ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Formal Analysis of 5G Protocols

Bachelor Thesis

David Lanzenberger

September 9, 2017

Supervisors: Dr. Ralf Sasse, Dr. Lucca Hirschi
Professor: Prof. Dr. David Basin

Department of Computer Science, ETH Zürich

Abstract

Security of telecommunication protocols is paramount. In this work, we formally model protocols of the new 5G standard and analyze them in the symbolic model of cryptographic protocol verification, making use of TAMARIN, a state-of-the-art symbolic protocol analysis tool.

We model two newly proposed 5G protocols called *Relay-Authentication* and *Aggregation-Authentication*, which make use of the Diffie-Hellman key exchange as well as ID-based signatures. Our analysis reveals multiple weaknesses and we propose a number of improvements for the protocol specifications in order to resolve them.

Moreover, we model the 5G authentication and key agreement protocols EPS-AKA* and EAP-AKA'. We provide a comprehensive analysis as well as a comparison between both protocols, showing that EAP-AKA' provides strictly stronger security guarantees than EPS-AKA*.

Finally, we present a modeling approach that leverages TAMARIN's support for mutable global state as well as its built-in exclusive-or (XOR) operator to account precisely for mechanisms of AKA-based protocols that are ignored or greatly simplified in all prior formal models. We apply our approach on EPS-AKA* to obtain a very precise model of the protocol, for which we again verify the same security properties as for our basic model.

Acknowledgements

I would like to express my sincere gratitude to everyone who contributed towards the successful completion of this thesis and who encouraged me throughout those six months.

First, I would like to thank my advisors Ralf Sasse and Lucca Hirschi for their valuable feedback on the draft versions of this thesis, for the illuminating discussions, and for their patient assistance. I could not have imagined having better advisors and mentors.

Furthermore, I would like to thank Professor David Basin for the opportunity to write this thesis in his group.

Finally, I thank my family and friends for their enduring support and continuous encouragement throughout my years of study.

Contents

Chapter 1

Introduction

With smartphones penetrating various facets of everyday life, telecommunication is ubiquitous. Furthermore, novel use cases arising from the Internet of Things (IoT) and so-called *smart devices* are going to increase the importance of telecommunication ever more rapidly.

Customers are increasingly relying on the availability and security of mobile networks, for example, when online banking transactions are authenticated with an SMS-delivered code. Clearly, private communication and, more generally, the transmission of sensitive data over mobile networks makes security an issue of critical importance.

Since the first generation (1G) of mobile network technology was introduced in 1982, a new generation appeared approximately every 10 years. With each of these generations, at least a few improvements concerning security were established. When 2G [20] was deployed in 1991, encryption of voice and text communication was supported for the first time. Subsequently, the 3G standard [3] elevated 2G's one-way authentication to mutual authentication by introducing a new security protocol which allowed the user equipment to authenticate the network it was attaching to. With 4G [4], the key hierarchy has been improved in order to limit the damage caused by a partial compromise of the infrastructure. Moreover, the protocols have been extended in order to provide enhanced privacy protection for the users.

The current draft of the 5G security architecture [6] contains a variety of security protocols. While some of those protocols build on older versions from 3G or 4G, there are numerous new security protocols presented in [6]. For the first time, protocols based on the Diffie-Hellman key exchange and on public key cryptography are taken into consideration.

Even though security protocols are known for being notoriously difficult to design, most protocols that run in today's mobile networks as well as those being developed for the next generation are not designed with the goal

of provable security in mind. Instead, protocols are often developed and deployed before any formal verification, let alone thorough cryptographic analysis. In the past, this has led to multiple attacks being found in widely deployed telecommunication protocols [10, 28], most notably using methods of formal verification. Needless to say, an attack on a widely deployed protocol in the global telecommunication network can cause severe damage. Moreover, it may be very difficult to fix the protocol and it will often involve expensive changes to the network infrastructure or even to the customer's cell phones.

Formal verification methods have been shown to be very effective in finding attacks on security protocols. Inspecting and verifying a protocol in the development phase and before it is deployed has the great advantage of being able to fix possible weaknesses and flaws in its design. Moreover, a successful verification can significantly increase the confidence in a protocol.

In this thesis, we analyze protocols of the upcoming 5G standard, making use of formal verification methods to formally specify the protocols in a symbolic model. Moreover, we prove confidentiality and authentication properties, encoded as reachability properties, of the analyzed protocols, ignoring issues related to privacy for the most part. All of our proofs are carried out with the Tamarin prover, a state-of-the-art symbolic protocol analysis tool, allowing us to automate large parts of most proofs.

1.1 Related Work

Most existing work on authentication and key exchange protocols in mobile networks analyzes the 3GPP AKA protocol [3]. In particular, some security requirements of AKA have been formally proved with an enhanced BAN logic in [2].

Furthermore, attacks on AKA privacy properties have been found using an automatic security protocol verifier called ProVerif in [10]. Moreover, [9] provides a cryptographic analysis of the AKA protocol in a computational model, and an unsuccessful attempt to increase the precision of the model of [10] by using stateful modeling.

Additionally, ProVerif has been used in [28] to formally verify some privacy properties of EAP-SIM as well as EAP-AKA, revealing new attacks on EAP-SIM.

All existing analyses of 3GPP authentication protocols are affected by at least one of the following issues and drawbacks:

- The analysis focuses only on the AKA key exchange protocol. However, in today's mobile networks the AKA protocol is not used in its pure form, but in variants such as EPS-AKA or EAP-AKA that have a

different message format and even exchange some of the values in a different order. This is a problem, because existing formal models and security proofs do not trivially carry over to those variants.

- The analysis is imprecise in at least one of the following ways:
  - The analysis models the protocols in a stateless fashion. This implies that AKA's sequence numbers cannot be modeled precisely, since they require the presence of state to store the number between different sessions. Instead, the protocol is approximated by replacing the sequence number by a different type of value, such as a fresh value for each session that is magically shared between the home network and the user equipment.
  - Additional protocol mechanisms such as re-synchronization or Fast Re-authentication are completely ignored.
  - The exclusive-or operator is either not modeled at all, or it is replaced by a different operator that has simpler algebraic relations.
  - The home network is modeled together with the serving network as one entity participating in the protocol, leading to less precise authentication properties.
- The analysis is outdated and does not take into account recent changes [4, 6] to the protocol.

In this work, we overcome these limitations.

1.2 Contributions

Throughout this work, we will make strong use of the Tamarin prover [32] in order to prove properties of all analyzed protocols. For more details about Tamarin we refer to Section 2.4 on page 21.

We present the following main contributions:

1. A formal analysis and verification of newly proposed authentication protocols which make use of identity-based cryptography. We highlight multiple weaknesses and propose explicit improvements to be added to the protocol specifications.
2. A formal analysis and verification of the complete EPS-AKA* protocol, as well as a comparison with its predecessor EPS-AKA.
3. A formal analysis and verification of the complete EAP-AKA' protocol. Additionally, we formally verify the corresponding re-authentication

mechanism in an isolated model. Furthermore, we provide a comprehensive comparison between EPS-AKA* and EAP-AKA', exposing interesting differences between the two protocols.

4. A precise model for the AKA protocol, leveraging Tamarin's support for mutable global state as well as its built-in exclusive-or (XOR) operator. In particular, we model an incrementing sequence number with its associated re-synchronization protocol. Finally, we demonstrate how this model can be integrated into EPS-AKA* and verify the same security properties as for our less precise model.

In all of our proposed models we aim for a high precision and try to make as few assumptions as possible. The downside of this design choice is that the complexity is considerably higher than in previous models. The Tamarin prover is very efficient and provides good heuristics, making it possible to generate proofs of various properties for different protocols fully automatically. While we are able to benefit greatly from Tamarin and obtain fully automatic proofs in many cases, the increased complexity reaches the tool's limits in some instances. In these cases, it was necessary either to provide additional lemmas that helped to split a proof into smaller parts, or to guide the proof search with manual choices which are automated by encoding them as a heuristic in a so-called oracle.

The results of the analyses mostly provide confidence in the design of the analyzed protocols, meaning that no completely new type of attack was found. Since the specification of the protocols is not always precise and the security goals are not stated accurately, it is however not straightforward to draw a conclusion on the security of some of the protocols.

1.3 Outline

The thesis is structured as follows. In the second chapter, we introduce a general background on 5G, some theory on informal as well as formal security protocols and the Tamarin prover.

In Chapter 3, we explain the basic modeling choices that need to be made for basic primitives, such as key derivation functions or digital signatures.

We proceed in Chapter 4 by presenting a model for the ID-based authentication protocols that are described in Solution #2.16 and Solution #2.15 of [6], followed by an analysis of their properties, revealing multiple weaknesses. We iteratively improve the analyzed protocols until they achieve the desired properties and conclude by proposing a set of corrections and clarifications to incorporate into the protocol specifications.

In Chapter 5, we introduce a basic model for the EPS-AKA protocol as well as its 5G successor EPS-AKA*. Moreover, we analyze and compare their

security properties.

We continue in Chapter 6 by building models for EAP-AKA' as well as its re-authentication mechanism and by formally verifying their properties. Additionally, we provide a comprehensive comparison between the properties of EAP-AKA' and EPS-AKA*.

In Chapter 7, we present a precise model for the sequence number and its re-synchronization mechanism. We show how the approach can be applied to EPS-AKA* and again verify the properties proved in Chapter 5.

Finally, we conclude in Chapter 8 by reflecting on the overall results and the difficulties that arose during the analysis of the protocols.

Chapter 2

Preliminaries

This chapter aims to establish all background information that is necessary for the following discussion of 5G security protocols.

We start by providing some background on 5G. Then, we introduce general notions that are useful when discussing security protocols informally and proceed by formalizing those notions. Finally, we discuss the tool that is used to prove properties of security protocols in this thesis.

2.1 Background on 5G

5G is the fifth generation of mobile network technology, a prospective standard that is currently being developed by the Third Generation Partnership Project (3GPP) [1]. Once completed, 5G is supposed to supersede the current 4G standard.

The aim of this section is to introduce the basic terminology as well as the basic structure of the next generation mobile network. We will restrict ourselves to what is relevant to the discussion of 5G security protocols in the following chapters.

Note that we will omit the prefix *NextGen* or *NG* that is used for many terms within 5G. For example, 3GPP documents sometimes refer to 5G's user equipment (UE) as *NextGen UE* or *NG-UE*.

2.1.1 Basic Network Structure

The mobile network consists of three essential types of entities:

- User Equipment (UE): This is the end user device. The user equipment contains a Universal Integrated Circuit Card ($UICC$), which is a smart card that has a Universal Subscriber Identity Module ($USIM$).

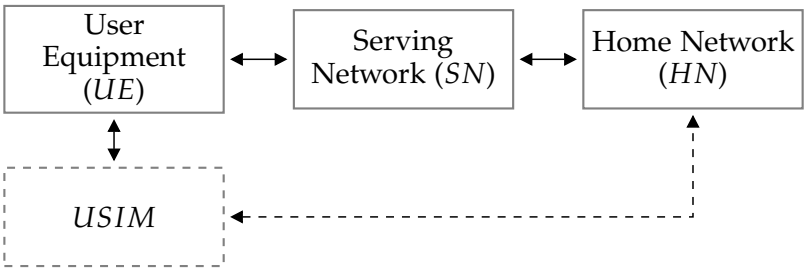

Figure 2.1: Basic network structure of 5G. The dashed line between HN and $USIM$ denotes shared knowledge of the $IMSI$ and the key K.

The $USIM$ contains all necessary information about the subscription at a home network, including an International Mobile Subscriber Identity ($IMSI$) that can be attributed to a unique home network. Moreover, it stores a long-term key K that is shared exclusively with the home network. All cryptographic operations which involve the key K run within the $USIM$. The intention is that K never leaves the $UICC$ smart card.

- Home Network (HN): This is the network that a specific user subscription is bound to. It knows the $IMSI$ and the long-term key K which are stored in the corresponding $USIM$.
- Serving Network (SN): This is the network that the user equipment attaches to via its radio. In the roaming case, the serving network and the home network are run by different network operators.

  Note that the home network and the serving network are usually connected over a secure channel (IPsec or TLS).

Figure 2.1 gives an overview of the basic network structure.

2.1.2 Abbreviations

Appendix B lists abbreviations that are relevant to the treatment of 5G in this thesis.

2.1.3 Security Functions

The current draft for the 5G security architecture [6] introduces new security-related entities.

- *Authentication Credential Repository and Processing Function* ($ARPF$): The $ARPF$ is a system residing in a secure environment in an operator's home network. It stores the long-term security credentials for user equipment authentication and executes any cryptographic algorithms that use those security credentials as input.

For example, the long-term key K that is shared between the home network and the $USIM$ is stored in the $ARPF$.

- *Authentication Server Function* ($AUSF$): This is a system residing in an operator's home network. It interacts with the $SEAF$ (see below) in order to authenticate user equipment.
- *Security Anchor Function* ($SEAF$): This is a physically protected system residing in the serving network. It interacts with the $AUSF$ in order to authenticate user equipment.

Note that we treat the $ARPF$ together with the $AUSF$ as one entity, namely the home network. This suffices for our purposes, since the $ARPF$ is located in a secure environment and can only communicate with the $AUSF$ in its home network. Moreover, we will make no distinction between $SEAF$ and the serving network.

2.1.4 5G Security Protocols

Numerous security protocols have been introduced in [6]. In this thesis, we focus mainly on authentication and key exchange protocols for 3GPP networks that are described in Security Area #2 of [6].

2.2 Informal Security Protocols

It is often useful to discuss security protocols informally before proceeding with a formal analysis. Therefore, we establish an informal understanding of our threat model, security properties, channels, and protocols.

2.2.1 Threat Model

Our threat model assumes a Dolev-Yao adversary similar as described in [17]. In particular, the adversary controls the network, i.e., she can read, intercept, and send messages. Moreover, the adversary can compromise clients, i.e., she can reveal their secrets. Furthermore, the adversary is allowed to apply public functions such as hashing, encryption, or signing on values that she knows.

In addition, our threat model allows unbounded message lengths, an unbounded number of fresh nonces, and an unbounded number of protocol sessions.

2.2.2 Security Properties

We give an informal definition of some basic security properties first, and formalize them in Section 2.3.5.

Definition 2.1 (Authenticity) *Information is* authentic *if the original message sender is who he or she claims to be and the message is unchanged.*

Definition 2.2 (Confidentiality, Secrecy) *Confidentiality (also called secrecy) is the property of information being protected from disclosure to unauthorized parties.*

Definition 2.3 (Integrity) *Information has* integrity *if it is not modified in any way by unauthorized parties.*

Authentication Properties for Protocols

We give an informal definition of a basic hierarchy of increasingly stronger authentication properties for security protocols, similar to [26].

Definition 2.4 (Aliveness, informal, [26]) *A protocol guarantees 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.*

Definition 2.5 (Weak agreement, informal, [26]) *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 2.6 (Non-injective agreement, informal, [26]) *A protocol guarantees to an agent a in role A non-injective agreement with an agent b in role B on a message M if, whenever a completes a run of the protocol, apparently with b in role B, then b has previously been running the protocol, apparently with a, and b was acting in role B in his run, and the two principals agreed on the message M.*

Definition 2.7 (Injective agreement, informal, [26]) *Injective agreement is defined to be non-injective agreement where additionally each run of agent a in role A corresponds to a unique run of agent b.*

The intuitive understanding of injective agreement it that it prevents replay attacks.

2.2.3 Channels

For two parties to exchange messages, it is crucial that they are connected in some way.

Definition 2.8 (Channel) *A* channel *is a logical connection between two parties that can be used to transmit messages.*

Recall that our threat model assumes that the adversary controls the network, i.e., she is able to read and send arbitrary messages of a regular channel. This motivates the following definition of an important type of channel, making use of the defined notions of informal security properties.

Definition 2.9 (Secure channel) *A* secure channel *is a channel that provides confidentiality and authenticity. However, it does not protect from messages being replayed or reordered by the adversary.*

2.2.4 Alice&Bob Notation

In this document, we will specify protocols mostly in an extended form of the so-called Alice&Bob notation before specifying them formally.

We understand protocols as a set of *roles*, where each role consists of a sequence of steps. Each step sends or receives messages. Moreover, we call the protocol participants *agents*. Each agent has a name and can execute a protocol in different roles with other agents.

In a nutshell, Alice&Bob notation is a compact and succinct description of the messages that the protocol agents exchange in absence of an attacker. We will not define the semantics of the Alice&Bob notation rigorously. Instead, we explain the basic conventions and give a small example.

We use the following conventions.

- If an agent receives a message containing a term x and x is known to the agent (e.g., because it is the peer's agent name or the agent has sent or received x in an earlier message of the same protocol run), then it must verify that the values of both x's match. This check is implicit in the notation.
- If an agent receives a message, it will verify its structure up to the level required to recover all subterms it needs, for example to match the values it knows already (see previous point) or to compute subsequent messages. This is crucial, since the recipient's view may differ from the sender's view.

  For example, if an Alice&Bob protocol specification describes a message $\langle x, hash(y)\rangle$ for some hash function $hash$, then an agent who knows only x (and not y) will accept any pair t with x as first element, e.g., $t = \langle x, 0\rangle$ or $t = \langle x, f(\langle 0, 1\rangle)\rangle$ for some function f.
- Messages marked with an asterisk * are optional, i.e., they can be skipped by both the sending and the receiving agent.
- $\{|x|\}_k$ denotes symmetric encryption of the message x with key k. The message x can be recovered from $\{|x|\}_k$ if and only if an agent knows k.
- $\{m\}_{sk}$ denotes the message m signed with key sk. A signature in Alice&Bob notation is always hiding, i.e., the message itself cannot be recovered from the signature.

- $[\![m]\!]_{sk}$ is an abbreviation for $\langle m, \{m\}_{sk}\rangle$. It can be understood as a form of non-hiding signature.
- $\bullet\!\!\rightarrow\!\!\bullet$ denotes a secure channel (see Definition 2.9 on the preceding page).
- Messages of a protocol are numbered consecutively, starting from 1.

Protocol 2.1 (Example)

$$\begin{aligned}
&1.\ A \rightarrow B : \langle A, \{|n|\}_k\rangle \\
&2.\ B \rightarrow A : \{B, n\}_k \\
&3.^*\ B \bullet\!\!\rightarrow\!\!\bullet C : \langle \{n\}_k, k\rangle
\end{aligned}$$

We informally describe how Protocol 2.1 runs.

Assume agent a is executing the protocol in role A with an agent b in role B. Moreover, b is executing the protocol with a in role A and with an agent c in role C.

1. a starts by sending the message $\langle a, \{|n|\}_k\rangle$ over an attacker-controlled channel to b. Here, n is to be understood[1] as a fresh nonce and k is a long-term key shared between a and b.

   When b receives the message, it verifies the value of a in the messages and tries to decrypt the value n.

2. If b has accepted the first message from a, it replies with the message $\{b, n\}_k$ over an attacker-controlled channel.

   When a receives the message, it verifies the value of b and n as well as the signature.

3. After b has sent the second message, it can optionally send the message $\langle \{n\}_k, k\rangle$ to c over a secure channel.

   c will accept the message in any case, even if it has the wrong structure (e.g., if it is a constant string). This is because c does not know the key k or the term $\{n\}_k$, and it does not need to extract one of those terms to compute a subsequent message.

Attack Scenarios

We use so-called attack scenarios to outline protocol attacks. An attack scenario uses the same notation as Alice&Bob protocols. Additionally, we use the following conventions.

[1] Note that this information is not implicit in the Alice&Bob notation. It is necessary to state such facts separately.

- We write $Adv(R)$ to denote the adversary masquerading as agent R.
- Parallel protocol instances are indented in case an attack requires multiple instances.
- We omit messages that are not relevant for the attack.
- We distinguish different runs of the same agent A (if there is more than a single run) with indices $A[0], A[1], \ldots$

Assume that in Protocol 2.1 on the facing page, B would claim injective agreement[2] with A on n after it received the first message. It is easy to see that the property is violated, because the adversary can masquerade as A and resend the first message to B. Attack Scenario 2.1 outlines the attack.

Attack Scenario 2.1 (Example)

$$1.\ A \to B[0] : \langle A, \{|n|\}_k \rangle$$

$$1.\ Adv(A) \to B[1] : \langle A, \{|n|\}_k \rangle$$

The outlined attack violates injective agreement of B with A on the value n. It is a trivial replay attack that makes B accept the same value n twice.

2.3 Formal Protocol Verification

In this section, we introduce a framework called *multiset rewriting* and then describe how it can be used to formally specify protocols. Finally, we formally define several useful security properties.

2.3.1 Term Rewriting

We recall basic notions of term rewriting, following [19].

Definition 2.10 (Signature) *An unsorted* signature Σ *is a set of function symbols, each having an arity $n \geq 0$. Nullary functions are constants.*

Definition 2.11 (Term Algebra, Ground terms) *Let Σ be a signature, $\mathcal{X}$ a set of variables, $\mathcal{V}$ a set of names, such that Σ, $\mathcal{X}$, and $\mathcal{V}$ are pairwise disjoint. We call the set $\mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V})$ the* term algebra *over Σ. It is the least set such that:*

- $\mathcal{X} \cup \mathcal{V} \subseteq \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V})$
- *If $t_1, \ldots, t_n \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V})$ and $f \in \Sigma$ with arity n, then $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V})$*

The set of ground terms *$\mathcal{T}_\Sigma$ consists of terms built without variables, i.e., $\mathcal{T}_\Sigma := \mathcal{T}_\Sigma(\emptyset, \mathcal{V})$.*

[2] See Definition 2.7 on page 10.

It is often useful to specify term algebras with additional properties. For example, we may want a binary function that is symmetric in its arguments, i.e., $f(x, y) = f(y, x)$ for all x and y. Therefore, we define *equations* on terms.

Definition 2.12 (Equation, Equational Theory, Rule) *An* equation *is a pair of terms, written $t = t'$, and a set of equations is called an* equational theory *(Σ, E).*

A rule *is an oriented equation, written $t \twoheadrightarrow t'$ (right-oriented) or $t \twoheadleftarrow t'$ (left-oriented).*

A set of equations E induces a congruence relation $=_E$ on terms and thus the equivalence class $[t]_E$ of a term modulo E. The quotient algebra $\mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V})/_{=_E}$ interprets each term by its equivalence class.

In the following, we define the concepts of *substitution* and *matching*.

Definition 2.13 (Substitution) *A* substitution *is a function $\sigma : \mathcal{X} \to \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V})$ where $\sigma(x) \neq x$ for finitely many $x \in \mathcal{X}$.*

We write substitutions in postfix notation and homomorphically extend them to a mapping $\sigma : \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V}) \to \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V})$ on terms:

$$f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$$

Definition 2.14 (Position) *A* position *p is a sequence of positive integers. The* subterm *$t|_p$ of a term t at position p is obtained as follows.*

- *If $p = []$ is the empty sequence, then $t|_p = t$.*
- *If $p = [i] \cdot p'$ for a positive integer i and a sequence p', and $t = f(t_1, \ldots, t_n)$ for $f \in \Sigma$ and $1 \leq i \leq n$ then $t|_p = t_i|_{p'}$, else $t|_p$ does not exist.*

Definition 2.15 (Matching, Matching substitution) *A term t* matches *a term l if there is a substitution σ so that $t = l\sigma$. We call σ the* matching substitution.

Now, we use the concepts of substitution and matching to define what it means to apply a rule on a term.

Definition 2.16 (Application of a rule) *A rule $l \twoheadrightarrow r$ is applicable on a term t, when a subterm $t|_p$ of t matches l, that is, there is a substitution σ so that $t|_p = l\sigma$.*

The result of such a rule application on t is the term $t[r\sigma]_p$, defined as t with the subterm at position p replaced by the instantiation of the right-hand side of the rule with the matching substitution, $r\sigma$.

2.3.2 Multiset Rewriting

We introduce the basic definitions of multiset rewriting, following [29] and [27]. In a nutshell, multiset rewriting allows specifying the execution of concurrent systems with independent state transitions in a very intuitive and simple way. In particular, it can be used to model security protocols.

Definition 2.17 (Multiset) *A* multiset *m over a set X is a set of elements, each imbued with a multiplicity, i.e., $m : X \to \mathbb{N}$, where $m(x)$ denotes the multiplicity of x.*

Notation 2.18 *We use $\subseteq^\#$ for multiset inclusion, $\cup^\#$ for multiset union, and $\setminus^\#$ for multiset difference without defining them formally.*

Moreover, we use $S^\#$ to denote the set of finite multisets over S.

Definition 2.19 (Fact) *We assume an unsorted signature Σ_{fact} of fact symbols, each with an arity $k \geq 0$. Then*

$$F(t_1, \ldots, t_k)$$

for $F \in \Sigma_{fact}$ with arity k and $t_1, \ldots, t_k \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V})$ is called a fact.

Moreover, a fact is always either linear *or* persistent.

Informally speaking, linear facts may be consumed, whereas persistent facts can be reused arbitrarily often. This notion will be made formal in Definition 2.26.

Definition 2.20 (Labeled multiset rewriting) *A* labeled multiset rewriting rule *is a triple*

$$l \xrightarrow{a} r$$

where l and r are multisets of facts, called state facts and a is a multiset of facts, called action facts or events.

A labeled multiset rewrite system *is a set of multiset rewriting rules.*

Definition 2.21 (Fresh rule) *We define a special rule for the creation of fresh values. This rule has no precondition and it is the only one allowed to produce such Fr facts:*

$$[] \to [Fr(N)]$$

Definition 2.22 (Looping Rule) *A labeled multiset rewriting rule $l \xrightarrow{a} r$ is called a* looping rule, *if $lin(l)\sigma \subseteq^\# r$ for some substitution σ, where $lin(l)$ denotes the multiset of linear facts in l.*

Intuitively, once a looping rule has been applied, an infinite number of consecutive applications of this very rule may follow without using any other rule.

Definition 2.23 (Fresh and public values) *Let FV and PV be two countably infinite and disjoint sets of* fresh values *and* public values. *We use terms in $\mathcal{T}_\Sigma(\mathcal{X}, FV \cup PV)$. Moreover, values in FV are called* fresh.

Notation 2.24 *We prefix a fact symbol F with an exclamation mark* ! *if and only if it is persistent.*

Moreover, we sometimes prefix a value $v \in \mathcal{V}$ with a dollar sign \$ *(respectively with a tilde* $\sim$*) if it is a public value (fresh value), i.e., $v \in PV$ ($v \in FV$).*

In the following, we introduce the concept of *ground instances* and use it to define the *multiset rewriting step*.

Definition 2.25 (Instance, Ground Instance) *An* instance *of an object X (such as a term, fact, or rewrite rule) is the result of applying a substitution σ to all terms in X, written $X\sigma$.*

A ground instance *of X is one where all resulting terms are ground (see Definition 2.11 on page 13).*

For a multiset rewrite system R we use $ginsts(R)$ to denote the set of all ground instances of rules in R. Moreover, the set of ground facts is denoted by $\mathcal{G}$.

Definition 2.26 (Labeled multiset rewriting step) *For a multiset rewrite system R we define the labeled transition relation $steps(R) \subseteq \mathcal{G}^\# \times ginsts(R) \times \mathcal{G}^\#$ as follows:*

$$steps(R) := \big\{(S, l \xrightarrow{a} r, S') \;\big|\; l \xrightarrow{a} r \in ginsts(R), lin(l) \subseteq^\# S, \\ per(l) \subseteq S, S' = (S \setminus^\# lin(l)) \cup^\# r\big\}$$

where $lin(l)$ denotes the multiset of linear facts in l and $per(l)$ denotes the set of persistent facts in l.

Recall that multiset rewriting allows to specify the execution of concurrent systems with independent state transitions. We use multiset rewriting steps to formally define what it means to execute a system specified with multiset rewriting. Moreover, we define the notion of execution *traces*.

Definition 2.27 (State, Execution) *A* state *is a multiset of facts.*

An execution *of a multiset rewrite system R is an alternating sequence:*

$$S_0, (l_1 \xrightarrow{a_1} r_1), S_1, \ldots, S_{k-1}, (l_k \xrightarrow{a_k} r_k), S_k$$

of states and multiset rewrite rule instances such that:

1. *The initial state is empty: $S_0 = \emptyset^\#$.*
2. *The sequence corresponds to a transition sequence, i.e., for all i:*

$$(S_{i-1}, l_i \xrightarrow{a_i} r_i, S_i) \in steps(R)$$

3. *Fresh names are unique, i.e., for all n, i, and j:*

$$(l_i \xrightarrow{a_i} r_i) = (l_j \xrightarrow{a_j} r_j) = ([] \rightarrow [Fr(n)]) \Longrightarrow i = j$$

Definition 2.28 (Trace) *The* trace *of an execution*

$$S_0, (l_1 \xrightarrow{a_1} r_1), S_1, \ldots, S_{k-1}, (l_k \xrightarrow{a_k} r_k), S_k$$

is defined by the sequence of the multisets of its action labels, i.e.,

$$a_1, \ldots, a_k$$

2.3.3 Formal Protocol Description

Having defined multiset rewriting, we can use it in order to formally specify actual protocols.

Definition 2.29 (Message facts) *We use the unary fact $In(m)$ to denote that a message m is received. Analogously, we use the fact $Out(m)$ to denote that a message m is sent.*

Definition 2.30 (Agent state fact) *An* agent state fact *for role R is a fact:*

$$St\_R\_s(A, id, k_1, \ldots, k_n)$$

where $St\_R\_s \in \Sigma_{fact}$ and

- *$s \in \mathbb{N}$ is the number of the protocol step within the role,*
- *A is the name of the agent executing the role,*
- *id is the thread identifier for this instantiation of role R, and*
- *$k_i \in \mathcal{T}_\Sigma(\mathcal{X}, \mathcal{V})$ are terms in the agent's knowledge.*

Definition 2.31 (Protocol rule, Protocol) *A multiset rewriting rule $l \xrightarrow{a} r$ is a* protocol rule*, if it is an initialization rule or if following conditions are satisfied:*

1. *l contains only In, Fr and agent state facts.*
2. *r contains only Out and agent state facts.*
3. *Either In or Out facts occur in the rule, never both.*
4. *Exactly one agent state fact occurs in each of l and r. If the fact*

$$St\_R\_s(A, id, k_1, \ldots, k_n)$$

occurs in l, then the fact $St\_R\_s'(A, id, k_1', \ldots, k_m')$ occurs in r, where $s' = s + 1$.

5. *Every variable in r that is not public most occur in l.*

A protocol *is a finite set of protocol rules.*

We give a small example of how a formal protocol specification with multiset rewriting might look like.

Consider the following protocol, in which n is a nonce generated by A.

Protocol 2.2

$$1.\ A \to B : \langle A, n \rangle$$
$$2.\ B \to A : \langle B, n \rangle$$

The Alice&Bob specification of Protocol 2.2 can be formalized using the following multiset rewrite system.

- Rule `Init_A`: $[Fr(id)] \xrightarrow{Create(A,id)} [St\_A\_0(A, id, B)]$
- Rule `Init_B`: $[Fr(id)] \xrightarrow{Create(B,id)} [St\_B\_0(B, id, A)]$
- Rule `A_1send`:

$$[St\_A\_0(A, id, B), Fr(n)] \to [St\_A\_1(A, id, B, n), Out(\langle A, n \rangle)]$$

- Rule `B_1recv`: $[St\_B\_0(B, id, A), In(\langle A, n \rangle)] \to [St\_B\_1(B, id, A, n)]$
- Rule `B_2send`: $[St\_B\_1(B, id, A, n)] \to [St\_B\_2(B, id, A, n), Out(\langle B, n \rangle)]$
- Rule `A_2recv`: $[St\_A\_1(A, id, B, n), In(\langle B, n \rangle)] \to [St\_A\_2(A, id, B, n)]$

Note that the presented rule naming pattern will be used for multiset rewrite rules in this thesis whenever possible. In particular, an Alice&Bob rule

$$i.\ A \to B : m$$

will be translated to two rules `A_isend` and `B_irecv`.

2.3.4 Formal Message Deduction and Dolev-Yao Adversary

The following set of rules is used for message deduction.

Definition 2.32 (Message deduction rules)

- $[Out(x)] \to [K(x)]$
- $[K(x)] \xrightarrow{K(x)} [In(x)]$
- $[Fr(x)] \to [K(x)]$
- $[K(t_1), \ldots, K(t_k)] \to [K(f(t_1, \ldots, t_k))] \quad \forall f \in \Sigma(k\text{-}ary)$

Intuitively, the rules of Definition 2.32 model a public channel that is controlled by a Dolev-Yao style adversary. The adversary can read and block every message sent. Moreover, she can send any message which is deducible from the messages that she has seen.

2.3.5 Formal Protocol Property Specification

In order to be able to discuss properties of protocols specified with labeled multiset rewriting systems, we instrument the protocol with so-called *events*. Said events are simply facts that can only occur in a label of a multiset rewriting rule.

In the following, we define a set of frequently used events.

Definition 2.33 (Frequently used events) *We define the following events:*

$Create(A, id, R)$	*Initialization event*
$Claim\_\texttt{claimtype}(A, t)$	*Claim event*
$Honest(A), Reveal(A)$	*Honesty and reveal events*
$K(t)$	*Adversary knowledge*

Recall that we assume an adversary that is able to compromise agents. Obviously, many security properties will only hold under the assumption that at least some of the involved agents are not compromised. Therefore, a protocol rule is instrumented with an event $Honest(A)$, if an agent's claims rely on agent A not being compromised by the adversary. Moreover, the rule that lets the adversary compromise an agent A is instrumented with $Reveal(A)$.

Definition 2.34 (Honesty) *An agent A is* honest *in a trace tr if $Reveal(A) \notin tr$.*

The defined events are used in a *property specification language* which is defined as follows.

Definition 2.35 (Property specification language) *We formulate security properties in first-order logic over the following predicates:*

$F@i$	*Timestamped event*
$t = u$	*Term equality*
$i = j$	*Timepoint equality*
$i < j$	*Timepoint inequality*

The predicate F@i holds on trace $tr = a_1, \ldots, a_n$ if $F \in a_i$.

Security Properties

Using the events from Definition 2.33 as well as the property specification language from Definition 2.35, we can formalize the security properties from Section 2.2.2 on page 9.

We begin by defining the formal secrecy properties.

Definition 2.36 (Secrecy)

$$\forall A\, M\, i.\ Claim\_\texttt{secret}(A, M)@i \\ \Rightarrow \neg(\exists j.K(M)@j) \vee (\exists B\, k.Reveal(B)@k \wedge Honest(b)@i)$$

Note that our definition of secrecy differs from classical semantic security which is based on indistinguishability in a computational model. Instead, we think of secrecy in terms of derivability. This is sometimes called weak secrecy.

We proceed by formally defining the notion of Perfect Forward Secrecy (PFS). Intuitively, Perfect Forward Secrecy on a value M means that the adversary does not learn the secret M even if she compromises the long-term keys of the involved agents only *after* PFS was claimed. It is easy to see that PFS implies secrecy.

Definition 2.37 (Perfect Forward Secrecy (PFS))

$$\forall A\, M\, i.\ Claim\_\texttt{secret}(A, M)@i \\ \Rightarrow \neg(\exists j.\ K(M)@j) \vee (\exists B\, k.Reveal(B)@k \wedge Honest(b)@i \wedge k < i)$$

Having defined the secrecy properties, we proceed with the authentication properties.

Definition 2.38 (Aliveness)

$$\forall a\, b\, i.\quad Claim\_\texttt{commit}(a, b, \langle\rangle)@i \\ \Rightarrow (\exists id\, R\, j.\ Create(b, id, R)@j) \\ \vee (\exists X\, r.\ Reveal(X)@r \wedge Honest(X)@i)$$

Definition 2.39 (Weak agreement)

$$\forall a\, b\, i.\quad Claim\_\texttt{commit}(a, b, \langle\rangle)@i \\ \Rightarrow (\exists j.\ Claim\_\texttt{running}(b, a, \langle\rangle)@j) \\ \vee (\exists X\, r.\ Reveal(X)@r \wedge Honest(X)@i)$$

Definition 2.40 (Non-injective agreement)

$$\forall a\, b\, R_1\, R_2\, t\, i.\quad Claim\_\texttt{commit}(a, b, \langle R_1, R_2, t\rangle)@i \\ \Rightarrow (\exists j.\ Claim\_\texttt{running}(b, a, \langle R_1, R_2, t\rangle)@j) \\ \vee (\exists X\, r.\ Reveal(X)@r \wedge Honest(X)@i)$$

Definition 2.41 (Injective agreement)

$$\forall a\, b\, R_1\, R_2\, t\, i.\quad Claim\_\texttt{commit}(a, b, \langle R_1, R_2, t\rangle)@i \\ \Rightarrow (\exists j.\ Claim\_\texttt{running}(b, a, \langle R_1, R_2, t\rangle)@j \\ \wedge \neg(\exists a_2\, b_2\, i_2.\ Claim\_\texttt{commit}(a_2, b_2, \langle R_1, R_2, t\rangle)@i_2 \\ \wedge \neg(i_2 = i))) \\ \vee (\exists X\, r.\ Reveal(X)@r \wedge Honest(X)@i)$$

2.4 Tamarin

Throughout this thesis, we will specify 5G security protocols with multiset rewriting systems and prove security properties for the protocols, making use of the Tamarin prover [30].

Tamarin is a tool that can prove many trace properties fully automatically, using a built-in heuristic to guide the proof search. If a proof search terminates, the tool presents either a correctness proof or a counterexample. However, the proof search may not terminate, in which case manual proof guidance may be necessary to find a proof or a counterexample.

While alternative formal verification tools such as Scyther [16], ProVerif [12], or Maude-NPA [18] can be used to run unbounded symbolic verification as well, these tools fail to analyze protocols that require non-monotonic global state, i.e., state that can be read and altered (by different parallel threads). This makes Tamarin a comparatively powerful tool and a good fit for our purposes.

In this section, we restrict ourselves to basic features of Tamarin along with some difficulties that may arise when using it. For a detailed discussion on the inner workings of Tamarin's algorithm, we refer to [29] and [27].

2.4.1 Equational Theories

Tamarin comes with several built-in equational theories. We present an overview of those that are used in this thesis.

- `asymmetric-encryption`: This theory models asymmetric encryption of messages with a public key encryption scheme. It defines the equation `adec(aenc(m, pk(sk)), sk) = m` for binary functions `aenc`, `adec` and the unary function `pk`.
- `signing`: This theory models a signature scheme. It defines an equation `verify(sign(m,sk),m,pk(sk)) = true` for binary function `sign`, ternary function `verify`, unary function `pk` and the constant function `true`.
- `symmetric-encryption`: This theory models a symmetric encryption scheme. It defines the equation `sdec(senc(m,k),k) = m` for binary functions `senc`, `sdec`.
- `diffie-hellman`: This theory models computation in a finite cyclic group, commonly used in protocols based on the Diffie-Hellman key exchange. It defines the binary function `^` which is used for exponentiation in the group. Moreover, it defines the unary function `inv`, the constant `1`, and the binary function `*` which are used to model the com-

mutative group of exponents. We omit the defined equations, referring to [30].

- `multiset`: This theory models multisets by introducing an associative and commutative operator `+`.
- `xor`: This theory models a commutative and associative operator `XOR` with the neutral element `0` and the equation `x XOR x = 0`. Note that at the time of writing, this theory is only provided in an experimental version of Tamarin.

Furthermore, we will make use of Tamarin's support for user-specified equations and functions.

2.4.2 Proof Strategy and Heuristics

Tamarin's constraint solving algorithm keeps a set of constraints as its state Ω. When it tries to prove a property specified by the formula φ, it starts the search in a state which is constrained by the negated formula, i.e., $\Omega_0 = \{\{\neg\varphi\}\}$. Loosely speaking, the algorithm then picks a *goal* of the current state and tries to solve it, until either the formula is found to be unsatisfiable (the property holds), or until a satisfying trace (representing a counterexample) is found.

The choice of the goal that is to be solved significantly impacts the runtime of the algorithm. Tamarin provides several built-in *heuristics* that can automatically make this decision. The default heuristic is the *smart* heuristic, which works very well for a variety of protocols. For more involved protocols, however, the default heuristic may not be able to generate a proof within a reasonable amount of time. In this case, it may help to use one of the alternative built-in heuristics, or even a combination of them.

Alternatively, it is possible to manually select the proof goals in Tamarin's *interactive* mode.

Oracle

Instead of selecting all proof goals manually in the interactive mode, it is often convenient to write a so-called *oracle*. An oracle is a program that runs completely independent of the Tamarin prover. Its input is a numbered list of proof goals, and its output is an ordered list of numbers, prioritizing which proof goals are to be solved first.

2.4.3 Restrictions

It is often convenient to restrict the set of traces to be considered in the protocol analysis using so-called *restriction* formulas. This is a versatile concept.

For example, assume we want to model an agent comparing two terms x and y. In order to do this, we can simply define a multiset rewrite rule with the label `Eq(x,y)` and add the following restriction to the theory:

```
restriction Eq: "All x y #i. Eq(x,y) @i ==> x = y"
```

Moreover, restrictions are also can be used to model the verification of signatures as well as branching behavior of protocols.

2.4.4 Partial Deconstructions

Before proving properties in a multiset rewriting system, Tamarin goes through a precomputation phase. In this phase, a set of possible sources for the premise of each rule is precomputed.

In some cases, the algorithm cannot resolve the source of some of the fact, in which case so-called *partial deconstructions* are left after the precomputation.

Partial deconstructions complicate automated proof generation significantly. Therefore, it is often beneficial to specify inductive invariants, so-called *sources lemmas* which are used in the precomputation phase only with the specific purpose of removing partial deconstructions. Typically, a sources lemma explains the origin of facts for which the algorithm is not able to derive the origin automatically.

Note that sources lemmas need to be proved themselves by induction. More often than not, Tamarin can prove the required sources lemmas automatically.

2.4.5 Presented Models

In this work, we develop multiple Tamarin models, all of which can be obtained via [24].

Moreover, all proofs are executed with Tamarin 1.3.0 (with Git revision hash `1b4bd1d166a7603ae9c7fda3b81c30a9721d5b8b`) on a machine with Intel Xeon E3-1231 v3 processor and 16GB memory, running Fedora 26.

Chapter 3

General Modeling Decisions

This chapter introduces general modeling decisions that are used throughout this thesis. In particular, we present a new modeling primitive for ID-based signature schemes.

3.1 Key Derivation Functions

Key derivation functions are modeled by a unary free function symbol `KDF`, meaning that there are no defining equations. In case the functions take multiple arguments $a_1, \ldots, a_n$, we pass them all as one tuple $\langle a_1, \ldots, a_n \rangle$. If a protocol makes use of multiple key derivation functions $f_1, \ldots f_n$, we set the first argument in the tuple to the constant string 'fi' when using the function f_i. For example, the key derivation function f_3 with the argument x is represented as `KDF(<'f3',x>)`

3.2 Digital Signatures

We model digital signature schemes by making use of Tamarin's built-in `signing` as explained in Section 2.4.1 on page 21.

Note that in contrast to the convention used in Alice&Bob notation, we treat signatures as hiding in all formal protocol specifications. That is, we use signatures only to authenticate the sender of a message, but never to extract the message itself. We believe that hiding signatures reflect the reality of today's signature schemes more precisely than non-hiding signatures.

3.3 Identity-Based Signatures

We first introduce the basic concepts of identity-based cryptography, before presenting our modeling approach for identity-based signatures.

3.3.1 Concepts of ID-based Cryptography

In a nutshell, identity-based cryptography is a type of public-key cryptography, in which every user's public key is computable from its identity name. There are different ID-based primitives, such as identity-based signatures schemes [31] or identity-based encryption schemes [13, 15].

Every identity-based system has a so-called *Private Key Generator* (PKG), which is a trusted entity that computes the private keys corresponding to each public key. First, the PKG generates a master key pair and publishes the master public key, whereas the master private key is kept secret. Then, participants of the system can compute public keys for arbitrary identities from the master public key and the identity name. In order to obtain a private key, a user authenticates to the PKG, which then generates the private key corresponding to the user's identity.

3.3.2 Model

In the following, we present our modeling approach for identity-based signatures.

We introduce five user-specified functions:

- The binary function `idsign` and the ternary function `idverify` are used to sign and verify messages. They are used completely analogously to the functions from Tamarin's built-in `signing`.
- The unary function `GetIBMasterPublicKey` is used to model the master public key. For a PKG with master private key `IBMasterPrivK`, the corresponding master public key is the term

  `GetIBMasterPublicKey(IBMasterPrivK)`
- The binary functions `IBPub` and `IBPriv` are used to model the derivation of identity-based public and private keys. The public and private key of an agent A for a PKG with master public key `IBMasterPubK` and master private key `IBMasterPrivK` are represented by the terms `IBPub(A,IBMasterPubK)` and `IBPriv(A,IBMasterPrivK)`, respectively.

Furthermore, we extend the equational theory by the following equation:

```
idverify(idsign(m,IBPriv(A, IBMasterPrivateKey)),m,
        IBPub(A, GetIBMasterPublicKey(IBMasterPrivateKey))) = true
```

Note that `true` is user-specified constant, i.e., a nullary function symbol.

We proceed by briefly describing the rules that are used to model the initialization of the important entities in an identity-based setup. Recall that we prefix persistent fact symbols with an exclamation mark ! and public values with a dollar sign $.

The rule `create_IB_PrivateKeyGenerator` is used to generate a master private key for a PKG:

```
[ Fr(IBMasterPrivK) ] →
                          [ !IB_MasterPrivateKey($PKG,
                                                 IBMasterPrivateKey),
                            Out(GetIBMasterPublicKey(IBMasterPrivK)) ]
```

Note that the master public key is sent out. This is crucial, because the adversary needs to know the master public key to be able to generate valid public keys for regular agents.

Next, we present the rule `create_IB_identity`, which uses the created master private key to generate actual ID-based credentials for an arbitrary agent A.

```
[ !IB_MasterPrivateKey(PKG, IBMasterPrivK) ]
→
[ !IB_Identity($A, PKG, GetIBMasterPublicKey(IBMasterPrivK),
               IBPriv($A, IBMasterPrivK)) ]
```

When an agent A instantiates a role R of a protocol, the persistent fact `!IB_Identity(...)` will be used in the rule `initialize_R` as follows.

```
[ Fr(id), !IB_Identity(A, PKG, IBMasterPublicKey, IBPrivKey) ]
   --Create(A,id,'R')-->
[ St_R_0(A, id, PKG, IBMasterPublicKey, IBPrivKey) ]
```

Note that we match only on the full master public key `IBMasterPublicKey`. Hence, the agent instantiating a role does not learn the master private key (unless it is leaked via another way).

3.4 Secure Channels

We model secure channels (see Definition 2.9 on page 11) between two agents by pre-provisioning both with a long-term symmetric key SK. Moreover, we make use of TAMARIN's built-in `symmetric-encryption` (see Section 2.4.1 on page 21) to encrypt and decrypt all messages that the two agents exchange over the secure channel.

Note that this allows the adversary to reorder and replay messages that are sent over the secure channel, which is consistent with our definition of a secure channel.

3.5 Compromise Scenarios

In all our models, we allow the adversary to reveal all long-term secrets from the protocol participants. For example, a secure channel between two

agents is always revealable by the adversary, giving her the ability to read messages are sent over the channel and to send arbitrary new messages over it.

Moreover, we carefully instrument all protocol specifications with precise claims of honesty (see Definition 2.34 on page 19).

3.6 5G Roles and Agent Names

Agent names are of critical importance, especially when proving authentication properties of a protocol. In the following, we present the agent names of the three entity types in 5G (see Section 2.1 of Chapter 2), together with a convention for naming the roles.

- The user equipment is one role, called UE. Its agent name is the user equipment's $IMSI$.
- A serving network and the $SEAF$ that resides within the serving network are treated as one role, called $SEAF$. Its agent name is denoted by $SNid$.

  Note that this modeling choice makes no distinction between $SEAF$ and the serving network.
- A home network and the $AUSF/ARPF$ residing within the home network are treated as one role, called HSS. Its agent name is denoted by HSS.

  Note that treating $AUSF$ and $ARPF$ as one role suffices for our purposes, since the $ARPF$ is located in a secure environment and can only communicate with the $AUSF$ in its home network.

Chapter 4

Protocols Using ID-Based Credentials

In this chapter, we analyze two newly proposed 5G protocols which make use of identity-based (ID-based) credentials. Both protocols are proposed in [6] and make use of the Diffie-Hellman key exchange.

Since [6] is a *Technical Report*, i.e., a proposal for a new standard and not a final specification, the protocol specifications are imprecise. Therefore, we model over-approximations of the protocols in cases of uncertainty. Moreover, we iteratively improve the protocols until the desired properties are achieved. Finally, we propose a set of corrections and clarifications to incorporate into the protocol specifications.

4.1 Relay-Authentication

Solution #2.16 of [6] describes a protocol which we call *Relay-Authentication*. It is a Diffie-Hellman based protocol that assumes the user equipment (UE) to be pre-provisioned with ID-based credentials. The protocol aims for mutual authentication between a remote UE (e.g., a wearable device such as a smart watch) and an authenticator $AUTH$. Moreover, the remote UE is connected with $AUTH$ only over a so-called *relay UE* (e.g., a smartphone).

The Relay-Authentication protocol runs as follows.

Protocol 4.1 (Relay-Authentication PKI v0)

1. $UE \to RELAY : [\![\text{`request'}, UE, g^x]\!]_{sk(UE)}$
2. $RELAY \to AUTH : [\![\text{`request'}, UE, g^x]\!]_{sk(UE)}$
3. $AUTH \to RELAY : [\![AUTH, g^y, g^x]\!]_{sk(AUTH)}$
4. $RELAY \to UE : [\![AUTH, g^y, g^x]\!]_{sk(AUTH)}$
5. $UE \to RELAY : [\![g^y]\!]_{sk(UE)}$
6. $RELAY \to AUTH : [\![g^y]\!]_{sk(UE)}$

Note that the relay node only forwards messages and hence has no influence on the security properties of the protocol. Therefore, we ignore it in our models for reasons of simplicity.

Moreover, we are interested in proving the following properties:

- Secrecy of g^{xy} for both UE and $AUTH$
- Perfect Forward Secrecy of g^{xy} for both UE and $AUTH$
- Injective agreement for UE with $AUTH$ on g^{xy}
- Injective agreement for $AUTH$ with UE on g^{xy}

In case injective agreement does not hold, we try to find the strongest authentication property that holds (if any).

4.1.1 Model Based on Simple PKI

In a first model, we assume a simple public key infrastructure (PKI), in which every agent knows all necessary public keys as well as its own private key.

Protocol 4.2 (Relay-Authentication PKI v1)

$$1.\ UE \to AUTH : [\![\text{`}request\text{`}, UE, g^x]\!]_{sk(UE)}$$
$$2.\ AUTH \to UE : [\![AUTH, g^y, g^x]\!]_{sk(AUTH)}$$
$$3.\ UE \to AUTH : [\![g^y]\!]_{sk(UE)}$$

We model this in the Tamarin theory `RelayAuthPKIv1`.

Confusable Messages

In this first model, there are several ways to exploit the fact that the signed messages can be confused. For example, the adversary can compromise one $AUTH$ agent and trick the UE into signing the message $\langle \text{`}request\text{`}, UE, g \rangle$ by claiming the Diffie-Hellman half-key $g^y = \langle \text{`}request\text{`}, UE, g \rangle$. This is also called a *type flaw attack*, and it makes it easy to violate secrecy and non-injective agreement from the perspective of a different (non-compromised) $AUTH$ agent.

A general method to prevent type flaw attacks is to add tags to the signed messages. At first, it seems that it could be enough to tag the third message and sign $\langle \text{`}confirm\text{`}, g^y \rangle$. Unfortunately, it is still possible to confuse the second message with the first in a very artificial case, in which $AUTH =$ ‘*request*‘ and an agent is running both roles UE and $AUTH$ (with the same key). Finally, if all three signed messages are tagged with distinct tags, all confusion attacks can be successfully prevented:

Protocol 4.3 (Relay-Authentication PKI v2)

$$1.\ UE \to AUTH : [\![\text{`}request\text{`}, UE, g^x]\!]_{sk(UE)}$$
$$2.\ AUTH \to UE : [\![\text{`}response\text{`}, AUTH, g^y, g^x]\!]_{sk(AUTH)}$$
$$3.\ UE \to AUTH : [\![\text{`}confirm\text{`}, g^y]\!]_{sk(UE)}$$

Incorporating the tags into the model results in theory `RelayAuthPKIv2`.

Weak Agreement

While Perfect Forward Secrecy (PFS) is automatically proved in the second model `RelayAuthPKIv2`, weak agreement for either UE or $AUTH$ is still not provided:

- If the adversary compromises the agent $AUTH_1$, she can mount a man-in-the-middle attack by forwarding the messages from an agent UE to a non-compromised agent $AUTH_2$ and re-signing the response from $AUTH_2$ with the compromised key. When UE sends the final message, the adversary can simply forward it to the non-compromised agent $AUTH_2$. This violates weak agreement for $AUTH_2$.

  We outline the described attack in Attack Scenario 4.1.

- Analogously, the adversary can compromise one agent UE_1 and mount a man-in-the-middle attack on a non-compromised agent UE_2. This violates weak agreement for UE_2.

Attack Scenario 4.1 *Let $AUTH_1$ and $AUTH_2$ be the (distinct) agent identifiers. Moreover, let the adversary have compromised $AUTH_1$, i.e., she is able to sign messages with $sk(AUTH_1)$.*

$$1.\ UE \to AUTH_1 : [\![\text{`}request\text{`}, UE, g^x]\!]_{sk(UE)}$$
$$\quad 1.\ Adv(UE) \to AUTH_2 : [\![\text{`}request\text{`}, UE, g^x]\!]_{sk(UE)}$$
$$\quad 2.\ AUTH_2 \to Adv(UE) : [\![\text{`}response\text{`}, AUTH, g^y, g^x]\!]_{sk(AUTH_2)}$$
$$2.\ Adv(AUTH_1) \to UE : [\![\text{`}response\text{`}, AUTH, g^y, g^x]\!]_{sk(AUTH_1)}$$
$$3.\ UE \to AUTH_1 : [\![\text{`}confirm\text{`}, g^y]\!]_{sk(UE)}$$
$$\quad 3.\ Adv(UE) \to AUTH_2 : [\![\text{`}confirm\text{`}, g^y]\!]_{sk(UE)}$$

Note that $AUTH_2$ executes the protocol with UE as its peer, whereas UE executes it with $AUTH_1$ as its peer. Moreover, note that neither UE nor $AUTH_2$ have been compromised by the adversary. Hence, this attack violates weak agreement for $AUTH_2$ with UE.

This attack can be prevented by including the peer's agent name into the second and third signed message (reminiscent of Lowe's fix to the Needham-Schroeder protocol described in [25]):

Protocol 4.4 (Relay-Authentication PKI v3)

1. $UE \rightarrow AUTH : [\![\text{'request'}, UE, g^x]\!]_{sk(UE)}$
2. $AUTH \rightarrow UE : [\![\text{'response'}, AUTH, g^y, UE, g^x]\!]_{sk(AUTH)}$
3. $UE \rightarrow AUTH : [\![\text{'confirm'}, g^y, AUTH]\!]_{sk(UE)}$

Incorporating this change into the model results in theory `RelayAuthPKIv3`.

Now, all desired properties are proved successfully. Tables 4.1 and 4.2 summarize the security properties of the Relay-Authentication protocol for all three versions.

	Perspective					
	UE			$AUTH$		
	v1	v2	v3	v1	v2	v3
Secrecy	×	✓	✓	×	✓	✓
PFS	×	✓	✓	×	✓	✓

Table 4.1: Secrecy properties of `RelayAuthPKI[v1,v2,v3]`. All properties are with respect to g^{xy}.

	Perspective					
	$UE \mapsto AUTH$			$AUTH \mapsto UE$		
	v1	v2	v3	v1	v2	v3
Aliveness	×	✓	✓	✓	✓	✓
Weak agreement	×	×	✓	×	×	✓
Non-injective agreement	×	×	✓	×	×	✓
Injective agreement	×	×	✓	×	×	✓

Table 4.2: Authentication properties of `RelayAuthPKI[v1,v2,v3]`. All properties are with respect to g^{xy}. The role name left of the arrow $\mapsto$ denotes the role claiming the corresponding property, while the role name right of the arrow denotes its peer. For example, injective agreement for UE with $AUTH$ has the perspective $UE \mapsto AUTH$.

4.1.2 Model with ID-based Credentials

All observed problems based on the simple PKI model immediately extend to an ID-based setup. Therefore, we start with theory `RelayAuthPKIv3` representing Protocol 4.4 and modify it, using the definitions and rules from Section 3.3 for ID-based modeling. The resulting theory is `RelayAuthIBS`.

All desired properties are proved successfully, just as in `RelayAuthPKIv3` (see Tables 4.1 and 4.2).

4.1.3 Conclusion

We propose three crucial changes to the specification in Solution #2.16 of [6], referring to the messages of Protocol 4.2 on page 30 and establishing the correspondence to [6]:

- All protocol messages should be extended by a unique tag in order to prevent them from being confused.
- The response message of $AUTH$ (message 2, corresponding to the *Authentication Response* in step 6 of Solution #2.16 in [6]) should include the identity name of the remote UE.
- The response message of UE (message 3, corresponding to the *Remote UE Authentication Response* in step 9 of Solution #2.16 in [6]) should include the identity name of the authenticator.

With all these changes applied to the protocol, we are able to prove the following properties automatically in our model:

- Perfect Forward Secrecy of g^{xy} for both UE and $AUTH$.
- Mutual injective agreement on g^{xy} between UE and $AUTH$.

Moreover, we have shown that only very weak security properties hold without the proposed changes.

4.2 Aggregation-Authentication

We proceed by analyzing the *Aggregation-Authentication* protocol proposed in Solution #2.15 of [6]. It strongly builds on Solution #2.16 (discussed in Section 4.1). In particular, the Aggregation-Authentication protocol also makes use of ID-based credentials as well as the Diffie-Hellman key exchange.

In contrast to Relay-Authentication, Aggregation-Authentication is designed to allow efficient mutual authentication in a situation in which many different devices (e.g., a group of IoT devices) are trying to access a network at the same time. The fundamental idea is to have the network authenticator $AUTH$ generate only a single Diffie-Hellman half-key which is used not only for one peer, but for a large batch of peers. Since each peer still generates its own half-key, this technique allows only the authenticator $AUTH$ to benefit in terms of efficiency.

The Aggregation-Authentication protocol runs as follows.

Protocol 4.5 (Aggregation-Authentication v0)

1. $UE_1 \rightarrow AGGR : [\![UE_1, g^{x_1}]\!]_{sk(UE_1)}$

...

1. $UE_n \rightarrow AGGR : [\![UE_n, g^{x_n}]\!]_{sk(UE_n)}$
2. $AGGR \rightarrow AUTH : \left\langle [\![UE_1, g^{x_1}]\!]_{sk(UE_1)}, ..., [\![UE_n, g^{x_n}]\!]_{sk(UE_n)} \right\rangle$
3. $AUTH \rightarrow AGGR : [\![AUTH, g^y, \langle g^{x_1}, \ldots, g^{x_n} \rangle]\!]_{sk(AUTH)}$
4. $AGGR \rightarrow UE_1, \ldots, UE_n : [\![AUTH, g^y, \langle g^{x_1}, \ldots, g^{x_n} \rangle]\!]_{sk(AUTH)}$
5. $UE_1 \rightarrow AGGR : \left\langle UE_1, [\![g^y]\!]_{sk(UE_1)} \right\rangle$

...

5. $UE_n \rightarrow AGGR : \left\langle UE_n, [\![g^y]\!]_{sk(UE_n)} \right\rangle$
6. $AGGR \rightarrow AUTH : \left\langle UE_1, [\![g^y]\!]_{sk(UE_1)}, ..., UE_n, [\![g^y]\!]_{sk(UE_n)} \right\rangle$

Intuitively, each UE_i is simply generating a Diffie-Hellman half-key g^{x_i} and sending it to $AGGR$. Then, $AGGR$ aggregates all received half-keys into one message and forwards it to $AUTH$ in message 2. Then, $AUTH$ generates a Diffie-Hellman half-key g^y which it reuses for each UE_i that wants to connect. Next, $AUTH$ creates a signed message that includes its own half-key g^y as well as all half-keys g^{x_i} and sends it out via message 3. Finally, each UE_i confirms $AUTH$'s half-key by sending a signed copy of it in message 5.

Unfortunately, [6] does not describe how the aggregated signature (for message 2) is generated or verified. It is, however, clear that such an aggregated signature must have the property that the receiver is able to verify that each of the received requests has been signed by the corresponding UE. Therefore, we model the aggregated signature in the simplest possible way that achieves this minimal property. As presented in the Alice&Bob version of Protocol 4.5, the different signed messages $[\![UE_i, g^{x_1}]\!]_{sk(UE_i)}$ are aggregated into one message by inserting them into a tuple.

Note that in practice, it would be best to use a signature scheme that supports space-efficient aggregate signatures [14, 22]. However, in such specialized signature schemes it is usually not possible to split up an aggregated signature into its initial signatures. We note that with our modeling choice, this is trivially possible. Furthermore, we believe this to be attack preserving, since it extends the adversary's possibilities to manipulate protocol messages.

In the original protocol described in [6], UE does not resend its own name together with the signed half-key of $AUTH$ in message 5. We add this, because it is required for $AUTH$ to be able to verify the received signature.

Note also that $AGGR$ only has the role of aggregating and forwarding messages. It has no influence on the security of the protocol. Therefore, we skip these intermediate forwarding steps for simplicity, which is completely attack preserving.

Protocol 4.6 (Aggregation-Authentication v1)

$$
\begin{aligned}
&1.\ UE_1 \to AUTH : [\![UE_1, g^{x_1}]\!]_{sk(UE_1)} \\
&\ldots \\
&1.\ UE_n \to AUTH : [\![UE_n, g^{x_n}]\!]_{sk(UE_n)} \\
&2.\ AUTH \to UE_1, \ldots, UE_n : [\![AUTH, g^y, \langle g^{x_1}, \ldots, g^{x_n} \rangle]\!]_{sk(AUTH)} \\
&3.\ UE_1 \to AUTH : \left\langle UE_1, [\![g^y]\!]_{sk(UE_1)} \right\rangle \\
&\ldots \\
&3.\ UE_n \to AUTH : \left\langle UE_n, [\![g^y]\!]_{sk(UE_n)} \right\rangle
\end{aligned}
$$

4.2.1 Learning from Relay-Authentication

In case only a single UE participates in the protocol, this is essentially the same as the original version of the Relay-Authentication protocol analyzed in Section 4.1 on page 29. Hence, all attacks and issues that have already been pointed out there trivially extend to the Aggregation-Authentication protocol. Note that we provide no TAMARIN model for Protocol 4.6.

We modify the protocol to prevent all issues discussed in Section 4.1. In particular, we extend all three messages with a unique tag, we add each UE_i's identity name to its half-key in the second message, and we add $AUTH$'s identity name to the third message.

Protocol 4.7 (Aggregation-Authentication v2)

$$
\begin{aligned}
&1.\ UE_1 \to AUTH : [\![\text{`request'}, UE_1, g^{x_1}]\!]_{sk(UE_1)} \\
&\ldots \\
&1.\ UE_n \to AUTH : [\![\text{`request'}, UE_n, g^{x_n}]\!]_{sk(UE_n)} \\
&2.\ AUTH \to UE_1, \ldots, UE_n : [\![\text{'response'}, AUTH, g^y, \\
&\qquad\qquad \langle \langle UE_1, g^{x_1} \rangle, \ldots, \langle UE_n, g^{x_n} \rangle \rangle]\!]_{sk(AUTH)} \\
&3.\ UE_1 \to AUTH : \left\langle UE_1, [\![\text{`}confirm\text{'}, g^y, AUTH]\!]_{sk(UE_1)} \right\rangle \\
&\ldots \\
&3.\ UE_n \to AUTH : \left\langle UE_n, [\![\text{`}confirm\text{'}, g^y, AUTH]\!]_{sk(UE_n)} \right\rangle
\end{aligned}
$$

4.2.2 Building a Model

$AUTH$ needs to collect and store all messages from UEs in its local state and then sign all Diffie-Hellman half-keys at once.

In general, multisets turn out to be convenient to model the necessary collection of half-keys, as Tamarin supports them natively and it is easy to check whether an element is contained in a multiset via simple pattern matching (whereas a pair-based collection would require a looping rule[1]).

Our first model, implemented in the Tamarin theory `AggregateAuthIBSv2`, achieves this with a looping rule that is adding a pair $\langle UE, g^x \rangle$ to a multiset for every received request. Finally, receiving and verifying the confirmation messages is modeled with a *looping rule*, too.

Unfortunately, the two looping rules and the large state make it very difficult to prove security properties in Tamarin. In fact, it seems that even executability of the protocol is provable automatically (or takes a very long time) unless the formulas are constrained heavily, guiding the search in the right direction.

To facilitate efficient automatic proving, we let the adversary construct the required collection of half-keys, i.e., we add a message to the protocol that is intended to be sent by the adversary (denoted by a channel arrow $\rightarrow$ without sender name):

Protocol 4.8 (Aggregation-Authentication v3)

$$
\begin{aligned}
&1.\ UE_1 \rightarrow AUTH : [\![\text{`}request\text{`}, UE_1, g^{x_1}]\!]_{sk(UE_1)} \\
&\ldots \\
&1.\ UE_n \rightarrow AUTH : [\![\text{`}request\text{`}, UE_n, g^{x_n}]\!]_{sk(UE_n)} \\
&\qquad \rightarrow AUTH : \langle \langle UE_1, g^{x_1} \rangle, \ldots, \langle UE_n, g^{x_n} \rangle \rangle \\
&2.\ AUTH \rightarrow UE_1, \ldots, UE_n : [\![\text{`}response\text{'}, AUTH, g^y, \\
&\qquad\qquad \langle \langle UE_1, g^{x_1} \rangle, \ldots, \langle UE_n, g^{x_n} \rangle \rangle]\!]_{sk(AUTH)} \\
&3.\ UE_1 \rightarrow AUTH : \left\langle UE_1, [\![\text{`}confirm\text{`}, g^y, AUTH]\!]_{sk(UE_1)} \right\rangle \\
&\ldots \\
&3.\ UE_n \rightarrow AUTH : \left\langle UE_n, [\![\text{`}confirm\text{`}, g^y, AUTH]\!]_{sk(UE_n)} \right\rangle
\end{aligned}
$$

Note that this additional message is a composition of messages known to the adversary. Hence, this is a completely attack preserving transformation of the original protocol.

The new message allows us to completely avoid looping rules, using the following modeling techniques:

[1] See Definition 2.22 on page 15.

- $AUTH$ generates its half-key g^y at the initialization, producing a persistent fact `!St_AUTH_0(...)`. For arbitrarily many UE, we create a new state fact `St_AUTH_1(...)` from `!St_AUTH_0(...)`.

  This can be understood as having multiple *sub-instances* of an agent's running protocol instance. Every sub-instance is responsible for receiving the request and the confirmation message of exactly one UE.

- After $AUTH$ has received the collection of half-keys from the adversary, it verifies whether a corresponding signed request has been received for all occurring half-keys[2]. In the original protocol, this would be checked implicitly, as the collection of half-keys is constructed by $AUTH$ instead of the adversary. To avoid another looping rule, we model this check via a restriction in the Tamarin theory.

We implement this new protocol in theory `AggregateAuthIBSv3`.

4.2.3 Security Properties

Secrecy and Perfect Forward Secrecy on g^{xy} are proved automatically for both UE and $AUTH$. Moreover, injective agreement for UE with $AUTH$ on g^{xy} is proved automatically. However, Tamarin finds attacks for both non-injective and injective agreement for $AUTH$ with UE on g^{xy}. In the following, we discuss the reasons for this and possible remedies.

Even though letting the adversary construct the aggregated signature tuple is completely attack preserving, it is possible that attacks are not applicable to the unmodified, more restrictive, protocol. Therefore, it is worth mentioning how a discovered attack relates to the original protocol.

Non-Injective Agreement for $AUTH$

Non-injective agreement from $AUTH$ with UE can be violated, because UE only confirms $AUTH$'s half-key but does not make any statement about its own half-key again.

This allows the following attack scenario. An agent is running two instances of the protocol as UE, sending out two requests with distinct half-keys g^{x_1} and g^{x_2}. The attacker can prevent one of the two instances from receiving $AUTH$'s response. After $AUTH$ has received the signed confirmation message, it should only commit to the shared secret constructed using the half-key of the instance which has received the response. Unfortunately, the confirmation message gives $AUTH$ no information to decide which instance this is. If it decides on the wrong half-key (or even on both half-keys),

[2]Note that in the original protocol, additionally the collection would not be allowed to contain elements which do not have a corresponding signed request. It turns out that it is no problem to allow this.

non-injective agreement with UE is violated. Note that weak agreement is unaffected by this.

It is easy to see that this attack can indeed be used to exploit the original protocol.

A possible solution is that $AUTH$ makes sure it accepts at most one half-key per UE, for example the first that has been received. Another way to address this issue would be to add UE's half-key to the signed confirmation message. We incorporate the former approach with a restriction into the model, resulting in the new theory `AggregateAuthIBSv4`. Non-injective agreement for $AUTH$ is now proved successfully.

Injective Agreement for $AUTH$

Assume UE_1 and UE_2 execute the protocol with the same $AUTH$. After UE_1 has sent its half-key g^{x_1}, the adversary can compromise UE_2 and send the same half-key g^{x_1}, masquerading as UE_2. Having compromised UE_2, the adversary can easily send the confirmation message on behalf of UE_2 and trick $AUTH$ into agreeing with both UE_1 and UE_2 on the same secret. Note that adversary does not learn the secret, but this violates injective agreement from $AUTH$ with UE_1.

It is easy to see that this attack extends to the original protocol. However, one could argue that it is not an issue concerning the security of the protocol. Recall that the intuitive idea behind injective agreement is to prohibit replay attacks. The formal definition of injective agreement captures this by demanding that for every role and every value there can be at most one commitment. This is a very strong requirement, and for this aggregated authentication protocol one might argue that it is too strong. We demonstrate this by proving the weakened injective agreement property that has been derived by allowing the attack described above.

Definition 4.1 (Weakened injective agreement)

$$
\begin{aligned}
\forall a\ b\ t\ i.\quad & Claim\_\texttt{commit}(a, b, \langle R_1, R_2, t\rangle)@i \\
& \Rightarrow (\exists j.\ Claim\_\texttt{running}(b, a, \langle R_1, R_2, t\rangle)@j \\
& \wedge \neg(\exists a_2\ b_2\ i_2.\ Claim\_\texttt{commit}(a_2, b_2, \langle R_1, R_2, t\rangle)@i_2 \wedge \neg(i_2 = i) \\
& \qquad\qquad \wedge \neg(a = a_2 \wedge \exists r.\ Reveal(b_2)@r \wedge r < i_2))) \\
& \vee (\exists X\ r.\ Reveal(X)@r \wedge Honest(X)@i)
\end{aligned}
$$

It is easy to see that weakened injective agreement fits into Lowe's hierarchy of authentication, meaning that injective agreement implies weakened injective agreement and weakened injective agreement implies non-injective agreement.

The weakened injective agreement property for $AUTH$ with UE allows every agent in the role $AUTH$ to make arbitrarily many additional commitments on a value with compromised UEs. Intuitively, this means that we allow multiple key exchanges on the same key g^{xy} as long as no two different UEs that are both honest are involved. If we additionally have secrecy of the value g^{xy} (as in this case), one might argue there is no security issue concerning authentication. However, this depends entirely on the exact threat model and on the way the agreed-upon values are to be used in following protocols. For example, if $AUTH$ sends critical commands (signed with a key derived from g^{xy}) to UE immediately after a successful execution of the authentication protocol, then $AUTH$ may be tricked into sending multiple of those commands, signed with the *same* key g^{xy}. This may make the honest UE susceptible to replay attacks that are impossible if an authentication protocol providing full injective agreement is executed.

A possible solution to provide full injective agreement is that $AUTH$ accepts every half-key at most once. For example, $AUTH$ can simply ignore a half-key if it has already received the same half-key from a different UE[3]. Using a restriction, we incorporate this into the model, resulting in the new theory `AggregateAuthIBSv5`. Injective agreement for $AUTH$ is now proved successfully.

	Perspective					
	UE			$AUTH$		
	v3	v4	v5	v3	v4	v5
Secrecy	✓	✓	✓	✓	✓	✓
PFS	✓	✓	✓	✓	✓	✓

Table 4.3: Secrecy properties of `AggregateAuthIBS[v3,v4,v5]`. All properties are with respect to g^{xy}.

Tables 4.3 and 4.4 summarize the security properties of the last three versions of the Aggregation-Authentication protocol.

4.2.4 Conclusion

We propose a set of changes to the protocol specification in Solution #2.15 of [6], referring to the messages in Protocol 4.6 on page 35 and establishing the correspondence to [6]:

- All protocol messages should be extended by a unique tag in order to prevent them from being confused.

[3]Note that such a check needs only to be applied within one session and needs only to store the half-keys of the current session. Hence, this approach does not require any additional storage.

	Perspective					
	$UE \mapsto AUTH$			$AUTH \mapsto UE$		
	v3	v4	v5	v3	v4	v5
Aliveness	✓	✓	✓	✓	✓	✓
Weak agreement	✓	✓	✓	✓	✓	✓
Non-injective agreement	✓	✓	✓	×	✓	✓
Weakened injective agreement	✓	✓	✓	×	✓	✓
Injective agreement	✓	✓	✓	×	×	✓

Table 4.4: Authentication properties of `AggregateAuthIBS[v3,v4,v5]`. All properties are with respect to g^{xy}.

- UE_i's identity name should be added to its half-key in message 2 of the protocol (corresponds to the message from step 6 of Solution #2.16 in [6]).
- The authenticators identity name should be added to message 3 (corresponds to message 9 of Solution #2.16 in [6]).
- The authenticator should check that it accepts at most one half-key per UE in message 2 (corresponds to message 4 of Solution #2.16 in [6]).
- The authenticator should check that it accepts every half-key at most once per session in message 2 (corresponds to message 4 of Solution #2.16 in [6]).

With all these changes applied to the protocol, we are able to prove the following properties automatically in our model:

- Perfect Forward Secrecy of g^{xy} for both UE and $AUTH$.
- Mutual injective agreement on g^{xy} between UE and $AUTH$.

Chapter 5

EPS-AKA*

This chapter addresses EPS-AKA*, which is described in Solution #2.22 of [6] as well as in Section 6.1.3.2 of [5]. EPS-AKA* is an authentication and key agreement protocol proposed for 5G, building on EPS-AKA [3] which is currently used in 4G telecommunication networks. We model both EPS-AKA and EPS-AKA* in order to be able to compare the security properties of the protocols.

Recall the basic 5G network structure from Section 2.1 as well as the corresponding role and agent names described in Section 3.6. Our protocol model has three essential roles: UE, $SEAF$ and HSS.

- UE is the user equipment (e.g., a smartphone). It contains a $USIM$ with a long-term key K that is shared with the home network, and a sequence number SQN that is synchronized with HSS in order to prevent replay attacks.
- $SEAF$ resides in the serving network. The serving network's identity name is denoted by $SNid$.
- HSS is the home network in which the $AUSF$ and $ARPF$ reside. Additionally, it knows the long-term key K that is shared with UE and the sequence number SQN that is synchronized with UE.

Moreover, all communication between $SEAF$ and $AUSF$ is envisaged to be Diameter-based, and runs over IPSec or TLS.

EPS-AKA and EPS-AKA* both are authentication and key agreement protocols. We are interested in the following security properties, all with respect to the key the protocol tries to agree on:

- Secrecy for $UE, HSS, SEAF$.
- Perfect Forward Secrecy for $UE, SEAF, HSS$ with respect to the permanent key K between UE and HSS.

- Injective agreement for UE with $SEAF$ and UE with HSS.
- Injective agreement for $SEAF$ with UE and HSS with UE.

Note that we do not discuss authentication properties between HSS and $SEAF$, because we assume they have already set up a secure channel.

In case injective agreement does not hold, we try to find the strongest authentication property that holds (if any).

5.1 EPS-AKA

EPS-AKA is the 4G authentication and key agreement protocol. It is specified in [4], building on UMTS AKA as described in [3]. The most important difference between UMTS AKA and EPS-AKA is that the values CK and IK in the authentication vector have been replaced by a new key K_{ASME} that depends on $SNid$.

Let f_1, f_2 be MAC-functions and f_3, f_4, f_5, KDF key generating functions. Moreover, the specification allows f_5 to be the constant zero function.

We define the following values:

- $MAC := f_1(K, AMF, SQN, RAND)$
- $XRES := RES := f_2(K, RAND)$, where $XRES$ is simply another name for RES.
- $CK := f_3(K, RAND)$, also called *confidentiality key*
- $IK := f_4(K, RAND)$, also called *integrity key*
- $AK := f_5(K, RAND)$, also called *anonymity key*
- $AUTN := \langle SQN \oplus AK, AMF, MAC \rangle$
- $K_{ASME} := KDF(CK, IK, SNid, SQN \oplus AK)$
- $AV := \langle RAND, XRES, K_{ASME}, AUTN \rangle$

EPS-AKA runs as follows.

Protocol 5.1 (EPS-AKA v0)

1. $SEAF \rightarrow UE : \text{`}IdentityRequest\text{`}$
2. $UE \rightarrow SEAF : IMSI$
3. $SEAF \bullet\!\!\rightarrow\!\!\bullet HSS : \langle IMSI, SNid \rangle$
4. $HSS \bullet\!\!\rightarrow\!\!\bullet SEAF : AV$
5. $SEAF \rightarrow UE : \langle RAND, AUTN \rangle$
6. $UE \rightarrow SEAF : RES$

Note the following:

- $RAND$ is a fresh random value generated by HSS.
- SQN is the sequence number generated by HSS. In this chapter, we treat the sequence number as a fresh value that is generated by HSS. The actual mechanism and a model for it is discussed in Chapter 7.
- $SNid$ is the identifier of the serving network which is represented by the role $SEAF$ in our model.
- The Authentication Management Field (AMF) is a 16-bit value, where the least significant bit is the AMF separation bit (see [4]). This is used to distinguish authentication vectors for UMTS AKA from those for EPS-AKA. Bits 1-7 are reserved and 8-15 are for proprietary purposes.
- Actually, HSS can send back many authentication vectors with consecutive sequence numbers SQN. However, in EPS-AKA* only one authentication vector shall be sent, which is why we decide to make this restriction already in the model of EPS-AKA.

Unfortunately, the protocol has a major correctness issue when specified like this. It is easy to see that an agent $SEAF$ that executes multiple instances of the protocol (in the role $SEAF$) in parallel has no information to match the different authentication vectors AV received from HSS to the corresponding $IMSI$ identifying UE. Remember that the communication between HSS and $SEAF$ is Diameter based. As specified in [21], Diameter includes a so-called *Session-Id AVP*, which must be set to a locally unique value for every new session. When receiving a response, the Session-Id AVP is used to match with the corresponding session[1].

We only include the mentioned Session-Id AVP into the messages between HSS and $SEAF$ and ignore all other Diameter headers.

Protocol 5.2 (EPS-AKA v1)

1. $SEAF \rightarrow UE : \text{`}IdentityRequest\text{`}$
2. $UE \rightarrow SEAF : IMSI$
3. $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle S, IMSI, SNid \rangle$
4. $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle S, AV \rangle$
5. $SEAF \rightarrow UE : \langle RAND, AUTN \rangle$
6. $UE \rightarrow SEAF : RES$

The main goal of EPS-AKA is for UE to agree with each of $SEAF$ and HSS on K_{ASME} and vice versa. However, K_{ASME} is the only value depending on

[1] In case no session is found, the received response message must be discarded.

$SNid$. Most importantly, the response value RES to the challenge $AUTN$ does not depend on $SNid$. Hence, UE never makes a statement about its view of $SNid$, which is why we will not achieve agreement on K_{ASME}. The EPS-AKA specification [4] addresses this issue in Section 6.1.1 as follows:

> *"SNid binding implicitly authenticates the serving network's identity when the derived keys from K_{ASME} are successfully used."*

In order to be able to discuss security properties with respect to the event in which derived keys from K_{ASME} are used successfully, we would need to consider the messages that are exchanged *after* the actual authentication protocol. Therefore, we extend the protocol by a simplified version of the so-called *NAS security mode command procedure* (described in Section 7.2.4.4 of [4]), which is mandatory in 4G.

We define two confirmation messages $confirmSEAF$ and $confirmUE$:

$$confirmSEAF := MAC_{confirm}\left(KDF_{confirm}(K_{ASME}), \text{`}confirmSEAF\text{`}\right)$$
$$confirmUE := MAC_{confirm}\left(KDF_{confirm}(K_{ASME}), \text{`}confirmUE\text{`}\right)$$

Note that $MAC_{confirm}$ is a new MAC-function and $KDF_{confirm}$ is a new key derivation function. The extended protocol runs as follows.

Protocol 5.3 (EPS-AKA v2)

1. $SEAF \rightarrow UE : \text{`}IdentityRequest\text{`}$
2. $UE \rightarrow SEAF : IMSI$
3. $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle S, IMSI, SNid \rangle$
4. $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle S, AV \rangle$
5. $SEAF \rightarrow UE : \langle RAND, AUTN \rangle$
6. $UE \rightarrow SEAF : RES$
7. $SEAF \rightarrow UE : confirmSEAF$
8. $UE \rightarrow SEAF : confirmUE$

5.1.1 Building a Model

We make the following modeling decisions:

- f_5 is the constant zero function. This is a valid instantiation according to the specification. In case the sequence number SQN is generated by a time-based scheme, it makes no sense to try to protect it with a (non-zero) anonymity key, as the time is publicly known anyway.
- We model the sequence number SQN as a fresh nonce and assume that UE is able to make sure it never accepts the same SQN twice. This ignores all kinds of synchronization issues, and assumes infinite state.

- Likewise, we model the Session-Id AVP S as a fresh nonce. In contrast to the sequence number, however, this is a completely realistic model as $SEAF$ is actually able to generate locally unique identifiers. For example, [21] proposes to simply use a counter which is increased for every new request message.
- We model the secure channel between $SEAF$ and HSS as described in Section 3.4 on page 27.

  Furthermore, we allow this key to be revealed by the adversary. As we do not consider the key to be long-term, the notion of Perfect Forward Secrecy is not affected by such a reveal.

This results in the TAMARIN theories `EPSAKAv1` and `EPSAKAv2` for Protocol 5.2 and Protocol 5.3, respectively.

5.1.2 Security Properties (no confirmation messages)

We start by discussing the security properties of EPS-AKA v1 (Protocol 5.2), which does not include the confirmation messages.

It is easy to see that Perfect Forward Secrecy cannot possibly be provided. This is because when the long-term key K shared between UE and HSS is revealed by the adversary, she can immediately recover K_{ASME}, using only K and the message $\langle RAND, AUTN \rangle$ which has been sent from $SEAF$ to UE in plain text.

In order to achieve Perfect Forward Secrecy, we would recommend running a Diffie-Hellman key exchange. It is possible, however, that this is considered to be computationally too expensive in the context of mobile devices.

In the following, we discuss the other security properties from the perspective of all three roles, namely UE, $SEAF$, and HSS. Recall that we do not discuss authentication properties between HSS and $SEAF$, because we assume they have already set up a secure channel.

- UE:
  - Secrecy of K_{ASME} is proved automatically.
  - Aliveness of $SEAF$ cannot be provided, since UE receives only the messages $'IdentityRequest'$ and $\langle RAND, AUTN \rangle$ from $SEAF$, and neither of those make a statement about the identity $SNid$ of $SEAF$. TAMARIN automatically finds an attack. For reasons of simplicity, we present a shorter version of this attack in Attack Scenario 5.1 on the next page.
  - Weak agreement with HSS is proved automatically. Non-injective agreement with HSS on K_{ASME}, however, can be easily violated.

The adversary only has to send the $IMSI$ to a different serving network ($SEAF$), and forward the resulting authentication vector AV to UE. This succeeds, violating non-injective agreement on K_{ASME}, because K_{ASME} depends on $SNid$ and UE has no way of verifying the $SNid$ that HSS used is the same as its own $SNid$ (recall that UE does not even have aliveness guarantee of $SEAF$). Attack Scenario 5.1 describes this attack in more detail.

- $SEAF$:
  - Secrecy of K_{ASME} is proved automatically.
  - Weak agreement with UE cannot be provided, because $SEAF$ does not receive any verifiable statement about UE's view of $SNid$ (analogously to agreement from UE's perspective). Once again, we refer to Attack Scenario 5.1 to illustrate this issue.

    Interestingly, aliveness of UE is still proved automatically. The reason is in essence that HSS is able to verify aliveness of UE (see below), and only if it has successfully done so it will respond with an authentication vector. As all communication between HSS and $SEAF$ runs over a secure channel, $SEAF$ is able to verify aliveness of UE.
- HSS:
  - Secrecy of K_{ASME} is proved automatically.
  - Aliveness of UE is provided, because only HSS and UE know the $IMSI$, and only UE sends it. Therefore, if HSS sees the $IMSI$ of an issued $USIM$, it can be sure that the protocol has been executed on a UE once before. Moreover, UE executes the protocol only with the correct HSS (stored in the $USIM$), which is why weak agreement of HSS with UE is satisfied and proved automatically, too. Note that weak agreement as specified in Definition 2.39 only requires that the peer has been running the protocol at least once before (with the correct HSS). In particular, a single execution of UE may suffice to satisfy those properties for HSS, even if HSS executes the protocol much later than UE and is actually only exchanging messages with the adversary.

    However, non-injective agreement with UE on K_{ASME} can be violated. This is no surprise, since HSS has no way of verifying what $SNid$ was used by UE. As K_{ASME} depends on $SNid$, non-injective agreement is trivially violated (see Attack Scenario 5.1).

Attack Scenario 5.1 *Let $SNid_1$ and $SNid_2$ be the (distinct) identifiers of $SEAF_1$ and $SEAF_2$, respectively.*

1. $SEAF_1 \rightarrow Adv(UE) : \text{`}IdentityRequest\text{`}$
   1. $Adv(SEAF_2) \rightarrow UE : \text{`}IdentityRequest\text{`}$
   2. $UE \rightarrow Adv(SEAF_2) : IMSI$
2. $Adv(UE) \rightarrow SEAF_1 : IMSI$
3. $SEAF_1 \bullet\!\!\rightarrow\!\!\bullet HSS : \langle S, IMSI, SNid_1 \rangle$
4. $HSS \bullet\!\!\rightarrow\!\!\bullet SEAF_1 : \langle S, AV \rangle$
   5. $Adv(SEAF_2) \rightarrow UE : \langle RAND, AUTN \rangle$
   6. $UE \rightarrow Adv(SEAF_2) : RES$
6. $Adv(UE) \rightarrow SEAF_1 : RES$

Note that $SEAF_2$ is not running an instance of the protocol. Hence, aliveness of $SEAF_2$ is violated for UE.

Moreover, weak agreement of $SEAF_1$ with UE is violated, because UE is apparently executing the protocol with $SEAF_2$ as its peer.

Furthermore, as K_{ASME} depends on SNid, this attack violates non-injective agreement on K_{ASME} for UE with HSS and vice versa.

Tables 5.1 and 5.2 summarize all discussed security properties.

	Perspective		
	UE	$SEAF$	HSS
Secrecy K_{ASME}	✓	✓	✓
PFS K_{ASME}	×	×	×

Table 5.1: Secrecy properties of theory `EPSAKAv1`, modeling EPS-AKA v1 (Protocol 5.2).

	Perspective			
	$UE \mapsto SEAF$	$UE \mapsto HSS$	$SEAF \mapsto UE$	$HSS \mapsto UE$
Aliveness	×	✓	✓	✓
Weak agreement	×	✓	×	✓
Non-injective agreement	×	×	×	×
Injective agreement	×	×	×	×

Table 5.2: Authentication properties of theory `EPSAKAv1`, modeling EPS-AKA v1 (Protocol 5.2). All properties are with respect to K_{ASME}.

5.1.3 Security Properties (with confirmation messages)

Next, we discuss the security properties of EPS-AKA v2 (Protocol 5.3), which includes the confirmation messages.

It is easy to see that Perfect Forward Secrecy cannot possibly be provided. The reason is the same as for Protocol 5.2 (see Section 5.1.2).

In the following, we examine the other security properties from the perspective of all three roles UE, $SEAF$ and HSS. Recall that we do not discuss authentication properties between HSS and $SEAF$, because we assume they have already set up a secure channel.

- UE: Secrecy of K_{ASME} and injective agreement with both $SEAF$ and HSS on K_{ASME} are proved automatically.
- $SEAF$: Secrecy of K_{ASME} and injective agreement with UE on K_{ASME} are proved automatically.
- HSS:
  - Secrecy of K_{ASME} is proved automatically.
  - Weak agreement with UE is proved automatically. However, non-injective agreement with UE on K_{ASME} can be violated and a corresponding attack is found. The reason is the same as in the version without confirmation messages (see Section 5.1.2). This is no surprise, since the two versions with and without confirmation messages are equivalent from the perspective of HSS.

Tables 5.3 and 5.4 summarize all discussed security properties.

	Perspective		
	UE	$SEAF$	HSS
Secrecy K_{ASME}	✓	✓	✓
PFS K_{ASME}	×	×	×

Table 5.3: Secrecy properties of theory `EPSAKAv2`, modeling EPS-AKA v2 (Protocol 5.3).

	Perspective			
	$UE \mapsto SEAF$	$UE \mapsto HSS$	$SEAF \mapsto UE$	$HSS \mapsto UE$
Aliveness	✓	✓	✓	✓
Weak agreement	✓	✓	✓	✓
Non-injective agreement	✓	✓	✓	×
Injective agreement	✓	✓	✓	×

Table 5.4: Authentication properties of theory `EPSAKAv2`, modeling EPS-AKA v2 (Protocol 5.3). All properties are with respect to K_{ASME}.

5.2 EPS-AKA*

EPS-AKA* is an authentication protocol proposed for 5G in [6] and [5]. It makes a few modifications to EPS-AKA.

Up to now, roaming agreements have been based solely on trust. This means that if a roaming partner claims that a UE has visited a specific serving network, the home network operator has no way of verifying such a claim. EPS-AKA* tries to address this very issue (see key issue #2.11 of [6], called *"Increasing home control in roaming situations"*). The main goal is to provide more guarantees to the home network HSS in order to reduce the trust that HSS needs to put in the serving network in a roaming situation.

We define the following values:

- $MAC := f_1(K, AMF, SQN, RAND)$
- $XRES := (XRES_1, XRES_2) := RES := (RES_1, RES_2) := f_2(K, RAND)$, i.e., $XRES$ and RES are simply other names for $(XRES_1, XRES_2)$ and (RES_1, RES_2), respectively.
- $CK := f_3(K, RAND)$, also called *confidentiality key*
- $IK := f_4(K, RAND)$, also called *integrity key*
- $AK := f_5(K, RAND)$, also called *anonymity key*
- $AUTN := \langle SQN \oplus AK, AMF, MAC \rangle$
- $K_{ASME} := KDF(CK, IK, SNid, SQN \oplus AK)$
- $AV^* := \langle RAND, XRES_1, K_{ASME}, AUTN \rangle$
- $confirmSEAF := MAC_{confirm}\left(KDF_{confirm}(K_{ASME}), \text{`}confirmSEAF\text{`}\right)$
- $confirmUE := MAC_{confirm}\left(KDF_{confirm}(K_{ASME}), \text{`}confirmUE\text{`}\right)$

EPS-AKA* runs as follows.

Protocol 5.4 (EPS-AKA*)

1. $SEAF \rightarrow UE : \text{`}IdentityRequest\text{`}$
2. $UE \rightarrow SEAF : IMSI$
3. $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle S, IMSI, SNid \rangle$
4. $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle S, AV^* \rangle$
5. $SEAF \rightarrow UE : \langle RAND, AUTN \rangle$
6. $UE \rightarrow SEAF : RES$
7. $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle \text{`}AuthConfirmation\text{`}, IMSI, SNid, RES \rangle$
8. $SEAF \rightarrow UE : confirmSEAF$
9. $UE \rightarrow SEAF : confirmUE$

Note the following:

- The value RES from EPS-AKA has been split up into two halves. Moreover, from the perspective of UE nothing has changed compared to EPS-AKA. This design choice has been primarily motivated by backwards compatibility. When $SEAF$ receives RES, it can only verify the first half of it, because the authentication vector AV^* contains only $XRES_1$. However, HSS is able to verify both RES_1 and RES_2.
- The string '$AuthConfirmation$' in message 7 is a tag which models the *Command Code* from the Diameter header. It facilitates automated proving of some of the security properties.
- Just as in EPS-AKA, we extended the raw protocol in two ways. First, we added the Session-Id AVP S from the Diameter packet to the request from $SEAF$ and the corresponding response from HSS. Then, we also appended the two confirmation messages, modeling a lightweight variant of the *Security Mode Command Procedure* (SMC), ignoring details such as security capability information for UE and choice of cryptographic algorithm. This remedies the exact same issues as in EPS-AKA v1 (Protocol 5.2), which we confirmed to be present when omitting the confirmation messages. Note that the SMC is not yet fully specified for 5G (see Section 5.2.4.7.2.4 of [6]). We discuss this issue in more detail in Section 5.2.2, where we try to generalize the confirmation messages.

Building on the Tamarin theory `EPSAKAv2` for EPS-AKA, we develop a new Tamarin theory for EPS-AKA*, called `EPSAKA_STAR`.

5.2.1 Security Properties

For the same reasons as in EPS-AKA, described in Section 5.1.2 on page 45, EPS-AKA* cannot provide Perfect Forward Secrecy of K_{ASME}.

In the following, we discuss the other security properties from the perspective of all three roles UE, $SEAF$, and HSS.

- UE: Secrecy of K_{ASME} and injective agreement with both $SEAF$ and HSS on K_{ASME} are proved automatically, just as in EPS-AKA.
- $SEAF$: Secrecy of K_{ASME} and injective agreement with UE on K_{ASME} are proved automatically, just as in EPS-AKA.
- HSS:
  - Secrecy of K_{ASME} is proved automatically.
  - Weak agreement of HSS with UE is satisfied (and proved automatically) for the same reasons as in EPS-AKA.

Non-injective agreement with UE on K_{ASME} fails. This is no surprise, since K_{ASME} depends on $SNid$ and the additional message makes no statement about the value of $SNid$ that is used by UE.

However, the additional message increases the guarantees for HSS. In particular, it allows HSS to agree injectively with UE on $RAND$.

Tables 5.5 and 5.6 summarize all discussed security properties.

	Perspective		
	UE	$SEAF$	HSS
Secrecy K_{ASME}	✓	✓	✓
PFS K_{ASME}	×	×	×

Table 5.5: Secrecy properties of theory `EPSAKA_STAR`, modeling EPS-AKA* (Protocol 5.4).

	Perspective			
	$UE \mapsto SEAF$	$UE \mapsto HSS$	$SEAF \mapsto UE$	$HSS \mapsto UE$
Aliveness	✓	✓	✓	✓
Weak agreement	✓	✓	✓	✓
Non-injective agreement	✓	✓	✓	×
Injective agreement	✓	✓	✓	×
Injective agreement ($RAND$)	✓	✓	✓	✓

Table 5.6: Authentication properties of theory `EPSAKA_STAR`, modeling EPS-AKA* (Protocol 5.4). All properties are with respect to K_{ASME}, unless otherwise stated.

At this point, we take a step back and reevaluate the security requirements of the protocol. Recall that the purpose of the additional message introduced in EPS-AKA* is to remedy Key issue #2.11 that is described in [6]. We briefly describe the two explicitly mentioned security threats, omitting some of the details.

- **Threat 1.** A fraudulent serving network that has a roaming agreement with the home network claims charges for an allegedly visited subscriber, even though the subscriber never completed authentication in the visited the serving network.
- **Threat 2.** A fraudulent serving network that has a roaming agreement with the home network claims charges for a subscriber, who successfully completed authentication, but never attached to the network or did not incur the claimed amount of traffic in the visited network.

It is obvious that Threat 2 cannot be addressed by a usual authentication protocol, because it would require the measurement of incurred traffic during the complete communication session.

Threat 1, however, could be solved by a conventional authentication protocol. Essentially, it suffices for HSS to have injective agreement on $SNid$. Unfortunately, EPS-AKA* does not guarantee this.

Moreover, the attack described in Threat 1 can be mounted in EPS-AKA* as follows. A fraudulent serving network simply requests an authentication vector for an arbitrary UE. It then masquerades as a different serving network (potentially even from a different country) and asks[2] the UE to solve the challenge bound to the authentication vector. UE's response RES is forwarded to HSS, concluding the attack. Attack Scenario 5.2 describes this attack in more detail.

Attack Scenario 5.2 *Let $SNid_1$ and $SNid_2$ be the (distinct) identifiers of $SEAF_1$ and $SEAF_2$, respectively. Furthermore, we assume $SEAF_1$ to be a fraudulent serving network having a roaming agreement with the home network HSS.*

1. $SEAF_1 \rightarrow UE : \text{`}IdentityRequest\text{`}$
2. $UE \rightarrow SEAF_1 : IMSI$
3. $SEAF_1 \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle S, IMSI, SNid_1 \rangle$
4. $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF_1 : \langle S, AV^* \rangle$
5. $Adv(SEAF_2) \rightarrow UE : \langle RAND, AUTN \rangle$
6. $UE \rightarrow E(SEAF_2) : RES$
7. $SEAF_1 \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle \text{`}AuthConfirmation\text{`}, IMSI, SNid_1, RES \rangle$

As K_{ASME} depends on SNid, this attack violates non-injective agreement of HSS with UE.

The core of the issue is that, as explained before, UE never makes a statement about its view on $SNid$. It seems that it is not possible to properly address Threat 1 without changing UE's role in the protocol.

Note, however, that the home network HSS achieves injective agreement on the value $RAND$. Since $RAND$ is a fresh value that HSS generates for each new authentication vector, HSS has the guarantee that UE attached to a serving network by solving the challenge that corresponds to $RAND$. Hence, all attacks can be prevented in which a serving network illicitly claims that a UE successfully attached to it even though UE has not provided a correct response for the challenge that HSS generated. EPS-AKA, in contrast, is trivially susceptible to those attacks, since HSS receives no message after sending out the challenge.

[2]Note that this requires the fraudulent serving network to be in radio-range of UE.

5.2.2 Generalizing the Confirmation Messages

The messages following the authentication protocol (e.g., EPS-AKA*) are not fully specified for 5G yet (see Section 5.2.4.7.2.4 of [6]). Most likely, it will be a *Security Mode Command procedure* (SMC) as proposed in Solution #1.31 of [5].

Recall that we model a simplified variant of such an SMC with a very specific message format (see Protocol 5.4). However, we believe that a large family of alternatives can be used to achieve the exact same security properties as shown in Section 5.2.1. More specifically, we believe that the following holds:

Claim 5.1 *If messages 1-7 of Protocol 5.4 are extended by additional messages, such that secrecy of K_{ASME} still holds for UE and $SEAF$, then:*

- *Injective agreement on K_{ASME} for UE with both $SEAF$ and HSS is established the moment UE receives a message containing the term $m_1 = MAC(k, inner_1)$, where m_1 has never been sent out by UE before, $inner_1$ is a value known to UE, and $k = K_{ASME}$ or $k = KDF(K_{ASME})$ for some key derivation function KDF.*
- *Injective agreement on K_{ASME} for $SEAF$ with UE is established as soon as $SEAF$ receives a message containing the term $m_2 = MAC(k, inner_2)$, where m_2 has never been sent out by $SEAF$ before, $inner_2$ is a value known to $SEAF$, and $k = K_{ASME}$ or $k = KDF(K_{ASME})$ for some key derivation function KDF.*

The condition that *secrecy* of K_{ASME} must still hold is essential, because in general, adding new messages to a protocol is dangerous, especially if they involve the secret key material. For example, one could extend the protocol by a message that sends out the key K_{ASME} in plain text.

Moreover, it is crucial for both UE and $SEAF$ not to have sent the message m_1 and m_2 before, because it would make them susceptible to a reflection attack. Furthermore, UE and $SEAF$ need to know the values $inner_1$ and $inner_2$, respectively, in order to be able to verify the MAC.

It seems that it is possible to further generalize the confirmation messages by allowing messages of the form $m = \{|inner|\}_k$. However, this would require an authenticated encryption scheme.

We believe that most practical use cases that occur when using the agreed-upon key material from EPS-AKA and EPS-AKA* are an instance of the proposed family of alternative protocol extensions. For example, the *NAS Security Mode Command procedure* (specified in Section 7.2.4.4 of [4]) that is used in 4G as well as Solution #1.31 of [6] that is proposed for 5G satisfy all conditions of Claim 5.1.

5.3 Conclusion

We analyzed the security of EPS-AKA as used in 4G, as well as its 5G successor EPS-AKA*. The main difficulty with both protocols was that UE never makes a cryptographically verifiable statement about its view on the identity $SNid$ of the serving network. This has the following effects:

- Since the key K_{ASME} does depend on $SNid$, agreement on K_{ASME} from UE as well as with UE is not possible without additional confirmation messages, modeling the Security Mode Command procedure (SMC). This is suboptimal especially for 5G, where SMC is not fully specified yet.
- While EPS-AKA* successfully increases the guarantees for the home network, some attacks cannot be prevented. This is exactly because agreement with UE on $SNid$ fails.

We believe a good solution would be for UE to provide a cryptographically verifiable statement on $SNid$. For example, it would suffice to make $XRES$ (the response to the challenge) additionally depend on $SNid$. However, the deployment costs of such a change are probably considered too high. The current proposal, EPS-AKA*, has the advantage of being equivalent to EPS-AKA from the perspective of UE, resulting in full backwards-compatibility for older devices.

Chapter 6

EAP-AKA'

This chapter addresses EAP-AKA' which is specified in [23] as well as in Section 6.1.3.1 of [5]. It makes use of the EAP framework [7, 8] and it builds on EAP-AKA [11]. Moreover, Solution #2.9 of [6] explains how the EAP framework is to be deployed in a 5G network.

Recall the basic 5G network structure from Section 2.1 as well as the corresponding role and agent names described in Section 3.6. Our protocol model has three essential roles: UE, $SEAF$ and HSS.

- UE is the user equipment (e.g., a smartphone). It contains a $USIM$ with a long-term key K that is shared with the home network.
- $SEAF$ resides in the serving network. The serving network's identity name is denoted by both $SEAF$ and $SNid$.
- HSS is the home network in which the $AUSF$ and $ARPF$ reside in. Additionally, it knows the long-term key K that is shared with UE.

Moreover, all communication between $SEAF$ and HSS is envisaged to be Diameter-based, and runs over IPSec or TLS.

EAP-AKA' is an authentication and key agreement protocol. We are interested in the following security properties, all with respect to the master key MK the protocol tries to agree on:

- Secrecy for $UE, HSS, SEAF$.
- Perfect Forward Secrecy for $UE, SEAF, HSS$ with respect to the permanent key K between UE and HSS.
- Injective agreement for UE with $SEAF$ and UE with HSS.
- Injective agreement for $SEAF$ with UE and HSS with UE.

Note that we do not discuss authentication properties between HSS and $SEAF$, because we assume they have already set up a secure channel.

In case injective agreement does not hold, we try to find the strongest authentication property that holds (if any).

Let f_1, f_2 be MAC-functions, f_3, f_4, f_5, KDF key generating functions, and PRF a pseudorandom function. Moreover, the specification allows f_5 to be the constant zero function.

In the following, we use the symbol $^\circ$ instead of the prime symbol $'$ to avoid confusion with the string delimiter. We define the following values:

- $MAC := f_1(K, AMF, SQN, RAND)$
- $XRES := RES := f_2(K, RAND)$, where $XRES$ is simply another name for RES.
- $CK := f_3(K, RAND)$, also called *confidentiality key*
- $IK := f_4(K, RAND)$, also called *integrity key*
- $AK := f_5(K, RAND)$, also called *anonymity key*
- $AUTN := \langle SQN \oplus AK, AMF, MAC \rangle$
- $AV := \langle RAND, AUTN, SNid \rangle$
- $CK^\circ := KDF(\text{`}CK^\circ\text{'}, SNid, SQN \oplus AK, CK, IK)$
- $IK^\circ := KDF(\text{`}IK^\circ\text{'}, SNid, SQN \oplus AK, CK, IK)$
- $K_{encr} := PRF(\text{`}Kencr\text{'}, IK^\circ, CK^\circ, \text{`}EAPAKA^\circ\text{'}, IMSI)$, also called *encryption key*
- $K_{aut} := PRF(\text{`}Kaut\text{'}, IK^\circ, CK^\circ, \text{`}EAPAKA^\circ\text{'}, IMSI)$, also called *authentication key*
- $K_{re} := PRF(\text{`}Kre\text{'}, IK^\circ, CK^\circ, \text{`}EAPAKA^\circ\text{'}, IMSI)$, also called *reauthentication key*
- $MSK := PRF(\text{`}MSK\text{'}, IK^\circ, CK^\circ, \text{`}EAPAKA^\circ\text{'}, IMSI)$, also called *Master Session Key*
- $EMSK := PRF(\text{`}EMSK\text{'}, IK^\circ, CK^\circ, \text{`}EAPAKA^\circ\text{'}, IMSI)$, also called *Extended Master Session Key*.
- $MK := \langle K_{encr}, K_{auth}, K_{re}, MSK, EMSK \rangle$, also called *Master Key*

Note that $\oplus$ denotes the (bitwise) exclusive-or (XOR) operator.

In the following, we present a version of EAP-AKA'.

Protocol 6.1 (EAP-AKA' v1)

1. $SEAF \rightarrow UE : \text{`}IdentityRequest\text{`}$
2. $UE \rightarrow SEAF : IMSI$
3. $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle \text{`}3\text{`}, S, IMSI, SNid \rangle$
4. $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle \text{`}4\text{`}, S, \langle P_{HSS}, AV \rangle, HMAC(k_{aut}, \langle P_{HSS}, AV \rangle) \rangle$
5. $SEAF \rightarrow UE : \langle \langle P_{HSS}, AV \rangle, HMAC(k_{aut}, \langle P_{HSS}, AV \rangle) \rangle$
6. $UE \rightarrow SEAF : \langle \langle P_{UE}, RES \rangle, HMAC(k_{aut}, \langle P_{UE}, RES \rangle) \rangle$
7. $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle \text{`}7\text{`}, S, \langle P_{UE}, RES \rangle, HMAC(k_{aut}, \langle P_{UE}, RES \rangle) \rangle$
8.* $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle \text{`}8\text{`}, S, \text{`}RequestSuccess\text{`}, HMAC(k_{aut}, \text{`}RequestSuccess\text{`}) \rangle$
9.* $SEAF \rightarrow UE : \langle \text{`}RequestSuccess\text{`}, HMAC(k_{aut}, \text{`}RequestSuccess\text{`}) \rangle$
10.* $UE \rightarrow SEAF : \langle \text{`}ResponseSuccess\text{`}, HMAC(k_{aut}, \text{`}ResponseSuccess\text{`}) \rangle$
11.* $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle \text{`}11\text{`}, S, \text{`}ResponseSuccess\text{`}, HMAC(k_{aut}, \text{`}ResponseSuccess\text{`}) \rangle$
12. $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle \text{`}12\text{`}, S, \text{`}EAPSuccess\text{`}, MK \rangle$
13. $SEAF \rightarrow UE : \text{`}EAPSuccess\text{`}$

Note the following:

- $RAND$ is a fresh random value generated by HSS.
- SQN is a fresh sequence number generated by HSS. In this chapter, we treat the sequence number as a fresh value that is generated by HSS.

  A precise model for the sequence number and its corresponding re-synchronization mechanism is discussed in Chapter 7.
- $SNid$ is the identifier of the serving network which is represented by the role $SEAF$ in our model.
- P_{HSS} and P_{UE} are the so-called *protection bits*. Messages 8-11 (marked with *) are optional and sent if and only if both P_{HSS} and P_{UE} are set to one. Moreover, P_{UE} must be zero if P_{HSS} is zero.
- S is the session identifier from the Diameter header. It is freshly generated by $SEAF$ and helps to bind the following message to the same session.
- All messages between HSS and $SEAF$ are tagged with their message number, modeling a reasonable choice of Diameter AVPs.
- The Authentication Management Field (AMF) is a 16-bit value, where the least significant bit is the AMF separation bit. This is used to distinguish authentication vectors for EAP-AKA from those for EAP-AKA'. Bits 1-7 are reserved and 8-15 are for proprietary purposes.

- Actually, the identification process is more involved. For example, UE can identify itself via a pseudonym that can be mapped to an $IMSI$ (if the mapping fails, HSS will request the full identity via an additional message). Additionally, there are different encodings to transmit the identity. We omit these options for reasons of simplicity, because the first message from UE is unprotected anyway.
- Some of the EAP attributes, like AT_{KDF} (for negotiating the key derivation function) have been omitted.
- All messages that serve a purely informational purpose and are not cryptographically protected, like error messages, have been omitted.
- In this initial version, we ignore the *Fast Re-authentication* mechanism and postpone its discussion to Section 6.4.

It is striking that there are no confirmation messages between the serving network $SEAF$ and UE after $SEAF$ has successfully received the master key MK from the home network. As in the analysis of EPS-AKA* in Chapter 5, we additionally discuss the security properties of EAP-AKA' (Protocol 6.1) extended by a lightweight variant of the so-called *Security Mode Command Procedure* (SMC). Said SMC is mandatory and always executed after the authentication protocol (in this case EAP-AKA'). Note that the SMC is not fully specified for 5G yet, which is why we base our model on Solution #1.31 and Solution #1.32 of [6]. Moreover, we model a simplified variant, ignoring details such as security capability information for UE and choice of cryptographic algorithm. For a more thorough discussion of the SMC in 5G, we refer to Section 5.2.2 on page 53.

We introduce two additional messages $confirmSEAF$ and $confirmUE$ that are defined as follows.

$$\begin{aligned} K_{NASMM} &:= KDF_{NASMM}(MK) \\ K_{NASMMint} &:= KDF_{NASMMint}(K_{NASMM}) \\ confirmSEAF &:= \langle \text{`}NAS\_SM\_COMMAND\text{`}, \\ &\qquad HMAC(K_{NASMMint}, \text{`}NAS\_SM\_COMMAND\text{`}) \rangle \\ confirmUE &:= \langle \text{`}NAS\_SM\_COMPLETE\text{`}, \\ &\qquad HMAC(K_{NASMMint}, \text{`}NAS\_SM\_COMPLETE\text{`}) \rangle \end{aligned}$$

Note that KDF_{NASMM} as well as $K_{NASMMint}$ are new key derivation functions.

The extended protocol runs as follows.

Protocol 6.2 (EAP-AKA' v2)

1. $SEAF \rightarrow UE : \text{`}IdentityRequest\text{`}$
2. $UE \rightarrow SEAF : IMSI$
3. $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle \text{`}3\text{`}, S, IMSI, SNid \rangle$
4. $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle \text{`}4\text{`}, S, \langle P_{HSS}, AV \rangle, HMAC(k_{aut}, \langle P_{HSS}, AV \rangle) \rangle$
5. $SEAF \rightarrow UE : \langle \langle P_{HSS}, AV \rangle, HMAC(k_{aut}, \langle P_{HSS}, AV \rangle) \rangle$
6. $UE \rightarrow SEAF : \langle \langle P_{UE}, RES \rangle, HMAC(k_{aut}, \langle P_{UE}, RES \rangle) \rangle$
7. $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle \text{`}7\text{`}, S, \langle P_{UE}, RES \rangle, HMAC(k_{aut}, \langle P_{UE}, RES \rangle) \rangle$
8.* $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle \text{`}8\text{`}, S, \text{`}RequestSuccess\text{`}, HMAC(k_{aut}, \text{`}RequestSuccess\text{`}) \rangle$
9.* $SEAF \rightarrow UE : \langle \text{`}RequestSuccess\text{`}, HMAC(k_{aut}, \text{`}RequestSuccess\text{`}) \rangle$
10.* $UE \rightarrow SEAF : \langle \text{`}ResponseSuccess\text{`}, HMAC(k_{aut}, \text{`}ResponseSuccess\text{`}) \rangle$
11.* $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle \text{`}11\text{`}, S, \text{`}ResponseSuccess\text{`}, HMAC(k_{aut}, \text{`}ResponseSuccess\text{`}) \rangle$
12. $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle \text{`}12\text{`}, S, \text{`}EAPSuccess\text{`}, MK \rangle$
13. $SEAF \rightarrow UE : \text{`}EAPSuccess\text{`}$
14. $SEAF \rightarrow UE : confirmSEAF$
15. $UE \rightarrow SEAF : confirmUE$

Note that messages 1 to 13 are exactly the same as in EAP-AKA' v1 (Protocol 6.1).

Before presenting our model for EAP-AKA', we briefly point out the high-level differences between EAP-AKA' and EPS-AKA*:

- Most importantly, in EAP-AKA' the $SEAF$ is acting essentially as a simple proxy, until HSS has authenticated UE and sends the agreed-upon master key MK to $SEAF$. As a consequence, all messages between $SEAF$ and HSS except message 12 would not need to be transmitted over a confidential channel. This is in contrast to EPS-AKA*, where $SEAF$ receives the secret key K_{ASME} together with the authentication vector and $XRES$ from HSS, and verifies UE's response RES to the challenge itself.

- Furthermore, the key hierarchy for EAP-AKA' is fundamentally different to EPS-AKA*. In particular, EAP-AKA' binds the name of the serving network $SNid$ to the keys CK° and IK°.

6.1 Building a Model

We make the following modeling decisions:

- The roles UE and HSS can be instantiated with an arbitrary choice of their protection bit P_{UE} and P_{HSS}, respectively. Note that the choice is not bound to a particular agent name, but only to a concrete instance of a role. In particular, it is possible for an agent UE to concurrently execute two instances of the protocol (in the role UE) with different protection bits.
- f_5 is the constant zero function. This is a valid instantiation according to the specification. In case the sequence number SQN is generated by a time-based scheme, it makes no sense to try to protect it with a (non-zero) anonymity key, as the time is publicly known anyway.
- We model the sequence number SQN as a fresh nonce and assume that the UE is able to make sure it never accepts the same SQN twice. This ignores all kinds of synchronization issues, and assumes infinite state.
- Likewise, we model the Session identifier S as a fresh nonce generated by $SEAF$.
- We model the secure channel between $SEAF$ and HSS as described in Section 3.4 on page 27.

This results in the Tamarin theories `EAPAKAPRIMEv1` and `EAPAKAPRIMEv2` for Protocol 6.1 and Protocol 6.2, respectively.

6.2 Security Properties (no confirmation messages)

We start by discussing the security properties of EAP-AKA' v1 (Protocol 6.1), which does not include the confirmation messages.

It is easy to see that Perfect Forward Secrecy cannot possibly be provided. This is because when the long-term key K shared between UE and HSS is revealed by the adversary, she can immediately recover MK, using only K and the message $\langle P_{HSS}, AV \rangle$ which has been sent from $SEAF$ to UE in plain text.

In order to achieve Perfect Forward Secrecy, we would recommend running a Diffie-Hellman key exchange. It is possible, however, that this is considered to be computationally too expensive in the context of mobile devices.

In the following, we discuss the other security properties from the perspective of all three roles, namely UE, $SEAF$, and HSS. Recall that we do not discuss authentication properties between HSS and $SEAF$, because we assume they have already set up a secure channel.

- *UE*:
  - Secrecy of *MK* is proved automatically.
  - Injective agreement with *HSS* on *MK* is proved automatically.
  - Weak agreement with *SEAF* is proved automatically. However, non-injective agreement with *SEAF* on *MK* can be violated, and an attack is found automatically. This is no surprise, since *SEAF* receives the master key *MK* only at the very end of the protocol (message 12), making it impossible for *UE* to have the guarantee that *SEAF* has actually received the (correct) *MK*. Such a guarantee, however, would hold if we would assume an uninterruptible channel between the serving network and the home network. In practice, this may be a reasonable assumption. Our model, however, allows the adversary to interrupt the channel between *SEAF* and *HSS*.
- *SEAF*:
  - Secrecy of *MK* is proved automatically, making use of the additional lemma `helper_secrecy_SEAF` that connects a secrecy claim of *SEAF* to the corresponding secrecy claim of *HSS*. The helping lemma itself is proved automatically, too. Moreover, the proof reuses lemma `secrecy_HSS` (see below).
  - Non-injective agreement with *UE* on *MK* is proved automatically, using the lemma `injectiveagreementHSS_UE` (see below) as well as the two helping lemmas `helper_authentication_SEAF1` and `helper_authentication_SEAF2` (both of which are proved automatically). `helper_authentication_SEAF1` connects every commit from *SEAF* with *UE* to a corresponding commit from *HSS* with *UE*, while `helper_authentication_SEAF2` connects every commit from *HSS* with *UE* to a corresponding commit from *UE* with *SEAF*.

    We prove injective agreement with *UE* on *MK*, too. The proof, however, is rather difficult. In essence, the idea is to specify an additional lemma `helper_authentication_SEAF5` that connects two different commits of *SEAF* with *UE* on the same master key *MK* to two distinct commits of *HSS* (on the same master key). Reusing a strengthened form of `secrecy_SEAF` (which expresses secrecy of *MK* for *SEAF*), said lemma can then be proved manually. As an alternative, we provide an oracle[1] which guides the

[1] See Section 2.4 on page 21.

proof search. Using this oracle, TAMARIN successfully proves the lemma `helper_authentication_SEAF5` within minutes.

Reusing the mentioned helping lemmas, injective agreement with UE on MK is proved automatically.

- HSS:
  - Secrecy of MK is proved automatically.
  - Injective agreement with UE on MK is proved automatically.

It is striking that injective agreement on MK is established between UE and HSS (in both directions), even though the protected confirmation messages (messages 8-11) are optional. Hence, those messages are not needed to achieve these security properties. It would be interesting to investigate whether those messages provide any additional guarantees. If not, it would be best to remove them from the protocol altogether.

Tables 6.1 and 6.2 summarize all discussed security properties.

	Perspective		
	UE	$SEAF$	HSS
Secrecy MK	✓	✓	✓
PFS MK	×	×	×

Table 6.1: Secrecy properties of theory `EAPAKAPRIMEv1`, modeling EAP-AKA' v1 (Protocol 6.1).

	Perspective			
	$UE \mapsto SEAF$	$UE \mapsto HSS$	$SEAF \mapsto UE$	$HSS \mapsto UE$
Aliveness	✓	✓	✓	✓
Weak agreement	✓	✓	✓	✓
Non-injective agreement	×	✓	✓	✓
Injective agreement	×	✓	✓	✓

Table 6.2: Authentication properties of theory `EAPAKAPRIMEv1`, modeling EAP-AKA' v1 (Protocol 6.1). All properties are with respect to MK.

6.3 Security Properties (with confirmation messages)

Next, we discuss the security properties of EAP-AKA' v2 (Protocol 6.2), which includes the confirmation messages. Naturally, we hope to be able to prove the same security properties as for EAP-AKA' v1 and additionally injective agreement from UE with $SEAF$ on the master key MK.

It is easy to see that Perfect Forward Secrecy cannot possibly be provided. The reason is the same as for Protocol 6.1 (see Section 6.2).

In the following, we examine the other security properties from the perspective of all three roles UE, $SEAF$ and HSS. Recall that we do not discuss authentication properties between HSS and $SEAF$, because we assume they have already set up a secure channel.

- UE:
  - Secrecy of MK is proved automatically.
  - Injective agreement with HSS on MK is proved automatically.
  - Injective agreement with $SEAF$ on MK is provable. The proof, however, is rather difficult. We reuse a strengthened form of the lemma `secrecy_UE` (which expresses secrecy of MK for UE) and provide an oracle that is able to guide the proof search for the first four depth levels of the proof. Using this oracle together with TAMARIN's standard *smart* heuristic (for all depth levels greater than four), the proof succeeds within minutes.
- $SEAF$:
  - We prove secrecy of MK, using the same approach as for EAP-AKA' v1 (as described in Section 6.2 on page 60).
  - We prove injective agreement with UE on MK, using the same approach as for EAP-AKA' v1 (as described in Section 6.2 on page 60).
- HSS:
  - Secrecy of MK is proved automatically.
  - Injective agreement with UE on MK is proved automatically.

Tables 6.3 and 6.4 summarize all discussed security properties.

	Perspective		
	UE	$SEAF$	HSS
Secrecy MK	✓	✓	✓
PFS MK	×	×	×

Table 6.3: Secrecy properties of theory `EAPAKAPRIMEv2`, modeling EAP-AKA' v2 (Protocol 6.2).

As desired, all analyzed security properties from EAP-AKA' v1 (Protocol 6.1) are preserved. Additionally, injective agreement from UE with $SEAF$ on the master key MK is established.

	Perspective			
	$UE \mapsto SEAF$	$UE \mapsto HSS$	$SEAF \mapsto UE$	$HSS \mapsto UE$
Aliveness	✓	✓	✓	✓
Weak agreement	✓	✓	✓	✓
Non-injective agreement	✓	✓	✓	✓
Injective agreement	✓	✓	✓	✓

Table 6.4: Authentication properties of theory `EAPAKAPRIMEv2`, modeling EAP-AKA' v2 (Protocol 6.2). All properties are with respect to MK.

6.4 Fast Re-Authentication

Recall that in EAP-AKA' v1 and EAP-AKA' v2 (Protocols 6.1 and 6.2), we ignored the Fast Re-authentication procedure. This is a protocol that can be run *after* a successful authentication via EAP-AKA'. It aims to establish a fresh Master Key MK_{re} and to re-authenticate UE.

In the following, K_{aut}, K_{encr}, and K_{re} refer to the values that have been exchanged in a previous run of a full EAP-AKA' authentication. Additionally, we let PRF° be a new pseudorandom function, and we define the following values:

$$\begin{aligned} MSK_{re} &:= PRF^{\circ}(\text{`}MSKre\text{`}, K_{re}, \text{`}EAPAKA^{\circ}reauth\text{`}, \\ &\qquad IMSI, counter, NONCE_S) \\ EMSK_{re} &:= PRF^{\circ}(\text{`}EMSKre\text{`}, K_{re}, \text{`}EAPAKA^{\circ}reauth\text{`}, \\ &\qquad IMSI, counter, NONCE_S) \\ MK_{re} &:= \langle MSK_{re}, EMSK_{re} \rangle \end{aligned}$$

The Fast Re-authentication protocol runs as follows.

Protocol 6.3 (EAP-AKA' Fast Re-authentication)

1. $SEAF \to UE : \text{`}IdentityRequest\text{`}$
2. $UE \to SEAF : IMSI$
3. $SEAF \to UE : \langle \{|\text{`}Reauth\text{`}, P_{SEAF}, counter, NONCE_S|\}_{K_{encr}}, HMAC(K_{aut}, \{|\text{`}Reauth\text{`}, P_{HSS}, counter, NONCE_S|\}_{K_{encr}}) \rangle$
4. $UE \to SEAF : \langle \{|P_{UE}, counter|\}_{K_{encr}}, HMAC(K_{aut}, \{|P_{UE}, counter|\}_{K_{encr}}) \rangle$
5.* $SEAF \to UE : \langle \{|\text{`}RequestSuccess\text{`}, counter|\}_{K_{encr}}, HMAC(k_{aut}, \{|\text{`}RequestSuccess\text{`}, counter|\}_{K_{encr}}) \rangle$
6.* $UE \to SEAF : \langle \{|\text{`}ResponseSuccess\text{`}, counter|\}_{K_{encr}}, HMAC(k_{aut}, \{|\text{`}ResponseSuccess\text{`}, counter|\}_{K_{encr}}) \rangle$
7. $SEAF \to UE : \text{`}EAPSuccess\text{`}$

Note the following:

- After successful execution, UE and $SEAF$ use the new master key MK_{re}.
- *counter* is a 16-bit value that is generated by $SEAF$. It is used in a way similar to the sequence number SQN from the AKA key exchange. UE must verify that for every new re-authentication request from $SEAF$, the included *counter* is strictly larger compared to the *counter* from the previous re-authentication (if there is any). This ensures uniqueness of *counter* and thereby establishes replay protection.

  As a side effect, this counter also limits the number of possible re-authentications.
- $NONCE_S$ is a fresh value generated by $SEAF$. It is used in a way similar to the value $RAND$ from the AKA key exchange.
- Similar to EAP-AKA' v1 and EAP-AKA' v2 (Protocols 6.1 and 6.2), messages 5 and 6 are optional. They are sent if and only if both protection bits P_{UE} and P_{SEAF} are set to one.

It is apparent that the secret key K that is shared between UE and the home network HSS is not used. Instead, Fast Re-authentication uses the previously exchanged key material (in particular K_{encr} and K_{aut}) to agree on a new key. As a result, there is no need for HSS to participate in the process of re-authentication.

Combining this with our existing model would nevertheless add a substantial amount of complexity, which we avoid by implementing the Fast Re-authentication protocol in its own isolated model. This has the drawback that the proved security properties do not necessarily carry over to a non-isolated model in which Fast Re-authentication runs together with EAP-AKA'. We discuss this further in Section 6.4.3.

6.4.1 Building a Model

We make the following modeling decisions:

- For reasons of simplicity, the symmetric message encryption is ignored. Instead, all encrypted messages are transmitted in plain text. This is completely attack preserving, and we are still able to prove all desired security properties.
- A full EAP-AKA' authentication is represented by a persistent fact `!INITAUTH(imsi, SEAF, Kre, Kaut)`. From this fact, arbitrarily many instances of $SEAF$ and UE can be instantiated.
- The roles UE and $SEAF$ can be instantiated with an arbitrary choice of their protection bit P_{UE} and P_{SEAF}, respectively. Note that for a single

EAP-AKA' authentication between UE and $SEAF$, the protection bits are not necessarily fixed, that is, we allow them to change for every new run of the Fast Re-authentication protocol.

- For reasons of efficient automatic verification, we let the adversary construct the counter value for the serving network $SEAF$.

  We add the restriction `SEAFCounter`, modeling that for a single EAP-AKA' authentication session, $SEAF$ uses a unique *counter* value for every new Fast Re-authentication. Note that we allow different authentication sessions to share counters.

  Furthermore, we add the restriction `UECounter`, modeling that for a single EAP-AKA' authentication session, UE only accepts unique *counter* values for every re-authentication attempt within one authentication session.

  The implementation of the counter that is used in practice (described above) is such that it can be seen as an instantiation of our model. $SEAF$ is able to generate unique counters, and UE is able to accept only new counters.

The resulting Tamarin theory is called `EPSAKAPRIMEREAUTH`.

6.4.2 Security Properties

Recall that the previously analyzed EAP-AKA' does not provide Perfect Forward Secrecy. As the long-term key is still the same, an attacker can easily exploit this fact to recover K_{re} and K_{aut} after the execution of the Fast Re-authentication protocol. The use of exclusively symmetric key cryptography makes it further trivial to compute MK_{re}, violating Perfect Forward Secrecy of the Fast Re-authentication protocol.

However, secrecy of MK_{re} is proved automatically for both $SEAF$ and UE, as well as injective agreement of $SEAF$ with UE on MK_{re} and vice versa.

Tables 6.5 and 6.6 summarize all discussed security properties.

	Perspective	
	UE	$SEAF$
Secrecy MK_{re}	✓	✓
PFS MK_{re}	×	×

Table 6.5: Secrecy properties of theory `EAPAKAPRIMEREAUTH`, modeling EAP-AKA' Fast Re-authentication (Protocol 6.3).

An interesting observation is that UE only confirms the value *counter* towards $SEAF$ (see messages 4 and 6), and not the value $NONCE_S$. As a

	Perspective	
	$UE \mapsto SEAF$	$SEAF \mapsto UE$
Injective agreement MK_{re}	✓	✓

Table 6.6: Authentication properties of theory `EAPAKAPRIMEREAUTH`, modeling EAP-AKA' Fast Re-authentication (Protocol 6.3).

consequence, non-injective agreement with UE on MK_{re} can be violated in case UE accidentally accepts the same *counter* twice, which should never happen if UE executes the protocol correctly. The reason is that message 4 does not depend on $NONCE_S$, so an attacker would be able to reuse an old version of message 4 that has been generated for a different $NONCE_S$ (but for the same *counter*). Note that the encryption that the protocol uses in practice to protect the messages (which is not implemented in our model) would not mitigate this issue. An effective way to avoid this issue would be to add $NONCE_S$ (integrity protected) to message 4.

Similar to the full EAP-AKA' authentication protocol, the protected confirmation messages do not seem to provide value from a security perspective.

6.4.3 Security of EAP-AKA' with Fast Re-Authentication

Recall that we analyzed the EAP-AKA' authentication protocol (Sections 6.2 and 6.3) and the Fast Re-authentication protocol in isolation.

Therefore, it is important to think about the consequences of merging these two protocols into one protocol, just as it is done in practice.

We are confident that the proved security properties in each of the protocols would also hold in a merged version. The key reason for this lies in the fact that all cryptographically protected messages of the two protocols are tagged integrity-protected in such a way that they cannot be confused with each other. Moreover, messages which are not cryptographically protected, i.e., transmitted in plain text, are completely controllable by the adversary in our model. The fact that some of those messages are confusable by each other is therefore irrelevant to the security properties. Hence, the two protocols are not expected to interfere with each other in any way that would affect the security properties.

6.5 Comparing EAP-AKA' to EPS-AKA*

Although both EAP-AKA' and EPS-AKA* are based on the same AKA key exchange, the protocols do not have the same security properties. In order to be able to see essential differences between the protocols, we compare them both with and without confirmation messages. Since their secrecy properties are the same, we focus on the authentication properties.

6.5.1 Comparison without Confirmation Messages

Table 6.7 contrasts the authentication properties of EPS-AKA* and EAP-AKA' without confirmation messages.

	Perspective							
	$UE \mapsto SEAF$		$UE \mapsto HSS$		$SEAF \mapsto UE$		$HSS \mapsto UE$	
	P_1	P_2	P_1	P_2	P_1	P_2	P_1	P_2
Aliveness	×	✓	✓	✓	✓	✓	✓	✓
Weak agreement	×	✓	✓	✓	×	✓	✓	✓
Non-injective agreement	×	×	×	✓	×	✓	×	✓
Injective agreement	×	×	×	✓	×	✓	×	✓

Table 6.7: Authentication properties of EPS-AKA* (P_1, left) and EAP-AKA' (P_2, right) without confirmation messages. All properties are with respect to K_{ASME} for EPS-AKA*, and with respect to MK for EAP-AKA'.

All differences are at least partially related to a failing agreement on the identity name $SNid$ of the serving network. Since the agreed-upon keys K_{ASME} and MK depend on $SNid$, agreement on the value of $SNid$ is critical for non-injective agreement on both K_{ASME} and MK.

For each perspective, we provide an intuitive understanding of the differences of the achieved authentication properties.

- $UE \mapsto SEAF$: EAP-AKA' achieves weak agreement on MK, while EPS-AKA* achieves only weak agreement on K_{ASME}. The reason for this difference is mainly that the authentication vector AV of EAP-AKA' contains the name of the serving network $SNid$, which is not the case for EPS-AKA*, where UE receives no cryptographically verifiable information on the value of $SNid$ from any of its peers.

  Note also that non-injective agreement fails for EAP-AKA' only because $SEAF$ receives the agreed-upon key MK only at the very end of the protocol. As we assume an interruptible channel, UE has no guarantee that $SEAF$ will ever receive the key MK. In case of an uninterruptible channel, however, we believe the property would hold.

- $UE \mapsto HSS$: EAP-AKA' achieves injective agreement on MK, while EPS-AKA* fails to achieve even non-injective agreement on K_{ASME}. The reason for this is identical to the previous case, i.e., UE receives cryptographically verifiable information on the value of $SNid$ only in EAP-AKA', but not in EPS-AKA*.

- $SEAF \mapsto UE$: EAP-AKA' achieves injective agreement on MK. This is essentially because $SEAF$ receives the key MK only from HSS after HSS has verified the challenge response from UE. Since $SEAF$ and

HSS communicate over a secure channel, HSS is able to guarantee the authentication properties to $SEAF$.

EPS-AKA*, however, fails to provide even weak agreement on K_{ASME}. The reason is that $SEAF$ receives no cryptographically verifiable information on the value of $SNid$ from UE, making it trivial to violate weak agreement.

- $HSS \mapsto UE$: EAP-AKA' achieves injective agreement on the key MK, while EPS-AKA* does not provide even non-injective agreement on K_{ASME}.

  The reason for this difference is that in EAP-AKA', UE's response is authenticated using the key K_{aut} which depends on the name of the serving network $SNid$. In EPS-AKA*, however, the home network HSS receives no cryptographically verifiable information on UE's view of the value $SNid$. In particular, the response RES does not depend on $SNid$ in any way.

6.5.2 Comparison with Confirmation Messages

Adding the confirmation messages between UE and $SEAF$ helps in resolving all authentication issues of EPS-AKA* except non-injective agreement of HSS with UE on the key K_{ASME}. This is no surprise, since HSS still receives no cryptographically verifiable information on UE's view of the value $SNid$.

Table 6.8 contrasts the authentication properties of EPS-AKA* and EAP-AKA' with confirmation messages.

	Perspective							
	$UE \mapsto SEAF$		$UE \mapsto HSS$		$SEAF \mapsto UE$		$HSS \mapsto UE$	
	P_1	P_2	P_1	P_2	P_1	P_2	P_1	P_2
Aliveness	✓	✓	✓	✓	✓	✓	✓	✓
Weak agreement	✓	✓	✓	✓	✓	✓	✓	✓
Non-injective agreement	✓	✓	✓	✓	✓	✓	×	✓
Injective agreement	✓	✓	✓	✓	✓	✓	×	✓

Table 6.8: Authentication properties of EPS-AKA* (P_1, left) and EAP-AKA' (P_2, right) with confirmation messages. All properties are with respect to K_{ASME} for EPS-AKA*, and with respect to MK for EAP-AKA'.

6.5.3 Conclusion

The comparison highlights the fact that EAP-AKA' provides strictly stronger authentication guarantees than EPS-AKA*, at least for agreement on the key MK and K_{ASME}, respectively.

Moreover, EPS-AKA* achieves multiple authentication properties only after the agreed-upon key material has been used at least once, for example in confirmation messages (modeling the Security Mode Command (SMC) procedure). In contrast, EAP-AKA' is able to provide all its authentication guarantees before exchanging additional confirmation messages. A drawback of EAP-AKA' is, however, that the protocol exchanges more messages than EPS-AKA*.

Chapter 7

Improving Model Precision

Our previously presented models of EPS-AKA* (Chapter 5) and EAP-AKA' (Chapter 6) are based on two major simplifications.

- First and most importantly, the sequence number is modeled as a fresh value and UE is simply assumed to never accept the same sequence number more than once. Additionally, the model completely ignores the *re-synchronization* protocol, which is the mandatory mechanism that is used to resynchronize the sequence number between UE and the home network once it gets out of sync.
- Second, the anonymity key AK is modeled as a constant zero value instead of a value derived from a key derivation function.

In this chapter, we present a modeling approach that leverages Tamarin's support for mutable global state as well as its experimental *XOR* built-in to precisely account for both the sequence number with re-synchronization as well as a non-zero anonymity key AK. Moreover, we apply the approach to EPS-AKA* and again verify the same security properties as in Chapter 5.

7.1 Sequence Number

The sequence number SQN is part of the AKA key exchange protocol described in [3]. As such, it is part of all AKA-based protocols. Annex C of [3] describes multiple management schemes for the sequence number, some of which are time-based. In this chapter, we focus on the management schemes that are *not* time-based. In particular, we treat Profile 2 as described in Section C.3.2 of [3].

The sequence number is a 48-bit value SQN, that is composed of the counter value SEQ and IND, i.e., $SQN = SEQ||IND$. Typically, SEQ is 43 bits long and IND is 5 bits long.

7.1.1 Storing the Sequence Number

Every UE stores an array SEQ_{MS} with a entries, indexed from 0 to $a-1$. All entries are initialized with zero. Typically, $a = 32$.

Every home network HSS stores a counter SEQ_{HE} for each UE. Said counter is initialized to zero, and increased for every new authentication value. The home network sends the value $SQN = SEQ_{HE}||IND$, where IND is a value between 0 and $a-1$.

7.1.2 Checking Sequence Number Freshness

Let $SQN_{MS} = SEQ_{MS}||IND_{MS}$ denote the highest accepted sequence number that is contained in the array, i.e., $SEQ_{MS} = \max_i SEQ_{MS}(i)$.

For fixed L and Δ, a UE accepts a new sequence number $SQN = SEQ||IND$ as fresh if and only if:

$$SEQ \in (SEQ_{MS} - L, SEQ_{MS} + \Delta] \text{ and } SEQ > SEQ_{MS}(IND)$$

Note the following:

- L is optional. In case it is not used, it is set to ∞.
- Δ is typically 2^{28}.

If a received SEQ is accepted as fresh, the value $SEQ_{MS}(IND)$ is set to SEQ.

Otherwise, a synchronization failure with the value SEQ_{MS} is sent.

7.2 Re-Synchronization

Let f_1^* be a new MAC function and let f_5^* be a new key derivation function. Moreover, let AV be the usual AKA authentication vector, let K be the symmetric key shared between UE and HSS, and let $RAND$ be a fresh value that is chosen by HSS during the usual AKA protocol. We define the following values.

$$\begin{aligned} MACS &:= f_1^*(AMF, RAND, K, SQN_{MS}) \\ AK &:= f_5^*(RAND, K) \\ AUTS &:= (SQN_{MS} \oplus AK)||MACS \end{aligned}$$

Note that the value AMF is the authentication management field, which is used for additional indications between the user equipment UE and the home network HSS.

If a synchronization failure with the value SEQ_{MS} is sent, the following re-synchronization procedure is run.

Protocol 7.1 (Re-Synchronization)

1. $UE \rightarrow SEAF : \langle \text{`}syncfailure\text{'}, AUTS \rangle$
2. $SEAF \bullet\!\!\rightarrow\!\!\bullet\ HSS : \langle \text{`}authrequest\text{'}, \text{`}syncfailure\text{'}, RAND, AUTS \rangle$
3. $HSS \bullet\!\!\rightarrow\!\!\bullet\ SEAF : \langle \text{`}authresponse\text{'}, AV \rangle$

When HSS receives the authentication request with synchronization failure indication from $SEAF$, it will set its stored counter SQN_{HE} to SQN_{MS} if $SQN_{HE} + 1 \leq SQN_{MS}$, i.e., if the next counter value would not be accepted by UE. The new SQN_{HE} is then used to generate an authentication vector AV.

7.3 Building a Model

In the following, we present a modeling approach for the sequence number itself, the re-synchronization protocol, and a non-zero anonymity key.

7.3.1 Sequence Number

We make the following modeling decisions concerning the sequence number $SQN = SEQ||IND$.

- We model SEQ as a possibly infinite multiset of ones (‘1‘), capturing the property that Δ is chosen such that the counter never wraps within the usual lifetime of a user equipment.
- We set the array size a to one. This allows us to ignore the index IND from SQN, as there is only one valid index.

  This is a reasonable modeling choice, as the array indexing mechanism is primarily used between different batches of authentication vectors. However, as specified in [6], it is recommended (in some cases even mandatory) that the serving network requests only one authentication vector at once. In this case, the array indexing mechanism is not needed anyway.
- We ignore the check $SEQ \in (SEQ_{MS} - L, SEQ_{MS} + \Delta]$, i.e., we set $L = \Delta = \infty$.

  Note that the lower bound $SEQ_{MS} - L$ is optional to check, and the upper bound $SEQ_{MS} + \Delta$ is only needed to ensure that the counter SEQ does not wrap within the usual lifetime of a user equipment UE. This property is captured in our model already, as we have an infinite-sized counter.

- The rule `Register_IMSI` is used to create a fact `SQN_UE(IMSI, '1')` and a fact `SQN_HSS(HSS,'1')` for every $IMSI$ with HSS as its corresponding home network name. This models the initialization of the counter.
- Whenever UE or HSS need to read or update the counter value of SQN, the access is realized by directly using the fact `SQN_UE(...)` or `SQN_HSS(...)`, respectively.

  This is crucial to avoid race conditions on the counter value. In practice, every correct implementation of the protocol should guarantee that there are no such race conditions.
- f_5^* is the constant zero function. This is a valid instantiation according to the specification. In case the sequence number SQN is generated by a time-based scheme, it makes no sense to try to protect it with a (non-zero) anonymity key, as the time is publicly known anyway.
- We let the adversary construct the counter in order to remove some of the partial deconstructions[1]. Note that the counter is not secret anyway, since we model f_5^* as the constant zero function.

7.3.2 Re-Synchronization

Concerning the re-synchronization protocol, we take the following modeling decisions:

- When a synchronization failure occurs, UE sends message 1 and $SEAF$ sends message 2 as described in Protocol 7.1. Then, UE, $SEAF$, and HSS abort the protocol execution.

  Instead of sending the following authentication vector in a dedicated message as in Protocol 7.1 (message 3), HSS reruns the protocol and sends a new authentication vector via the regular AKA-based protocol. This is an approximation which allows us to reduce the complexity of the model significantly.
- In order to make it possible to execute the re-synchronization in our model, we introduce a mechanism that modifies the sequence numbers of UE and HSS such that a synchronization failure can be triggered.

  For UE, we add a rule `Increase_SQNUE` which increases UE's sequence number to an arbitrary value. Note that decreasing UE's sequence number would trivially allow replay attacks, and a correct implementation of the protocol must ensure that the sequence number is only increased.

[1] See Section 2.4.4 on page 23.

For HSS, we add a rule `Havoc_SQNHSS` which changes HSS's sequence number arbitrarily.

- The authentication management field AMF is modeled as the constant string 'AMF'.

7.3.3 Non-Zero Anonymity Key

In case the sequence number SQN is generated by a time-based scheme, it is publicly known anyway, making it pointless to protect it with an anonymity key AK. However, if SQN is not generated by a time-based scheme, it is reasonable to generate a non-zero anonymity key in order to protect the user's privacy.

This section is to be understood as an extension of the previous modeling choices. We model f_5^* as a key derivation function and aim to prove secrecy of the sequence number SQN for both UE and HSS.

In order to achieve this, it is necessary to modify the model of the sequence number, because a multiset consisting only of ones ('1') cannot be a (weak) secret from the adversary. We introduce the following modifications:

- The counter between UE and HSS consists of the two parts $SQNBASE$ and $SQNOFFSET$. SQN's value is $SQNBASE + SQNOFFSET$.

  The value $SQNBASE$ is set to a fresh value $SQNBASE$ that is only known to UE and HSS. After being generated, its value never changes.

  The value $SQNOFFSET$ is initialized to zero. It is increased by one on every authentication attempt by the home network HSS. Note that $SQNOFFSET$ essentially corresponds to the full sequence number SQN in the previous model.

- When UE or HSS receive an SQN value, they separate it to the two values $SQNBASE$ and $SQNOFFSET$, allowing a more efficient proof for some of the security properties.

  Intuitively, this can be understood as the agents checking whether the received value is actually greater or equal than $SQNBASE$ and storing only the new offset $SQN - SQNBASE$. Note that values smaller than $SQNBASE$ are not accepted.

- As in the previous model, we let the adversary construct the part of the counter that is known to the adversary anyway, namely $SQNOFFSET$. This allows us to remove some of the partial deconstructions[2].

The sequence number being composed of a secret base and a public offset intuitively represents a threat model in which the adversary is imagined

[2] See Section 2.4.4 on page 23.

to join the network at some point, not knowing the current value of the sequence number. However, from the moment she has joined, she can easily count the number of authentication runs and therefore infer the offset from the secret base.

7.4 EPS-AKA*

We extend the previous model of EPS-AKA* as described in Chapter 5 iteratively with our new model, starting with the sequence number, then adding the re-synchronization protocol, and finally the non-zero anonymity key. The resulting theories are `EPSAKA_STAR_REALSQN`, `EPSAKA_STAR_RESYNC` and `EPSAKA_STAR_RESYNC_XOR`. Note that `EPSAKA_STAR_RESYNC_XOR` builds on `EPSAKA_STAR_RESYNC`, which itself builds on `EPSAKA_STAR_REALSQN`.

As the models become more involved, proving security properties quickly becomes rather difficult. We briefly comment on the challenges encountered with the three theories.

- `EPSAKA_STAR_REALSQN`: In order to prove authentication properties as injective agreement on K_{ASME} between different roles, it is necessary to use the helping lemmas `UEcounterINIT` and `uniqueUECounter`.

  `UEcounterINIT` describes the property that every use of UE's sequence number SQN is preceded by an initialization of said sequence number.

  `uniqueUECounter` describes the property that UE will never accept the same sequence number SQN twice. It is proved automatically, making use of the helping lemma `HELPERuniqueUECounter`.

  `HELPERuniqueUECounter` describes the property that UE's counter is only increased. Unfortunately, the lemma is not automatically provable with the default *smart* heuristic. We provide an oracle[3] that allows to prove said lemma.

  Note that `HELPERuniqueUECounter` is a helping lemma that is only used to prove `uniqueUECounter`. In order to prove injective agreement properties involving UE's counter, only `uniqueUECounter` is used.

- `EPSAKA_STAR_RESYNC`: This theory is based on `EPSAKA_STAR_REALSQN` (see above). It additionally models the re-synchronization protocol.

  Similar as in theory `EPSAKA_STAR_REALSQN`, we add helping lemmas, accounting for the different ways in which UE's sequence number is updated during the re-synchronization protocol.

- `EPSAKA_STAR_RESYNC_XOR`: This theory builds on `EPSAKA_STAR_RESYNC` (see above).

[3] See Section 2.4 on page 21.

Moreover, the theory makes use of Tamarin's experimental XOR feature to model the value $SQN \oplus AK$.

In order to prove some of the secrecy properties, we specify two additional helping lemmas describing the secrecy of $SQNBASE$ as well as secrecy of the anonymity key AK. This allows us to automatically prove secrecy of K_{ASME} for UE, $SEAF$, and HSS, as well as secrecy of the sequence number SQN for both UE and HSS.

Some of the authentication properties, however, are not provable automatically with Tamarin's default *smart* heuristic anymore. We provide an oracle that allows us to prove injective agreement on K_{ASME} for UE with both $SEAF$ and HSS, and injective agreement on K_{ASME} for $SEAF$ with UE.

With the techniques described above, we successfully prove the same security properties as in Section 5.2 for all three theories `EPSAKA_STAR_REALSQN`, `EPSAKA_STAR_RESYNC` and `EPSAKA_STAR_RESYNC_XOR`.

Furthermore, we are able to prove secrecy of the sequence number SQN for both UE and HSS in theory `EPSAKA_STAR_RESYNC_XOR`.

Tables 7.1, 7.2, and 7.3 summarize the security properties.

	Perspective		
	UE	$SEAF$	HSS
Secrecy K_{ASME}	✓	✓	✓
PFS K_{ASME}	×	×	×

Table 7.1: Secrecy properties that hold for the theories `EPSAKA_STAR_REALSQN`, `EPSAKA_STAR_RESYNC`, and `EPSAKA_STAR_RESYNC_XOR`, which model EPS-AKA* (Protocol 5.4) with different levels of extensions.

	Perspective	
	UE	HSS
Secrecy SQN	✓	✓

Table 7.2: Additional secrecy properties on SQN that hold exclusively for theory `EPSAKA_STAR_RESYNC_XOR`, modeling EPS-AKA* (Protocol 5.4) with resynchronization and non-zero anonymity key.

	Perspective			
	$UE \mapsto SEAF$	$UE \mapsto HSS$	$SEAF \mapsto UE$	$HSS \mapsto UE$
Aliveness	✓	✓	✓	✓
Weak agreement	✓	✓	✓	✓
Non-injective agreement	✓	✓	✓	×
Injective agreement	✓	✓	✓	×
Injective agreement ($RAND$)	✓	✓	✓	✓

Table 7.3: Authentication properties of theories `EPSAKA_STAR_REALSQN`, `EPSAKA_STAR_RESYNC` and `EPSAKA_STAR_RESYNC_XOR`, modeling EPS-AKA* (Protocol 5.4) with different levels of extensions. All properties are with respect to K_{ASME}, unless otherwise stated.

Chapter 8

Conclusion

We formalized multiple authentication protocols of the new 5G standard and were able to prove numerous security properties in our symbolic model. The results mostly provide confidence in the design of today's and future telecommunication protocols.

In Chapter 4, we analyzed the newly proposed identity-based protocols Relay-Authentication and Aggregation-Authentication as specified in Solution #2.16 and Solution #2.15 of [6], respectively. The results showed different weaknesses. Most importantly, a design flaw affecting both protocols allows mounting a man-in-the-middle attack on both roles of the protocols. Moreover, the imprecise specification of the protocols was found to cause several issues. For both protocols, we proposed changes to resolve all discovered problems.

Then, in Chapter 5, we introduced a basic model for EPS-AKA as well as its 5G successor EPS-AKA*. The subsequent analysis confirmed that EPS-AKA provides the home network only with weak authentication properties on the user equipment. EPS-AKA* was designed with the explicit goal of improving authentication guarantees for the home network and we successfully proved stronger authentication properties. However, the protocol does not achieve agreement for the home network with the user equipment on the identity of the serving network. This has already been noted in the comments that are provided with the current specification's draft [6]. A solution that would actually allow the home network to agree with the user equipment on the identity of the serving network is most likely not possible without changing the user equipment's role in the protocol. The 3GPP seems to be reluctant to implement such a change, presumably because it would break backwards compatibility with current user equipment.

In Chapter 6, we presented models for EAP-AKA' as well as its Fast Reauthentication mechanism and formally verified their properties. The com-

parison between the properties of EAP-AKA' and EPS-AKA* has shown that EAP-AKA' achieves strictly stronger security guarantees than EPS-AKA*. This is the case both before and after running the so-called Security Mode Command procedure (SMC) which succeeds the actual authentication protocol, although the difference is more profound before running the SMC. However, a drawback of EAP-AKA' is that the protocol exchanges more messages than EPS-AKA*.

Finally, we presented a precise model for the sequence number and its resynchronization mechanism in Chapter 7. We demonstrated how our approach can be applied to EPS-AKA* and again verified the properties proved in Chapter 5.

In the following, we give an overview of the general types of problems that we encountered during the analysis of the protocols.

- Protocol specifications are often imprecise, especially for non-standard documents such as [6] in which the new 5G protocols using ID-based credentials are described. This requires making additional assumptions on the protocols.

  In general, we tried to make as few assumptions as possible, and whenever needed, we aimed to make assumptions in an attack-preserving way. In some cases, however, this was very challenging. For example, the specification of the protocols analyzed in Chapter 4 describes some of the messages in words only. Hence, we needed to determine our own message format for our model. This is a problem, since there may exist attacks that depend on the concrete message format, for example when exploiting that different messages are confusable.

  For particularly imprecise protocol specifications, most notably the ID-based protocols described in [6], we proposed a set of explicit improvements to be added to the protocol specification.

  Furthermore, some protocol specifications deliberately leave room for a variety of different implementations. The intention is to enable the different network operators to adjust their implementation to their own needs. When modeling such a protocol for a specific variant, the results do not easily translate to the alternative variants. Whenever we modeled such a specific variant, we preferred the implementation which we considered to be most common in practice.

- Security goals are often not stated explicitly or they are very imprecise. Therefore, we tried to prove straightforward but strong security properties from the perspectives of the different protocol participants. In some cases, this has led to attacks which may not be considered critical with respect to the implicit security goals of the telecommunication network.

We strongly suggest that the protocol designers provide an accurate specification along with a precise statement of the desired security goals. Especially for the new 5G protocols, this is of critical importance, since for most of them it is not determined yet in what exact context and environment they are going to be used.

- The security properties of some of the analyzed protocols strongly depend either on a higher layer protocol such as Diameter, or on a protocol that is run subsequently to the actual authentication protocol. Therefore, it was sometimes necessary to take the behavior of these additional dependencies at least partially into account. In general, this is suboptimal, because the results of the formal analysis may not trivially hold anymore if only one of these dependencies is modified or replaced.
- Overloaded protocols with many seemingly unnecessary messages, most notably EAP-AKA', have led to a high model complexity. This created the need for many helping lemmas in order to prove the desired properties, or even the development of a so-called oracle to guide the proof search.

Throughout this work, we made strong use of the Tamarin prover which turned out to be a very valuable tool. It allowed us to prove many properties fully automatically. In case a property did not hold, the tool was often able to find an attack. However, for the most complex models, in particular those for EAP-AKA' and the precise version of EPS-AKA*, a significant manual effort was required in order for the proofs to succeed. More specifically, the built-in heuristics of Tamarin were not able to provide choices that led to a successful proof for these models within a reasonable amount of time. Hence, it was necessary to provide either additional helping lemmas that helped to split a proof into smaller parts, or a so-called oracle to guide the proof search with superior choices than the built-in heuristics. Even though some of the models represented a major challenge for the tool in its present version, we expect Tamarin to evolve, the heuristics to improve, and the efficiency to increase, such that no manual work will be required in the future for similar models.

Within the telecommunication protocol landscape and especially 5G, we expect the following challenges to be addressed in future work:

- Include a precise model for sequence number with re-synchronization and exclusive-or operator, such as presented in Chapter 7, into EAP-AKA'.
- Model EAP-AKA' in combination with its Fast Re-authentication protocol, as opposed to our separated modeling approach.
- Model and formally verify other 5G protocols.

- Model and formally verify indistinguishability properties of various protocols. This is particularly interesting for privacy concerns.
- Combine different authentication protocols that current devices support into one model and verify the combined properties. Since even the most recent devices support obsolete protocols (e.g., from 2G), there is the potential risk of downgrade attacks.

For some of this future work, it will most likely be necessary for formal verification tools to become more efficient. There are multiple directions to pursue this goal. One direction might be to improve the heuristics in order to be able to find more complex proofs fully automatically. Another direction might be to provide a more expressive input language, for example by adding special annotations that allow guiding the proof search algorithm in a more convenient way than what is currently possible using oracles for Tamarin.

Appendix A

Tamarin Model - EPS-AKA*

```
theory EPSAKA_STAR_RESYNC_XOR
begin

//######################################################################
//Sources: 3GPP TR 33.899 V1.3.0 (2017-08),
//         3GPP TS 33.401 V14.2.0 (2017-03),
//         3GPP TS 33.102 V14.1.0 (2017-03)
//Author:  David Lanzenberger
//Date:    July 2017
//######################################################################

builtins: symmetric-encryption, multiset, xor

functions: KDF/1

//########################KEY GENERATION RULES#########################

// both UE and HSS initialize their counter to 1
rule Register_IMSI:
  [ Fr(~imsi), Fr(~Kimsi), Fr(~SQNBASE) ]
  --[ UEInitCounter(~imsi, $HSS, ~SQNBASE)
    , HSSInitCounter($HSS, ~imsi, ~SQNBASE)
    , REGISTER_IMSI(~imsi, $HSS, ~Kimsi)
    ]->
  [ !IMSI(~imsi, $HSS)
  , !IMSIK(~imsi, $HSS, ~Kimsi)
  , SQNUE(~imsi, $HSS, '1')
  , SQNHSS($HSS, ~imsi, '1')
  , !SQNBASEUEHSS(~imsi, $HSS, ~SQNBASE)
  ]

// This shared symmetric key is used to model a secure channel
// between SEAF and HSS. It ensures authenticated and confidential
```

```
// messages, as well as integrity. However, it allows the adversary
// to replay messages arbitrarily many times and to reorder them.
rule Shared_Symmetric_Key:
  [ Fr(~SK) ]
  --[ Shared_Symmetric_Key(~SK,$SEAF,$HSS) ]->
  [ !SYMKEY(~SK,$SEAF,$HSS) ]

//########################REVEAL RULES##############################

rule Reveal_IMSI:
  [ !IMSIK(imsi, HSS, Kimsi) ]
  --[ Reveal(<'IMSI',imsi>) ]->
  [ Out(Kimsi) ]

rule Reveal_Shared_Symmetric_Key:
  [ !SYMKEY(SK,SEAF,HSS) ]
  --[ Reveal(<'SEAFHSS',SEAF,HSS>) ]->
  [ Out(SK) ]

//#####################SEQUENCE NUMBER RULES#####################

// we allow HSS's counter to be changed to an arbitrary value
// this models situations in which HSS gets out of sync with UE
rule Havoc_SQNHSS:
  [ SQNHSS(HSS, imsi, SQNBASEOFFSET)
  , !SQNBASEUEHSS(imsi, HSS, ~SQNBASE)
  , In(newSQNBASEOFFSET)
  ]
  --[ HAVOCSQNHSS(HSS, imsi, ~SQNBASE,
                  SQNBASEOFFSET, newSQNBASEOFFSET)
    , USE_SQNHSS(HSS, imsi, ~SQNBASE, newSQNBASEOFFSET)
    ]->
  [ SQNHSS(HSS, imsi, newSQNBASEOFFSET) ]


// we allow HSS's counter to be changed to an arbitrary value
// this models situations in which HSS gets out of sync with UE
rule Increase_SQNUE:
  [ SQNUE(imsi, HSS, SQNBASEOFFSET)
  , !SQNBASEUEHSS(imsi, HSS, ~SQNBASE)
  , In(newSQNBASEOFFSET)
  ]
  --[ INCREASESQNOFFSETUE(imsi, HSS, ~SQNBASE,
                          SQNBASEOFFSET, newSQNBASEOFFSET)
    , UE_SET_SQN(imsi, HSS, ~SQNBASE, newSQNBASEOFFSET)
    , LessThan(SQNBASEOFFSET, newSQNBASEOFFSET)
    ]->
  [ SQNUE(imsi, HSS, newSQNBASEOFFSET) ]
```

```
//######################INITIALIZATION RULES######################

rule initialize_UE:
  [ Fr(~id)
  , !IMSI(imsi, $HSS)
  , !IMSIK(imsi, $HSS, Kimsi)
  ]
  --[ Create(imsi, ~id, 'UE')  ]->
  [ St_UE_0(imsi, Kimsi, ~id, $HSS, $SNid) ]

rule initialize_SEAF:
  [  Fr(~id)
  ,  !SYMKEY(SK,$SEAF,$HSS)
  ]
  --[ Create($SEAF, ~id, 'SEAF') ]->
  [ St_SEAF_0($SEAF, ~id, $HSS, SK) ]

rule initialize_HSS:
  [ Fr(~id)
  , !IMSIK(imsi, $HSS, Kimsi)
  , !SYMKEY(SK,$SEAF,$HSS)
  ]
  --[ Create($HSS, ~id, 'HSS')]->
  [ St_HSS_0($HSS, ~id, imsi, Kimsi, $SEAF, SK) ]

//########################PROTOCOL RULES############################

rule SEAF_1send:
  let m = 'IdentityRequest'
  in
  [ St_SEAF_0($SEAF, id, $HSS, SK) ]
  --[ SEAF_1send(id) ]->
  [ Out(m)
  , St_SEAF_1($SEAF, id, $HSS, SK)
  ]

rule UE_1recv:
  let m = 'IdentityRequest'
  in
  [ St_UE_0(imsi, Kimsi, ~id, $HSS, $SNid)
  , In(m)
  ]
  --[]->
  [ St_UE_1(imsi, Kimsi, ~id, $HSS, $SNid) ]

rule UE_2send:
  [ St_UE_1(imsi, Kimsi, ~id, $HSS, $SNid) ]
  --[//This is necessary for weak agreement from HSS with UE,
     //because the protocol tries to agree on KASME,
```

```
      //which is not known at this step.
      //Furthermore, this is the only message sent from UE
      //before the last message is received by HSS
      Running(imsi, $HSS, <'HSS','UE','dummy'>)
     ]->
   [ St_UE_2(imsi, Kimsi, ~id, $HSS, $SNid)
   , Out(imsi)
   ]


rule SEAF_2recv:
   [ St_SEAF_1($SEAF, id, $HSS, SK)
   , In(imsi)
   ]
   --[]->
   [ St_SEAF_2($SEAF, id, imsi, $HSS, SK) ]


rule SEAF_3send:
   let m = <~S,imsi,$SEAF>
       menc = senc{m}SK
   in
   [ Fr(~S)
   , St_SEAF_2($SEAF, id, imsi, $HSS, SK)
   ]
   --[ SEAF_SendEncrypted($SEAF,~S) ]->
   [ Out(menc)
   , St_SEAF_3($SEAF, id, imsi, $HSS, SK, ~S)
   ]

rule HSS_3recv:
   let m = <S,imsi,$SEAF>
       menc = senc{m}SK
   in
   [ St_HSS_0($HSS, id, imsi, Kimsi, $SEAF, SK)
   , In(menc)
   ]
   --[]->
   [ St_HSS_1($HSS, id, imsi, Kimsi, $SEAF, SK, S) ]


rule HSS_4send:
       //the old counter is increased by one
   let SQNBASEOFFSET = OLDSQNBASEOFFSET+'1'
       SQN = ~SQNBASE + SQNBASEOFFSET
       MAC = KDF(<'f1', Kimsi, 'AMF', SQN, ~RAND>)
       XRES1 = KDF(<'f2', Kimsi, ~RAND, '1'>)
       XRES2 = KDF(<'f2', Kimsi, ~RAND, '2'>)
       XRES = <XRES1,XRES2>
```

```
      CK = KDF(<'f3', Kimsi, ~RAND>)
      IK = KDF(<'f4', Kimsi, ~RAND>)
      AK = KDF(<'f5', Kimsi, ~RAND>)
      SQNxorAK = SQN XOR AK
      KASME = KDF(<'KASME',CK,IK,$SEAF,SQNxorAK>)
      AUTN = <SQNxorAK, 'AMF', MAC>
      AV_STAR = <~RAND, XRES1, KASME, AUTN>
      AV_STARenc = senc{<S,AV_STAR>}SK
  in
  [ St_HSS_1($HSS, id, imsi, Kimsi, $SEAF, SK, S)

  // Modeling trick to get rid of partial deconstructions
  , In(OLDSQNBASEOFFSET)
  , SQNHSS($HSS, imsi, OLDSQNBASEOFFSET)
  , !SQNBASEUEHSS(imsi, $HSS, ~SQNBASE)
  , Fr(~RAND)
  ]
  --[ HSS_4SendEncrypted($HSS, <S,AV_STAR>)
    , Running($HSS, imsi, <'UE','HSS',KASME>)
    , HSS_SQN_NEXT($HSS, imsi, ~SQNBASE, SQNBASEOFFSET)
    , SecretSQN(<'HSS',$HSS>, SQN)
    , Honest(<'IMSI',imsi>)
    , USE_SQNHSS($HSS, imsi, ~SQNBASE, SQNBASEOFFSET)
    ]->
  [ St_HSS_2($HSS, id, imsi, Kimsi, $SEAF, SK, S, SQN, ~RAND)
  , SQNHSS($HSS, imsi, SQNBASEOFFSET)
  , Out(AV_STARenc)
  ]

rule SEAF_4recv:
  let AV_STAR = <RAND, XRES1, KASME, AUTN>
      AV_STARenc = senc{<~S,AV_STAR>}SK
  in
  [ In(AV_STARenc)
  , St_SEAF_3($SEAF, id, imsi, $HSS, SK, ~S)
  ]
  --[]->
  [ St_SEAF_4($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR) ]

rule SEAF_5send:
  let AUTN = <SQNxorAK, 'AMF', MAC>
      AV_STAR = <RAND, XRES1, KASME, AUTN>
      m = <RAND,AUTN>
  in
  [ St_SEAF_4($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR) ]
  --[ Running($SEAF, imsi, <'UE','SEAF',KASME>)
    , OUT_SEAF_5send(m)
    ]->
  [ St_SEAF_5($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR)
```

```
  , Out(m)
  ]

rule UE_5recv_SEQGOOD:
  let SQN = SQNxorAK XOR AK
      MAC = KDF(<'f1', Kimsi, 'AMF', SQN, RAND>)
      CK = KDF(<'f3', Kimsi, RAND>)
      IK = KDF(<'f4', Kimsi, RAND>)
      AK = KDF(<'f5', Kimsi, RAND>)
      AUTN = <SQNxorAK, 'AMF', MAC>
      m = <RAND,AUTN>
      KASME = KDF(<'KASME',CK,IK,$SNid,SQNxorAK>)
  in
  [ In(m)
  , St_UE_2(imsi, Kimsi, ~id, $HSS, $SNid)

  // Modeling trick to get rid of partial deconstructions
  , In(SQNMAXBASEOFFSET), In(SQNBASEOFFSET)
  , SQNUE(imsi, $HSS, SQNMAXBASEOFFSET)
  , !SQNBASEUEHSS(imsi, $HSS, ~SQNBASE)
  ]
  --[ LessThan(SQNMAXBASEOFFSET,SQNBASEOFFSET)
    , UE_SQN_PREV(imsi, $HSS, ~SQNBASE, SQNMAXBASEOFFSET)
    , UE_SQN_NEXT(imsi, $HSS, ~SQNBASE, SQNBASEOFFSET)
    , SQNBASE(imsi, $HSS, ~SQNBASE)
    , Secret(<'IMSI',imsi>, KASME)
    , SecretSQN(<'IMSI',imsi>, SQN)

    // local check making sure that the received
    // SQN really is larger than SQNBASE
    , Eq(~SQNBASE+SQNBASEOFFSET, SQN)
    , Running(imsi, $SNid, <'SEAF','UE',KASME>)
    , Honest(<'SEAFHSS',$SNid,$HSS>)
    , Honest(<'IMSI',imsi>)
    , Running(imsi,$HSS,<'HSS','UE',<'RAND',RAND>>)
    , UE_5recv_SEQGOOD(~id)
    , UE_SET_SQN(imsi, $HSS, ~SQNBASE, SQNBASEOFFSET)
    ]->
  [ St_UE_3(imsi, Kimsi, ~id, $HSS, $SNid, RAND, KASME)

  //the received SQN is the new maximum
  , SQNUE(imsi, $HSS, SQNBASEOFFSET)
  ]

rule UE_5recv_SEQBAD:
  let SQN = SQNxorAK XOR AK
      MAC = KDF(<'f1', Kimsi, 'AMF', SQN, RAND>)
      CK = KDF(<'f3', Kimsi, RAND>)
      IK = KDF(<'f4', Kimsi, RAND>)
```

```
      AK = KDF(<'f5', Kimsi, RAND>)
      AUTN = <SQNxorAK, 'AMF', MAC>
      m = <RAND,AUTN>
      SQNMAX = ~SQNBASE+SQNMAXBASEOFFSET
      MACS = KDF(<'f1star','AMF',RAND,Kimsi,SQNMAX>)
      AK_STAR = KDF(<'f5star', RAND, Kimsi>)
      AUTS = <SQNMAX XOR AK_STAR, MACS>
  in
  [ In(m)
  , St_UE_2(imsi, Kimsi, ~id, $HSS, $SNid)

  // Modeling trick to get rid of partial deconstructions
  , In(SQNMAXBASEOFFSET), In(SQNBASEOFFSET)
  , SQNUE(imsi, $HSS, SQNMAXBASEOFFSET)
  , !SQNBASEUEHSS(imsi, $HSS, ~SQNBASE)
  ]
  --[ GreaterOrEqualThan(SQNMAXBASEOFFSET,SQNBASEOFFSET)
    , Eq(~SQNBASE+SQNBASEOFFSET, SQN)
    , UESyncFailure(imsi, $HSS, ~SQNBASE,
                    SQNMAXBASEOFFSET, SQNBASEOFFSET)
    , UE_5recv_SEQBAD(~id)
    , UE_SQN_NOCHANGE(imsi, $HSS, ~SQNBASE, SQNMAXBASEOFFSET)
    ]->
  [ //UE's run of the protocol is aborted (needs to be restarted)
    Out(<'syncfailure', AUTS>)

    //SQNMAX remains unchanged
  , SQNUE(imsi, $HSS, SQNMAXBASEOFFSET)
  ]


rule UE_6send:
  let RES1 = KDF(<'f2', Kimsi, RAND, '1'>)
      RES2 = KDF(<'f2', Kimsi, RAND, '2'>)
      RES = <RES1, RES2>
  in
  [ St_UE_3(imsi, Kimsi, ~id, $HSS, $SNid, RAND, KASME) ]
  --[]->
  [ Out(RES)
  , St_UE_4(imsi, Kimsi, ~id, $HSS, $SNid, RAND, KASME)
  ]

rule SEAF_6recv_NOSYNCFAILURE:
  let AUTN = <SQNxorAK, 'AMF', MAC>
      AV_STAR = <x, XRES1, KASME, AUTN>

      //we can only match on the first half of RES
      RES = <XRES1,RES2>
  in
```

```
  [ St_SEAF_5($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR)
  , In(RES)
  ]
  --[ Secret(<'SEAF',$SEAF>,KASME)
    , Honest(<'SEAFHSS',$SEAF,$HSS>)
    , Honest(<'IMSI',imsi>)
    , SEAF_6recv_NOSYNCFAILURE(id)
    ]->
  [ St_SEAF_6($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR, RES) ]

rule SEAF_6recv_SYNCFAILURE:
  let AUTN = <SQNxorAK, 'AMF', MAC>
      AV_STAR = <RAND, XRES1, KASME, AUTN>
      //we can only match on the first half of RES
      RES = <XRES1,RES2>
  in
  [ St_SEAF_5($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR)
  , In(<'syncfailure', AUTS>)
  ]
  --[ SEAFSyncFailure($SEAF)
    , SEAF_6recv_SYNCFAILURE(id)
    ]->
  [ // SEAF's run of the protocol is aborted (needs to be restarted)
    Out(senc{<'authrequest','syncfailure',RAND,AUTS>}SK)
  ]


rule SEAF_7send:
  let m = <'AuthConfirmation', imsi, $SEAF, RES>
      menc = senc{m}SK
  in
  [ St_SEAF_6($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR, RES) ]
  --[ SEAF_SendEncrypted($SEAF,m) ]->
  [ St_SEAF_7($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR, RES)
  , Out(menc)
  ]


rule HSS_7recv_NOSYNCFAILURE:
  let CK = KDF(<'f3', Kimsi, ~RAND>)
      IK = KDF(<'f4', Kimsi, ~RAND>)
      AK = KDF(<'f5', Kimsi, ~RAND>)
      SQNxorAK = SQN XOR AK
      XRES1 = KDF(<'f2', Kimsi, ~RAND, '1'>)
      XRES2 = KDF(<'f2', Kimsi, ~RAND, '2'>)
      RES = <XRES1,XRES2>
      m = <'AuthConfirmation', imsi, $SEAF, RES>
      menc = senc{m}SK
      KASME = KDF(<'KASME',CK,IK,$SEAF,SQNxorAK>)
```

```
  in
  [ St_HSS_2($HSS, id, imsi, Kimsi, $SEAF, SK, S, SQN, ~RAND)
  , In(menc)
  ]
  --[ Secret(<'HSS',$HSS>, KASME)
    , Honest(<'SEAFHSS',$SEAF,$HSS>)
    , Honest(<'IMSI',imsi>)
    , Commit($HSS,imsi,<'HSS','UE',KASME>)
    , Commit($HSS,imsi,<'HSS','UE',<'RAND',~RAND>>)
    , HSS_7recv_NOSYNCFAILURE(id)
    , HSSCOMPLETED(id)
    ]->
  [ St_HSS_3($HSS, id, imsi, Kimsi, $SEAF, SK, S, SQN, ~RAND) ]


// if the next counter would not be accepted by UE,
// HSS verifies the value of MACS
// and sets its sequence number to SQNMAX
rule HSS_7recv_SYNCFAILURE_UPDATECOUNTER:
  let MACS = KDF(<'f1star','AMF',~RAND,Kimsi,SQNMAX>)
      AK_STAR = KDF(<'f5star',~RAND,Kimsi>)
      SQNMAX = SQNMAXXORAKSTAR XOR AK_STAR
      AUTS = <SQNMAXXORAKSTAR, MACS>
      m = <'authrequest','syncfailure',~RAND,AUTS>
      menc = senc{m}SK
  in
  [ St_HSS_2($HSS, id, imsi, Kimsi, $SEAF, SK, S, SQN, ~RAND)
  , SQNHSS($HSS, imsi, CURRENTSQNBASEOFFSET)
  , !SQNBASEUEHSS(imsi, $HSS, ~SQNBASE)
  , In(menc)

  // Modeling trick to get rid of partial deconstructions
  , In(SQNMAXBASEOFFSET)
  ]
  --[ // as specified in the protocol,
      // HSS's counter is only updated if it is necessary
      LessThan(CURRENTSQNBASEOFFSET,SQNMAXBASEOFFSET)
    , HSSSyncFailure($HSS, imsi, ~SQNBASE,
                     CURRENTSQNBASEOFFSET, SQNMAXBASEOFFSET)
    , HSS_7recv_SYNCFAILURE_UPDATECOUNTER(id)

    // local check to verify that SQNMAX
    // is really an greater or equal SQNBASE
    , Eq(~SQNBASE+SQNMAXBASEOFFSET, SQNMAX)
    , USE_SQNHSS($HSS, imsi, ~SQNBASE, SQNMAXBASEOFFSET)
    ]->
  [ // HSS's run of the protocol is aborted (needs to be restarted)
    SQNHSS($HSS, imsi, SQNMAXBASEOFFSET)
  ]
```

```
rule SEAF_8send:
  let AV_STAR = <x, XRES, KASME, AUTN>
      confirmSEAF = KDF(<'MACconfirm',
                         KDF(<'KDFconfirm', KASME>),
                         'confirmSEAF'>)
  in
  [ St_SEAF_7($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR, RES) ]
  --[]->
  [ St_SEAF_8($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR, RES)
  , Out(confirmSEAF)
  ]

rule UE_8recv:
  let confirmSEAF = KDF(<'MACconfirm',
                        KDF(<'KDFconfirm', KASME>),
                        'confirmSEAF'>)
  in
  [ St_UE_4(imsi, Kimsi, ~id, $HSS, $SNid, RAND, KASME)
  , In(confirmSEAF)
  ]
  --[ Commit(imsi, $SNid, <'UE','SEAF',KASME>)
    , Commit(imsi, $HSS, <'UE','HSS',KASME>)
    , Honest(<'SEAFHSS',$SNid,$HSS>)
    , Honest(<'IMSI',imsi>)
    ]->
  [ St_UE_5(imsi, Kimsi, ~id, $HSS, $SNid, RAND, KASME) ]


rule UE_9send:
  let confirmUE = KDF(<'MACconfirm',
                      KDF(<'KDFconfirm', KASME>),
                      'confirmUE'>)
  in
  [ St_UE_5(imsi, Kimsi, ~id, $HSS, $SNid, RAND, KASME) ]
  --[ UECOMPLETED(~id) ]->
  [ St_UE_6(imsi, Kimsi, ~id, $HSS, $SNid, RAND, KASME)
  , Out(confirmUE)
  ]

rule SEAF_9recv:
  let AV_STAR = <x, XRES, KASME, AUTN>
      confirmUE = KDF(<'MACconfirm',
                      KDF(<'KDFconfirm', KASME>),
                      'confirmUE'>)
  in
  [ St_SEAF_8($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR, RES)
  , In(confirmUE)
```

```
  ]
  --[ Commit($SEAF, imsi, <'SEAF','UE',KASME>)
    , Honest(<'SEAFHSS',$SEAF,$HSS>)
    , Honest(<'IMSI',imsi>)
    , SEAFCOMPLETED(id)
    ]->
  [ St_SEAF_9($SEAF, id, imsi, $HSS, SK, ~S, AV_STAR, RES) ]

//########################RESTRICTIONS#################################

restriction Equality:
  "All x y #i. Eq(x,y)@#i ==> x=y"

restriction LessThan:
  "All x y #i. LessThan(x,y)@#i ==> Ex z. x + z = y"

restriction GreaterOrEqualThan:
  "All x y #i. GreaterOrEqualThan(x,y)@#i ==> not (Ex z. x + z = y)"

//########################SOURCES LEMMA################################

// this is necessary to remove the partial deconstructions
// AUTOMATIC PROOF (2017-09-09)
lemma types [sources]:
"
(
All RAND AUTN #i. OUT_SEAF_5send(<RAND,AUTN>)@i ==>
    (Ex #j. KU(RAND)@j & #j<#i) & (Ex #j. KU(AUTN)@j & #j<#i)
  | (Ex #j HSS XRES KASME S.
      HSS_4SendEncrypted(HSS, <S,<RAND, XRES, KASME, AUTN>>)@j
       & #j<#i)
)
&
(
All HSS S AV_STAR #i. HSS_4SendEncrypted(HSS, <S,AV_STAR>)@i ==>
    (Ex #j. KU(S)@j & #j<#i)
  | (Ex #j SEAF. SEAF_SendEncrypted(SEAF,S)@j & #j<#i)
)
"

//########################EXECUTABILITY LEMMAS#################

// FINDS EXPECTED TRACE AUTOMATICALLY (2017-09-09)
// ~1min
lemma key_setup_possible:
  exists-trace
  "
    Ex imsi SEAF HSS KASME #i #j #k.
      Secret(<'IMSI',imsi>, KASME)@i & Secret(<'SEAF',SEAF>,KASME)@j
```

```
      & Secret(<'HSS',HSS>,KASME)@k & (not imsi = SEAF)
      & (not imsi = HSS) & (not SEAF = HSS)
      & (Ex #q. Commit(SEAF, imsi, <'SEAF','UE',KASME>)@q )
      & (Ex #q. Commit(imsi, HSS, <'UE','HSS',KASME>)@q )
      & (Ex #q. Commit(imsi, SEAF, <'UE','SEAF',KASME>)@q )
    &
    not (Ex X #r. Reveal(X) @ r)
    &
    (All imsi1 imsi2 id1 id2 #i #j. (Create(imsi1,id1,'UE')@i
      & Create(imsi2,id2,'UE')@j ) ==> #i = #j)
    &
    (All SEAF1 SEAF2 id1 id2 #i #j. (Create(SEAF1,id1,'SEAF')@i
      & Create(SEAF2,id2,'SEAF')@j ) ==> #i = #j)
    &
    (All HSS1 HSS2 id1 id2 #i #j. (Create(HSS1,id1,'HSS')@i
      & Create(HSS2,id2,'HSS')@j ) ==> #i = #j)
    &
    (All HSS imsi SQNBASE SQNBASEOFFSET #i.
      HSS_SQN_NEXT(HSS, imsi, SQNBASE, SQNBASEOFFSET)@i
      ==> SQNBASEOFFSET = '1'+'1')
    &
    (All imsi HSS SQNBASE SQNBASEOFFSET #i.
       UE_SQN_NEXT(imsi, HSS, SQNBASE, SQNBASEOFFSET)@i
       ==> SQNBASEOFFSET = '1'+'1')
    &
    (not (Ex HSS imsi SQNBASE SQNOFFSET newSQNOFFSET #r.
          HAVOCSQNHSS(HSS, imsi,
                      SQNBASE,
                      SQNOFFSET,
                      newSQNOFFSET)@r)
     )
    &
    (not (Ex HSS imsi SQNBASE SQNBASEOFFSET newSQNBASEOFFSET #r.
      INCREASESQNOFFSETUE(imsi, HSS,
                          SQNBASE, SQNBASEOFFSET,
                          newSQNBASEOFFSET)@r
      )
    )
    "


// FINDS EXPECTED TRACE AUTOMATICALLY (2017-09-09)
// ~1min
lemma resync_possible:
  exists-trace
  "
  (
  Ex imsi HSS SEAF SQNBASE KASME SK
     idUE1 idSEAF1 idHSS1 idUE2 idSEAF2 idHSS2
```

```
 #i1 #i2 #i3 #i4 #i5 #i6 #i7 #i8 #i9 #i10
 #i11 #i12 #i13 #i14 #i15 #i16 #i17 #i18.
(not imsi = SEAF) & (not imsi = HSS) & (not SEAF = HSS)
&
// We start by setting UE out of sync with HSS
INCREASESQNOFFSETUE(imsi, HSS, SQNBASE, '1', '1'+'1'+'1')@i1
&
Shared_Symmetric_Key(SK,SEAF,HSS)@i2
&

// The first authentication attempt is
// supposed to result in a synchronization failure
Create(imsi,idUE1,'UE')@i3
&
Create(SEAF,idSEAF1,'SEAF')@i4
&
Create(HSS,idHSS1,'HSS')@i5
&
HSS_SQN_NEXT(HSS, imsi, SQNBASE, '1'+'1')@i6
&
UESyncFailure(imsi, HSS, SQNBASE, '1'+'1'+'1', '1'+'1')@i7
&
SEAFSyncFailure(SEAF)@i8
&
HSSSyncFailure(HSS, imsi, SQNBASE, '1'+'1', '1'+'1'+'1')@i9
&


// Having received the synchronization failure message
// with correct SQNMAX from UE,
// the following protocol run is successful

Create(imsi,idUE2,'UE')@i10
&
Create(SEAF,idSEAF2,'SEAF')@i11
&
Create(HSS,idHSS2,'HSS')@i12
&
HSS_SQN_NEXT(HSS, imsi, SQNBASE, '1'+'1'+'1'+'1')@i13
&
UE_5recv_SEQGOOD(idUE2)@i14
 & UE_SQN_NEXT(imsi, HSS, SQNBASE, '1'+'1'+'1'+'1')@i14
 & Secret(<'IMSI',imsi>, KASME)@i14
&
SEAF_6recv_NOSYNCFAILURE(idSEAF2)@i15
    & Secret(<'SEAF',SEAF>,KASME)@i15
&
HSS_7recv_NOSYNCFAILURE(idHSS2)@i16
    & Secret(<'HSS',HSS>,KASME)@i16
```

```
  &
  HSSCOMPLETED(idHSS2)@i16
  & UECOMPLETED(idUE2)@i17
  & SEAFCOMPLETED(idSEAF2)@i18

  // We add a few more restrictions to speed up
  // the search for the desired trace
  &
  (All imsi2 HSS2 sqnbase sqn sqnnew #j.
    INCREASESQNOFFSETUE(imsi2, HSS2, sqnbase, sqn, sqnnew)@j
    ==> #j = #i1
  )
  &
  (All id #j. SEAF_6recv_SYNCFAILURE(id)@j ==> id = idSEAF1)
  &
  (All id #j. SEAF_6recv_NOSYNCFAILURE(id)@j
    ==> id = idSEAF2 & #j=#i15)
  &
  (All id #j. HSS_7recv_SYNCFAILURE_UPDATECOUNTER(id)@j
    ==> id = idHSS1)
  &
  (All id #j. HSS_7recv_NOSYNCFAILURE(id)@j
    ==> id = idHSS2 & #j=#i16)
  &
  (All id #j. UE_5recv_SEQBAD(id)@j ==> id = idUE1)
  &
  (All id #j. UE_5recv_SEQGOOD(id)@j ==> id = idUE2 & #j=#i14)
  &
  (All imsi2 id #j. Create(imsi2,id,'UE')@j
    ==> (#j = #i3 | #j = #i10) &  imsi=imsi2)
  &
  (All SEAF2 id #j. Create(SEAF2,id,'SEAF')@j
    ==> (#j = #i4 | #j = #i11) & SEAF=SEAF2)
  &
  (All HSS2 id #j. Create(HSS2,id,'HSS')@j
    ==> (#j = #i5 | #j = #i12) & HSS=HSS2)
  &
  (All SK2 SEAF2 HSS2 #j. Shared_Symmetric_Key(SK2,SEAF2,HSS2)@j
    ==> #j=#i2)
  &
  (
   #i1 < #i2 & #i2 < #i3 & #i3 < #i4 & #i4 < #i5 & #i5 < #i6
   & #i6 < #i7 & #i7 < #i8 & #i8 < #i9 & #i9 < #i10
   & #i10 < #i11 & #i11 < #i12 & #i12 < #i13 & #i13 < #i14
   & #i14 < #i15 & #i15 < #i16 & #i16 < #i17 & #i17 < #i18
  )
)
  &
  (All imsi HSS SQNBASE SQNBASEOFFSETprev SQNBASEOFFSETnext #i.
```

```
      UE_SQN_PREV(imsi, HSS, SQNBASE, SQNBASEOFFSETprev)@i
      & UE_SQN_NEXT(imsi, HSS, SQNBASE, SQNBASEOFFSETnext)@i
     ==> SQNBASEOFFSETnext = SQNBASEOFFSETprev + '1'
    )
    &
    (not (Ex X #r. Reveal(X) @ r))
    &
    (not (Ex HSS imsi SQNBASE SQNOFFSET newSQNOFFSET #r.
      HAVOCSQNHSS(HSS, imsi, SQNBASE, SQNOFFSET, newSQNOFFSET)@r )
     )
  "


//#########################SECRECY HELPER LEMMAS#####################

// AUTOMATIC PROOF (2017-09-09)
lemma HELPERsecrecySQNBASE_UE [reuse]:
"
All imsi HSS SQNBASE #i. UEInitCounter(imsi, HSS, SQNBASE)@i
 ==>    (not (Ex #j. KU(SQNBASE)@j))
      | (Ex #j. Reveal(<'IMSI',imsi>)@j)
"

// AUTOMATIC PROOF (2017-09-09)
lemma HELPERsecrecySQNBASE_HSS [reuse]:
"
All imsi HSS SQNBASE #i. HSSInitCounter(HSS, imsi, SQNBASE)@i
 ==>   (not (Ex #j. KU(SQNBASE)@j))
     | (Ex #j. Reveal(<'IMSI',imsi>)@j)
"

// AUTOMATIC PROOF (2017-09-09)
lemma HELPERsecrecyKimsiDerivations [reuse]:
"
All imsi HSS Kimsi #i. REGISTER_IMSI(imsi, HSS, Kimsi)@i
==>
  (not (Ex f X #j. not(f = 'f1') & not(f = 'f2')
   & KU(KDF(<f, Kimsi, X>))@j))
  | (Ex #j. Reveal(<'IMSI',imsi>)@j)
  // note that f1 is used for the MAC contained in AUTN,
  // so the attacker can learn this by revealing
  // the secret channel between SEAF and HSS
  // moreover, note that f2 is used for the challenge
  // response XRES, so the attacker can learn this
  // by revealing the secret channel between SEAF and HSS
"
```

```
//#######################SECRECY LEMMAS############################

// AUTOMATIC PROOF (2017-09-09)
// <1min
// reuses the HELPERsecrecy lemmas
lemma secrecy_UE:
  "All A x #i.
    Secret(<'IMSI',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(X)@r & Honest(X) @i)"

// AUTOMATIC PROOF (2017-09-09)
// <1min
// reuses the HELPERsecrecy lemmas
lemma secrecy_HSS:
  "All A x #i.
    Secret(<'HSS',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(X)@r & Honest(X) @i)"

// AUTOMATIC PROOF (2017-09-09)
// <1min
// reuses the HELPERsecrecy lemmas
lemma secrecy_SEAF:
  "All A x #i.
    Secret(<'SEAF',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(X)@r & Honest(X) @i)"


// FINDS _NO_ ATTACK AUTOMATICALLY (see following lemma)
lemma secrecy_PFS_UE:
  "All A x #i.
    Secret(<'IMSI',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(<'IMSI',X>)@r
            & Honest(<'IMSI',X>) @i & r < i)"

// (this enforces an attack without
//   revealing the key shared between SEAF and HSS)
// FINDS ATTACK AUTOMATICALLY (2017-09-09)
// ~1min
// reuses the HELPERsecrecy lemmas
lemma secrecy_PFS_UE_special:
  "
   (not (Ex SEAF HSS #j. Reveal(<'SEAFHSS',SEAF,HSS>)@j))
  ==>
  All A x #i.
    Secret(<'IMSI',A>,x) @i ==>
```

```
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(<'IMSI',X>)@r
           & Honest(<'IMSI',X>) @i & r < i)"

// FINDS _NO_ ATTACK AUTOMATICALLY (see following lemma)
lemma secrecy_PFS_HSS:
  "All A x #i.
    Secret(<'HSS',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(<'IMSI',X>)@r
           & Honest(<'IMSI',X>) @i & r < i)"

// (this enforces an attack without
//   revealing the key shared between SEAF and HSS)
// FINDS ATTACK AUTOMATICALLY (2017-09-09)
// ~1min
// reuses the HELPERsecrecy lemmas
lemma secrecy_PFS_HSS_special:
  "
   (not (Ex SEAF HSS #j. Reveal(<'SEAFHSS',SEAF,HSS>)@j))
  ==>
  All A x #i.
    Secret(<'HSS',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(<'IMSI',X>)@r
           & Honest(<'IMSI',X>) @i & r < i)"

// FINDS _NO_ ATTACK AUTOMATICALLY (see following lemma)
lemma secrecy_PFS_SEAF:
  "All A x #i.
    Secret(<'SEAF',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(<'IMSI',X>)@r
           & Honest(<'IMSI',X>) @i & r < i)"

// (this enforces an attack without
//   revealing the key shared between SEAF and HSS)
// FINDS ATTACK AUTOMATICALLY (2017-09-09)
// ~1min
// reuses the HELPERsecrecy lemmas
lemma secrecy_PFS_SEAF_special:
  "
   (not (Ex SEAF HSS #j. Reveal(<'SEAFHSS',SEAF,HSS>)@j))
  ==>
  All A x #i.
    Secret(<'SEAF',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(<'IMSI',X>)@r
           & Honest(<'IMSI',X>) @i & r < i)"
```

```
//########################COUNTER HELPER LEMMAS####################

//Before proving authentication properties,
// we need a few properties about the counters of UE and HSS
// AUTOMATIC PROOF (2017-09-09)
// <1min
lemma HSScounterInit [use_induction, reuse]:
"
All imsi HSS SQNBASE SQNBASEOFFSET #i.
  USE_SQNHSS(HSS, imsi, SQNBASE, SQNBASEOFFSET)@i
  ==> (Ex #j. HSSInitCounter(HSS, imsi, SQNBASE)@j & #j<#i)
"


// AUTOMATIC PROOF (2017-09-09)
// <1min
lemma UEcounterINIT [use_induction, reuse]:
"
(All imsi HSS SQNBASE SQNBASEOFFSET #i.
  UE_SQN_NEXT(imsi, HSS, SQNBASE, SQNBASEOFFSET)@i
  ==> (Ex #j. UEInitCounter(imsi, HSS, SQNBASE)@j & #j<#i))
&
(All imsi HSS SQNBASE SQNBASEOFFSET #i.
 UE_SQN_NOCHANGE(imsi, HSS, SQNBASE, SQNBASEOFFSET)@i
 ==> (Ex #j. UEInitCounter(imsi, HSS, SQNBASE)@j & #j<#i))
&
 (All imsi HSS SQNBASE SQNBASEOFFSET newSQNBASEOFFSET #i.
   INCREASESQNOFFSETUE(imsi, HSS,
                       SQNBASE,
                       SQNBASEOFFSET,
                       newSQNBASEOFFSET)@i
   ==> (Ex #j. UEInitCounter(imsi, HSS, SQNBASE)@j & #j<#i)
  )
"

// reuses UEcounterINIT
// NO AUTOMATIC PROOF, but provable with the provided oracle
// ~10min
lemma HELPERuniqueUECounter
[use_induction, reuse,
 hide_lemma=HSScounterInit,
 hide_lemma=uniqueHSSCounter,
 hide_lemma=HELPERsecrecySQNBASE_UE,
 hide_lemma=HELPERsecrecySQNBASE_HSS,
 hide_lemma=HELPERsecrecyKimsiDerivations]:
"
(All imsi HSS SQNBASE SQNBASEOFFSET1 SQNBASEOFFSET2 #i #j.
  UE_SET_SQN(imsi, HSS, SQNBASE, SQNBASEOFFSET1)@i
```

```
  & UE_SET_SQN(imsi, HSS, SQNBASE, SQNBASEOFFSET2)@j
  & #i<#j
  ==> (Ex dif. SQNBASEOFFSET1+dif=SQNBASEOFFSET2 )
 )
&
(All imsi HSS SQNBASE SQNBASEOFFSET1 SQNBASEOFFSET2 #i #j.
  UE_SET_SQN(imsi, HSS, SQNBASE, SQNBASEOFFSET1)@i
  & UE_SQN_NOCHANGE(imsi, HSS, SQNBASE, SQNBASEOFFSET2)@j
  & #i<#j
  ==> SQNBASEOFFSET1=SQNBASEOFFSET2
      | (Ex dif. SQNBASEOFFSET1+dif = SQNBASEOFFSET2)
  )
&
(All imsi HSS SQNBASE SQNBASEOFFSET1 SQNBASEOFFSET2 #i #j.
  UE_SQN_NOCHANGE(imsi, HSS, SQNBASE, SQNBASEOFFSET1)@i
  & UE_SET_SQN(imsi, HSS, SQNBASE, SQNBASEOFFSET2)@j
  & #i<#j
  ==> (Ex dif. SQNBASEOFFSET1+dif = SQNBASEOFFSET2)
 )
&
(All imsi HSS SQNBASE SQNBASEOFFSET1 SQNBASEOFFSET2 #i #j.
  UE_SQN_NOCHANGE(imsi, HSS, SQNBASE, SQNBASEOFFSET1)@i
  & UE_SQN_NOCHANGE(imsi, HSS, SQNBASE, SQNBASEOFFSET2)@j
  & #i<#j
 ==> SQNBASEOFFSET1=SQNBASEOFFSET2
     | (Ex dif. SQNBASEOFFSET1+dif = SQNBASEOFFSET2)
 )
"

//reuses HELPERuniqueUECounter
// AUTOMATIC PROOF (2017-09-09)
// <1min
lemma uniqueUECounter
[reuse,
 hide_lemma=HSScounterInit,
 hide_lemma=uniqueHSSCounter]:
"
All imsi HSS SQNBASE SQNBASEOFFSET #i #j.
  UE_SQN_NEXT(imsi, HSS, SQNBASE, SQNBASEOFFSET)@i
   & UE_SQN_NEXT(imsi, HSS, SQNBASE, SQNBASEOFFSET)@j
  ==> #i=#j
"

//########################SQN SECRECY LEMMAS########################

// reuses the HELPERsecrecy lemmas
// AUTOMATIC PROOF (2017-09-09)
// <1min
lemma secrecySQN_UE
```

```
[hide_lemma=HSScounterInit,
 hide_lemma=UEcounterINIT,
 hide_lemma=HELPERuniqueUECounter]:
  "All A x #i.
    SecretSQN(<'IMSI',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(X)@r & Honest(X) @i)"

// reuses the HELPERsecrecy lemmas
// AUTOMATIC PROOF (2017-09-09)
// <1min
lemma secrecySQN_HSS
[hide_lemma=HSScounterInit,
 hide_lemma=UEcounterINIT,
 hide_lemma=HELPERuniqueUECounter]:
  "All A x #i.
    SecretSQN(<'HSS',A>,x) @i ==>
    not (Ex #j. K(x)@j)
        | (Ex X #r. Reveal(X)@r & Honest(X) @i)"


//####################AUTHENTICATION FROM UE#########################

// PROOF USING THE PROVIDED ORACLE (2017-09-09)
// ~1min
lemma injectiveagreementUE_SEAF
[hide_lemma=HSScounterInit,
 hide_lemma=UEcounterINIT,
 hide_lemma=HELPERuniqueUECounter]:
  "All a b t #i.
   Commit(a,b,<'UE','SEAF',t>) @i
    ==> (Ex #j. Running(b,a,<'UE','SEAF',t>) @j
        & j < i
        & not (Ex a2 b2 #i2. Commit(a2,b2,<'UE','SEAF',t>) @i2
                          & not (#i2 = #i)))
              | (Ex X #r. Reveal(X)@r & Honest(X) @i)"


// PROOF USING THE PROVIDED ORACLE (2017-09-09)
// ~1min
lemma injectiveagreementUE_HSS
[hide_lemma=HSScounterInit,
 hide_lemma=UEcounterINIT,
 hide_lemma=HELPERuniqueUECounter]:
  "All a b t #i.
    Commit(a,b,<'UE','HSS',t>) @i
    ==> (Ex #j. Running(b,a,<'UE','HSS',t>) @j
        & j < i
        & not (Ex a2 b2 #i2. Commit(a2,b2,<'UE','HSS',t>) @i2
```

```
                        & not (#i2 = #i)))
              | (Ex X #r. Reveal(X)@r & Honest(X) @i)"


//####################AUTHENTICATION FROM SEAF#####################

// PROOF USING THE PROVIDED ORACLE (2017-09-09)
// ~1min
lemma injectiveagreementSEAF_UE
[hide_lemma=HSScounterInit,
 hide_lemma=UEcounterINIT,
 hide_lemma=HELPERuniqueUECounter]:
  "All a b t #i.
    Commit(a,b,<'SEAF','UE',t>) @i
    ==> (Ex #j. Running(b,a,<'SEAF','UE',t>) @j
              & j < i
              & not (Ex a2 b2 #i2.
                      Commit(a2,b2,<'SEAF','UE',t>) @i2
                      & not (#i2 = #i)))
              | (Ex X #r. Reveal(X)@r & Honest(X) @i)"

//####################AUTHENTICATION FROM HSS#####################

// AUTOMATIC PROOF (2017-09-09)
// <1min
lemma weakagreementHSS_UE
[hide_lemma=HSScounterInit,
 hide_lemma=UEcounterINIT,
 hide_lemma=HELPERuniqueUECounter]:
  "All a b t #i.
    Commit(a,b,<'HSS','UE',t>) @i
    ==> (Ex t2 #j. Running(b,a,t2) @j)
        | (Ex X #r. Reveal(X)@r & Honest(X) @i)"

// FINDS ATTACK AUTOMATICALLY (2017-09-09)
lemma noninjectiveagreementHSS_UE
[hide_lemma=HSScounterInit,
 hide_lemma=UEcounterINIT,
 hide_lemma=HELPERuniqueUECounter]:
  "All a b t #i.
    Commit(a,b,<'HSS','UE',t>) @i
    ==> (Ex #j. Running(b,a,<'HSS','UE',t>) @j)
        | (Ex X #r. Reveal(X)@r & Honest(X) @i)"

// AUTOMATIC PROOF (2017-09-09)
// <1min
lemma injectiveagreementHSS_UE_ONLYRAND
[hide_lemma=HSScounterInit,
 hide_lemma=UEcounterINIT,
```

```
 hide_lemma=HELPERuniqueUECounter]:
  "All a b t #i.
    Commit(a,b,<'HSS','UE',<'RAND',t>>) @i
    ==> (Ex #j. Running(b,a,<'HSS','UE',<'RAND',t>>) @j
              & j < i
              & not (Ex a2 b2 #i2.
                      Commit(a2,b2,<'HSS','UE',<'RAND',t>>) @i2
                       & not (#i2 = #i)))
              | (Ex X #r. Reveal(X)@r & Honest(X) @i)"

end
```

Appendix B

Abbreviations

AKA	Authentication and Key Agreement
AMF	Authentication Management Field
ARPF	Authentication Credential Repository and Processing Function
AUSF	Authentication Server Function
AUTN	Authentication token
CK	Cipher Key
EAP	Extensible Authentication Protocol
HN	Home Network
HSS	Home Subscriber Server
IK	Integrity Key
IMEI	International Mobile Station Equipment Identity
IMSI	International Mobile Subscriber Identity
KDF	Key Derivation Function
NAS	Non Access Stratum
NG	NextGen
PSK	Pre-shared Key
SEAF	Security Anchor Function
SIM	Subscriber Identity Module
SN	Serving Network
SNid	Serving Network Identifier
SQN	Sequence Number
UE	User Equipment
UICC	Universal Integrated Circuit Card
UMTS	Universal Mobile Telecommunication System
USIM	Universal Subscriber Identity Module
XRES	Expected Response

Bibliography

[1] Third Generation Partnership Project (3GPP). http://www.3gpp.org/.

[2] 3GPP. Formal Analysis of the 3G Authentication Protocol. TR 33.902 V4.0.0, 3rd Generation Partnership Project (3GPP), Sept. 2001.

[3] 3GPP. 3G security; Security architecture. TS 33.102 V14.1.0, 3rd Generation Partnership Project (3GPP), Mar. 2017.

[4] 3GPP. 3GPP System Architecture Evolution (SAE); Security architecture. TS 33.401 V14.2.0, 3rd Generation Partnership Project (3GPP), Mar. 2017.

[5] 3GPP. Security Architecture and Procedures for 5G System. TS 33.501 V0.3.0, 3rd Generation Partnership Project (3GPP), Aug. 2017.

[6] 3GPP. Study on the security aspects of the next generation system. Draft TR 33.899 V1.3.0, 3rd Generation Partnership Project (3GPP), Aug. 2017.

[7] B. Aboba et al. Extensible Authentication Protocol (EAP). RFC 3748, RFC Editor, June 2004.

[8] B. Aboba, D. Simin, and P. Eronen. Extensible Authentication Protocol (EAP) Key Management Framework. RFC 5247, RFC Editor, August 2008.

[9] S. Alt, P.-A. Fouque, G. Macario-Rat, C. Onete, and B. Richard. A Cryptographic Analysis of UMTS/LTE AKA. In *International Conference on Applied Cryptography and Network Security*, pages 18–35. Springer, 2016.

[10] M. Arapinis, L. Mancini, E. Ritter, M. Ryan, N. Golde, K. Redon, and R. Borgaonkar. New Privacy Issues in Mobile Telephony: Fix and Verification. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 205–216. ACM, 2012.

[11] J. Arkko and H. Haverinen. Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA). RFC 4187, RFC Editor, January 2006.

[12] B. Blanchet et al. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW*, pages 82–96, 2001.

[13] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *Advances in Cryptology — CRYPTO 2001*, pages 213–229. Springer, 2001.

[14] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology — EUROCRYPT 2003*, pages 416–432. Springer, 2003.

[15] C. Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *Cryptography and Coding*, pages 360–363. Springer, 2001.

[16] C. J. Cremers. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In *Computer Aided Verification CAV 2008*. Springer, 2008.

[17] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on information theory*, 29(2):198–208, 1983.

[18] S. Escobar, C. Meadows, and J. Meseguer. Maude-NPA: Cryptographic Protocol Analysis Modulo Equational Properties. In *Foundations of Security Analysis and Design V*, pages 1–50. Springer, 2009.

[19] S. Escobar, R. Sasse, and J. Meseguer. Folding Variant Narrowing and Optimal Variant Termination. In *Proceedings of the 8th International Conference on Rewriting Logic and Its Applications*, WRLA'10, pages 52–68. Springer-Verlag, 2010.

[20] ETSI. GSM Technical Specification - GSM 04.08. Draft TR GSM 04.08 V5.3.0, European Telecommunications Standards Institute (ETSI), July 1996.

[21] V. Fajardo, J. Arkko, J. Loughney, and G. Zorn. Diameter Base Protocol. RFC 6733, RFC Editor, October 2012.

[22] C. Gentry and Z. Ramzan. Identity-Based Aggregate Signatures. In *Public Key Cryptography - PKC 2006*. Springer, 2006.

[23] P. E. J. Arkko, V. Lehtovirta. Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA'). RFC 5448, RFC Editor, January 2009.

[24] D. Lanzenberger. 5G Tamarin Models. https://www.ethz.ch/content/dam/ethz/special-interest/infk/inst-infsec/information-security-group-dam/research/software/5G_lanzenberger.zip.

[25] G. Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information processing letters*, 56(3):131–133, 1995.

[26] G. Lowe. A hierarchy of authentication specifications. In *Computer security foundations workshop, 1997. Proceedings., 10th*, pages 31–43. IEEE, 1997.

[27] S. Meier. *Advancing automated security protocol verification*. PhD thesis, ETH Zurich, 2013.

[28] P. O'Hanlon, R. Borgaonkar, and L. Hirschi. Mobile Subscriber WiFi Privacy. In *IEEE Symposium on Security and Privacy 2017 workshop on Mobile Security Technologies*. IEEE, 2017.

[29] B. Schmidt. *Formal analysis of key exchange protocols and physical protocols*. PhD thesis, ETH Zurich, 2012.

[30] B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *Computer Security Foundations Symposium (CSF), 2012 IEEE 25th*, pages 78–94. IEEE, 2012.

[31] A. Shamir et al. Identity-Based Cryptosystems and Signature Schemes. In *Advances in Cryptology — CRYPTO 1984*, pages 47–53. Springer, 1984.

[32] Tamarin prover. https://tamarin-prover.github.io/.