the AKC vulnerability, the protocol can be made AKC resilient by prepending a message $\{nb\}_{\text{pk}(A)}$ from B to A , and replacing ya with $\{ya\}_{nb}$ in message 1. The AKC secure version of the protocol is shown in Figure 3.13.

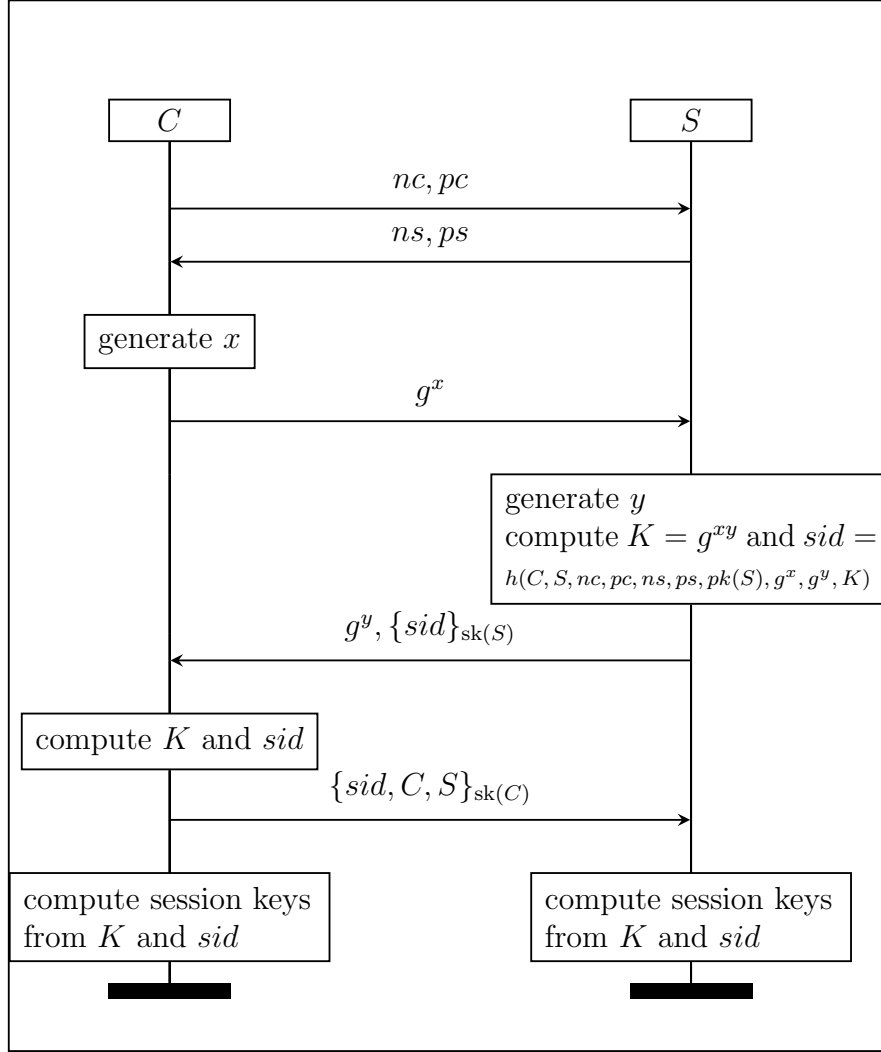


Figure 3.14: SSH Transport Layer protocol

3.5.3 SSH

The Secure Shell (SSH) protocol is used to establish a secure channel between two endpoints, mainly for remote login and command execution purposes. The mutually authenticated public-key version of the SSH Transport Layer protocol [103] consists of establishing cryptographic parameters, and what is essentially a Diffie-Hellman key exchange with transcript confirmation. A simplified depiction of the protocol is provided in Figure 3.14.

We use Scyther to verify the AKC security of session key secrecy and synchronisation in the SSH Transport Layer protocol with respect to $\{\text{LKR}_{\text{actor}}, \text{LKR}_{\text{others}}\}$.

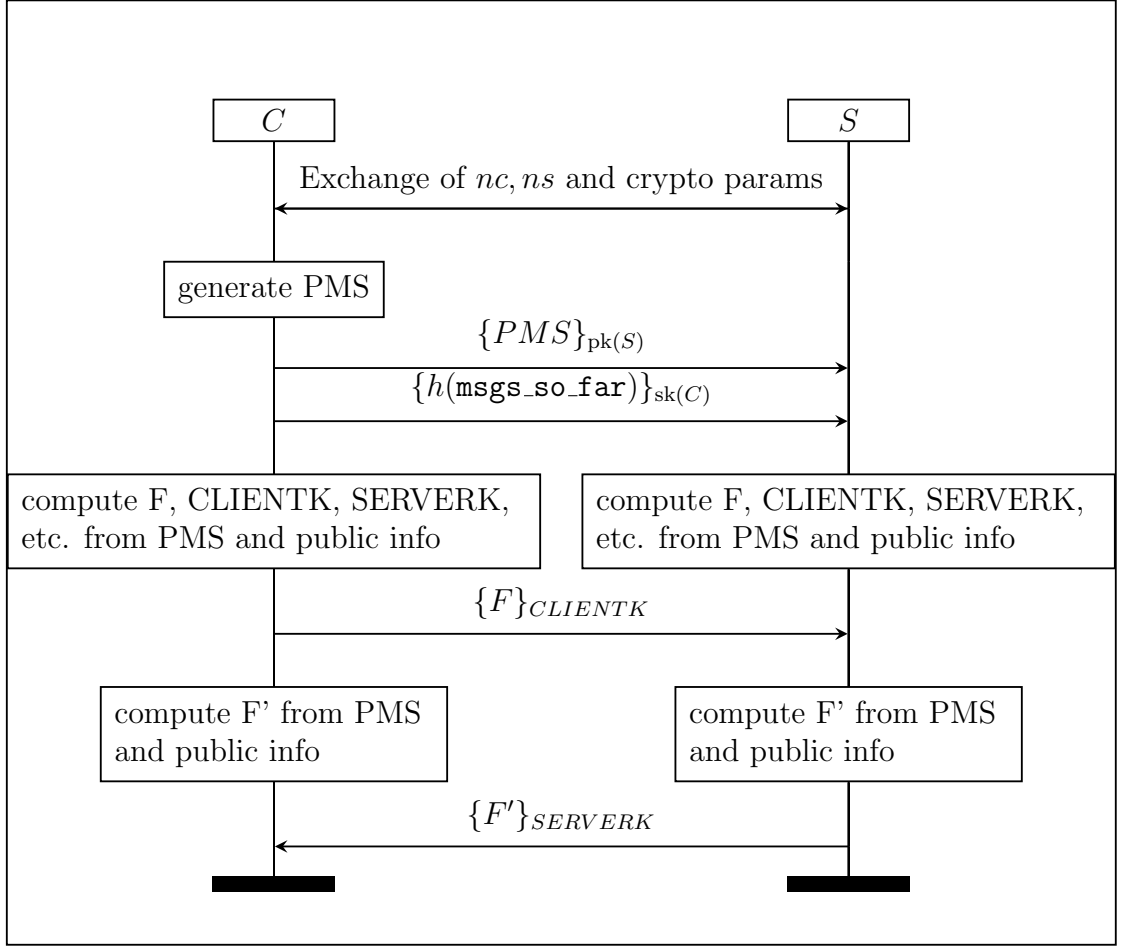


Figure 3.15: Abstract depiction of mutually authenticated TLS-RSA

3.5.4 Mutually authenticated TLS

The TLS protocol [28] is a widely deployed protocol for secure communications on the Internet. It can be used to unilaterally authenticate a server to a client and also supports mutually authenticated modes. We analyse mutual authentication before returning to unilateral authentication in the next section.

The mutually authenticated modes of TLS are typically used in, for example, specialised banking applications [102] and VPN access [78]. A commonly used [100] public-key cryptosystem in TLS is RSA [90]. Abstractly (omitting e.g. the explicit certificate exchange), the TLS-RSA mode of TLS proceeds as depicted in Figure 3.15. The computed session keys are used to encrypt and authenticate the application data in subsequent communications.

We observe that the only secret information involved in the computation of all session keys is the premaster secret PMS . Therefore, the secrecy of the session keys

critically depends on the secrecy of PMS , which in turn is based on the secrecy of the server’s long-term secret key $sk(S)$ (see message 2).

Scyther finds a server-side AKC attack by $\{LKR_{actor}\}$ on session key secrecy. In the attack, the adversary essentially eavesdrops on a regular handshake. Then, by using $sk(S)$, it decrypts $\{PMS\}_{pk(S)}$ to get PMS , enabling it to compute the session keys. Hence the adversary can intercept any subsequently transmitted messages, or inject its own, thereby rendering the established communication channel completely insecure.

We implement this attack against an Apache web server running TLS 1.2 [28], using a man-in-the-middle attack script written in Python. The script connects to an OpenSSL integrated client program (`s_client`) on one end, and the Apache web server on the other. For the AKC attack we provide our script with the long-term secret key of the web server. The attack script eavesdrops on a regular handshake between the server and the client, and uses the messages and the long-term key to compute the session keys. The script can then decrypt all sent application data and is able to insert or modify all received application data, both to and from the web server. While the attack does not depend on specific choices for hash algorithms, ciphers, and their modes of operation, we use SHA-256 and AES-256 in CBC mode in our experiments. The required files and instructions to run the attack can be downloaded from [25].

The simplest way to prevent the AKC attack is to switch to the mutually authenticated DHE_RSA mode of TLS. This mode is not vulnerable to AKC attacks, because it uses temporary Diffie-Hellman public keys (sometimes also called *half keys* or *key shares*) of the form g^x , where x is a freshly generated value. The client’s temporary public key g^x is combined with the server’s temporary secret key y , and vice versa, to obtain g^{xy} . The adversary learns both temporary public keys g^x and g^y , but does not learn x or y , and therefore cannot construct the session key. In this case, we use the Tamarin prover [70] (for its more precise modelling of Diffie-Hellman exponentiation) to successfully verify that session key secrecy is AKC secure in the DHE_RSA mode of our TLS model.

Until now, the reason put forward for using the DHE modes of TLS instead of the RSA mode has been that they offer *Perfect Forward Secrecy* (PFS): even if all long-term keys are compromised at some point, previously sent application data is still secure. Our analysis reveals that there is an additional advantage to TLS-DHE_RSA over TLS-RSA. Namely, the AKC security implies that a server running TLS-DHE_RSA can still communicate securely with clients even if the server’s key is compromised. Our attack proves that this is not the case for the RSA mode.

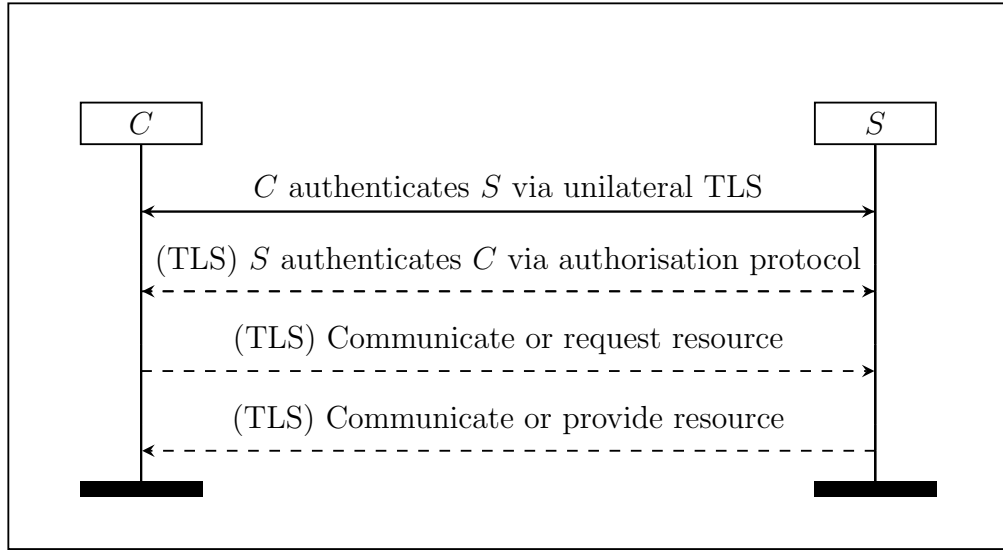


Figure 3.16: Abstract unilateral TLS with client authentication over TLS channel

3.5.5 Unilateral TLS combined with authorisation protocols

A ubiquitous use of TLS involves the unilateral modes, in which only the server has a certificate. In these modes, the client authenticates the server, but the server does not authenticate the client. The message exchanges are similar to mutually authenticated TLS, except that the client does not send a certificate and does not send the so-called CertificateVerify message (message 3 in the earlier TLS description).

Because the only security requirements stem from the client, who uses no secret key or has no secret key to reveal, the unilateral modes are resilient against AKC attacks. However, many applications such as e-banking and online e-mail services require mutual authentication even when the client has no certificate. For such applications, mutual authentication is usually achieved by combining (unilateral) TLS with an authorisation protocol in the following way. First, the client establishes a unilateral TLS session with the server, authenticating the server based on the server's certificate. Then the server initiates an authorisation protocol inside the TLS connection. Typical examples of such protocols are Apache's password-based `mod_auth_basic` [4], or single sign-on protocols such as OAuth [43] or SAML [76]. Abstractly, all these examples take the approach shown in Figure 3.16, where the last three communications are encrypted using the previously established TLS session keys.

We modelled the above setup for unilateral TLS-RSA followed by the default Apache password authentication `mod_auth_basic`. Scyther finds an AKC attack on the server that, upon inspection, is straightforward: the adversary eavesdrops on the TLS handshake and the following authentication, which correspond to steps 1 and 2

above. As in the mutually authenticated case, the adversary can decrypt *PMS* and compute the session keys. It can then arbitrarily eavesdrop on, modify, and inject messages in steps 3 and 4, regardless of the authorisation protocol used. Thus, the mutual authentication protocols obtained by combining unilateral TLS-RSA with either `mod_auth_basic`, OAuth, or SAML, are all vulnerable to AKC attacks on the server.

Because Apache’s `mod_auth_basic` relies on a secret that is known to both the server and the client, compromising the server compromises the secret, which enables AKC attacks against the above setup even when no clients are present. This is not the case in a weaker threat model, where the adversary learns the long-term secret key of the server through cryptanalysis but does not have access to the server’s password store. In both of the above situations, switching to the mutually authenticated DHE-RSA mode of the TLS handshake provides the guarantee of AKCS of session key secrecy to the server.

3.6 Conclusions

One of the guiding principles of modern information security is containment: given that security mechanisms may be compromised, it is prudent to design systems that limit the resulting damage as much as possible. In the domain of security protocols, AKC resilience and security are desirable features because they help contain the effects of key compromise [77, 81]. We have provided the first systematic analysis of this phenomenon and have given conditions under which it can and cannot be achieved.

Our transformations show how to construct protocols that are resilient against AKC. Our work thereby facilitates incremental protocol design, and enables protocol designers to provide strong security guarantees to the users of their protocols, even under actor key compromise.

For existing, widely deployed protocols, we have introduced fixes that use the underlying structure of the protocols at hand. In comparison with using the generic transformations developed in this chapter, this approach enables less intrusive fixes and more efficient results. For TLS-RSA, the most efficient fix is to use the Diffie-Hellman mode.

We showed that asymmetric cryptography is needed to obtain authentication guarantees, which has direct consequences for improving existing protocols and developing new protocols.

Our AKC attacks on protocols such as mutually authenticated and unilateral TLS-RSA show that there is still room for improvement in practice, and reveal that perfect forward secrecy is not the only advantage of using TLS-DHE_RSA.

Chapter 4

Improving the ISO/IEC 11770 standard

In the previous chapter, we gave an in-depth analysis of the consequences of KCI and its generalisation, and constructing countermeasures. We apply the developed theory in this chapter, where we provide the first systematic analysis of the ISO/IEC 11770 standard for key management techniques [46, 47]. The standard describes a set of key establishment, key agreement, and key transport protocols. We analyse the claimed security properties, as well as additional modern requirements on key management protocols, for over 30 protocols and their variants. Our formal, tool-supported analysis of the protocols uncovers several incorrect claims in the standard. We provide concrete suggestions for improving the standard.

4.1 Introduction

The International Organisation for Standardization (ISO) develops and promotes international standards, which include a wide variety of security mechanisms. Many large vendors aim to support ISO standards, for example because they are mandated by oversight bodies [31] or to prevent trade barriers. Hence, it is critical that the ISO standards for security mechanisms are thoroughly scrutinised. However, most previous analyses of the ISO security standards were very limited in scope, e.g. [20, 44, 64, 72]. One exception is the analysis of Basin et al. of the ISO/IEC 9798 standard for entity authentication [8] in 2012. Their analysis uncovered a series of issues that led to an updated version of this standard.

We focus on the ISO/IEC 11770 standard for key management protocols, in particular on Parts 2 and 3 of this standard. In the most recent versions, as of September 2009, these two parts together describe 33 base protocols for key establishment, key

agreement, and key transport. Many of the protocols in the standard are based on protocols such as Diffie-Hellman, variants of Menezes-Qu-Vanstone (MQV) [60], and the TLS handshake. For many of the protocols, at least two variants are described. Thus, analysing these two parts is a significant undertaking.

The main idea behind participating in key management protocol executions is to share a secret with others in order to communicate securely with them. The shared secret can be used for symmetric encryption and, if this usage is correct, ensure confidentiality and integrity of the encrypted data. We give a toy example of a key management protocol in Figure 4.1. First, A sends to B a message encrypted with the public key of B . The payload of the message consists of A 's identity and a freshly generated value n . After B receives and decrypts this message with the help of its corresponding private key, A and B share the value n . Now B sends to A a message symmetrically encrypted with n used as a key. The secrecy of n and m depends on the kind of encryption algorithms A and B use and the kind of attackers they might face.

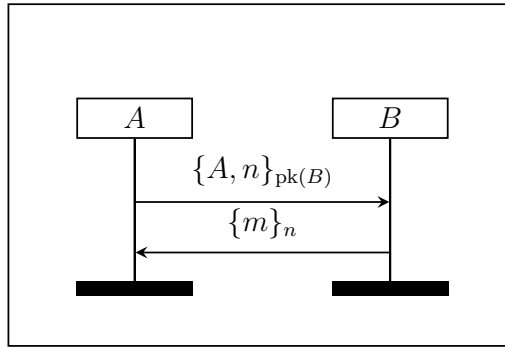


Figure 4.1: Key management protocol example

In positive contrast to other security protocol standards [9], the ISO/IEC 11770 standard explicitly specifies security properties for each of its protocols. Two of these properties are structural properties, i.e. key control and replay detection. Additionally, there are four security properties that relate to active adversaries, namely key authentication, key confirmation, entity authentication, and forward secrecy.

We use tool-supported formal methods to determine if the protocols indeed satisfy their claimed non-structural security properties, building on and extending abstract protocol models originally developed for an earlier analysis of ISO/IEC 11770 by Schmid [93]. Moreover, we analyse the protocols for modern key exchange security properties, such as resilience against Key Compromise Impersonation (KCI) and Unknown Key Share (UKS) attacks.

We provided a version of the publication [26] that this chapter was based on to the ISO/IEC working group responsible for the 11770 standard. Our protocol models and tools used are available for download from [45].

Overview. We give some background on ISO/IEC 11770 and illustrate some of its protocols in Section 4.2. We describe our analysis approach in Section 4.3 and present the results in Section 4.4. We provide concrete recommendations for improving the standard in Section 4.5 and conclude in Section 4.6.

4.2 Background on ISO/IEC 11770

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

The ISO/IEC 11770 standard is divided into five parts as of September 2015. Part 1 was originally released in 1996 and has been updated over the years. It describes the context and framework. Parts 2 and 3 describe mechanisms based on symmetric and asymmetric techniques. Part 4 describes mechanisms based on weak secrets, such as password-based key exchange protocols. Part 5 describes group key management mechanisms. The standard is expected to be extended with a sixth part on key derivation functions.

4.2.1 Protocols

We focus on Part 2 [46] and Part 3 [47] of the ISO/IEC 11770 standard. Part 2 describes 13 key establishment mechanisms. Part 3 describes 11 key agreement mechanisms, 6 key transport mechanisms, and 3 public key transport mechanisms. Many of these 33 mechanisms have optional message fields and message flows, giving rise to a large number of variants.

Additionally, the mechanisms produce keying material that must be used with a key derivation function to form an encryption key for further messages. The standard does not specify a single key derivation function; instead, it gives examples of various possible key derivation functions. Thus, using a single mechanism with different key derivation functions can be regarded as having multiple variants of the same base

mechanism. As we will see in Section 4.4.4, the choice of a key derivation function can influence the security of a mechanism.

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

We next describe two protocols from the standard. This enables us to provide an indication of the type of protocols contained in both Part 2 and Part 3 of the standard, as well as introduce additional notation.

Key Establishment Mechanism 12 (2-12). We give an example of a protocol described in Part 2 [46], referenced in the standard in clause 7.2 as Key Establishment Mechanism 12. The protocol is stated to be derived from, but not fully compatible with, the four-pass mutual authentication mechanism specified in ISO/IEC 9798-2 [46]. The protocol has several variants. For this example, we consider the variant with all optional parts included, depicted using a Message Sequence Chart (MSC) in Figure 4.2. In the figure, T_A/N_A is either a time stamp T_A or sequence number N_A of entity A . I_A and I_B respectively identify entities A and B . $e_K(m)$ denotes the encryption of the message m with the key K . The protocol assumes that entities A and B respectively share long-term symmetric keys K_{AP} and K_{BP} with a trusted third party P . $Text_1$ through $Text_5$ are text fields whose contents are not specified by the standard. F denotes keying material.

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

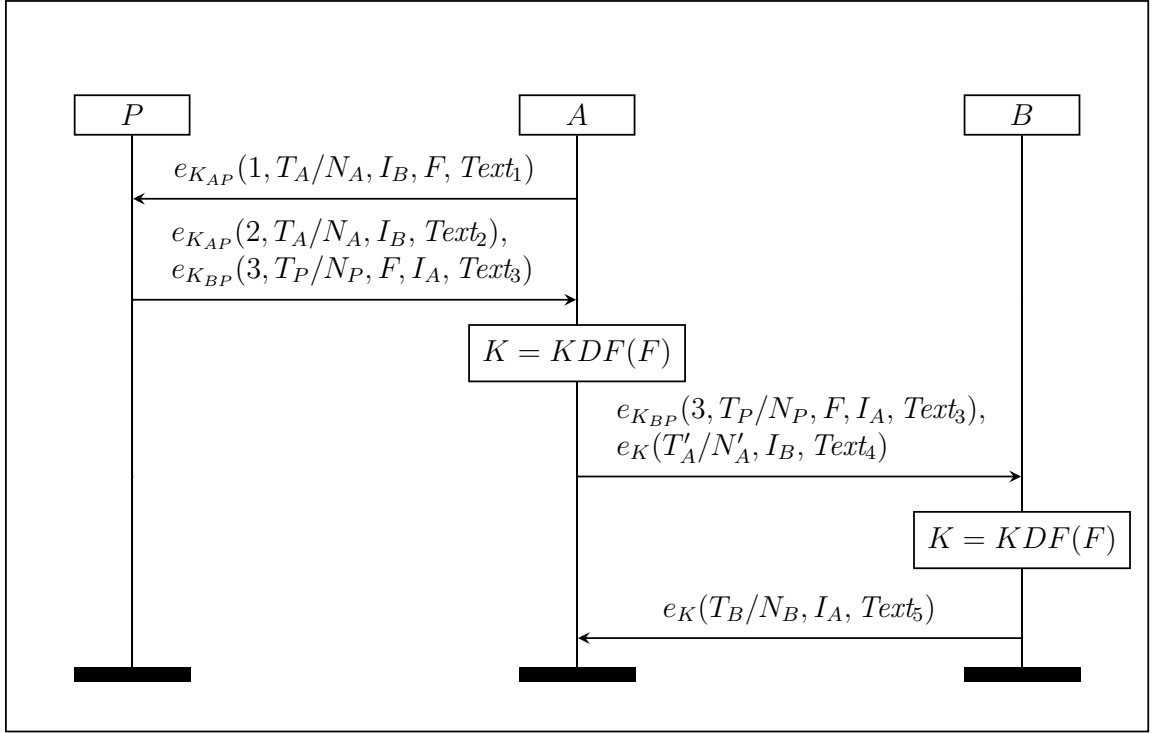


Figure 4.2: Protocol 2-12 with optional parts

Notation	Meaning
T_A	time stamp of entity A
N_A	sequence number of A
I_A	identifier of entity A
$e_K(m)$	encryption of m with K
K_{AB}	key shared by A, B
$Text_i$	text field
F	keying material

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

Key Agreement Mechanism 11 (3-KA-11). Key Agreement Mechanism 11 from Part 3, shown in Figure 4.3, establishes a key shared by entities A and B . First, A generates a random value r_A and sends it to B . B responds with its own random value r_B and its certificate. Upon receiving this message, A generates a new random value r'_A . r'_A is used with the other two random values to derive a session key K . Then r'_A is encrypted using B 's public key, and sent to B along with a message authentication code (MAC) keyed with K that includes the earlier randomness r_A . B decrypts the message, computes K , and checks the MAC. B then responds with its

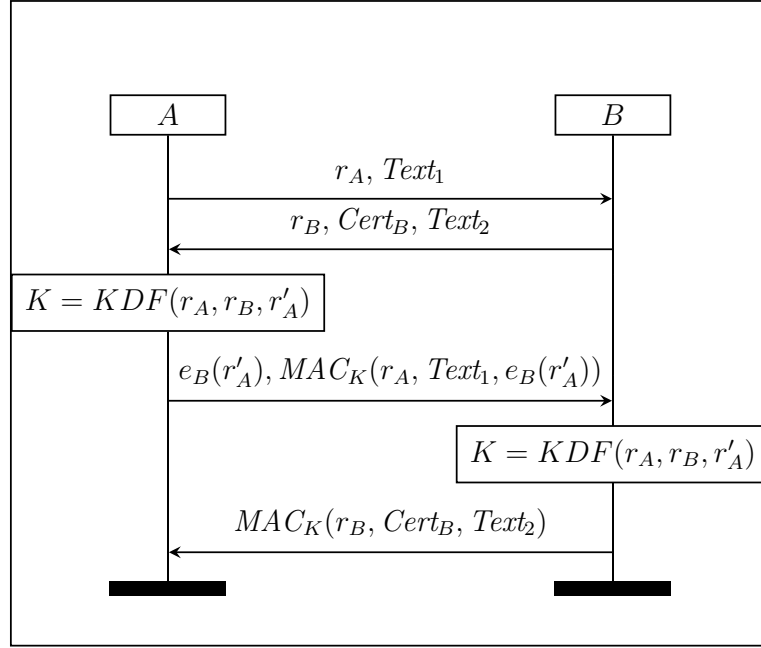


Figure 4.3: Protocol 3-KA-11

own MAC of r_B and its certificate.

According to the standard, this protocol is derived from the TLS Handshake Protocol [28]. In particular, since only B uses his private key (to decrypt the message) and the random values are directly input to the key derivation function, the protocol resembles the unilaterally authenticated RSA mode of TLS, where A corresponds to the client and B to the server. The random value r'_A in 3-KA-11 plays the same role as the TLS *premaster secret* and the two text fields are used in TLS for the cipher suite negotiation.

4.2.2 Security properties and threat model of the standard

Most standards for security protocols do not specify threat models or intended security properties [9]. In this respect, ISO/IEC 11770 is an exception since it explicitly specifies a set of security properties, and states for each protocol which of these properties it satisfies. ISO/IEC 11770 defines the following properties [46, 47]:

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

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

Explicit key authentication from entity A to entity B : Assurance for entity B that A is the only other entity that is in possession of the correct key.¹

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

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

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

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

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

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

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

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

¹The standard notes that “Implicit key authentication from A to B and key confirmation from A to B together imply explicit key authentication from A to B ” [46, p. 2]. One might expect also that explicit key authentication implies the other two properties, but the standard does not state this.

Compromising long-term private keys: Forward secrecy is claimed for some protocols. The adversary can only violate forward secrecy by compromising the long-term private keys of some entities.

4.3 Formally modelling the protocols and properties

We analyse all 33 protocols specified in Part 2 and Part 3 of the standard, along with their described variants, by using formal methods. In particular, we use the Scyther framework [27] for the automatic symbolic analysis of security protocols. Scyther [23] has built-in support for compromising adversaries [6], including support for the analysis of (weak) Perfect Forward Secrecy, resilience against KCI attacks, and finding Unknown Key Share attacks. It is therefore especially suitable for analysing security notions common in the domain of protocols for key agreement, establishment, and transport. Another feature of Scyther which is very helpful in our research is its option-packed back end, which allows for a conveniently scriptable analysis of whole classes of security protocols.

4.3.1 Protocol specification

We briefly recall the Scyther framework detailed in Chapter 3. Within the framework, protocols are specified using so-called role scripts. A protocol can have any finite number of roles, and is run by entities who execute those roles. Entities may execute each role multiple times, and every role can be executed by any entity. We call each such role instance a session. We assume that, prior to protocol execution, every entity has generated or securely received a long-term asymmetric key pair consisting of a public and a private key, it has authentic and secret copies of all its long-term symmetric keys shared with other entities, and authentic copies of the public keys of all other entities.

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

```

1 option "--partner-definition=2";
2
3 usertype KeyingMaterial;
4 usertype String;
5 usertype Integer;
6
7 hashfunction KDF;
8 const N1,N2,N3: Integer;
9
10 macro key = KDF(F);
11 macro sid = (A,B,key);
12
13 protocol 2-12-withOptional(A,B,P)
14 {
15   role A
16   {
17     fresh TNA,TNA2: Nonce;
18     fresh F: KeyingMaterial;
19     fresh Text1,Text4: String;
20     var Text2,Text5: String;
21     var T: Ticket;
22     var TNB: Nonce;
23
24     claim(A,SID,sid);
25     claim(A,Running,B,key);
26     send_1(A,P,{N1,TNA,B,F,Text1}k(A,P));
27     recv_2(P,A,{N2,TNA,B,Text2}k(A,P),T);
28     send_3(A,B,T,{TNA2,B,Text4}key);
29     recv_4(B,A,{TNB,A,Text5}key);
30
31     claim(A,SKR,key);
32     claim(A,Commit,B,key);
33     claim(A,Alive,B);
34   }
35
36   role B
37   {
38     var TNP,TNA2: Nonce;
39     var F: KeyingMaterial;
40     var Text3,Text4: String;
41     fresh TNB: Nonce;
42     fresh Text5: String;
43
44     recv_3(A,B,{N3,TNP,F,A,Text3}k(B,P),
45           {TNA2,B,Text4}key);
46     claim(B,SID,sid);
47     claim(B,Running,A,key);
48     send_4(B,A,{TNB,A,Text5}key);
49
50     claim(B,SKR,key);
51     claim(B,Commit,A,key);
52     claim(B,Alive,A);
53   }
54
55   role P
56   {
57     var TNA: Nonce;
58     var F: KeyingMaterial;
59     var Text1: String;
60     fresh Text2: String;
61     fresh TNP: Nonce;
62     fresh Text3: String;
63
64     claim(P,SID,P);
65     recv_1(A,P,{N1,TNA,B,F,Text1}k(A,P));
66     send_2(P,A,{N2,TNA,B,Text2}k(A,P),
67           {N3,TNP,F,A,Text3}k(B,P));
68   }
69 }

```

Figure 4.4: Scyther input file for 2-12 with confirmation messages and claimed properties.

For example, in Figure 4.4, we give the input for Scyther that encodes protocol 2-12 described in Section 4.2.1. Send, receive and claim events are respectively specified with `send`, `recv`, and `claim`. Freshly generated values are declared with `fresh`, variables with `var`, user-defined types with `usertype`, and hash functions with `hashfunction`. Every function, constant, fresh value, and variable can have a different type, such as `Nonce` or a user-defined type such as `Integer`, `KeyingMaterial`, or `String`—the types are used to restrict the pattern matching in the execution of a receive event. The keyword `macro` can be used to define shorthands.

4.3.2 Specifying security properties

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

Implicit key authentication. According to the standard, implicit key authentication requires that if an entity A uses a protocol to establish a key K with entity B , then only A and B can learn the key. We model this by analysing the secrecy of K whilst allowing the adversary to impersonate any entity except for A and B . The possibility of impersonation is modelled by allowing the adversary to learn the long-term private key of any entity except for A and B .

Key confirmation. This property corresponds to one of the authentication properties in Lowe’s hierarchy [63]. In particular, key confirmation from A to B corresponds to non-injective data agreement on the key, which we model with two claims: a running claim in the A role and a commit claim in the B role. If the commit claim is executed, we require that the corresponding running claim is executed as well: it must have the entities in reverse order, and the same contents (the entities are said to *agree* on the contents). It is called *non-injective* data agreement because replays are not considered.

Explicit key authentication. In contrast to implicit key authentication, explicit key authentication additionally requires that entities in fact compute the key. We model this by the previously defined key confirmation.

Recall that the standard only states that implicit key authentication and key confirmation imply explicit key authentication [46, p. 2], but not necessarily the other way around. Thus, if the standard is taken literally, this suggests that there exist protocols that offer explicit key authentication but that do not satisfy implicit key authentication or key confirmation. We attempted to find a formal interpretation of these concepts that makes the suggestion true, but failed. Thus, we define explicit key authentication as the conjunction of key confirmation and implicit key authentication.

Entity authentication. Entity authentication of A to B corresponds to aliveness [63]: an Alive claim of A is placed in the specification of role B . Whenever the claim is executed, the entity assumed to be performing the A role is required to have executed some event.

Forward secrecy. There are several definitions of forward secrecy in the literature, and it is not clear from the standard which property is intended. The *mutual forward secrecy* (*MFS*) notion from the standard seems to be closest to two common formal definitions. *Weak Perfect Forward Secrecy* (*wPFS*) [6,24,52] requires that the adversary does not actively tamper with the session that it attacks, e.g. by injecting messages. In contrast, (*strong*) *Perfect Forward Secrecy* (*PFS*) [29] allows the adversary to actively interfere with the messages received by the session under attack. Scyther directly

supports checking both properties through its support of the `LKRaftercorrect` and `LKRafter` rules [6]. Our analysis reveals that the majority of protocols for which MFS is claimed in fact only achieve wPFS, and we therefore interpret MFS as wPFS. The internal Scyther definition of wPFS is given in [6].

Key compromise impersonation (KCI). Another desirable property of key exchange protocols is resilience to KCI attacks [13], in which the adversary exploits its knowledge of the long-term private key of Alice to impersonate any entity in later communication with Alice. This property is modelled in Scyther by a session key secrecy claim of an entity whose long-term private keys the adversary is allowed to reveal.

Unknown key share (UKS). Unknown key share attacks are attacks in which only Alice and Bob know the session key K ; however, Alice and Bob disagree on who they share K with [14]. For example, Alice correctly thinks K is shared with Bob, but Bob might think that K is shared with Charlie. Even though the adversary does not learn the key in such attacks, using the key is not sufficient to authenticate subsequent messages: if Alice sends a message encrypted with K or accompanied by a MAC keyed by K , Bob will assume that the message came from Charlie. Similarly, Bob will send messages intended for Charlie that will be received by Alice.

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

We enable session identifier (SID) support in Scyther input files with the line option `--partner-definition=2`; and we specify SIDs for all role instances by annotating each role with SID claims. For example, in Figure 4.4 we enable the manual specification of a partner session on line 1, define the session identifier on line 11, and insert it into role specifications on lines 24 and 45. When session key secrecy is analysed for a session s , and the Scyther SKR adversary rule (Session Key Reveal) is enabled in the GUI or `--SKR=1` is provided as a command-line option, the adversary is able to obtain the session keys computed by any session whose identifier differs from that of s .

Protocol	Optional fields	entity authentication		implicit key authentication	KCI resilience	key confirmation/ explicit key authentication	
		of A	of B			of A	of B
2-1		×		✓	×	×	
2-2		×		✓	×	×	
2-3		✓		✓	×	✓	
2-3	all	✓		✓	×	✓	
2-4		✓		✓	×	✓	
2-4	all	✓		✓	×	✓	
2-5	all	✓	✓	✓	×	×	×
2-6	Fa,Fb	×	×	✓	×	×	×
2-6	Fb	×	×	✓	×	×	×
2-6	B,Fb	✓	✓	✓	×	✓	✓
2-6	B,Fa	✓	✓	✓	×	✓	×
2-6	all	✓	✓	✓	×	✓	✓
2-7		×	×	✓	×	×	×
2-8	N2	×	×	✓	×	×	×
2-8	N2,MACN3	✓	✓	✓	×	×	×
2-9		×	×	✓	×	×	×
2-9	all	×	×	✓	×	×	×
2-10		×	×	✓	×		
2-11		×	×	✓	×		
2-12		×		✓	×	×	
2-12	all	×	✓	✓	×	×	×
2-13		×		✓	×	×	
2-13	all	×	×	✓	×	×	×

Table 4.1: Main formal analysis results for key establishment protocols from Part 2

4.4 Results of the formal analysis

For the protocols in Part 2 and Part 3 of the standard, we model the claimed security properties as well as some additional properties that serve to sanity-check the models. We also consider KCI and UKS attacks.

4.4.1 Main analysis results

We analyse each of the resulting models in the default Scyther setting. This standard setting covers a wide range of scenarios, some of which may not apply to all real-world implementations. We return to this below. The results are displayed in Tables 4.1–4.4. We use a cross (×) to denote that an attack is found, and a check (✓) to denote that no attacks are found. These tables are automatically generated using a script that uses Scyther as a back end.

In the case that we find attacks for a specific protocol and property, we use further automated analysis to narrow down the scenarios in which the protocol is vulnerable. In particular, we consider the following aspects of an attack:

Protocol	Optional fields	entity authentication		implicit key authentication		KCI resilience		key confirmation/ explicit key authentication		PFS		weak PFS	
		of A	of B	of A	of B	for A	for B	of A	of B	for A	for B	for A	for B
3-KA-1				✓	✓	×	×			×	×	×	×
3-KA-2				×	✓	✓	×			×	×	×	×
3-KA-3	TVP	✓		✓	✓	✓	×	✓		✓	×	✓	✓
3-KA-3		✓		✓	✓	✓	×	✓		✓	×	✓	✓
3-KA-4			×	×	×	×	×	×	×	×	×	×	×
3-KA-5	MAC		✓	✓	✓	✓	✓	×	✓	✓	×	✓	✓
3-KA-6	MAC	×	✓	✓	✓	×	✓	×	✓	×	×	×	✓
3-KA-6		×	✓	✓	✓	×	✓	×	✓	×	×	×	✓
3-KA-7		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3-KA-8				✓	✓	✓	×			✓	×	✓	×
3-KA-9		×	×	✓	✓	✓	✓	×	×	×	×	✓	✓
3-KA-10		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3-KA-11		×	✓	×	✓	✓	×	×	×	×	×	×	×

Table 4.2: Main formal analysis results for key agreement protocols from Part 3

Protocol	Optional fields	entity authentication		implicit key authentication		KCI resilience		key confirmation/ explicit key authentication		PFS	weak PFS	
		of A	of B	of A	of B	for A	for B	of A	of B		for A	for B
3-KT-1		×	×	×	✓	✓	×	×		×	×	×
3-KT-1	all	×	×	×	✓	✓	×	×		×	×	×
3-KT-2		✓	×	✓	✓	✓	×	×		×	×	×
3-KT-2	all	✓	×	✓	✓	✓	×	✓		×	×	×
3-KT-3		✓	×	✓	✓	✓	×	✓		×	×	×
3-KT-3	all	✓	×	✓	✓	✓	×	✓		×	×	×
3-KT-4	all	×	✓	✓	✓	×	✓	×	✓	×	×	✓
3-KT-5	MACN3	✓	✓	✓	✓	×	×	✓	✓	×	✓	✓
3-KT-6	noText	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×
3-KT-6		✓	✓	✓	✓	✓	✓	✓	✓	×	✓	×

Table 4.3: Main formal analysis results for key transport protocols from Part 3

Protocol	Optional fields	entity authentication		implicit key authentication		KCI resilience		key confirmation/ explicit key authentication		PFS		weak PFS	
		of A	of B	of A	of B	for A	for B	of A	of B	for A	for B	for A	for B
3-PKT-1		✓						✓					
3-PKT-2	N3 sign.	✓						✓					
3-PKT-2		✓						✓					
3-PKT-3								✓					

Table 4.4: Main formal analysis results for public key transport protocols from Part 3

Agents in multiple roles. Some attacks require that an agent performs multiple roles. This behaviour is allowed by our formal models and in many application scenarios. However, there are application scenarios in which each agent performs only one role, and where such an attack would not apply.

Alice-talks-to-Alice. Some attacks require that an agent starts a role with itself in

one of the other roles. This occurs, for example, in many implementations of the Kerberos protocol. However, such attacks would not work on all implementations.

In the tables, crosses imply that we find attacks that might rely on these non-standard, subtle requirements on attack scenarios. On the other hand, if no attacks are found, we consider the following strengthening of our threat model and include its impact on the standard in Table 4.5:

Type flaw. We say an attack requires a type flaw if it depends on an agent misinterpreting a term as a term of another type. For example, an agent may misinterpret an agent name as a nonce.

Note that not all crosses in the tables imply a serious flaw in the protocol. Rather, they indicate that a different protocol could have achieved these properties, perhaps at a reduced efficiency. Also note that we model some properties beyond those claimed in the standard. We analyse the exact discrepancies between our results and the claims in the standard in Section 4.4.2, where we also return to the implementation scenario assumptions required for the attacks.

Finally, we manually inspect the attack graphs generated by Scyther to sanity-check the results and to understand which aspects of the design of a protocol make it vulnerable.

4.4.2 Implications for properties claimed in the standard

We give an overview of the properties claimed in the standard in Table 4.5. The contents of this table are directly taken from the tables in [46, 47], with the difference that we add notes and use red and bold to mark incorrect statements, based on our formal analysis results. We classify the incorrect claims in the standard into five categories AT1...AT5, which we describe below. Note that the table in [46] only has a key authentication column with “yes” or “no” in the cells. This information has to be combined with NOTE 2 [46], which states that all protocols in Part 2 achieve implicit key authentication, and that “yes” is to be interpreted as explicit key authentication.

The standard provides a reasonable level of detail in its specification of security properties and assumptions, but does not provide sufficient detail to unambiguously construct a formal model. We have therefore chosen to focus on positively claimed properties and use the formal analysis to construct counterexamples in the form of attacks. The benefit of this approach is that we can often exhibit straightforward

Table 4.5: **Claimed properties** Security properties claimed for the protocols in Parts 2 and 3 of the standard. Our analysis reveals that some claims are incorrect, and we mark them using bold and red.

Protocol in Part 2	Key Auth.	Key Conf.	Entity Auth.
2-1	implicit	no	no
2-2	implicit	no	no
2-3	explicit	no	A
2-4	explicit	no	A
2-5	explicit	no	A & B
2-6	explicit	no	A & B
2-7	implicit	no	no
2-8	explicit (AT1)	opt. (AT1)	opt. (AT1)
2-9	explicit (AT1)	opt. (AT1)	opt. (AT1)
2-10	explicit	no	no
2-11	explicit (AT4)	no	no
2-12	explicit (AT1)	opt. (AT1)	opt. (AT1)
2-13	explicit (AT1)	opt. (AT1)	opt. (AT1)

Protocol in Part 3	Implicit Key Auth.	Key Conf.	Entity Auth.	Forward Secrecy
3-KA-1	A,B	no	no	no
3-KA-2	B	no	no	A
3-KA-3	A,B	B	A	A
3-KA-4	no	no	no	MFS
3-KA-5	A,B	opt	no	A,B
3-KA-6	A,B	opt	B	B
3-KA-7	A,B	A,B	A,B	MFS
3-KA-8	A,B	no	no	A
3-KA-9	A,B	no	no	MFS
3-KA-10	A,B	A,B	A,B	MFS
3-KA-11	A, B (AT2)	A, B (AT2)	B	MFS (AT3)
3-KT-1	B	no	no	A
3-KT-2	B	B	A	A
3-KT-3	B	B	A	A
3-KT-4	A	A	B	B
3-KT-5	A,B	(A),B	A,B	no
3-KT-6	A,B	A,B (AT5)	A,B	no

attacks without having to argue about the full details of the assumed threat models, protocol execution model, and modelling properties.

The drawback is that we cannot provide conclusive statements about other oddities in the standard. For example, according to the standard, protocol 2-5 has a “no” for key confirmation, but it has a “yes” for explicit key authentication. This seems to contradict the informal definitions of these notions in the standard: from their definitions, one would expect protocols with explicit key authentication to also satisfy key confirmation. Future versions of the standard would benefit from a clarification of the exact relations between these properties.

AT1: Entity authentication failures for 2-8, 2-9, 2-12, and 2-13. We find several possible entity authentication failures for protocols in Part 2 that are derived from protocols in an earlier version of the ISO/IEC 9798-2 standard for entity authentication [46].

These attacks are closely related to the attacks on the corresponding protocols from the 9798 standard as presented in [8]. The attacks work in all implementations where a single entity can perform not only the role of the trusted third party, but also another role. The adversary can then cause A to complete the protocol, apparently with B , even though B is not present. Thus, the attacks violate even the weakest form of entity authentication.

We show an example of such an attack on protocol 2-12 in Figure 4.5. It depends on the fact that the entity running role A does not check the contents of the message encrypted for entities running roles B and P . In fact, normally such a check is impossible because all three roles are run by different entities. Seeing the payload of that particular message would be the only way for the honest entity Pete to detect that something is wrong: he could see that the message contains I_{Alice} where I_{Pete} should be. Since he does not see the payload, he gladly confirms the session key to Bob in role B , who falsely thinks that Alice just confirmed it.

Fixes for these protocols have been proposed in [8], and they have been integrated into the ISO/IEC 9798 standard. As a result, these attacks no longer work on ISO/IEC 9798. However, no changes have been made to the derived protocols in ISO/IEC 11770, so they are still vulnerable to similar attacks.

AT2: 3-KA-11 key authentication/confirmation failure for B. According to the standard, this mechanism (depicted in Figure 4.3) offers mutual explicit key authentication and mutual key confirmation. However, as stated earlier, 3-KA-11 is derived from the unilaterally authenticated RSA mode of TLS [28]. In this mode, the server cannot be certain whether the client is who it claims to be. The same issue occurs for the B role of 3-KA-11.

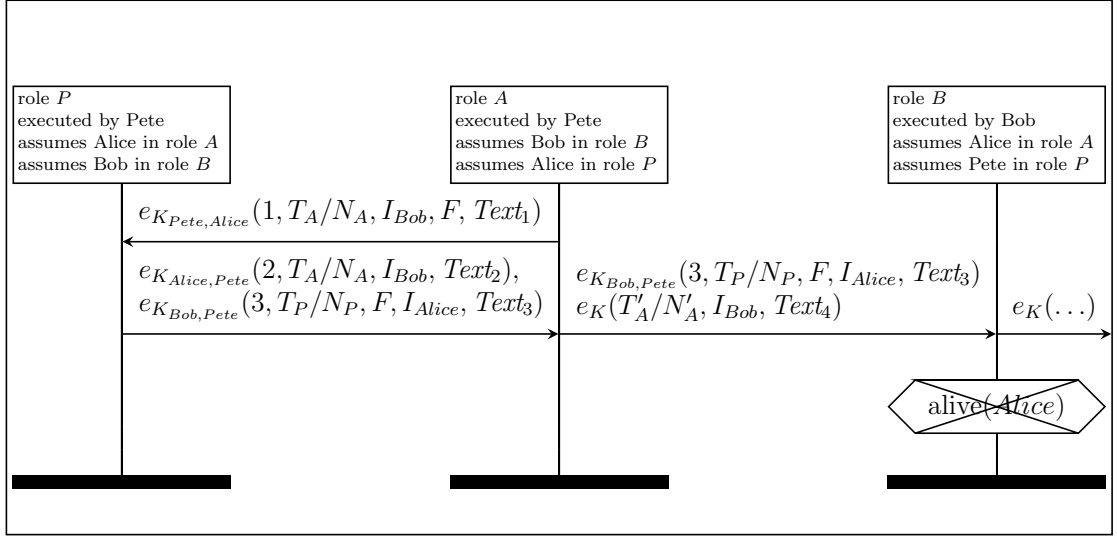


Figure 4.5: Entity authentication attack on protocol 2-12 with optional parts

Consequently, there is an attack on entity authentication on the B role that violates both of the claimed properties. In the attack, the adversary performs the A role, pretending to be Alice, and sends messages to Bob in the B role. Because executing the A role does not require the use of any long-term secrets, the adversary can simply claim to be anybody. The entity performing the B role therefore cannot obtain any authentication guarantees about its communication partner or ascertain the secrecy of the key.

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

AT4: Failure of key authentication for 2-11. The 2-11 protocol, which is depicted in Figure 4.6, assumes pre-shared symmetric keys and a trusted third party P . In a regular execution of the protocol, A sends a request to P for a ticket to forward to B . The request is a triple $(I_B, F, Text_1)$, which contains keying material F generated by A and is encrypted with the key shared between A and P . P then returns the triple $(F, I_A, Text_2)$ encrypted with the key shared between B and P , which A forwards to B .

Depending on the implementation, it may be possible for an agent to misinterpret an agent identity as (random) keying material, for example if both are of the same bit length. If an implementation of 2-11 cannot tell the difference between these, it can

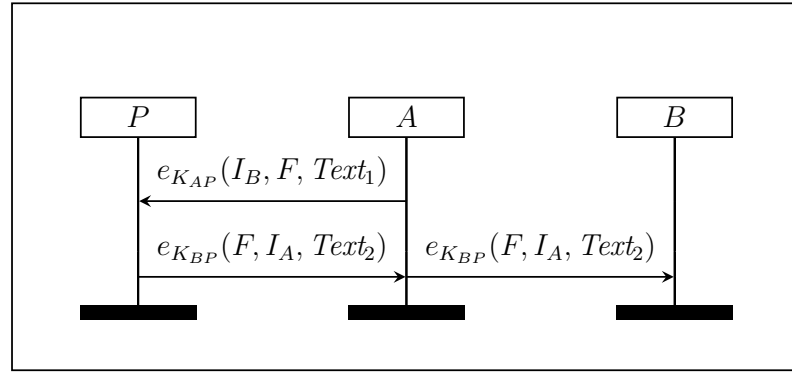


Figure 4.6: Protocol 2-11

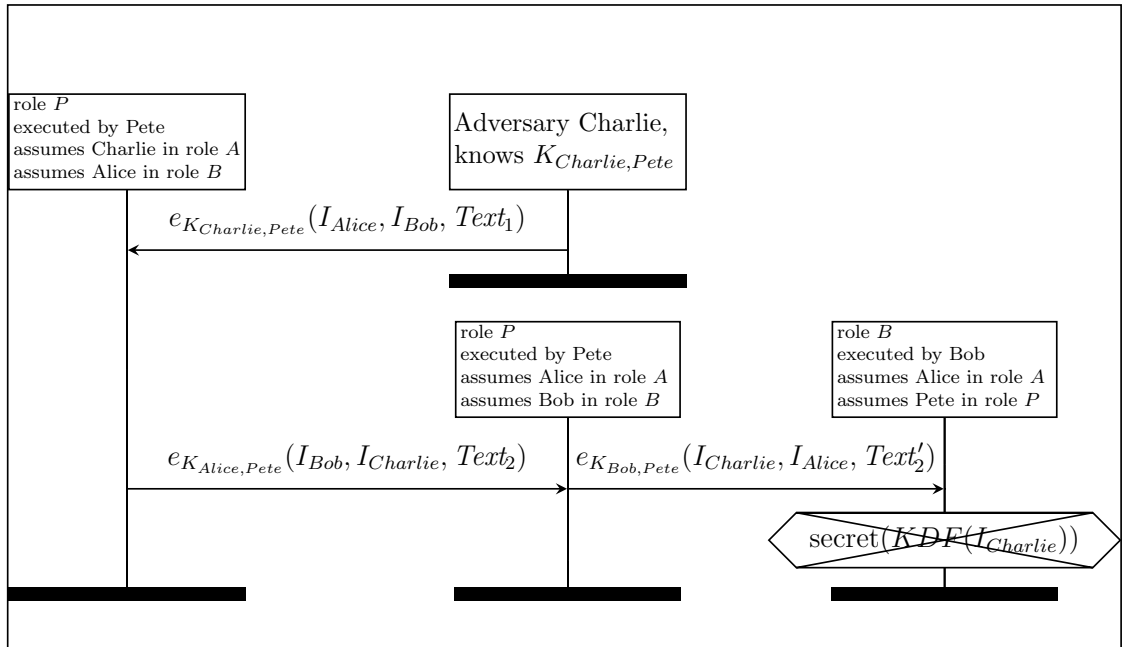


Figure 4.7: Protocol 2-11 key authentication attack

be vulnerable to a type-flaw attack on key authentication. The attack can be seen in Figure 4.7.

The adversary Charlie can attack a session in which Bob assumes to be talking to Alice, even though Alice and Bob are not compromised. Charlie encrypts a message for the trusted third party Pete, requesting a key for Alice. However, instead of generating new keying material F , Charlie instead includes Bob's identity in the keying material field. Pete's response therefore is the triple $(I_{Bob}, I_{Charlie}, Text_2)$ encrypted with $K_{Alice,Pete}$. Charlie resends this message to Pete. There is nothing in the standard that prevents Pete from accepting this message as a valid request. Now, Pete responds with the triple $(I_{Charlie}, I_{Alice}, Text'_2)$ encrypted with $K_{Bob,Pete}$. If Bob receives this

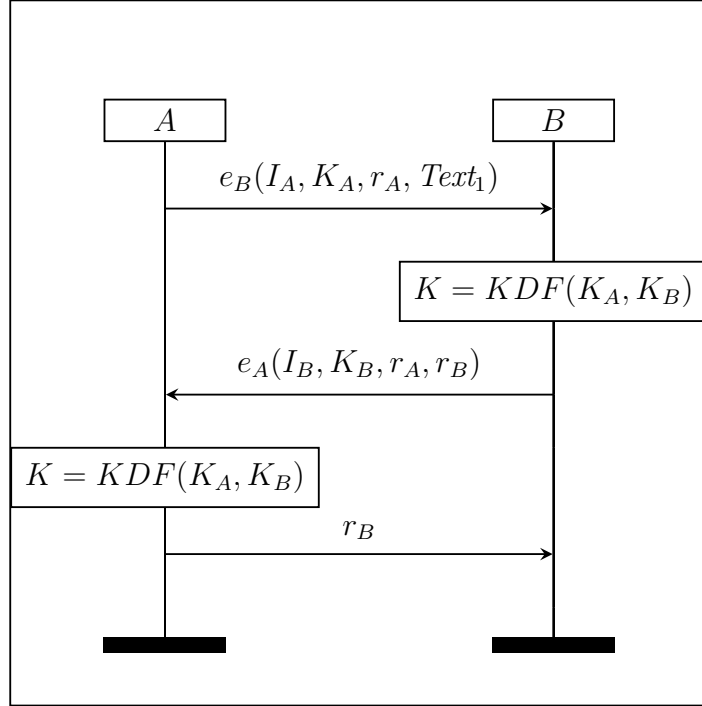


Figure 4.8: Protocol 3-KT-6 combined key variant with $Text_1$ optional field

message, he will assume that it is a valid message and that I_{Charlie} is secure keying material for communicating with Alice. The adversary can then compute the session key that Bob computes.

AT5: Failure of key confirmation for 3-KT-6. The 3-KT-6 protocol is a three-pass protocol that transfers two secret keys, K_A and K_B . After the exchange, a session key can be computed from either or both of these keys. There are five text fields designated as optional in the protocol specification. We choose to depict a simple implementation with only $Text_1$ enabled in Figure 4.8.

A complex attack is possible on some implementations of this protocol. There are three preconditions for the attack, which will not be met by most implementations. However, there is nothing in the standard that ensures that they are not met. The first precondition is that the $Text_1$ field is implemented, and $Text_3$ is not implemented. Second, fresh values must be acceptable values for the $Text_1$ field. Third, entities must be able to perform both the A and the B role of the protocol.

If an implementation meets these conditions, the adversary can attack an instance of the A role by exploiting three instances of the B role. We give a graphical representation of such an attack in Figure 4.9. The adversary redirects each sent message into the first receive of a new instance of the B role, and the entity assumptions

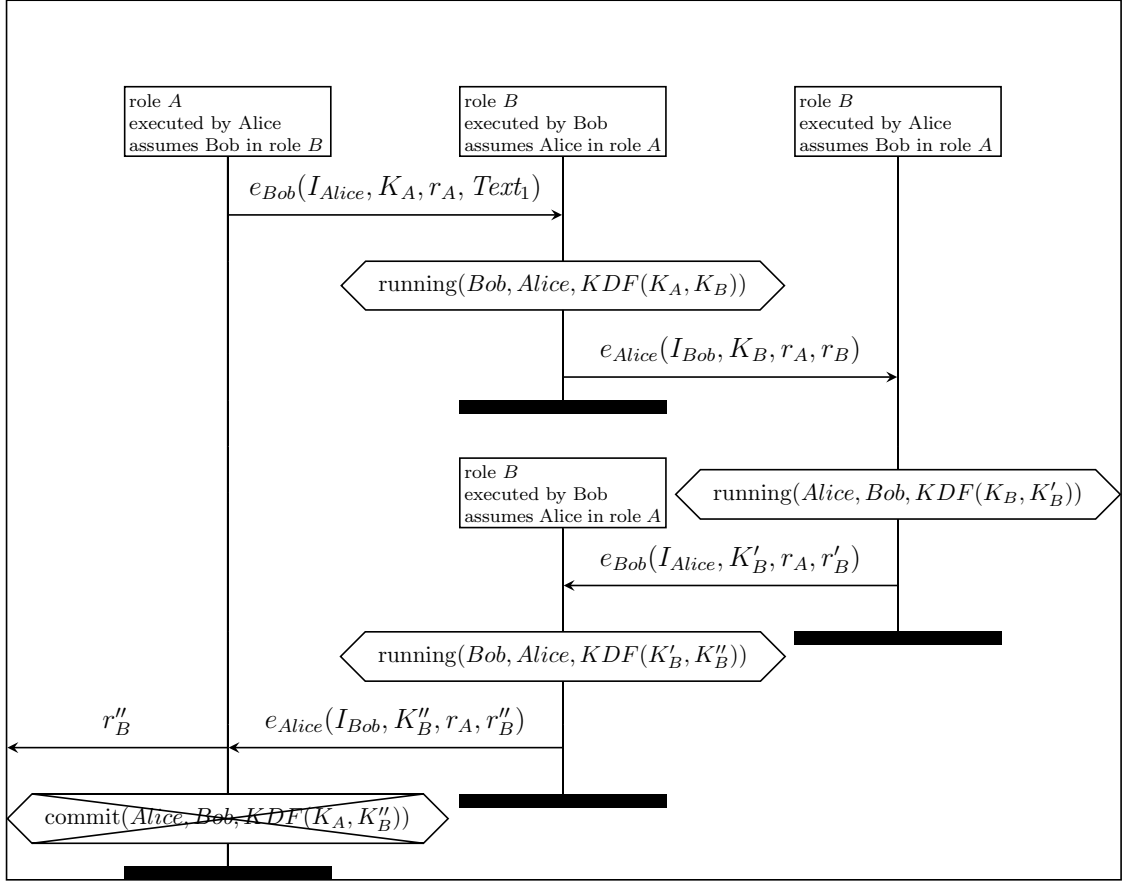


Figure 4.9: 3-KT-6 combined key variant with $Text_1$ optional field key confirmation attack

for the next instance of B are swapped. This is possible since entities can perform multiple roles, and enabled by the fact that the fresh values in messages sent by instances of the B role can be accepted into the $Text_1$ field. After three instances of the B role, the final message is then rerouted back to the final receive of the A instance. Consequently, there is no instance of B that agrees with the A instance on *both* K_A and K''_B . Thus, when the session key is computed from both of these keys, key confirmation fails for the instance of A .

4.4.3 Key Compromise Impersonation (KCI) results

All of the protocols in Part 2 use symmetric cryptography and hashing only. Hence, they are necessarily vulnerable to KCI attacks, which is implied by the impossibility result from Section 3.4. All the modelled security properties of key transport protocols from Part 3 that are not satisfied can be violated even without considering KCI. In

some sense, we can consider all these attacks to be false positives of KCI attacks (cf. Chapter 3.2), which is the view we adopt in the continuation of our KCI discussion.

The automatic analysis shows that four of the eleven key agreement protocols in Part 3 are vulnerable to KCI attacks: 3-KA-1, 3-KA-3, 3-KA-6, and 3-KA-8. Mechanisms 3-KA-1 and 3-KA-3 are variants of the unsigned Diffie-Hellman protocol. Mechanism 3-KA-1 is the static Diffie-Hellman protocol, so as expected the session key is not secret when the adversary knows one of the static keys. Similarly, 3-KA-3 is a one-pass Diffie-Hellman variant where A 's ephemeral and B 's static half keys (cf. Section 3.5.4) are used: if the adversary gets B 's static private key, it can use A 's half key to infer the session key.

In 3-KA-6, the fact that the input to the key derivation function is only protected by the public key of A allows an adversary who knows A 's private key to impersonate B in subsequent communications with A that are only protected by the established session key.

Lastly, 3-KA-8 is a one-pass mechanism derived from the one-pass variant of the MQV protocol [60]. It uses an elliptic curve agreed upon by entities A and B to establish a shared secret key. A base point G of prime order and a function π that assigns integer representations to points on the elliptic curve also needs to be prearranged. As shown in Figure 4.10, A generates an ephemeral secret r_A and uses it to transmit to B a public value, modelled as a point $r_A G$ on the elliptic curve. B then computes a fresh session key K from its static private key h_B and the public values of A , the latter of which include $P_A = h_A G$ (similarly, A uses its secrets and B 's public values). The adversary can use B 's private key to infer the session key computed by B without tampering with the message which B gets from A .

None of the KCI attacks that we find in this set of protocols require the adversary to use the actor's key to interfere before the attacked session ends. As a result, the attacker can delay the use of the actor's key until after the attacked session ends, and then use it to compute the session key. This means that the KCI attacks we find can also be regarded as attacks on wPFS (and thus PFS). In other words, all protocols in the standard that satisfy wPFS or PFS, also satisfy KCI resilience.

This observation about the standard may lead to the hypothesis that *all* protocols that satisfy PFS are also KCI resilient. However, this is not the case. For example, consider the 3-message version of the Unified Model (UM) protocol as described in [71]. This protocol is standardised in the NIST standard SP 800-56A [5]. The protocol is based on Diffie-Hellman (DH): the session key derivation includes the ephemeral DH key (g^{xy}) based on the exchanged DH values g^x and g^y . As shown in [71], this helps

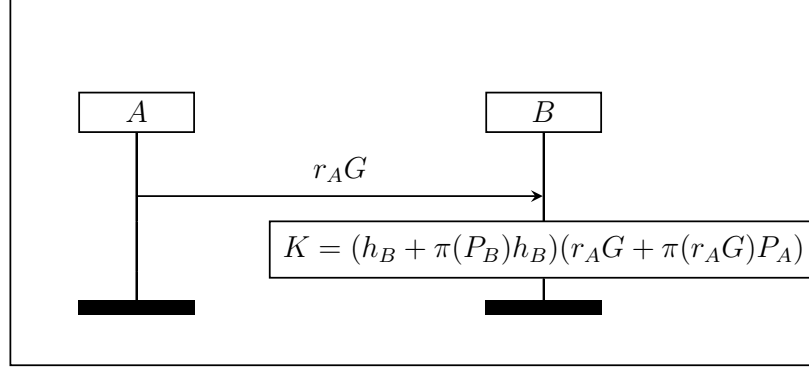


Figure 4.10: Protocol 3-KA-8

to ensure that the protocol satisfies PFS. However, it is not KCI resilient because the exchanged DH values are authenticated by a MAC whose key depends critically on the static DH key g^{ab} . If the adversary obtains the actor's long-term key, it can compute g^{ab} and authenticate DH values of its choosing. Inserting a message that contains a DH value g^z , where z is known to the adversary, leads to a KCI attack. Note that [71] explicitly excludes KCI resilience from its adversary model. Thus, this protocol proves that PFS does not imply KCI resilience in general.

However, for all the protocols in the ISO/IEC 11770 standard, PFS does imply KCI resilience. Hence, we can replace each protocol vulnerable to KCI attacks with one that achieves all the already satisfied security guarantees, plus forward secrecy with respect to both entities, by following the Forward Secrecy column in Table 4.5:

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

4.4.4 Unknown Key Share (UKS) results

We use Scyther to analyse all protocols for which key authentication is claimed for UKS vulnerabilities. We find that UKS attacks are possible on every implementation of two such protocols, and on some implementations of several other protocols.

We first explain the unknown key share attack on the 3-KA-11 protocol in detail. A graphical representation is given in Figure 4.11. In the attack, the adversary does not modify the contents of any messages, but only changes the implicit sender/recipient

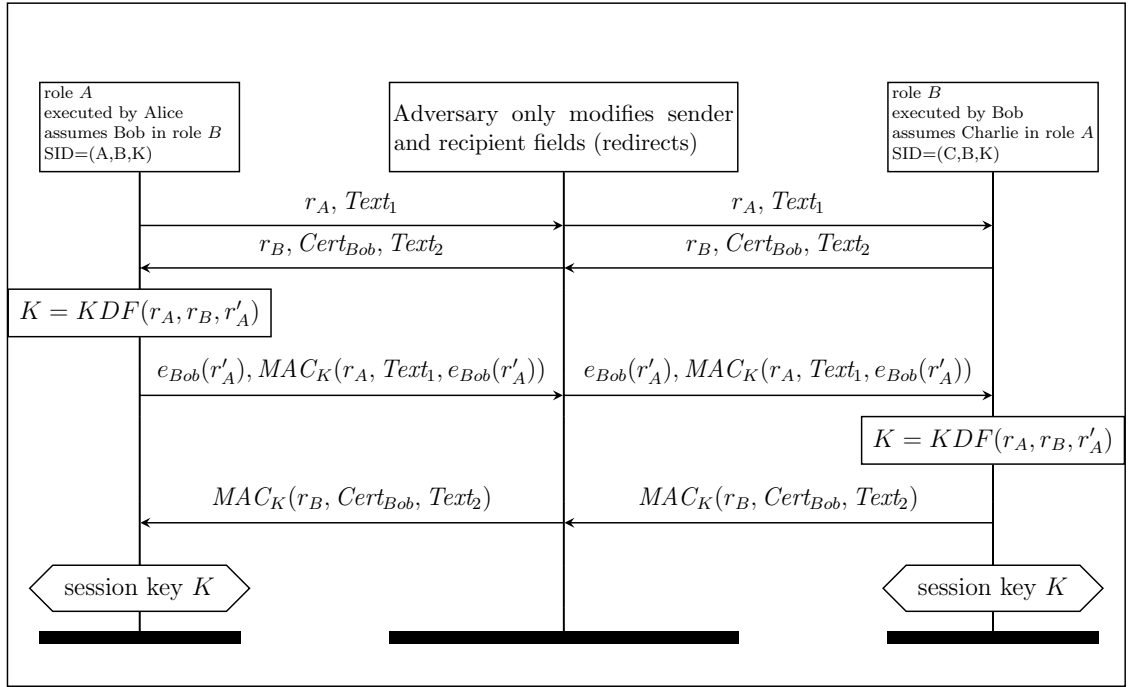


Figure 4.11: Unknown key share attack on protocol 3-KA-11

fields. When Alice executes role A with her intended partner Bob, she sends out her first message. The adversary modifies the sender field to “Charlie” and forwards the message to Bob. Bob assumes Charlie wants to communicate with him, so Bob starts to execute the B role and sends the response message to Charlie. The adversary redirects this message to Alice. The protocol continues as usual, except that the adversary continues to modify the sender fields and redirecting the responses. There is nothing in the messages that allows the entities to check each other’s beliefs about their communication partner. In the end, Alice and Bob compute the same key K . Although the adversary does not know this key, Bob will believe that any subsequent messages he receives, which are encrypted or authenticated using K , are coming from Charlie, when in fact they come from Alice. This can lead to a serious authentication flaw [16, p. 139].

A second UKS attack is possible on the 2-10 protocol. The protocol is shown in Figure 4.12, and it suffers from a role-mixup attack in which Alice and Bob both perform the A role and compute the same session key. This can lead to later reflection attacks and misinterpretation attacks when the session key is used to encrypt payloads. The attack is depicted in Figure 4.13. In implementations in which entities can perform multiple roles, protocols 2-2, 2-8, 2-9, 2-11, and 2-12 are also vulnerable to UKS attacks.

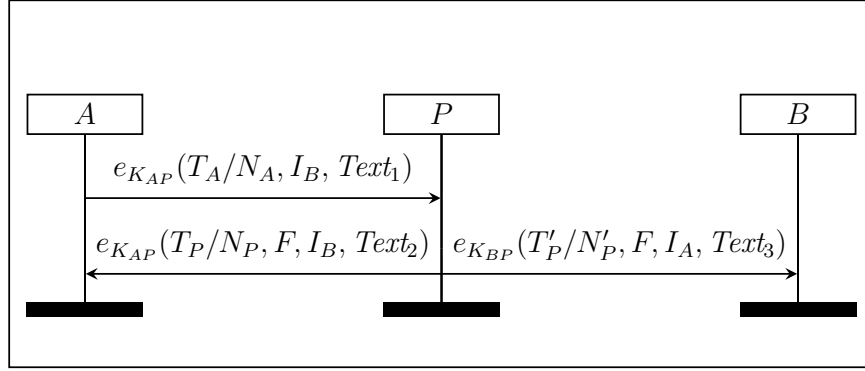


Figure 4.12: Protocol 2-10

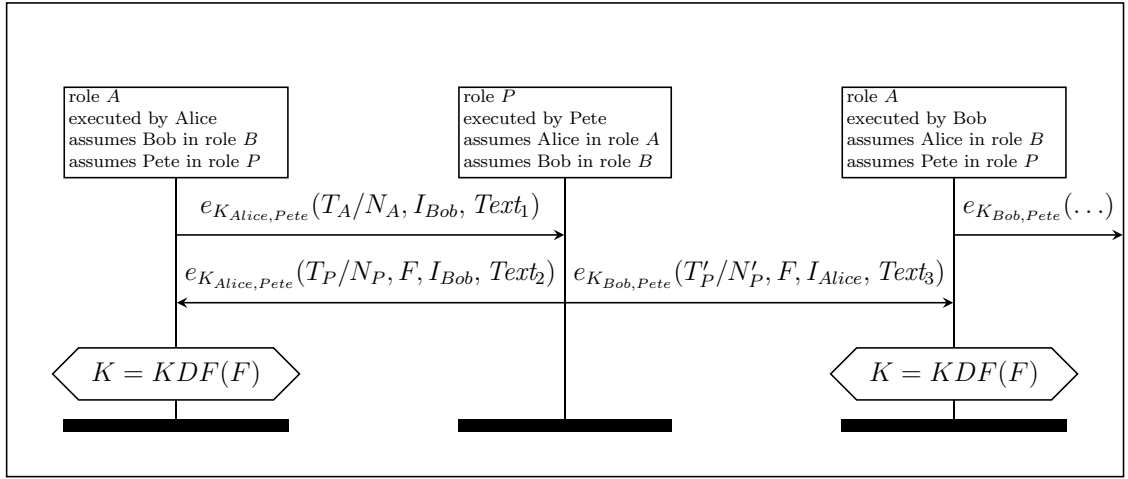


Figure 4.13: Protocol 2-10 UKS attack

Fortunately, UKS attacks can be prevented by choosing a key derivation function that includes the identifiers (I_A and I_B) of the involved entities [14, 16]. For example, this is required by the NIST SP-800-56A key derivation [5], which is included in Part 3 of the standard. We model the use of this KDF and use automated analysis to confirm that this prevents the UKS attacks. Intuitively, including the identities in the KDF ensures that entities that have different beliefs about their intended peers compute different keys, which thwarts UKS attacks.

4.5 Recommendations

In this section, we provide four recommendations to improve the ISO/IEC 11770 standard.

1. Making the threat model explicit. It is commendable that for every protocol in this standard there is a list of fairly precisely defined security requirements. However,

an essential, yet missing ingredient to unambiguously state what is meant by these properties is an explicit threat model. Without the threat model, it is impossible to assess if the security requirements are met, as also discussed in [9]. We recommend its addition to the standard, with the proviso that any introduced differences from our threat model might require additions to or revisions of our other recommendations.

2. Improving protocols to achieve stated properties. Our second recommendation is to make small changes to the protocols to achieve their stated properties, if possible. The most straightforward way is to adopt the recommendations made for ISO/IEC 9798 in [8, p. 13]. In particular, we require that

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

Following these recommendations addresses all of the issues in Table 4.5 except for the problems with protocol 3-KA-11.

3. Using appropriate key derivation functions. Our third recommendation improves the security of the standard by preventing unknown key share attacks. If the input to the key derivation function includes the identities of the communicating parties, UKS is directly prevented. For example, the execution of protocol 3-KA-11 depicted in Figure 4.11 no longer constitutes a UKS attack: Alice and Bob simply compute different session keys. We therefore recommend making the inclusion of identities in the key derivation an explicit requirement. A key derivation function from NIST SP-800-56A [5], which is described in ISO/IEC 11770, meets this requirement.

4. Addressing remaining issues with 3-KA-11. 3-KA-11 inherently does not offer perfect forward secrecy or mutual authentication. Switching to a protocol that does may substantially change the environmental assumptions, including the pre-distribution of keys. For example, replacing 3-KA-11 with the mutually authenticated TLS-DHE.RSA additionally requires that the client owns a private signing key.

A simpler solution is to adapt the statements made about the protocol. In particular, it should *not* be claimed in the overview table [47, p. 42] that 3-KA-11 achieves implicit key authentication for both entities, that it achieves key confirmation for both entities, or that it achieves MFS. Similarly, the running text [47, p. 26] should not claim that 3-KA-11 achieves mutual explicit key authentication.

4.6 Conclusions

Commendably, the ISO/IEC 11770 standard includes statements about the security guarantees achieved by its protocols, such as those reflected in Table 4.5. It is rare for a standard to include such statements. Specifying the required security guarantees substantially helps the users of the standard in selecting the appropriate protocol for a given scenario. We recommend that other standards follow this example and try to include more precise statements about the intended security guarantees in their specifications.

However, there exist attacks which render some of the statements in the standard false. In retrospect, although we found all the attacks through automatic analysis, some attacks should have been found by manual inspection. This holds especially for 3-KA-11, which is based on the TLS unilaterally authenticated RSA handshake: it is clear that this protocol cannot offer key authentication or confirmation for both parties, since only one party is authenticated.

One way in which standardisation bodies could be more proactive is by being aware of analyses of standards on which they build. For example, many protocols in ISO/IEC 11770 are mentioned to be derived from authentication protocols in ISO/IEC 9798. In 2012, the ISO/IEC 9798 standard was analysed, several problems were identified [8], and it was subsequently updated to fix the identified problems. However, it seems that no attempt was made to determine if the derived protocols inherited these problems. Our analysis shows that this was in fact the case, implying that the attacks on protocols from Part 2 of ISO/IEC 11770 could have been identified earlier. In fact, applying the recommendations for ISO/IEC 9798 as described in [8] to ISO/IEC 11770 would have prevented all of the issues in Table 4.5 except for those with 3-KA-11.

Standards that cover protocols for a wide range of different usage scenarios benefit from periodic updates with modern security requirements. The standard does not claim resilience to UKS or KCI attacks. One could consider identifying the protocols that achieve these properties and improving the other protocols. For example, all UKS attacks that we found can easily be prevented at negligible cost by using key derivation functions that include the identities of the participants. We therefore recommend including the identities in the input to the KDFs in the standard.

Compared to other security protocol standards, ISO standards have been less analysed in the academic literature. A possible reason for this difference is that people who are not members of the working groups can only access the standards

by purchasing the final versions. One possible way to promote the external analysis of ISO standards is to publish early drafts of proposed changes or new standards. Parties that are interested in applying the standards will still need to purchase the final versions to ensure they comply. However, interested parties can freely analyse the designs from the early drafts, which may help identify and prevent problems before the standards are deployed.

Chapter 5

Formal analysis of TLS 1.3

We looked at many standardised security protocols in the last two chapters. This includes TLS 1.2 [28], a protocol that is critical in securing communications over the Internet, but is lacking in both efficiency and security. The TLS 1.3 proposal [89] coming from the TLS working group seems promising: it features a handshake reworked for efficiency, it supports encrypting and confirming more handshake messages than previous versions of TLS, and it does away with some outdated and hastily incorporated mechanisms.

With the only other proposal having been merged with the working group one, there is no apparent contender for TLS 1.3. However, it is essential to objectively evaluate the state of this merged proposal and the direction in which it is heading before its full deployment. We perform a formal analysis of both the unilaterally and mutually authenticated modes of the draft of an earlier revision, rev 06, of the TLS 1.3 proposal. We verify session key secrecy and perfect forward secrecy in both modes with respect to a powerful symbolic attacker and an unbounded number of threads by using the Tamarin prover. We build on our model and use it to analyse revision 10. The results of the analysis directly influenced revision 11.

5.1 Introduction

The TLS working group was established in 1996 to standardise a *transport layer* security protocol based on SSL 3.0 [36]. The group has completed a series of specifications that describe the TLS and DTLS protocols, with the latest deployed TLS version being 1.2 [28]. TLS remains the most important security protocol for the everyday Internet user. However, some users may be surprised to find that, in spite of hearing inadequately restricted rumours that researchers [49, 55] “proved the security of TLS”, numerous vulnerabilities have been found in TLS 1.2. These vulnerabilities can lead

to timing attacks [3], partial plaintext recovery [2, 73], and client impersonation [12]. Some attacks have even incorrectly been considered fixed [58, 101].

Moreover, likely due to the complexity of the protocol, some implementations contain serious vulnerabilities [11]. These include negotiating weak export ciphers and skipping critical handshake messages, which respectively allow attackers to perform server impersonation against some mainstream browsers or to deny any kind of security benefits at all to the protocol participants.

It is therefore not surprising that the TLS working group is preparing a proposal for a new version of TLS. As of September 2015, the specification is not yet entirely set in stone, even after absorbing the second proposal which has been considered on and off since late 2014. Krawczyk’s rough sketch of a protocol he calls OPTLS [53] has influenced multiple working group proposal drafts.

Regardless of the frequently changing state of affairs, the September 2015 draft of TLS 1.3 [88] does fully describe a security protocol which includes the major features of OPTLS and at first glance seems to address the security vulnerabilities of TLS 1.2: more of the handshake is being encrypted, more messages are being confirmed, Authenticated Encryption with Additional Data (AEAD) has replaced MAC-Then-Encrypt in the record layer, renegotiation is no more, etc.

Feature-wise, the protocol latency has been improved. The initial handshake has been reworked to take just one round-trip instead of two. Moreover, two new modes have been added, a zero round-trip time mode and a resumption replacement mode, which are so rich and general that they might undermine one’s confidence in the security of the full protocol.

Giving an unambiguous, simple, backwards-compatible, yet secure specification of a protocol based on the complex TLS 1.2 is very tricky, especially with the added features that bring in additional security considerations. Although a simplification of earlier versions was initially an important guiding principle for the design of TLS 1.3, some parts of the protocol are clearly becoming increasingly complicated. This might ultimately lead to making new mistakes and repeating old ones, be it merely in future implementations or even the critical security infrastructure of the protocol itself. Before TLS 1.3 is deployed, it is therefore critical to analyse it and iron out as many bugs as possible.

In particular, we contribute towards this goal by first performing a formal security analysis of revision 06 of the TLS 1.3 specification draft [84]. We use the Tamarin prover to symbolically verify the Handshake and Record Protocols, with respect to a powerful attacker model. We establish that both the unilaterally and mutually

authenticated modes of TLS 1.3, rev 06 satisfy session key secrecy with respect to an unbounded number of threads and an active, Dolev-Yao style adversary capable of Actor Key Compromise, Diffie-Hellman Exponent Reveal, and breaking Perfect Forward Secrecy. We then build on our model with Cremers, Scott, and van der Merwe in order to analyse the revision 10 draft of the TLS 1.3 specification [88]. Our work directly influences an update of revision 11 [89].

The protocol model and tools used are available for download from [45].

Overview. In Section 5.2, we give a brief overview of the TLS 1.3 design process up to rev 06. We present our rev 06 protocol model in Section 5.3, and describe the rev 06 security properties and threat model in Section 5.4. We detail our rev 06 analysis and its results in Section 5.5, and build on this work in Section 5.6 to perform the analysis of rev 10. We describe a potential attack on one of its proposed extensions that influences an update of rev 11 in Section 5.7, and conclude in Section 5.8.

5.2 TLS 1.3, rev 06 overview

The principal idea behind the TLS Handshake and Record Protocols is to allow two applications to communicate securely. The Handshake Protocol enables the applications to authenticate each other and negotiate encryption algorithms and cryptographic keys. The Record Protocol is then used to achieve data confidentiality and integrity by using the negotiated keys. Records can comprise multiple protocol messages, and a single message can be fragmented into multiple records.

We give an overview of the design rationale and the technical details of revision 06 of the TLS 1.3 Handshake and Record Protocols specification [84] in this section.

5.2.1 TLS 1.3 design rationale

The TLS working group is responsible for improving the security and efficiency of TLS 1.2 in order to build TLS 1.3. According to the charter of the working group [34], some of the main design goals for TLS 1.3 are as follows:

- encrypt as much of the handshake as possible,
- reduce handshake latency—one Round-Trip Time (1-RTT) for full handshakes, zero Round-Trip Time (0-RTT) for repeated handshakes,
- reevaluate handshake contents, and

- update record protection mechanisms.

Before we get into the details of the specification itself, we comment briefly on how TLS 1.3 implements these four points.

Handshake encryption

The reason for handshake encryption is to reduce the amount of data observable to both passive and active attackers [34]. In contrast to TLS 1.2, which only provides its users with session keys to protect application data, TLS 1.3 features additional session keys used for resumption and for handshake encryption purposes. The handshake encryption begins immediately after the handshake keys themselves are negotiated in plaintext with the help of a Diffie-Hellman (DH) exchange.

Handshake latency

The TLS 1.2 handshake required clients to twice go through the trouble of sending a message and receiving a response from the server, i.e. two Round-Trip Times (2-RTT), before they could send any application data. The handshake was reworked in TLS 1.3 to require just one round trip (1-RTT) if no mismatches occur. Additionally, an option for clients to send application data before receiving any messages from the server in TLS 1.3 was a highly desirable option that would provide a clear efficiency advantage over TLS 1.2. This concept was long under debate, especially when Krawczyk's OPTLS (cf. Chapter 6) was still competing with the TLS working group proposal. One of the main objections to both of the early proposals, due to Kahn Gillmor at the interim meeting of the working group in Seattle [82], was that they offered limited anti-replay protection for application data sent in the first flight, i.e. immediately with the client's Hello message.

The anti-replay mechanism in TLS 1.2 is very simple: at the beginning of every session, each side generates a fresh random value, and later computes session keys that contain their own random value. Therefore, replays of messages encrypted with the session keys, such as Finished messages or exchanged application data, can never go unnoticed in other sessions. In TLS 1.3, however, we would like anti-replay guarantees for the server to extend to the client's 0-RTT data, where no session keys have been established yet and a random value of the server has yet to be received.

Borrowing an idea from the TLS Snap Start draft [57] may appear to be a good way to ensure anti-replay for the server: it can simply keep a list of all the clients' random values within a time window, along with a server context that e.g. indicates a

server's data center. However, if server reboots are considered, i.e. when a server's state is lost, then the question arises of how the server should react to new 0-RTT connections. For availability, the design should avoid dropping all of them, so one solution is that it ignores the application data sent with the client's initial message, gets the random value from the message, and requests that the client retransmits the ignored application data. If an attacker

1. intercepts the client's first message in a resumed handshake,
2. sends it to the server (server accepts 0-RTT),
3. drops the server's acceptance message,
4. reboots the server,
5. sends the client's message to the server again (server rejects, asks for retransmission),
6. delivers the retransmission request to the client, and
7. sends the client's retransmission to the server (accepts same data again),

then the attacker mounted a successful replay attack, which can be problematic if the message was "POST /buy-something". The attack is not too realistic because it requires forcing a server to reboot, which together with the message transmissions needs to happen before the client times out. However, the attack can be made more realistic if the server is distributed. In this scenario, talking to one server might process the POST request, talking to another might fail, and after the client's resend the first server might process the same data a second time.

Two ways to avoid the above issues were discussed by the working group. One was to mandate that applications allow unreliability, where the TLS stack would not automatically retransmit the initial application data if the client gets the server's 0-RTT rejection message. The application would then have decided how to proceed. Another one was requiring the applications to consider non-confidential any application data sent in the first flight of 0-RTT. We return to this issue in Section 5.6, where we discuss the state of 0-RTT support in TLS 1.3.

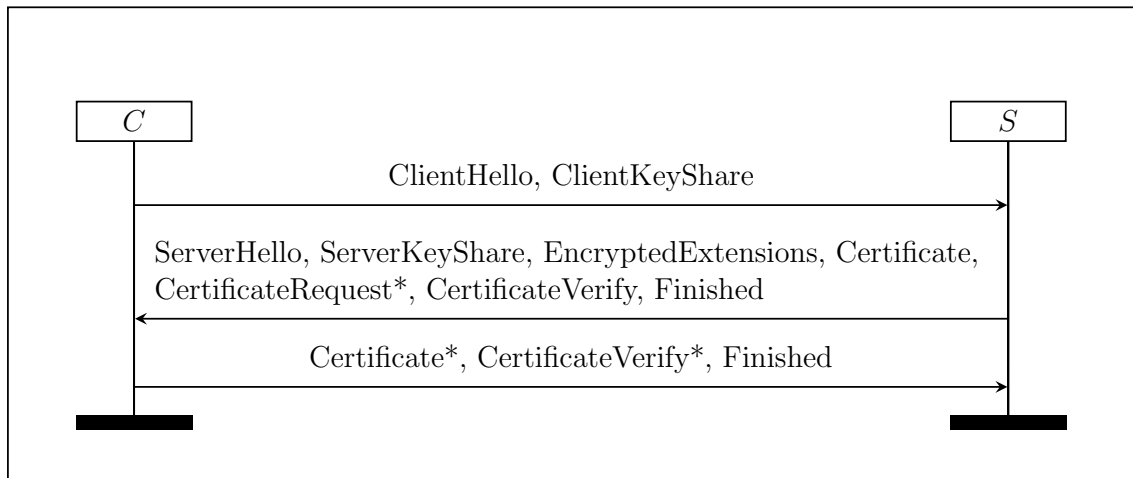


Figure 5.1: Full handshake

Handshake contents

As discussed above, the handshake has been reworked to provide a 1-RTT mode, an additional server message has been added when there is a parameter mismatch, and compression has been removed. Static DH and RSA have been removed in favour of finite field ephemeral DH (DHE) and elliptic curve ephemeral DH (ECDHE) key exchange modes for their Perfect Forward Secrecy support. RSA certificates are still being used for signature keys used to verify transcripts in both the DHE and, alongside ECDSA certificates, ECDHE modes.

Record protection mechanisms

The biggest change is the transition to state-of-the-art Authenticated Encryption with Additional Data (AEAD) ciphers [67] such as AES-GCM [92] and ChaCha/Poly [59]. All of the non-AEAD ciphers have been removed in TLS 1.3.

5.2.2 TLS 1.3, rev 06 technical details

Now we briefly explain the technical details behind the Handshake Protocol in TLS 1.3, rev 06. The full handshake with immediate agreement on cryptographic parameters is shown in Figure 5.1. Every protocol message followed by an asterisk can be omitted if only unilateral (server) authentication is required.

1. A client sends to a server an offer of cryptographic parameters that are later used to establish session keys (**ClientHello**), and freshly generated Diffie-Hellman (DH) half keys (cf. Section 3.5.4) in the offered groups (**ClientKeyShare**).

2. The server responds with:

- its choice of cryptographic parameters and half key among the ones offered (ServerHello),
- its own freshly generated DH half key (ServerKeyShare),
- encrypted extensions not used for key establishment (EncryptedExtensions),
- its public key certificate for checking a soon-to-follow signed confirmation (Certificate),
- an optional request for the client's certificate (CertificateRequest),
- the signed confirmation of all messages exchanged thus far (CertificateVerify),
- a confirmation of the whole handshake, encrypted and authenticated (Finished).

3. Finally, the client optionally sends its own certificate (Certificate) and signature on the whole handshake so far (CertificateVerify), and it confirms all messages by encrypting and authenticating them (Finished).

If the server cannot or will not accept the parameters suggested by the client in its first message, then an additional HelloRetryRequest message is sent by the server in an attempt to persuade the client to change its offer of parameters and DH half key (cf. Figure 5.2). After a successful handshake, the same session can be resumed on later connections (cf. Figure 5.3).

Most of the cryptographic computations in TLS 1.3, rev 06 include a derived secret from the shared DH group element, g^{xy} , which is called the *premaster secret* (PMS). All of the other, intermediate shared secrets are derived from PMS by using a pseudo-random function (PRF); they are the *handshake master secret* (HMS), the *(extended) master secret* (MS), and the *resumption premaster secret* (RMS). They are used to compute *session keys*, which protect the rest of the handshake and the application data exchanged by the client and the server later on.

The handshake session keys computed from HMS are used to encrypt the rest of the handshake after the ServerKeyShare message. In other words, the server encrypts the handshake starting with EncryptedExtensions, and the client from (his) Certificate. MS is used to compute record layer keys for encrypting all application data before a session is resumed, and RMS is kept in the session cache to be used as the premaster

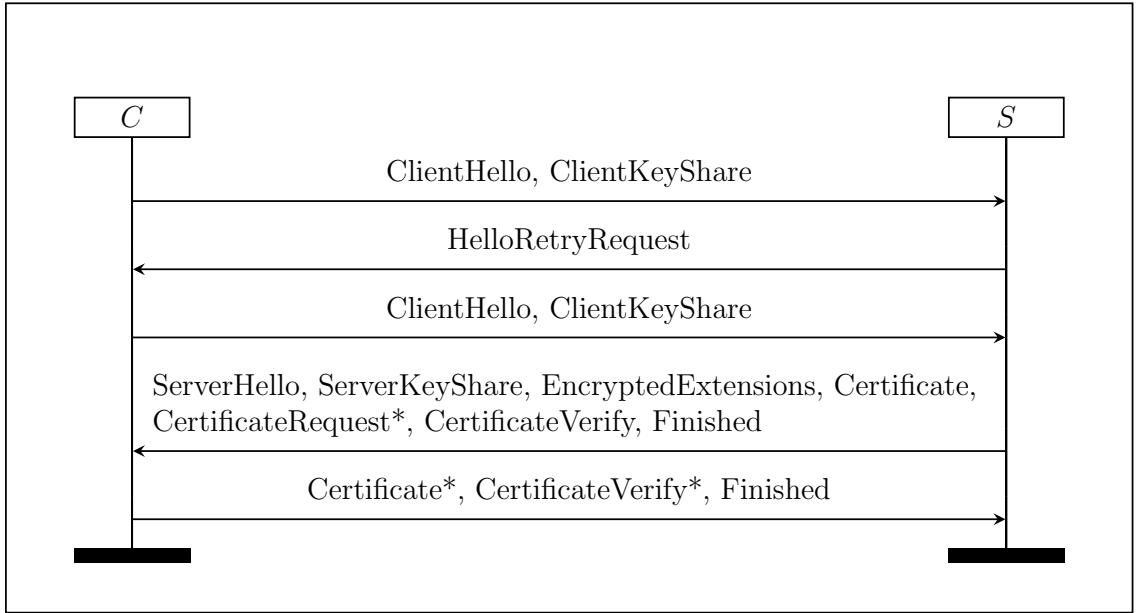


Figure 5.2: Full handshake with mismatched parameters

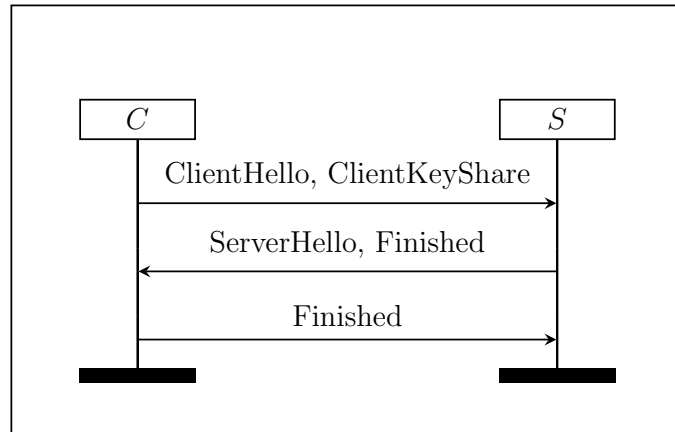


Figure 5.3: Resumed handshake

secret in resumed handshakes. Also note that `Finished` messages in TLS 1.3, rev 06 are protected by HMS instead of MS, where the latter was the case in TLS 1.2.

We now briefly explain the cryptographic computations. The DH group element PMS is computed as follows: the client sends a number of cryptographic parameter suggestions, including a list of DH groups, each paired with one of its freshly generated elements $X = g^x$, in the first message. Then the server chooses a group and element pair from the ones offered, generates a fresh y such that $Y = g^y$ is in the chosen group, and sends its choices in the second messages. The two selected values are respectively called the client and server DH share. The client and server then respectively compute Y^x and X^y , both of which should be equal to g^{xy} if all is well. The derived secrets are

computed as the first 48 bytes of PRF values:

$$\begin{aligned} \text{HMS} &= \text{PRF}(\text{PMS}, \text{“handshake master secret”}, \text{session_hash})[0..47] \\ \text{MS} &= \text{PRF}(\text{HMS}, \text{“extended master secret”}, \text{session_hash})[0..47] \\ \text{RMS} &= \text{PRF}(\text{HMS}, \text{“resumption premaster secret”}, \text{session_hash})[0..47] \end{aligned}$$

The session hash is simply a hash of all preceding handshake messages up to the time when the session hash is computed. It contains the concatenation of ClientHello, ClientKeyShare, ServerHello, ServerKeyShare, and HelloRetryRequest messages (if any) in the computation of HMS because HMS is needed immediately after that to encrypt the rest of the handshake. The hash contains additional messages when MS and RMS are computed: it comprises all the preceding handshake messages other than the Finished messages.

At this point we can explain the session key derivation in TLS 1.3, rev 06. Two key blocks are generated from HMS and MS in the following way:

$$\text{key_block} = \text{PRF}(\text{HMS/MS}, \text{“key expansion”}, \text{server_random} + \text{client_random})$$

The random values in the above calculations are the ones from the Hello messages, and the plus sign denotes concatenation. The output is split into four equal parts to respectively form four values: client_write_key, server_write_key, client_write_IV, and server_write_IV. Depending on whether HMS or MS is used, the write keys extracted from key_block are respectively handshake keys or record layer keys. The IV values are used as per-record nonces for the AEAD construction.

5.3 Modelling the protocols in TLS 1.3, rev 06

In this and the next section, we discuss our modelling of the protocols in revision 06 of the TLS 1.3 specification draft [84], the tools used in our formal analysis, and explain the results in detail. We want to build a formal model of the Handshake and Record Protocols as input files for the Tamarin prover [95]. Tamarin allows for fine-grained control when specifying security properties and adversarial capabilities. It uses multiset-rewriting techniques to verify security properties with respect to symbolic attackers and supports the Diffie-Hellman key exchange.

We choose to model the unilaterally authenticated mode of TLS 1.3, rev 06 as well as the mutually authenticated mode, because in most of the common use cases, including private users utilising online services such as banking, voting, and mail, only

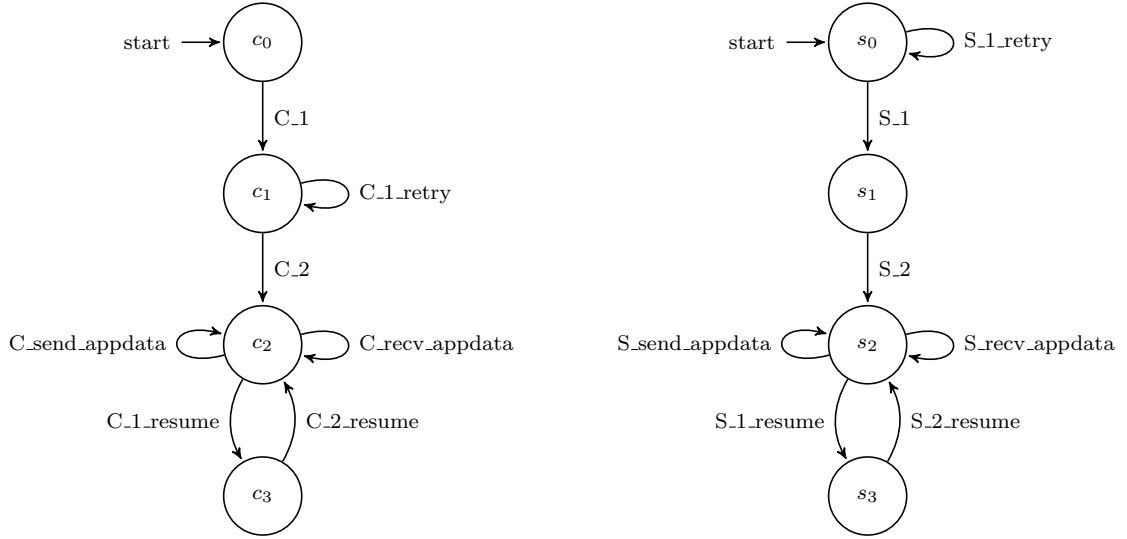


Figure 5.4: Simplified client and server state machines for TLS 1.3, rev 06

the server is authenticated to the client during the TLS handshake. Any required client authentication is subsequently done over the secure TLS channel established by the handshake, e.g. by using two-factor authentication. Moreover, many web servers do not care about any client authentication at all and only require sender invariance, i.e. knowing whether two requests supposedly coming from the same client are indeed coming from the same client. Therefore, clients interested in web browsing and the above online services often do not possess long-term asymmetric keys or certificates for their public keys.

The first step in modelling the protocols is to construct their abstraction which we can then analyse. As with any other abstraction, we attempt to strike a balance between a completely accurate, yet prohibitively complex, model and capturing the most important cryptographic and algorithmic aspects of the protocols. In our rev 06 model, we approximate AEAD by Encrypt-then-MAC as per RFC 7366 [40], and refine it in our future work on the analysis of rev 10. We focus on the interaction of the initial handshake with any number of retries and interleavings of resumed handshakes and sending/receiving application data for both sides. The state machines which model the described behaviour are shown in Figure 5.4. We believe that a complete state machine, which would contain considerably more states and transitions to account for all the possible side-cases contained in the draft, should be added to the final specification to avoid any kind of ambiguity when the protocols will be implemented or analysed. Such ambiguities can lead to incorrect implementations and the inadvertent introduction of serious vulnerabilities [11].

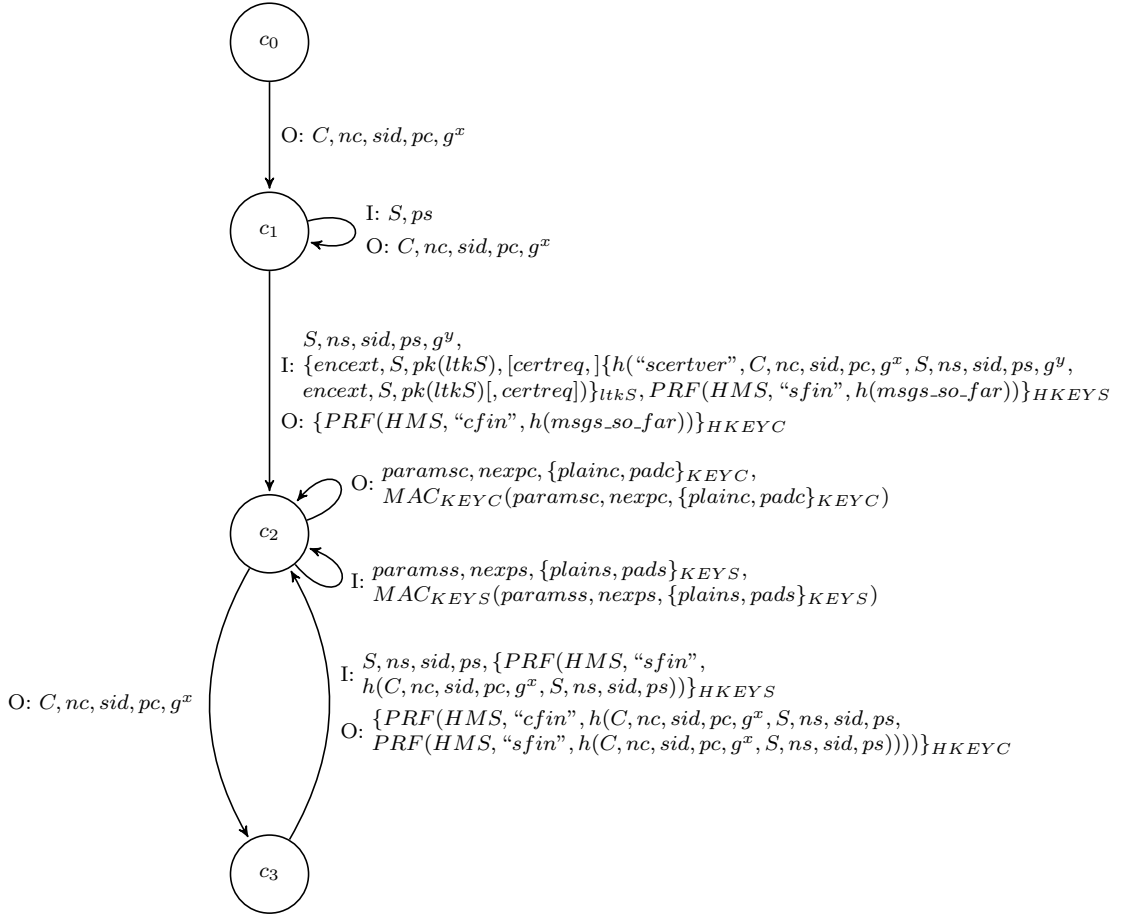


Figure 5.5: Simplified Mealy state machine for client in our TLS 1.3, rev 06 model

We adapted our previous model of TLS 1.2 with renegotiation and resumption for this purpose. The initial plan was to use the model to look for more attacks similar to [12], and check if TLS 1.2, after being updated by fixes outlined in that paper, could withstand attacks from more powerful adversaries. However, the TLS working group decided to remove renegotiation from TLS 1.3, so we transitioned from TLS 1.2 to TLS 1.3, rev 06. While both are fairly similar in their core structure, there were two clear advantages of doing a formal analysis of the latter: (1) it allowed us to evaluate the then current design directions of TLS 1.3, and (2) it had great potential as a quality modelling foundation with respect to future versions of the specification.

We refine our state machine for the client into a Mealy state machine [68] by labeling the transitions with our modelling of the message contents (cf. Figure 5.5). For readability, we did not unfold all the composite terms shown in the figure. Note that we use abstractions for some of the message contents, e.g. all cryptographic parameter offers are represented as a single atomic value; this and the perfect cryptography

```

1 rule C_1:
2 let
3   C   = $C
4   tid = nc
5   nc  = ~nc
6   sid = ~sid
7   pc  = $pc
8   X   = 'g', ~x
9 in
10  [ Fr(nc)
11    , Fr(sid)
12    , Fr(~x)
13  ]
14  --[ Start(tid, 'client', 'init')
15      , DH(tid, ~x)
16      , InitSid(tid, sid)
17      , GenNc(tid, nc)
18      , InitTid(tid, sid, 'client')
19  ]->
20  [ Out(<C1>)
21    , DHExp(tid, ~x)
22    , St_init(C, 1, tid, C, nc, sid, pc, ~x)
23  ]

```

Figure 5.6: Rule C_1 in our TLS 1.3, rev 06 model

assumption (cf. Section 1.2) preclude¹ our analysis from detecting attacks such as Logjam [1].

We can directly transform the above Mealy machine into an input for the Tamarin prover, which consists of *rules* that model transitions between (*global*) *states*. As an example rule, we give the first rule for the client to show how an execution of the TLS 1.3, rev 06 handshake is initiated in our model. The rule requires the generation of three fresh values in that need to be sent out in the client’s first message. These fresh values can only be generated with the help of the **Fresh** rule. More details on these and other concepts within the Tamarin framework are provided in Chapter 2.

We use the variable `tid` to name the newly created client thread. The DH action `DH(tid, ~x)` allows us to connect the thread identifier `tid` to its private DH value `~x`, which is of type Fresh as denoted by the tilde sign preceding the variable name. The `St_init` fact serves as a program counter by allowing the client to remember that it already sent its first message in thread `tid`. The `DHExp` fact allows the adversary to compromise the private DH value of thread `tid`, in a rule that we detail in the next section, where we also specify our security properties and threat model for TLS 1.3, rev 06.

¹While Tamarin can support rules for weak encryption, we do not specify them in our analysis.

5.4 Security properties and threat model

The top-priority security requirement of TLS is traditionally listed as “cryptographic security”, and this remains true in TLS 1.3, rev 06 [84]. It is vaguely defined as “establishing a secure connection between two parties”. A more precise, yet informal explanation can be found in Appendix E [84, pp. 82-86] of the proposal.

We focus on the most important security property necessary to obtain a secure TLS connection, and that is session key secrecy: whenever two agents complete an initial handshake, perhaps followed by a number of repeated handshakes and application data transmissions, all established session keys must be unknown to the attacker. In the unilateral case, we require that only the client obtains session key secrecy guarantees: an unauthenticated client might be an attacker who injects a DH half key with an exponent of its own choosing, and computes the session keys legitimately in communication with a server.

We would of course like to find out how secure TLS 1.3, rev 06 is with respect to powerful, active adversaries. A standard black-box assumption we make is that the cryptography involved works as it should, e.g. a message can only be decrypted with the inverse key of the used encryption key, and digital signatures verifiable with a public key can only be created with the corresponding private key. This view simplifies the proofs and enables the analysis of many different security contexts, but also hinders the adversary; not all of its computational avenues are taken into account. To offset this as much as possible, we ideally want to let the attacker reveal all the secret information directly involved in every protocol execution:

- all long-term private keys,
- all freshly generated DH exponents,
- all (pre)master secrets, and
- all established session keys.

This list can be collapsed into a shorter one:

- all long-term private keys, and
- all freshly generated DH exponents.

The second list is equivalent to the first because as soon as *either* of the freshly generated DH exponents in a thread (also called a run, cf. Chapter 3) becomes known to the attacker, it can immediately compute the same PMS that the thread computes. The PMS is a value which is critical for session key secrecy because all the derived (pre)master secrets and session keys can easily be computed from it.

Note that we do need to restrict the compromises that the attacker can make, similarly to e.g. the eCK model [56]: if it could trivially reveal the PMS of the thread under attack, then it could trivially compute the session keys of that thread. We can allow the attacker to compromise the private DH exponents in most threads that are not under attack. The only exception are so-called *partner threads* to the attacked thread. Each thread is uniquely identified by the random value it generates, i.e. `client_random` or `server_random`, and this identifier is used for the thread regardless of its retries, or later application data exchanges and resumptions. We call two threads partner threads if they agree on their thread identifiers, as well as their roles and identities. Without the above exceptions, which together with the ones in the next paragraph are usually called *the freshness condition* in game-based AKE security definitions, our threat model would make every secrecy and authentication property trivially false.

Apart from disabling the attacker from revealing some DH exponents, we need to disallow long-term key reveal queries for some protocol participants and at certain timepoints. To avoid making properties trivial as described in the preceding paragraph, we disallow reveals of long-term keys that belong to the intended peer of the attacked thread. However, we weaken this restriction to allow reveal queries on the peer's long-term keys after the attacked thread finishes its initial handshake, i.e. computes its initial pair of session keys. The reason for this weaker restriction is to check for Perfect Forward Secrecy (PFS). Note that we also check for Actor Key Compromise (AKC) because the attacker may reveal the long-term keys of whoever is running the attacked thread (the *actor*, cf. Chapter 3). For more details on the mentioned notions and properties, the reader is referred to the previous two chapters.

We give the two mentioned unrestricted rules here, and announce that the explained restrictions are only introduced in the property specifications, which are called *lemmas* in the Tamarin input language:

```

1 rule Reveal_Ltk:
2   [ !Ltk($A, ~ltkA) ] --[ RevLtk($A) ]-> [ Out(~ltkA) ]
3
4 rule Reveal_DHExp:
5   [ DHExp(~tid,~x) ] --[ RevDHExp(~tid) ]-> [ Out(~x) ]

```

This is the general form of our property specifications:

```

1 lemma <name> [<options>]:
2   "All params1 params2 paramsn #i1 #i2 #in.
3     <action1>(<params1>) @ i1
4     & <action2>(<params2>) @ i2
5     & <actionn>(<paramsn>) @ in
6     ==>
7     <consequences>
8     | <negated_freshness_condition>

```

In other words, we claim that whenever certain actions happen at arbitrary timepoints, which are typed with `#` in quantifications and mapped to actions with `@`, either the consequences we expect happen as well, or the attacker “cheated” by violating the freshness condition. For example, the session key secrecy result with both PFS and AKC enabled is specified as follows:

```

1 lemma secret_sessionkeys [use_induction]:
2   "All tid actor peer pms key #i #j.
3     SessionKey(tid, actor, peer, pms, key) @ i
4     & KU(key) @ j
5     ==>
6     (Ex #r. RevLtk(peer)@r & #r<#i)
7     | (Ex #r. RevDHEExp(tid)@r)
8     | (Ex tid2 #r #s. RevDHEExp(tid2) @ r &
9       Partner(tid2,tid,peer,actor) @ s)"

```

5.5 Formal analysis of TLS 1.3, rev 06

After modelling TLS 1.3, rev 06 by closely following the rev 06 specification and deciding which property is the critical one to verify, we encode our model in a header file called `TLS-13-resum.h`. This file is written in a human-readable way by taking advantage of macro expansions. The properties we verify are specified as first-order guarded trace formulae, each separate in its own file, e.g. `secret_sessionkeys.cpp`, with all the lemmas and axioms Tamarin needs for the proof. The Tamarin files required to reproduce our analysis results are available at [45].

Toggling the authentication mode to be analysed in our Tamarin model is easily done by (un)commenting a single line in the corresponding property file. Note that this provides only limited coverage because we assume that all sessions have the same client authentication status. We refine this view in our later analysis of rev 10, where we allow any combination of client-authenticated and client-unauthenticated sessions.

We now want to check that our Tamarin model indeed supports all of the important TLS 1.3, rev 06 execution branches, such as retries, resumptions, and application data transmissions. A helpful method in doing so is proving the existence of traces as specified in `send_appdata_possible.cpp`, which involves passing through all the states in

our state machines to ensure that no binding or typing mistakes block Tamarin from visiting some state. This is straightforward to do when all adversarial interference is switched off by asserting in the property file that every incoming message matches some outgoing message:

```
1 #define In(...) AuthenticMessage(__VA_ARGS__)
2 #define Out(...) AuthenticMessage(__VA_ARGS__)
```

The ellipsis ... in the macro name is referred to as `__VA_ARGS__` in its definition because we are using the C preprocessor for macro expansion in our TLS 1.3, rev 06 files. These macros in particular replace all the In and Out facts with AuthenticMessage facts, the effect being that every consumed In fact matches a produced Out fact, i.e. every received message is sent by some protocol participant. With the help of this auxiliary assumption, we fix a blocking typo in our model that completely disabled resumption traces.

Having made sure that our Tamarin model will consider all the transitions that occur in our state machines, we can now proceed with the formal verification of session key secrecy as specified in `secret_sessionkeys.cpp`. We initially avoid non-termination of verification by only verifying session key secrecy without retries and session resumptions, and with at most one pair of sent/received application messages per thread. After that, we analyse the full protocol by applying the reasoning behind two examples from the Tamarin example library, *Minimal-Loop-Example.spthy* and *Minimal-HashChain.spthy*. Both of these theory files contain protocols with loops. The first one explains how adding actions to rules that form protocol loops allows for easier navigation through the loops during verification. The second example deals with iterated function application, which coincides with the way all the relevant secret values in TLS 1.3, rev 06 are computed—namely, by hashing the premaster secret over and over.

We prove the session key secrecy lemma shown at the end of the previous section by incrementally building on the secrecy of intermediate computation steps, including the secrecy of the fresh DH exponents and the premaster secret. For an easier analysis, we start off with a passive adversary and only strengthen the adversary model by adding in one adversary rule at a time. We find the single remaining blocking issue by scrutinising the proofs in the Tamarin GUI: the session ID value *sid* and the long-term server key *ltkS* are both of type Fresh. This potentially allows *ltkS* to be received in resumed handshakes, which is simply a case we need to prove impossible. We address this case in a lemma we call `sid_invariant_for_client`.

```

1 lemma sid_invariant_for_client [use_induction,reuse]:
2   "All tid sid #i. Sid(tid,sid)@i==>
3     (Ex #j. GenSid(tid,sid)@j & #j<#i)
4     | (Ex #j. RecvSid(tid,sid)@j & #j<#i)"

```

Note that no freshness condition on the receiving thread is necessary. We are simply saying that the client always remembers what it generated for the session ID, and the server always remembers what it received as the session ID. The above lemma has been made slightly simpler by merging the two cases.

```

1 lemma tid_invariant [use_induction,reuse]:
2   "All tid sid role #i. Tid(tid,sid,role)@i==>
3     (Ex #j. InitTid(tid,sid,role)@j & #j<#i)"

```

We prove the `tid_invariant` lemma by trace induction, taking care of each case by backwards-unfolding the appropriate state facts. The secrecy proofs themselves are located in the *proofs-unilat.spthy* and *proofs-mutual.spthy* theory files.

5.6 Formal analysis of TLS 1.3, rev 10

While the results from the previous sections of this chapter are valuable in assessing the quality of earlier versions of the TLS 1.3 specification, they are especially convenient as a building block for tackling the verification of later revisions. In this section, we detail our work on our Tamarin model of TLS 1.3, rev 10 [88]. The corresponding Tamarin input files can be found at [45].

Some time after our three-month effort of studying and modelling, and one month of verifying (mid January to mid May 2015) what will later become revision 06 of the TLS 1.3 specification draft, we started to work jointly with Scott and van der Merwe, who were both Mozilla employees at the time, to produce a much more elegant and flexible model of the revision 10 specification. The bulk of this work took three months (mid July to mid October 2015) to complete. In this time, the model files were updated to conform to TLS 1.3, revision 10, many new and desirable properties were modelled, and a large number of man-hours was allocated to proving the properties and automating their proofs. A seemingly cosmetic, but helpful change was to use the more powerful m4 macro processor instead of the C preprocessor.

5.6.1 Differences between rev 06 and rev 10

The revision 10 specification draft of TLS 1.3 is very different from revision 06, with the most significant change having occurred in revision 07 [85]: two different handshake

modes were added. The first mode is essentially a flexible resumption replacement, implemented in the proposal via Pre-Shared Keys (PSKs). PSKs can be used just like session keys to authenticate a connection. There are two possible sources of PSKs: tickets and out-of-band mechanisms. While the former is specified in [88], the latter has not yet been entirely clarified in terms of intended implementation or assumed security properties, which surely depend on the context. We model out-of-band PSKs with a rule `Gen_PSK` that simply generates two nonces, which represent the PSK ID and the PSK itself, adds them to both the client and server state, and shows the PSK in the observable trace:

```

1 rule Gen_PSK:
2   [ Fr(~psk_id), Fr(~rs)]
3   --[ GenPSK($S)
4     , GenPSK($C)
5     , FreshPSK(~rs)
6   ]->
7   [ ClientPSK($C, ~psk_id, $S, ~rs)
8     , ServerPSK($S, ~psk_id, $C, ~rs)
9   ]

```

The second mode, a 0-RTT mode called Known Configuration (KC), includes an OPTLS-based option where clients can use a server-provided, semi-ephemeral (EC)DH key share to encrypt early/0-RTT application data.

In revision 08, the ciphers were updated, some extensions in the Hello messages were merged, server-side signatures were mandated even when KC is used, and some minor changes were included, such as tweaks of message contents [86]. While the subsequent revision 09 [87] contained another major change, namely the key computation has been completely redone, the other changes in revisions 09 and 10 [88] are mostly irrelevant to our analysis.

5.6.2 State machine and Tamarin model changes

Our state machines for revision 10 can be viewed in Figure 5.7. Note that the `C_1_KC_Auth` and `S_2_Auth` edges may occur at most once per handshake. Also, the figure lacks some states and transitions where the client initiates or the server responds to a new PSK or KC handshake. We list the omissions here:

- `C_2→C_1_KC`
- `C_2_KC→C_1_KC`
- `S_1→S_1_KC`

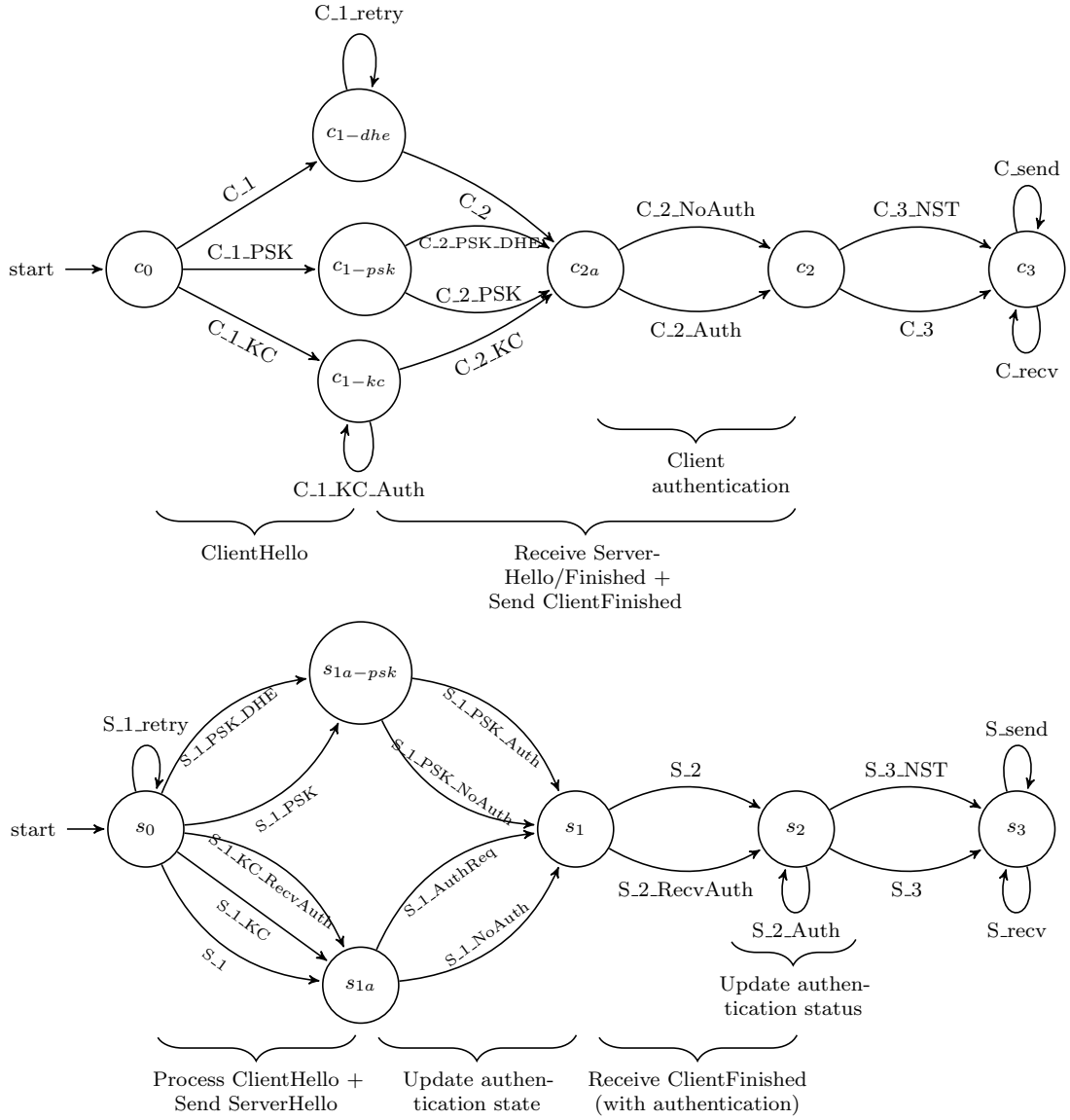


Figure 5.7: Simplified client and server state machines for TLS 1.3, rev 10

- $S_1_KC \rightarrow S_1_KC$
- $S_1 \rightarrow S_1_KC_RecvAuth$
- $S_1_KC \rightarrow S_1_KC_Recvauth$
- $C_3_NST \rightarrow C_1_PSK$
- $S_3_NST \rightarrow S_1_PSK$
- $S_3_NST \rightarrow S_1_PSK_DHE$

	Security property	Lemma(s)
(a)	Unilateral authentication (server)	entity_authentication
(b)	Mutual authentication	mutual_entity_authentication
(c)	Total anonymity	N/A
(d)	Confidentiality of session keys	secret_session_keys
(e)	Perfect forward secrecy	secret_session_keys
(f)	Integrity of handshake messages	transcript_agreement mutual_transcript_agreement
(g)	Protection of application data	N/A
(h)	Confidentiality of early data keys	secret_early_data_keys

Table 5.1: TLS 1.3, rev 10 properties as Tamarin lemmas

We note that while the most important model excerpts for our presentation are given in this section, all of the details on the exact implementation of our state machines can be seen in the *tls13* directory of our archive available at [45].

Table 5.1 shows how we map some of the required security properties in rev 10 to Tamarin lemmas. Note that the properties are still being discussed on multiple levels, from both the user and technical perspectives. We conjecture that some properties that are not explicitly required by the current specification, such as the confidentiality of early data keys, will appear in the final version, so we add them to the table. The security requirement of total anonymity will likely be removed from the specification, so we do not specify it at all. The mentioned security guarantees regarding application data, especially 0-RTT data in the newly introduced KC mode, such as confidentiality, integrity, and replay protection, are also not currently specified, and are hoped to be provided by the switch to AEAD algorithms. This needs to be verified by explicitly modelling AEAD in the future, when we might add new features such as KC expiration to our model. While the transcript agreement lemmas ensure agreement on all cryptographic values, we currently do not have a lemma that captures session key confirmation.

The Tamarin lemmas indicated in Table 5.1 are specified in the *properties.m4* file as shown in Figure 5.8. The *entity_authentication* lemma states the following: if a client sent *nc* and received *ns* respectively as the ClientHello and ServerHello random values, then the intended server indeed received *nc* and sent *ns* as the mentioned values, or the adversary revealed a long-term key of the intended server. The lemma *mutual_entity_authentication* provides an analogous guarantee to the server while ignoring AKC attacks (cf. Chapter 3).

Lemma *secret_session_keys* includes the forward secrecy requirement, and states that for each client thread whose session key is computed by the adversary, the

```

1 lemma entity_authentication:
2   "All actor peer nonces #i.
3     CommitNonces(actor, peer, 'client', nonces)@i ==>
4     (Ex #j. RunningNonces(peer, actor, 'server', nonces)@j & #j < #i) |
5     (Ex #r. RevLtk(peer)@r)"
6
7 lemma mutual_entity_authentication:
8   "All actor peer nonces #i.
9     CommitNonces(actor, peer, 'server', nonces)@i ==>
10    (Ex #j. RunningNonces(peer, actor, 'client', nonces)@j & #j < #i) |
11    (Ex #r. RevLtk(actor)@r) |
12    (Ex #r. RevLtk(peer)@r)"
13
14 lemma secret_session_keys:
15   "All actor peer k #i #j.
16     SessionKey(actor, peer, 'client', k)@i &
17     KU(k)@j ==>
18     (Ex #r. RevLtk(peer)@r & #r < #i)"
19
20 lemma transcript_agreement:
21   "All actor peer transcript #i.
22     CommitTranscript(actor, peer, 'client', transcript)@i ==>
23     (Ex peer2 #j. RunningTranscript(peer, peer2, 'server', transcript)@j & #j < #i) |
24     (Ex #r. RevLtk(peer)@r)"
25
26 lemma mutual_transcript_agreement:
27   "All actor peer transcript #i.
28     CommitTranscript(actor, peer, 'server', transcript)@i ==>
29     (Ex #j. RunningTranscript(peer, actor, 'client', transcript)@j & #j < #i) |
30     (Ex #r. RevLtk(actor)@r) |
31     (Ex #r. RevLtk(peer)@r)"
32
33 lemma secret_early_data_keys:
34   "All actor peer k #i #j.
35     EarlyDataKey(actor, peer, 'client', k)@i &
36     KU(k)@j ==>
37     (Ex #r. RevLtk(peer)@r)"

```

Figure 5.8: Tamarin model excerpt with lemmas from Table 5.1

adversary must have revealed a long-term key of the intended server before the client thread computed its session key. The transcript_agreement lemma states that when a client successfully computes a handshake hash in order to authenticate, the intended server computed the same handshake hash earlier, or the adversary must have revealed a long-term key of the server. The mutual version gives a similar guarantee to the server while ignoring AKC attacks. The secret_early_data_keys lemma is similar to the secret_session_keys lemma, with the difference that its focus is on early data keys and forward secrecy is not taken into account.

5.6.3 Analysis and proof automation

We prove the main results with respect to a powerful attacker very similar to the one in our previous analysis, with the single difference that we leave Diffie-Hellman reveal considerations to future work. We again prove the key secrecy results incrementally,

starting with the intermediate Diffie-Hellman secrets. Our authentication proofs are based on the combination of the established properties of the initial handshake, and the forward propagation of the PSK and KC values. For example, if the PSK or PSK ID sent to a client within an encrypted session ticket is subsequently received by the server, it acts as a challenge-response mechanism for client (re)authentication purposes in the resumed handshake.

While we have not yet done any proofs related to application data, we expect some of the security properties that hold for 1-RTT data to extend to 0-RTT data, depending of course on the possible strengthening of our threat model by compromising PSKs or KC values in the future. Note that 0-RTT application data is not forward secret because it is only protected with keys derived from the server’s semi-static (EC)DH share.

In order to facilitate proof automation, we decided to use a very flexible refactoring of the property files. In the new *aux* directory, we keep auxiliary-lemma include files (extension *.m4i*). All of the auxiliary lemmas are defined as macros and labelled with the reuse flag, so that they can be effortlessly included where needed to prove actual security properties, e.g. in *properties.m4*. When the command *make proofs* is run from the command line, the Makefile first ensures that all of the auxiliary lemmas are proved, and then used to complete the remaining automatic proofs.

Our approach to proof automation combines the described refactoring, and the ability of m4 to temporarily redefine macros in order to customise per-lemma heuristics. The selected heuristics influence the ranking of proof goals that Tamarin uses during verification. One possible way to achieve this in Tamarin is by introducing *F_* and *L_* prefixes to facts that we would respectively like Tamarin to rank first and last.

However, for some proofs this is not fine-grained enough. Another way to change the Tamarin heuristics, which we found especially useful in automating the proof of a version of the auxiliary *es_basic* lemma, is to use strings over the alphabet {s, S, c, S} (smart and consecutive ranking, where uppercase denotes that loop-breakers are allowed; loop-breakers by definition are constraints which occur infinitely often in every infinite sequence of solving premise constraints, cf. [69, 95] for details). This feature of Tamarin allows one to modify the heuristics at a certain proof depth. In other words, every goal ranking at depth *n* is done according to the character in the string at index *n*. More details on this method are given in the Tamarin manual distributed with the Tamarin prover.

As stated earlier in this section, it is not clear which kinds of security properties TLS 1.3 ought to provide against which kinds of attackers; in the rev 10 draft [88], the

properties are not precisely specified and the whole Security Analysis part (Appendix D) is scheduled to be rewritten. Additionally, some desirable features are not supported by rev 10, including a more flexible client authentication mechanism. We dedicate the next section to the analysis of one proposal for such a mechanism.

5.7 Post-handshake client authentication

While rev 10 does not include the possibility of delayed, or post-handshake, certificate-based client authentication, we extended our model as specified in one of the proposals for this intended functionality [80]. By enabling unrestricted post-handshake client authentication, where a signature over the handshake hash as defined in rev 10 is included, we find a client impersonation attack by using Tamarin. We note that the attack as described here is for the delayed authentication setting, but it can be adapted to cover client authentication as part of the handshake.

5.7.1 Attack on client authentication in PSK mode

Our attack is depicted in Figure 5.9. Alice plays the role of the victim client, and Bob the role of the targeted server. Charlie is an active man-in-the-middle adversary, whom Alice believes to be a legitimate server. The attack can be reproduced using our code at [45].

The attack proceeds in three main steps, each involving a different kind of TLS handshake.

Step 1: Establish legitimate PSKs

In the first stage of the attack, Alice starts a connection with Charlie, and Charlie starts a connection with Bob. In both connections, a PSK is established. At this point, both handshakes are computed honestly. Alice shares a PSK denoted PSK_1 with Charlie, and Charlie shares a PSK denoted PSK_2 with Bob.

Note that Charlie ensures the PSK identifier, psk_id , is the same across both sessions by replaying the value obtained from Bob.

Step 2: Resumption with matching freshness

In the next step, Alice wishes to resume a connection with Charlie using PSK_1 . As usual, Alice generates a random nonce nc , and sends it together with the PSK identifier, psk_id .

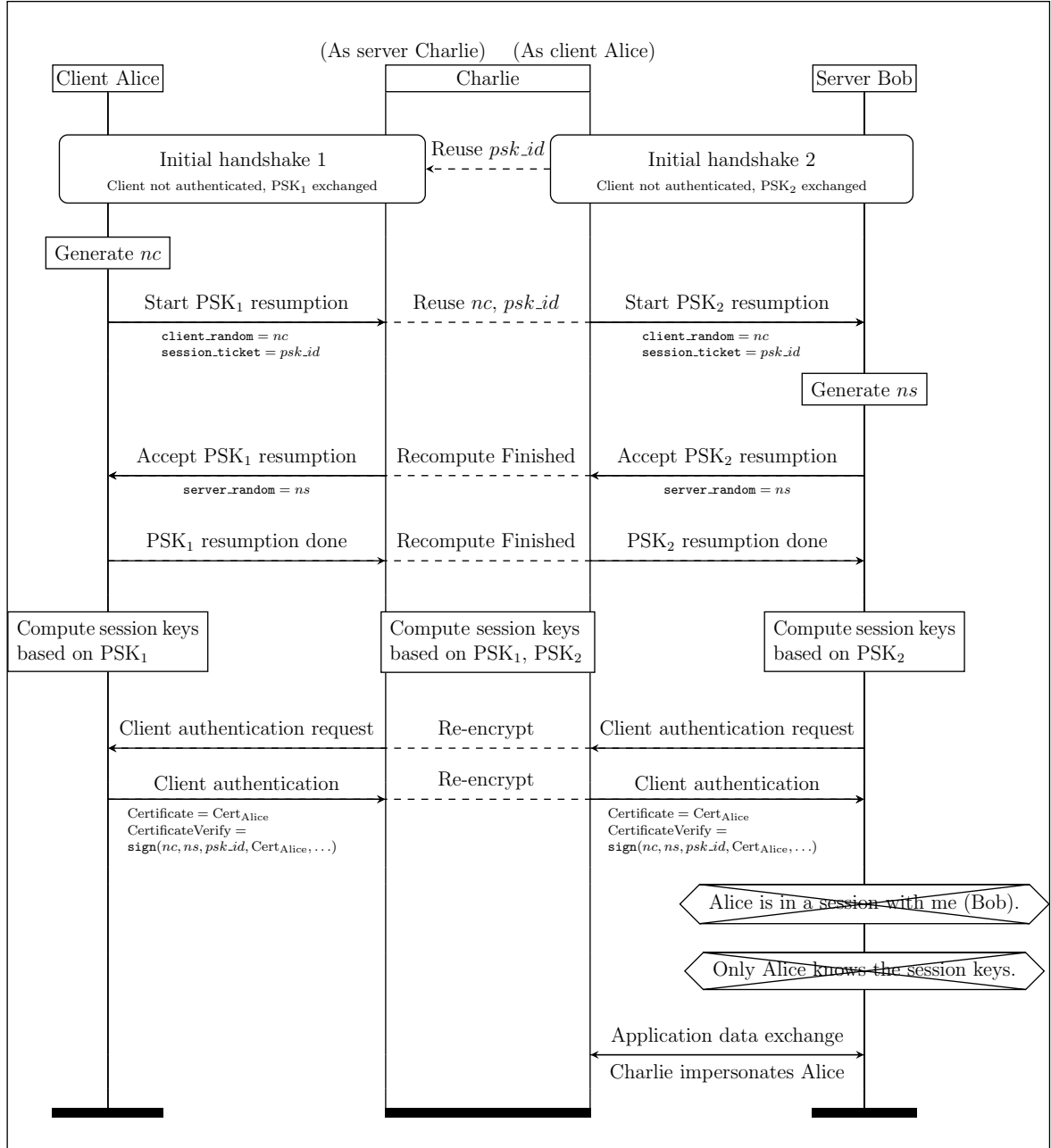


Figure 5.9: Client impersonation attack on TLS 1.3 rev 10 if delayed client authentication allowed in PSK mode

Charlie re-uses the value nc to initiate a PSK-resumption handshake with Bob, using the same identifier, psk_id . Bob responds with a random nonce ns , and the server Finished message computed by using PSK_2 .

Charlie now re-uses the nonce ns , and recomputes the server Finished message using PSK_1 . Alice returns her Finished message to Charlie, who recomputes it using PSK_2 and sends the resulting message to Bob.

At this point, Alice and Charlie share session keys, i.e. application traffic keys, derived from PSK_1 , and Charlie and Bob share session keys derived from PSK_2 . Note that the keys that Charlie respectively shares with Alice and with Bob are distinct.

Step 3: Delayed client authentication

Following the resumption handshake, Charlie requests a sensitive resource from Bob². The request calls for client authentication, so Charlie is subsequently prompted for his certificate and verification. Charlie re-encrypts this request for Alice³.

Alice then sends her public-key certificate and certificate verification message to Charlie. To compute the verification signature, Alice uses the `session_hash` value, which is defined as the hash of all handshake messages excluding Finished messages. In particular, the session hash contains her certificate, along with the values nc , ns , and psk_id . Notice that, if Charlie chooses to transmit the certificate of Alice instead of his own to Bob, the described session hash will match the one of Charlie and Bob. Therefore, Alice's signature will also be accepted by Bob.

Charlie proceeds as above by simply re-encrypting Alice's message for Bob, who accepts it as valid authentication for Charlie. By doing this, Charlie successfully impersonates Alice in his session with Bob, and even has full knowledge of the session keys. This enables Charlie to impersonate Alice in future communication with Bob, allowing him to read messages that Bob intends for Alice, fake messages for Bob that supposedly come from Alice, and/or access resources that are normally unavailable to Charlie, but are available to Alice. Thus, the attack completely breaks client authentication.

5.7.2 Underlying cause and mitigation

The above attack is possible due to an inadequate binding between an established session and the subsequent connections to the session. While the server authenticates in the initial handshake and the server certificate is included (transitively, via the

²This is one of the main use cases for the delayed client authentication mode [80].

³Charlie can respond to any incoming request from Alice by asking her to authenticate.

PSK) in the Finished messages during the PSK resumption step, the server is not transitively authenticated postresumption; the Finished messages can be recomputed by the man-in-the-middle attacker, and clients never sign them in rev 10 [88].

In parallel to our analysis, the TLS Working Group has proposed several modifications to rev 10 in the move towards rev 11 [89]. One of these proposals is PR#316 [83], which explicitly allows delayed client authentication, but takes a different approach from [80]. PR#316 additionally redefines the client signature based on a new Handshake Context value, which includes the server Finished message. While this new definition certainly thwarts our attack, it also appears to resolve the issue because the adversary would need to force the server Finished messages to match across the two sessions. However, these messages are bound to public-key certificates of different servers.

5.8 Conclusions

We analyse the TLS 1.3 Handshake and Record Protocols, as specified in revision 06 [84] of its IETF draft, with respect to a powerful, active attacker. We model the protocols in a similar way to an earlier symbolic model of TLS 1.2 we constructed, for which we designed a state machine according to the protocol specification in RFC 5246 [28]. We use the Tamarin prover to verify that both the unilaterally and mutually authenticated modes of TLS 1.3, rev 06 satisfy session key secrecy with respect to an unbounded number of threads and a Dolev-Yao style adversary capable of revealing different kinds of cryptographic keys and fresh values. We also verify modern security requirements such as Key Compromise Impersonation Resilience and (strong) Perfect Forward Secrecy.

We build on our revision 06 model and work with Cremers, Scott, and van der Merwe to model and verify the required authenticated key exchange properties in revision 10 of the TLS 1.3 draft. We find a potential vulnerability in an unrestricted version of the desired mechanism of post-handshake client authentication, which directly impacts the way this mechanism is introduced in revision 11. Our detailed Tamarin models enable the rapid prototyping and security analysis of the upcoming changes to the protocol, so we expect our work to further influence the current and future development of TLS 1.3.

A recommendation we would like to make to the TLS working group is to include a fully fleshed out state machine and precise security requirements into the final TLS

1.3 specification. This would be a welcome addition for the mutual benefit of people looking to implement, use, and do research on TLS 1.3.

Chapter 6

Related work

6.1 Actor Key Compromise

The vast majority of related work has been on Key Compromise Impersonation in the domain of key establishment protocols. Just and Vaudenay [51] published in 1996 the first supposed KCI attack, but it was later retracted [16]. However, a year later, actual KCI attacks and the first explicit, but informal definition of this notion were given by Blake-Wilson, Johnson and Menezes [13]. These papers consider an adversary who obtains long-term secret keys of a party, usually referred to as *the actor*, who is running a key establishment protocol. The adversary uses the keys to establish a session key as another protocol participant with the actor, without being detected. This results in a session key known to the adversary, which is therefore useless for securing subsequent communication.

Following [13, 51], researchers have examined concrete protocols or small classes of protocols, and classified KCI attacks [17, 18, 39, 96, 99]. This has led to a partial understanding of KCI. However, determining if an attack is a KCI attack is still done on a per-case basis, guided by minor variations in the early informal definitions.

Starting with [104], researchers have considered the compromise of the actors' keys in computational models [39, 56, 104] and proved KCI resilience for concrete protocols. The underlying idea is that the absence of KCI attacks can be proved without formally stating what constitutes a KCI attack, as long as it is informally argued that the absence is implied by KCI resilience in a monolithic computational security model. Conversely, attacks are informally argued to be KCI attacks on a case-by-case basis.

Examples of KCI attacks on one-pass key establishment protocols can be found in [17, 18], where they are classified as one of two types, depending on whether the responder authenticates the initiator. In [39], KCI attacks are either “insider attacks” or “outsider attacks”, depending on whether the adversary actively participates in the

execution of a protocol on behalf of a party whose long-term keys have been revealed. We assume that every AKC-capable adversary can actively use any keys it reveals and make no distinction between these two types of KCI attacks. A third classification is outlined in [96], where KCI attacks by adversaries with a session key reveal capability are classified according to how that capability is used in the attack. These attacks would be captured in our model by adding the session key reveal capability from [6].

In [97], the fact that derivability of session keys from long-term secrets and public values of just one party can be a source of insecurity is demonstrated by KCI attacks on four protocols. It is argued that session keys should be derived from long-term secrets and, ideally, run-specific data of another party.

A KCI attack by an adversary with a randomness reveal capability against the 3-pass HMQRV protocol is shown in [99]. The protocol is fixed by adding a confirmation message consisting of a signed hash of both ephemeral and long-term public values. However, since the adversary can get the actor's randomness and its long-term secret key, the adversary can compute all session keys. Therefore, their fixed protocol does not provide any security guarantees for the subsequent communications with respect to their adversary model.

In [50], it is proven that the DHE modes of TLS satisfy a monolithic security notion that implies KCI resilience. This is in line with our findings. In contrast, it is proven in [55] that all modes of TLS are secure in a weaker security model. This weaker security model does not capture AKC attacks; the proofs therefore do not contradict our AKC attacks on TLS. Note that Paulson's simplified model of TLS [79] is vulnerable to an additional AKC attack on authentication where the adversary can replace the client's nonce with an arbitrary value to make the client's and server's views of all session keys diverge. The reason for this is that in Paulson's simplified version, the hash in the client's `CertificateVerify` message does not contain all previously transmitted data. However, this is not an actual attack on the TLS protocol.

In April 2014, a critical vulnerability [77] (CVE-2014-0160, also known as Heartbleed) was discovered in OpenSSL, versions 1.0.1–1.0.1f. The vulnerability allowed an attacker to retrieve parts of an affected server's memory, potentially including the server's long-term secret keys. If the attacker obtained the keys, it could impersonate the compromised server. As we have shown, the attacker could additionally perform an AKC attack on the server using TLS-RSA. By upgrading to a Heartbleed-unaffected implementation of TLS-RSA such as OpenSSL 1.0.1g, servers could prevent further key leakage and simply switch to using new keys. OpenSSL servers that did not use

TLS-RSA, and instead opted for TLS-DHE_RSA, were not vulnerable to the above attack.

6.2 ISO/IEC 11770

In 1998, Horng and Hsu presented an attack on an early version of the 3-KT-6 protocol [44]. This version contained no identity I_B of B in the second message, which enabled an attack similar to the 1995 attack by Gavin Lowe on the Needham-Schroeder Public Key protocol [61]. The attack on this version of 3-KT-6 violated key confirmation and showed that the protocol did not offer any strong mutual authentication. In the same year, Mitchell and Yeun proposed a fix [72] that was later introduced in the standard. They essentially performed Lowe’s Needham-Schroeder fix by adding I_B to the second message.

In 2004, Cheng and Comley presented two attacks on a previous version of the 2-12 protocol [20]. Their first attack is a replay attack that depends on compromising session keys of threads not under attack, and the fact that random or sequence numbers are used where timestamps would be appropriate. Cheng and Comley fixed the protocol by replacing the used sequence numbers with timestamps.

A second attack is possible even when timestamps are used. It is a type-flaw attack based on the possibility of interpreting an identity field as a fresh key. The protocol was fixed by cryptographically binding the two parts of the second message (the second part became part of the payload encrypted to form the first part).

Initially, protocol 2-12 was withdrawn from the standard, but it was later updated in 2008 with a new version that did not suffer from these attacks. This new version is replay-protected by tagging with constants [46, p. 17], so that a mixup of messages can no longer occur. Since the type-flaw attack was also essentially a replay attack, it was automatically prevented as well.

Mathuria and Sriram used Scyther to discover in 2008 [64] more complex type-flaw attacks on a modified version of protocol 2-13 and on Cheng’s and Comley’s proposed fixed protocol. The attacks relied on the possibility that complex fields (concatenations, encryptions) could be interpreted as atomic fields (random values, keys, identities) in some implementations. While the first attack does apply to the 2-13 protocol itself, the second one does not apply to the updated version of 2-12, because this version of 2-12 is not based on fixes from [20].

In 2010, Chen and Mitchell [19] generalised some of the concepts occurring in this class of type-flaw attacks and presented countermeasures, some of which found their

way into later versions of ISO standards. They called the generalised phenomenon *parsing ambiguity attacks* and showed how many of these attacks could be found in the then current versions of ISO/IEC 11770 and ISO/IEC 9798. We continued their work by systematically analysing all the protocols in Parts 2 and 3 of the current version of the ISO/IEC 11770 standard.

6.3 TLS 1.3

There have been many points in the development of TLS 1.3 with major, sweeping changes done to the protocol specification. We would first like to comment on an initially promising parallel proposal for TLS 1.3 that was later merged with the main proposal. Then we point out the previous analyses of TLS 1.2, which is the direct predecessor of TLS 1.3.

Hugo Krawczyk sent a proposal [53] of his own to the TLS mailing list on 1 November 2014, sketching a protocol that he developed with Hoeteck Wee. The protocol was to be called OPTLS, where OPT stands for both OPTimised and One-Point-Three, and it differed from the then current TLS 1.3 proposal in that it used static Diffie-Hellman instead of signatures for entity authentication. The idea was to use an existing server signature key offline to authenticate a number of static DH half keys, instead of using it every time a new session is established—the signature key itself could even be destroyed and a fresh one generated for the next batch of offline signatures.

The application keys in the protocol were a combination of the ephemeral DH secret g^{xy} , and the values g^{xs} and g^{yc} where c and s are respectively static DH keys of the client and the server. This made them less vulnerable to State Reveal. The most highly advertised characteristic of OPTLS was to seamlessly support 0-RTT. This was done by employing the value g^{xs} to encrypt 0-RTT application data, which was to be sent as part of the client’s first handshake message.

The TLS working group has ultimately decided to proceed with the original TLS 1.3 proposal, but the interest to revisit OPTLS has been rekindled on multiple occasions, mostly for its 0-RTT support. Krawczyk’s announced, detailed write-up of the since modified OPTLS has not been completed as of September 2015, but has been stated by Krawczyk [54] that “it is in the same spirit of OPTLS but it is not signature-less, i.e. semi-static keys for the servers are now signed with online signatures that prove freshness by including the client’s nonce under the signature”. Since that

time, Krawczyk has collaborated with the TLS working group to add the most useful features of OPTLS to the main TLS 1.3 proposal.

Two of the most prominent analyses of TLS 1.2 are by Jager et al. [49] and by Krawczyk et al. [55]. In the former, Jager et al. introduce the notion of Authenticated and Confidential Channel Establishment or ACCE security. They prove that TLS as a protocol with two subprotocols, namely the Handshake and Record Protocols, satisfies this new notion of security. They examine mutually authenticated modes of TLS 1.2, including TLS_DHE. In the latter paper, by Krawczyk et al., the unilateral version of TLS is proven secure under a security definition derived from ACCE (called SACCE, where S stands for Server-only). The paper focuses on the unilateral modes and RSA PKCS #1 v1.5 encryption due to their wide deployment.

The threat models in the above two papers are not directly comparable to ours: while theirs are stronger by being less abstract and allowing, e.g. bit manipulation, they are also weaker by disallowing State Reveal before the end of the attacked thread. Note that we basically allow the attacker to reveal all the secrets of threads not under attack or partners to the attacked thread. This restriction is necessary because knowing only one of the two DH exponents involved in computing a PMS value implies knowledge of the PMS, and therefore of all session keys.

Chapter 7

Conclusions

After having presented the results of the work we carried out in the area of formalisation, modelling, and verifying standardised security protocols, we are now in a position to answer the research questions we formulated in Chapter 1.

Research question 1: What security guarantees can an entity expect if its own long-term keys are compromised, and how can security protocols help achieve these guarantees?

From our experience of talking to security researchers who have never heard of, or at least never investigated Key Compromise Impersonation (KCI), the usual initial response to the first part of this question is “none”.

However, as we have shown in Chapter 3, where we provided the first systematic analysis of Actor Key Compromise (AKC), AKC resilience and security can be achieved by either vulnerability prevention in the incremental protocol design stage, or by later making custom, tool-guided tweaks in the protocol deployment stage. These security features are desirable because they help contain the effects of key compromise [77, 81].

We have also shown that authentication guarantees under AKC can never be based on symmetric cryptography and hashing alone, which has direct consequences for improving existing protocols and developing new protocols. We used this result in Chapter 4 to instantaneously establish that 13 standardised key exchange protocols and their variants are insecure under AKC.

For many existing protocols, we have introduced efficient, unintrusive fixes that take advantage of the protocol structure or the alternative protocol modes, and result in AKC-secure protocols. For example, the most straightforward AKC fix for both the mutually and unilaterally authenticated TLS-RSA protocols, even when combining the latter with authorisation protocols, is to use TLS-DHE_RSA.

In our second research question, we considered the possible harm that modern adversaries, including AKC-capable ones, can inflict by exploiting security protocol vulnerabilities in widely deployed protocols.

Research question 2: How do advanced adversaries, i.e. ones who can compromise all kinds of short-term and long-term secret values, impact security protocols in modern security standards, and how can we use formal approaches to improve them?

Our research in the area of security protocol standardisation mostly focused on the ISO/IEC 11770 standard for key management techniques (cf. Chapter 4), and different versions and revisions of TLS specifications (cf. Chapters 3 and 5). The choice of advanced adversaries that we looked at was a result of a whole range of different usage scenarios and security contexts for these protocols, where the possible threats vary widely.

Security standards, as well as other standards, clearly benefit from periodic updates because their provisions are meant to cover at least the foreseeable future. In security standards, such updates ought to cover strong security guarantees that are impenetrable to newly emerging threats. However, the ISO/IEC 11770 standard does not claim resilience to UKS or KCI attacks in Parts 2 and 3, even when UKS can be easily prevented at negligible cost, and most of the protocols are not vulnerable to KCI and can replace the ones that are. Although Part 4 of the standard received some attention in practice as well as in research recently [41, 42], the standard as a whole may have been overlooked due to a lack of a comprehensive, formal security analysis.

While the ISO/IEC 11770 standard includes statements about the security guarantees intended to be achieved by its protocols, which is unfortunately rare for a standard, there are some claims that our automated analysis showed to be incorrect. The most surprising fact is that many of the mistakes we found were old ones, and they could have been caught earlier if the working group responsible for the standard had been aware of the 2012 ISO/IEC 9798 analysis [8]. An especially bad example is the 3-KA-11 protocol, which supposedly offers key authentication and confirmation for both parties despite being a stripped-down version of the TLS 1.2 unilaterally authenticated RSA handshake.

The ISO/IEC 11770 working group is releasing an updated version of the standard based on our suggestions. Regrettably, the only way for people who are not working-group members to access this and other ISO standards is by purchasing the final version. We therefore hope that this and other working groups will decide to adopt our recommendation to promote the external analysis of ISO standards by publishing

early drafts of proposed changes or new standards free of charge. While this would help identify and prevent problems before the standards are deployed, the parties that are interested in applying the standards will still need to purchase the final versions to ensure they comply.

In our analysis of the TLS 1.3 Handshake and Record Protocols (cf. Chapter 5), as specified in revision 06 [84] of their IETF draft, we considered a powerful, active attacker who can reveal almost all the secret information involved in protocol executions (cf. Section 5.5). We based our rev 06 model on an earlier symbolic model of TLS 1.2 that we constructed according to RFC 5246 [28]. We verified with the Tamarin prover that both the unilaterally and mutually authenticated modes of TLS 1.3, rev 06 satisfy session key secrecy with respect to an unbounded number of threads, including modern security requirements such as Key Compromise Impersonation Resilience and (strong) Perfect Forward Secrecy.

The flexibility of the Tamarin prover allows us to rapidly prototype and analyse any current and future changes made to TLS 1.3, which tend to be both frequent and sudden. We updated our revision 06 model to a revision 10 [88] model with Cremers, Scott, and van der Merwe. We used the model to verify the standard authenticated key exchange requirements in TLS 1.3, rev 10. When we enabled unrestricted post-handshake client authentication according to one of the proposals for this desired functionality [80], we found a client impersonation attack that effected an update of rev 11 [89].

Chapter 8

Future work

While our analysis so far has been successful in influencing the development of multiple security protocol standards, we consider the following future extension of our work to be an important step toward the further proper specification and verification of the upcoming TLS 1.3 revisions.

Firstly, we want to review the required TLS 1.3 security properties with the help of other TLS research groups and the TLS working group, especially the properties concerning the new 0-RTT and PSK modes, e.g. entity authentication when a PSK is established out of band. This will lead to a much more uniform focus across different groups, while the benefit of following different approaches will be retained.

Secondly, we would like to adapt our Tamarin model accordingly, while refining it to support AEAD, KC expiration, and corruption of PSKs and KC values as mentioned in Section 5.6. Enlarging the attack surface by refining formal models can lead to discovering further practical attacks, which in turn leads to strengthening their design. We believe that our model represents a valuable contribution to the TLS 1.3 analysis effort in this regard.

Thirdly, we plan to promptly include the changes introduced by new TLS 1.3 revisions in our model. We can then verify each revision with Tamarin by repeating our typical workflow as described in Section 5.5 and Section 5.6, and help guide the design of TLS 1.3 until its deployment.

It is clear that every standard, including the ones analysed in this thesis, is a continually moving target, and the work associated with improving its quality does not necessarily end even when it is fully deployed. Specifically, in security standards, users always want additional features, and potential adversaries never cease to grow more powerful.

Luckily, the number of research groups dedicated to analysing security standards, and the willingness of working groups to establish a feedback loop with them, seem

to be rising. The combination of diverse techniques employed by the research groups and the wealth of experience gathered by the working groups together help shape the future of security protocol standards in a scientific manner. While the formal methods experts bridge the gap between current and future designs by efficiently breaking abstract models of the intended communication in security protocols, the cryptography experts analyse the protocols in depth to come up with more solid designs and resulting realistic proofs of security.

We hope that the fruitful collaboration described above will continue, and that the essential parts of security protocol standards will be thoroughly analysed prior to deployment. The results can then be included in final specifications to officially and unambiguously cover the intended executions, and to the greatest possible extent the desirable security guarantees and the conceivable attack scenarios.

Bibliography

- [1] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security (CCS)*, pages 5–17, 2015.
- [2] N. J. AlFardan, D. J. Bernstein, K. G. Paterson, B. Poettering, and J. C. N. Schuldt. On the security of RC4 in TLS. In *Proceedings of the 22th USENIX Security Symposium*, pages 305–320, 2013.
- [3] N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P, Oakland)*, pages 526–540, 2013.
- [4] Apache Software Foundation. mod_auth_basic – Apache HTTP Server Version 2.2. Apache HTTP Server Project. http://httpd.apache.org/docs/2.2/mod/mod_auth_basic.html.
- [5] E. B. Barker, D. Johnson, and M. E. Smid. SP 800-56A. Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised). Technical report, Gaithersburg, MD, United States, 2007.
- [6] D. Basin and C. Cremers. Modeling and analyzing security in the presence of compromising adversaries. In *Proceedings of the 15th European Symposium on Research in Computer Security (ESORICS)*, volume 6345 of *Lecture Notes in Computer Science*, pages 340–356. Springer, 2010.
- [7] D. Basin, C. Cremers, and M. Horvat. Actor key compromise: Consequences and countermeasures. In *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF)*, pages 244–258, 2014.

- [8] D. Basin, C. Cremers, and S. Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. In *Proceedings of the 1st International Conference on Principles of Security and Trust (POST)*, volume 7215 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2012.
- [9] D. A. Basin, C. J. F. Cremers, K. Miyazaki, S. Radomirovic, and D. Watanabe. Improving the security of cryptographic protocol standards. *IEEE Security & Privacy*, 13(3):24–31, 2015.
- [10] L. Bello. CVE-2008-0166, 2008. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0166>.
- [11] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue. A messy state of the union: taming the composite state machines of TLS. In *Proceedings of the 36th IEEE Symposium on Security & Privacy (S&P, Oakland)*, pages 535–552, 2015.
- [12] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Pironti, and P. Strub. Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In *Proceedings of the 35th IEEE Symposium on Security and Privacy (S&P, Oakland)*, pages 98–113, 2014.
- [13] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *Proceedings of the 6th IMA International Conference on Cryptography and Coding*, pages 30–45, 1997.
- [14] S. Blake-Wilson and A. Menezes. Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol. In *Proceedings of the 2nd International Workshop on Practice and Theory in Public Key Cryptography (PKC)*, pages 154–170, 1999.
- [15] C. Boyd, C. Cremers, M. Feltz, K. G. Paterson, B. Poettering, and D. Stebila. ASICS: Authenticated Key Exchange Security Incorporating Certification Systems. In *Proceedings of the 18th European Symposium on Research in Computer Security (ESORICS)*, volume 8134 of *Lecture Notes in Computer Science*, pages 381–399. Springer, 2013.
- [16] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment*. Springer, 2003.

- [17] K. Chalkias, F. Mpaldimtsi, D. Hristu-Varsakelis, and G. Stephanides. On the key-compromise impersonation vulnerability of one-pass key establishment protocols. In *Proceedings of the International Conference on Security and Cryptography (SECRYPT)*, pages 222–228. Springer, 2007.
- [18] K. Chalkias, F. Mpaldimtsi, D. Hristu-Varsakelis, and G. Stephanides. Two types of key-compromise impersonation attacks against one-pass key establishment protocols. In *Proceedings of the International Conference on Security and Cryptography (SECRYPT)*, pages 227–238. Springer, 2008.
- [19] L. Chen and C. J. Mitchell. Parsing ambiguities in authentication and key establishment protocols. *International Journal of Electronic Security and Digital Forensics*, 3(1):82–94, 2010.
- [20] Z. Cheng and R. Comley. Attacks on an ISO/IEC 11770-2 key establishment protocol. Cryptology ePrint Archive, Report 2004/249, 2004.
- [21] K. Cohn-Gordon and K. Milner. iButton Hackery, 2014. <http://blog.katriel.co.uk/ibutton-hackery>.
- [22] C. Cremers. *Scyther – Semantics and Verification of Security Protocols*. PhD dissertation, Eindhoven University of Technology, 2006.
- [23] C. Cremers. The Scyther Tool: Verification, falsification, and analysis of security protocols. In *Proceedings of the 20th International Conference on Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.
- [24] C. Cremers and M. Feltz. Beyond eCK: perfect forward secrecy under actor compromise and ephemeral-key reveal. *Designs, Codes and Cryptography*, 74(1):1–36, 2013.
- [25] C. Cremers and M. Horvat. AKC protocol models. <http://www.cs.ox.ac.uk/people/cas.cremers/scyther/AKC/>.
- [26] C. Cremers and M. Horvat. Improving the ISO/IEC 11770 standard for key management techniques. In *Proceedings of the 1st International Conference on Security Standardisation Research (SSR)*, volume 8893 of *Lecture Notes in Computer Science*, pages 215–235. Springer, 2014.

- [27] C. Cremers and S. Mauw. *Operational Semantics and Verification of Security Protocols*. Information Security and Cryptography. Springer, 2012.
- [28] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) protocol version 1.2. IETF RFC 5246, August 2008. <http://tools.ietf.org/html/rfc5246>.
- [29] W. Diffie, P. C. Van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.
- [30] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 350–357. IEEE Computer Society, 1981.
- [31] European Payments Council. Guidelines on algorithms usage and key management, 2009. EPC342-08 Version 1.1.
- [32] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 34–39. IEEE Computer Society, 1983.
- [33] fail0verflow. Console Hacking 2010: PS3 Epic Fail. 27th Chaos Communication Congress, 2010.
- [34] S. Farrell. Transport Layer Security Working Group Charter, February 2014. <https://datatracker.ietf.org/doc/charter-ietf-tls/04-02/>.
- [35] M. Feltz and C. Cremers. On the limits of authenticated key exchange security with an application to bad randomness. Cryptology ePrint Archive, Report 2014/369, 2014.
- [36] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) protocol version 3.0. IETF RFC 6101, August 2011. <http://tools.ietf.org/html/rfc6101>.
- [37] F. D. Garcia, G. Koning Gans, R. Muijers, P. Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling MIFARE Classic. In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS)*, volume 5283 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2008.

- [38] I. Goldberg and D. Wagner. Randomness and the Netscape browser. *Dr. Dobbs's Journal*, 1996.
- [39] M. Gorantla, C. Boyd, and J. G. Nieto. Modeling key compromise impersonation attacks on group key exchange protocols. In *Proceedings of the 12th International Workshop on Practice and Theory in Public Key Cryptography (PKC)*, pages 105–123, 2009.
- [40] P. Gutmann. Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). IETF RFC 7366, September 2014. <https://tools.ietf.org/html/rfc7366>.
- [41] F. Hao. J-PAKE adopted by ISO/IEC standard, 2014. <https://blogs.ncl.ac.uk/security/2014/10/24/j-pake-adopted-by-isoiec-standard>.
- [42] F. Hao and S. F. Shahandashti. The SPEKE Protocol Revisited. Cryptology ePrint Archive, Report 2014/585, 2014.
- [43] D. Hardt. The OAuth 2.0 Authorization Framework. IETF RFC 6749, January 2012. <http://tools.ietf.org/html/rfc6749>.
- [44] G. Horng and C.-K. Hsu. Weakness in the Helsinki protocol. *Electronics Letters*, 34:354–355(1), February 1998.
- [45] M. Horvat. Thesis file archive. <https://www.cs.ox.ac.uk/people/marko.horvat/archive.tar.bz2>.
- [46] International Organization for Standardization, . ISO/IEC 11770-2:2008, Information technology – Security techniques – Key Management – Part 2: Mechanisms using Symmetric Techniques, 2008. Incorporating corrigendum September 2009.
- [47] International Organization for Standardization, . ISO/IEC 11770-3:2008, Information technology – Security techniques – Key Management – Part 3: Mechanisms using Asymmetric Techniques, 2008. Incorporating corrigendum September 2009.
- [48] International Telecommunication Union. Recommendation ITU-T X.509 : Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks, October 2012.

- [49] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. Cryptology ePrint Archive, Report 2011/219, 2011.
- [50] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk. On the security of TLS-DHE in the standard model. In *Advances in Cryptology – CRYPTO 2012 – 32nd Annual Cryptology Conference*, volume 7417 of *Lecture Notes in Computer Science*, pages 273–293. Springer, 2012.
- [51] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT)*, pages 36–49. Springer, 1996.
- [52] H. Krawczyk. HMQV: A high-performance secure Diffie-Hellman protocol. Cryptology ePrint Archive, Report 2005/176, 2005.
- [53] H. Krawczyk. OPTLS: Signature-less TLS 1.3. TLS mailing list, November 2014. <http://article.gmane.org/gmane.ietf.tls/15142>.
- [54] H. Krawczyk. Re: OPTLS: Signature-less TLS 1.3. TLS mailing list, March 2015. <https://www.ietf.org/mail-archive/web/tls/current/msg15682.html>.
- [55] H. Krawczyk, K. G. Paterson, and H. Wee. On the security of the TLS protocol: A systematic analysis. In *Advances in Cryptology – CRYPTO 2013 – 33rd Annual Cryptology Conference*, volume 8042, pages 429–448, 2013.
- [56] B. LaMacchia, K. Lauter, and A. Mityagin. Stronger security of authenticated key exchange. In *Proceedings of the 1st International Conference on Provable Security (ProvSec)*, volume 4784 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2007.
- [57] A. Langley. Transport Layer Security (TLS) Snap Start. draft-agl-tls-snapstart-00, June 2010. <https://tools.ietf.org/html/draft-agl-tls-snapstart-00>.
- [58] A. Langley. Imperialviolet blog, December 2014. <https://www.imperialviolet.org/2014/12/08/poodleagain.html>.

- [59] A. Langley, W. Chang, N. Mavrogiannopoulos, J. Strombergson, and S. Josefsson. The ChaCha20-Poly1305 AEAD Cipher for Transport Layer Security, draft-ietf-tls-chacha20-poly1305-00, June 2015. <https://tools.ietf.org/html/draft-ietf-tls-chacha20-poly1305-00>.
- [60] L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, 28(2):119–134, 2003.
- [61] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1995.
- [62] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of the 2nd International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer, 1996.
- [63] G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Workshop on Computer Security Foundations (CSFW)*, pages 31–44. IEEE Computer Society, 1997.
- [64] A. Mathuria and G. Sriram. New attacks on ISO key establishment protocols. Cryptology ePrint Archive, Report 2008/336, 2008.
- [65] Maxim Integrated Products, Inc. 1-Wire®Extended Network Standard, 2006. <http://pdfserv.maximintegrated.com/en/an/AN3925.pdf>.
- [66] Maxim Integrated Products, Inc. Serial Number iButton, 2010. <http://datasheets.maximintegrated.com/en/ds/DS1990R-DS1990R-F5.pdf>.
- [67] D. McGrew. An Interface and Algorithms for Authenticated Encryption. IETF RFC 5116, January 2008. <http://tools.ietf.org/html/rfc5116>.
- [68] G. H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.
- [69] S. Meier. *Advancing automated security protocol verification*. PhD dissertation, ETH Zurich, 2013.

- [70] S. Meier, B. Schmidt, C. Cremers, and D. Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In N. Sharygina and H. Veith, editors, *Proceedings of the 25th International Conference on Computer Aided Verification (CAV)*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.
- [71] A. Menezes and B. Ustaoglu. Security arguments for the UM key agreement protocol in the NIST SP 800-56A standard. In M. Abe and V. D. Gligor, editors, *Proceedings of the 3rd ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 261–270. ACM, 2008.
- [72] C. J. Mitchell and C. Y. Yeun. Fixing a problem in the Helsinki protocol. *SIGOPS Operating Systems Review*, 32(4):21–24, 1998.
- [73] B. Moeller, T. Duong, and K. Kotowicz. This POODLE Bites: Exploiting the SSL 3.0 Fallback, 2014. <https://www.openssl.org/~bodo/ssl-poodle.pdf>.
- [74] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
- [75] K. Nohl, D. Evans, S. Starbug, and H. Plötz. Reverse-engineering a Cryptographic RFID Tag. In *Proceedings of the 17th USENIX Security Symposium*, pages 185–193. USENIX Association, 2008.
- [76] OASIS Security Services Technical Committee. Security Assertion Markup Language v2.0. <https://www.oasis-open.org/standards#samlv2.0>.
- [77] OpenSSL Security Advisory [07 Apr 2014]. TLS heartbeat read overrun (CVE-2014-0160). https://www.openssl.org/news/secadv_20140407.txt.
- [78] OpenVPN Technologies. Facts about OpenVPN. <https://openvpn.net/index.php/about-menu/openvpn-facts.html>.
- [79] L. C. Paulson. Inductive analysis of the Internet Protocol TLS. *ACM Transactions on Information and System Security*, 2(3):332–351, 1999.
- [80] A. Popov. TLS 1.3 client authentication. In *Proceedings of the IETF-93 Workshop*, 2015. <https://www.ietf.org/proceedings/93/slides/slides-93-tls-2.pdf>.

- [81] K. Poulsen. Edward Snowdens E-mail provider defied FBI demands to turn over crypto keys, documents show, October 2013. http://www.wired.com/threatlevel/2013/10/lavabit_unsealed/.
- [82] E. Rescorla. 0-RTT and Anti-Replay, March 2015. <http://www.ietf.org/mail-archive/web/tls/current/msg15594.html>.
- [83] E. Rescorla. TLS 1.3 specification pull request: Wip client auth revision #316, 2015. <https://github.com/tlswg/tls13-spec/pull/316/>.
- [84] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, draft-ietf-tls-tls13-06, April 2015. <https://tools.ietf.org/html/draft-ietf-tls-tls13-06>.
- [85] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, draft-ietf-tls-tls13-07, July 2015. <https://tools.ietf.org/html/draft-ietf-tls-tls13-07>.
- [86] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, draft-ietf-tls-tls13-08, August 2015. <https://tools.ietf.org/html/draft-ietf-tls-tls13-08>.
- [87] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, draft-ietf-tls-tls13-09, October 2015. <https://tools.ietf.org/html/draft-ietf-tls-tls13-09>.
- [88] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, draft-ietf-tls-tls13-10, October 2015. <https://tools.ietf.org/html/draft-ietf-tls-tls13-10>.
- [89] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3, draft-ietf-tls-tls13-11, December 2015. <https://tools.ietf.org/html/draft-ietf-tls-tls13-11>.
- [90] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [91] P. Rogaway. Formalizing human ignorance: Collision-resistant hashing without the keys. In *Proceedings of the 1st International Conference on Cryptology in*

- Vietnam (VIETCRYPT)*, volume 4341 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2006.
- [92] J. Salowey, A. Choudhury, and D. McGrew. AES Galois Counter Mode (GCM) Cipher Suites for TLS. IETF RFC 5288, August 2008. <https://tools.ietf.org/html/rfc5288>.
 - [93] L. Schmid. *Improving the ISO/IEC 11770 standard*. BSc thesis, ETH Zurich, 2013.
 - [94] B. Schmidt. *Formal analysis of key exchange protocols and physical protocols*. PhD dissertation, ETH Zurich, 2012.
 - [95] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF)*, pages 78–94. IEEE, 2012.
 - [96] K. Shim. The risks of compromising secret information. In *Proceedings of the 4th International Conference on Information and Communications Security (ICICS)*, pages 122–133, 2002.
 - [97] M. Strangio. On the Resilience of Key Agreement Protocols to Key Compromise Impersonation. In *Proceedings of the 3rd European Public Key Infrastructure Workshop (EuroPKI)*, volume 4043 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2006.
 - [98] A. Stubblefield, J. Ioannidis, and A. D. Rubin. Using the Fluhrer, Mantin, and Shamir attack to break WEP. In *Proceedings of the 9th Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2002. <http://www.isoc.org/isoc/conferences/ndss/02/proceedings.shtml>.
 - [99] Q. Tang and L. Chen. Extended KCI attack against two-party key establishment protocols. *Information Processing Letters*, 111(15):744–747, 2011.
 - [100] Trustworthy Internet Movement. SSL Pulse, October 2015. <https://www.trustworthyinternet.org/ssl-pulse/>.
 - [101] S. Vaudenay. Security Flaws Induced by CBC Padding – Applications to SSL, IPSEC, WTLS.... In *Proceedings of the 21st Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*,

- volume 2332 of *Lecture Notes in Computer Science*, pages 534–546. Springer, 2002.
- [102] T. Weigold and A. Hiltgen. Secure confirmation of sensitive transaction data in modern Internet banking services. In *Proceedings of the World Congress on Internet Security (WorldCIS)*, pages 125–132, 2011.
 - [103] T. Ylonen. The Secure Shell (SSH) Transport Layer Protocol. IETF RFC 4253, January 2006. <http://tools.ietf.org/html/rfc4253>.
 - [104] R. Zhu, X. Tian, and D. Wong. Enhancing CK-model for key compromise impersonation resilience and identity-based key exchange. Cryptology ePrint Archive, Report 2005/455, 2005.

Index

- $(\cdot)^{\#rid}$ function, 14
- AK , adversary knowledge, 14
- GS , set of ground substitutions, 14
- $[\cdot/\cdot, \cdot/\cdot, \dots]$, substitution, 10
- Agent, set of agent names, 8
- Claim, set of claim names, 10
- Const, set of constants, 8
- Func, set of function names, 8
- LKR, 11
- LTK(), set of long-term secret keys, 11
- Label, set of labels, 10
- Fresh, set of freshly generated terms, 8
- Π function, 12
- RID, set of run identifiers, 8
- Role, set of roles, 8
- Var, set of variables, 8
- $\sqsubseteq_{acc}$, accessible subterm relation, 9
- rid_A , adversary run identifier, 8
- $cont(\cdot)$, contents of a term, 11
- create**, 11
- Event, set of events, 11
- ev**, event type, 11
- RunTerm₀, set of ground run terms, 10
- k , long-term symmetric key, 9
- label($\cdot$), label of a term, 11
- $\downarrow$ operator, 11
- $Test$, test run identifier, 8
- PRF, pseudo-random function, 85
- $\models$ relation, 17
- pk , long-term public key, 9
- RoleEvent, set of role events, 11
- RunEvent, set of run events, 11
- σ function, 14
- $\sqsubseteq$, syntactic subterm relation, 9
- State, set of states, 13
- τ , tagging function, 34
- Term, set of terms, 8
- TA , agreement transformation, 37
- KC , key confirmation transformation, 31
- TS , secrecy transformation, 36
- TraceEvent, set of trace events, 11
- upto operator, 11
- $\vdash$ relation, 10
- $\{\cdot/\cdot\}$, replacement, 10
- th function, 14
- 0-RTT, 81
- actor key compromise (AKC), 21
 - attack, 24
 - resilience, 26
 - security, 24
- adversary knowledge, 14
- AEAD, 80, 84, 87, 98
- agent, 8
- at least as strong security claim, 41

- claim, 10
 - alive, 10
 - commit, 10
 - running, 10
 - secret, 10
- composition rule, 10
- contents of a term, 11
- decomposition rule, 10
- derivation tree, 10
 - height, 10
- Encrypt-then-MAC, 88
- encryption, 9
- entity authentication, 57, 60
- event
 - LKR, 11
 - create, 11
 - claim, 11
 - recv, 11
 - send, 11
 - role event, 8
 - run event, 8
- event type, 11
- forward secrecy, 57, 60
 - (strong) perfect (PFS), 48, 61
 - weak perfect (wPFS), 62
- ground substitution, 14
- handshake master secret (HMS), 85
- Heartbleed, 107
- initial adversary knowledge, 14
- initial state, 14
- ISO/IEC
 - 9798, 52
 - 11770, 52
- key agreement, 54
- key authentication, 57, 60
- key compromise impersonation (KCI),
 - 23, 62, 71
 - resilience, 27
- key confirmation, 57, 60
- key derivation function (KDF), 56
- key establishment, 54
- key management, 52
- key transport, 54
- Known Configuration (KC), 96
- label, 10
- long-term key, 9
- MAC-Then-Encrypt, 80
- master secret (MS), 85
- Mealy state machine, 89
- mechanism, *see* protocol, 54
 - 3-KA-11, 56
- message sequence chart (MSC), 55
- nonce, 8
- OpenSSL, 107
- OPTLS, 80, 109
- pairing, 9
- Pre-Shared Key (PSK), 96
- premaster secret, 57
- premaster secret (PMS), 85
- protocol, 12
 - 2-11, 68
 - 2-12, 55
 - 3-KA-1, 72
 - 3-KA-11, 67, 68
 - 3-KA-3, 72
 - 3-KA-6, 72

- 3-KA-8, 72
- 3-KT-6, 70
- ITU-T X.509, 42
- Needham-Schroeder (NS), 27
- Needham-Schroeder-Lowe (NSL), 22
- NSL, 42
- NSL-AKC, 43
- SSH, 42
- SSL, 79
- TLS
 - 1.2, 42
 - 1.3, rev 06, 80, 93
 - 1.3, rev 10, 81, 95
 - Handshake Protocol, 81
 - Record Protocol, 81
 - Snap Start, 82
- replacement, 10
- resumption premaster secret (RMS), 85
- role, 8
- rules
 - adversary-compromise, 15
 - LKR_{actor} , 15
 - LKR_{others} , 15
 - execution-model, 15
 - `create`, 15
 - `claim`, 15
 - `recv`, 15
 - `send`, 15
 - Scyther, 14
 - Tamarin, 18
- run
 - adversary run, 8
 - test run, 8
- Scyther, 42, 59
- security claim, 17
 - alive, 17
 - at least as strong, 41
 - commit, 17
 - secret, 17
- session, 8
 - hash in TLS, 87
 - ID in TLS, 94
 - in TLS, 82
- state, 13
- substitution, 10
- subterm, 9
 - accessible, 9
 - syntactic, 9
- tagging function, 34
- Tamarin prover, 6, 18, 48, 87
- term, 8
 - ground run term, 10
 - role term, 8
 - run term, 8
- thread, 8
- trace, 14
- transformation
 - TA , 37
 - KC , 31
 - TS , 36
- transition relation, 16
- type flaw, 65
- type function, 12
- unknown key share (UKS), 60, 62
- well-formed sequence of role events, 12