An automated formal analysis of the security of the internet key exchange (IKE) protocol in the presence of compromising adversaries

Master Thesis

Author(s):
Kyburz, Adrian

Publication date:
2010

Permanent link:
https://doi.org/10.3929/ethz-a-006250423

AN AUTOMATED FORMAL ANALYSIS OF THE SECURITY OF THE INTERNET KEY EXCHANGE (IKE) PROTOCOL IN THE PRESENCE OF COMPROMISING ADVERSARIES

MASTER THESIS
DEPARTMENT OF COMPUTER SCIENCE
SWISS FEDERAL INSTITUTE OF TECHNOLOGY (ETH)
ZURICH

Supervision: Dr. Cas Cremers

Adrian Kyburz
November 2010

Abstract

Our dependence on sophisticated security services has largely increased in recent years. To address this trend, a lot of research effort has been put into the development and usage of methods for proving security properties of network protocols.

To date, many security protocols have already been analyzed by formal methods, using either symbolic computation or complexity theory. One important family of protocols which has only been analyzed in a very limited way is the IKE protocol family. IKE stands for Internet Key Exchange and is the default protocol suite for key management in the IPSec standard. Due to the sheer complexity of the IKE protocol specification, only simplified versions have been subject to formal analysis to date. Previously conducted analyses on IKE are limited to the individual analysis of some of the (many) subprotocols and many of them do not investigate more advanced security properties such as Perfect Forward Secrecy, Key Compromise Impersonation, or the loss of old session keys.

This thesis provides the first automatic security analysis of IKEv1 and IKEv2 in the presence of compromising adversaries. We scrutinize the protocols with respect to the loss of session keys, forward secrecy and Key Compromise Impersonation. We describe new attacks against the protocol suites and establish a security hierarchy among the various variants of the protocol. Moreover, our results demonstrate that a tool-supported formal analysis of large-scale security protocols is feasible.

Acknowledgements

This Master's thesis is the culmination of my studies at the Swiss Federal Institute of Technology. During the last several years I was given the great opportunity to not only learn about all the fascinating topics the area of Computer Science has to offer, but also to meet a myriad of interesting, and open-minded people who greatly enhanced my horizon.

It is thus a pleasure to thank the many people who made this thesis possible. First of all, I would like to thank Dr. Cas Cremers, my supervisor, for being a vibrant source of inspiration and guidance. Without his support and great experience in the area of security protocols and their analysis, this thesis would be very different.

I also would like to thank Prof. David Basin for being my supervising professor at ETH Zurich.

Although my studies consumed a large portion of my life, it would not have been complete without the hours and hours of sharing, laughing, partying, and traveling with my friends, most notably Benno Schildknecht, Christian Zeiler, Christof Rissi, Daniel Naeff, Dejan Juric, Dominique Kronenberg, Flavio Pfaffhauser, Jan-Filip Zagalak, Luciano Franceschina, Manuel Hess, Philip Reichen, Serge Gebhardt, Simon Brunner, Tobias Flueckiger, Urs Bitterlin, and last but not least, my flatmate Thomas Frei.

A very special word of gratitude goes to my parents, Thomas and Rita Kyburz, for showing me the light of the world and providing me this great privilege of education. Without their loving support I could not have achieved this degree, which I believe is only the start in a wonderful career.

Last but not least, I also want to thank you, dear reader, for the interest in my work. Would it not be for you, my effort put into this thesis would be in vain.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In todays networked environment, more and more people rely on key management and related security protocols to securely communicate with each other or carry out commercial transactions. Ever since the publication of *New Directions in Cryptography* by Diffie and Hellman [18], the development of innovative key management mechanisms has been a hot topic in the cryptographic research community, and new security protocols arise constantly to overcome the challenges of the "Internet Society". In spite of the mission critical services security protocols should provide, we rarely have correctness proofs for many of these relatively simple distributed programs.[1]

In light of this challenge, a lot of research effort has been put into the development and usage of methods for proving security properties of network protocols. This area of research has had two important foundations, one based on logic and symbolic computation, and one based on computational complexity theory. The symbolic approach, which uses a highly idealized representation of cryptographic primitives, has been a successful basis for formal logics and automated tools. Conversely, the computational approach yields more insight into the strength and vulnerabilities of protocols, but requires a lot more effort and is often difficult to apply.

To date, many security protocols have been analyzed by formal methods, using either symbolic computation or complexity theory. One important family of protocols which has only been analyzed in a very limited way is the IKE protocol family [22, 26]. IKE stands for Internet Key Exchange and is the default protocol suite for key management in the IPSec

[1] "Security protocols are three-line programs that people still manage to get wrong". This famous quote by the late Roger Needham clearly expresses the difficulty of designing robust and flawless security protocols.

standard.

1.1 Contributions

IPSec protocols have become of great interest in cryptographic research in recent years and some previously established provable security results already found its way into the IPSec RFCs. However, it appears that few design choices therein have been made because of a firm theoretical basis. Thus, IPSec – and in particular IKE – presents many interesting challenges and surely deserves a formal analysis.

Due to the sheer complexity of the IKE protocol specification, only simplified versions have been subject to formal analysis to date [13, 14, 36, 37, 40]. Previously conducted analyses on IKE are limited to individual subprotocol of IKE and many of them do not investigate more advanced security properties such as Perfect Forward Secrecy (PFS), Key Compromise Impersonation (KCI), or the loss of old session keys. But a full treatment of such properties is necessary because they capture the essence of Murphy's law; secret information stored at a party is potentially vulnerable to break-ins or other forms of leakage and therefore, in some cases, will be leaked. It is thus important to classify protocols according to their behavior in the advent of leakage of some form of secret information. Ideally, such leakage has the *least possible effect* on the security of other secrets. For example, we want to guarantee that the leakage of information specific to one session (such as the leakage of a session key or ephemeral state information) will have no effects on the security of other sessions (known-key attacks), or that the leakage of crucial long-term secrets (such as private keys) that are used across multiple sessions will not necessarily compromise secret information from all past sessions (Perfect Forward Secrecy). Compromising adversaries, i. e., adversaries which are able to dynamically compromise protocol participants during protocol execution, are one way to model such behavior.

In this thesis, we analyze the behavior of IKE in the advent of compromising adversaries and present the following contributions to the security analysis of IKE:

1. We formally describe IKE version 1 (IKEv1) and version 2 (IKEv2) in a symbolic framework which has recently been developed by Basin and Cremers [7].

2. We use Scyther [15] to perform a fully automated security analysis of IKEv1 and IKEv2 with respect to the secrecy of session keys and two forms of authentication.

By this, we demonstrate that a tool-supported formal analysis of large-scale security protocols is feasible.

3. We describe new attacks against both protocol suites, some of them previously overlooked, some of them arising from the more sophisticated adversary model which comes with the framework.

4. We establish a security hierarchy as proposed in [6] among the numerous variants of IKEv1 and IKEv2 protocols.

Moreover, we contribute to the further development of both the theory and the verification tool that support the analysis of complex and practically more relevant real-world security protocols.

1.2 Related Work

We briefly sketch related work here and we will discuss these works in more detail in Chapter 6 and 7. Most research has been performed on the initial version of IKE, although there exist studies on simplified versions of IKEv2. To the best of our knowledge, there exists no thorough analysis of IKEv2 yet.

Formal Analysis A first formal analysis of IKE has been conducted at the Naval Research Laboratory (USA). In her work [36], Meadows uses the NRL Protocol Analyzer, a special-purpose formal methods tool for the verification of cryptographic protocols, to scrutinize IKEv1 with respect to several security properties, such as secrecy and authentication. Her analysis uncovers several problems with the IKE specification. Most of these problems were not so much flaws in the protocol design itself, but rather ambiguities and omissions in the specification that could lead to insecure or incorrect implementations if the specification was not correctly understood. Meadows shows that several subprotocols of IKE fail to achieve a certain form of authentication, called *penultimate authentication*. She further shows that quick mode achieves Perfect Forward Secrecy but also finds quick mode vulnerable to a replay attack if the optional identities are omitted.

Canetti and Krawczyk use computational complexity theory to analyze IKEv1 under an adversary that is more powerful than a traditional Dolev-Yao attacker [13, 14]. Their security model, which they call SK-security, allows them to analyze protocol behavior in the

presence of compromising adversaries, i. e., adversaries which are capable of compromising protocol participants at runtime. In a first analysis, Canetti and Krawczyk show that several subprotocols of IKE are SK-secure in the "pre-specified peer" model, i. e., a model in which the protocol participants know their peer at the beginning of execution. A second analysis extends the initial security model ([13]) to the scenario where the identity of the peer is determined only *during* protocol execution ("post-specified peer" model). They then give a full security proof of IKE's digital signature variant with respect to SK-security within the relaxed model.

Roy, Datta and Mitchell, in [40], use the Protocol Composition Logic (PCL) to develop axioms for reasoning about protocols using Diffie-Hellman key exchange. They use the axiom system to formally prove authentication and secrecy theorems for digital signature authenticated IKEv2.

Mödersheim and Drielsma use the AVISPA tool [5] to model and verify the signature authenticated variant of IKEv2. They report [37] a similar authentication flaw as Meadows above and, as a consequence, suggest to slightly adapt the protocol by adding key confirmation.

Informal Analysis In their *Analysis of IKE* [39], Perlman and Kaufman suggest several modifications to improve IKEv1. They argue that both the public signature key main mode and pre-shared key main mode are vulnerable to an active attacker trying to discover the initiator's identity.

Ferguson and Schneier [20] conclude that IKEv1 authenticators are defined unacceptably weak. They present several reflection attacks against IKEv1 and additionally demonstrate how an adversary can manipulate IKE exchanges such that the two parties executing the protocol eventually agree on a security policy which may be considerably weaker than anticipated by the parties. The same weakness has also been pointed out by Zhou in [41]. Furthermore, Zhou argues that the property of identity concealment cannot be achieved in the main mode protocol with digital signature authentication, even though the identities are encrypted.

1.3 Organization

In Chapter 2, we present a brief introduction to IPSec and IKE before we introduce the symbolic security model, which we base this work on, in Chapter 3. In Chapter 4 we describe the Internet Key Exchange (IKE) protocols IKEv1 and IKEv2 in detail, and highlight the differences among the two versions. Chapter 5 outlines how we modeled the IKE protocol family in the formalism of Chapter 3. We present our experiments and the results thereof in Chapter 6 and discuss them in context of related work. In Chapter 7, we provide additional information about related topics and conclude the thesis in Chapter 8.

Chapter 2

IPSec and IKE

Originally, the Internet was designed without the specification of security requirements. Back then, its development was driven by the need of the scientific community for a global medium that allowed for open communication and resource sharing. However, its de-facto standardization as a global network, as well as its growth in popularity and commercial use made it evident that the Internet protocols sooner or later would have to be adapted to support security.

To date, there exist a number of security services for Internet users, the most popular ones being IPSec [27] and SSL/TLS [17]. Transport Layer Security (TLS) and its predecessor, Secure Socket Layer (SSL), are cryptographic protocols that build upon the traditional functionality of TCP to provide confidentiality and integrity. They are layered between TCP and the applications that use the protocol (e.g. web browsing, electronic mail, Internet faxing, instant messaging and voice-over-IP). IPSec on the other hand operates in layer 3 and thus is fully transparent to applications. It comprises numerous special-purpose protocols, one of which is the Internet Key Exchange (IKE) protocol. We describe IPSec and IKE below.

2.1 An Overview of IPSec

IPSec provides security at the IP network layer of the TCP/IP protocol stack. This means that all IP packets can be protected, no matter what payloads (of protocols running at upper layers) they carry. In contrast to TLS, there is no need to modify applications to take advantage of the security services provided by IPSec. Moreover, IPSec services can be

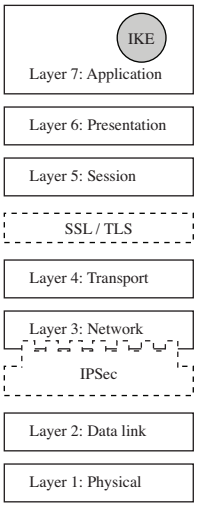

Figure 2.1: SSL/TLS, IPSec, and IKE in the OSI model

made fully transparent to end users. For these reasons, IPSec forms the basis of many virtual private networking (VPN) solutions where it is used to provide security for communications over an untrusted network such as the Internet.

The set of security services offered by IPSec includes access control, integrity protection, data origin authentication, replay detection and rejection, and confidentiality. The IPSec protocols also support automated key management (via ISAKMP, cf. Section 2.1.3), with key exchange protocols using both symmetric and asymmetric cryptographic techniques.

IPSec can be deployed in two basic modes: tunnel mode and transport mode. In tunnel mode, cryptographic protection is provided for entire IP packets. In essence, a whole packet is treated as the new payload of an outer IP packet and encapsulated therein. Tunnel mode is typically supported by security gateways which are located at e.g. corporate network borders. The use of gateways has the advantage that hosts inside the network need not be aware of IPSec. By contrast, in transport mode, the header of the original IP packet itself is preserved, only enriched by some security fields, and the payload together with some header fields undergo cryptographic processing. Transport mode is typically used when end-to-end services are needed. It provides protection mostly for the packet payload. In either mode, one can think of the IPSec infrastructure as intercepting normal IP packets and performing cryptographic processing on them before passing them on to the upper or lower layers, depending on whether inbound or outbound traffic is handled.

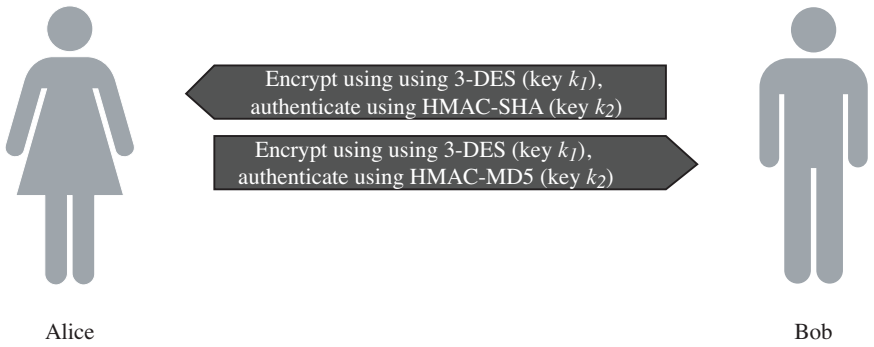

Figure 2.2: An example of a bidirectional SA between Alice and Bob which supports encryption and authentication: Alice and Bob both use the same encryption algorithm (3DES) with the key k_1. Alice authenticates herself using HMAC-MD5 with k_2, while Bob uses HMAC-SHA with k_2.

2.1.1 Security Associations

Security associations are a central concept in IPSec. They refer to unidirectional "connections" that offer security services to the traffic carried by them [27, section 4]. Each security association (SA) can support encryption, authentication, or both. The two peers on either side of the SA store (amongst other information) the cryptographic keys, encryption algorithms, authentication schemes, and integrity protection mechanisms supported by that connection, in the SA. Security associations are unidirectional: if Alice wants to setup a bi-directional secure communication channel with Bob, then at least two SAs are required (cf. Fig. 2.2). There exist two different types of security associations in IPSec: ISAKMP SAs and IPSec SAs. While the latter is used to protect IP packets, ISAKMP SAs are used to protect ISAKMP messages.

2.1.2 Security Policies

The protection offered by IPSec is based on requirements that are stored in a Security Policy Database (SPD). For every outbound IP packet, IPSec checks the SPD for finding "instructions" on how the packet is to be processed. If the packet needs protection, the SPD specifies the SAs to be used. Thus, IP packets are intercepted and compared to the SPD, each match with an SPD entry identifies a policy and a collection of SAs that implement the policy, and these SAs are then "applied" to the packets (cf. Fig. 2.3).

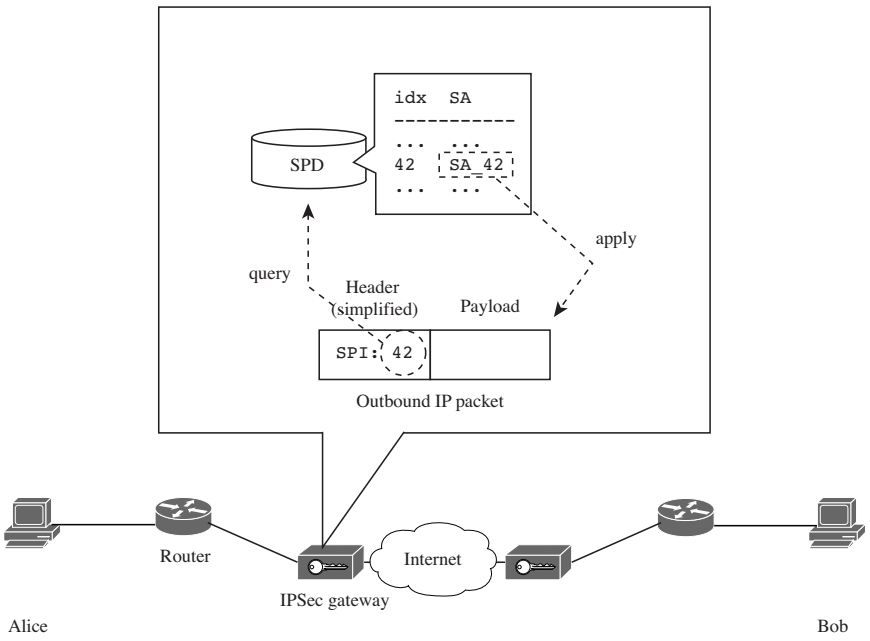

Figure 2.3: Schematic overview of applying a policy from the Security Policy Database to an outbound IP packet in tunnel mode

2.1.3 Key Management

To establish a SA, Alice and Bob must first agree on the cryptographic algorithms to be used and second, jointly determine keying material over a possibly insecure channel. Instead of negotiating keying material, cryptographic keys could also be deployed manually, this approach however only works well for small-scale deployments. For larger scale and more robust use of IPSec, an automated method is needed. The Internet Security Association and Key Management Protocol (ISAKMP) [34] provides methods for both SA negotiation and associated cryptographic parameter establishment. Although the "P" in its abbreviation suggests that ISAKMP is a protocol, it merely provides a framework for authentication and key exchange and is designed to be independent of key exchange protocols.

2.2 An Overview of IKE

An instance of ISAKMP and the default IPSec method for secure key negotiation is the Internet Key Exchange (IKE) protocol. Other key negotiation protocols may be used, but IKE is mandatory in all IPSec implementations and is the most common protocol that is

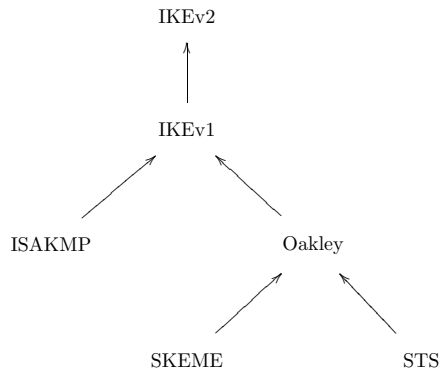

Figure 2.4: Schematic evolvement of IKE

currently used. Unlike the rest of IPSec, which resides at, or just below the network layer, IKE is an application-layer protocol (cf. Fig. 2.1).

The aim of the protocol is to dynamically establish shared state for the provision of security services between two parties in the network, for example between a client and a server. As such, it does much more than simply distribute keys; it also establishes security associations which, among other features necessary for secure communication, specify the protocol format and the cryptographic and hashing algorithms used. Because flexibility was one of the design goals, IKE supports a number of different authentication schemes, including digital signatures, public key encryption, and conventional encryption using shared keys.

IKE has evolved from a number of different protocols, including ISAKMP [34], Oakley [38], the Station-to-Station (STS) protocol [19], and SKEME [28]; the last two of which influenced the development of Oakley. IKE can be thought of as the result of combining the ISAKMP packet formats with the message exchange modes and key management aspects defined by the Oakley protocol. The current version of IKE, IKEv2 [26] supersedes the original version specified in [22]. We examine both versions in Chapter 4.

Chapter 3

Symbolic Security Model

In this chapter we summarize the symbolic framework from [7] which is the basis of our analysis. We explain how security protocols can be modeled and how their properties can be expressed within that model. We illustrate how the various security notions from literature are covered by the framework and give a reason for their existence. Readers familiar with [7] may skip this chapter.

Notational preliminaries Let f be a function. We write $dom(f)$ and $img(f)$ to denote f's domain and image. We write $f[b \hookleftarrow a]$ to denote f's update, i.e., the function f' where $f'(x) = b$ when $x = a$ and $f'(x) = f(x)$ otherwise. We write $f : X \nrightarrow Y$ to denote a partial function from X to Y. For any set S, $\mathcal{P}(S)$ denotes the power set of S and S^* denotes the set of finite sequences of elements from S. We write $\langle s_0, \ldots, s_n \rangle$ to denote the sequence of elements s_0 to s_n, and we omit brackets when no confusion can result. For s a sequence of length $|s|$ and $i < |s|$, s_i denotes the i-th element in the sequence. We write $s \hat{} s'$ for the concatenation of the sequences s and s'. Abusing set notation, we write $e \in s$ iff $\exists i.s_i = e$. We write $union(s)$ for $\bigcup_{e \in s} e$. We define $last(\langle\rangle) = \emptyset$ and $last(s \hat{} \langle e \rangle) = e$.

We write $[t_0, \ldots, t_n / x_0, \ldots, x_n] \in \mathcal{S}ub$ to denote the substitution of t_i for x_i, for $0 \leq i \leq n$. We extend the functions dom and img to substitutions. We write $\sigma \cup \sigma'$ to denote the union of two substitutions, which is defined when $dom(\sigma) \cap dom(\sigma') = \emptyset$, and write $\sigma(t)$ for the application of the substitution σ to t. Finally, for R a binary relation, R^* denotes its reflexive transitive closure.

3.1 Protocols and Their Execution

3.1.1 Protocol Model

We use a simple protocol description language to present a *protocol* as a set of "programs", one for each *role* such as initiator or responder. Each role program is a sequence of protocol actions, called *events*, to be executed by an honest participant, referred to as an *agent*. Formally, a protocol defines a partial mapping from role names to event sequences, i.e., $P : Role \nrightarrow ProtocolEvent^*$, where $Role$ is the (infinite) set of roles and $ProtocolEvent$ denotes the set of events every protocol participant can engage in. Protocols are performed by agents who execute role programs, thereby instantiating role names with agent names. Agents may execute any role multiple times, possibly in parallel. Each instance of a role is called a *thread* and is identified by a thread identifier *tid*.

We assume given the inifinte sets of $Agent, Fresh, Var, Func$, and TID of agent names, roles, freshly generated terms (nonces, session keys, coin flips, etc.), variables, function names, and thread identifiers. We assume that TID contains two distinguished thread identifiers, $Test$ and $tid_{\mathcal{A}}$. These identifiers single out a distinguished "point of view" tread of an arbitrary agent and and an adversary thread, respectively.

Definition 1 (Protocol events)**.**

$$\begin{aligned} ProtocolEvent ::= {} & \mathsf{create}(Role, Agent) \mid \mathsf{send}(Term) \mid \mathsf{recv}(Term) \\ & \mid \mathsf{generate}(\mathcal{P}(Fresh)) \mid \mathsf{state}(\mathcal{P}(Term)) \mid \mathsf{sessionkeys}(\mathcal{P}(Term)) \end{aligned}$$

The event $\mathsf{create}(Role, Agent)$ denotes role instantiation. The next two events are standard and model role communication. The last three events are used to tag runtime information, e.g., the fresh values generated by a thread, the current state of a thread, and the set of session keys computed by a thread.

Definition 2 (Terms)**.**

$$\begin{aligned} Term ::= {} & Agent \mid Role \mid Fresh \mid Fresh \sharp TID \mid Var \sharp TID \\ & \mid (Term, Term) \mid pk(Term) \mid sk(Term) \mid k(Term, Term) \\ & \mid \{\!|\, Term \,|\!\}^{a}_{Term} \mid \{\!|\, Term \,|\!\}^{s}_{Term} \mid Func(Term^*) \end{aligned}$$

For each $X, Y \in Agent$, $sk(X)$ denotes the long-term private key, $pk(X)$ denotes the

long-term public key, and $k(X, Y)$ denotes the long-term symmetric key shared between X and Y. Moreover, $\{\!|\, t_1 \,|\!\}^a_{t_2}$ denotes the asymmetric encryption (for public keys) or the digital signature (for signing keys) of the term t_1 with the key t_2, and $\{\!|\, t_1 \,|\!\}^s_{t_2}$ denotes symmetric encryption. The set $Func$ is used to model other cryptographic functions, such as hash functions. Freshly generated terms and variables are assumed to be local to a thread. To bind a term t to a protocol role instance with thread identifier tid, we write $t\sharp tid$.

Example 1 (Challenge-Response protocol)**.** *Let $\{I, R\} \subseteq Role, ni \in Fresh$, and $x \in Var$. We define the simple protocol CR as follows.*

$$\begin{aligned} CR(I) = \langle\, &\mathsf{generate}(\{ni\}), \mathsf{state}(\{ni\}), \mathsf{send}(I, R, ni), \\ &\mathsf{recv}(R, I, \{\!|\, I, ni \,|\!\}^a_{sk(R)})\rangle \\ CR(R) = \langle\, &\mathsf{recv}(I, R, ni), \mathsf{state}(\{x, \{\!|\, I, x \,|\!\}^a_{sk(R)}\}), \\ &\mathsf{send}(\{\!|\, I, x \,|\!\}^a_{sk(R)})\rangle \end{aligned}$$

In this protocol, the initiator generates a nonce and sends it along with the initiator and responder names to the network. The responder excepts to receive a message of this form and subsequently computes the signature of the received value and the initiator's name. R responds with the computed signature. The additional events mark state information. The state information is implementation-dependent and marks the parts of the state that is stored at a protection level lower than the long-term private keys. The state information in CR corresponds to, e.g., implementations that use a hardware security module (HSM) for encryption and signing and perform all other computations in ordinary memory.

Depending on the protocol analyzed, the framework assumes that symmetric or asymmetric long-term keys have been distributed prior to protocol execution. Further, the existence of an inverse function on terms is assumed, where t^{-1} denotes the inverse key of t. We have that $pk(X)^{-1} = sk(X)$ and $sk(X)^{-1} = pk(X)$ for all $X \in Agent$, and $t^{-1} = t$ for all other terms t.

Inference of terms To denote that the term t can be inferred from the set of terms M a binary relation $\vdash$ is defined. Let $t_0, \ldots, t_n \in Term$ and let $f \in Func$. Then $\vdash$ is the

smallest relation satisfying:

$$\begin{aligned} t \in M \Rightarrow M \vdash t \quad & M \vdash t_1 \wedge M \vdash t_2 \Leftrightarrow M \vdash (t_1, t_2) \\ M \vdash \{\!|\, t_1 \,|\!\}^a_{t_2} \wedge M \vdash t_2^{-1} \Rightarrow M \vdash t_1 \quad & M \vdash t_1 \wedge M \vdash t_2 \Leftrightarrow M \vdash \{\!|\, t_1 \,|\!\}^a_{t_2} \\ M \vdash \{\!|\, t_1 \,|\!\}^s_{t_2} \wedge M \vdash t_2 \Rightarrow M \vdash t_1 \quad & M \vdash t_1 \wedge M \vdash t_2 \Leftrightarrow M \vdash \{\!|\, t_1 \,|\!\}^s_{t_2} \\ & \bigwedge_{0 \le i \le n} M \vdash t_i \Rightarrow M \vdash f(t_o, \ldots, t_n) \end{aligned}$$

The term t' is a subterm of t, written $t' \sqsubseteq t$, if t' is a syntactic subterm of t, e.g., $t_1 \sqsubseteq \{\!|\, t_1 \,|\!\}^s_{t_2}$. $FV(t)$ denotes the set of free variables in t, where $FV(t) = \{t' \mid t' \sqsubseteq t\} \cap (Var \cup \{v \sharp tid \mid v \in Var \wedge tid \in TID\})$.

Threads An agent may execute multiple threads in different roles from various protocols. To distinguish between the fresh terms and variables of each thread, we assign them unique names, using the framework's auxiliary function $localize : TID \to \mathcal{S}ub$, defined as

$$localize(tid) = \bigcup_{cv \in Fresh \cup Var} [cv \sharp tid / cv].$$

Additionally, the function $thread : (ProtocolEvent^* \times TID \times \mathcal{S}ub \to ProtocolEvent^*$ yields the sequence of events that may occur in an agent's thread.

Definition 3 (Thread)**.** *Let l be a sequence of events, $tid \in TID$, and let σ be a substitution. Then $thread(l, tid, \sigma) = \sigma(localize(tid)(l))$.*

Example 2. *Let $\{A, B\} \subseteq Agent$. For a thread $t_1 \in TID$ performing role I from Example 1, we have $localize(t_1)(ni) = ni \sharp t_1$ and*

$$\begin{aligned} & thread\,(CR(I), t_1, [A, B/I, R]) = \\ & \quad \langle \mathsf{generate}(\{ni \sharp t_1\}), \mathsf{state}(\{ni \sharp t_1\}), \mathsf{send}(A, B, ni \sharp t_1), \\ & \quad \mathsf{recv}(B, A, \{\!|\, A, ni \sharp t_1 \,|\!\}^a_{sk(B)}) \rangle. \end{aligned}$$

Every protocol execution comprises two distinguished threads: the adversary thread, if present, identified by thread identifier $tid_{\mathcal{A}}$, and the test thread, identified by thread identifier $Test$. The test thread is used while verifying security properties. In the computational setting, this is the thread where the adversary performs the so-called *test query*. In the same spirit, [7] calls the thread under consideration the *test thread*. For the test

thread, the substitution of role names by agent names, and all free variables by terms, is given by σ_{Test} and the role is given by R_{Test}. For example, if the test thread is performed by Alice in the role of the initiator, trying to talk to Bob, we have that $R_{Test} = I$ and $\sigma_{Test} = [Alice, Bob/I, R]$.

3.1.2 Execution Model

The framework uses a trace-based approach. The set $Trace$ is defined as $(TID \times Event^*)$, representing possible execution histories. The system-state is a tuple $(tr, IK, th, \sigma_{Test}) \in Trace \times \mathcal{P}(Term) \times (TID \nrightarrow Event^*) \times \mathcal{S}ub$, whose components are a trace tr, the adversary's knowledge IK, a partial function th mapping the thread identifiers of iniitiated threads to sequences of events (that have yet to be executed), and the role to agent and variabele assignments of the test thread. For a protocol P, the set of initial system states $IS(P)$ is defined as

$$IS(P) = \bigcup_{\sigma_{Test} \in \mathcal{T}est\mathcal{S}ub_P} \{(\langle\rangle, Agent \cup \{pk(a) \mid a \in Agent\}, \emptyset, \sigma_{Test})\}$$

where $\mathcal{T}est\mathcal{S}ub_P$ is the set of ground substitutions σ_{Test} such that

$$dom(\sigma_{Test} = dom(P) \cup \{v \sharp Test \mid v \in Var\} \wedge \forall r \in dom(P).\sigma_{Test}(r) \in Agent.$$

In contrast to Dolev-Yao models, the initial adversary knowledge does not contain any long-term secrets. The adversary may learn these from long-term key reveal events (adversary capabilities are introduced in Section 3.2). The semantics of protocol P is defined by a transition system that combines the execution-model rules from Fig. 3.1 with a set of adversary rules from Fig. 3.2. We first introduce the execution-model rules.

The create rule starts a new instance of a protocol role R. A fresh thread identifier tid is assigned to the thread, thereby distinguishing it from existing threads, the adversary thread, and the test thread. The rule takes the protocol P as a parameter. The role names of P, which can occur in events associated with the role, are repalced by agent names by the substitution σ. Similarly, the createTest rule starts the test thread. However, instead of choosing an arbitrary role, it takes an additional parameter R_{Test}, which represents the test role. Additionally, instead of choosing an arbitrary substitution, the test substitution σ_{Test} is used.

$$\frac{R \in dom(P) \quad dom(\sigma) = Role \quad ran(\sigma) \subseteq Agent \quad tid \notin (dom(th) \cup \{tid_{\mathcal{A}}, Test\})}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{create}(R, \sigma(R)))\rangle, IK, th[thread(P(R), tid, \sigma) \leftarrowtail tid], \sigma_{Test})}[\mathsf{create}]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \qquad Test \notin dom(th)}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(Test, \mathsf{create}(R_{Test}, a))\rangle, IK, th[thread(P(R_{Test}), Test, \sigma_{Test}) \leftarrowtail Test], \sigma_{Test})}[\mathsf{createTest}]$$

$$\frac{th(tid) = \langle \mathsf{send}(m)\rangle^\frown l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{send}(m))\rangle, IK \cup \{m\}, th[l \leftarrowtail tid], \sigma_{Test})}[\mathsf{send}]$$

$$\frac{th(tid) = \langle \mathsf{recv}(pt)\rangle^\frown l \qquad IK \vdash \sigma(pt) \qquad dom(\sigma) = FV(pt)}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{recv}(\sigma(pt)))\rangle, IK, th[\sigma(l) \leftarrowtail tid], \sigma_{Test})}[\mathsf{recv}]$$

$$\frac{th(tid) = \langle \mathsf{generate}(M)\rangle^\frown l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{generate}(M))\rangle, IK, th[l \leftarrowtail tid], \sigma_{Test})}[\mathsf{generate}]$$

$$\frac{th(tid) = \langle \mathsf{state}(M)\rangle^\frown l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{state}(M))\rangle, IK, th[l \leftarrowtail tid], \sigma_{Test})}[\mathsf{state}]$$

$$\frac{th(tid) = \langle \mathsf{sessionkeys}(M)\rangle^\frown l}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid, \mathsf{sessionkeys}(M))\rangle, IK, th[l \leftarrowtail tid], \sigma_{Test})}[\mathsf{sessionkeys}]$$

Figure 3.1: Execution-model rules

The send rule sends a message m to the network. In contrast, the recv rule accepts messages from the network that match a given pattern pt, where pt is a term and may contain free variables. The resulting substitution σ is applied to the remainning protocol steps l.

The last three rules support the framework's adversary-compromise model, which is explained shortly. The generate rule marks the fresh terms that have been generated, the state rule marks the current local state, and the sessionkeys rule marks a set of terms as session keys.

During protocol execution, the test thread may intentionally share some of its short-term secrets with other threads, for example a session key. Hence some adversary rules require distinguishing between the intended *partner threads* and other threads. The framework supports the notion of partnering based on matching histories for protocols with two roles. Before we formally introduce partnering we need to introduce two auxillary operators, $\downarrow$ and $\lfloor$. For traces, $\downarrow$ projects traces on events belonging to a particular thread identifier. For all $tid, tid' \in TID$, and $tr \in Trace$, $\langle\rangle \downarrow tid = \langle\rangle$ and

$$\big(\langle(tid', e)\rangle^\frown tr\big) \downarrow tid = \begin{cases} \langle e\rangle^\frown(tr \downarrow tid) & \text{if } tid = tid', \text{ and} \\ tr \downarrow tid & \text{otherwise.} \end{cases}$$

Similarly, for event sequences, $\lfloor$ selects the contents of events of a particular type. For all $\mathsf{evtype} \in \{\mathsf{create}, \mathsf{send}, \mathsf{recv}, \mathsf{generate}, \mathsf{state}, \mathsf{sessionkeys}\}$, $\langle\rangle\lfloor\mathsf{evtype} = \langle\rangle$ and

$$(\langle e\rangle\hat{}\, l)\lfloor\mathsf{evtype} = \begin{cases} \langle m\rangle\hat{}\,(l\lfloor\mathsf{evtype}) & \text{if} e = \mathsf{evtype}(e), \text{ and} \\ l\lfloor\mathsf{evtype} & \text{otherwise.} \end{cases}$$

Definition 4 (Matching histories)**.** *For sequences of events l and l', we define* $MH(l, l') \stackrel{def}{=} (l\lfloor\mathsf{recv} = l'\lfloor\mathsf{send}) \wedge (l\lfloor\mathsf{send} = l'\lfloor\mathsf{recv})$

The partnering definition of [7] is parameterized over the protocol P and the thest role R_{Test}.

Definition 5 (Partnering for two-role protocols)**.** *Let R be the non-test role,i. e., $R \in dom(P)$ and $R \neq R_{Test}$. For tr a trace,*

$$Partner(tr, \sigma_{Test}) = \{tid \mid tid \neq Test \wedge (\exists a.\, \mathsf{create}(R, a) \in tr\downarrow tid) \\ \wedge\ \exists\, l. MH(\sigma_{Test}(P(R_{Test})), (tr\downarrow tid)\hat{}\, l)\}$$

A thread *tid* is a partner iff *tid* is not $Test$, *tid* performs the role different from the test role, and *tid*'s history matches the test thread's (for $l = \langle\rangle$) or the thread may be completed to a matching one (for $l \neq \langle\rangle$).

3.2 Adversaries

We explain the capabilities of our adversary in Fig. 3.2. We call each of these capabilities an adversary-compromise rule. From a theoretical perspective, these rules factor in the various security definitions from the cryptographic protocol literature along three dimensions of adversarial compromise: *which* kind of data is compromised, *whose* data it is, and *when* the compromise occurs. From a practical point of view, the rules describe the not-so-perfect, but real world of security systems where long-term private keys are lost, session keys are susceptible to cryptanalysis, random number generators suffer from flaws, and the soft- and hardware implementing security services contain bugs such as overflowing buffers. For short, the rules are used to model the behavior of such systems in times of failures.

Long-term key compromise Let the long-term keys of an agent a be defined as

$$LongTermKeys(a) = \{sk(a)\} \cup \bigcup_{b \in Agent} \{k(a,b), k(b,a)\}.$$

The first four rules model the compromise of agent's long-term keys, represented by the trace event $\mathsf{LongtermKeyReveal}(a)$. This event models the adversary learning the long-term keys of agent a, expressed as $IK \cup LongTermKeys(a)$, and in traditional Dolev-Yao models implicitly occurs for dishonest agents before the honest agents start their threads.

The $\mathsf{LongtermKeyReveal}_{\mathsf{others}}$ rule models standard Dolev-Yao behavior: the adversary is allowed to corrupt the long-term keys of any number of agents as long as they are not intended partners of the test thread. Hence, if the test thread is performed by Alice, communicating with Bob, the adversary can learn, e.g., Dave's long-term key. This rule dates back to Lowe's famous man-in-the-middle attack on the Needham-Schroeder protocol.

The $\mathsf{LongtermKeyReveal}_{\mathsf{actor}}$ rule allows the adversary to learn the long-term key of the agent executing the test thread ($actor$). This rule allows the adversary to perform so-called Key Compromise Impersonation attacks [23]. We briefly discuss KCI in Appendix A.1.

The $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$ and $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$ rules have restrictions on when the compromise may occur. In particular, they allow the compromise of long-term keys only after the test thread has finished. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$, it is said to satisfy Perfect Forward Secrecy A.1. Both rules are used to model the behavior of the protocol in case of the corruption of at least one party's long-term key (break-in, loss, etc.).

The $\mathsf{LongtermKeyReveal}_{\mathsf{aftercorrect}}$ rule additionally requires that a finished partner thread must exist for the test thread. If a protocol satisfies secrecy properties with respect to an adversary that can use $\mathsf{LongtermKeyReveal}_{\mathsf{after}}$, it is said to satisfy weak Perfect Forward Secrecy, a notion that stems from [30] and excludes the adversary from inserting fake messages during protocol execution and learning the key of the involved agents later.

Short-term key compromise The remaining rules model the compromise of short-term data, i.e., data local to a specific thread. Whereas long-term key compromise is assumed to reveal *all* long-term keys of an agent, here it is differentiated between the different kinds of local data: *randomness*, *session keys*, and *other local data* such as the results of intermediate computations. The trace events $\mathsf{SessionKeyReveal}(tid)$ and $\mathsf{SessionstateReveal}(tid)$ indicate

$$\frac{a \in Agent \qquad a \notin \{\sigma_{Test}(R) \mid R \in dom(P)\}}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR}_{\mathsf{others}}]$$

$$\frac{a = \sigma_{Test}(R_{Test}) \qquad a \notin \{\sigma_{Test}(R) \mid R \in dom(P) \setminus \{R_{Test}\}\}}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR}_{\mathsf{actor}}]$$

$$\frac{a \in Agent \qquad th(Test) = \langle\rangle}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR}_{\mathsf{after}}]$$

$$\frac{a \in Agent \qquad th(Test) = \langle\rangle \qquad tid \in Partner(tr, \sigma_{Test}) \qquad th(tid) = \langle\rangle}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{LongtermKeyReveal}(a))\rangle, IK \cup LongTermKeys(a), th, \sigma_{Test})}[\mathsf{LKR}_{\mathsf{aftercorrect}}]$$

$$\frac{tid \neq Test \qquad tid \notin Partner(tr, \sigma_{Test})}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{SessionKeyReveal}(tid))\rangle, IK \cup union((tr \downarrow tid) \upharpoonright \mathsf{sessionkeys}), th, \sigma_{Test})}[\mathsf{SKR}]$$

$$\frac{tid \neq Test \qquad tid \notin Partner(tr, \sigma_{Test}) \qquad th(tid) \neq \langle\rangle}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{StateReveal}(tid))\rangle, IK \cup last((tr \downarrow tid) \upharpoonright \mathsf{state}), th, \sigma_{Test})}[\mathsf{SR}]$$

$$\frac{}{(tr, IK, th, \sigma_{Test}) \longrightarrow (tr^\frown\langle(tid_{\mathcal{A}}, \mathsf{RandomReveal}(tid))\rangle, IK \cup union((tr \downarrow tid) \upharpoonright \mathsf{generate}), th, \sigma_{Test})}[\mathsf{RNR}]$$

Figure 3.2: Adversary-compromise rules

that the adversary reveals the session key or, respectively, the local state of the thread *tid*. The RandomReveal(tid) trace event indicates that the adversary learns the random numbers generated in thread *tid*.

The rules Session Key Reveal and Sessionstate Reveal allow the adversary to corrupt the session keys and the local state of any thread that is not a partner of the test thread, respectively. These rules allow to formally model leakage of information on either specific session keys or on a thread's virtual memory that may result from events such as break-ins, cryptanalysis, careless disposal of keys, buffer overflow attacks, etc.. In contrast, the Random Reveal rule has no such premise. It allows the adversary to gain access to the random numbers generated by any thread, including the partner threads. This rule provides the means to account for flawed random number generators.

We call each subset of the set of adversary-compromise rules from Fig. 3.2 an *adversary-compromise model*. We introduce the adversary-compromise models from [7] in Chapter 6.

3.3 Security Properties

We now provide three security property definitions. We give symbolic definitions for session key secrecy, aliveness, and weak agreement which, when combined with different adversary-compromise models, allow us to express different security notions from the literature, e. g., Perfect Forward Secrecy, Key Compromise Impersonation, etc. The set $RS(P, Adv, R_{Test})$ denotes the set of reachable states as defined in [6, Definition 7].

3.3.1 Secrecy

Definition 6 (Session key secrecy [6])**.** *Let P be a protocol and Adv an adversary model. We say that P satisfies* session-key secrecy *with respect to Adv if and only if*

$$\forall R_{Test} \in dom(P). \forall (tr, IK, th, \sigma_{Test}) \in RS(P, Adv, R_{Test}).$$
$$th(Test) = \langle\rangle \Rightarrow \forall k \in union\,((tr \downarrow Test) \lfloor \mathsf{sessionkeys} \rfloor)\,.IK \not\vdash k.$$

A session key k is secret if and only if it is impossible for the adversary to infer k in any of the reachable system states.

3.3.2 Authentication

Later in this thesis, we will analyze IKE with respect to two authentication properties: aliveness and weak agreement. Both properties were introduced by Lowe in [33]. The weakest authentication property specified by Lowe is *aliveness*. A protocol is said to guarantee to an agent a in role A aliveness of another agent b if, whenever a completes a run of the protocol, apparently with b in role B, then b has previously been running the protocol. Note that this not necessarily means that previously b assumed role B. The following definition captures the formal definition of aliveness within the framework.

Definition 7 (Aliveness for two-party protocols [6])**.** *Let P be a protocol and Adv an adversary model. We say that P satisfies* aliveness *with respect to Adv, if and only if*

$$\forall R_{Test} \in dom(P). \forall (tr, IK, th, \sigma_{Test}) \in RS(P, Adv, R_{Test}).$$
$$th(Test) = \langle\rangle \Rightarrow \exists R_{Test}, R, tid.\ (Test, \mathsf{create}(R_{Test}, \sigma_{Test}(R_{Test}))) \in tr$$
$$\wedge\ R \neq R_{Test} \wedge (tid, \mathsf{create}(R, \sigma_{Test}(R))) \in tr.$$

Another authentication property is *weak agreement*. A protocol guarantees to an agent a in role A weak agreement with another agent b if, whenever agent a completes a run of the protocol, apparently with b in role B, then b has previously been running the protocol, *apparently with a*.

Definition 8 (Weak agreement for two party protocols)**.** *Let P be a protocol and Adv an adversary model. We say that P satisfies* weak agreement *with respect to Adv, if and only if*

$$\forall R_{Test} \in dom(P).\forall (tr, IK, th, \sigma_{Test}) \in RS(P, Adv, R_{Test}).$$
$$th(Test) = \langle\rangle \Rightarrow \exists R_{Test}, R, tid.\ (Test, \mathsf{create}(R_{Test}, \sigma_{Test}(R_{Test}))) \in tr$$
$$\wedge\ R \neq R_{Test} \wedge (tid, \mathsf{create}(R, \sigma_{Test}(R))) \in tr \wedge \sigma_{Test}(R_{Test}) = \sigma(R_{Test}).$$

Chapter 4

The Internet Key Exchange Protocol

The Internet Key Exchange (IKE) protocol is being developed by the IP Security (IPSec) Working Group of the Internet Engineering Task Force (IETF); its current version number is 2. IKE has evolved from a number of key exchange protocols, among them ISAKMP [34] and Oakley [38]. It unifies the packet formats defined in the ISAKMP standard with the key management aspects introduced in Oakley.

IKEv2 as well as its predecessor are structured into two consecutive phases. In the first phase, initiator and responder use pre-established security mechanisms such as pre-shared keys or public key infrastructures to negotiate security policy and exchange authenticated keying material. The security policy defines the cryptographic algorithms, hash functions, etc. to be used during the exchange. Together with the agreed upon keying material it represents a security association. In the second phase, initiator and responder use the established security associations to negotiate further SAs which are then handed over to IPSec to protect subsequent communication channels. We refer to Fig. 4.1 for a high level overview of IKE.

Notational preliminaries Let G be a multiplicative group of prime order q and let g be a generator of G. We write $\alpha = g^{x_i}$ and $\beta = g^{x_r}$ to denote the initiator's and responder's public Diffie-Hellman (DH) token, respectively. We write $Z = \alpha^{x_r} = \beta^{x_r}$ to denote the ephemeral DH secret shared between initiator and responder. Concatenation of two terms is denoted as $(\cdot)$.

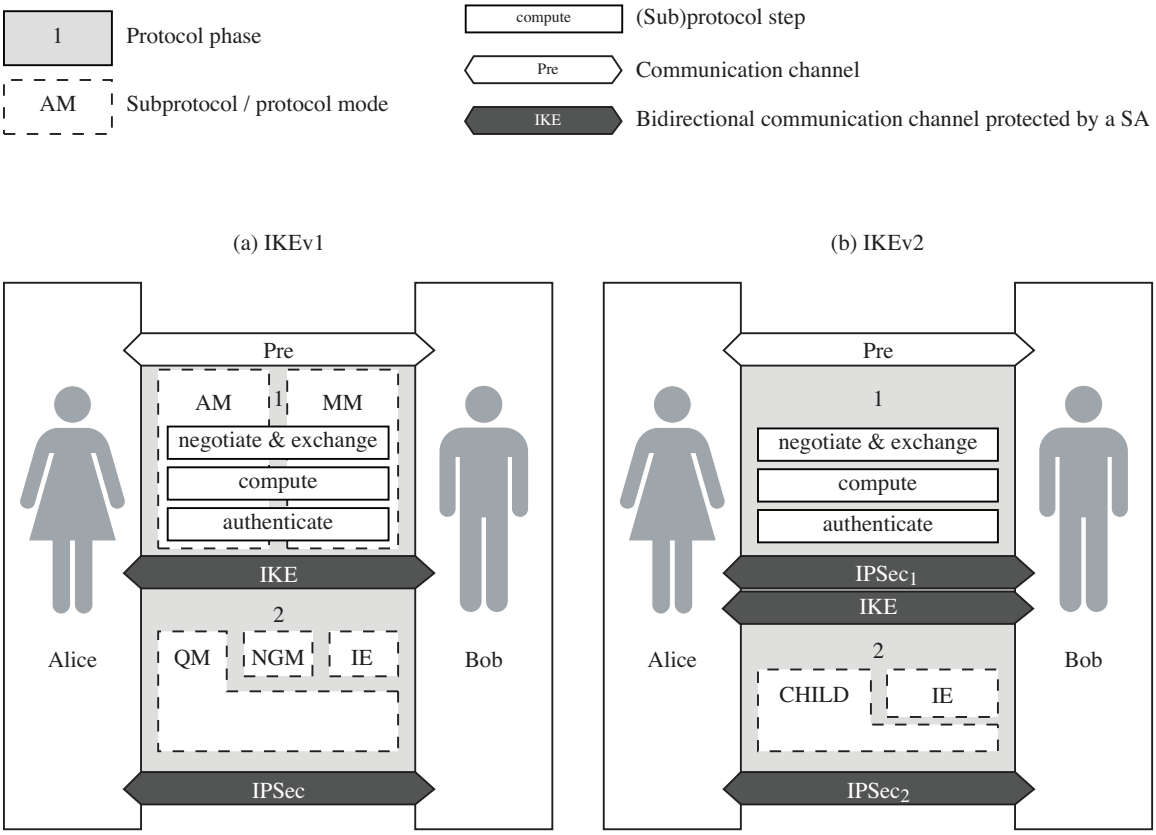

Figure 4.1: IKE overview: in phase 1, Alice and Bob use a communication channel, which is protected by long-term secrets, to negotiate security policy and exchange authenticated keying material, resulting in the first SA(s), the IKE SA (and, for IKEv2, the first IPSec SA). In phase 2, both parties use the IKE SA to protect negotiation and exchange of new cryptographic material (QM, child), resulting in an IPSec SA. Phase 2 can also be used to renegotiate IKE SA policy (NGM, child), or to exchange other information (IE).

4.1 IKE Version 1

4.1.1 Phase 1 Exchange

The establishment of an ISAKMP SA (IKE SA) must always be the first step of an IPSec transaction. The IKE SA may be considered a "control channel" upon which IKE is the control protocol. As such, it is not required to be present for the lifetime of the SAs that it helps to create. Several (child) IPSec SAs may be created from a single IKE SA; they remain active even if the parent has been terminated. Two hosts can maintain more than one IKE SA concurrently.

Phase 1 uses the well-known Diffie-Hellman exchange to establish initial shared keying material for the protection of the IKE SA. Both parties authenticate one another to anticipate Man-in-the-Middle attacks. To mitigate the risk of being the target of a Denial-of-Service (DoS) attack, both parties additionally employ Photuris style anti-clogging tokens [25], so-called cookies.

Phase 1 offers two modes: main mode (MM) and aggressive mode (AM). Each mode results in the establishment of an IKE SA but differs in the number of exchanged messages between initiator and responder. Main mode consists of six messages, three in each direction. It offers identity protection and considerable flexibility in terms of the parameters and configurations that can be negotiated. Aggressive mode consists of a three way handshake between initiator and responder. Its main advantage is speed; this comes at the cost of slightly less security – the exchange occurs without identity protection, except when public key encryption is used for authentication – and flexibility. Each mode supports 4 authentication schemes which influence the derivation of keying material as well as the structure of the exchange itself.

4.1.1.1 Negotiation

MM is an instantiation of the ISAKMP Identity Protect Exchange [34, section 4.5]. The first two messages negotiate policy; the next two exchange the Diffie-Hellman tokens and ancillary data (e. g. nonces) necessary for the exchange; and the last two messages authenticate the Diffie-Hellman exchange. The authentication method negotiated as part of the initial ISAKMP exchange influences the composition of the payloads but not their purpose.

Similarly, AM is an instantiation of the ISAKMP Aggressive Exchange [34, section 4.7]. The first two messages negotiate policy, exchange Diffie-Hellman public values and ancillary

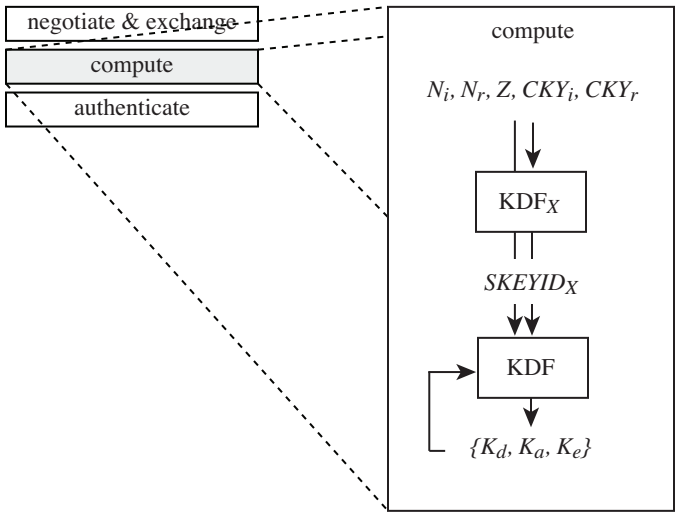

Figure 4.2: The computation of IKEv1 keying material is a multi-step process. First, the initial master secret is derived dependent on the authentication method which is being used. Second, various keys are computed for different purposes.

data necessary for the exchange, and identities. In addition, the second message authenticates the responder. The third message authenticates the initiator and provides a proof of participation in the exchange. The final message may be sent either with or without protection of the IKE SA.[1] This allows each party to postpone exponentiation, if desired, until this exchange is completely negotiated.

4.1.1.2 Key Derivation and Authentication

MM and AM both rely on the same multi-step process of key derivation. The first step for initiator and responder is to derive the ephemeral Diffie-Hellman secret Z. This initial secret is then used together with additional randomness to derive $SKEYID$, a secret value all further keying material is derived from. As key derivation function, denoted $\mathsf{prf}(\cdot)$, initiator and responder either use the negotiated pseudo-random function, or, if no pseudo-random function has been negotiated, the HMAC [8] version of the negotiated hash function.

[1] Our results indicate that encrypting the last message does not improve the protocol's security. See Chapter 6 for more information.

Definition 9 (Key seed)**.** *Let N_i, N_r, CKY_i, and CKY_r be random values. Then,*

$$SKEYID_{SIG} \stackrel{def}{=} \mathsf{prf}(N_i \cdot N_r, Z) \tag{4.1}$$

$$SKEYID_{PK} \stackrel{def}{=} \mathsf{prf}\left(\mathsf{prf}(N_i \cdot N_r), CKY_i \cdot CKY_r\right) \tag{4.2}$$

$$SKEYID_{PSK} \stackrel{def}{=} \mathsf{prf}\left(k(I, R), N_i \cdot N_r\right). \tag{4.3}$$

The computation of the key seed is parametrized over the authentication method being used. If the exchange is authenticated using digital signatures, then $SKEYID_{SIG}$ is used as key seed. Otherwise, either $SKEYID_{PK}$ or $SKEYID_{PSK}$ is used for public key encrypted or pre-shared key authenticated IKEv1, respectively. N_i and N_r refer to the nonces of initiator and responder, respectively, and the terms CKY_i and CKY_r denote their cookies. Cookies are random values that are used to uniquely identify the security association the messages belong to, and as an anti-clogging token. The final set of keying material is derived according to Definition 10.

Definition 10 (Keying material)**.** *Let CKY_i be the initiator's and CKY_r be the responder's cookie, let 0, 1, 2 be of type integer, and $SKEYID \in \{SKEYID_{SIG}, SKEYID_{PK}, SKEYID_{PSK}\}$.*

1. *The keying material for the* derivation[2] *of non-ISAKMP security associations is defined as*

$$K_d \stackrel{def}{=} \mathsf{prf}\left(SKEYID, Z \cdot CKY_i \cdot CKY_r \cdot 0\right). \tag{4.4}$$

2. *The keying material to protect the* integrity *of ISAKMP/IKE is defined as*

$$K_a \stackrel{def}{=} \mathsf{prf}\left(SKEYID, K_d \cdot Z \cdot CKY_i \cdot CKY_r \cdot 1\right). \tag{4.5}$$

3. *The keying material to protect the* confidentiality *of ISAKMP/IKE is defined as*

$$K_e \stackrel{def}{=} \mathsf{prf}\left(SKEYID, K_a \cdot Z \cdot CKY_i \cdot CKY_r \cdot 2\right). \tag{4.6}$$

To provide authentication for their messages, initiator and responder each compute an authentication payload, referred to as authenticator.

[2]Note that K_d is only used to derive more keying material if perfect forward secrecy is not required. Otherwise, quick mode must be used.

Definition 11 (Authenticators)**.** *Let $PROPOSALS$ denotes the list of SA proposals, and let I and R denote the identities of initiator and responder, respectively.*

1. *The initiator's authenticator is defined as*

$$HASH_i \stackrel{def}{=} \mathsf{prf}\,(K_a, \alpha \cdot \beta \cdot CKY_i \cdot CKY_r \cdot PROPOSALS \cdot I). \tag{4.7}$$

2. *The responder's authenticator is defined as*

$$HASH_r \stackrel{def}{=} \mathsf{prf}\,(K_a, \beta \cdot \alpha \cdot CKY_r \cdot CKY_i \cdot PROPOSALS \cdot R). \tag{4.8}$$

4.1.1.3 Main Mode

As mentioned above, IKE supports four types of authentication: digital signatures, two forms of public key encryption, and pre-shared keys. We illustrate them below.

Digital signatures Digital signatures, such as those provided by DSA and RSA, can also be used for phase 1 authentication. Both of these methods use public/private key pairs. Initiator and responder both sign their hash values using their long-term private keys. Thus,

$$SIG_i \leftarrow \{\!|\, HASH_i \,|\!\}^a_{sk(I)} \tag{4.9}$$

$$SIG_r \leftarrow \{\!|\, HASH_r \,|\!\}^a_{sk(R)}\,. \tag{4.10}$$

The recipient of a signature will use the signer's public key to decrypt and verify the signature. It is not required that the entity's public keys are known by their respective peers in advance; initiator and responder transmit their identities, along with an optional certificate, in the last two messages. The exchange is shown in Fig. 4.2(a).

Public key encryption Using encryption for authentication may seem non-standard. However, public key encryption can also be used to authenticate the participants in a key exchange. In addition to the cookies and DH tokens in the third message, the initiator places its identity and nonce, each encrypted with the responder's public key. This requires the initiator to already know the responder's public key, since the responder has not transmitted its identity yet. This implies that the responder's identity cannot be hidden from an active adversary. Similarly, the responder replies with its identity and nonce in message

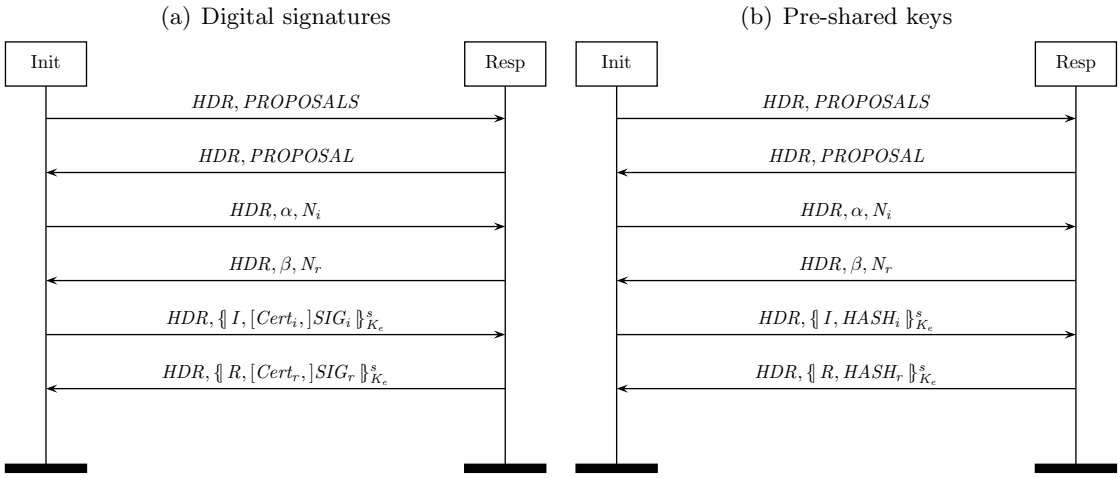

Figure 4.3: IKEv1 MM authenticated with digital signatures and pre-shared keys

4, each encrypted with the initiator's public key. Since the identity of the initiator can be determined by decrypting the identity field of the previous message, the responder does not need to know the initiator's public key ahead of time. Because the intended recipient is the only one who can decrypt the third message, the initiator knows that the responder is valid if it receives the proper value of $HASH_r$. Likewise, the converse case applies for authentication of the initiator. The main disadvantage of this type of exchange is that it requires a total of four public key operations on each side (two encryptions and two decryptions), each of which requiring expensive processing. A revised version reduces the amount of public key operations by introducing a temporary, symmetric shared key k_x. Both variants of this exchange are presented in Figure 4.4.

Pre-shared keys The use of pre-shared keys allows for the simplest form of authentication. In order for authentication to occur, initiator and responder must have had previously agreed upon a key, typically using some offline technique, e. g., the company's network administrator distributing passwords to employees. When attempting a key negotiation, the employee is challenged for the password.

A serious drawback of using pre-shared keys is that since $k(I, R)$ is used to derive the encryption key K_e, the recipient of an authentication payload will not be able to view the payload *without* knowing which pre-shared key to use (see Figure 4.2(b)). The only information that the recipient can use to look up the proper key is the source IP address of the packet. This not only puts severe limitations on the practical use of pre-shared keys

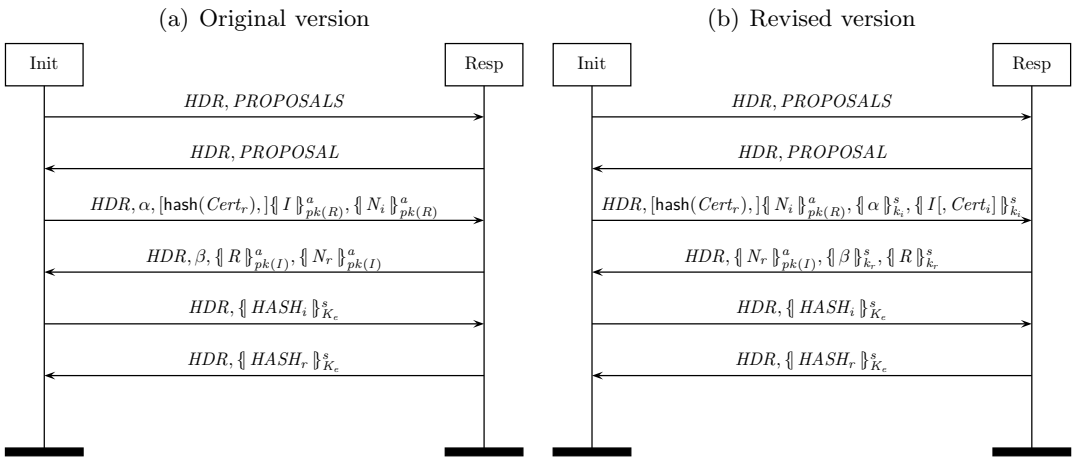

Figure 4.4: IKEv1 MM authenticated with public key encryption: the initiator of a public key authenticated exchange sends the hash of the certificate used for the encryption along with the initial message.

in environments where no static IP addresses are used (e. g., with DHCP) but also violates the identity concealment property (please consult [39] for more information on identity concealment problems).

4.1.1.4 Aggressive Mode

For each authentication scheme there exists an AM exchange that performs key negotiation and authentication in a three-way handshake (at the cost of slightly less security). Except for authentication with public key encryption, AM cannot hide the participant's identities. Figure 4.5 shows the AM exchanges for all authentication schemes.

4.1.2 Phase 2 Exchanges

IKE phase 2 can be used to either negotiate a non-ISAKMP SAs (quick mode), to negotiate new security parameters (new group mode), or to send an unacknowledged notification message (the latter is not discussed here since it is not directly related to key exchange). Phase 2 negotiations can only occur if there already exists a successfully established ISAKMP SA (phase 1) as all phase 2 packets will be protected by that SA.

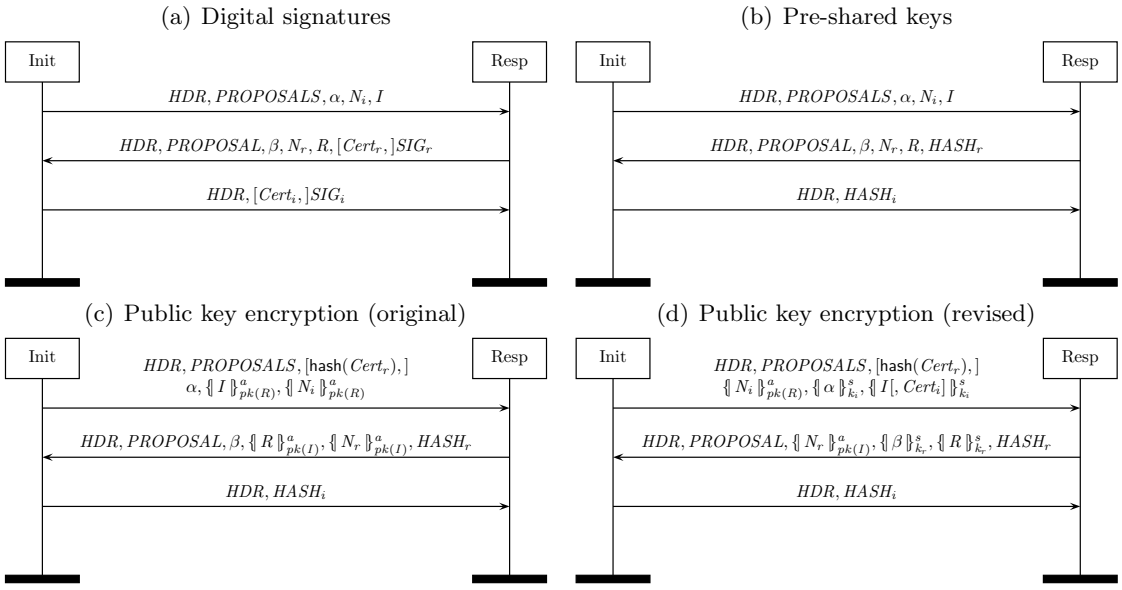

Figure 4.5: IKEv1 AM exchanges: the initiator of a public key authenticated exchange sends the hash of the certificate used for the encryption along with the initial message.

4.1.2.1 Quick Mode

The goal of IKEv1's quick mode (QM) is to generate a security association with a purpose outside ISAKMP/IKE, e. g. a standard IPSec SA. If Perfect Forward Secrecy is required, fresh DH tokens can be exchanged. Multiple QM negotiations may take place simultaneously between two participants. Since all of these negotiations use the same pair of cookies as in phase 1, each negotiation must be assigned a unique identifier so that it can be distinguished from its siblings. This is accomplished through the use of a message identifier, M_{ID}, which is part of the generic ISAKMP header included in all IKE messages.

Figure 4.6 depicts the message exchange for QM. The first message contains the initiator's cookies, its IPSec SA offer, and its nonce. If Perfect Forward Secrecy is required, the message will also contain a DH token, optional group information, and the identities of initiator and responder. The initiator authenticates the message with the hash H_1 which is computed as

$$H_1 \leftarrow \begin{cases} \mathsf{prf}\left(K_a, M_{ID} \cdot PROPOSALS \cdot N_i' \cdot \alpha' \cdot I \cdot R\right), & \text{if PFS} \\ \mathsf{prf}\left(K_a, M_{ID} \cdot PROPOSALS \cdot N_i'\right), & \text{otherwise.} \end{cases} \tag{4.11}$$

The second message contains the responder's cookies, its IPSec SA selection, and its nonce.

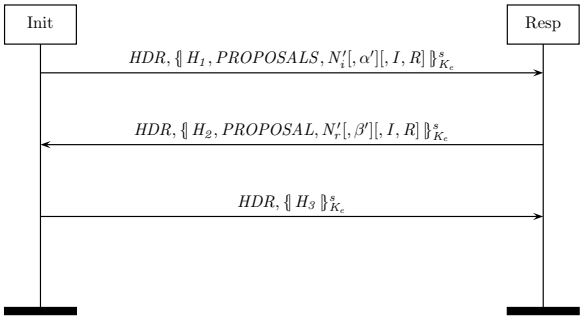

Figure 4.6: IKEv1 QM exchange

If PFS is required, the message will also contain a DH token and the identities of initiator and responder. The responder authenticates the message with the hash H_2 computed as

$$H_2 \leftarrow \begin{cases} \mathsf{prf}\left(K_a, M_{ID} \cdot PROPOSAL \cdot N_i' \cdot N_r' \cdot \beta' \cdot I \cdot R\right), & \text{if PFS} \\ \mathsf{prf}\left(K_a, M_{ID} \cdot PROPOSALS \cdot N_i' \cdot N_r'\right), & \text{otherwise.} \end{cases} \tag{4.12}$$

Message 3 contains the hash H_3 with which the initiator authenticates the whole transaction

$$H_3 \leftarrow \mathsf{prf}\left(K_a, 0 \cdot M_{ID} \cdot N_i' \cdot N_r'\right). \tag{4.13}$$

Keying material to protect the newly established IPSec SA is derived according to Definition 12.

Definition 12. *Let N_i' and N_r' be random values. Let P be a protocol identifier and let SPI denote a security parameter index. The keying material for IPSec security association is derived from K_d as*

$$KEYMAT \stackrel{def}{=} \begin{cases} \mathsf{prf}\left(K_d, Z' \cdot P \cdot SPI \cdot N_i' \cdot N_r'\right) & \textit{if PFS} \\ \mathsf{prf}\left(K_d, P \cdot SPI \cdot N_i' \cdot N_r'\right) & \textit{otherwise} \end{cases} \tag{4.14}$$

where $Z' = g^{x_i' x_r'}$ is the newly derived Diffie-Hellman ephemeral secret.

4.1.2.2 New Group Mode

This mode is used to change the cryptographic group which future negotiations will base their DH computations on. It does not establish new SAs. Further, new group mode (NGM) does not replace or preempt QM.

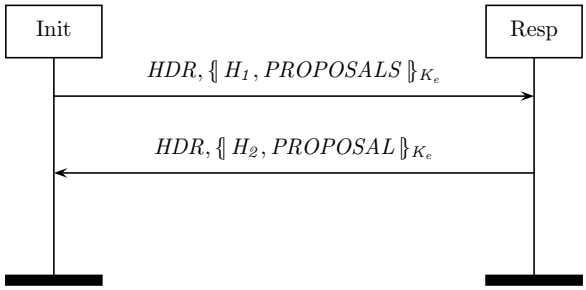

Figure 4.7: IKEv1 NGM exchange

Figure 4.7 depicts the NGM exchange. In the first message, the initiator proposes a new group with an SA payload. H_1 authenticates the message. The second message contains the responder's response to the new group proposal. H_2 authenticates the message. The hashes are defined as

$$H_1 \leftarrow \mathsf{prf}\,(K_a, M_{ID} \cdot PROPOSALS) \tag{4.15}$$

$$H_2 \leftarrow \mathsf{prf}\,(K_a, M_{ID} \cdot PROPOSAL). \tag{4.16}$$

4.2 IKE Version 2

Complexity has been a recurring theme in most of the analyses of IKE [4, 20, 36, 39]. Indeed, the initial version of IKE contains no less than eight key establishment protocols which can be divided into two sets of four, according to whether MM or AM is used (cf. Section 4.1). The newest incarnation of IKE's specification addresses these complexity issues by simplifying the structure of the exchange itself as well as by reducing the degree of flexibility. Some of the major similarities and differences between IKEv2 and its predecessor include the following.

- IKEv2 replaces IKEv1's eight different phase 1 exchanges by a single four-message exchange which is suitable for the majority of IKE negotiations.
- IKEv2 primarily supports two authentication schemes (digital signatures and pre-shared secrets), although a wide range of legacy authentication methods can be integrated by using the Extensible Authentication Protocol (EAP, Section 4.2.1.3).
- IKEv2 is more efficient in setting up the first non-IKE SA as this SA can be established

as part of the initial exchange after four messages, while IKEv1 requires at least six messages (using three messages in AM followed by three more in QM).

- IKEv1 QM is roughly equivalent to IKEv2's `CREATE_CHILD_SA` exchange, though IKEv2 uses one message less than IKEv1.

A more detailed list of differences between IKEv1 and IKEv2 can be found in [26, Appendix A].

4.2.1 Phase 1 Exchange

IKEv2's initial exchange is derived from what was previously known as the ISAKMP Identity Protect Exchange. It combines the first four messages of the ISAKMP exchange into a single two-message exchange called `IKE_SA_INIT`. The output of the `IKE_SA_INIT` exchange is a (yet unauthenticated) IKE SA representing the shared state for the efficient establishment of subsequent SAs (child SAs). The next two messages are grouped into `IKE_AUTH` and authenticate the `IKE_SA_INIT` exchange. Additionally, `IKE_AUTH` establishes a second SA, the first child SA. The output of IKEv2's phase 1 is therefore a set of two bidirectional SAs; an IKE SA containing the shared secret keys and a set of cryptographic algorithms to be used in phase 2 to negotiate supplementary child SAs, and the first child security association.

While Diffie-Hellman key negotiation remains as a cornerstone in version 2, IKEv2 replaces the 8 individual "protocols", arising from the combination of two modes with four different authentication methods, with a single four-message exchange that covers most scenarios. Exotic situations, e. g., a peer finding itself under a flooding attack, are covered by the possibility of having extra round trips. Public key encryption as a form of authentication is no longer supported by IKEv2's standard four-message exchange.

4.2.1.1 Negotiation

During `IKE_SA_INIT`, initiator and responder exchange their Diffie-Hellman tokens α and β and a pair of nonces N_i and N_r. Additionally, the initiator proposes a set of cryptographic algorithms (e. g. encryption, authentication, and integrity protection), to be used during the third and the fourth message exchange, in $PROPOSALS$. The responder expresses his corresponding choice in $PROPOSAL$. The message headers contain the Security Parameter Indices (SPI), version numbers and flags of various sorts. The optional CR payload

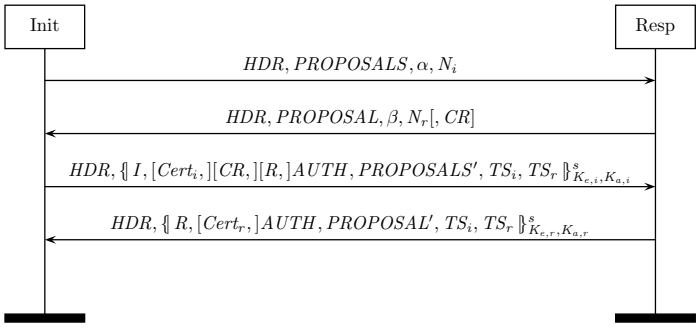

Figure 4.8: IKEv2 phase 1 exchange: $\{\!| m |\!\}_{K_{e,x},K_{a,x}}$ denotes integrity protection of message m using key $K_{a,x}$ *and* encryption using key $K_{e,x}$.

denotes the responder's certificate request; a certificate request provides a means to request certificates of preferred certification authorities. Note that unlike in IKEv1, the initiator directly starts with exchanging the DH token. This has two security implications. First, the initiator must guess the Diffie-Hellman group expected by the responder and second, the responder must somehow protect itself if it is under a DoS attack. To prevent this kind of attack, the default exchange can be prefixed by a cookie exchange, resulting in an extra round trip. Figure 4.8 describes the initial exchange of IKEv2 (without the prefixed cookie exchange).

Each message in `IKE_AUTH` is integrity protected. This is equivalent to computing the HMAC of the identity as described in SIGMA protocol [29]. Thus, the protocol exchange, as presented, is secure against key-misbinding attacks [19]. To protect the protocol against Man-in-the-Middle attacks inherent to the standard Diffie-Hellman protocol, the initiator performs the authentication operation on the first message of `IKE_SA_INIT` and sends the result in the *AUTH* payload. Similarly, the responder authenticates the second message of the `IKE_SA_INIT` exchange. By this, both Diffie-Hellman tokens are authenticated as described by Diffie et al. [19]. Moreover, an active adversary is no longer able to fool the peers into agreeing on weaker encryption/authentication algorithms as has been described for IKEv1 by both Ferguson and Schneier [20], and Zhou [41]. The exact method used to compute the *AUTH* payload is based on the authentication algorithm to be used for the IKE SA and will be discussed in Section 4.2.1.3. Note that by separating the *AUTH* payload from the default key exchange algorithm, it is possible for initiator and responder to authenticate its peer using entirely different authentication schemes.

Unlike IKEv1, IKEv2 establishes the first non-IKE SA during phase 1. Therefore,

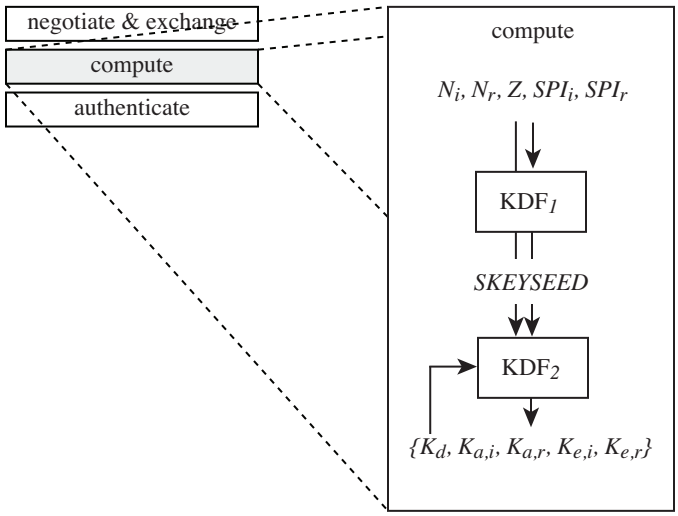

Figure 4.9: The computation of IKEv2 keying material is a multi-step process. First, the initial master secret is derived as a function of the nonces and the ephemeral Diffie-Hellman secret. Second, a multitude of keys are computed for different purposes.

the `IKE_AUTH` payloads contain the security association algorithms to be used for IPSec processing ($PROPOSALS', PROPOSAL'$). In addition, the message also contains traffic selectors (TS_i, TS_r) which allow for fine-grained IPSec policy setting based on IP address and port.

4.2.1.2 Key Derivation

After completing the `IKE_SA_INIT` exchange, both initiator and responder will have enough information to derive the ephemeral Diffie-Hellman key Z. This key, together with the previously negotiated pseudo-random function $\mathsf{prf}(\cdot)$, is then used for the construction of keying material for all of the cryptographic algorithms used in both the IKE SA and the child SAs.

First, both parties compute the value $SKEYSEED$[3] as a function of the nonces and DH tokens exchanged in `IKE_SA_INIT`. $SKEYSEED$ is used to derive seven other secrets: K_d used for deriving new keys for the child SAs established through this IKE SA; $K_{a,i}$ and $K_{a,r}$ used as a key to the integrity protection algorithm for authenticating subsequent exchanges; $K_{e,i}$ and $K_{e,r}$ used for encrypting (and of course decrypting) all subsequent exchanges; and $K_{p,i}$ and $K_{p,r}$, which are used for the generation of $AUTH$ payloads.

[3]Note that IKEv2, unlike IKEv1, no longer derives different key seeds per authentication scheme.

Definition 13 (Key seed)**.** *Let N_i, N_r be random values. The secret value $SKEYSEED$ is defined as*

$$SKEYSEED \stackrel{def}{=} \mathsf{prf}\,(N_i \cdot N_r, Z) \tag{4.17}$$

Definition 14 (Keying material)**.** *Let N_i, N_r be random values. Let SPI_i denote the initiator's and SPI_r be the responder's security parameter index, and let $0, 1, \ldots, 7$ be of type integer.*

1. *The keying material for the* derivation *of non-ISAKMP security associations is defined as*

$$K_d \stackrel{def}{=} \mathsf{prf}\,(SKEYSEED, N_i \cdot N_r \cdot SPI_i \cdot SPI_r \cdot 1). \tag{4.18}$$

2. *The keying material to protect the* integrity *of ISAKMP/IKE is given as*

$$K_{a,i} \stackrel{def}{=} \mathsf{prf}\,(SKEYSEED, K_d \cdot N_i \cdot N_r \cdot SPI_i \cdot SPI_r \cdot 2) \tag{4.19}$$

$$K_{a,r} \stackrel{def}{=} \mathsf{prf}\,(SKEYSEED, K_{a,i} \cdot N_i \cdot N_r \cdot SPI_i \cdot SPI_r \cdot 3). \tag{4.20}$$

3. *The keying material to protect the* confidentiality *of ISAKMP/IKE is defined as*

$$K_{e,i} \stackrel{def}{=} \mathsf{prf}\,(SKEYSEED, K_{a,r} \cdot N_i \cdot N_r \cdot SPI_i \cdot SPI_r \cdot 4) \tag{4.21}$$

$$K_{e,r} \stackrel{def}{=} \mathsf{prf}\,(SKEYSEED, K_{e,i} \cdot N_i \cdot N_r \cdot SPI_i \cdot SPI_r \cdot 5). \tag{4.22}$$

4. *Additional keys for authentication purposes are given as*

$$K_{p,i} \stackrel{def}{=} \mathsf{prf}\,(SKEYSEED, K_{e,r} \cdot N_i \cdot N_r \cdot SPI_i \cdot SPI_r \cdot 6) \tag{4.23}$$

$$K_{p,r} \stackrel{def}{=} \mathsf{prf}\,(SKEYSEED, K_{p,i} \cdot N_i \cdot N_r \cdot SPI_i \cdot SPI_r \cdot 7) \tag{4.24}$$

The two directions of traffic flow use different keys. The keys used to protect messages from the original initiator are $K_{a,i}$ and $K_{e,i}$. The keys used to protect messages in the other direction are $K_{a,r}$ and $K_{e,r}$.

4.2.1.3 Authentication

The initial message exchange presented in Section 4.2.1.1 supports two authentication methods; digital signatures and message authentication codes (MAC). Additionally, to cover

legacy authentication schemes, [26] describes the integration of EAP [3] into IKEv2. We describe the three mechanisms below.

Note that there is no requirement that initiator and responder authenticate themselves with the same cryptographic algorithms. The choice of cryptographic algorithms depends on the type of key each party has. In particular, the initiator may be using a shared key while the responder may have a public signature key and certificate.

Digital signatures With digital signatures, initiator and responder both compute the signature of a block of data that includes messages 1 and 2, respectively. For the responder, the data to be signed contains SPI_r (from the header of the second message), the payload of the second message, the initiator's nonce, and the value $\mathsf{prf}\,(K_{p,r}, R)$. Similarly, the initiator signs the first message, starting with SPI_i. Appended to this are the responder's nonce and the value $\mathsf{prf}\,(K_{p,i}, I)$.

Definition 15. *Let N_i, N_r, SPI_i and SPI_r be random values, $PROPOSALS$ be the list of security association proposals and $PROPOSAL$ be the corresponding choice thereof; and let I and R denote the identities of initiator and responder.*

$$AUTH_i \stackrel{def}{=} \{\!|\, SPI_i, 0, PROPOSALS, \alpha, N_i, N_r, \mathsf{prf}\,(K_{p,i}, I) \,|\!\}_{sk(I)} \tag{4.25}$$

$$AUTH_r \stackrel{def}{=} \{\!|\, SPI_i, SPI_r, PROPOSAL, \beta, N_r, N_i, \mathsf{prf}\,(K_{p,r}, R) \,|\!\}_{sk(R)} \tag{4.26}$$

The recipient of a signature will use the signer's public key to decrypt and verify the signature. If the public key is unknown, it can be requested through a certificate request (CR) in the second or the third message, depending on the recipients role in the protocol.

Pre-shared secrets If initiator and responder already posses a shared secret, they can use a MAC, using the shared secret as the key, to authenticate one another. To authenticate itself to the responder, the initiator computes the MAC over the first message including SPI_i from the header and the value $\mathsf{prf}\,(K_{p,i}, I)$. Similarly, the responder computes the MAC over SPI_r, the payload of the second message and the value $\mathsf{prf}\,(K_{p,r}, R)$ to authenticate itself to the initiator.

Definition 16. *Let N_i, N_r, SPI_i and SPI_r be random values, $PROPOSALS$ be the list of security association proposals and $PROPOSAL$ be the corresponding choice thereof; and*

let I and R denote the identities of initiator and responder.

$$AUTH_i \stackrel{def}{=} \mathsf{prf}(k(I,R), SPI_i, 0, PROPOSALS, \alpha, N_i, N_r, \mathsf{prf}\,(K_{p,i}, I)) \quad (4.27)$$

$$AUTH_r \stackrel{def}{=} \mathsf{prf}(k(R,I), SPI_i, SPI_r, PROPOSAL, \beta, N_r, N_i, \mathsf{prf}\,(K_{p,r}, R)) \quad (4.28)$$

Typically, although not required by the specification, the same key is used in both directions.

Extensible Authentication Protocol (EAP) In addition to authentication using public key signatures and shared secrets, IKEv2 supports "Legacy Authentication" through EAP. Typically, EAP methods are asymmetric (designed for a user authenticating to a server), and they may not be mutual. These protocols are typically used to authenticate the initiator to the responder and therefore require public key signature based authentication of the responder to the initiator.

An initiator indicates a desire to use EAP by leaving out the $AUTH$ payload from message 3. By including an ID_i payload but not an $AUTH$ payload, the initiator has declared an identity but has not yet proven it. If the responder is willing to use an extensible authentication method, it will place an EAP payload in message 4 and defer sending $PROPOSAL'$, TS_i, and TS_r until initiator authentication is complete. A minimal extensible authentication exchange is described in Figure 4.10.

4.2.2 Phase 2 Exchanges

Similarly to its predecessor, IKEv2 allows for the efficient creation of child SAs and the rekeying of existing SAs. Furthermore, initiator and responder may convey control messages to each other regarding errors or notifications of certain events through informational exchanges that are also protected by the IKE SA. Note that informational exchanges are not discussed here since they are not directly related to key exchange.

The negotiation of new SAs based on an existing IKE SA is referred to as `CREATE_CHILD_SA`, which is comparable to IKEv1 QM. It consists of a single request/response pair and may be initiated by either end of the IKE SA after the initial exchanges are completed.

A first child SA is created by the `IKE_AUTH` exchange in phase 1. Additional child SAs can be negotiated by sending `CREATE_CHILD_SA` requests. To enable stronger guarantees (e. g. PFS), an optional Diffie-Hellman token may be included in the request. The keying

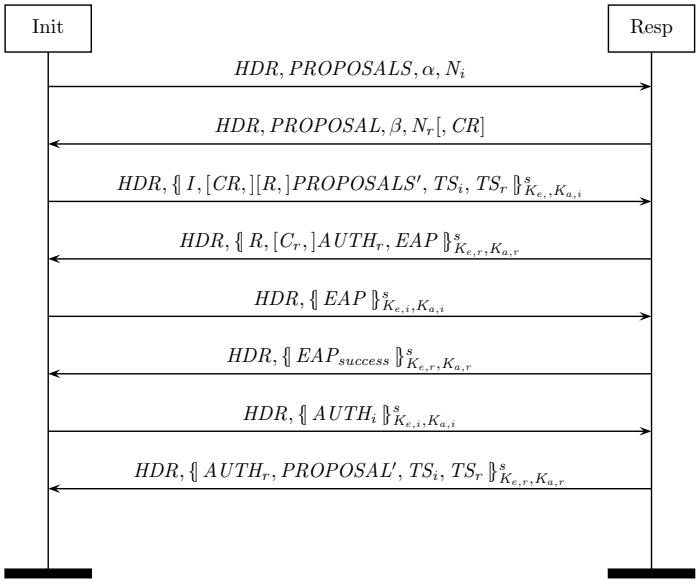

Figure 4.10: IKEv2 Extensible Authentication Protocol

material for the child SA is a function of K_d established during phase 1, the nonces N_i and N_r, and Z', a fresh Diffie-Hellman secret (if included in the `CREATE_CHILD_SA` exchange). The exchange is depicted in Figure 4.11.

Definition 17 (IPSec keying material)**.** *Let N'_i and N'_r be fresh nonces and let $Z' = \alpha'^{x'_r} = \beta'^{x'_r}$ denote a fresh DH secret. The keying material for IPSec SAs is defined as*

$$KEYMAT_1 \stackrel{def}{=} \mathsf{prf}\left(K_d, N_i \cdot N_r\right), \tag{4.29}$$

for the security association established in phase 1, and

$$KEYMAT_n \stackrel{def}{=} \mathsf{prf}\left(K_d, [Z'] \cdot N'_i \cdot N'_r\right), \quad n > 1 \tag{4.30}$$

for all subsequent SAs (established via `CREATE_CHILD_SA`).

The initiator sends SA offer(s) in $PROPOSALS$, a nonce N'_i, an optional DH token α', and the proposed traffic selectors in TS_i and TS_r. If this exchange is rekeying an existing IPSec SA, the leading N denotes the message id and is used to identify the SA being rekeyed, otherwise it is omitted. The message following the header is encrypted and the message including the header is integrity protected using the cryptographic algorithms negotiated for the IKE SA.

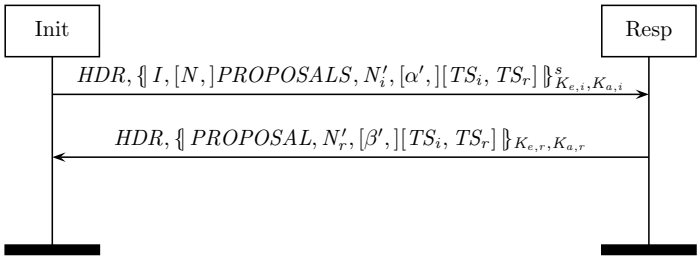

Figure 4.11: IKEv2 `CREATE_CHILD_SA`

The responder replies (using the same message id to respond) with the accepted SA offer in $PROPOSAL$, a nonce N'_r, and a Diffie-Hellman value β' if such a token was included in the request.

Chapter 5

Modeling Protocols

Before we present the results of our analysis, we briefly explain how we modeled the IKE protocols from Chapter 4 in the framework of Chapter 3. The design of protocol models in a given formalism is always a tradeoff between being precise, i. e., we do not want to deviate too much from the specification, and being pragmatic, e. g., it makes no sense to model TCP/IP specific details in a model where protocol participants are identified by just their name. Moreover, most formal models exhibit certain restrictions. In our case for example, operations on sets and lists are not supported in the formalism.

The aim of this chapter is to present some of the workarounds and abstractions we had to perform during the modeling phase, and to facilitate the correct interpretation of our experiments, which we present in the next chapter. The protocol models (cf. Section 5.5) can be found online at [1].

5.1 General Abstractions

SA negotiation was greatly abstracted and simplified. Instead of modeling the negotiation as the responder selecting from a list of proposals sent by the initiator, we modeled it as a simple nonce exchange, where the nonce created by the initiator ($PROPOSALS$) represents the list of security association proposals and the responder's nonce ($PROPOSAL$) represents a choice thereof. This approach bases on the assumption that, although the initiator can propose numerous security associations, the responder can only pick one, and still gives enough flexibility to express certain known agreement problems [20]. A drawback of this solution is the loss of dependency between the proposal and the choice. However, we

do not know of any attack that exploits this kind of dependency, i. e., making a responder choose a proposal which has not been proposed by the initiator.

We did not account for public key certificates since their only purpose is to indicate which public key should be used for encryption or signature verification and as such are a technical facilitation without any security relevance. Moreover, we abstracted away from IKEv2's traffic selectors. Traffic selectors are used to enrich the security association with information on what kind of traffic is allowed to be transmitted over the SA, and are meaningless in our abstracted communication model.

5.2 Modular Exponentiation

Our verification tool works in a free term algebra where equality is defined as syntactic equality over terms. This poses some problems for modeling, e. g., modular exponentiation.[1] Nevertheless, our tool allows the definition of "helper" protocols, which we call *oracles*. Oracles can be queried by the adversary at protocol execution to gain certain information based on information he or she already has. We describe their behavior shortly, for now we simply state that oracles can be compared to the rewrite rules of Meadows [36]. Oracles allow us to under-approximate the properties of a key derived by the Diffie-Hellman algorithm. In the following, we assumed that the derivation of a Diffie-Hellman key (in our model) involved distinguishable operations for initiator and responder, whereas in reality, they are both the same.

To capture modular exponentiation, we defined two functions $f(x)$ and $h(G, y)$. The initiator computes her public Diffie-Hellman token α by computing $f(x_i) := g^{x_i}$ where x_i represents the initiator's private exponent and g denotes the agreed upon group generator. Similarly, the responder derives β. To derive the final Diffie-Hellman key Z_i, the initiator computes $h(\beta, x_i) := \beta^{x_i}$. Likewise, the responder computes $Z_r = h(\alpha, x_r)$. Note that if initiator and responder exchange their DH tokens correctly, they both derive the *same* Diffie-Hellman key.

$$Z_i \leftarrow h(f(x_r), x_i) \tag{5.1}$$

$$Z_r \leftarrow h(f(x_i), x_r) \tag{5.2}$$

[1] The extension of the Scyther tool to support modular exponentiation is ongoing work.

To express the equivalence $Z_i = (g^{x_r})^{x_i} \equiv (g^{x_i})^{x_r} = Z_r$ in our protocol model, we let the adversary query predefined oracles at runtime. The oracles were defined with respect to the protocol specification, i.e. if the initiator sends a message m which is encrypted under a DH key $Z_i = h(f(x_r), x_i)$, $\{\!|\, m \,|\!\}_{Z_i}$, then the adversary, by querying a "Diffie-Hellman oracle", learns $\{\!|\, m \,|\!\}_{Z_r}$ where $Z_r = h(f(x_i), x_r)$.

The functions together with the oracles obey *some* of the algebraic rules associated with Diffie-Hellman, including the ones necessary for the computation of a Diffie-Hellman key, but not all of them. Hence any attack we find will remain valid under the full set of Diffie-Hellman equivalences, but the absence thereof will not prove anything.

5.3 Functions, Key Derivation, and Authenticators

IKEv1 and IKEv2 both make use of cryptographic functions such as (keyed) pseudo-random functions or cryptographic hashes. In this section we briefly describe how functions are represented within our model and highlight some of the restrictions we faced while modeling IKE session keys and authenticators. We give detailed descriptions of how we simplified the various constructs from the specification and how we modeled, e. g., session keys and authenticators.

First, we note that apart from encryption, decryption, and tupling, all other function applications are represented as collision-free hash functions in our model.[2] In particular this means that, in general, the results obtained from function application are irreversible and we therefore conjecture that repeated application does not increase the security of our models.

5.3.1 Key Derivation

Key derivation is a multi-step process in both protocols. First, a master secret is agreed upon, and second, numerous keys are generated as a function of the master secret and the (already) derived keys themselves (cf. Fig. 4.2 and 4.9). We simplify this process by providing a suitable key derivation function which, with the exception of f and h from Section 5.2, contains no nested function applications.

[2] As implied by Definition 2, hash functions can consume an arbitrary number of input values

IKEv1 As a first step in simplifying the three keys K_d, K_a, and K_e from Definition 10, we remove the nested application of prf which is caused by $SKEYID$. We stress that we only abstract from the application itself, the arguments remain untouched. We also replace term concatenation by tupling. For digital signature authenticated IKEv1 where $SKEYID = \mathsf{prf}(N_i \cdot N_r, Z)$ we obtain:

$$K_d \leftarrow \mathsf{prf}\left((N_i, N_r, Z), Z, CKY_i, CKY_r, 0\right) \tag{5.3}$$

$$K_a \leftarrow \mathsf{prf}\left((N_i, N_r, Z), K_d, Z, CKY_i, CKY_r, 1\right) \tag{5.4}$$

$$K_e \leftarrow \mathsf{prf}\left((N_i, N_r, Z), K_a, Z, CKY_i, CKY_r, 2\right) \tag{5.5}$$

Similarly, we delete K_d and K_a from the definitions of K_a and K_e, respectively. Also, we simplify the definitions such that the arguments to the pseudo-random functions describe a set instead of a bag.

$$K_d \leftarrow \mathsf{prf}\left(N_i, N_r, Z, CKY_i, CKY_r, 0\right) \tag{5.6}$$

$$K_a \leftarrow \mathsf{prf}\left(N_i, N_r, Z, CKY_i, CKY_r, 1\right) \tag{5.7}$$

$$K_e \leftarrow \mathsf{prf}\left(N_i, N_r, Z, CKY_i, CKY_r, 2\right) \tag{5.8}$$

Because constants do not improve the security of the derivation scheme, we omit them in our model. We observe that the three keys are now equal (modulo their identifiers). Thus, we model the three (originally unique keys) as a single session key, with variants for initiator and responder, respectively.

$$SK_i \leftarrow \mathsf{KDF}\left(N_i, N_r, Z_i, CKY_i, CKY_r\right) \tag{5.9}$$

$$SK_r \leftarrow \mathsf{KDF}\left(N_i, N_r, Z_r, CKY_i, CKY_r\right). \tag{5.10}$$

KDF is a new function introduced to capture the essentials of the key derivation process. Due to the way Diffie-Hellman secrets are represented in our model (see Section 5.2), the two variants are semantically equal.

IKEv2 Instead of three, IKEv2 defines 7 short-term keys in its specification. After applying the same procedure to these keys as above, we obtain

$$SK_i \leftarrow \mathsf{KDF}\,(N_i, N_r, Z_i, SPI_i, SPI_r) \tag{5.11}$$

$$SK_r \leftarrow \mathsf{KDF}\,(N_i, N_r, Z_r, SPI_i, SPI_r). \tag{5.12}$$

By simplifying key derivation we are able to further reduce the complexity of our models: the specified two-fold encryption with different symmetric keys $(K_{a,x}, K_{e,x})$ is no longer useful and is omitted in all IKEv2 protocol models.

5.3.2 Authenticators

The representation of authenticators posed similar challenges as key derivation. Below, we briefly describe the transformation from the authenticators defined in Chapter 4 to the way we represented them in our models.

IKEv1 The IKE specification defines several authentication payloads, the most prominent ones being $HASH$ and SIG. As the latter is just a signed version of the first, we concentrate on the transformation of $HASH$. Other hashes mentioned by the specification were transformed in a similar fashion and are not covered here.

The initiator's as well as the responder's authenticator, as defined in Definition 11, is obtained from applying a keyed pseudo-random function to a number of exchanged values. To simplify authentication, we first replace the key K_a by either SK_i or SK_r, depending on whose authenticator is being modeled.

$$HASH_i \leftarrow \mathsf{prf}\,(SK_i, \alpha, \beta, CKY_i, CKY_r, PROPOSALS, I) \tag{5.13}$$

$$HASH_r \leftarrow \mathsf{prf}\,(SK_r, \beta, \alpha, CKY_r, CKY_i, PROPOSALS, R) \tag{5.14}$$

Following the argument of Section 5.3.1, we remove nested function applications and remove duplicates from the input arguments. Diffie-Hellman tokens of the form g^x are substituted by $f(x)$.

$$HASH_i \leftarrow \mathsf{prf}\,(N_i, N_r, Z_i, f(x_i), f(x_r), CKY_i, CKY_r, PROPOSALS, I) \tag{5.15}$$

$$HASH_r \leftarrow \mathsf{prf}\,(N_i, N_r, Z_r, f(x_r), f(x_i), CKY_r, CKY_i, PROPOSALS, R) \tag{5.16}$$

IKEv2 Analogous to the simplifications above, we adapt the digital signature authenticators from Definition 15.

$$AUTH_i \leftarrow \{\!|\, SPI_i, 0, PROPOSALS, f(x_i), N_i, N_r, \mathsf{prf}\,(SK_i, I)\,|\!\}^a_{sk(I)} \quad (5.17)$$

$$AUTH_r \leftarrow \{\!|\, SPI_i, SPI_r, PROPOSAL, f(x_r), N_r, N_i, \mathsf{prf}\,(SK_r, R)\,|\!\}^a_{sk(R)} \quad (5.18)$$

Similarly, we modify Definition 16.

$$AUTH_i \leftarrow \mathsf{prf}(k(I,R), SPI_i, 0, PROPOSALS, f(x_i), N_i, N_r, \mathsf{prf}\,(SK_i, I)) \quad (5.19)$$

$$AUTH_r \leftarrow \mathsf{prf}(k(R,I), SPI_i, SPI_r, PROPOSAL, f(x_r), N_r, N_i, \mathsf{prf}\,(SK_r, R)) \quad (5.20)$$

5.4 Modeling Security Goals

Unfortunately, the specifications of IKEv1 as well as IKEv2 do not clearly state what security goals the protocols should achieve. The only goal which is explicitly mentioned by both RFCs is PFS. Further, we could derive that the protocols should achieve *confidentiality* and *strong entity authentication*. IKEv1 should additionally provide *anonymity* in MM (i. e., identities should not be revealed to the adversary) and *plausible deniability* (i. e., agents should be able to deny their participation in a given protocol execution) for public key authenticated exchanges.

Additionally to these somehow informally stated security goals we modeled numerous desirable attributes of key agreement protocols [10, section 2]. We discuss them below.

1. (Resistance to) *known session keys*. A protocol should still achieve its goals in the face of an adversary who has learned session keys that were established in previous sessions.

2. *Perfect Forward Secrecy*. The secrecy of short-term keys must not be affected by the compromise of one or many long-term secrets.

3. (resistance to) *unknown key-share*. An agent A cannot be coerced into sharing a session key with agent B without A's knowledge, i. e., when A believes the key is shared with agent $E \neq B$.

4. (Resistance to) *Key Compromise Impersonation*. The protocol's ability to prevent the adversary from impersonating other agents to an agent A if A's long-term secrets

have been revealed.

5. *Loss of information.* Compromise of other information that would not ordinarily be available to the adversary does not affect the security of the protocol.

6. *Message independence.* Individual flows of a protocol run between two honest agents are unrelated.

5.5 IKE Protocol Models

So far, we covered general simplifications and abstractions. In this section we describe our approach to not only master the complexity of large-scale security protocols but also to describe them as executable models.

Following the strategy of *Divide and Conquer* we extracted numerous subprotocols from the specifications of IKEv1 and IKEv2. First, we identified all the obvious candidates from Chapter 4 such as the different phases, modes, and the various authentication methods. We came up with 9 subprotocols for IKEv1 (two different modes, each of which supporting four authentication schemes, and quick mode) and four subprotocols for IKEv2 (three authentication methods and IKEv2's phase 2 exchange). A crucial design choice facilitated the separate modeling of phase 1 and 2: the keys established during phase 1 are treated as (independent) pre-shared keys when used in phase 2. This not only allowed us to model phase 1 and 2 independently but also facilitated the analysis of PFS. Second, whenever one of the subprotocols specified optional message payload fields, we would create a new subprotocol which does not include the optional fields (while keeping the original subprotocol unchanged). Finally, we modeled additional subprotocols to cover the parts of the specification which are open to interpretation. For example, it is not clear whether an implementation of IKEv1's public key main mode would encrypt the identity together with the nonce ($\{\!|\, N_i, I \,|\!\}^a_{pk(R)}$), or separately (($\{\!|\, N_i \,|\!\}^a_{pk(R)}$, $\{\!|\, I \,|\!\}^a_{pk(R))}$).

In the following, we give graphical descriptions of all the variants we derived from the specifications. A brief description of their distinctive properties can be found in Tables 5.1-5.4. We did not model IKEv1's new group mode because the implications of changing the Diffie-Hellman group are outside the scope of this thesis. We also excluded the informational exchanges of both IKEv1 and IKEv2 from the scope because its purpose is unrelated key exchange.

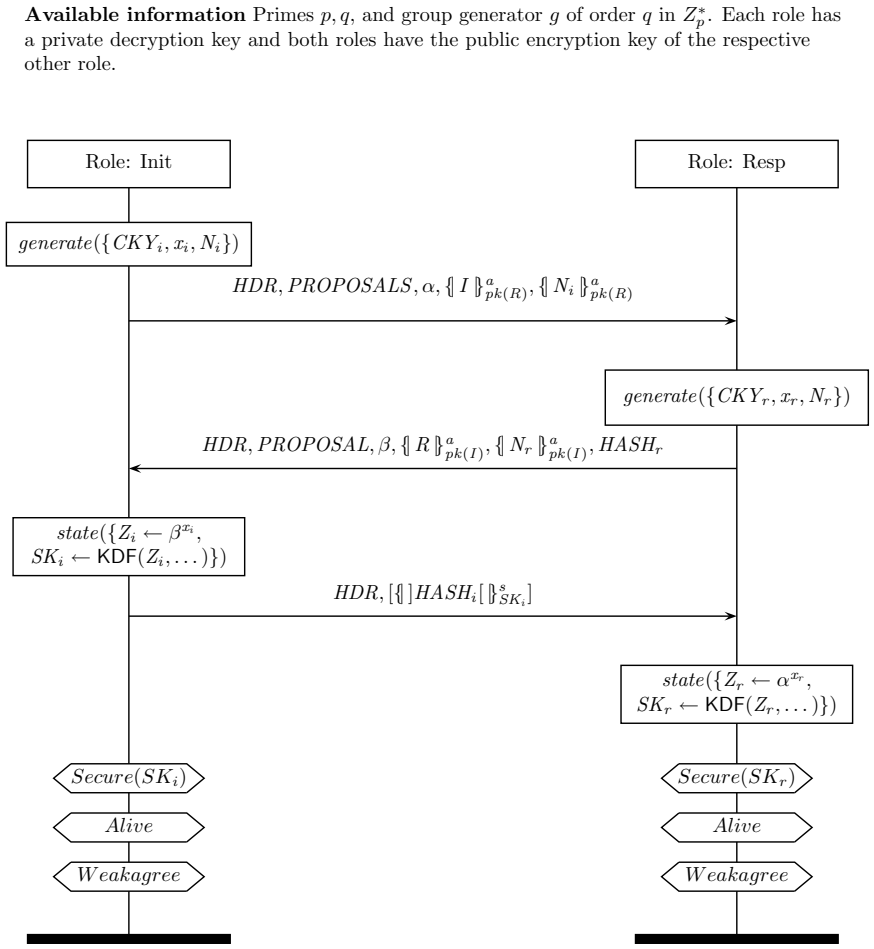

Figure 5.1: Protocol model of public key authenticated IKEv1 AM: Note that the initiator could delay the computation of Z_i and SK_i if the last message is not encrypted (which is optional).

Notation For the graphical representation of our protocol models we rely on the syntax of Mauw and Bos [35]. All models contain two roles, the initiator (Init) and the responder (Resp). Roles execute threads. Message exchange between the roles is illustrated with labeled arrows, where the label denotes the message to be transmitted. Role actions, such as the generation of fresh terms or state manipulation, are marked as boxes within the thread. Finally, hexagons mark the security properties to be analyzed (cf.. Chapter 3). Note that due to space constraints, agent names are abbreviated.

Available information Primes p, q, and group generator g of order q in Z_p^*. Each role has a private decryption key and both roles have the public encryption key of the respective other role.

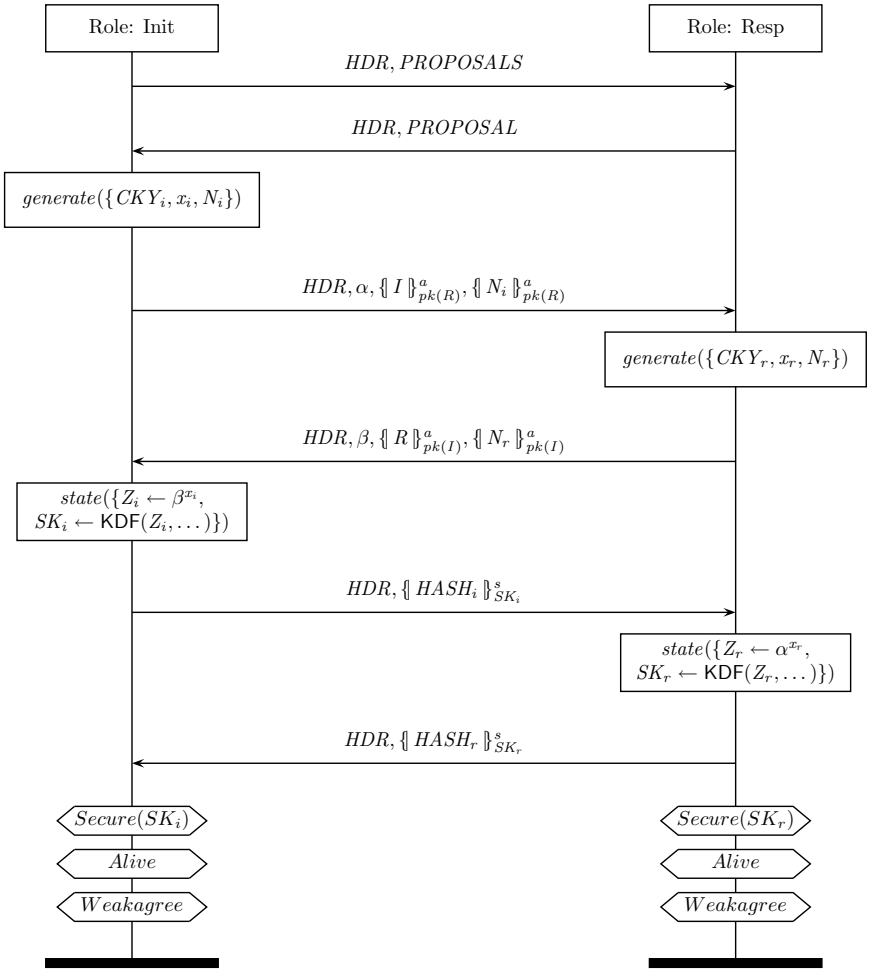

Figure 5.2: Protocol model of public key authenticated IKEv1 MM

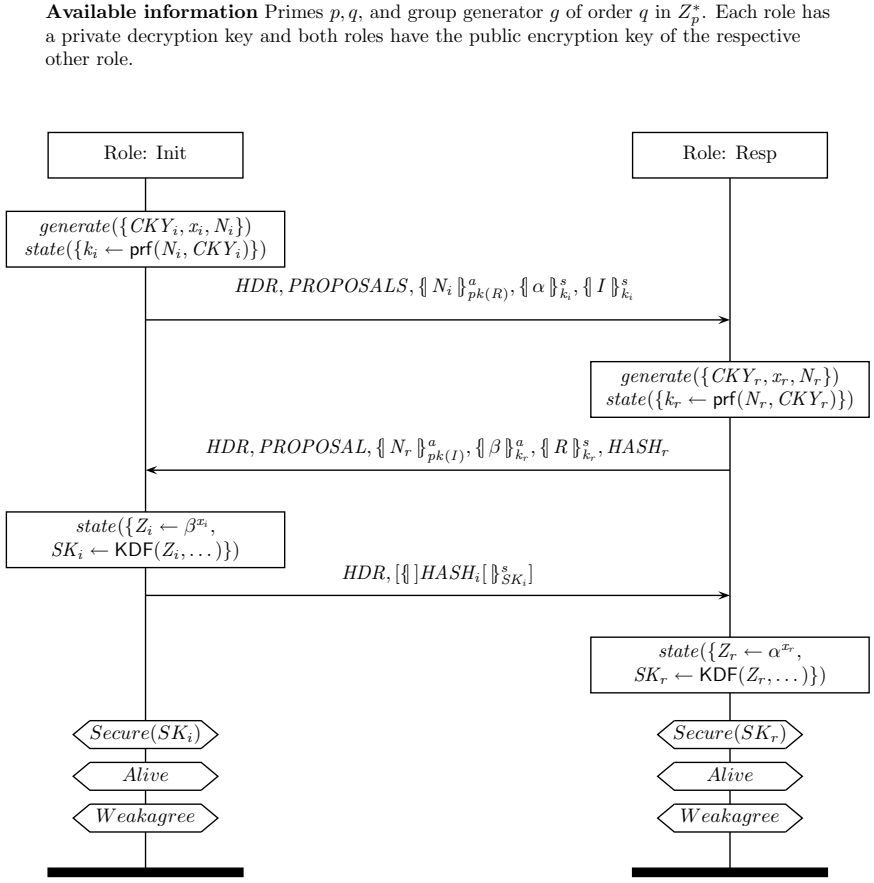

Figure 5.3: Protocol model of the revised version of public key authenticated IKEv1 AM

Available information Primes p, q, and group generator g of order q in Z_p^*. Each role has a private decryption key and both roles have the public encryption key of the respective other role.

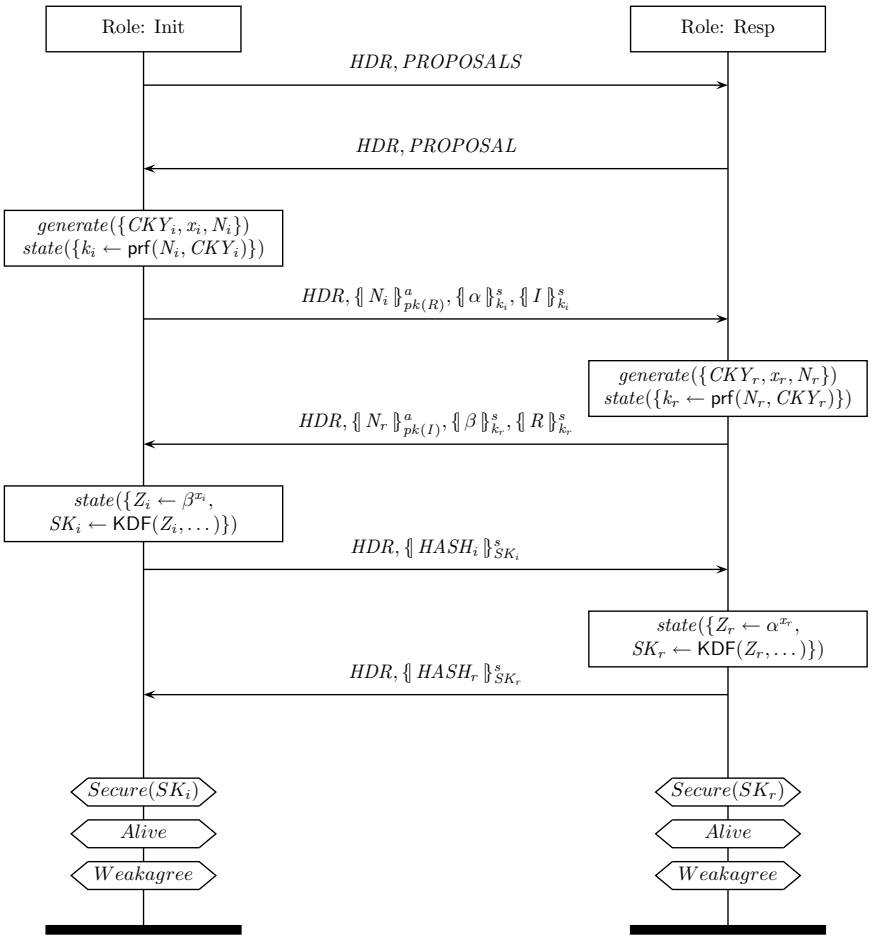

Figure 5.4: Protocol model of the revised version of public key authenticated IKEv1 MM

Available information Primes p, q, and group generator g of order q in Z_p^*. Initiator and responder possess pre-shared keys $k(I, R)$ and $k(R, I)$, respectively.

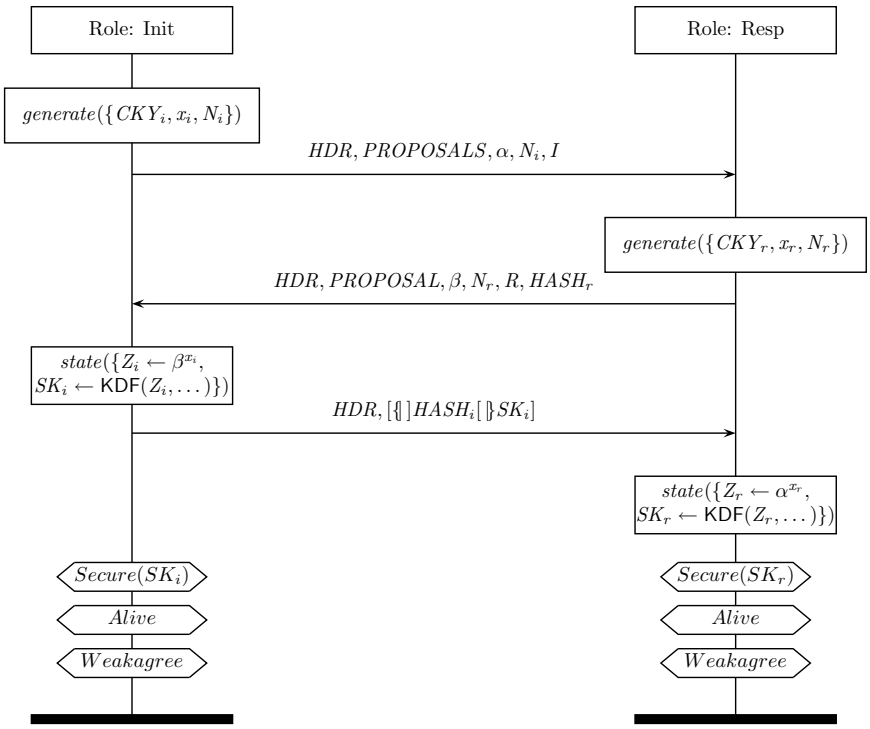

Figure 5.5: Protocol model of pre-shared key authenticated IKEv1 AM: Note that the initiator could delay the computation of Z_i and SK_i if the last message is not encrypted (which is optional).

Available information Primes p, q, and group generator g of order q in Z_p^*. Initiator and responder possess pre-shared keys $k(I, R)$ and $k(R, I)$, respectively.

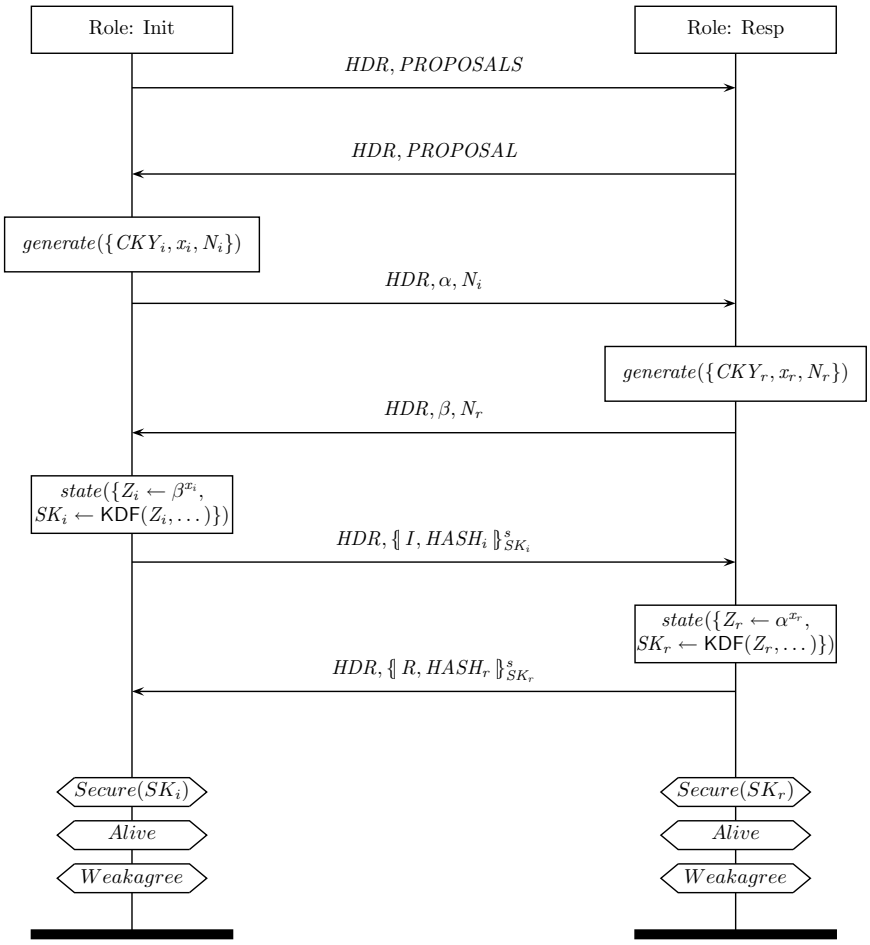

Figure 5.6: Protocol model of pre-shared key authenticated IKEv1 MM

Available information Primes p, q, and group generator g of order q in Z_p^*. Each role has a private signature key and both roles have the public verification key of the respective other role.

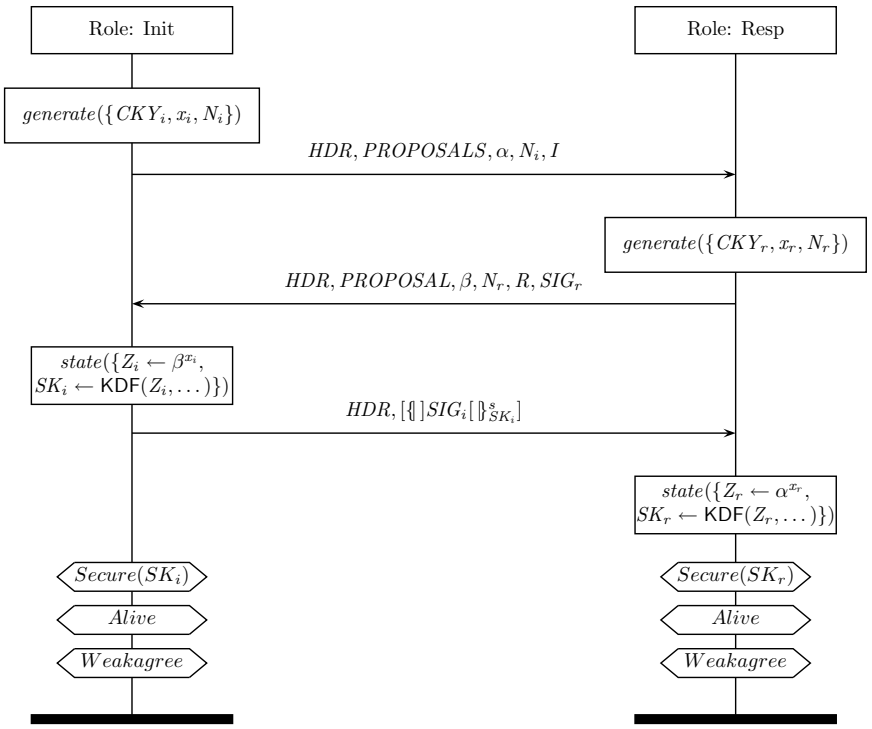

Figure 5.7: Protocol model of signature authenticated IKEv1 AM: Note that the initiator could delay the computation of Z_i and SK_i if the last message is not encrypted (which is optional).

Available information Primes p, q, and group generator g of order q in Z_p^*. Each role has a private signature key and both roles have the public verification key of the respective other role.

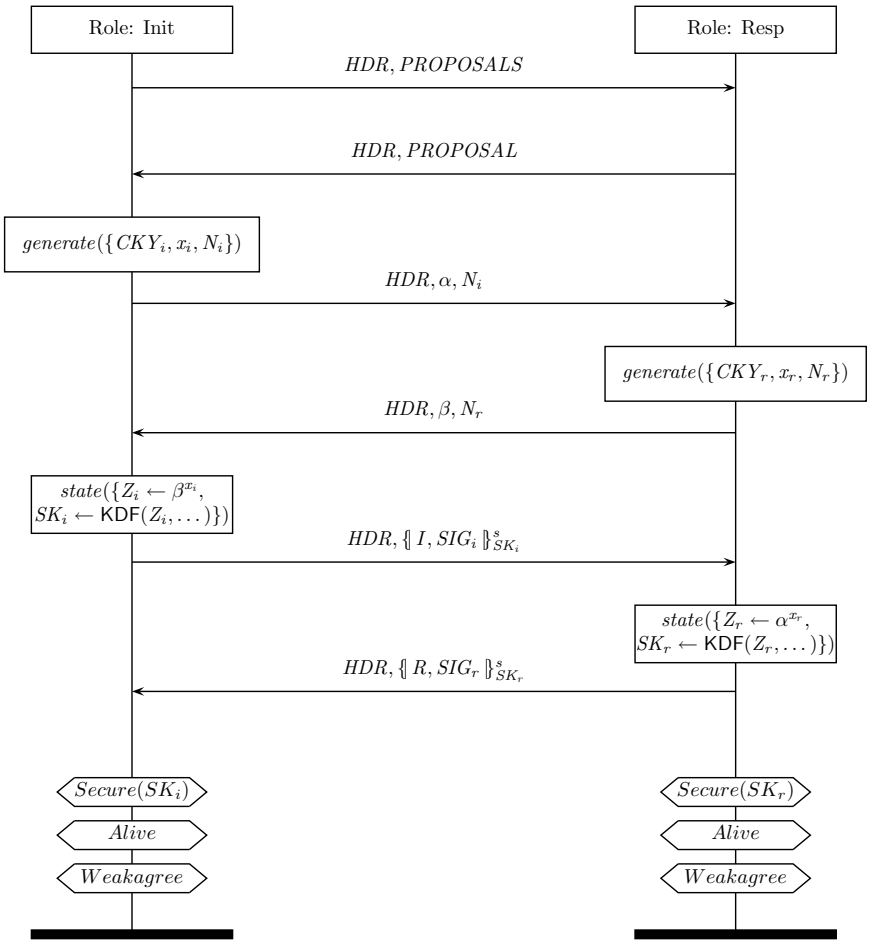

Figure 5.8: Protocol model of signature authenticated IKEv1 MM

Available information Primes p, q, and group generator g of order q in Z_p^*. Initiator and responder possess pre-shared keys $k(I, R)$ and $k(R, I)$, respectively, denoting the keys established during phase 1.

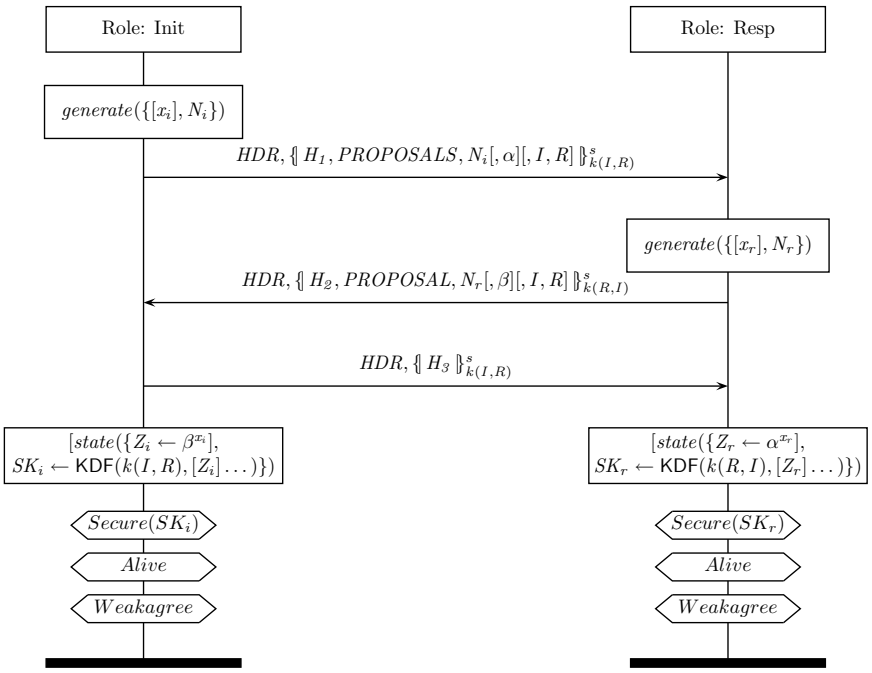

Figure 5.9: Protocol model of IKEv1 QM

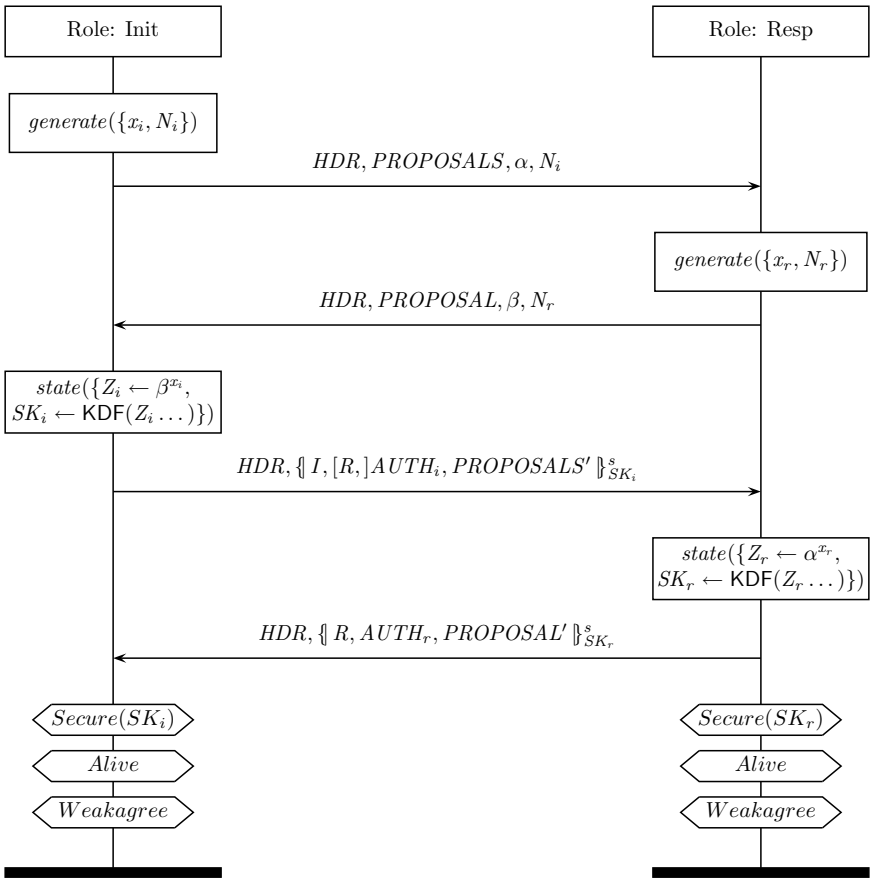

Figure 5.10: Protocol model of IKEv2's standard exchange

Available information Primes p, q, and group generator g of order q in Z_p^*. Each role has a private signature key and both roles have the public verification key of the respective other role.

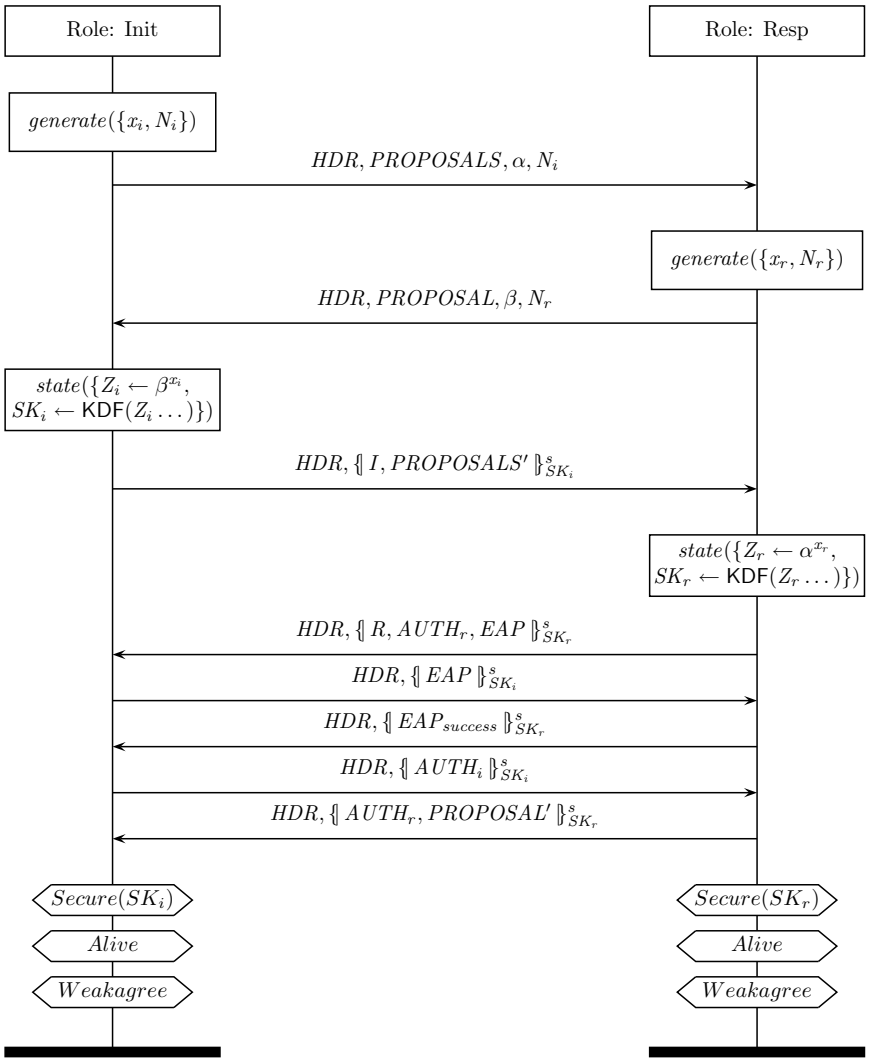

Figure 5.11: Protocol model of IKEv2's EAP exchange

Available information Primes p, q, and group generator g of order q in Z_p^*. Initiator and responder possess pre-shared keys $k(I, R)$ and $k(R, I)$, respectively, denoting the keys established during phase 1.

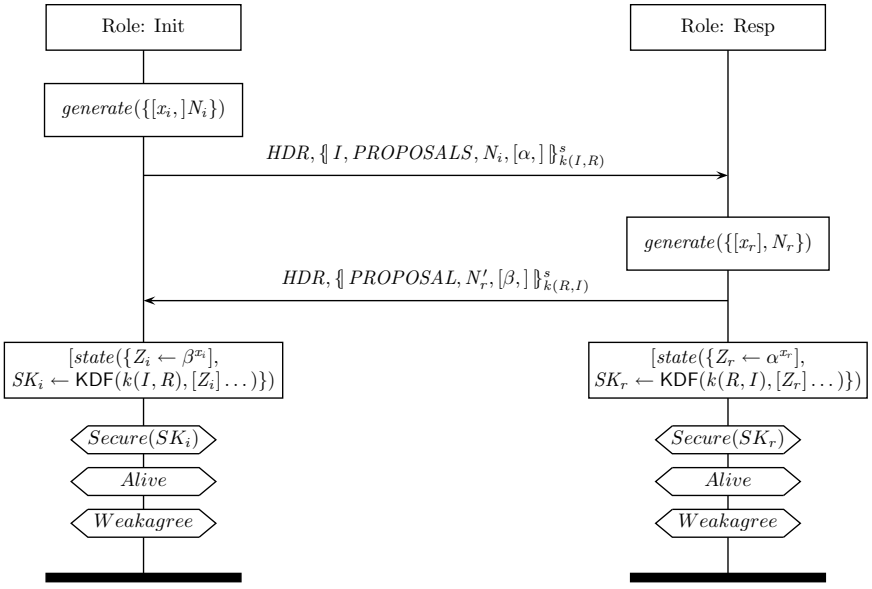

Figure 5.12: Protocol model of IKEv2's `CREATE_CHILD_SA` exchange

Mode	i	**Pid**	**Encrypt last msg**	**Comments**
AM	1	**ikev1-sig-a1**	×	-
AM	2	**ikev1-sig-a2**	✓	-
AM	3	**ikev1-sig-a-perlman1**	×	Includes a fix suggested by Perlman and Kaufman to guarantee the initiator's anonymity even in the presence of an active adversary. The responder's anonymity is guaranteed only for passive attackers.
AM	4	**ikev1-sig-a-perlman2**	✓	Includes the same fix as above.
MM	5	**ikev1-sig-m**	-	-
MM	6	**ikev1-sig-m-perlman**	-	Includes a fix suggested by Perlman and Kaufman to provide anonymity for the initiator in the presence of an active adversary.

Table 5.1: Variants of IKEv1 signature authenticated message exchanges: the column "Encrypt last msg" only is relevant for AM exchanges and indicates whether the last message is encrypted or not (doing so is optional according to the specification).

Mode	i	**Pid**	**Encrypt last msg**	**Separate enc.**	**Comments**
AM	7	**ikev1-pk-a1**	×	✓	-
AM	8	**ikev1-pk-a12**	✓	✓	-
AM	9	**ikev1-pk-a2**	×	×	-
AM	10	**ikev1-pk-a22**	✓	×	-
AM	11	**ikev1-pk2-a**	×	✓	Revised version of public key based authentication
AM	12	**ikev1-pk2-a2**	×	×	Revised version of public key based authentication
MM	13	**ikev1-pk-m**	-	✓	-
MM	14	**ikev1-pk-m2**	-	×	-
MM	15	**ikev1-pk2-m**	-	✓	Revised version of public key based authentication
MM	16	**ikev1-pk2-m2**	-	×	Revised version of public key based authentication

Table 5.2: Variants of IKEv1 public key authenticated message exchange: the column "Encrypt last msg" only is relevant for AM exchanges and indicates whether the last message is encrypted or not. "Separate enc." indicates the format used to encrypt nonce and identity, e. g., whether $(\{\!|\, N_i \,|\!\}^a_{pk(R)}, \{\!|\, I \,|\!\}^a_{pk(R)})$ or $\{\!|\, N_i, I \,|\!\}^a_{pk(R)}$ is used.

Mode	i	**Pid**	**Comments**
AM	17	**ikev1-psk-a**	-
MM	18	**ikev1-psk-m**	-
MM	19	**ikev1-psk-m-perlman**	Includes a fix suggested by Perlman and Kaufman to provide anonymity for both endpoints.
QM	20	**ikev1-quick**	Includes optional identities as well as optional Diffie-Hellman tokens to support PFS.
QM	21	**ikev1-quick-noid**	Includes optional Diffie-Hellman tokens only.
QM	22	**ikev1-quick-nopfs**	Create additional IPSec SAs without support for PFS. The optional identities are omitted as well.

Table 5.3: Variants of IKEv1 pre-shared key authenticated message exchange and QM

Mode	i	**Pid**		**Comments**
SIG	23	**ikev2-sig**	✓	-
SIG	24	**ikev2-sig2**	×	-
MAC	25	**ikev2-mac**	✓	-
MAC	26	**ikev2-mac2**	×	-
EAP	27	**ikev2-eap**	✓	-
EAP	28	**ikev2-eap2**	×	-
29	SM	**ikev2-sigtomac**	✓	The initiator authenticates itself using digital signatures while the responder is authenticated by using a MAC. This variant stems from [26, Section 2.15].
30	SM	**ikev2-sigtomac2**	×	The initiator authenticates itself using digital signatures while the responder is authenticated by using a MAC. This variant stems from [26, Section 2.15].
31	SM	**ikev2-mactosig**	✓	The initiator authenticates itself using a MAC while the responder is authenticated by using digital signatures. This variant stems from [26, Section 2.15].
32	SM	**ikev2-mactosig2**	×	The initiator authenticates itself using a MAC while the responder is authenticated by using digital signatures. This variant stems from [26, Section 2.15].
33	C	**ikev2-child**	-	Includes optional Diffie-Hellman tokens to support PFS.
34	C	**ikev2-child-nopfs**	-	Create additional IPSec SAs without PFS support.

Table 5.4: Variants of IKEv2 message exchanges: the column "Optional identity" indicates whether the initiator transmits the responder's identity in message 3 or not.

Chapter 6

Analysis

In this chapter we use the symbolic security model of Chapter 3 to perform an extensive security analysis of IKEv1 and IKEv2 using the models of Chapter 5. In our experiments we automatically rediscover several attacks that were previously reported in the literature. Moreover, we discover additional, previously unreported weaknesses and highlight, by establishing a security hierarchy after [6], the differences between IKEv1 and its successor, IKEv2.

We proceed by presenting our approach in Section 6.1. Subsequently we describe rediscovered attacks in Section 6.2, novel attacks in Section 6.3, and outline several undiscovered, yet known vulnerabilities in Section 6.4. We conclude the analysis by establishing a security hierarchy among IKEv1 and IKEv2.

Notational preliminaries All attacks presented throughout this chapter should work on the minimal implementations (unless noted otherwise) and will be described textually and graphically. For the graphical representation we again rely on the syntax of Mauw and Bos [35]. Graphical attack descriptions are given in the form of a (property violating) trace, e. g., Fig. 6.1. An attack trace typically consists of two threads, one of them being the *test* thread. Both threads are executed by agents performing a certain role. For example, in Fig. 6.1(a), Alice performs the initiator role and Bob performs the responder role. If partnering is relevant for the attack, the partner relationship is expressed in the thread header. Agent actions, e. g., the generation of terms, are denoted as boxes containing event identifiers from Definition 1. Due to space constraints we dispense with marking state content unless it is essential to the understanding of the attack. A send event is indicated

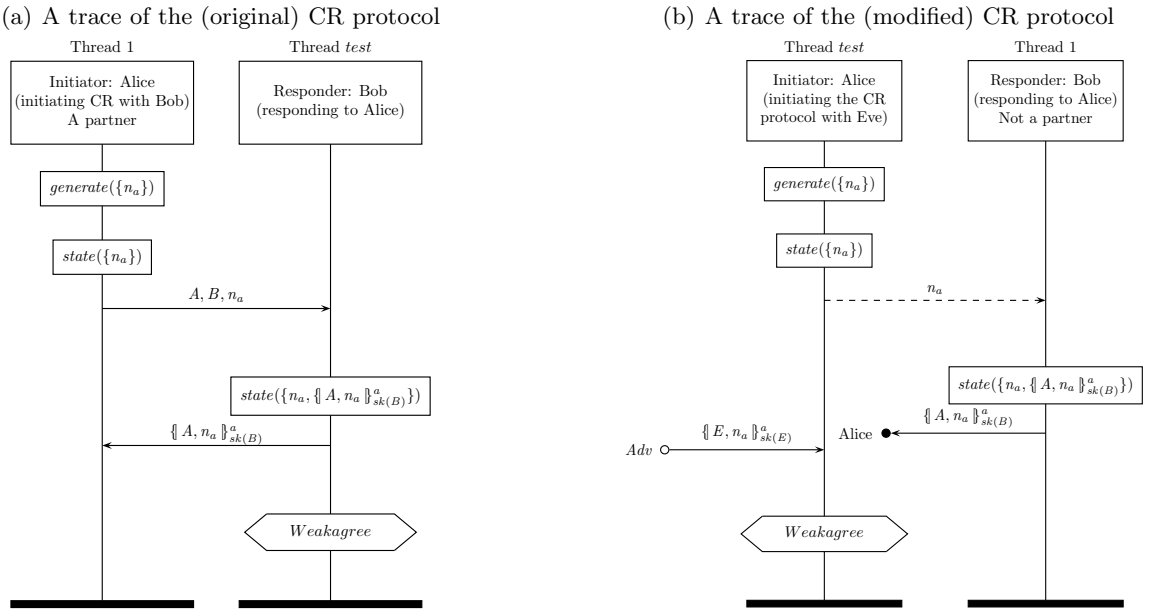

Figure 6.1: Two examples of CR execution traces

by an outbound arrow from the sender where the label denotes the message to be sent (message payload, except for identities, appears in lower-case). Conversely, a recv event is expressed as an inbound arrow. A solid arrow from sender to receiver indicates that the adversary did not tamper with the message in any way. Redirection of messages by the adversary is expressed with dashed arrows. An arrow pointing to nowhere indicates that the adversary blocked or intercepted the message (the intended recipient is given at the arrow tip). Message injection is expressed as arrows which originate from *Adv*. Finally, hexagons mark the property to be analyzed. Note that due to space constraints, we often abbreviate agent names in messages.

6.1 Approach

6.1.1 Settings

Tool selection We performed all our experiments using the automatic protocol verification tool Scyther [15] (*version id: scyther-compromise-0.5*). Scyther, on a high level, operates as follows. When given a protocol description as input, it infers from it the system which describes the behavior of agents in the context of an adversary. It then explores

the state space within that system and verifies a number of runs, bounded or unbounded, using a symbolic backwards search based on patterns [16]. By default, Scyther explores the system with a bounded number of runs (5), is guaranteed to terminate, and one of the following three situations can occur. First, the tool can establish that a certain property holds for the bounded system (but not necessarily for the unbounded). Second, the property is false, resulting in an attack pattern. Third, the property can be proven correct for the unbounded system.

Hardware selection To perform our large-scale experiments, we relied on the Brutus Cluster, the central high-performance computing environment of ETH Zurich. Brutus is a heterogeneous system with a total of 9912 processor cores in 1108 compute nodes. The peak performance of Brutus is approximately 90 teraflops.

Additional infrastructure Because our verification tool itself is not optimized for a multi-core environment, we built a special-purpose infrastructure which allowed us to generate and distribute all sorts of verification jobs over the cluster, eventually collecting their output for later consolidation. For producing the security hierarchy, we relied on a suit of Python scripts which interface with the Scyther Python API and come bundled with the latest release. The scripts are available online at [1].

6.1.2 Adversary Models

In Chapter 3 we introduced a number of adversary-compromise rules, each of which can be used to model the environments where a given protocol is expected to operate in. For example, a protocol that is said to guarantee Perfect Forward Secrecy is expected to satisfy session key secrecy even if the adversary has the power to corrupt long-term secrets after the session has terminated. Any combination of adversary-compromise rules is called an adversary-compromise model. The number of possible models is thus $2^7 = 128$, where only 96 of them are relevant ($2^5 = 32$ models include $\mathsf{LKR}_{\mathsf{after}}$ and $\mathsf{LKR}_{\mathsf{aftercorrect}}$, where the latter is implied by the former). Verifying every protocol in 96 distinct adversary-models would have been infeasible in the amount of time we had planned for our experiments. Therefore we restricted ourselves to the 10 adversary-models presented in Table 6.1. These models were established in [7] as a result of unifying various existing adversary models from the computational setting into the symbolic framework our work is based on.

Model	$\mathsf{LKR}_{\mathsf{others}}$	$\mathsf{LKR}_{\mathsf{actor}}$	$\mathsf{LKR}_{\mathsf{after}}$	$\mathsf{LKR}_{\mathsf{aftercorrect}}$	SKR	SR	RNR	Origin
Adv_{EXT}								External Dolev-Yao
Adv_{INT}	✓							Dolev-Yao [32]
Adv_{CA}		✓						Key Compromise Impersonation [23]
Adv_{AFC}				✓				Weak Perfect Forward Secrecy [30]
Adv_{AF}			✓	✓				Perfect Forward Secrecy [19, 21]
Adv_{BR}	✓				✓			BR95 [9]
Adv_{CKw}	✓	✓		✓	✓	✓		CK2001-wPFS [30]
Adv_{CK}	✓		✓	✓	✓	✓		CK2001 [13]
Adv_{e-CK1}	✓				✓		✓	eCK [31]
Adv_{e-CK2}	✓	✓		✓	✓			eCK [31]

Table 6.1: Adversary-compromise models

6.1.3 Analysis

We analyzed all subprotocols from Chapter 5 with respect to the security goals from Chapter 3; *session key secrecy* (S), *aliveness* (A), and *weak agreement* (W). Other authentication properties such as *non-injective agreement* or *non-injective synchronization* were left out of scope, mainly for the reason that they would have failed for all protocols because of how we approximated the Diffie-Hellman exchanges (cf. Section 5.2). We then let Scyther individually verify each security goal in each of the adversary models from Table 6.1. In a first phase, we let Scyther compute an attack overview (cf. Table 6.2). For that, we limited the time, Scyther had available to investigate a single trace, to 20 minutes, and the number of runs to 4. The computation of Table 6.2 took less than 5 hours on Brutus (the rows were computed independently). In a second phase, we let Scyther verify each protocol in each adversary model without limiting investigation time. The purpose of this was twofold. First, we wanted to see whether our overview was complete. It was for the most part; only about 5% of the attacks in Table 6.2 were discovered in the second verification phase. Second, we wanted Scyther to produce the attack patterns, which we needed for further analysis and were not produced in the first phase. Again, we limited the number of runs

to 4. Now, the verification of a single goal took Scyther between several seconds and a few hours.

6.2 Automatically Rediscovered Attacks

The IKE protocol family has been subjected to both formal and informal analysis over the years and many researchers have revealed a variety of attacks. We were able to rediscover most of these attacks in the context of our analysis.

6.2.1 Reflection Attacks

6.2.1.1 Reflection Attacks Against IKEv1 MM

Ferguson and Schneier report in [20] that IKEv1 MM is susceptible to a reflection attack if the exchange is authenticated either with digital signatures or pre-shared keys. The attack is carried out by a dishonest responder who assumes the identity of the initiator and subsequently passes the authentication phase without being detected. They identify the symmetry (with regard to swapping the identities of initiator and responder) of the authenticators (cf. Definition 11) as the root cause of the attack. Under the assumption that the initiator accepts her own identity as the identity of her peer, the two authenticators are equal. Thus a reflection attack becomes possible. According to Boyd and Mathuria [12], the assumption of Alice accepting her own identity as the identity of her peer may not be unreasonable in a setting where only IP addresses are authenticated.

Our study rediscovers this type of reflection attack for both protocol variants as a violation of *weak agreement*. We stress that the adversary does not obtain the session key. We further observe that *all* phase 1 subprotocols of IKEv1 suffer from this flaw, which contradicts the statement of Ferguson and Schneier. We will present the attack against public key authenticated IKEv1 in Section 6.3.1.1 below.

6.2.1.2 A Reflection Attack Against IKEv1 Quick Mode

In [36], Meadows finds IKEv1 quick mode vulnerable to a reflection attack which includes multiple threads of the same agent. The attack (cf. Fig. 6.2) proceeds as follows. Alice wants to establish an IPSec SA with Bob and, for this purpose, initiates a quick mode session with him (to avoid confusion, we refer to this as the *test* session). The adversary

	Adv_{EXT}			Adv_{INT}			Adv_{CA}			Adv_{AFC}			Adv_{AF}			Adv_{BR}			Adv_{CKw}			Adv_{CK}			Adv_{eCK-1}			Adv_{eCK-2}		
	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W	S	A	W
Signature authenticators																														
ikev1-sig-a1	○			○			○			○			○			★		○	★		○	○		○	★		★	★		○
ikev1-sig-a2	○			○			○			○			○			★		○	★		○	○		○	★		★	★		○
ikev1-sig-a-perlman1	○			○			○			○			○			★		○	★		○	○		○	★		★	★		○
ikev1-sig-a-perlman2	○			○			○			○			○			★		○	★		○	○		○	★		★	★		○
ikev1-sig-m	○			○			○			○			○			★		○	★		○	○		○	★		★	★		○
ikev1-sig-m-perlman	○			○			○			○			○			★		○	★		○	○		○	★		★	★		○
ikev2-sig																									★		★			
ikev2-sig2	○			○			○			○			○			★		○	★		○	★		○	★		★	★		○
Public key authenticators																														
ikev1-pk-a1	○			○			○			○			○			★		○	★		○	○		○	★	★	★	★		○
ikev1-pk-a12	○			○			○			○			○			★		○	★		○	○		○	★	★	★	★		○
ikev1-pk-a2																			★			○			★	★	★			
ikev1-pk-a22																			★			○			★	★	★			
ikev1-pk2-a																									★	★	★			
ikev1-pk2-a2																									★	★	★			
ikev1-pk-m	★			★			★			★			★			★		★	★		★	○		★	★	★	★	★		★
ikev1-pk-m2	★			★			★			★			★			★		★	★		★	○		★	★	★	★	★		★
ikev1-pk2-m	○			○			○			○			○					○			○			○	★	★	★			○
ikev1-pk2-m2	○			○			○			○			○					○			○			○	★	★	★			○
Pre-shared key authenticators																														
ikev1-psk-a							★	★	★										★	★	★							★	★	★
ikev1-psk-m	○			○			★	★	★	○			○					○	★	★	★			○			○	★	★	★
ikev1-psk-m-perlman	○			○			★	★	★	○			○					○	★	★	★			○			○	★	★	★
ikev2-mac							★	★	★										★	★	★				★			★	★	★
ikev2-mac2							★	★	★										★	★	★				★			★	★	★

Continued on next page

Table 6.2: Attacks found against IKEv1 and IKEv2 subprotocols: automatically rediscovered attacks are marked with ○, new attacks are denoted by ★.

Table 6.2 – Continued from previous page

	Adv_{EXT}			Adv_{INT}			Adv_{CA}			Adv_{AFC}			Adv_{AF}			Adv_{BR}			Adv_{CKw}			Adv_{CK}			Adv_{eCK-1}			Adv_{eCK-2}		
	S	A	WA	S	A	WA	S	A	WA	S	A	WA	S	A	WA	S	A	WA	S	A	WA	S	A	WA	S	A	WA	S	A	WA
Other authenticators																														
ikev2-eap																									⋆	⋆	⋆			
ikev2-eap2																									⋆	⋆	⋆			
ikev2-mactosig							⋆	⋆	⋆										⋆	⋆	⋆				⋆		⋆	⋆	⋆	⋆
ikev2-mactosig2							⋆	⋆	⋆										⋆	⋆	⋆				⋆		⋆	⋆	⋆	⋆
ikev2-sigtomac							⋆	⋆	⋆										⋆	⋆	⋆				⋆		⋆	⋆	⋆	⋆
ikev2-sigtomac2			○			○	⋆	⋆	⋆			○			○	⋆		○	⋆	⋆	⋆	⋆		○	⋆		⋆	⋆	⋆	⋆
Phase 2 subprotocols																														
ikev1-quick							⋆	⋆	⋆										⋆	⋆	⋆							⋆	⋆	⋆
ikev1-quick-nopfs	○	○		○	○		⋆	⋆	⋆		○	○	⋆	○	○		○	○	⋆	⋆	⋆	⋆	○	○		○	○	⋆	○	○
ikev1-quick-noid	○	○		○	○		⋆	⋆	⋆		○	○		○	○		○	○	⋆	⋆	⋆		○	○		○	○	⋆	○	○
ikev2-child	⋆	⋆		⋆	⋆		⋆	⋆	⋆		⋆	⋆		⋆	⋆		⋆	⋆	⋆	⋆	⋆		⋆	⋆		⋆	⋆	⋆	⋆	⋆
ikev2-child-nopfs	⋆	⋆		⋆	⋆		⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆		⋆	⋆	⋆	⋆	⋆	⋆	⋆	⋆		⋆	⋆	⋆	⋆	⋆

intercepts her initial message and initiates another quick mode session with Alice (session 1), thereby impersonating Bob and replaying the previously intercepted message. Alice, in session 1, replies with her second message. The adversary again intercepts this message and redirects it to *test*, disguised as Bob's reply. Finally, Alice finishes *test* by sending her last message. The adversary intercepts this message and redirects it to session 1 so that Alice can finish session 1. The end result is that Alice winds up thinking that she shares *two* keys with Bob, whereas in fact she does not share any keys at all. As a matter of fact, Bob does neither send nor receive a single message.

At first, we were not able to reconfirm this attack because of the way we modeled (already established) shared keys within our framework: our first attempt was to model a key shared between Alice and Bob as $k(A, B)$, not reflecting the role assignment in the key. For example, Alice, acting as initiator, would encrypt a message m for Bob, who is acting as responder, as $\{\!|\, m \,|\!\}^s_{k(A,B)}$. Bob, on the other hand, would encrypt a message m' for Alice with the exact same key ($\{\!|\, m' \,|\!\}^s_{k(A,B)}$), even if he is performing the initiator role. The consequence was that the adversary could not inject messages from one session into another where the same agents were performing different roles. However, Meadows' attack bases on the assumption that a key shared between initiator Alice and responder Bob cannot be distinguished from the key shared between responder Alice and initiator Bob. After we

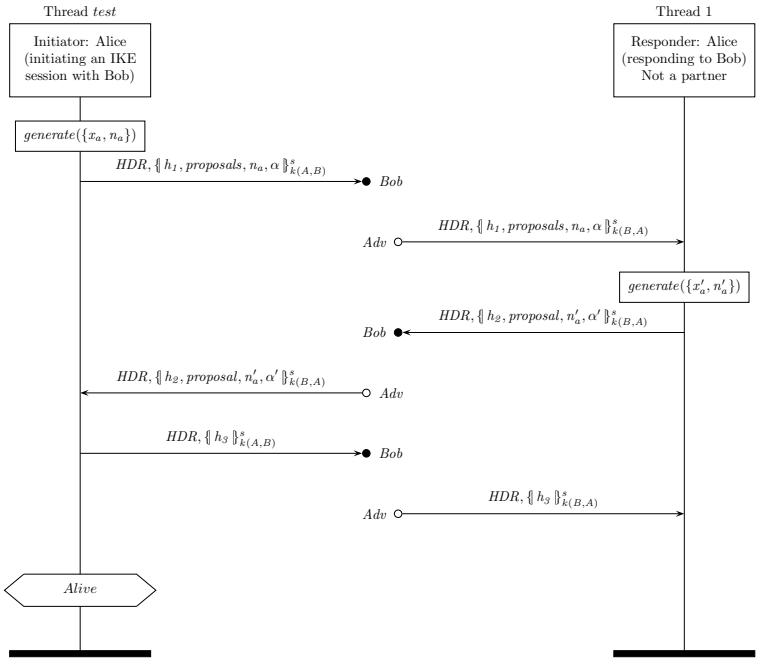

Figure 6.2: Reflection attack against the initiator of IKEv1 QM: Alice winds up thinking that she shares two keys with Bob, whereas Bob did not even execute the protocol.

adopted our models to allow key equality of the form $k(A, B) = k(B, A)$, we were able to reconfirm the attack which presents itself as a violation of the aliveness property.

6.2.2 Proposal Attacks

The following attack exploits another weakness in the definition of IKEv1 authenticators. In all adversarial environments, an attacker can influence the exchange such that Alice and Bob agree on a weak set of cryptographic algorithms for their security associations which could potentially be exploited by the adversary at a later stage. The problem exists because both authenticators only include Alice's proposal list but not Bob's choice thereof. The attack works as follows. Suppose that Alice sends a bunch of different security association proposals in order of preference where the least preferred proposal provides only marginal security. Regardless of what Bob chooses from Alice's list, the adversary modifies Bob's response such that both eventually agree on the weakest possible security association. Neither Bob nor Alice will detect the alterations and start using the newly negotiated SA, which is considerably weaker than it should be. The attack was first described by Ferguson and Schneier in [20] and works against all IKEv1 subprotocols. Note that this is an attack

against *(non-injective) agreement* [33] and, although we explicitly excluded non-injective agreement from our scope, we mention the attack for the sake of completeness.

6.2.3 Penultimate Authentication Flaws

Penultimate authentication is an authentication property introduced by Meadows in [36]. The property describes desirable behavior at the penultimate stage of the protocol in the following sense: if Bob accepted a security association as being negotiated with Alice, then Alice had also accepted the same security association (minus the keying material contributed by Bob if that has not been sent yet). Meadows finds attacks against penultimate authentication with signature authenticated as well as with public key authenticated subprotocols of IKEv1 whereas subprotocols using pre-shared key satisfy the property. She further conjectures that subprotocols which use revised public key authenticators also achieve penultimate authentication. Our results confirm these findings and additionally provide evidence to support her statement regarding revised public key authentication. We also find that penultimate authentication is violated for signature authenticated IKEv2 and by this confirm results from [37]. In our setting, penultimate authentication flaws manifest themselves as a violation of the weak agreement property.

We will now have a closer look at two penultimate authentication failures. First, we revisit the attack against digital signature authenticated IKEv1 AM, previously described by Meadows. Second, we look at a scenario where the property is violated for signature authenticated IKEv2.

6.2.3.1 Attack against IKEv1

In the following attack the adversary assumes the identity of Eve.[1] The attack proceeds as follows. Alice initiates an IKE session with Bob, using signature authenticated AM. The adversary intercepts her initial message and substitutes Alice's identity with the identity of Eve, forwarding the result to Bob. Now Bob replies with his second message which he sends to Eve. The adversary intercepts this message and forwards it to Alice. Alice believes that Bob sent this message as a reply to her initial message and successfully completes the protocol. Bob, on the other hand, thinks that it is Eve who initiated the protocol with him, and when he receives a message that is signed by Alice he will reject it. Eventually Bob

[1] Recall that in our model, the adversary explicitly compromises Eve's long-term secrets before or during protocol execution.

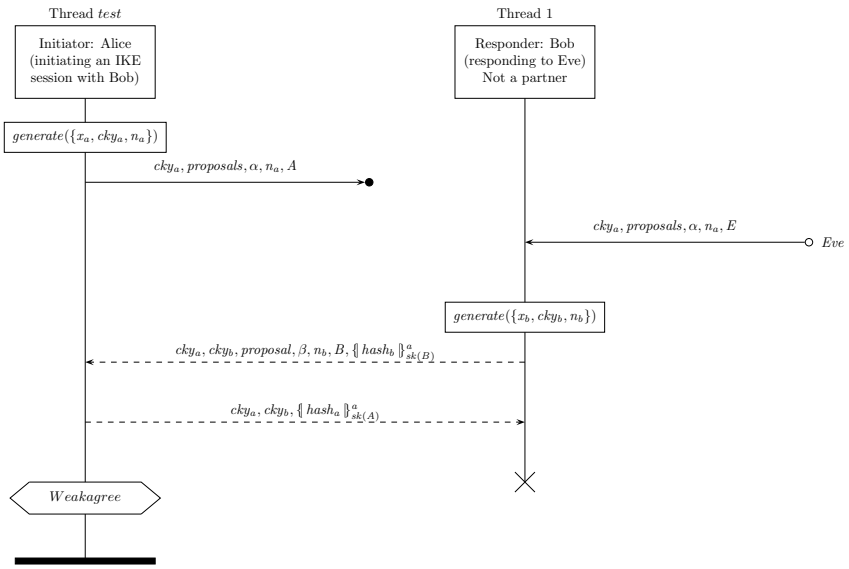

Figure 6.3: Penultimate authentication attack against signature authenticated IKEv1 AM: Alice accepts a security association as good for communication with Bob, while he eventually aborts the protocol waiting for Eve's last message.

times out as the last message which he expects to come from Eve will never be sent. The problem now is that Alice accepted the SA as good for communication with Bob while Bob aborted the protocol. We show the attack in Fig. 6.3.

6.2.3.2 Attack against IKEv2

The attack, depicted in Fig. 6.4, starts with Alice initiating an IKE session with Eve using signature authenticated IKEv2. The adversary intercepts her message and forwards it to Bob. Bob generates his reply (for Alice) according to the protocol. The adversary intercepts Bob's reply, masquerades as Eve and sends it to Alice. Alice authenticates her exchange by computing $auth_a$. She subsequently derives the keying material for this session and encrypts $auth_a$ together with her identity and other payloads, sending the result to Eve. The adversary intercepts and redirects this message to Bob. Meanwhile, Bob derives his share of the keying material. When receiving Alice's last message, he decrypts it and verifies her signature. The verification succeeds and Bob accepts the security association as shared with Alice. Next, he authenticates his part of the exchange using the cryptographic algorithms of that SA. When Alice receives the last message from Bob, she rejects it because she is expecting the message to come from Eve. Eventually, Alice times out because she

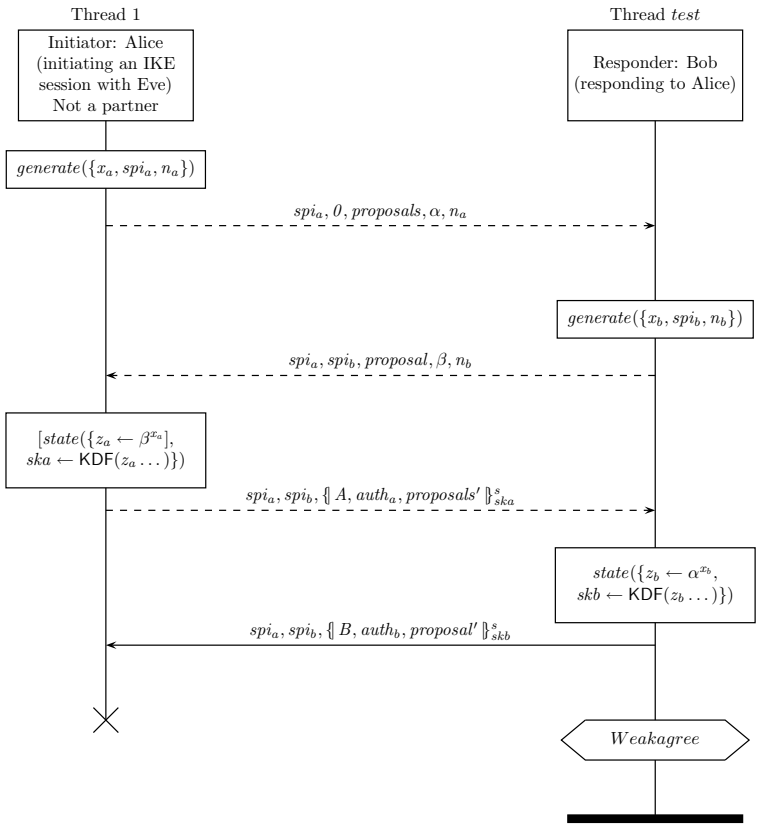

Figure 6.4: Penultimate authentication attack against signature authenticated IKEv2: Bob accepts a security association as good for communication with Alice, while she eventually aborts the protocol waiting for Eve's last message.

will never receive the expected message from Eve.

6.2.3.3 Discussion

A first observation is that both attacks, although exhibiting similar patterns, differ in the role which is attacked. In the former, the vulnerability only exists for the initiator while in the latter, the responder is being attacked. We briefly sketch why the respective other roles are protected from being attacked. For signature authenticated IKEv1 AM, we assume Alice initiating the protocol with Eve while it is Bob who responds to Alice. When Alice then receives the first message from Bob, she will fail to verify his signature. Because Eve will never reply, Alice eventually times out. As a consequence, Bob times out too. Similarly, Bob will time out while waiting for an answer from Eve when executing signature

authenticated IKEv2.

Second, we note that IKEv2 is only vulnerable if Alice does not include the identity of the responder in her second message. The inclusion thereof is declared optional by the specification. By including the identity, Alice expresses her belief of who she is communicating with. When Bob receives her message, he will realize that Alice believes to communicate with Eve and abort the protocol.

6.2.4 Consequences of Penultimate Authentication Flaws

When analyzing the subprotocols in the environments $Adv_{BR}, Adv_{eCK-2}, Adv_{CKw}$ and Adv_{CK} (see Table 6.2), we found numerous attacks against session key secrecy, all of which could be thought of as variants of the following. The adversary acts as a man in the middle and impersonates Eve towards one of the participants. Through clever manipulation of the exchange the adversary subsequently establishes a scenario in which one of the participants believes to share the security association with its honest peer, while the peer believes to share the security association with Eve.

This scenario looks very similar to the penultimate authentication problems discussed before. In fact, the attacks can be viewed as an immediate consequence of failing to achieve this authentication property. We discover that all signature authenticated variants of IKEv1 are susceptible to a Session Key Reveal and Sessionstate Reveal attacks in the above scenario. Public key authenticated exchanges are only vulnerable in the original exchange and if nonces and identities are encrypted separately; the revised public key exchange provides penultimate authentication and does not suffer from this type of attack. Below, we describe attacks against the responder and the initiator of public key authenticated MM. The attacks are depicted in Fig. 6.5 and Fig. 6.6, respectively.

Our results, at first, appear to be conflicting with statements made by Canetti and Krawczyk in [13, 14] where the authors performed an extensive analysis of IKE in Adv_{CK}. A closer look, however, quickly reveals that the conflicts only arise because our analysis is based on different assumptions. Namely, Canetti and Krawczyk proved the signature authenticated variant secure under the assumption that all incomplete sessions are considered as matching sessions and thus can not be compromised by the adversary (cf. first condition of [14, Definition 2]). The variant relying on public key encryption is proved secure under the assumption that the session key is not kept in local state. Both assumptions do not apply in our setting.

6.2.4.1 Attack Against the Responder

This attack requires an active adversary that can reveal the local state of an agent. The adversary can compute SK on the basis of the revealed information (based on the algebraic properties of modular exponentiation). The attack proceeds as follows. Alice initiates a protocol session with (corrupted) Eve. The adversary intercepts Alice's first message and relays it to the target of the attack, Bob. Bob, who believes that Alice initiated a protocol session with him, assumes the responder role and generates his response for Alice. The adversary intercepts Bob's reply, fakes its origin as to have come from Eve and sends it to Alice. Alice computes her Diffie-Hellman token and encrypts both her name and a fresh nonce with Eve's public key. Next, she sends her second message to Eve. The adversary intercepts this message, decrypts Alice's nonce, re-encrypts it for Bob and replaces $\{\!|\, A \,|\!\}^a_{pk(E)}$ with $\{\!|\, A \,|\!\}^a_{pk(B)}$. After receiving the second message, Bob computes his DH token and encrypts his name and his nonce for Alice. The adversary intercepts Bob's next message and fakes its content such that Alice will accept it as to have originated from Eve. We note that now both agents posses enough information to compute the shared secret, which we assume they will do. To authenticate the exchange, Alice generates her authenticator $hash_a$ and encrypts it with the shared key. We stress that up until now, the attack progressed exactly as the penultimate authentication attack on the same protocol. The adversary forwards the encrypted authenticator to Bob who decrypts and verifies it. Next, Bob generates his authenticator and encrypts it with the shared key. Meanwhile, the adversary reveals Alice's session state (using Sessionstate Reveal). The adversary is now able to construct SK herself. She intercepts Bob's final message and substitutes $\{\!|\, hash_b \,|\!\}^s_{skb}$ with her own authenticator $\{\!|\, hash_e \,|\!\}^s_{ska}$, forwarding the result to Alice.

Discussion Contrary to the attacks presented in [36], the adversary, in the above scenario, is able to let both peers finish their protocol executions. The difference is that Alice does not time out while waiting for Eve's message because the adversary is capable of constructing it with the knowledge of the session key. The ability of the adversary to derive the session key, however, bases on our assumption that all relevant information is kept in the local state of Alice (and thus is revealed by Sessionstate Reveal).

We also note that if we perform public key encryption such that we get $\{\!|\, A, n_a \,|\!\}^a_{pk(B)}$ and $\{\!|\, B, n_b \,|\!\}^a_{pk(A)}$ for initiator and responder, respectively, then there is no attack on the protocol.

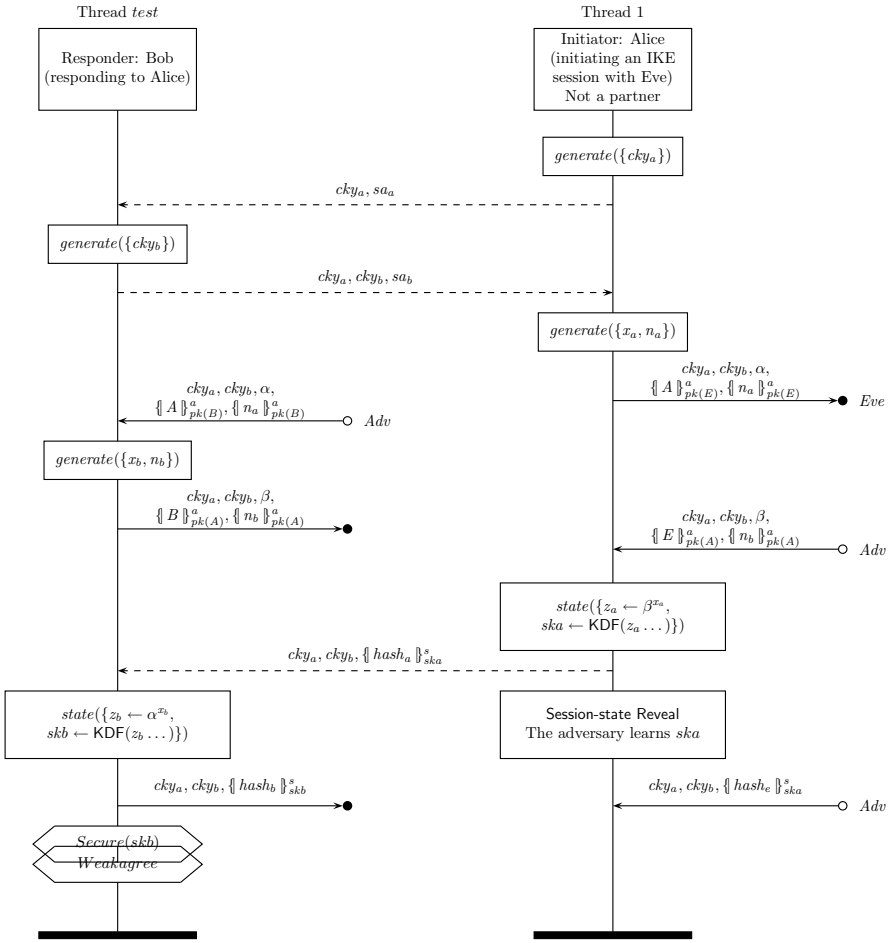

Figure 6.5: Sessionstate Reveal attack against the responder of public key authenticated IKEv1 MM: Bob believes to share the key *skb* with Alice, whereas Alice believes to be sharing *ska* with Eve. The adversary compromises Bob's secrecy claim by revealing Alice's state.

6.2.4.2 Attack Against the Initiator

Attacking the initiator of public key authenticated IKEv1 MM is analogous to the attack presented above. The only difference is the timing of when the responder's session state is compromised. Above, the adversary defers the compromise of Alice (e. g., the initiator) until she sent her last message. In this attack the adversary *must* compromise Bob (e. g., the responder) before he receives message 5. Otherwise, Bob would recognize that instead of running the protocol with Eve, he is running it with Alice; the adversary cannot construct Eve's authenticator without knowing skb.

Discussion The subtle difference is in the moment of the compromise. Were we allowing the adversary to corrupt Bob only after he had received the last message, Bob would not be able to complete the protocol since the last message to be received by Bob could not be constructed by the adversary. However, if the implementation allows Bob to compute the session key at the earliest possible stage, then this attack can be considered a weakness of the protocol. It can even be considered more serious than the attack against the responder presented above because the adversary, to attack a given target, has the liberty to relay traffic to any node in the network, especially to those which have weak security in place.

6.3 Newly Discovered Attacks

6.3.1 Reflection Attacks

6.3.1.1 A Novel Reflection Attack Against IKEv1 MM

Our analysis reveals that IKEv1 MM is not only susceptible to reflection attacks if authenticated with digital signatures or pre-shared keys (as reported by Ferguson and Schneier, cf. Section 6.2.1), but also if authentication is based on public key encryption. To the best of our knowledge, we are the first ones to report this kind of vulnerability. The attack (cf. Fig. 6.7) proceeds as follows. Alice, in the initiator role, initiates the protocol by sending her fresh session cookie cky_a together with her SA proposal to the network. The responder claims Alice's identity and responds with his cookie $cky_b = cky_a$ and his SA selection. Next, Alice proceeds by sending α together with a fresh nonce N_a. The responder chooses his Diffie-Hellman token as $\beta = \alpha$ and replies with (β, N_b) where N_b is a fresh nonce. Subsequently, Alice computes $hash_a$ according to Definition 11 and authenticates her side

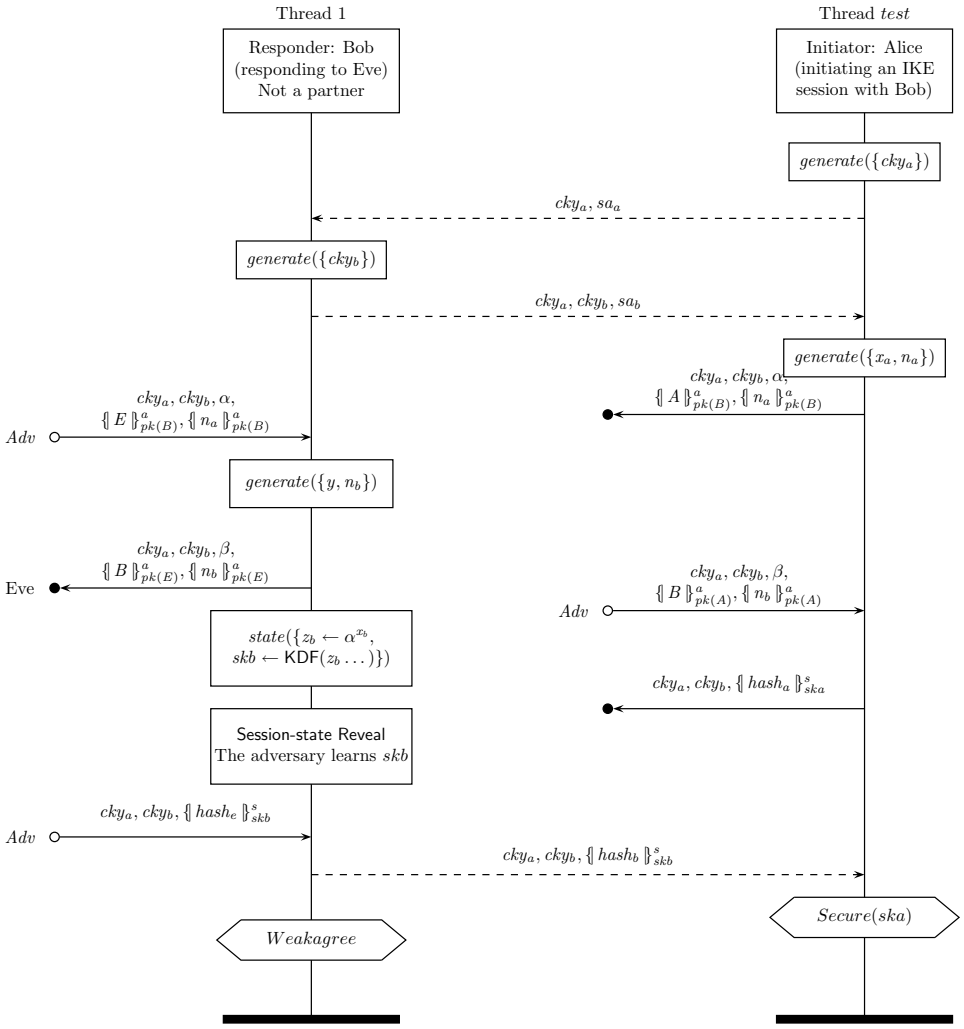

Figure 6.6: Sessionstate Reveal attack against the initiator of public key authenticated IKEv1 MM: the adversary relays traffic to an insufficiently protected host to subsequently reveal its session state.

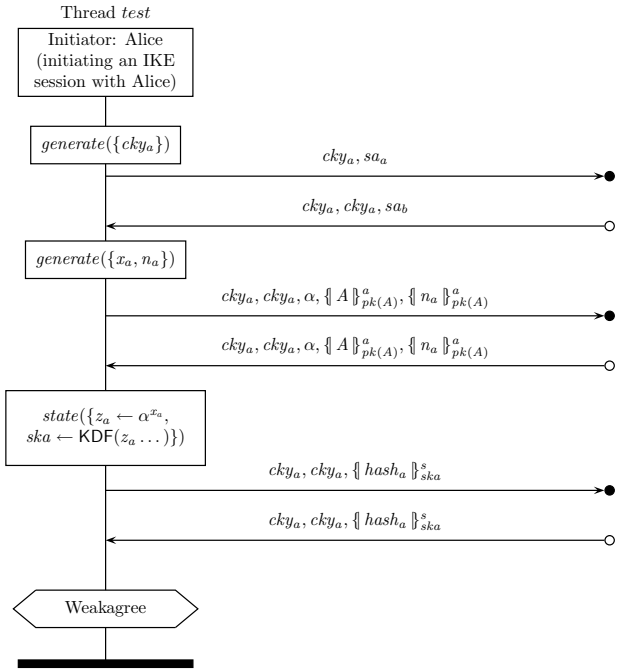

Figure 6.7: Reflection attack against public key authenticated IKEv1 MM: a fraudulent responder claims Alice's identity and passes the authentication phase by reflecting her own authenticator.

of the exchange. The responder is now able to complete (and authenticate his share of) the exchange by reflecting Alice's last message because he computed the same key as her ($skb = ska$) and thus

$$hash_b = \mathsf{prf}\,(skb, \beta, \alpha, cky_b, cky_a, proposals, A) \equiv$$
$$\mathsf{prf}\,(ska, \alpha, \beta, cky_a, cky_b, proposals, A) = hash_a.$$

The attack presents a violation of weak agreement. Again, such an attack may be relevant in a setting where only IP addresses are authenticated.

6.3.1.2 A (Familiar) Reflection Attack Against IKEv2's Phase 2 Exchange

We discover that IKEv2's phase 2 exchange suffers from the same reflection attack as its predecessor. The attack (cf. Fig. 6.8) proceeds analogously to Section 6.2.1.2. Again, Alice eventually believes that she shares *two* keys with Bob, whereas in fact she does not share any keys at all. The attack is a violation of weak agreement and as such presents an

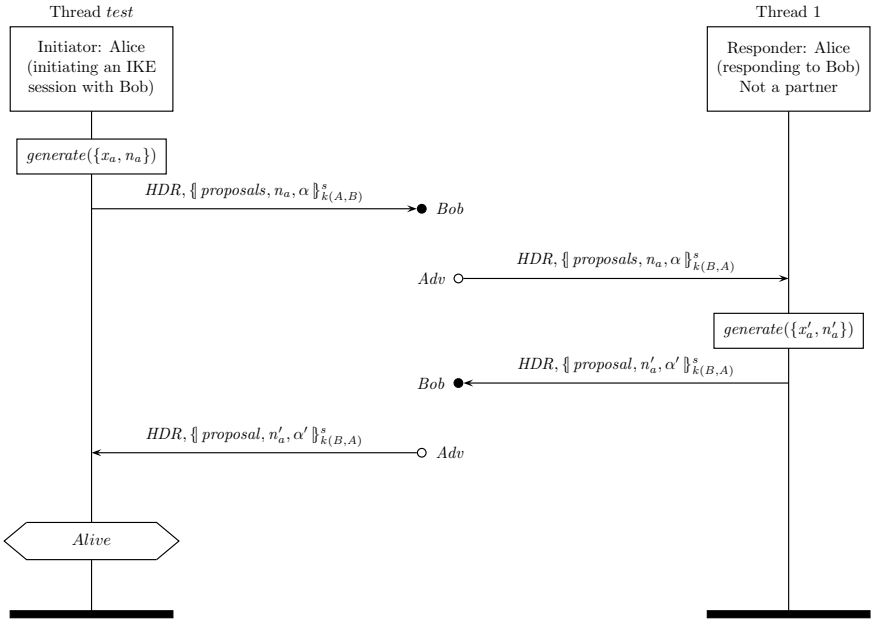

Figure 6.8: Replay attack against the initiator of a IKEv2 phase 2 exchange: Alice winds up thinking that she shares two keys with Bob, whereas Bob did not even execute the protocol.

authentication flaw. Additionally, Alice has been denied a service, namely the setup of an IPSec security association.

6.3.2 Consequences of Penultimate Authentication Flaws

Below, we show an attack on a responder session of IKEv2 authenticated with digital signatures. This attack requires an active adversary that can reveal the local state of an agent, and assumes that agents keep their inputs to the key derivation function in their local state. We abstain from presenting a similar attack on the initiator session but note that it proceeds analogous to the one against IKEv1 (cf. Fig. 6.6), i. e., the adversary reveals Bob's state *just before* he receives Alice's encrypted authenticator.

The attack is launched as follows. Alice initiates the protocol with Eve. The adversary, acting as man in the middle, relays this message to another agent, Bob. Bob's reply is then intercepted and forwarded to Alice, whereby the adversary impersonates Eve towards Alice. Alice subsequently authenticates her share of the exchange by sending an encryption of her authenticator together with her identity (among other data) to Bob. Bob decrypts the message and verifies Alice's signature. Meanwhile, the adversary compromises the session

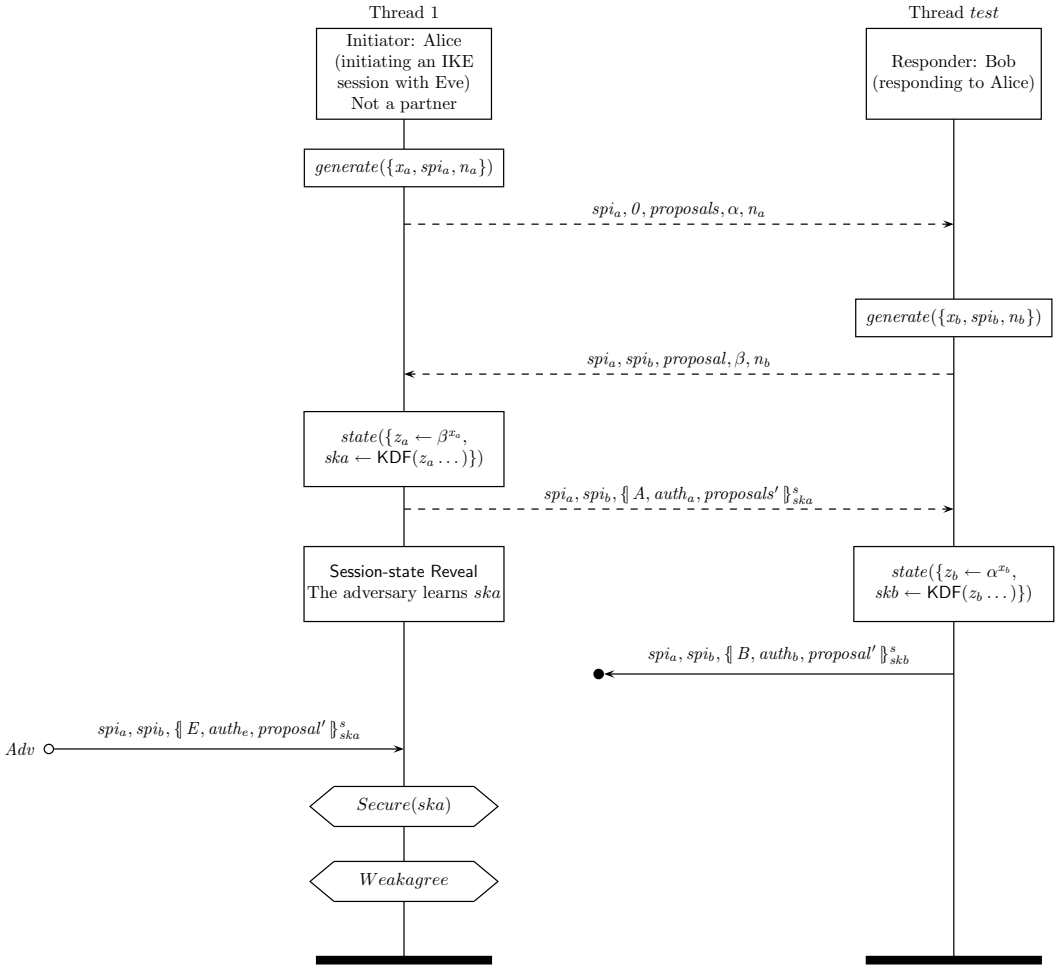

Figure 6.9: Sessionstate Reveal attack against the responder of a signature authenticated IKEv2 session

state of Alice, thereby eventually revealing the session key. Finally, the adversary intercepts Bobs last message, decrypts it and substitutes $auth_b$ with the authenticator of Eve, $auth_e$. The adversary re-encrypts the result, forwarding it to Alice.

6.3.3 Other Attacks Relying on State Corruption

In the following, we present two attacks which require an adversary capable of Sessionstate Reveal or Random Reveal. The first attack violates session key secrecy and weak agreement of public key authenticated IKEv1 AM. The second attack violates all security properties of MAC authenticated IKEv2, signature authenticated IKEv2, and of all IKEv1 subprotocols

that either rely on authentication based on digital signatures or public key encryption.

Session-state reveal The following attack (cf. Fig. 6.10) against the responder of public key authenticated IKEv1 AM requires an adversary which is capable of Sessionstate Reveal. Moreover, we assume that agents generate and keep their nonces in local memory during protocol execution. The attack proceeds as follows. To attack Alice, who is acting as the responder, the adversary initiates an aggressive mode exchange with her, thereby impersonating Bob ($\gamma = g^{x_a dv}$). Alice replies according to the protocol and sends her public Diffie-Hellman value α together with $\{\!|\, A, n_a \,|\!\}^a_{pk(B)}$. The adversary intercepts this message and initiates another session with Bob. In doing so, the adversary injects $\{\!|\, A, n_a \,|\!\}^a_{pk(B)}$. Note that Bob also acts as a responder and therefore will never be a partner of Alice. Bob decrypts Alice's nonce and identity and stores both values in his local state. Subsequently, the adversary corrupts the session-state of Bob using Sessionstate Reveal, thereby learning N_a. Next, the adversary computes

$$hash_{adv} = \mathsf{prf}(n_{adv}, n_a, \alpha^{x_{adv}}, \gamma, \alpha, cky_{adv}, cky_a, proposals, B)$$

and sends it to Alice who finishes the protocol. The result is that Alice winds up believing that she shares a session key with Bob whereas she does share it with the adversary. The attack violates session key secrecy as well as weak agreement. Note that this attack even works if the nonce is encrypted together with the identity.

Random reveal The weakest link of almost all IKE protocols is key derivation. The problem there is that as soon as the private exponent of one of the participants is leaked, the protocol is broken. For example, suppose that Alice and Bob establish an IKE SA via signature authenticated IKEv2. If the adversary is able to learn the secret exponent y of Bob (for example by using Random Reveal), then he or she can compute Z and, because all other inputs to the key derivation function were exchanged unprotected, also derive the key (see Fig. 6.11). Of course, the risk of such an exposure is relatively small but nevertheless existent[2].

All IKE subprotocols, except for pre-shared key authenticated IKEv1, exhibit vulnerabilities in Adv_{eCK-1}, the only adversary model which includes RNR. This is as expected;

[2]Think of flawed random number generators, etc.

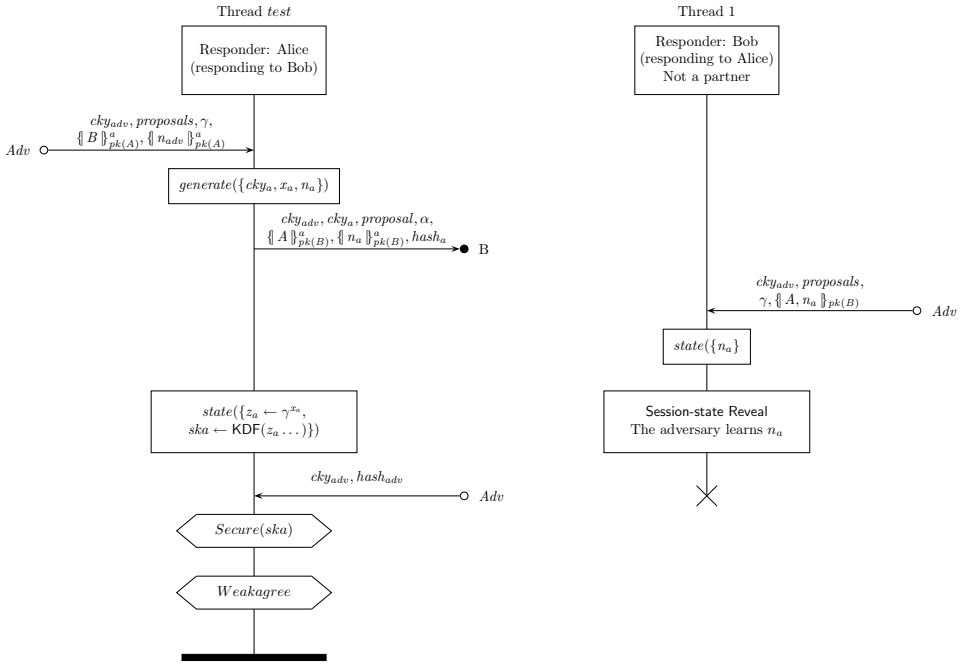

Figure 6.10: Sessionstate Reveal attack against the responder of public key authenticated IKEv1 AM: Alice ends up believing that she shares a session key with Bob whereas Bob did not finish the protocol.

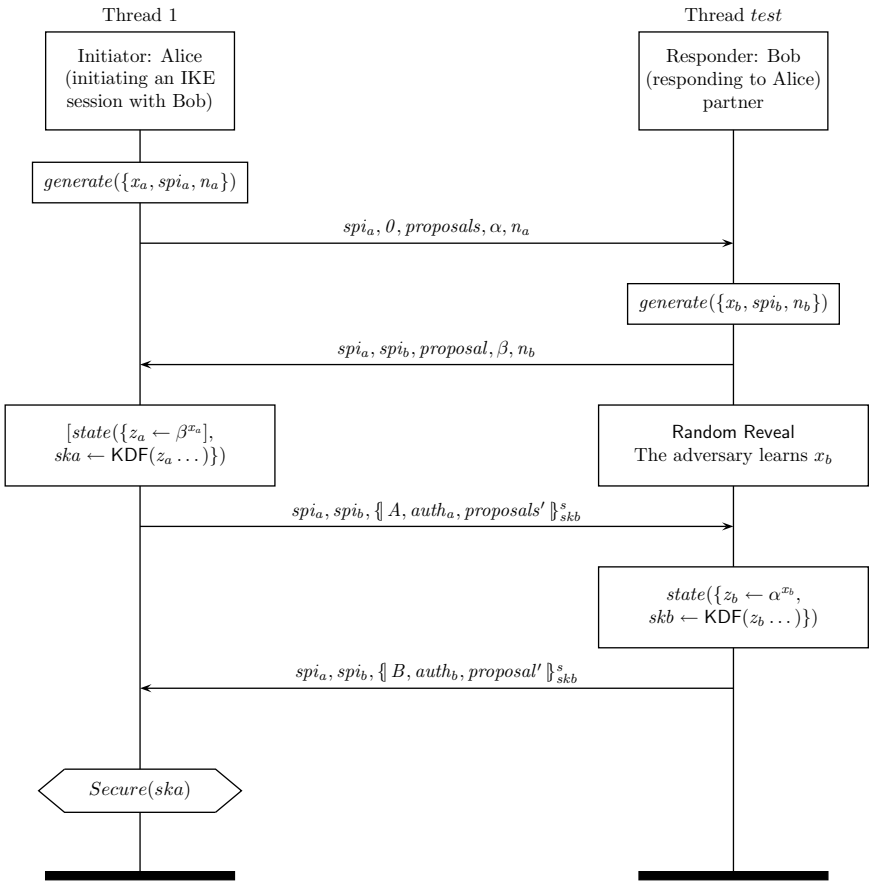

Figure 6.11: Random Reveal attack against the initiator of signature authenticated IKEv2

an adversary endowed with the Random Reveal capability is the only one allowed to compromise the test session and its matching session. The motivation for this capability stems from a recent extension of the Canetti-Krawczyk security model where it is shown to be possible to construct security protocols which achieve their goals even in an environment where parts of the session state of partners may be exposed [31]. To minimize this risk of exposure, LaMacchia et al. suggest that the public Diffie-Hellman token should be computed as $g^{H_1(y,sk(B))}$, where H_1 is a hash function, instead of solely relying on g^y. Note that with the knowledge of the private exponent of one of the agents, the adversary can simply block traffic from and to that agent and complete the exchange with the test session, thereby also jeopardizing the authentication goal of the protocol.

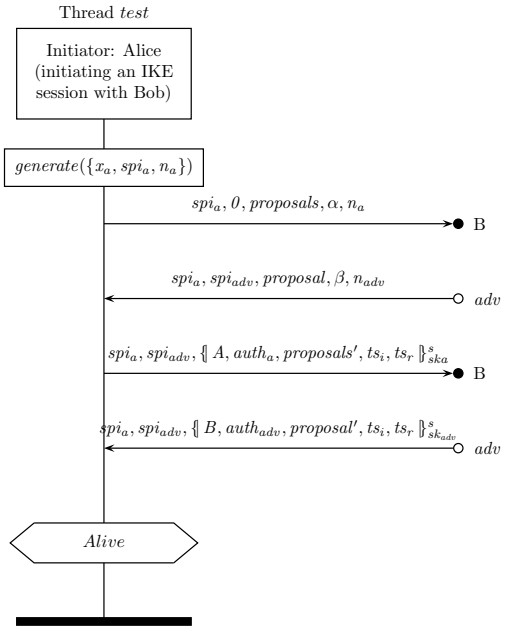

Figure 6.12: Key Compromise Impersonation attack against the initiator of MAC authenticated IKEv2

6.3.4 Key Compromise Impersonation

IKEv1, when authenticated with pre-shared keys, and IKEv2, when using MAC authentication are vulnerable to Key Compromise Impersonation attacks. If the adversary is in possession of a long-term secrets of Alice, he or she can impersonate other agents to her. An example is depicted in Fig. 6.12. The attack requires an adversary capable of $\mathsf{LongtermKeyReveal}_{\mathsf{actor}}$. The attack violates aliveness and proceeds as follows. Before Alice initiates MAC authenticated IKEv2 with Bob, the adversary compromises her long-term secret $k(A, B)$. With the knowledge of $k(A, B)$ the adversary is able to produce all protocol messages of Bob.

6.4 Other Known Attacks

Apart from session key secrecy, entity authentication, and integrity, there exist other security properties which IKE has been analyzed for in the literature. Mainly, these properties fall into two categories: identity protection and resource exhaustion. Properties in the

former category include anonymity, i. e., an adversary cannot reveal the identity of the participants during or after protocol execution, and deniability, i. e., a protocol participant A cannot convince a third party that he or she has executed the protocol with B. The second category includes all forms of Denial-of-Service attacks, such as state or computational exhaustion.

Our analysis of IKE did not include such properties. Resource exhaustion was not considered because it is currently out of scope of the Scyther tool. Identity protection, on the other hand, was out of scope because our model does not yet support the formalization of such properties. Therefore we could not reproduce the attacks discovered by Perlman and Kaufman [39]. We mention them briefly for the sake of completeness. In signature authenticated IKEv1 MM, Alice's identity is exposed to an active attacker impersonating Bob to Alice. The adversary will negotiate a session key with Alice and discover her identity in message 5. However, the adversary is not able to complete the protocol since he or she is unable to generate Bob's signature in the last message. Perlman and Kaufman suggested to move the payload of message 6 to message 4. This would expose the responder's identity to an active attacker which, as they argue, is less critical than exposing the identity of the initiator. We modeled their suggestion to analyze whether it influences secrecy or authentication.

In pre-shared key authenticated IKEv1 MM, both identities are revealed, even to a passive adversary. Bob cannot decrypt message 5, which reveals Alice identity, without knowing who he is talking to, unless he derives this information from messages 1-4. Perlman and Kaufman suggest to modify the exchange such that messages 5 and 6 are encrypted with a temporary key which does not rely on the pre-shared secret but only on the agreed-upon DH secret. Again, we modeled their suggestion to see whether it has implications on secrecy or authentication.

6.5 Security Hierarchy

The establishment of a protocol-security hierarchy among IKE protocols enables the selection of certain protocol variants based on implementation requirements and worst-case expectations for adversaries in the application domain. In Fig. 6.14, we show the protocol-security hierarchy for the session key secrecy property of IKEv1 and IKEv2 with respect to all adversary models from Table 6.1. In Fig. 6.15, we show a protocol-security hierarchy for

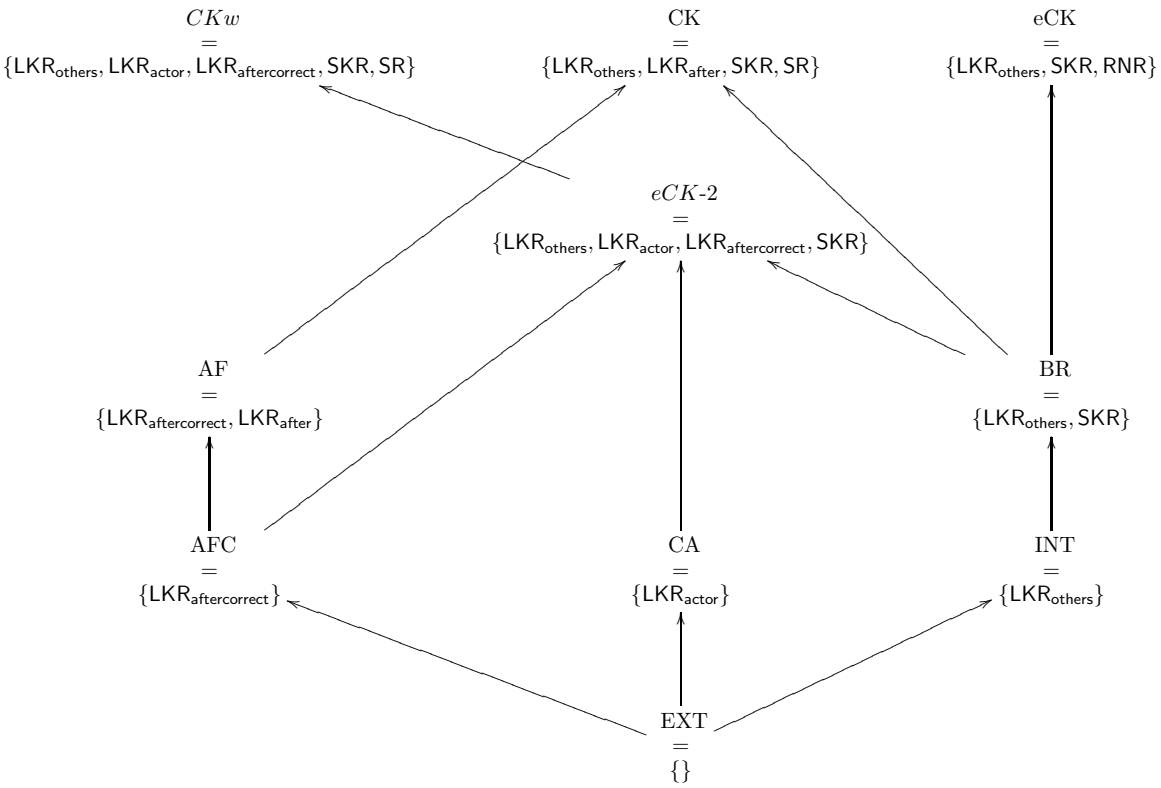

Figure 6.13: A hierarchy of adversary-compromise models

authentication properties. Hierarchies for security protocols were proposed in [6] and are interpreted as follows. Each node in Fig. 6.14 and 6.15 corresponds to a set of protocols and is annotated with a set of adversary models. For each adversary model in the set, it is required that no attacks are found in this or any weaker model, and also that attacks are found in all stronger models. The model hierarchy can be deduced from Table 6.1 and is given in Fig. 6.13.

The security hierarchies in Fig. 6.14 and 6.15 can be generated automatically using Scyther. We refer to [6] for details about the implementation. Below, we discuss some of the relations among the protocols that were not covered by attack the descriptions above, but are nevertheless worth mentioning.

IKEv1-PK-A1 vs. IKEv1-PK-A2 In the original version of public key authenticated IKEv1 AM, the initiator sends his identity and nonce N_i as $(\{\!|\, I \,|\!\}^a_{pk(R)}, \{\!|\, N_i \,|\!\}^a_{pk(R)})$. This is

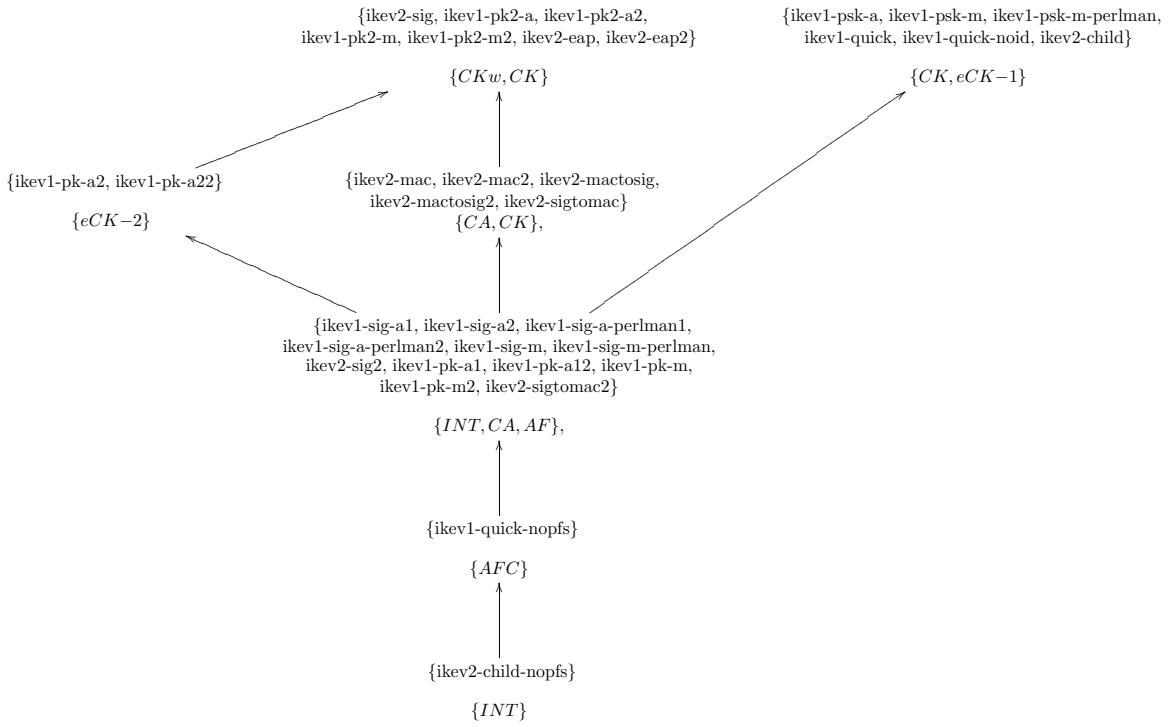

Figure 6.14: Security hierarchy among IKEv1 and IKEv2 protocol variants with respect to session key secrecy

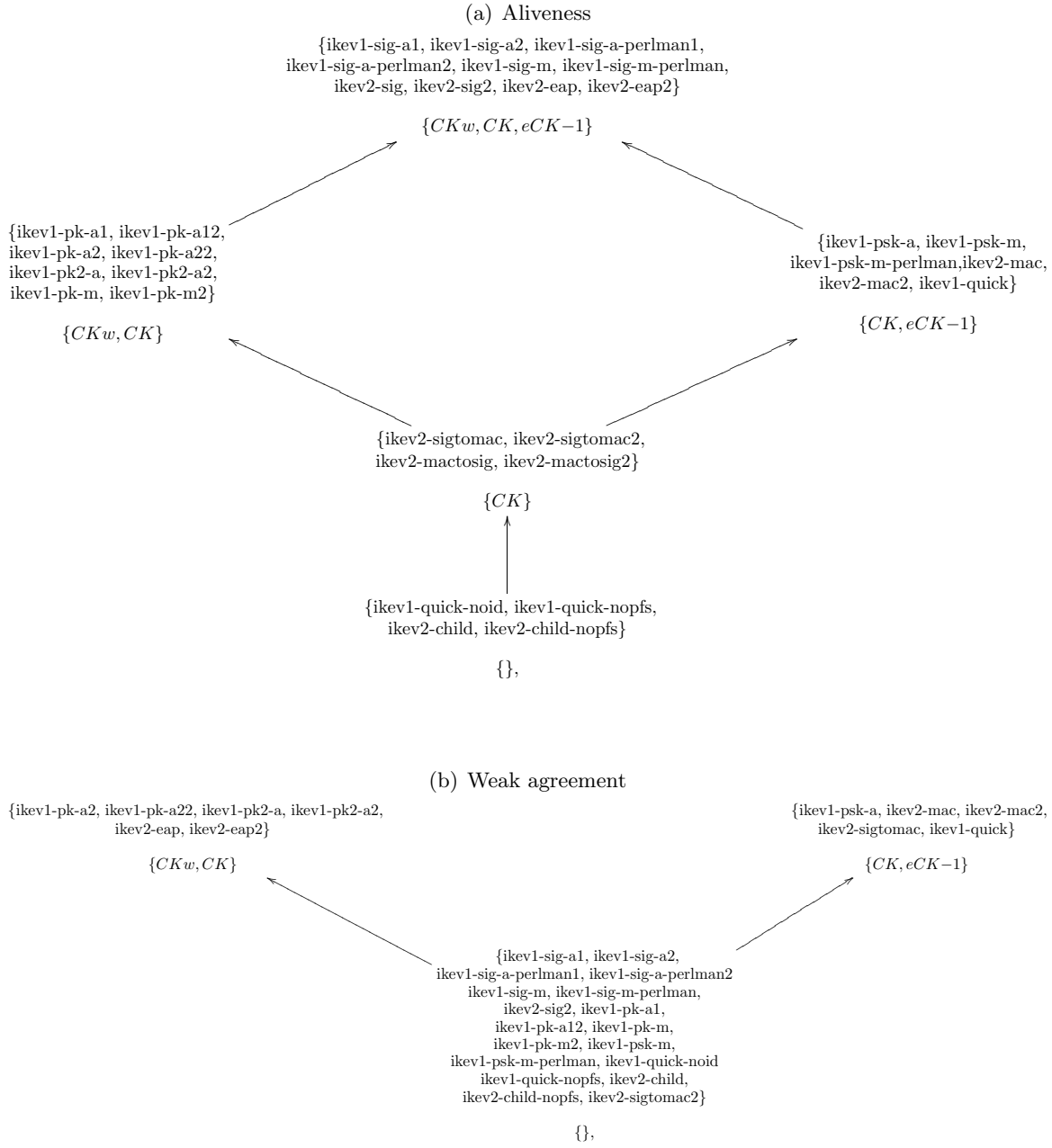

Figure 6.15: Security hierarchy among IKEv1 and IKEv2 protocol variants with respect to aliveness and weak agreement

reflected in the models **ikev1-pk-a1** and **ikev1-pk-a12**. Models **ikev1-pk-a2** and **ikev1-pk-a22** reflect a modified version of the exchange where identity and nonce are transmitted as $\{\!|\, X, n_x \,|\!\}^a_{pk(Y)}$ for sender X and receiver Y, and thereby follow principle 3 of [2]. Scyther finds attacks against session key secrecy on the original variants for all models that are strictly stronger than Adv_{INT}, and attacks against weak agreement for all models. This is due to the failure of the protocol to achieve penultimate authentication as discussed in Section 6.2.3.

IKEv1-PK vs. IKEv1-PK2 The revised version of public key authenticated IKEv1 was originally proposed to reduce the amount of (expensive) public key operations. It turns out that the revision also had an impact on the security of the protocol. Scyther only finds attacks against secrecy on the revised version for Adv_{eCK-1} whereas the original exchange is vulnerable in all models stronger than Adv_{INT}. The reason for this difference is that the adversary, in order to launch an attack, needs to derive the key $\mathsf{prf}(N_x, CKY_x)$ which is impossible if the long-term secrets of the participants are uncompromised.

IKEv2-SIG vs. IKEv2-SIG2 We already discussed in Section 6.2.3 that the initiator may optionally include the responder's identity in the third message of signature signature authenticated IKEv2. Scyther finds attacks on signature authenticated IKEv2 for all models stronger than Adv_{INT} if the optional field – which expresses the initiator's belief in who her peer is – is not included in message 3. This payload thus is security relevant and we suggest that the specification is adapted to include the responder's identity in message 3. This also is along the lines with commonly accepted design principles for security protocols (cf. [2, Principle 3]).

Chapter 7

More on Related Work

We previously presented the outcome of performing a fully automated security analysis of IKE in various adversarial settings. Where it was appropriate, we discussed our results in the context of related work. The purpose of this chapter is to relate our findings to other related topics.

Unknown key share In [11], Blake and Menezes describe what they call *unknown key-share (UKS) attacks*. In a UKS attack on a key agreement protocols, Bob ends up believing that he shares a key with Alice, and although this is in fact the case, Alice mistakenly believes the key is instead shared with Eve. This scenario appears to be very similar to what we have discovered in Sections 6.2.4 and 6.3.2. A closer look reveals that the scenario described by Blake and Menezes requires both participants to successfully complete the protocol *without the adversary dynamically compromising protocol participants*. In our situation, Alice can only complete *because* the adversary compromised her session key prior to forging the last message from Eve. Kaliski [24], on the other hand, describes a UKS attack as a scenario where the adversary coerces honest parties into establishing a secret key where at least one of the parties does not know that the secret key is shared with the other. If regarded from Kaliski's perspective, then our attacks fall into the category of UKS since the (although more powerful) adversary *does* coerce Alice and Bob into establishing a secret key where one of them does not know that the secret key is shared with the other.

Protocol Composition Logic (PCL) In [40] Roy et al. scrutinized IKEv2 authenticated with digital signatures using PCL. Their protocol model is comparable to **ikev2-sig2**

(cf. Chapter 5) in terms of exchanged messages. In Theorem 8, the authors state that IKEv2 satisfies strong entity authentication as well as message integrity if Diffie-Hellman exponentials are never reused. This statement contrasts with the attack reported in the previous chapter, where digital signature authenticated IKEv2 fails to achieve Meadows' penultimate authentication property. Finding the source of this difference in analysis is outside of the scope of the thesis due to time limitations.

AVISPA As we have seen, penultimate authentication can be achieved for signature authenticated IKEv2 if the initiator includes the recipient's identity in the third message of the exchange. Thus the fix of Mödersheim and Drielsma [37] which extends the protocol by two messages to add key confirmation is not necessary. This not only reduces the risk of cryptanalysis but also is much more efficient with respect to the number of exchanged messages.

Mödersheim and Drielsma also analyzed the MAC authenticated variant of IKEv2 and found no security flaw. This "contradicts" our findings where we show that MAC authenticated IKEv2 is vulnerable to Key Compromise Impersonation. However, we must note that their underlying security model did not allow for the expression of adaptive key compromise.

Finally, we note that Mödersheim and Drielsma (as well as Meadows) did not consider penultimate authentication as a serious threat. Such an statement is perfectly reasonable when implied by a security analysis in the (still standard) Dolev-Yao security model. However, we saw that as soon as we give the adversary more power, e. g., the ability to reveal the session state of *non-matching* sessions, such an initially insignificant attack may suddenly uncover serious deficiencies in protocol design.

Chapter 8

Conclusion

In this thesis, we have conducted the first formal analysis of IKEv1 and IKEv2 in a symbolic framework which supports the notion of compromising adversaries. Our study has shown that real-world security protocols, which often are complex and offer a lot of flexibility, can be formally analyzed with the help of an automatic tool, such as Scyther. Our analysis revealed several new weaknesses of IKE, the most important ones being:

- Public key authenticated IKEv1 is vulnerable to the same reflection attack previously discovered by Ferguson and Schneier on signature authenticated IKEv1 and IKEv1 authenticated with pre-shared keys.
- Signature authenticated IKEv2 fails to achieve Meadows' penultimate authentication property. This finding is surprising because IKEv2 was aimed at resolving the many security issues of its predecessor. It is even more surprising considering the fact that the attack could easily be avoided by including a field which is marked optional in the specification.
- The IKEv2 subprotocol used to establish subsequent IPSec SAs from an existing IKE SA is found vulnerable to a reflection attack which was already known to exist for its predecessor and was also discovered by Meadows.
- The stronger adversary models resulted in a number of attacks against session key secrecy which we consider as a consequence of lack of penultimate authentication. These attacks particularly demonstrate that vulnerabilities previously considered harmless, suddenly become a serious threat when the adversary is given more power; protocols

that fail to achieve penultimate authentication are vulnerable to adversaries which are allowed to compromise the state (or the session-keys for that matter) of agents in non-matching sessions.

We also used Scyther to automatically establish a security hierarchy among the various subprotocols of IKEv1 and IKEv2, based on the verification data we gained from the analysis. This not only visualizes the differences between IKEv2 and its predecessor but also allow us to select protocol variants according to the (adversarial) environment we want to use them in.

A lot of effort went into the design and development of the formal models of the Internet Key Exchange protocol. We attempted to cover the full spectrum of the specification, which, considering the enormous amount of flexibility such a specification has to offer and the limitations of our model, was impossible. Nevertheless, we believe to have found a reasonable degree of abstraction which is expressed through the quality of attacks we have found. Moreover, we are convinced that our models are suitable as a basis for ongoing research, also within other frameworks.

In the future, we hope to refine our models such that we can help implementors making the right choice when it comes to the selection of security components, such as hardware security modules (HSM). A first step would be to adapt the models such they clearly indicate the type of data which is kept in local memory, and the data that is computed externally, e. g., in a HSM.

Also, to increase the efficiency of the analysis, we would like to optimize the Scyther tool for the operation in a high-performance environment such as the Brutus cluster. Especially the verification of security properties in the presence of many adversary models could be improved by allocating a process per claim and model.

An optimized verification tool would also have the benefit that we could perform a similar analysis with respect to all 96 possible adversary-compromise models. This would allow us to get a more fine-grained perspective on the security of IKE.

This thesis has focused on finding attacks on the Internet Key Exchange protocol. While the documentation of attacks certainly helps in improving the security of a protocol, it remains an unsatisfactory task.[1] Our ultimate goal, therefore, is to come up with a system that automatically generates proofs of security properties in a given symbolic model.

[1] It probably provides as little satisfaction as debugging a piece of software.

Appendix A

Additional Terminology

A.1 Perfect Forward Secrecy and Key Compromise Impersonation

In an environment where we assume that any key can be compromised, be it by break-in, bribery, theft, cryptanalysis, or simply loss, an important goal in the design of a security protocol (especially key exchange) is to limit the harm caused by the exposure of cryptographic keys. Contrary to short-term keys, where the damage of a compromised key can be confined relatively well, long-term keys need special treatment. In the following we introduce two attributes which describe the behavior of protocols in an environment of key compromise and are well-known in the literature.

Perfect Forward Secrecy Consider a scenario where all session keys received by a Alice are encrypted under Alice's public key, then an attacker that breaks her private key also learns all past, and even future, session keys that Alice shares with her peers.

While key revocation mechanisms can mitigate the risk of future attacks, there is no such mechanism to protect past exchanges *after* the compromise has been detected. Thus, security protocols must account for the protection of short-term keys at the time of their exchange.

A security protocol that protects short-term keys from compromise even in case of the exposure of long-term keys, is said to provide *Perfect Forward Secrecy*. This term was first coined by Günther [21] and Diffie, van Oorshot, and Wiener [19].

The most well-known mechanism that provides perfect forward secrecy is the Diffie-Hellman algorithm for key exchange where Alice's private key is only used to sign the exchange. In that case, a much better level of security is achieved as an adversary compromising Alice's private key will learn nothing about past communications.

Therefore, PFS is, in general, a very desirable property of a key exchange protocol. However, as there is currently no other solution to provide this property except for the Diffie-Hellman exchange, there is a computational cost to achieve it and hence it is worthwhile asking if PFS is necessary in all cases and all scenarios.

Key Compromise Impersonation Contrary to PFS, which is a security property, the term Key Compromise Impersonation refers to an authentication problem in conjunction with the loss of long-term keys. If an attacker obtains the long-term private key of Alice he or she obviously can impersonate Alice in any future conversation with any other agent. A more subtle attack, however, exists if the attacker impersonates *another agent towards Alice* and is able to establish a valid session with her. The consequences of this kind of attack can be far more serious since Alice may not be aware of the fact that her long-term private key has been compromised. Any protocol that prevents these kinds of attacks is said to be *KCI resilient*.

Appendix B

Modeling Complex Security Protocols with Scyther: Pitfalls and Workarounds

Modeling security protocols in Scyther is pretty straight forward as long as they rely on primitive cryptographic notions. As soon as these protocols comprise state-of-the-art cryptographic primitives, e. g. Diffie-Hellman keys (g^{xy}), things can get pretty ugly. This appendix describes mechanisms to work around commutativity issues and protocol executability and discusses pitfalls that result from implementing these mechanisms.

B.1 Diffie-Hellman Key Agreement

A vast majority of contemporary security protocols use some form of the Diffie-Hellman algorithm to agree on session keys. Modeling such protocols pose some difficulties as the commutative properties of exponentiation are not fully supported by Scyther. The problem is best explained by means of an example.

Example 3 (Traditional Diffie-Hellman)**.** *Let Alice assume the role of the initiator in the traditional Diffie-Hellman key exchange. To start the protocol, Alice randomly chooses her secret exponent $x \in G_p$, where G_p is the multiplicative integer group modulo p with generator g, and sends $\alpha = g^x$ to Bob, who is acting as the responder. Similarly, Bob computes $\beta = g^y$ and sends it to Alice. Finally, Alice and Bob compute their shared Diffie-Hellman key as*

$Z_i = \beta^x$ *and* $Z_r = \alpha^y$*, respectively.*

To capture the commutative nature of modular exponentiation, we recommend the definition of two functions g and h where the semantics are given as $g(x) := g^x$ and $h(\alpha, y) := \alpha^y$. These functions enable the roles of the protocol model to "compute" their view of the shared secret (Z_i and Z_r in the above example). We stress that, although semantically equivalent, the two views are syntactically different because of Scyther operates in a free-term algebra.

The equivalence $Z_i \equiv Z_r$ can be expressed in Scyther by using Scyther's ability to define so-called helper protocols, denoted by a `@`-prefix. Such a protocol typically comprises a single role that receives any message of the form $h(Y, x) = h(g(y), x)$ and sends $h(g(x), y) = h(X, y)$. By this, it is ensured that whenever the adversary knows the session key of e. g. the initiator, she also knows the responder's semantic equivalent. The exact role definition is given in Listing B.1.

Listing B.1: Helper protocol capturing the semantic equivalence of modular exponentiation

```
1  protocol @oracle (DH) {
2
3      /* Diffie-Hellman oracle: If the adversary
4       * is in possession of g^xy, he can obtain
5       * g^yx.
6       */
7      role DH {
8          var x, y: Nonce;
9
10         recv_!DH1( DH, DH, h(g(x),y) );
11         send_!DH2( DH, DH, h(g(y),x) );
12     }
13 }
```

B.2 Message Complexity

Another problem shared by almost all modern security protocols is message complexity. By message complexity we do not necessarily mean the ever increasing number of payloads carried by individual messages, but rather the inherent complexity of the payload itself.

Compared to ancient security protocols, where message payloads merely consisted of single atomic values (i. e. nonces, timestamps,etc.), messages of contemporary protocols often carry composite values such as message authentication codes (MACs) or a signature computed over a dozen of message fields. RFC4306, the document specifying the Internet Key Exchange (IKE) protocol, for instance describes a single authentication payload ($AUTH$) as

$$AUTH = \{\!|\, SPI_i, 0, SA, g^x, N_i, N_r, \mathsf{prf}\,(K_p, I)\,|\!\}_{sk(I)}$$

where SPI_i is a 32bit integer value, SA_i is a set of cryptographic algorithm identifiers, g^{x_i} is a Diffie-Hellman half-key, N_x are nonces, K_p is a negotiated key and I denotes a protocol role, e. g., the initiator. prf denotes a pseudo-random function and by $\{\!|\, M \,|\!\}_{sk(I)}$ we denote the signature over M using secret key $sk(I)$.

When specifying such a protocol in Scyther's specification language, one quickly runs into the problem that each message is larger than the screen can display. Additionally, identical payloads may occur in multiple messages, thus leaving the modeler with copying and pasting message parts most of the time, which is a nightmare (!) regarding maintainability.

To improve maintainability, we recommend employing `cpp`, the preprocessor of the C programming language. `cpp` allows the specification of macros by using the directive

```
#define <identifier> <replacement token list>
```

to replace `identifier` with `replacement token list` whenever it appears in the protocol specification file. To simplify the aforementioned authentication payload $AUTH$ we defined the following macro:

```
#define AUTHi {SPIi, 0, SA, g(i), Ni, Nr, prf(Kp, I)}sk(I)
```

and from thereon only used `AUTHi` whenever a message included this particular authentication payload.

B.3 Protocol Executability

An artifact of the way we recommend treating Diffie-Hellman key agreement in Scyther is that the resulting protocol models may no longer be executable[1]. The reason for this is that

[1]Protocol executability can easily be checked with Scyther's *role characterization* (from menu select Verify→Characterize roles or hit F2)

messages being sent do not match the pattern of the recipient due to syntactic differences. To work around this problem we recommend the following approach.

1. Prefix all message labels that possibly do not match their counterpart by `!` (bang).
2. Define a helper protocol containing a single role which acts as an intermediary and "handles" the matching as follows. For every pair of non-matching messages, the role receives the message being sent, reformats it and subsequently sends it onto the network so it can be received by the original recipient.

Listing B.2 (line 25ff) highlights the process.

Listing B.2: Protocol model for signature based IKEv2

```
1  /*********************************************************************
2   * @protocol     Internet Key Exchange Protocol (IKEv2)
3   * @subprotocol Signature authenticated IKEv2
4   * @reference    RFC 4306
5   * @variant      Includes optional payloads
6   *********************************************************************/
7
8  /**
9   * MACRO DEFINITIONS
10  * Needs preprocessing by cpp before fed to scyther
11  */
12
13 #define __IKEV2__
14 #ifndef __ORACLE__
15 #include "common.h"
16 #endif
17
18 #define AUTHii {SPIi, O, SA1, g(i), Ni, Nr, prf(SKi, I)}sk(I)
19 #define AUTHir {SPIi, O, SA1, Gi, Ni, Nr, prf(SKr, I)}sk(I)
20 #define AUTHri {SPIi, SPIr, SA1, Gr, Nr, Ni, prf(SKi, R)}sk(R)
21 #define AUTHrr {SPIi, SPIr, SA1, g(r), Nr, Ni, prf(SKr, R)}sk(R)
22
23
24 usertype Number, SecurityAssociation, TrafficSelector;
25 const O: Number;
```

```
26 const SA1 ,SA2: SecurityAssociation;
27 const TSi, TSr: TrafficSelector;
28
29 /**
30  * This role serves as an "oracle" to ensure the executability of the
31  * protocol by taking care of the problems that arise from our way of
32  * modelling Diffie-Hellman keys.
33  */
34 protocol @executability(E) {
35 #define Gi g(i)
36 #define Gr g(r)
37     role E {
38         var i, r, Ni, Nr, SPIi, SPIr: Nonce;
39         var I, R: Agent;
40
41         // msg 3
42         recv_!E1( E, E, {I, R, AUTHii, SA2, TSi, TSr}SKi );
43         send_!E2( E, E, {I, R, AUTHir, SA2, TSi, TSr}SKr );
44
45         // msg 4
46         recv_!E3( E, E, {R, AUTHrr, SA2, TSi, TSr}SKr );
47         send_!E4( E, E, {R, AUTHri, SA2, TSi, TSr}SKi );
48
49     }
50 #undef Gi
51 #undef Gr
52 }
53
54
55 protocol ikev2-sig(I, R)
56 {
57     role I {
58         fresh i, Ni, SPIi:  Nonce;
59         var   Nr, SPIr:     Nonce;
60         var   Gr:           Ticket;
61
62
```

```
63          /* IKE_SA_INIT */
64          send_1( I, R, SPIi, O, SA1, g(i), Ni );
65          recv_2( R, I, HDR, SA1, Gr, Nr );
66
67          /* IKE_AUTH */
68          send_!3( I, R, HDR, {I, R, AUTHii, SA2, TSi, TSr}SKi );
69          recv_!4( R, I, HDR, {R, AUTHri, SA2, TSi, TSr}SKi );
70
71          /* SECURITY CLAIMS */
72          claim( I, SKR, SKi );
73
74          claim( I, Alive );
75          claim( I, Weakagree );
76
77      }
78
79      role R {
80          fresh r, Nr, SPIr:  Nonce;
81          var   Ni, SPIi:     Nonce;
82          var   Gi:           Ticket;
83
84
85          /* IKE_SA_INIT */
86          recv_1( I, R, SPIi, O, SA1, Gi, Ni );
87          send_2( R, I, HDR, SA1, g(r), Nr );
88
89          /* IKE_AUTH */
90          recv_!3( I, R, HDR, {I, R, AUTHir, SA2, TSi, TSr}SKr );
91          send_!4( R, I, HDR, {R, AUTHrr, SA2, TSi, TSr}SKr );
92
93          /* SECURITY CLAIMS */
94          claim( R, SKR, SKr );
95
96          claim( R, Alive );
97          claim( R, Weakagree );
98      }
99  }
```

B.4 Pitfalls

This section lists some of the pitfalls that we experienced when using the workarounds described above.

- Depending on the analyzed scenario, the maximum number of runs must be increased to afford for the helper roles. For example, to analyze session key secrecy of IKEv2 (Listing B.2), a minimum of 5 runs is already consumed by the initiator, the responder, the role ensuring executability and two additional roles which let the adversary derive the semantic equivalence of certain Diffie-Hellman keys. To afford for two extra roles which the adversary may actually "use" to attack the protocol, the bound needs to be lifted to 7.

- Using ! (bang) to prevent Scyther from reporting that message matching has failed has the drawback that the property $Nisynch$ (non-injective synchronization) always fails because the property states that messages must be received exactly as they were sent and as specified by the protocol which is not possible because equality checks are based on syntactic equivalence in a free-term algebra.

References

[1] `http://people.inf.ethz.ch/cremersc/scyther/IKE/`. 41, 66

[2] M. Abadi and R. Needham, *Prudent Engineering Practice for Cryptographic Protocols*, Software Engineering, IEEE Transactions on **22** (1996), no. 1, 6–15. 91

[3] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetz, *Extensible Authentication Protocol (EAP)*, RFC 3748 (Proposed Standard), June 2004, Updated by RFC 5247. 37

[4] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold, *Just fast keying: Key agreement in a hostile internet*, ACM Transactions on Information and System Security **7** (2004), no. 2, 242–273. 32

[5] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P. C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron, *The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications*, Computer Aided Verification (Kousha Etessami and Sriram K. Rajamani, eds.), vol. 3576, Springer Berlin Heidelberg, 2005, pp. 281–285. 4

[6] D. Basin and C. J. F. Cremers, *Degrees of Security: Protocol Guarantees in the Face of Compromising Adversaries*, Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings, Lecture Notes in Computer Science, vol. 6247, Springer, 2010, pp. 1–18. 3, 20, 64, 88

[7] ———, *Modeling and Analyzing Security in the Presence of Compromising Adversaries*, Computer Security - ESORICS 2010, Lecture Notes in Computer Science, vol. 6345, Springer, 2010, pp. 340–356. 2, 11, 14, 17, 19, 66

[8] M. Bellare, R. Canetti, and H. Krawczyk, *Keying Hash Functions for Message Authentication*, Springer-Verlag, 1996, pp. 1–15. 25

[9] M. Bellare and P. Rogaway, *Provably Secure Session Key Distribution: The Three Party Case*, Proceedings of the twenty-seventh annual ACM symposium on Theory of computing (New York, NY, USA), STOC '95, ACM, 1995, pp. 57–66. 67

[10] S. Blake-Wilson, D. Johnson, and A. Menezes, *Key Agreement Protocols and Their Security Analysis*, Proceedings of the 6th IMA International Conference on Cryptography and Coding (London, UK), Springer-Verlag, 1997, pp. 30–45. 46

[11] S. Blake-Wilson and A. Menezes, *Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol*, 1999. 92

[12] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*, 1 ed., Springer, September 2003. 68

[13] R. Canetti and H. Krawczyk, *Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels*, EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques (London, UK), Springer-Verlag, 2001, pp. 453–474. 2, 3, 4, 67, 75

[14] ———, *Security Analysis of IKE's Signature-based Key-Exchange Protocol*, In: Proc. CRYPTO'02, Springer LNCS 2442, Springer-Verlag, 2002, pp. 143–161. 2, 3, 75

[15] C. J. F. Cremers, *The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols*, Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, USA, Proc., Lecture Notes in Computer Science, vol. 5123/2008, Springer, 2008, pp. 414–418. 2, 65

[16] ———, *Unbounded Verification, Falsification, and Characterization of Security Protocols by Pattern Refinement*, CCS '08: Proceedings of the 15th ACM conference on Computer and communications security (New York, NY, USA), ACM, 2008, pp. 119–128. 66

[17] T. Dierks and E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, RFC 5246 (Proposed Standard), August 2008, Updated by RFCs 5746, 5878. 6

[18] W. Diffie and M. E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory **IT-22** (1976), no. 6, 644–654. 1

[19] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, *Authentication and Authenticated Key Exchanges*, Des. Codes Cryptography **2** (1992), no. 2, 107–125. 10, 34, 67, 96

[20] N. Ferguson and B. Schneier, *A Cryptographic Evaluation of IPsec*, Tech. report, Counterpane Internet Security, Inc, 2000. 4, 32, 34, 41, 68, 71

[21] C. G. Günther, *An Identity-based Key-exchange Protocol*, EUROCRYPT '89: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology (New York, NY, USA), Springer-Verlag New York, Inc., 1990, pp. 29–37. 67, 96

[22] D. Harkins and D. Carrel, *The Internet Key Exchange (IKE)*, RFC 2409 (Proposed Standard), November 1998, Obsoleted by RFC 4306, updated by RFC 4109. 1, 10

[23] M. Just and S. Vaudenay, *Authenticated Multi-party Key Agreement*, Advances in Cryptology — ASIACRYPT '96 (Kwangjo Kim and Tsutomu Matsumoto, eds.), Lecture Notes in Computer Science, vol. 1163, Springer Berlin / Heidelberg, 1996, 10.1007/BFb0034833, pp. 36–49. 18, 67

[24] B. S. Kaliski Jr., *An Unknown Key-share Attack on the MQV Key Agreement Protocol*, ACM Transactions on Information and System Security **7** (2001), no. 3, 275–288. 92

[25] P. Karn and W. Simpson, *Photuris: Session-Key Management Protocol*, RFC 2522 (Experimental), March 1999. 24

[26] C. Kaufman, *Internet Key Exchange (IKEv2) Protocol*, RFC 4306 (Proposed Standard), December 2005, Updated by RFC 5282. 1, 10, 33, 37, 63

[27] S. Kent and K. Seo, *Security Architecture for the Internet Protocol*, RFC 4301 (Proposed Standard), December 2005. 6, 8

[28] H. Krawczyk, *SKEME: A Versatile Secure Key Exchange Mechanism for Internet*, Network and Distributed System Security, Symposium on **0** (1996), 114. 10

[29] ———, *SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols.*, CRYPTO, 2003, pp. 400–425. 34

[30] ———, *HMQV: A High-Performance Secure Diffie-Hellman Protocol*, Protocol, Advances in Cryptology — CRYPTO '05, LNCS 3621, Springer-Verlag, 2005, pp. 546–566. 18, 67

[31] B. LaMacchia, K. Lauter, and A. Mityagin, *Stronger Security of Authenticated Key Exchange*, Proceedings of the 1st international conference on Provable security (Berlin, Heidelberg), ProvSec'07, Springer-Verlag, 2007, pp. 1–16. 67, 85

[32] G. Lowe, *Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR*, Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems (London, UK), Springer-Verlag, 1996, pp. 147–166. 67

[33] ———, *A Hierarchy of Authentication Specifications*, IEEE Computer Society Press, 1997, pp. 31–43. 20, 72

[34] D. Maughan, M. Schertler, M. Schneider, and J. Turner, *Internet Security Association and Key Management Protocol (ISAKMP)*, RFC 2408 (Proposed Standard), November 1998, Obsoleted by RFC 4306. 9, 10, 22, 24

[35] S. Mauw and V. Bos, *Drawing Message Sequence Charts with LaTeX*, TUGBoat **22** (2001), no. 1-2, 87–92. 48, 64

[36] C. Meadows, *Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer*, 1999. 2, 3, 32, 42, 68, 72, 76

[37] S. Moedersheim, P. H. Drielsma, et al., *AVISPA Project Deliverable D6.2: Specification of the Problems in the High-Level Specification Language.* 2, 4, 72, 93

[38] H. Orman, *The Oakley Key Determination Protocol*, Tech. report, Tucson, AZ, USA, 1997. 10, 22

[39] R. Perlman and C. Kaufman, *Key Exchange in IPSec: Analysis of IKE*, IEEE Internet Computing **4** (2000), no. 6, 50–56. 4, 29, 32, 87

[40] A. Roy, A. Datta, and J. C. Mitchell, *Formal Proofs of Cryptographic Security of Diffie-Hellman-based Protocols*, Tech. report, 2007. 2, 4, 92

[41] J. Zhou, *Further Analysis of the Internet Key Exchange protocol*, Digital Labs, 21 Heng Mui Keng, pp. 1606–1612. 4, 34