DEGREE PROJECT IN COMMUNICATION SYSTEMS, SECOND LEVEL
STOCKHOLM, SWEDEN 2014

Formal Verification of a LTE Security Protocol for Dual-Connectivity

An Evaluation of Automatic Model Checking Tools

KATHARINA PFEFFER

KTH ROYAL INSTITUTE OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY

Formal Verification of a LTE Security Protocol for Dual-Connectivity

An Evaluation of Automatic Model Checking Tools

Katharina Pfeffer

Master of Science Thesis

Communication Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

17 July 2014

Supervisor and Examiner: Professor Gerald Q. Maguire Jr.
Industrial Supervisors: Karl Norrman, Noamen Ben Henda

Abstract

Security protocols are ubiquitously used in various applications with the intention to ensure secure and private communication. To achieve this goal, a mechanism offering reliable and systematic protocol verification is needed. Accordingly, a major interest in academic research on formal methods for protocol analysis has been apparent for the last two decades. Such methods formalize the operational semantics of a protocol, laying the base for protocol verification with automatic model checking tools.

So far, little work in this field has focused on protocol standardization. Within this thesis a security analysis of a novel Authenticated Key-Exchange (AKE) protocol for secure association handover between two Long-Term Evolution (LTE) base stations (which support dual-connectivity) is carried out by applying two state-of-the-art tools for automated model checking (Scyther and Tamarin Prover). In the course of this a formal protocol model and tool input models are developed. Finally, the suitability of the used tools for LTE protocol analysis is evaluated.

The major outcome is that none of the two applied tools is capable to accurately model and verify the dual-connectivity protocol in such detail that it would make them particularly useful in the considered setting. The reason for this are restrictions in the syntax of Scyther and a degraded performance of Tamarin when using complex protocol input models. However, the use of formal methods in protocol standardization can be highly beneficial, since it implies a careful consideration of a protocol's fundamentals. Hence, formal methods are helpful to improve and structure a protocol's design process when applied in conjunction to current practices.

Keywords: security, authenticated key-exchange, 3GPP, LTE, formal methods, protocol verification, automated model checking

Sammanfattning

Säkerhetsprotokoll används i många typer av applikationer för att säkerställa säkerhet och integritet för kommunikation. För att uppnå detta mål behövs en behövs mekanismer som tillhandahåller pålitlig och systematisk verifiering av protokollen. Därför har det visats stort akademiskt intresse för forskning inom formell verifiering av säkerhetsprotokoll de senaste två decennierna. Sådana metoder formaliserar protokollsemantiken, vilket lägger grunden till automatiserad verifiering med modellverifieringsverktyg.

Än så länge har det inte varit stort focus på praktiska tillämpningar, som t.ex. hur väl metoderna fungerar för de problem som dyker upp under en standardiseringsprocess. I detta examensarbete konstrueras en formell modell för ett säkerhetsprotokoll som etablerar en säkerhetsassociation mellan en terminal och två Long-Term Evolution (LTE) basstationer i ett delsystem kallat Dual Connectivity. Detta delsystem standardiseras för närvarande i 3GPP. Den formella modellen verifieras sedan med bästa tillgängliga verktyg för automatiserad modellverifiering (Scyther och Tamarin Prover). För att åstadkomma detta har den formella modellen implementerats i inmatningsspråken för de två verktygen. Slutligen ha de två verktygen evaluerats.

Huvudslutsatsen är att inget av de två verktygen tillräckligt väl kan modellera de koncept där maskinstödd verifiering som mest behövs. Skälen till detta är Scythers begränsade syntax, och Tamarins begränsade prestanda och möjlighet att terminera för komplexa protokollmodeller. Trots detta är formella metoder andvändbara i standardiseringsprocessen eftersom de tvingar fram väldigt noggrann granskning av protokollens fundamentala delar. Därför kan formella metoder bidra till att förbättra strukturen på protokollkonstruktionsprocessen om det kombineras med nuvarande metoder.

Nyckelord: säkerhet, autentiserad etablering av nycklar, 3GPP, LTE, formella metoder, protokollverifiering, automatiserad modellverifiering

Acknowledgements

I would like to sincerely thank my academic supervisor Prof. Gerald Q. Maguire Jr. for his outstanding guidance throughout my thesis work. His accurate and constructive feedback, strengthened by his immense professional experience, helped me to improve the content of my thesis as well as the academic writing.

Moreover, I am highly grateful to my two supervisors at Ericsson, Karl Norrman and Noamen Ben Henda for their continuous support and the fruitful discussions I had with them. Karl's precise, subject-specific comments made me try to get to the bottom of things. Noamen helped out a lot with his holistic view and analytic way of thinking, especially when I got stuck with the model checking tools.

I thank the whole team at the Ericsson Research Area Security for integrating me in their collaborative, nice working environment.

Finally, I want to express my gratefulness to my friends and family for their great support throughout this thesis work and my whole study.

Contents

List of Figures

List of Tables

Listings

List of Acronyms and Abbreviations

AKE	Authenticated Key-Exchange
DH	Diffie-Hellman
DoS	Denial of Service
DRB	Data Radio Bearer
E-UTRAN	Evolved Universal Terrestrial Radio Access Network
eNB	E-UTRAN NodeB
EPC	Evolved Packet Core
HMAC	Hash-based Message Authentication Code
IV	Initialization Vector
KDF	Key Derivation Function
LTE	Long-Term Evolution
LTK	Long-Term Key Reveal
MAC	Message Authentication Code
MeNB	Master E-UTRAN NodeB
PKE	Public Key Exchange
PDCP	Packet Data Convergence Protocol
SeNB	Secondary E-UTRAN NodeB
SCC	Small Cell Counter
UE	User Equipment

Chapter 1

Introduction

Security protocols, frequently used in various applications of today's communication networks, sometimes contain flaws which are initially detected only after standardization. As a result, there has been a great interest in research on formal protocol verification during the last decades, aiming at reliably evaluating protocol security, detecting vulnerabilities automatically, and thus enabling the standards bodies to avoid standardizing a protocol with security flaws.

Protocol security can only be verified with respect to possible attacks, which are numerous and hard to predict in the absence of real adversaries. In this regard, the impersonation attack on the Needham-Schroeder protocol [1] is typically chosen as example, to point out that protocols can be insecure although all underlying cryptographic properties hold.

Accordingly, an interest arises for formal protocol modeling combined with automated model checking. The latter turns out to be a sophisticated task of verification and testing, since an unbounded number of new sessions* can be created during a protocol's execution, leading to an infinite search space. Several proposals have been made to deal with this issue by either limiting the number of sessions or applying heuristics and abstractions. [2, 3]

Although research in formal protocol verification is increasing, its utilization is still limited within the protocol standardization process. However, this field could be enriched by the use of automated model checkers, not in the least due to the fact that real attack patterns are outputted if weaknesses exist, which can be beneficial for discovering and avoiding vulnerabilities early in the protocol's design process.

* A session is a single partial execution of a protocol.

Within this thesis a novel Authenticated Key-Exchange (AKE) protocol, which is currently in its 3GPP standardization process, will be formalized and verified with two different state-of-the-art model checking tools, namely Scyther [4] and Tamarin Prover [5]. The main purpose of this protocol is the secure handover of a connection from one LTE base station to another base station. Such a protocol can be beneficial since it enables load balancing between base stations while still maintaining security.

1.1 Problem Description and Context

This thesis targets the investigation of existing approaches for formal automated security protocol verification and the evaluation of their suitability for verifying LTE protocols. In the course of this, two different state-of-the-art model checking tools (Scyther and Tamarin Prover) will be applied to a newly designed AKE protocol for secure handover between LTE terminals that support dual-connectivity. This protocol is currently in its 3GPP standardization process. The initial protocol design model will be constructed with regard to existing 3GPP drafts of the dual-connectivity protocol. This design model will be evaluated and possibly extended or improved with regards to the verification results.

Each formal verification result can only be seen as a verification in view of a certain formal protocol model. This model should describe the protocol's execution, the adversary assumptions, and the required security properties. Accordingly, a formal model of the dual-connectivity security protocol will initially be constructed within this thesis, laying the base for modeling the protocol using input languages, specifically for Scyther and Tamarin Prover. After running the formal verification, the advantages, limitations, performance and the usability of the applied model checking tools will be evaluated. Moreover, an assessment of the general usefulness of formal methods in a protocol's standardization process will be carried out.

Summarized, the goals of this thesis can be described as:

1. Design model
2. Formal model
3. Tool models and protocol verification in Scyther and Tamarin
4. Protocol design refinement
5. Evaluation of the applied tools

1.2 Structure of this Thesis

Chapter 1 describes the relevance of formal protocol verification and the main goals of this thesis. Chapter 2 provides a survey of AKE protocols, discusses several AKE protocol architectures, describes their basic cryptographic properties and operations, as well as AKE design goals with respect to potential threats. Following this, Chapter 3 deals with the issue of formal protocol verification by initially describing the basic requirements for creating a formal model and afterwards, discusses approaches to verify protocol models automatically and describes state-of-the art model checking tools.

Chapter 4 offers an overview of related work in the field of automated model checking. In Chapter 5 the methodology that has been applied is discussed. Chapter 6 describes the design modeling and subsequent formalizing and verification of the dual-connectivity protocol, followed by an evaluation of the applied model checking tools in Chapter 7. Finally, Chapter 8 concludes with a discussion of possible future work and reflections of economic and social issues.

Chapter 2

Authenticated Key-Exchange (AKE) Protocols

Key-Exchange Protocols aim to establish symmetric session keys between a defined group of entities in order to secure subsequent communication. Furthermore, authentication of each of those entities involved in the key establishment is usually desired, hence AKE protocols typically combine *Key Establishment Protocols* and *Entity Authentication Protocols*.

AKE protocols are widely used in today's communication networks, building the base for securing electronic communication, since secure session key establishment and assurance about the identities of involved entities are prerequisites for the reliability of any subsequent cryptographic operation. Accordingly, numerous research efforts have been carried out on AKE protocols, leading to more and more sophisticated protocols, enabling security claims even in the presence of strong adversaries, who can reveal session keys, long-term-private keys, and compromise random number generators. [6]

To give a proper overview of AKE protocols, this chapter will initially discuss basic security properties and cryptographic operations with regard to possible attacks. Afterwards, a survey of AKE design goals will be conducted, laying the base for the dual-connectivity LTE protocol design, carried out within this thesis. Boyd and Mathuria's book [7], dealing with basic concepts of AKE protocols, was used as the main reference for this chapter, since it provides a thorough discussion of the topic.

2.1 AKE Protocol Architecture

AKE protocols can be classified based on three criteria:

1. Which keys have already been established?
2. How is the key establishment carried out?
3. How many users are involved in the AKE procedure?

Regarding the first question, principals can either already maintain a shared secret key or a trusted third party can be used to obtain one. If a trusted third party is used, then a mechanism to secure communication between this party and the protocol participants is needed. This can conceivably be achieved by using a Public Key Infrastructure (PKI) and signed certificates. Alternatively, the participants could already share a secret with the third party.

A criterion for categorization when analyzing the procedure of key establishment is whether a protocol is mainly concerned with key transport or key agreement. *Key Transport Protocols* are defined by one participant who generates the key and transfers it to the other users. Alternatively, *Key Agreement Protocols* establish a session key as a function of inputs provided by several participants, as for instance occurs with the Diffie-Hellman (DH) algorithm [8], where each participant inputs nonces and applies modulo operations to each in order to compute the final shared secret key. Moreover, protocols can have features of both, key transport and key agreement protocols, thus they are *Hybrid Protocols*. For instance, the session key can be derived by computing a function of multiple, but not all users' inputs. Thus, the protocol appears to be a key agreement protocol from the viewpoint of one subset of users, while it is seen as transport protocol from the viewpoint of another subset of users. [7]

2.2 Cryptography

This section will cover basic cryptographic properties and related cryptographic operations, which may be applied by security protocols in order to achieve those properties. Cryptographic operations can be implemented by various algorithms. As this thesis deals with protocols on a conceptual level, specific algorithms are not considered, hence they will be neglected in the following discussion.

2.2.1 Symmetric and Asymmetric Encryption

Confidentiality means that data is only available to entities authorized to use it. Such a demand can be met by encrypting messages with a key, assuring that only entities in possession of the corresponding decryption key can read them.

An encryption scheme consists of three sets: the key set K, the message set M, and the ciphertext set C. Furthermore, three algorithms are utilized:

1. A **Key Generation Algorithm** which outputs a valid encryption key $\mathrm{k} \in K$ and decryption key $\mathrm{k}^{-1} \in K$.

2. An **Encryption Algorithm** which takes an argument $\mathrm{m} \in M$ and an encryption key $\mathrm{k} \in K$ and outputs $\mathrm{c} \in C$, defined as $\mathrm{c} = \mathrm{E}_k(\mathrm{m})$. The encryption process should be randomized, for example by the addition of a nonce to the inputs or prepending a nonce to the argument m, ensuring that the same message never leads to the same m, and hence a given message never leads to the same c.

3. A **Decryption Function** which takes an argument $\mathrm{c} \in C$ and a decryption key $\mathrm{k}^{-1} \in K$ and outputs $\mathrm{m} \in M$, defined as $\mathrm{m} = \mathrm{D}_{k^{-1}}(\mathrm{c})$. It is a required that $\mathrm{D}_{k^{-1}}(\mathrm{E}_k(\mathrm{m})) = \mathrm{m}$. If a a nonce was added to the set of inputs, it has to be input to both the encryption and decryption functions. If the nonce was prepended to m before encryption, then it must be removed after decryption.

In a *symmetric* encryption scheme, the encryption and decryption keys are equal, fulfilling the equation $\mathrm{k} = \mathrm{k}^{-1}$. In contrast, an *asymmetric* encryption scheme requires different keys (generally referred to as public and private keys) for encryption and decryption, where it is assumed to be computationally hard to compute the private key from the public key.

Two properties should always hold for an encryption scheme, namely semantic security and non-malleability. Semantic security demands that anything which can be efficiently computed given a cipher text, can also be efficiently computed without it. Non-malleability concerns the infeasibility of taking an existing cipher text and transforming it into a related text without knowledge of the plain text. [7]

2.2.2 Hash Functions

A hash function is a function $f\colon X \to Y$, which maps an input bit-string x of arbitrary finite length to an output bit-string y of fixed length (*compression*), whereby *ease of computation* of $y = f(x)$ has to be guaranteed.

In addition, the following potential properties may be fulfilled, which are used to classify various hash functions:

1. *collision resistance* - a hash-function should ideally never produce the same output twice when applying f to two different inputs. As this property is mathematically infeasible, hash functions are chosen to produce a very high improbability of such collisions. Thus, it is computationally hard to find two different inputs *x* and *x'*, which hash to the same output and *h(x) = h(x')*.

2. *preimage resistance* - for any given output *y*, it should be computationally infeasible to find the related input *x'* which hashes to the given output, that is *h(x') = y*. Accordingly, hash functions are also referred to as *one-way functions*.

3. *2nd-preimage resistance* - given any input value *x*, it is computationally hard to find a second input *x'*, which hashes to the same output as *x*, so that *h(x) = h(x')* holds. [7, 9]

2.2.2.1 Message Authentication Codes (MACs)

Message Authentication Codes (MACs) are specific hash functions that include a secret key *k* in their operation, therefore they are also called *keyed hash functions*. Just as for un-keyed hash functions, MACs attempt to assure the two properties of computational ease to compute $MAC_k(m)$ when key *k* and the message *m* are known and computational resistance against creating new MACs for any input when any number of text-MAC pairs (as well as optionally *k*) are given.

When constructing input strings to MAC functions it has to be carefully considered how the secret key is included, otherwise various attacks become feasible. For instance, it may become possible to append data to a message without knowledge of the secret key or create MACs for new input values, when the concatenation of the key and the message string is chosen poorly.

A sophisticated version of a MAC that meets this challenge is the **Hash-based MAC (HMAC)**. HMACs compute the hash of a message x as $HMAC(x) = h(k\|p_1\|h(k\|p_2\|x))$*, where p_1 and p_2 are XORed with k. The strings p_1 and p_2 are used for padding k to the required block size of the compression function. [9]

* Within this thesis, the double vertical bar (‖) is used to denote concatenation

2.2.2.2 Integrity and Data Origin Authentication

The cryptographic property of *Integrity* demands that if data has been modified by adversaries during transmission, this modification is detected. This is usually linked to *Data Origin Authentication*, assuring that data came from its stated source. Data origin authentication can be carried out only on messages which have not been altered, otherwise they would have a different source.

The usage of MACs ensures integrity and data origin authentication since the sender must have been in the possession of the shared secret key in order to be able to produce the received message. Upon receiving a message with a MAC a recipient computes the MAC in the same manner as the sender (as the hash algorithm and the secret key are assumed to be known by sender and receiver), then the receiver compares the received and computed MAC values and validates the integrity and data origin authentication of the received message according to the outcome of this comparison. If the MACs are not equal, a modification of the message has been detected. Additionally, encryption can be carried out on the message with an appended MAC to provide message *confidentiality*. [7, 9]

2.3 Possible Attacks

When dealing with feasible attacks on protocols, it is crucial to define the adversary's assumed capabilities. A detailed survey of various adversary models is given in Section 3.1.3. As a base for that discussion, an adversary based on the Dolev-Yao model [10] will be assumed, capable of intercepting all messages, sending them out to the network and altering, re-routing, or injecting captured or newly generated messages in an arbitrary way at any time. Furthermore, it will be assumed that any legitimate protocol participant, any external entity, or a combination of both can act maliciously.

Eavesdropping describes the intercepting of protocol messages by an adversary. Eavesdropping is a prerequisite for several other, presumably more sophisticated, attacks. In order to protect against eavesdropping, assurance of confidentiality can be achieved by applying encryption.

Modification of messages occurs whenever an eavesdropping adversary modifies content of messages. Such a modification can remain undiscovered if no cryptographic integrity operations such as MACs are used for introducing redundancy.

If an adversary eavesdrops on a message and re-injects the whole message or just a part of the message either immediately or at a later time, the attack is referred to as *Replay*. Usually, message replay is combined with other attacks. A specific case of replay, called *Preplay*, appears when an adversary captures a message while being involved in one protocol thread* and re-injects it in another simultaneous or a later protocol run. A different form of replaying is referred to as *Reflection*, whereby an adversary sends back a message to the sender, typically with the intention to get a nonce challenge signed by the sender, where this message was initially addressed to the attacker itself. Reflection is only possible when parallel protocol runs are allowed. To prevent replay attacks, freshness of messages has to be assured.

Denial of Service (DoS) attacks can be conducted by preventing or hindering legitimate agents from being able to execute a protocol. Such attacks are typically carried out against servers as these hosts are communicating with many clients simultaneously. Two types of DoS attacks can be identified: resource depletion attacks (aiming to use up computational server resources) and connection depletion attacks (aiming to exhaust the number of possible connections to a server).

It is hard to avoid DoS attacks completely, since a connection attempt usually results in a resource allocation at the server side or the connection has to be proven invalid, which includes at least some computational work. However, the ease of conducting DoS attacks can be decreased, for instance by the use of stateless connections, where most of the information is kept in storage at the client side and only sent to the server when it is needed. When taking such an approach each message sent from the client has to be integrity protected.

Typing attacks refer to the replacement of protocol message fields of one type, encrypted or not, with the message field of another type. Thereby, tricking the protocol participant to accepting elements as a key which were originally intended to be something else (as for instance origin identifiers) becomes possible. To prevent such attacks, cryptographic operations such as MACs can be applied, which eliminate the possibility of changing the message field order.

When designing a protocol, it is usually assumed that the underlying cryptographic primitives are ideal and immune against *cryptanalysis*. In some cases a combination of cryptographic protocols and cryptographic systems can undermine this assumption. For instance, a cryptanalysis attack has been shown

* See Section 3.1.1 for a definition of protocol threads.

on a protocol using the XOR function as an encryption scheme in a way that a simple XORing of the exchanged cipher text messages reveals the encryption key. Cryptographic attacks take various approaches, thus it is impossible to suggest a countermeasure that will prevent all possible attacks. [11]

The flaw of *Protocol Interaction* describes a maliciously created interaction between a run of a new protocol with a run of a known one. Such an attack becomes feasible if long-term keys are used in multiple protocols, therefore such use should be avoided. [7, 12]

2.4 AKE Design Goals

A sound definition of goals, describing the desired achievements of a certain protocol, lays the foundation of proper protocol design and analysis. When designing AKE protocols, each message field should be justified in view of the defined design goals. In the course of analyzing protocols, an evaluation of robustness against attacks and satisfiability of properties is only expressive if the specific design goals are considered. Any possible attack on a protocol is only harmful if it violates a property which is crucial to hold for this protocol. [13]

The basic design goals of AKE protocols comprise entity authentication and session key establishment related features. The former refer to assurance about identities of those entities taking part in an AKE protocol, whereas the later concerns establishing session keys with goals such as key freshness, key authentication, and key integrity. This section discusses both classes of design goals along with possible combinations and overlaps between those goals in order to establish a hierarchy of AKE design goals.

2.4.1 Entity Authentication

The issue of entity authentication is broadly discussed in the literature, but with various slightly differing definitions given. A common denominator of these definitions is that entity authentication refers to the assurance of an entity, i.e. it is whom it claims to be. [3]

However, this description does not indicate which entity has provided this assurance. For example, if entity authentication should be established between the entities A and B then it is unclear whether A authenticated to B, B authenticated to A, or A and B both authenticated each other, called mutual authentication. Moreover, no assertion can be derived from the above definition about the time

of authentication, since entity authentication does not include information about when an entity has executed authentication. [7]

In order to be clear about entity authentication, this thesis will use the definition of entity authentication given by Lowe in [14]. This definition will be discussed in detail in Section 3.1.4.2.

2.4.2 Good Key Property

A session key, established by an AKE protocol, has to satisfy several features in order to be called a 'good key'. These features basically concern the claim of key freshness and the need to assure that only the correct entities obtain this key.

2.4.2.1 Key Freshness

Session keys are expected to be vulnerable to cryptanalysis attacks, since they are used to repeatedly secure data in regular formats. Hence, it is easy to collect a lot of messages encrypted with the same session key. Accordingly, it is crucial to assure that replaying messages from previous sessions is not possible. Additionally, the likelihood of insecure storage arises.

Such a replay attack on session keys can possibly be carried out by an adversary intercepting A's request for a new session key with B and replaying a known old session key to A in order to decrypt all ongoing communication between A and B. Furthermore, a replay attack can increase the ease of cryptanalysis, since it holds the possibility for collecting additional ciphertext for cracking a session key.

2.4.2.1.1 Establishing Key Freshness

Assurance of key freshness can be achieved by bounding the use of the session key and ensuring a fresh value so that only the sender could have generated it. This fresh value can either be chosen by the user or a received value from a trusted entity has to be verified as fresh by the user.

The former approach is usually taken when dealing with a key agreement. For instance, the entities A and B can both select a random value, thus the session key is computed as a function f, taking these two values as input. As a prerequisite, it should not be feasible for neither A nor B to force the newly computed session key to be the same as a previous one, even if one entity knows the freshness value

of the other. This implies that f has to be a hash function.

The latter proposal includes an entity A requiring a way to verify the freshness of a session key created by another party B. How a value can be checked for freshness is discussed in detail in the next paragraph. Additionally, the received message, including the freshness value N, must satisfy data origin authentication and data integrity in order for A to know that the message has been generated by B and the message has not been altered during transmission. If it can be assumed that N is fresh, then it can be derived that K_{AB} is fresh, since B is a trusted, authenticated entity. [7]

2.4.2.1.2 Freshness values

The critical expectation of a freshness value is to guarantee that it has not been used before. According to L. Gong [15], three basic types of freshness values can be utilized: timestamps, nonces, and counters.

Timestamps contain the current time, appended to a message by the sender at transmission time and checked by the receiver at reception. A check is carried out by comparing the timestamp with the local time. The timestamp is only accepted if it is within an acceptable time window of the current local time. The complexity of using timestamps is that it requires clock synchronization as well as an assurance of secure clocks at sender and receiver sides.

Nonces are values, created by a message recipient A and sent to a sender B. B applies a cryptographic function on A's nonce and sends it back to A, bundled with the actual message. Now A can be assured that the message containing A's nonce is fresh, since there would have been no possibility for B to generate the message at any time before it has received A's nonce. The main disadvantage of this approach is the additional number of messages needed for the interactive nonce exchange. Furthermore, a reliable and high quality (pseudo) random number generation mechanism is a prerequisite for the nonce approach to work, because capture and replay attacks become feasible as soon as nonces can be predicted.

Counters are synchronized values, stored by the sender and the recipient, and are appended to each send message, after which they are increased. The drawback of this concept is the demand to maintain the state information separately for each communication partner, which can lead to a large number of counter values, linearly proportional to the number of communication partners. Furthermore, problems can arise when a given user can use multiple devices (potentially in parallel). Moreover, replay attacks become possible whenever channel errors

appear or counters are not properly synchronized. Hence, a mechanism is needed to recover from synchronization failures. [7, 13]

2.4.2.2 Key Authentication

Key authentication demands, that a certain key K is only known by those protocol participants, who are meant to know it. Accordingly, key authentication is linked to confidentiality, i.e., the secrecy of K. It can be assumed that an authenticated key also implies key freshness, because a key which is not fresh cannot be assured to be confidential. This property of key authentication is sometimes referred to as *implicit key authentication*. [7]

2.4.3 Key Integrity

Key integrity claims that a key has not been modified by any adversary. When designing a key transport protocol, this implies that any key, accepted by a receiver, must be the exact same key as chosen by the sender. Even if the good key property holds and a key is fresh and only known by intended, authenticated entities, the key integrity property can still be unsatisfied. [16]

2.4.4 Combined Goals

AKE protocols usually require a combination of both entity related (entity authentication) and session key related (key freshness, key authentication) goals. These requirements may necessitate enhanced goals, ensuring even stronger properties.

In this regard, *Key Confirmation* of an entity A to an entity B combines the good key property and the possession assurance of a certain key K from A to B. Even if key confirmation is satisfied, keys can still be used for different sessions, since the involved entities can run several sessions simultaneously. No entity authentication is carried out and the only assurance key confirmation gives about entities is the so-called *far-end operative*, which means that the partner wishes to talk to at least one other entity.

Explicit Key Authentication satisfies key confirmation and additionally, a key K is assured to be known only by the correct entities, who can be mutually confident about the possession of K by the other entity. Finally, the strong property of *Mutual belief in a Key* extends explicit key authentication in such a way that the

partners can additionally be assured that the key maintained by the other entity is a good key. [7]

2.4.5 Dealing with Compromised Keys

Particularly strong protocol design goals refer to more powerful adversaries, who can reveal long-term keys and session keys. In this regard, the property of *Perfect Forward Secrecy* demands that even if an adversary compromises the long-term private keys of all agents, the keys of the previous sessions should still remain secret. As soon as any exposed long-term key has been used for encrypting the session key in a key transport protocol, this claim does not hold any more. Perfect Forward Secrecy is usually linked to *Key Independence*, assuring that the revealing of one session key does not facilitate the compromise of other session keys. [17]

Key Compromise Impersonation describes a case where an adversary compromises a long-term key or session key of an agent to impersonate this agent to other protocol participants. To protect against Key Compromise Impersonation, asymmetric cryptography should be used, for instance signing with private keys. [7]

Chapter 3

Formal Verification of Security Protocols

When applying formal methods to verify security protocols, a generic model is required which formalizes the operational semantics of a protocol, the network and the desired security properties. Automatic model checking tools (as for instance Scyther and Tamarin) rely on such formal models. Various slightly differing ways of constructing formal models have been introduced in the course of scientific research on formal verification of protocols with some common characteristics described in [18] and further discussed in [3].

The following chapter will initially introduce these common basics of formal models. Afterwards, different automatic model verification approaches and state-of-the-art model checking tools (building on formal models) are discussed. A formal model for the Dual-Connectivity Protocol is constructed in Chapter 6.2.1.

3.1 Formal Model

In formal verification, a security protocol can only be verified with respect to a formal model. This formal model comprises a protocol model (describing the structure, elements, and semantics of this protocol), an execution model, an adversary model (characterizing the communication network, holding possible intruders), and a specification of the required security properties.

The model abstracts from cryptographic methods used by specific protocols for achieving security. A *Perfect Cryptography Assumption* is made in this thesis, which means that a protocol's algorithms are handled as idealized mathematical constructs and furthermore as black-boxes, since only the outcome is important.

It is assumed that the properties stated in [7, 19, 20] always hold. For example, it is presumed that each encrypted message can only be decrypted with the corresponding decryption key. Hence, an adversary is not able to decrypt messages as long as the decryption key is not revealed.

3.1.1 Protocol Model

A context-free syntax is required to enable a meta-theoretical view of the composition of protocols. Therefore, implementation details of protocols are abstracted away and a symbolic model is created. Messages are represented as a combination of basic *terms* using a term algebra where terms describe either agent names, roles, freshly generated terms (nonces, session keys, etc.), variables, or functions (encryption, decryption, hashing, etc.). These basic terms can be combined in order to achieve various functionality. For example, *pk(X)* denotes the long-term public key of *X*, whereas *sk(X)* refers to the related long-term private key of *X*, and *k(X,Y)* represents the long-term symmetric key shared between *X* and *Y*. Furthermore, $\{t1\}^{a}_{t2}$ describes the asymmetric encryption of term *t1* with the key *t2* and $\{t1\}^{s}_{t3}$ the symmetric encryption of *t1* with *t3*. Finally, a message is a combination of an arbitrary number of terms.

Protocols comprise a set of *roles*, where each role is defined by a sequence of *events*, which can be either the creation, sending, or receiving of messages. Events are executed by *agents* who play specific roles such as the initiator or responder role. Each execution of a role by an agent can be seen as a separate **thread** and accordingly, a single thread is a distinct role instance.

A system consists of one or more agents, each of which can simultaneously execute multiple roles in one or more protocols. Thus, one agent can for instance at the same time act as initiator in two different threats of the same protocol, while acting as responder in another protocol. Therefore, it is necessary to bind roles to actual agents and variables to actual threads. This is achieved by adding a thread identifier to each local variable *var*, for example *var#tid*. [3, 18, 21]

3.1.2 Execution Model

The protocol execution is modeled using system **states** and **transitions** between them. A system state consists of the triple *(tr, IK, th)*, whereby *tr* denotes a specific trace, *IK* stands for the Intruder (adversary) Knowledge, and *th* represents a function, mapping thread identifiers of initiated threads to traces.

Traces track the execution history of events executed by specific threads, i.e. role instances. The *IK* of a Dolev-Yao adversary comprises all agent names and their long-term public keys. Additionally, some long-term private keys of a set of agents may also have been compromised. It has to be kept in mind that multiple diverse adversary models can be used as alternatives to the standard Dolev-Yao model (see Section 3.1.3), which implies different initial *IKs*.

State transitions follow transition rules, describing how the execution of events should be carried out. There are three basic transition rules: the *create* rule, the *send* rule, and the *receive* rule. The *create* rule initiates a new role instance (thread), the *send* rule sends a message to the network, and the *receive* rule describes how an agent, running a thread, receives a message from the network. Based upon these transition rules, it can be decided whether a specific state of a protocol is reachable or not, which forms the basis for verification or falsification of protocols. [3, 20, 18]

3.1.3 Network and Adversary Model

Protocol messages are exchanged via communication networks with various properties and the possibility of there being different adversaries. These adversaries have to be taken into account when constructing a formal model. Whether a protocol is verified as secure or not depends upon the adversary specification and thus, the characteristics of the network in which a protocol is executed have to be carefully considered.

Commonly, the Dolev-Yao adversary model [10] is used to specify such a formal network model. This model assumes that the intruder has complete knowledge of the network and can remove, alter, and send arbitrary messages at any time during the protocol's execution. [7]

However, in some cases weaker or stronger adversary models can be required. For instance, in wireless communication networks it can be assumed that an intruder simply eavesdrops, but does not alter messages [3, 19]. An example of a weak adversary model was suggested by Burrow et al. in [22], where the adversary model claims that legitimate principals will always act honestly and each authenticated entity will follow the protocol specification. In contrast, some protocols may require a stronger adversary definition, for instance for the various AKE Protocols [17]. A particularly strong intruder model is introduced by Bellare and Rogway in [23], where even authenticated principals can act maliciously, thus the adversary can compromise any agent, corrupt random number generators, and reveal long-term keys and session keys.

3.1.4 Security Properties Specification

Within this thesis work a point of view is taken, which considers an attack on a protocol harmful only if it violates a property explicitly stated as crucial for this specific protocol. This leads to the need to account for the required security properties in the formal model [7]. Basically, there are two different classes of security claims, one related to secrecy and the other one related to entity authentication. Security properties are defined in terms of properties of reachable states, thus the security properties must be valid for all states which a protocol can reach during its execution, based on defined transition rules [3, 18].

3.1.4.1 Secrecy

A protocol P holds the claim of Secrecy of term t if all reachable states of P ensure that t is not part of the adversary knowledge IK. Here, t can refer to any term, which is *intended* to be kept secret, for instance a session key. [3, 21]

3.1.4.2 Authentication

Basically, the term authentication can be described as assurance of two agent's identities to each other. However, the detailed specification of the authentication property is a widely discussed topic in the literature. This thesis will use the definitions introduced by Lowe in [14], where a distinction between Aliveness, Weak Agreement, Non-injective Agreement, and (Injective) Agreement is made to classify various forms of authentication, thus offering different degrees of strength.

Definition (Aliveness): We say that a protocol guarantees to an initiator A aliveness of another agent B if, whenever A (acting as initiator) completes a run of a protocol, apparently with responder B, then B has previously been running the protocol.

The definition of aliveness turns out to be the weakest definition of authentication. Aliveness neither assures that B has been running the protocol recently nor that B has been running the same protocol as A. Moreover, it is not ensured that B believes it has been running the protocol together with A, as B can also believe it has been talking to C. As a result, it is easy to carry out simple mirror attacks by reflecting messages of an agent back to itself.

Definition (Weak Agreement): We say that a protocol guarantees to an initiator *A* weak agreement with another agent *B* if, whenever *A* (acting as initiator) completes a run of a protocol apparently with responder *B*, then *B* has previously been running the protocol apparently with *A*.

Weak agreement extends aliveness by additionally assuring that *B* agreed on running the protocol with *A*. However, it is still not ensured that *B* has been acting as responder to A. Thus, an attack could be carried out where an intruder initializes a parallel protocol run in which it impersonates *B* to *A*. Accordingly, *A* would believe that it has been running the protocol with *B*, whereas *B* would think it ran the protocol with the intruder rather than with *A*. This attack is well-known and has for instance been conducted on the Needham-Schroeder Public Key Protocol. [1]

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

The definition of non-injective agreement can be seen as an extension of weak agreement, where the agents additionally agree on their roles. Moreover, agreement on a set of data items (for instance nonces, variables, keys, etc.) exchanged during the protocol execution is carried out. However, still no one-to-one-relationship between agent runs can be assured, thus *A* may believe it has run the protocol twice, while *B* could think it has executed the same protocol only once.

Definition (Injective Agreement): We say that a protocol guarantees to an initiator *A* non-injective agreement with a responder *B* on a set of data items *ds* if whenever *A* (acting as initiator) completes a run of the protocol apparently with responder *B*, then *B* has previously been running the protocol apparently with *A*, *B* was acting as responder in this run, the two agents agreed on the data values corresponding to all the variables in *ds*, and each such run of *A* corresponds to a unique run of *B*.

Injective agreement, also simply called agreement, finally guarantees that each single run of a protocol executed by *A* corresponds to exactly one run of the same protocol carried out by *B*.

Definition (Recentness): It is non-trivial to define what the term recent means as it depends highly on the specific implementation. For instance, it is questionable whether something has happened recently if it appeared during A's run or if it appeared within t time units before A's run. In general, all the above definitions of authentication say nothing about recentness of the authenticated entities, but these definitions can easily be extended in order to assure recentness by adding fresh values (see Section 2.4.2.1.2).

3.2 Automated Model Checking

Different approaches of automated model checking and various proposals to solve the problem of an infinite state search space will be discussed in this section. The paper [3] by Basin, Cremers, and Meadows will serve as the main reference, since it offers a sound and solid discussion of these topics.

3.2.1 State Space Infinity Problem

In order to verify properties of security protocols by automated model checkers the execution of the protocols is represented in terms of reachable states. If *Reachable*(P) refers to all states which can be reached during the execution of a protocol P and S represents the set of states referring to a desired security property S, then a protocol satisfying this property S should fulfill the following formula, claiming that all states reachable by P are included in S:

$$\textit{Reachable}(\mathrm{P}) \subseteq S$$

If $\bar{S}$ refers to the complement of S, including all states describing possible attacks, the above formula can also be expressed as follows:

$$\textit{Reachable}(\mathrm{P}) \cap \bar{S} = \emptyset$$

This formula specifies that no state included in $\bar{S}$ is reachable by P, which means that no attack exists and no counterexample can be constructed.

When it comes to implementing an automatic model checking algorithm to verify the reachability of states, a severe challenge appears due to the fact that the search space becomes infinite (for two reasons). First, it is always possible to start additional threads and sessions (where a session is a single partial execution of a protocol) by the *create* rule (see Section 3.1.2). Second, the number of different

messages which can be received by the *receive* rule is infinite due to the fact that an unbounded number of different messages can be sent by an adversary at any time, using information contained in the adversary's knowledge (as long as the message matches a pattern defined by the receive rule). The latter challenge can be neglected, since it has been proven in [24] that the number of messages involved in an attack is polynomial bounded by the size of the protocol and the number of threads. Thus, the problem of an infinity of messages can be narrowed down to the problem of an infinity of threads.

In such an infinite state space the secrecy problem (see Section 3.1.4.1) is undecidable if there is no bound on the number of sessions. By introducing a bound, the problem becomes NP-complete. The number of states that need to be searched is limited to a specific number and accordingly, the number of possible messages is bounded as well. [3]

3.2.2 Representation of States

When developing state space search algorithms one question is how the reachable states should be represented. Basically, there are two ways of dealing with this issue: explicit and symbolic representations.

When representing states *explicitly*, the operational semantics of a protocol are used to encode each state as a finitely encoded triple. The disadvantage of this approach is that it may lead to state space explosion when verifying complex protocols. A proposal to shorten this problem is compression by using hash tables, for instance.

Alternatively, states can be represented *symbolically* using formulas to describe messages as non-ground term with variables instantiated during the search. Such an approach is preferable to an explicit state representation in terms of efficiency. [3]

3.2.3 Forward and Backward Search

Forward Searching Algorithms compute all reachable states of a protocol, respectively a subset of them, in an iterative manner by beginning with the initial state s_{init}. As soon as a state is reached which is part of $\bar{S}$ (see Section 3.2.1), then the desired property does not hold and a counterexample can be constructed. When a fix-point is reached, i.e, a subsequent state equals the current state, then it can be assumed that the desired property holds for the protocol. Fix-points are always reached in finite-state models, i.e, where the number of sessions and hence

the number of threads is limited. However, in infinite state models the reachability of a fix-point cannot be guaranteed.

In contrast, *Backward Searching Algorithms* take the state set of possible attacks $\bar{S}$ as starting point from which a chain of possible predecessors is iteratively constructed. The search checks whether s_{init} is part of these preceding states. If so, then the desired property does not hold, since there is a possible state sequence leading from s_{init} to a state in $\bar{S}$, thus to a possible attack.

The closure of states is infinite for both forward and backward searching algorithms, although the reasons are different. In forward search infinitely many states can be reached from the starting point s_{init}, whereas in backward search, the set $\bar{S}$ contains infinitely many states. In general, the negation $\bar{S}$ contains more information about states than the initial state s_{init}, since states in $\bar{S}$ include prerequisites such as the adversary knowledge of certain terms or the claim that particular events must have been executed before. Accordingly, applying a backwards search approach, starting from $\bar{S}$, is more suitable when it comes to infinite state models. Conversely, when dealing with finite state spaces it is simple and straight-forward to conduct a forward search starting from s_{init}. [3]

3.2.4 Bounded and Unbounded Model Checking

The main challenge of search algorithms for infinite state spaces is to overcome the infinite state problem by somehow limiting the search space. In this regard, two approaches can be identified: Bounded Model Checking and Unbounded Model Checking.

Bounded Model Checking is a strategy of introducing a bound on the number of protocol sessions, so that only a finite number of a possibly infinite number of states has to be searched. Such an approach has been used by various automated model checking tools, since it turned out to be sufficient to consider only a small number of threads as a function of the number of roles appearing in a protocol. For example, when using a number of threads which is twice the number of roles, it is possible to replay a message from one session in another session.

Alternatively, the *Unbounded Model Checking* approach uses heuristics or abstractions to handle the infinite state space problem. A symbolic representation of states is used, usually combined with a backwards-style search on trace *patterns*. Patterns describe a finite set of events, representing the infinite set of events. For instance, the pattern of a secrecy violation would contain a set of events, for which the secrecy claim does not hold. During the backwards

search it is checked whether the traces of the actual protocol match the specified pattern. Thereby, additional constraints on the protocol traces are added (such as preceding events or limitations on the adversary knowledge) or messages can be unified. Such a course of action is called *constraint solving*. When the algorithm completes, the result is either a contradiction (meaning that no traces of the protocol match the pattern) or a trace of the protocol exists which contains an instance of the pattern (meaning that the pattern can be proven). [3, 25]

3.3 Model Checking Tools

Various model checking tools are currently available, utilizing different algorithms and approaches to realize automated protocol verification. The earliest tools were NPA and Maude-NPA [26], followed by AVISPA [27], Athena [28], and ProVerif [29]. Within this thesis, two of the newest model checking tools, relying on backward searching algorithms, namely Scyther [4] and Tamarin Prover [5], will be utilized and described in detail in the following subsections.

3.3.1 Scyther

The Scyther tool uses an unbounded model checking approach and applies a backwards search algorithm on (trace) patterns. Such patterns describe a partially ordered set of events that must occur in the protocol traces in order for these pattern to be verified. The occurrence of events in protocol traces is checked by matching them to specific criteria defined in the protocol's semantics. Additionally, Scyther offers the possibility to introduce a bound and apply bounded model checking if the unbounded search does not terminate. When using a bound, the result is only valid for the specific bound on the number of sessions.

3.3.1.1 Verification Algorithm

Scyther's backwards search algorithm is a pattern refinement algorithm, which applies a case distinction on the source of messages (to enable constraint solving). During the search, additional information about the patterns is derived, which is used to add constraints. For instance, events and ordering constraints can be added or terms can be unified, thus merged. Furthermore, restrictions on the instantiation of variables can be applied, limiting how the variables can be replaced during the backward search. As an example it could be claimed that variables can only be changed by honest agents.

Usually, patterns which should be verified by automated model checkers represent attack patterns, for example a secrecy violation pattern. This is accomplished by defining an infinite set of traces, representing the contradiction of the security property. In the case of secrecy violation this would refer to all states where an adversary knows a term, which is claimed to be secret.

There are three possibilities for Scyther's algorithm to terminate:

1. A matching protocol trace is found, which means that the pattern is realizable. If the pattern is an attack pattern, this infers that an attack is possible and a trace with the minimal length can be selected from a potentially infinite set of actual protocol traces in order to construct a representative counterexample.

2. No matching protocol trace is found and no bound is reached. From this it can be derived that the pattern is not realizable for any bound. If the pattern is an attack pattern, it can now be deduced that no such attack is possible for any number of protocol sessions.

3. No matching protocol trace is found, but a bound is reached. Accordingly, the verification of the property (a non-realizable attack pattern) is only valid for the specified bounded number of sessions. [3]

As a result of Scyther's protocol evaluation, a summary showing the verification or falsification of security claims is displayed. Optionally, visual graphs of possible attacks can be constructed if a claim has been falsified and a counterexample can be created. The default setting of the Scyther tool limits the session bound to a number, which allows the algorithm to always terminate. Furthermore, it is possible for the user to manually introduce a bound by specifying a custom number of sessions in the settings. Even if a bound is chosen, the protocol can still be verified for an unbounded number of sessions, when the bound is not reached. In contrast, if the bound is reached, this circumstance is displayed as *'No attack within bound'*, claiming that the search tree has not been fully explored. [30]

3.3.1.2 Protocol Description Language

The Scyther tool takes a .spdl (security protocol description language) file as input, which includes a specification of the protocol and the claimed security properties. Scyther's input language syntax is based on C and Java. Scyther uses the formal model discussed in Section 3.1 as a base for defining protocols as a set of *roles*, consisting of sequences of *events*.

A simple input file, describing a protocol with two roles A and B, sending two messages containing strings, could be modeled as follows:

```
protocol SimpleProtocol (A,B) {
        role A {
                send_1(A,B,'ping');
                recv_2(B,A,'pong');
        }

        role B {
                recv_1(A,B,'ping');
                send_2(B,A,'pong');
        }
};
```

3.3.1.2.1 Send and Receive Events

Events can be either the sending and receiving of messages (modeled as *terms*) or *security claims*. Basically, each send event has to refer to a matching receive event, otherwise Scyther does not compile the input file. However, if a single receive or send event has to be modeled (for instance the revealing of a term to the adversary), this can be expressed by adding a ! to the event specification such as:

```
        send_!(A,B,secretKey);
```

3.3.1.2.2 Terms

Atomic terms are described as strings or alphanumeric characters and can refer to any identifier (constants, freshly generated values, variables, etc.). Such atomic terms can be combined through pairing, which enables more complex operations, such as encryption and decryption of messages or hashing. If a term gets too complicated, then macros can be utilized in order to replace longer names with shorter ones. For example, a macro such as *m1* could replace the sophisticated hash *h(A, B, nonce1, term1, term2)*.

3.3.1.2.2.1 Encryption and Hash Functions

Any term can act as a symmetric encryption key. For example, the term $\{ni\}kir$ refers to the encryption of the atomic term *ni* with *kir*. Furthermore, a *symmetric key infrastructure* is pre-defined, enabling the usage of the default key *k(A,B)* as a

long-term shared secret between A and B.

For instance, to denote the sending of a nonce *n1*, encrypted with a symmetric key shared between A and B, one can write:

```
send_1(A,B,{n1}k(A,B));
```

Moreover, a *public key infrastructure* is implemented a priori. A default long-term key pair including the keys *sk(X)*, denoting X's private key and *pk(X)*, denoting X's public key, is available to realize asymmetric encryption as well as signing. As an example, it is possible to send a nonce n1 from A to B, signed with A's private key, encrypted with B's public key as follows:

```
send(A,B,{n1,{n1}sk(A)}pk(B));
```

Hash functions can be expressed in Scyther, usually by a global definition (outside the protocol) of an identifier as hash function. In contrast, the predefined hash function *h* can be used, for instance to produce a hash of the term *ni* by writing *h (ni)*. In order to check hashes, Scyther offers the match function, which takes two parameters as input for comparison [30]. For instance, to check the equality of the terms *Y* and *hash(X, I, R)* the following match would be used:

```
match(Y,hash(X,I,R));
```

3.3.1.2.2.2 Predefined Types and Usertypes

Scyther offeres several predefined, ready-to-use types, in particular *Agents*, *Functions* (defined as function terms, which take a list of parameters as input and are hash functions by default), *Nonces* (fresh values), and *Tickets* (a type, that can be replaced by any arbitrary type of variable). Additionally, new types can be globally defined as *Usertypes*. Such a global declaration can be achieved by using the term *const*, which can be helpful when defining string constants, labels, or protocol identifiers. [30]

3.3.1.2.3 Claim Events and Security Properties

Security properties are modeled as special role events, so-called *claims*, which are part of a certain role's description. Agents have a local view of system states, which they create based on received messages. Properties are always claimed

from this local view and thus, they are only valid from the viewpoint of the specific agent inside whose role description they have been defined. The following paragraphs describe security property claims which can be made in Scyther.

In order to distinguish between different runs of a protocol, an arbitrary number, acting as identifier, is assigned to each run. Local variables can be freshly instantiated by appending this run identifier, for example *nr#1*. Thereby, a run always refers to a single execution of a protocol by a certain agent. [20]

3.3.1.2.3.1 Secrecy Claim

The notation *claim (Initiator, Secret, ni)* defines that a term *ni* is meant to be secret from the perspective of the role *Initiator*. It is possible to declare a secret term *SessionKey* explicitly as session key by using the term 'SKR' (Session Key Reveal) by writing *claim (Initiator, SKR, SessionKey)*. This claim would be falsified if the reveal session key adversary rule is set, since then the adversary would be able to reveal the *SessionKey*.

3.3.1.2.3.2 Authentication Claims

Scyther's authentication claims basically rely on the authentication definitions introduced by Lowe in [14]. These have already been discussed in Section 3.1.4.2.

The *claim (R, Alive, R')* requests **aliveness** of the role R' from the local viewpoint of role R. This infers that R' has at least been talking to R, thus R' has sent a message to R, including a secret that only R' can know. Aliveness offers no assurance about either R' believing it has run the protocol with R, nor whether R' has recently been running the protocol. Additionally, there is no agreement on the roles or exchanged data.

A stronger form of authentication can be demanded when using the claim for **weak agreement** *claim (R, Weakagree, R')*, which additionally requests the agreement of the responder role R' with the fact that R' has been running the protocol with R. However, still no agreement on the specific roles has been carried out.

When it comes to **non-injective agreement**, a distinction between agreement on roles and agreement on exchanged data can be made. By stating *claim (R, Niagree, R')*, non-injective agreement on all roles as well as on exchanged data between the roles can be inquired. This authentication claim can only be modeled

between all of the roles of the protocol and not between certain pairs of roles.

Alternatively, non-injective agreement can be demanded for a certain set of data items which have been exchanged during a specified time. Therefore, the signal *claim(I, Commit, R, terms)* is inserted at the end of the initiator role definition and *claim(R, Running, I, terms)* is placed in the receiver role definition before the last send statement. Agreement is only demanded for the events taking place between the *Running* and *Commit* signals, where the *Commit* refers to the agreement claim and *Running* describes the last communication of the responder role, preceding the *Commit* claim. Thus, the *Running* claim is always placed before the *Commit* claim. Injective agreement can be reached by adding a nonce to the communication and the security claims, since the use of nonces ensures that a one-to-one mapping between roles is present.

The *claim (R, Nisynch, R')* requests **injective agreement** and thereby extends the claim for non-injective agreement by asking for a synchronisation of roles, which means that each role is always mapped to exactly one other role and vice versa. Non-injective Synchronisation can only be claimed for all involved roles, not between pairs of roles.

Finally, the strongest form of authentication which can be demanded in Scyther is **injective synchronization**. This demands a unique set of runs fulfilling all roles claimed to be executed by agents and moreover, the execution of those roles has to be in the exact same order for all agents. For each instance of the claim of a role R in a trace there has to be exactly one unique instance of the role R' to synchronize with. Synchronization means that the execution oder of the roles has to match exactly. In contrast, if only agreement were requested, it would still be possible that a message could be received before it has been send.

3.3.1.2.3.3 Reachability Claim

If *claim(R, Reachable)* is inserted, Scyther will check whether the claim can be reached at all, thus if the protocol is executable until this claim.

3.3.2 Tamarin

Tamarin utilizes and extends Scyther's backwards search verification algorithm. Additionally, it offers two different modes to verify protocols, an automated and an interactive mode, which enables users to 'guide' the tool while executing.

3.3.2.1 Extending Scyther's Verification Algorithm

Tamarin's verification algorithm basically builds on Scyther's backward searching algorithm, but generalizes it in a way that offers additional functionality and makes it possible to define protocols, security properties, and adversaries in a more fine-grained and expressive manner. Additionally, a novel constraint solving approach is introduced, which is described in detail in [5, 17].

3.3.2.2 Fully-automated versus interactive Mode

The Tamarin tool offers two different ways of constructing model proofs, namely the fully-automated and the interactive modes.

The *fully-automated mode* uses Tamarin's heuristics to verify the correctness of claims for an unbounded number of threads and fresh values. If the algorithm terminates, it returns whether the correctness could be verified and in case of an existent attack, a counterexample can be constructed. However, it is not assured that the algorithm terminates, since the nature of the possible specified properties is undecidable.

Thus, the user is able to interactively explore the steps taken by the verification algorithm (proof of single states) in the *interactive mode* by viewing the attack graph and possibly guide the tool as to which approach to take in oder to prove protocol correctness. There may exist protocols which cannot be solved by using Tamarin's constraint solving algorithm, thus the interactive mode cannot help in these cases. However, if there exists a proof, but Tamarin's heuristic selects the wrong way to solve it, then interactive intervention can be used to guide the system to a solution. The user can tell the tool to apply different heuristics during constraint solving by using the flag '–heuristic'. Tamarin offers four different heuristics, which can either be used alone or be combined in any arbitrary way to guide the tool to termination:

s: the 'smart' ranking is the default ranking of premises to be solved, which works well for most protocols;

S: like the 'smart' ranking, but it does not delay the solving of loop breaking premises of a protocol's multiset rewriting rules (loop breaking premises can be inspected in the interactive mode);

c: 'consecutive' or 'conservative' ranking, solves the goals in the order of occurrence in the constraint system, thus no goal can be delayed indefinitely, but also typically infers large proofs;

C: like the 'consecutive' or 'conservative' ranking, but without delaying the solving of loop-breaking premises.

When combing different heuristics, the chosen options are applied in a round-robin fashion. For example, if the option '–heuristic = ssC' is selected, the tool will consequently apply the smart ranking twice followed by the consecutive ranking once. [31]

3.3.2.3 Protocol Description Language

The Tamarin tool is written in the Haskell programming language. The tool takes a .spthy (security protocol theory) file as input. This file describes the protocol messages (defined by multiset rewriting rules), the equational theory and the security properties (specified by axioms and lemmas).

3.3.2.3.1 Multiset Rewriting Rules

The multiset rewriting rules that specify a protocol are represented as sequences of left-hand-sides, labels, and right-hand-sides. These rules consist of facts, which are modeled in the form *F(t1,...,tk)* with a fact symbol *F* and term symbols t_i. The built-in fact *Fr(t)* denotes the generation of a fresh term *t*.

Input facts from the network, modeled as *In(msg)*, must always be placed on the left-hand-side, representing the consuming of a message from the network. The output of messages to the network, written as *Out(msg)*, has to be a right-hand-side fact. By default output facts can only be consumed once (as input fact). Declaring a fact with an exclamation mark defines it as persistent, which means that it can be consumed arbitrary often. Different prefixes are used to describe various types of variables, with '∼' describing a fresh value, '$' a public value, and '#' a temporal value. [5, 6, 31, 32]

To gain a clearer understanding, the following example input file will be described in detail:

```
rule Step1:
        [ Fr(~t1),
        In(t2)]
                --->
        [ Out(~t1),
        Step1(~t1, t2) ]
```

In the above listing a fresh value *t1* is generated and sent out to the network. Furthermore, a message containing the term *t2* is consumed as network input. Both values are saved as a state of Step1, offering the possibility of obtaining them in later steps, by requesting *Step1(t1,t2)*.

3.3.2.3.2 Equational Theory

The equational theory is defined by functions, equations, and boolean flags.

Functions with arbitrary arities can be described by the user. For instance, the term *aenc/2* can be used to denote a binary asymmetric encryption function and *adec/2* the related decryption function. The term *h/1* describes an unary hash function, or *g/0* models a constant function. Additionally, equations can be specified. For instance, the following equation declares the asymmetric decryption of an encrypted message *m*:

```
adec(aenc(m,pk(k)),k) = m
```

Moreover, boolean flags state which built-in functions are required by the protocol. The most commonly used built-in functions are those for hashing, asymmetric encryption, symmetric encryption, Diffie-Hellman, and signing.

3.3.2.3.3 Axioms and Lemmas

Desired security properties are modeled as axioms and lemmas, describing trace properties. Axioms define certain trace properties which have to hold for a trace in order to be included in the search. In contrast, lemmas specify security properties for whose validity the traces should be checked. [5, 6, 32] The syntax for defining axioms and lemmas uses the following terms:

```
All       for describing that all traces must fulfill the claim
Ex        for describing that at least one trace must exist
                 which fulfills the claim
==>       for implication
&         for conjunction
|         for disjunction
not       for negation
f @ i     binds f to the temporal variable i
i < j     for temporal ordering
#i = #j   for an equality between temporal variables
x = y     for an equality between message variables
```

An axiom could for example request, that all equality checks of a trace must have been successful as a prerequisite for this trace to be included in the search space. Such an axiom can be denoted as:

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

A lemma stating session key secrecy could for instance be defined as:

```
lemma session_key_secrecy:
     /* It cannot be that */
   "not(
         Ex A B SKeNB #n.
           /* somebody claims to
                have setup session keys, */
           SessionKeys(A, B, SKeNB) @ n
           /* but the adversary knows one of them */
         & (Ex #i. K(SKeNB) @ i)
       )"
```

Chapter 4

Related Work

Automated model checkers have been applied to a wide range of security protocols. When creating a formal model as a base for protocol verification with automated model checking tools, it is desirable to keep the abstraction on a level which still allows valuable results. This chapter will initially discuss different approaches that have been taken to create formal protocol models. Afterwards, already conducted projects dealing with automatic protocol verification will be covered with a focus on AKE protocols, since this thesis is targeting AKE protocols and their verification.

4.1 Formal Protocol Modeling

A method to automatically derive a model (executable by the model checker ProVerif) from F# code was introduced in [33]. Such a course of action aims to fill the gap between the formal model and the protocol implementation. Other projects dealing with the translation of implemented protocols to semantic, formal models are [34] and [35]. These projects try to establish rules for deriving semantic models from various classes of Public Key Exchange (PKE) Protocols.

In contrast to the verification of an already implemented protocol, automatic protocol verification can be done during a protocol's design process. This is the case explored in this thesis, where the formal model will be constructed based on declared design requirements and prerequisites.

4.2 Automatic Verification of Protocols

Scyther was used for verification of multiple IPSec key-exchange protocols, in particular IKEv1 and IKEv2 in [37], where severe vulnerabilities of the

authentication properties could be identified. An evaluation of Menezes–Qu–Vanstone (MQV) protocols, a family of AKE protocols satisfying strong security properties, has been conducted in [18]. In the course of this, an extension of Scyther, offering additional adversary models, was implemented. With this adversary model extension, verification of advanced security properties such as perfect forward secrecy and key compromise impersonation (see Section 2.4.5) becomes feasible in Scyther. In [38], various entity authentication protocols defined by the ISO/IEC 9798 standard [39] have been modeled and tested in Scyther, revealing multiple weaknesses violating the authentication property.

The Tamarin Prover tool has been applied to a series of AKE protocols in [6], where various adversary models and security properties were modeled and used. Furthermore, an authentication protocol based on one-time passwords has been evaluated by utilizing the Tamarin Prover in [40]. In [41], a study verifying multiple AKE protocols in view of different adversary models was carried out. Specifically, the two-pass AKE protocols NAXOS and RYY as well as the triparty Key-Exchange protocol Joux were evaluated with regard to very strong security property assumptions, such as perfect forward secrecy & session key and long-term key revelations.

A verification of the Elgamal Algorithm was conducted by using Linear Temporal Logic in [42]. A transition system was formulated as the base on which the formal verification was carried out.

Automatic verification of LTE protocols was conducted in [36], where LTE session management and mobility procedures are automatically verified using the model checking tool ProVerif. In the course of this, several restrictions of ProVerif were explored when using the tool for LTE protocol modeling. These restrictions are mainly linked to the tool's inability of modeling state information.

Chapter 5

Method

In general, an *engineering approach* is used to design a solution to a human problem, describing a need or desire of today's society, where the focus is on construction aspects and answering the question of how things should be. In contrast, a *scientific approach* aims at studying existing phenomena, starting with a hypothesis to be verified and investigating how things are. [43]

This thesis' objective is to formalize and verify a security protocol for LTE dual-connectivity during its standardization process and evaluate the used verification tools. To achieve this, a combination of an engineering approach and a scientific approach is applied. An engineering method is used to formalize the protocol, evaluate the used real-world software tools for model checking, and possibly refine the initial protocol design with regards to the verification results. During the formal verification process, a more scientific strategy is taken, since the model checking tools are used to experimentally verify (or falsify) an initial hypothesis, claiming that the protocol is secure in regards to the defined formal model.

The basic steps of the method used within this thesis, can be summarized as:

1. Requirement analysis
2. Protocol design modeling and derivation of a formal model
3. Automated model checking
4. Protocol design refinement
5. Evaluation of tools that have been used

It is impossible and undesirable to completely separate the above steps, as they overlap in several points. For example, the requirement analysis affects the design model, which lays the basis for the formal model and thus, also influences the input models provided to the tool(s). In general, an iterative software development approach was taken within this thesis, which infers adjustments of earlier steps with regard to the outcome of later steps. Accordingly, the execution of these steps does not imply a sequential execution. For instance, the formal model can be altered in view of limitations discovered in the model checking tools or refinements of the design model can be made with respect to verification results.

The first step, the requirement analysis, aims to define the problem to be solved and to constrain the design modeling of the protocol. To meet these goals, an in-depth literature study of AKE protocols and formal verification was carried out. Furthermore, a study of the currently available 3GPP drafts concerning the dual-connectivity protocol was used as a basis for defining design constraints and preliminary requirements.

The second step includes the creation of a design model of the dual-connectivity protocol and derivation of a formal model. The design model comprises a description of the protocol's architecture as well as the design goals and the detailed message flow. The literature study carried out in step 1 serves as a basis for creating the design model. Subsequently, a formal model is derived from the design model by following rules- extracted during the literature study on formal verification. The formal model lays the foundation for implementing the dual-connectivity protocol with the syntax required by the two model checking tools Scyther and Tamarin.

In the third step, the protocol models are verified with Scyther and Tamarin to check whether the initial hypothesis (that the protocol is secure with regards to the defined formal model) holds. The results of the formal verification are documented. Within the fourth step, the design model created in the second step may be refined with regards to possible existent flaws.

Finally, a detailed evaluation of the model checking tools that were applied and their suitability for LTE protocol modeling will be conducted in step four. In particular, the usability of the tools as well as their scope and limitations will be analyzed.

As a result, tool models of the dual-connectivity protocol (in Scyther and Tamarin) and the corresponding results (verification or falsification of the initial hypothesis that the protocol is secure) will be available. Furthermore, the

evaluation of the protocol modeling with the chosen model checking tools will be documented.

Chapter 6

Dual-Connectivity Protocol Formalizing and Verification

This chapter introduces the design model of the dual-connectivity protocol in the first section. Following this, the second section describes the formal models, which provide the basis for protocol verification using the automated model checking tools Scyther and Tamarin.

6.1 Design Model

Currently, the dual-connectivity security protocol is being standardized by 3GPP. In the course of this standardization process, various 3GPP drafts have been published. This section first discusses the basic protocol architecture as defined in [44] and the overall key establishment schemas introduced in [45]. Afterwards, a detailed protocol description including preliminary requirements, design goals, and the basic message flow of the dual-connectivity security protocol is introduced. The design goals and the detailed message flow have been newly created within this thesis project and have not previously appeared in any 3GPP draft.

6.1.1 Overall Architecture

In an Evolved Universal Terrestrial Radio Access Network (E-UTRAN) multiple E-UTRAN NodeBs (eNBs) provide the E-UTRAN user plane and control plane to User Equipments (UEs). These eNBs are inter-connected via X2-C interfaces and further connected to the Evolved Packet Core (EPC), as described in Figure 6.1.

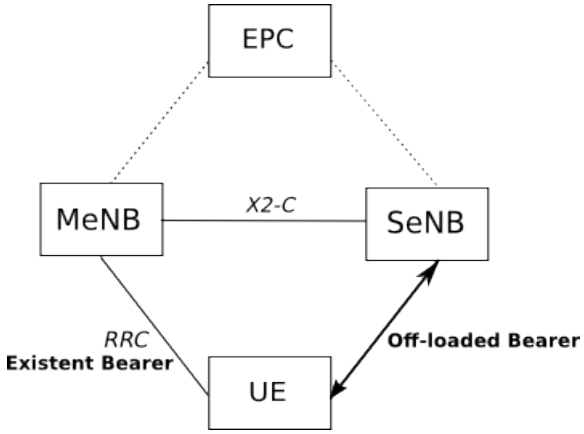


Figure 6.1: Overall Architecture

The Dual-Connectivity Protocol is designed for a specific constellation, where a UE is simultaneously connected to two (or more) eNBs. The security context for a specific bearer * *b* is already established between the UE and one of these eNBs, referred to as the Master eNB (MeNB). During execution of the Dual-Connectivity protocol, a secure connection for *b* is off-loaded from the MeNB to the Secondary eNB (SeNB). The purpose of the bearer off-load from the MeNB to the SeNB is to have the UE utilizing the potentially unused resources of SeNB.

It is assumed that the UE has already authenticated to the EPC. As a result of a successful authentication, a secure channel between the MeNB and the UE is established (using *Radio Resource Control (RRC)*), where both parties share a session key K_{eNB}. From the K_{eNB} further keys are derived as described in Section 6.1.2.2. The channel between MeNB and SeNB (*X2-C*) is secured via an IPsec session key K_{X2}, which is the origin for deriving keys for integrity and confidentiality protection of signaling traffic between MeNB and SeNB. Both channels, RCC and X2-C, are authenticated as well as integrity, confidentiality, and replay protected. [44]

6.1.2 Protocol Description

The aim of the Dual-Connectivity key establishment is to derive a session key S-K_{eNB}, known by SeNB and UE. The S-K_{eNB} is used to derive a K_{UPenc}, serving as encryption key for data traffic between SeNB and UE, related to a specific bearer.

* A bearer is a logical connection between two endpoints (UE and eNB), aggregating one or multiple data plane flows.

When the first bearer is off-loaded from the MeNB to a SeNB, the MeNB derives a S-K_{eNB} from the K_{eNB}, using a Small Cell Counter (SCC) as freshness input to the Key Derivation Function (KDF). The MeNB sends S-K_{eNB} to the SeNB and the value of SCC to the UE, so that the UE can calculate S-K_{eNB}. Afterwards, SeNB and UE can derive K_{UPenc} from S-K_{eNB}, which is used for encryption of data plane traffic between SeNB and UE. The key derivations are described in Figure 6.2.

If an additional bearer is offloaded from a MeNB to a SeNB and this SeNB has already established a dual-connectivity security context with the MeNB (thus, the SeNB is in possession of a S-K_{eNB}), then the same S-K_{eNB} can be used to derive a K_{UPenc} for the new bearer as long as there is no re-use of Data Radio Bearer (DRB) IDs, which would lead to key-stream reuse. (see Section 6.1.2.4). Before a DRB-ID is reused, refreshing of the S-K_{eNB} is necessary.

6.1.2.1 Preliminary Requirements and Assumptions

When designing the dual-connectivity protocol, it has to be kept in mind that security messages are attached to actual messages sent via the radio link. This implicates restrictions in terms of message size, order, and number of sent messages.

A 16 bits counter (SCC) serves as freshness input to the KDF for computing a S-K_{eNB} from a K_{eNB}. Other approaches for providing freshness have been rejected due to the following reasons:

The dual-connectivity security protocol should be decoupled from the underlaying protocols, which is one argument why a time-stamp based approach, relying on synchronized clocks, has not been chosen. Another reason for rejecting the usage of time-stamps is that the clocks in LTE are only synchronized in terms of relative time (time which has passed between two time-stamps), while the actual sense of time can vary between different devices.

The use of nonces would not meet the demands of this specific protocol, because the possibility of collision (repeating values) would be too high when using a small range for the SCC. If one used a pseudo-random sequence to generate nonces and used a method for detecting when re-use happens, then one has to add additional messages to the system. Additional logic would have to be implemented in both UE and MeNB on how to behave when a collision occurs. A collision can be expected to occur after roughly 2^8 derivations (due to the birthday paradox). In contrast, for a counter there is more control on handling repeating

values, since one knows exactly when the repetition will occur and there is a simple rule to detect it (wrap-around). Accordingly, when using a counter, the repetition is guaranteed to not come earlier than the full 2^{16} bits are used. Hence, the usage of a counter makes the design of the overall architecture simpler then the usage of a nonce. The randomness, provided by a nonce, would give some increase in security (it provides some protection against pre-computation attacks due to that the attacker cannot know the input when doing the pre-computations for a given K-eNB). However, as the SCC variable is only 16 bits long, a nonce does not provide that much of protection.

It is assumed, that MeNB can not be compromised at any time. Compromising the MeNB would lead to revealing of the K_{eNB} and the S-K_{eNB}. The channel between MeNB and UE and the channel between MeNB and SeNB are assumed to be authenticated and secure.

6.1.2.2 Key Hierarchy

The K_{eNB} is used to derive keys for confidentiality and integrity protection of messages, exchanged via the secure channel (*RCC*) between MeNB and UE. Furthermore, a session key S-K_{eNB} is derived from the K_{eNB}, which itself is used as input to a derivation of K_{UPenc}s. For the sake of completeness it should be mentioned that there are actually two different K_{UPenc}s:

1. A $K_{UPenc(MeNB)}$ for (user plane) data traffic encryption between MeNB and UE is already established before execution of the dual-connectivity protocol. However, this $K_{UPenc(MeNB)}$ is not relevant for the description of the dual-connectivity protocol.

2. A $K_{UPenc(SeNB)}$ for (user plane) data traffic encryption between SeNB and UE is established by the dual-connectivity protocol. From now on, the term K_{UPenc} will only be used for describing this $K_{UPenc(SeNB)}$.

In summary, the following keys are derived from the K_{eNB}:

K_{RCCenc} for encryption of signaling traffic between MeNB and UE

K_{RCCint} for integrity protection of signaling traffic between MeNB and UE

S-K_{eNB} for derivation of the K_{UPenc}

K_{UPenc} for data traffic encryption between SeNB and UE

Figure 6.2 describes the derivation process of the K_{UPenc}. The derivation is carried out by different entities involved in the dual-connectivity Protocol execution. The K_{eNB} is initially known by the MeNB and the UE. During a protocol run messages are sent by the MeNB to the UE and SeNB, so that afterwards the UE and the SeNB both know the S-K_{eNB} and can derive K_{UPenc}. The exact messages, exchanged during the dual-connectivity protocol execution, are described in Section 6.1.2.6.

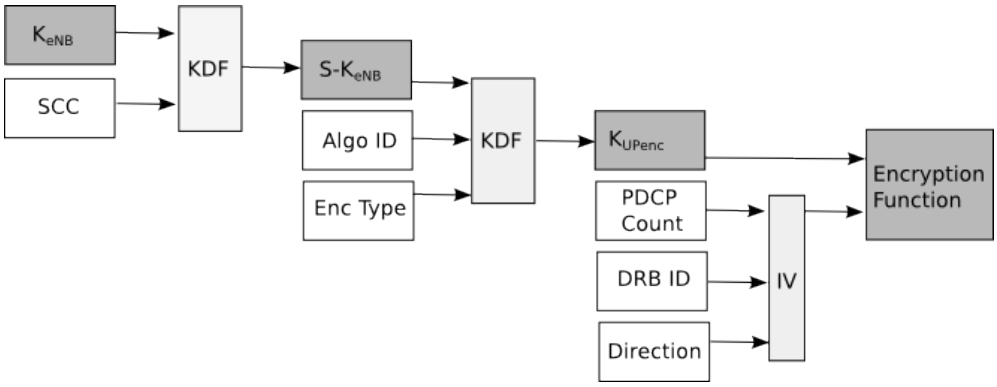


Figure 6.2: Key Derivation of K_{UPenc}

The KDF for deriving a session key S-K_{eNB} takes as input the K_{eNB} and the value of a SCC. The K_{UPenc} is derived from the S-K_{eNB}, where an algorithm identifier string and the encryption type serve as further inputs to the KDF. In order to achieve a distinct per packet encryption, an Initialization Vector (IV) consisting of the Packet Data Convergence Protocol (PDCP) packet count, the DRB ID, and the direction of the communication is used as input to the encryption algorithm.

6.1.2.3 Design Goals

For the dual-connectivity protocol the following design goals can be derived from the current 3GPP drafts [44, 45]:

1. Key authentication (key secrecy) of K_{eNB} and *S-K_{eNB}* (respectively K_{UPenc})
2. Key freshness of *S-K_{eNB}* (respectively K_{UPenc})
3. Mutual agreement on the key possession of *S-K_{eNB}* (respectively K_{UPenc})

These three goals can be summarized by using the term *Explicit Key Authentication* (described in Section 2.4.4).

6.1.2.4 Security Considerations

In order to offload established bearers to SeNBs, the MeNB derives S-K_{eNB}s from a K_{eNB} using SCCs as freshness inputs. One SCC value is always used for deriving S-K_{eNB}s for one specific SeNB and S-K_{eNB}s are never repeating for different SeNBs. A data plane encryption key K_{UPenc} is derived by SeNB and UE from S-K_{eNB} as described in Figure 6.2.

The input parameters to the KDF for K_{UPenc} derivation (algorithm identifier and encryption type) can be considered as static values and the IV for the per-packet encryption function comprises either static or possibly repeating values, since the DRB-IDs and PDCP counters can wrap around. The PDCP count increases every time a new packet is transmitted. Thus, the same S-K_{eNB} could be used with the same PDCP count and the same DRB-ID as soon as a DRB-ID is reused. Accordingly, refreshing of the S-K_{eNB} is required before a DRB-ID repetition in order to prevent key-stream reuse. Therefore, the MeNB keeps track of used DRB-IDs in each SeNB and before a DRB-ID reuse would occur, MeNB increases the SCC value and derives a new S-K_{eNB}.

Additionally, a wrap-around of the SCC requires choosing one of the following three options to prevent key-stream reuse:

Option 1: Refresh of the K_{eNB}

Option 2: Release of the UE

Option 3: No more offloads are done

The reason for this is that the derivation process of the K_{UPenc} from the S-K_{eNB} includes only static or possibly repeating values (counters). A K_{eNB} refresh enables the MeNB will derive a fresh S-K_{eNB} next time and a fresh K_{UPenc} will then be derived at SeNB and UE. [45]

6.1.2.5 Small Cell Counter (SCC) Maintenance

The SCC is a counter stored by the MeNB, which wraps around as soon as the capacity of the variable (16 bits) is exhausted. The SCC is associated with the security context of one UE and is maintained as long as this security context is existent. The purpose of the SCC is to serve as freshness input to the KDF for deriving a session key S-K_{eNB} from K_{eNB}. The SCC is sent over the air from the MeNB to the UE, where it is assumed to be impossible to modify the SCC's value since the channel between MeNB and UE is integrity and replay protected.

It is assumed by the UE, that the MeNB generates and increases SCCs correctly and every time it is needed. When a security context for one UE is established at the MeNB for the first time, the SCC's value is set to '0' and the first S-K_{eNB} is computed. In contrast, an initialization of the SCC with a random value would be conceivable, but such a course of action would lead to an earlier wrap around of the SCC and moreover, re-computation tables would become bigger. Hence, the initialization with a '0' value has been chosen.

When the first bearer is offloaded to a SeNB, the MeNB sets the SCC's value to '1' and subsequently the SCC is increased whenever the S-K_{eNB} needs to be refreshed (due to a wrap around of the DRB-ID counter). Before the SCC wraps around (which means that it starts from '0' again) a refresh of the K_{eNB} is carried out by the MeNB and accordingly, a new S-K_{eNB} is computed with a SCC of '0' as input.

6.1.2.6 Generic Message Flow

Figure 6.3 describes an example message flow exchanged during an execution of the dual-connectivity protocol, which is based on Figure 2.1.1-1 (”Possible scenarios of dual connectivity with a SeNB and mechanism for key handling”) introduced in the 3GPP draft [45]. Figure 6.3 includes only a subset of the messages shown in Figure 2.1.1-1, since this subset is assumed to be sufficient to derive formal models for automated verification of the required security properties.

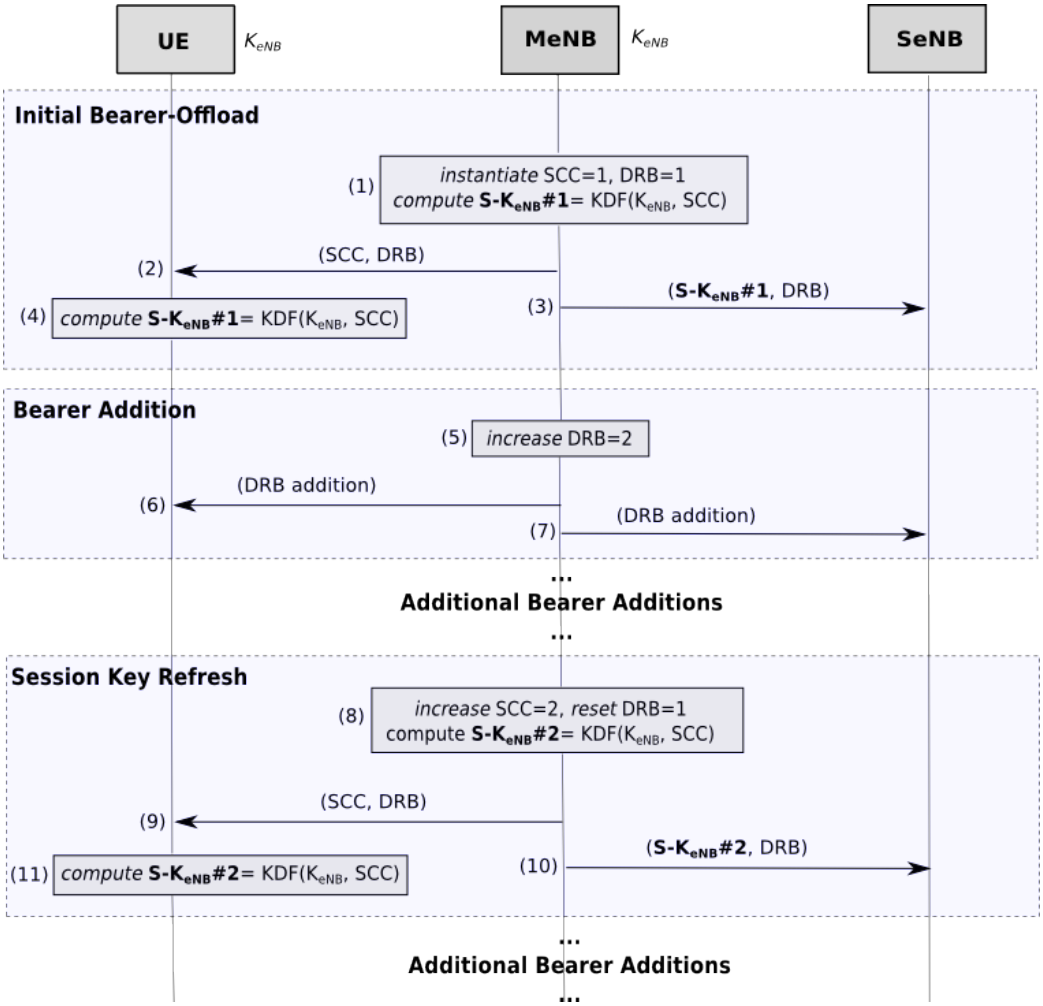


Figure 6.3: Generic Dual-Connectivity Message Flow Example

As it is infeasible to properly model all of the necessary concepts of the dual-connectivity protocol using the tools Scyther and Tamarin, two specific message

flows viable for each of these tools will be introduced in Sections 6.2.2.1 and 6.2.2.2. These two tool specific message flows will extend or simplify the generic message flow described in Figure 6.3 .

In the generic dual-connectivity message flow example above, UE and MeNB are initially in possession of the secret key K_{eNB} and SeNB and MeNB share the secret key K_{x2}. After the protocol's successful termination, the S-K_{eNB} (derived from the K_{eNB} by using a KDF) should be known to SeNB and UE.

6.1.2.6.1 Initial Bearer-Offload

In step 1 MeNB instantiates the SCC's value and the DRB count with '1' (which implies the first bearer addition) and computes the first S-K_{eNB} (S-K_{eNB}#1). In step 2 the MeNB sends the SCC's value together with the DRB to the UE. Prior knowledge of the K_{eNB} by SeNB should be avoided and the control of the resources, allocated to the UE, should remain at the MeNB. Hence, only the S-K_{eNB}#1 is transmitted to SeNB, together with the DRB, in step 3.

Finally, the UE can compute S-K_{eNB}#1 (using K_{eNB}) in step 4 and both parties, UE and SeNB, are now able to derive K_{UPenc}#1 to encrypt data plane traffic. After the derivation of K_{UPenc}#1 the S-K_{eNB}#1 can be deleted at the SeNB, since there is no need for further storage. *

6.1.2.6.2 Bearer Addition

When an additional bearer should be offloaded to a SeNB, which has already established S-K_{eNB}#1 with the MeNB, then this S-K_{eNB} and the derived K_{UPenc} can be used to encrypt data plane traffic related to the additional bearers (if they can be assigned DRB-IDs for the same PDCP counts, thus the DRB-ID space is not exhausted). The K_{UPenc}#1 remains valid while the MeNB continues to use S-K_{eNB}#1 (as the derivation of K_{UPenc}#1 does not change until S-K_{eNB} changes). For example, in Figure 6.3 the MeNB increases the DRB count to '2' in step 5 and sends a DRB addition message to the UE in step 6 and to the SeNB in step 7. In this case, the key K_{UPenc}#1, which has been derived from S-K_{eNB}#1, can be used to encrypt data plane traffic related to DRB-ID '1' and '2', since the encryption algorithm takes the DRB-ID and the PDCP count as freshness input to the IV in order to prevent key-stream reuse.

* In keeping with the general desire of eliminating storage of keying material when it is not necessary, thus minimizing the risk of exposing it.

6.1.2.6.3 Session Key Refresh

As soon as the DRB-ID space is depleted, the possibility of DRB-ID reuse with the same K_{UPenc} would occur, hence this has to be prevented. In order to avoid this, the MeNB increases the SCC's value and performs a refresh of the S-K_{eNB} when the DRB-ID space is close to being exhausted, as described in step 8.

As soon as a new bearer should be off-loaded, the new value of SCC and the ID of the bearer to be added (which is again '1' after a wrap-around of the DRB-IDs) is sent to the UE in step 9 and S-K_{eNB}#2 is sent to SeNB in step 10. Finally, the UE derives the S-K_{eNB}#2 in step 11 and both, UE and SeNB can derive a fresh K_{UPenc}#2 to encrypt data plane traffic related to subsequently added bearers. K_{UPenc}#1 continues to be used for the existing bearers (i.e., those that have been previously set up)

Modeling a possible wrap-around of the SCC, which would additionally require either a refresh of K_{eNB} at the MeNB or a release of the UE, has been left out in the above figure in order to avoid unnecessary complexity.

6.2 Formal Verification

In this section, different formal models (one for Scyther and one for Tamarin) will be derived from the design model specified in Section 6.1. The need for different models for the two model checking tools that have been used has arisen due to distinct modeling possibilities and limitations of each of these tools. The common basics of formal models of both tools are described in Section 6.2.1. Afterwards, the tool specific models and related message flows (based on the generic Dual-Connectivity message flow described in Section 6.1.2.6) are introduced in Section 6.2.2.

6.2.1 Generic Formal Model

The tool specific formal models for Scyther and Tamarin both comprise a protocol model, an adversary model, and a specification of the desired security properties, which lays a common base for protocol verification of the Dual-Connectivity Protocol.

6.2.1.1 Generic Protocol Model

The dual-connectivity protocol requires three **roles**: a MeNB, an UE, and a SeNB. The **keys** which are used and constructed during the protocol's execution in the

formal model are only a subset of those described in Section 6.1.2.2. The reason for this is that two keys *k1* and *k2* can be modeled as one key in the formal model when *k2* is derived from *k1* by using a publicly known KDF with publicly known and static input (for instance an algorithm identifier). If so, then it can be assumed that it makes no difference in terms of security whether *k1* or *k2* is used to apply cryptographic operations to messages, since the KDF is part of the adversary's knowledge and the adversary can compute *k2* as soon as it knows *k1*. Accordingly, keys which are derived from K_{eNB} for encryption and integrity protection of the RCC link (K_{RCCenc}, K_{RCCint}) are both modeled as K_{eNB} in the formal model. Furthermore, keys derived from K_{x2} for encryption and integrity protection of the X2-C link are modeled as K_{x2}.

Finally, the protocol model will be expressed through message flows consisting of terms understandable by the two model checking tools. These message flows, which differ for the two tools Scyther and Tamarin, will be introduced in Section 6.2.2, in particular in Figure 6.4 and Figure 6.5.

6.2.1.2 Generic Adversary Model

To evaluate the dual-connectivity protocol, the basic Dolev-Yao model [10] will be used, which was described in Section 3.1.3. Additionally, it is a prerequisite that no compromising of the MeNB is possible and neither long-term keys nor session keys can be revealed.

6.2.1.3 Generic Security Properties

With regard to the design goals of the dual-connectivity protocol described in Section 6.1.2.3, various security properties can be defined in the formal model. All properties linked to entity authentication can be neglected, since authenticated channels are assumed to exist pairwise between MeNB and UE, and between MeNB and SeNB. Key-related properties can be taken from the design goals previously defined in Section 6.1.2.3.

6.2.2 Tool Specific Formal Models

This section describes the basics of modeling the dual-connectivity protocol with Scyther and Tamarin Prover by introducing tool specific message flows derived from the generic formal model (see Section 6.2.1) and generic message flow (see Section 6.1.2.6). As the two used tools offer different possibilities and limitations, distinct message flows have been designed for Scyther and Tamarin. These message flows lay the base for implementing the dual-connectivity protocol using

the tool specific protocol input languages. The input files for protocol verification with Scyther and Tamarin can be found in the Appendices A.1, B.1, and B.3.

6.2.2.1 Scyther Model

Due to specific properties of the Scyther tool, the protocol input model for Scyther is a modified version of the generic formal model. Unfortunately, counters could not be modeled in a satisfying way, hence DRB-IDs were neglected and the SCC was implemented as a simple nonce generated by the MeNB. This creates a problem since a nonce is never predictable, while a counter is. Thus, the protocol model in Scyther has a stronger computational protection against pre-computational attacks than the real protocol. Furthermore, a wrap-around of counters could not be implemented in Scyther, which represents another problem caused by limitations of the tool.

In order to make Scyther understand the secure channels, all messages between MeNB and UE, and between MeNB and SeNB are encrypted with secret keys shared between the two related parties. To be precise, this encryption would not be necessary for all exchanged messages, as for instance the sending of the SCC from MeNB to UE is only in need of integrity and replay protection. However, in order to be consistent, all messages sent via secure channels are encrypted in the model in Figure 6.4.

Figure 6.4 outlines an example message flow of the Scyther model.

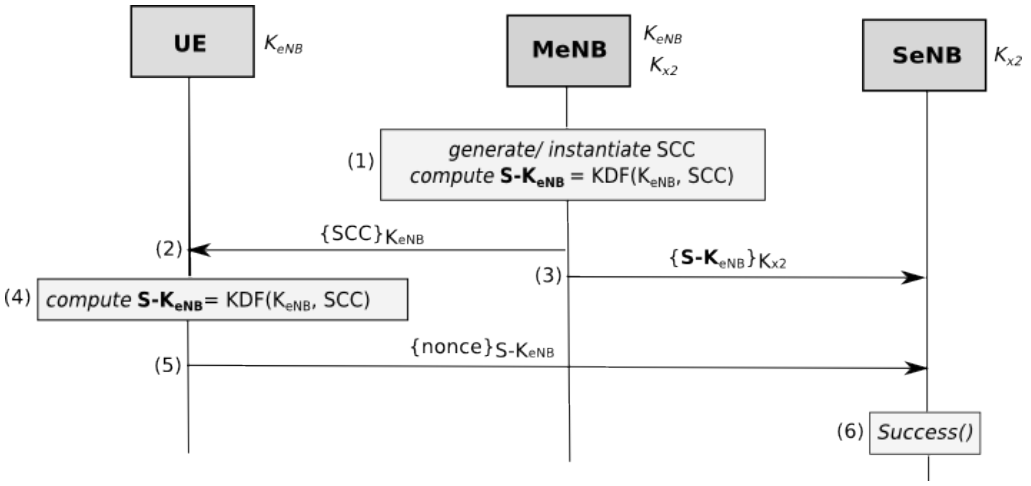


Figure 6.4: Dual-Connectivity Message Flow Example (Scyther)

In step 1 the MeNB creates and instantiates the counter SCC. The session key S-K_{eNB} is computed using a KDF, taking the SCC and K_{eNB} as input.

In step 2 the MeNB sends the value of the SCC to the UE, encrypted with K_{eNB}. This enables the UE to compute S-K_{eNB} in step 4 by applying a KDF to the SCC and the K_{eNB}. The MeNB sends the derived session key S-K_{eNB} to the SeNB in step 3, encrypted with K_{x2}.

Finally, in step 5 the UE sends a nonce to the SeNB, encrypted with S-K_{eNB}, to check whether the SeNB can decrypt it correctly, hence the S-K_{eNB} exchange was correct. If so, the protocol execution succeeds in step 6.

The complete Scyther input file can be found in the Appendix A.1.

6.2.2.1.1 Adversary Modeling

When it comes to adversary modeling, Scyther offers two possible options for describing how Long-Term Key Reveal (LKR) can be carried out: LKR_{others} and LKR_{actor}. When choosing the first option, the adversary can learn the key of any agent who is not an intended partner of the protocol to be verified. For example, if Alice communicates with Bob, Dave's long-term key can be revealed, but never Alice's or Bob's long-term keys.

In contrast, the latter option describes an adversary who can learn the long-term-keys of agents by executing a test thread. In this case, if Alice communicates with Bob, Alice's and Bob's long-term keys can be revealed to the adversary. The setting LKR_{others} is chosen as default option by Scyther. To satisfy the adversary model of the dual-connectivity protocol, the options LKR_{others} and LKR_{actors} will both be disabled, since revelation of long-term keys is assumed to be impossible.

6.2.2.1.2 Security Properties Modeling and Proving

Since the Scyther tool has several restrictions, it was not possible to model all of the initially claimed security properties (see Section 6.1.2.3). Implementing *key freshness* of the S-K_{eNB} has not been feasible due to an inability to model counters in Scyther. As a result, the SCC is defined as a simple nonce and DRB-IDs are not implemented at all in the Scyther input file. Hence, it is impossible to check whether the same value of the SCC is reused with the same DRB-ID, which is the crucial property leading to verification of K_{UPenc} freshness.

When trying to model *mutual session key possession* of the S-K_{eNB} (by SeNB and UE), an additional test message containing a nonce was introduced, which does not exist in the generic model. The receive event listed below is the actual Scyther implementation of this message sending in step 5 in Figure 6.4.

```
recv(UE, SeNB, {n-ue}SK-eNB)
```

Scyther implicitly compares every incoming message to the specified pattern in the receive event and drops the message if it does not match this pattern. This pattern matching can be seen as an implicit MAC on every sent message, which is why there is no need to model MACs explicitly. The above receive event requires a message containing a value for *n-ue*, encrypted with the session key *SK-eNB*.

If all roles of the dual-connectivity protocol are reachable, then it means that there exists at least one trace where all roles were able to run the protocol until the last defined event. If so, UE and SeNB were in possession of the same session key in at least one trace, because they were able to encrypt and decrypt the nonce *n-ue* properly. Since all roles of the dual-connectivity protocol could be verified as reachable, the property of mutual session key possession holds for at least one trace. However, mutual key possession cannot be proven for all traces, thus not all protocol runs have been proven to lead to mutual key possession.

It is possible to model *key authentication* (or *key secrecy*) of K_{eNB} and S-K_{eNB} from the local view of a specific role.

The related claim in Scyther is written as:

```
claim (<Role>,Secret,k(MeNB,UE));
claim (<Role>,Secret,SK-eNB);
```

In the above listing, which is a simplified extract of the Scyther protocol input file, the initially shared key K_{eNB} is modeled as *k(MeNB, UE)*, using Scyther's predefined symmetric key infrastructure. *S-KeNB* denotes the session key derived from the K_{eNB}.

The property of *key secrecy* can be proven in the unbounded model from the local view of the roles MeNB and UE. In contrast, secrecy from the local view of SeNB can only be verified after introducing a bound. This is linked to a limitation of Scyther to process ticket variables, which have been used to model the session key from the perspective of SeNB. However, if this introduced bound is greater

than twice the number of involved roles (accordingly, greater than 6), the result can still be meaningful, as described in Section 3.2.4.

The following table shows the results of the formal verification of the dual-connectivity protocol with Scyther. It indicates whether each of the required properties could be modeled and if so, whether this property could be verified or falsified in the unbounded or bounded model.

Table 6.1: Scyther Verification Results

Property	**Modeling Possibility**	**Result**	**Performance**
Key Secrecy	Yes	Verified	Unbounded/Bounded Termination
Key Freshness	No	-	-
Key Possession	No	-	-

6.2.2.2 Tamarin Models

Similiar to the Scyther model (see Section 6.2.2.1), modeling in Tamarin requires several modifications of the generic formal model. In order to create secure channels, encryption was added to all messages transferred between MeNB and UE, and between MeNB and SeNB.

Several features of the generic model, which are not needed from a security point of view, were not included in order to keep the Tamarin models simple and increase the possibility of the tool successfully reaching termination. The DRB-IDs were abstracted away and only the SCC (counter) was implemented, since both inputs function in a similar way. The SCC is steadily increased until the range of the storing variable is exhausted, causing a reset of the counter to '0'. A reset of the SCC requires a refresh of the K_{eNB}, while a DRB-ID reuse leads to an increase of the SCC (implying a refresh of the S-K_{eNB}). Accordingly, it can be assumed as sufficient for a logical verification of the protocol's security to model only the SCC and ignore the DRB-IDs.

In order to model the dual-connectivity protocol in Tamarin, built-in functions for symmetric encryption and multiset modeling (needed for the modeling of counters) were used. Furthermore, the following new functions were defined:

k/1 : a symmetric key generation function, taking one value as input (unary function)

KDF/2 : a key derivation function, taking two values as input (binary function)

Two Tamarin models were implemented in order to show the behavior of the dual-connectivity protocol with regard to key freshness of the S-K_{eNB}. The first model illustrates that session key freshness fails if the SCC counter wraps around and neither the K_{eNB} is refreshed nor the UE is released. The related Tamarin Input File 1 (Without UE Release) can be found in Appendix B.1. A counterexample, showing the falsification of key freshness when checking Input File 1, is included in Appendix B.2.

The second Tamarin model implements the release of the UE as soon as the SCC is about to wrap around (described as option 2 in Section 6.1.2.4), accordingly session key freshness holds for this model. The related Tamarin Input File 2 (UE Release before SCC Wrap-Around) can be found in Appendix B.3.

6.2.2.2.1 Tamarin Model 1: Without UE Release

Figure 6.5 shows an example message flow of the Tamarin model 1 (Without UE Release).

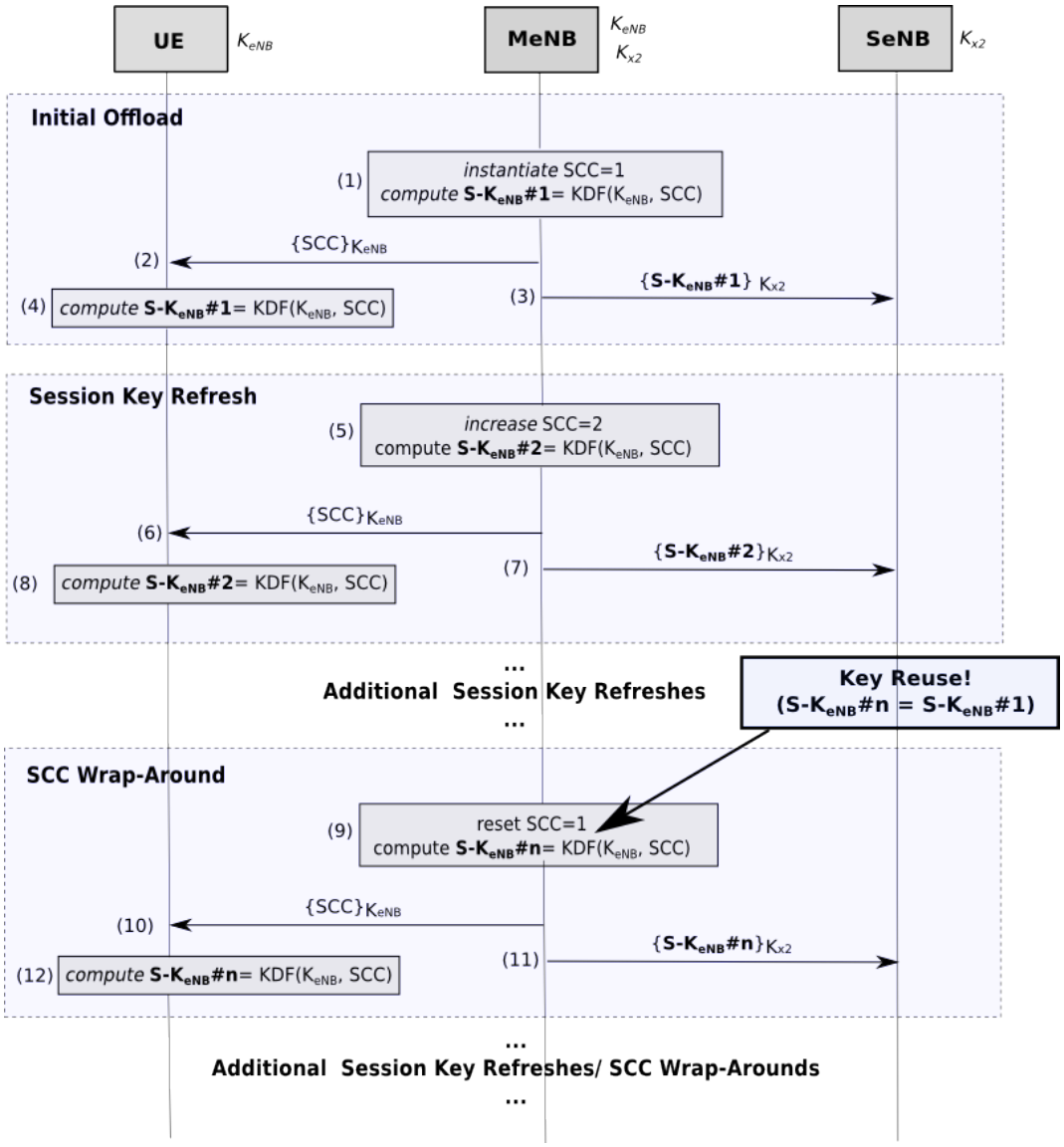


Figure 6.5: Dual-Connectivity Message Flow Example (Tamarin)

During the initial offload, the MeNB instantiates the SCC's value with '1' (implying the first bearer addition at a UE) and computes the first S-K_{eNB} (S-K_{eNB}#1) in step 1. In step 2 and step 3 the MeNB sends the SCC's value to the UE and the S-K_{eNB}#1 to SeNB. Following this, the UE can compute S-K_{eNB}#1

(using K_{eNB}) in step 4.

Subsequently, the SCC can either be increased or reset to '1'. If the SCC is increased, a new S-K_{eNB} is derived for a SCC with value 'SCC + 1' as described in step 5. Next, the increased SCC is sent to the UE in step 6 and the new S-K_{eNB}#2 is transmitted to SeNB. After that, the SCC might be increased $n<2^{16}$ times (since the SCC is a 16-bit counter), leading to the derivation of *n* new S-K_{eNB} values.

At some point in time the SCC is reset to '1' as illustrated in step 9. This represents the wrap-around of the SCC and leads to a reuse of the same S-K_{eNB} twice in a row, since the SCC becomes '1' again. Thus, the S-K_{eNB}#n has the same value as the initial S-K_{eNB}#1, which was derived during the initial offload. This circumstance is prevented by the Dual-Connectivity protocol by either refreshing the K_{eNB} (Option 1), releasing the UE before a possible SCC wrap-around (Option 2), or stopping the offloading of bearers (Options 3) as described in Section 6.1.2.4.

6.2.2.2.2 Tamarin Model 2: UE Release before SCC Wrap-Around

The second Tamarin input model was implemented in order to show that choosing one of the three options to prevent key stream reuse leads to verification of session key freshness. This model utilizes option 2. Thus, a case were the UE is released before the SCC wraps around is modeled. Accordingly, steps 9-12 in Figure 6.5 are never executed.

The reason why modeling option 2 was chosen is that this option leads to a simple Tamarin input file which terminates easily. When implementing option 1 (K_{eNB} refresh) the computing of a hashed key chain for the K_{eNB} led to a sophisticated input file, which did not terminate. The implementation of option 2 implicitly includes option 3 (no more bearer offloads) as well, since no more bearers for one specific dual-connecitivity context (between one SeNB and one UE) are offloaded when the UE is released. The input file 2 can be found in Appendix B.3.

6.2.2.2.3 Security Properties Modeling and Proving

All properties which were defined as design goals in Section 6.1.2.3 could be modeled in Tamarin. However, if the model becomes too complicated due to the use of several complex structures such as counters and loops, the likelihood for

non-termination increases. Despite the use of different heuristics (see Section 3.3.2.2) no termination for all required security properties could be reached when implementing the SCC, DRB-IDs, and the K_{eNB} refresh. Accordingly, simpler models were created, abstracting away the DRB-IDs and the K_{eNB} refresh.

No verification of *key secrecy* and *key possession* could be achieved with either of the two Tamarin input models due to non-termination of the tool. After the tool did not terminate for over 24 hours, it quitted automatically.

However, the property of *key freshness* could be falsified for input model 1 and verified for input model 2. This shows the need for preventing the S-K_{eNB} reuse (caused by a SCC wrap-around) by either refreshing the K_{eNB} (Option 1) or releasing the UE (Option 2). If option 2 is taken and the UE is released before the SCC wraps around (as it is done when executing input model 2), session key freshness can be verified. For both input models, the tool terminated within a few seconds for the key freshness property.

In order to model key freshness in Tamarin, additional constructs were used which do not exist in the actual protocol model. To prevent message replay, nonces and one time facts (facts which can only be consumed once) were added to each sent message. Additionally, a process identifier (pid) was introduced to differentiate between threads (see Section 3.1.2 for an explanation of threads) when testing for session key freshness. One pid is always related to one security association between one MeNB and one UE. The property of session key freshness is expressed as a Tamarin lemma as follows:

```
lemma key_freshness:
        "not(Ex actor key pid #n #m.
        SessionKey(actor, key, pid) @ n &
        SessionKey(actor, key, pid) @ m & n < m)"
```

The above lemma demands that no trace exists where the same session key k is reused by the same actor a with the same *pid* at different points in time n and m.

Three lemmas, which are not directly linked to any security property, were added to check the required properties of the SCC. The first lemma (“counters_linear_order”) proves whether the counter’s value increases in a linear order. The second lemma (“counter_reset”) assures that the initialization of the counter always takes place before the reset. The third lemma (“counter_increases”) checks that the counter initialization always preceded the counter increase. For the

first Tamarin Input File (without UE Release), all counter property related lemmas could be verified. In contrast, the lemma"counter_reset" cannot be proven for the second Tamarin Input File (UE Release before SCC Wrap-Around), since the UE is released before the counter wraps around and hence, no counter reset takes place.

The following table shows an overview of the required security properties, whether their modeling was possible in Tamarin, the verification result and the performance. The performance indicates whether the tool terminated for the respective property and if so, whether the termination could be reached in the bounded or unbounded model.

Table 6.2: Tamarin Verification Results

Property	**Modeling Possibility**	**Result**	**Performance**
Key Secrecy	yes	-	no termination
Key Freshness	yes	verified	unbounded termination
Key Possession	yes	-	no termination
Counter Linear Order	yes	verified	unbounded termination
Counter Reset	yes	verified	unbounded termination
Counter Increase	yes	verified	unbounded termination

Chapter 7

Evaluation of Applied Model Checking Tools

Within this chapter, the two used model checking tools Scyther and Tamarin are evaluated with regards to the following criteria:

Modeling possibilities and restrictions: Each tool should be evaluated as to whether the required concepts for protocol modeling such as control flows (e.g. loops, if-statements), state information, or cryptographic primitives can be implemented using the respective tool.

Usability: Each tool should be analyzed as to how easy it is to learn the tool input language of the respective tool. Moreover, the (graphical) user interface should be evaluated concerning the ease of use. The helpfulness of the user manual should be discussed.

Usefulness of results: Each tool should be assessed as to how useful the respective tool's output is in view of understandability and concreteness of the verification results.

Performance: The likelihood of the tool terminating in the bounded and unbounded model should be evaluated.

7.1 Evaluation of Scyther

Scyther had several *modeling restrictions*, as it was mainly designed for evaluation of key establishment protocols. Difficulties arise when implementing protocols for scenarios where keys or trust relations are already established. As an example,

it is infeasible to represent secure channels in Scyther.

Scyther relies on Lowe's definitions of authentication properties (see Section 3.1.4.2), but it is not clear how the definitions should be extended when using protocols with more than two communicating parties. Hence, there is no possibility to model injective agreement, weak agreement, or synchronization between only two parties when implementing a multi-party protocol in Scyther. For instance, in the case of a three-party protocol, agreement or synchronization between all three parties is required in order to achieve verification of the respective property, even if the property's proof is only needed between two parties. Additionally, no definition of custom equations is supported in Scyther and it is impossible to model custom security properties. Only the predefined security properties (see Section 3.3.1.2) are available.

Moreover, Scyther does not support the modeling of complex control structures such as counters, loops, or states, as already discussed in Section 6.2.2.1. Available options to model counter-like structures in Scyther are the use of either nonces or variables. When using the first option, the counter is implemented as a freshly generated value. In contrast, the second option implements the counter as a variable, which is received within a message and consumed from the network. However, since Scyther cannot keep a variable alive, this received value (in option two) can again only be either a freshly generated or random value from another party or the adversary. Hence, these two options of modeling counters are insufficient in cases where incrementing a counter and possibly repeating values of a counter variable need to be taken in account.

The *usability* of Scyther is high, since the tool's input language is intuitive and easy to learn. It was straight-forward to model the design model of the dual-connectivity-protocol in Scyther (in consideration of Scyther's limitation to model certain constructs). The graphical user interface is understandable and allows the user to set various options (for example a bound can be defined or the adversary model can be redefined). The Scyther user manual is still a draft, but it is helpful in terms of learning the protocol input syntax. Furthermore, several example protocol implementations are available as guidance.

The *tool output* of the verification results is understandable. It comprises a list of all requested security properties and the actual result (verification or falsification). It points out whether the termination was reached in the bounded or the unbounded model. In case of a property's falsification, counterexamples can be constructed and illustrated in a graphical way.

Regarding Scyther's *performance*, it could be observed that the termination in the unbounded model usually succeeds. This is due to the fact that the modeling of complex protocol structures is restricted by the limited input language. However, if ticket variables are used, then the tool only succeeds in the bounded model.

7.2 Evaluation of Tamarin

Tamarin's *modeling possibilities* turned out to be versatile. It is feasible to model and verify protocols using complex control flows such as branching and looping, which cannot be achieved when using Scyther. User-defined equational theories can be handled, thus the modeling of algebraic properties of cryptographic operations is possible. For instance, the modulo equational theory, bilinear pairing, and Diffie-Hellmann exponentiation can be specified. Additionally, the modeling of explicit states is provided; for instance, this enables the storing of the last value of a variable received by a role, which could be used to model concepts such as counters.

Tamarin supports a wide range of security properties, which can be defined in-depth by the user by using first-order logic, inferring quantifications over messages and time points. Hence, it is possible to define customized properties such as key freshness or diverse features of a counter (linear order of values, increasing values, etc.) as discussed in Section 6.2.2.2. Moreover, adversaries can be described in more detail than in Scyther, even though an extension of Scyther's adversary models has been suggested in [46]. For instance, a particularly strong intruder revealing random numbers and short-term and long-term keys can be described in Tamarin.

The *usability* of Tamarin is slightly lowered as it initially takes some time to become familiar with the protocol input language, especially for non-professionals or people without programming background. However, once the concept of the multi-set rewriting rules is understood, the syntax is intuitive to use. Tamarin offers a command line interface and an interactive mode with a user-friendly graphical user interface. The Tamarin manual and tutorial are still draft versions, but they are helpful when learning the input language. There are reference implementations of several real-world protocols available. However, these are mainly related to PKI protocols though.

The counter (SCC) is modeled as a persistent fact, storing an identifier *pid* and a value *scc*. This fact can be consumed and the counter's value can be increased or reset as described in the following listing.

```
rule CounterInit:
        [Fr(~pid)] //create a fresh pid
                -->
        [Counter(~pid, '1')]  //create a counter for pid
                              //and initialize it with '1'

rule CounterIncrease:
        [Counter(pid, scc)]
                -->
        [Counter(pid, scc + '1')] //increase counter's value

rule CounterReset:
        [Counter(pid, scc)]
                -->
        [Counter(pid, '1')] //reset counter's value to '1'
```

The *tool output* in the command-line interface is a listing of the properties and the respective verification results. Additionally, the steps of the tool's calculation are displayed, which are quite high-level and thus, hard to understand for a person who is not familiar with the internals of Tamarin's algorithm. Alternatively, the interactive mode offers the possibility of constructing graphical counterexamples of falsified security properties, which are easy to understand and helpful in many cases. A sample of such a counterexample can be found in Appendix B.2.

The biggest drawback of Tamarin is its *performance* as there is a high probability of non-termination when the input model becomes too complex. As soon as either several control structures (such as loops, and counters) are used within a protocol definition or numerous roles are defined, the likelihood of termination decreases. Hence, the tool does not scale very well. Tamarin offers different heuristics to increase the possibility of termination (see Section 3.3.2.3). However, these heuristics are still in an experimental state and no differences in terms of termination were identified when applying different heuristics to the protocol examined in this thesis project.

7.3 Evaluation Summary

The two applied model checking tools Scyther and Tamarin show different drawbacks and advantages. Scyther is easy to use due to its simplicity. On the

other hand, it reveals severe restrictions in protocol modeling as many concepts cannot be implemented. In contrast, Tamarin has a more sophisticated protocol input language and offers more possibilities to model real-world protocols (using loops and states). However, Tamarin does not scale well and tool termination can often not be reached when the protocol input model is too complex.

In summary, both tools revealed several problems when applied to a real-world LTE protocol. Neither tool is mature enough to capture what is needed for general use in the practical setting this thesis was written in. Too many concepts have to be abstracted away and still, termination often could not be reached. In general, it is advisable to choose a tool for protocol verification that offers a good trade-off between language complexity and modeling options.

Chapter 8

Conclusions

This chapter wraps up this thesis by discussing major results, insights, and possibilities for future work. Furthermore, reflections on the ethical and social impact of the work carried out are brought up.

8.1 Conclusion

This section discusses whether the initially claimed goals were reached and which insights were gained. Moreover, several suggestions for follow-up work are given.

8.1.1 Goals

All of the primary goals (described in Section 1.1) were achieved within this thesis. A design model of the dual-connectivity protocol was created (see Section 6.1) and a formal model derived (see Section 6.2.1). Following this, tool specific models were constructed, taking into account the possibilities and limitations of the chosen model checking tools: Scyther and Tamarin (see Section 6.2.2.1 and Section 6.2.2.2). These tool models served as a base for implementing protocol models (tool input files) using the tool specific input languages. The input files can be found in the Appendices A.1, B.1, and B.3. The verification results of these protocol models revealed no security flaws, hence no further refinement of the initial design model was required since the initial hypothesis (claiming that the dual-connectivity protocol is secure with regards to the formal model) has not been falsified. Finally, the applied tools and the general usefulness of formal methods for protocol standardization were evaluated in Section 7.1 and Section 7.2.

8.1.2 Insights and Suggestions for Further Work

During the work with formal methods, some restrictions of the applied model checking tools were encountered. These restrictions made it difficult to properly model the LTE dual-connectivity protocol with the chosen tools. The main drawback of Scyther turned out to be its focus on key exchange protocols, which introduced limitations when modeling other types of protocols. In contrast, Tamarin offers plenty of possibilities to model various protocols. However, the tool does not scale well. The likelihood of non-termination of Tamarin increases as soon as the protocol input models become too complex.

The observed limitations of the applied model checking tools slightly hinder the initially assumed benefits of using these tools in the process of protocol standardization. Still, in general formal methods can be highly useful since they imply a careful consideration of a protocol's fundamentals when designing a formal model. The associated reflection on the logical blocks constituting a protocol can be considered as very helpful with regard to improving and structuring the process of protocol design. Hence, increased usage of formal methods in protocol standardization is recommended in conjunction with current practices.

8.2 Future Work

As the model checkers used within this thesis project revealed several limitations, an extension of these tools with regards to the discovered weaknesses would be desirable. Since Tamarin is an extension of Scyther and offers versatile options to model real-world protocols, it seems reasonable to put effort into extending the Tamarin tool.

Furthermore, it would be conceivable to study the termination problems of Tamarin in-depth. Additional modifications of the current dual-connectivity protocol models could be tried out to possibly achieve termination for those properties, which did not terminate yet. Prospectively, it would be feasible to model and verify the dual-connectivity protocol with different model checking tools in addition to Scyther and Tamarin.

The gained knowledge of model checkers could be used to formalize and verify protocols during their standardization process in conjunction to current practices. Potentially, formal verification could also be applied to protocols,

developed in a different area than LTE. These protocols might be more suitable for model checking tools and hence, the verification results might be more helpful.

8.3 Required Reflections

The formal modeling and verification of the dual-connectivity protocol carried out within this thesis project raises and solves several economic and ethical issues. Using formal methods can speed up and improve the standardization process of a protocol as it outputs real attack patterns in the presence of flaws in the protocol design. This offers the possibility of detecting and then repairing possible weaknesses earlier in the development process.

However, a problem may arise due to the fact that the construction of protocol input models is subject to possible human failure. Although clear rules are defined for describing formal models, a margin is left when it comes to the definition of desired security properties and adversary models. This can affect the final verification result and in the worst case, falsify the result (for instance if an overly weak adversary model is defined or the definition of necessary security properties is left out).

The protocol, which has been verified, handles the security of the dual-connectivity protocol which currently is in the middle of its standardization process. The evaluation presented in this thesis assures that non-authorized entities are hindered from eavesdropping and altering communication and thereby violating the security and privacy of the user.

In general, the dual-connectivity protocol facilitates load balancing between base stations (MeNBs and SeNBs), thus enabling efficient usage of available resources at the SeNBs. Through bearer handovers, established by the dual-connectivity protocol, better system performance and better use of cell capacity can be achieved. This offers increased user throughput (on both, upload and download links) while maintaining user mobility. [47, 48]

Bibliography

[1] G. Lowe, "An attack on the needham-schroeder public-key authentication protocol," *Inf. Process. Lett.*, vol. 56, no. 3, p. 131–133, Nov. 1995.

[2] B. Blanchet, "Automatic verification of correspondences for security protocols," *J. Comput. Secur.*, vol. 17, no. 4, p. 363–434, Dec. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1576303.1576304

[3] D. Basin, C. Cremers, and C. Meadows, "Model checking security protocols," in *Handbook of Model Checking*, E. Clarke, T. Henzinger, and H. Veith, Eds. Springer, 2012, to appear.

[4] Cas Cremers, "The scyther tool: Verification, falsification, and analysis of security protocols (tool paper)," in *Proceedings of the 20th International Conference on Computer Aided Verification*, ser. CAV '08. Berlin, Heidelberg: Springer-Verlag, p. 414–418. [Online]. Available: http://www.cs.ox.ac.uk/people/cas.cremers/downloads/papers/Cr2008-Scyther_tool.pdf

[5] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *Computer Aided Verification, 25th International Conference, CAV 2013, Princeton, USA, Proc.*, ser. Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds., vol. 8044. Springer, 2013, pp. 696–701.

[6] Benedikt Schmidt, "Formal analysis of key exchange protocols and physical protocols," Ph.D. dissertation, ETH Zurich, Information Security Institute, 2012.

[7] C. Boyd and A. Mathuria, *Protocols for Authentication and Key Establishment*, 1st ed. Springer Publishing Company, Incorporated, 2010. ISBN 3642077161, 9783642077166

[8] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, pp. 644–654, Sep. 2006.

[9] A. J. Menezes, P. C. V. Oorschot, S. A. Vanstone, and R. L. Rivest, "Handbook of applied cryptography," 1997.

[10] D. Dolev and A. C. Yao, "On the security of public key protocols," *Information Theory, IEEE Transactions on*, vol. 29, no. 2, pp. 198–208, Mar. 1983.

[11] J. Massey, "An introduction to contemporary cryptology," *Proceedings of the IEEE*, vol. 76, no. 5, pp. 533–549, May 1988.

[12] U. Carlsen, "Cryptographic protocol flaws: know your enemy," in *Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings*, Jun. 1994, pp. 192–200.

[13] M. Abadi and R. Needham, "Prudent engineering practice for cryptographic protocols," *IEEE Trans. Softw. Eng.*, vol. 22, no. 1, p. 6–15, Jan. 1996.

[14] G. Lowe, "A hierarchy of authentication specifications, 10th computer security foundations workshop." IEEE Computer Society Press, 1997, p. 31–43. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=596782

[15] L. Gong, "A variation on the themes of message freshness and replay or, the difficulty in devising formal methods to analyze cryptographic protocols." in *Computer Security Foundations Workshop*. IEEE Computer Society, 1993. ISBN 0-8186-3950-4 pp. 131–136.

[16] P. Janson and G. Tsudik, "Secure and minimal protocols for authenticated key distribution," *Computer Communications Journal*, vol. 18, p. 645–653, 1995.

[17] 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*, Jun. 2012. doi: 10.1109/CSF.2012.25 pp. 78–94.

[18] D. Basin and C. Cremers, "Modeling and analyzing security in the presence of compromising adversaries," vol. 6345, pp. 340–356, 2010.

[19] C. Cremers and S. Mauw, "Operational semantics of security protocols," in *Scenarios: Models, Transformations and Tools, International Workshop, Dagstuhl*. Springer, 2005, p. 66–89.

[20] Cas Cremers, "Scyther - semantics and verification of security protocols," Ph.D. dissertation, Eindhoven University of Technology, Institute for Programming research and Algorithmics, 2006.

[21] S. Andova, C. Cremers, K. Gjøsteen, S. Mauw, S. F. Mjølsnes, and S. Radomirović, "A framework for compositional verification of security protocols," *Inf. Comput.*, vol. 206, no. 2-4, pp. 425–459, Feb. 2008. doi: 10.1016/j.ic.2007.07.002. [Online]. Available: http://dx.doi.org/10.1016/j.ic.2007.07.002

[22] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *ACM Trans. Comput. Syst.*, vol. 8, no. 1, p. 18–36, Feb. 1990.

[23] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Advances in Cryptology — CRYPTO' 93*, ser. Lecture Notes in Computer Science, D. Stinson, Ed. Springer Berlin Heidelberg, 1994, vol. 773, pp. 232–249. ISBN 978-3-540-57766-9

[24] M. Rusinowitch and M. Turuani, "Protocol insecurity with finite number of sessions is NP-complete," in *Theoretical Computer Science*, 2001, p. 174–190.

[25] C. J. Cremers, "Unbounded verification, falsification, and characterization of security protocols by pattern refinement," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, ser. CCS '08. New York, NY, USA: ACM, 2008. ISBN 978-1-59593-810-7 p. 119–128.

[26] S. Escobar, C. Meadows, and J. Meseguer, "A rewriting-based inference system for the NRL protocol analyzer and its meta-logical properties," *Theor. Comput. Sci.*, vol. 367, no. 1, pp. 162–202, Nov. 2006. doi: 10.1016/j.tcs.2006.08.035. [Online]. Available: http://dx.doi.org/10.1016/j.tcs.2006.08.035

[27] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. C. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron, "The AVISPA tool for the automated validation of internet security protocols and applications," in *Proceedings of the 17th International Conference on Computer Aided Verification*, ser. CAV'05. Berlin, Heidelberg: Springer-Verlag, 2005. doi: 10.1007/11513988_27. ISBN 3-540-27231-3, 978-3-540-27231-1 pp. 281–285. [Online]. Available: http://dx.doi.org/10.1007/11513988_27

[28] D. X. Song, "Athena: A new efficient automatic checker for security protocol analysis," in *Proceedings of the 12th IEEE Workshop on Computer Security Foundations*, ser. CSFW '99. Washington, DC, USA: IEEE Computer Society, 1999, pp. 192–.

[29] B. Blanchet, "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," in *Computer Security Foundations Workshop, 2001. Proceedings. 14th IEEE*, 2001, pp. 82–96.

[30] Cas Cremers, "Scyther user manual," Dec. 2013. [Online]. Available: https://github.com/cascremers/scyther/blob/master/gui/scyther-manual.pdf

[31] Simon Meier and Benedikt Schmidt, "Tamarin prover user manual," Sep. 2012. [Online]. Available: https://github.com/tamarin-prover/tamarin-prover/blob/develop/data/doc/MANUAL

[32] Simon Meier, "Tamarin tutorial," Sep. 2012. [Online]. Available: https://github.com/tamarin-prover/tamarin-prover/blob/develop/data/examples/Tutorial.spthy

[33] K. Bhargavan, C. Fournet, A. D. Gordon, and S. Tse, "Verified interoperable implementations of security protocols," *ACM Trans. Program. Lang. Syst.*, vol. 31, no. 1, p. 5:1–5:61, Dec. 2008. doi: 10.1145/1452044.1452049. [Online]. Available: http://doi.acm.org/10.1145/1452044.1452049

[34] R. Canetti and J. Herzog, "Universally composable symbolic analysis of mutual authentication and key-exchange protocols," in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, S. Halevi and T. Rabin, Eds. Springer Berlin Heidelberg, 2006, vol. 3876, pp. 380–403. ISBN 978-3-540-32731-8. [Online]. Available: http://dx.doi.org/10.1007/11681878_20

[35] Itsuki Suzuki, Yoshiki Kamano, Maki Yoshida, and Toru Fujiwara, "Development of a verification tool for composable security," Aug. 2004. [Online]. Available: http://www.rcis.aist.go.jp/files/events/csps2009/schedule/CoSyProofs2009_Suzuki.pdf

[36] K. N. Noomene Ben Henda, "Formal analysis of security procedures in lte - a feasibility study." RAID 2014, 2014.

[37] C. Cremers, "Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2," in *Computer Security – ESORICS 2011*, ser. Lecture Notes in Computer Science, V. Atluri and C. Diaz, Eds. Springer Berlin Heidelberg, 2011, vol. 6879, pp. 315–334. ISBN 978-3-642-23821-5

[38] D. Basin, C. Cremers, and S. Meier, "Provably repairing the ISO/IEC 9798 standard for entity authentication," in *Principles of Security and Trust*, ser. Lecture Notes in Computer Science, P. Degano and J. Guttman, Eds. Springer Berlin Heidelberg, 2012, vol. 7215, pp. 129–148. ISBN 978-3-642-28640-7

[39] International Organization for Standardization, "ISO/IEC 9798-1:2010, information technology (security techniques entity authentication) part 1: General," 2010, third Edition.

[40] R. Künnemann and G. Steel, "YubiSecure? formal security analysis results for the yubikey and YubiHSM," in *Security and Trust Management*, ser. Lecture Notes in Computer Science, A. Jøsang, P. Samarati, and M. Petrocchi, Eds. Springer Berlin Heidelberg, 2013, vol. 7783, pp. 257–272. ISBN 978-3-642-38003-7

[41] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Proceedings of the 1st International Conference on Provable Security*, ser. ProvSec'07. Berlin, Heidelberg: Springer-Verlag, 2007, p. 1–16.

[42] Arpit and A. Kumar, "Verification of elgamal algorithm cryptographic protocol using linear temporal logic," in *Multimedia Technology (ICMT), 2011 International Conference on*, Jul. 2011. doi: 10.1109/ICMT.2011.6002114 pp. 6662–6665.

[43] Henry Petroski, "Reference guide on engineering practice and methods," Washington, D.C.: Federal Judicial Center, p. 577–624, 2000, second Edition.

[44] 3GPP, "Living SuperCR on dual connectivity and SCE," 3rd Generation Partnership Project (3GPP), CR 33.401, Apr. 2014. [Online]. Available: http://www.3gpp.org/ftp/tsg_sa/WG3_Security/TSGS3_74b_Sophia/Docs/S3-140578.zip

[45] ——, "Handling the Small Cell Counter (SCC) for S-KeNB Derivations," 3rd Generation Partnership Project (3GPP), S3 140583, Apr. 2014. [Online]. Available: http://www.3gpp.org/ftp/tsg_sa/WG3_Security/TSGS3_74b_Sophia/Docs/S3-140583.zip

[46] D. Basin and C. Cremers, "Degrees of security: Protocol guarantees in the face of compromising adversaries," in *Computer Science Logic*, ser. Lecture Notes in Computer Science, A. Dawar and H. Veith, Eds. Springer Berlin Heidelberg, 2010, vol. 6247, pp. 1–18.

[47] 3GPP, "Technical specification group radio access network; study on small cell enhancements for e-UTRA and e-UTRAN; higher layer aspects (release 12)," 3rd Generation Partnership Project (3GPP), TR 36.842, Dec. 2013. [Online]. Available: http://www.3gpp.org/ftp/Specs/archive/36_series/36.842/36842-c00.zip

[48] ——, "Technical specification group radio access network," 3rd Generation Partnership Project (3GPP), TR 36.932, Mar. 2013. [Online]. Available: http://www.3gpp.org/ftp/Specs/archive/36_series/36.932/36932-c10.zip

Appendix A

Scyther

A.1 Scyther Input File

```
hashfunction KDF;
macro SK-eNB = KDF(k(MeNB,UE), scc);

protocol dual-connectivity (UE, MeNB, SeNB)
{

role UE {
                fresh n-ue: Nonce;
                var scc: Nonce;

                recv_1(MeNB, UE, {scc}k(MeNB,UE));
                send_3(UE, SeNB, {n-ue}SK-eNB);

                claim (UE, Secret, k(MeNB,UE));
                claim (UE, Secret, n-ue);
        }


role MeNB {
                fresh scc: Nonce;

                send_1(MeNB, UE, {scc}k(MeNB,UE));
                send_2(MeNB,SeNB,{SK-eNB}k(MeNB,SeNB));

                claim (MeNB, Secret, k(MeNB,UE));
                claim (MeNB, Secret, SK-eNB);
        }
```

```
role SeNB {
                var SK-eNB-SeNB: Ticket;
                var n-ue: Nonce;

                recv_2(MeNB,SeNB,{SK-eNB-SeNB}k(MeNB,SeNB));
                recv_3(UE, SeNB, {n-ue}SK-eNB-SeNB);

                claim(SeNB, Secret, SK-eNB-SeNB);
                claim (SeNB, Secret, n-ue);
        }
}
```

Appendix B

Tamarin

B.1 Tamarin Input File 1: Without UE Release

```
theory MyDualConnectivity

begin

functions: KDF/1
builtins: multiset, hashing, symmetric-encryption

// Counter creation and Provisioning of symmetric keys
rule CounterInit:
        [Fr(~pid),
        Fr(~ltk)]
                --[Start(~pid)]->
        [Counter(~pid, '1'),
        !Ltk($A, $B, ~pid, h(~ltk)),
        !Pid(~pid)]

// Offload start (scc counter increment)
rule MeNB_Offload_Cmd_Inc:
let
        SKenb = KDF(<Kenb, scc>)
in
        [!Pid(pid),
        Fr(~n),
        Counter(pid, scc),
        !Ltk(MeNB, UE, pid, Kenb),
        !Ltk(MeNB, SeNB, pid, Kx2)]
                --[Inc(pid, scc)]->
        [Out(senc{<scc, ~n>}Kenb),
        OneTime_Offload_UE(pid, scc, ~n), //prevent replay
        Out(senc{<SKenb, ~n>}Kx2),
```

```
        OneTime_Offload_SeNB(pid, scc, ~n),
        Counter(pid, scc + '1')]

rule MeNB_Offload_Cmd_Reset:
let
        SKenb = KDF(<Kenb, scc>)
in
        [!Pid(pid),
        Fr(~n2),
        Counter(pid, scc),
        !Ltk(MeNB, UE, pid, Kenb),
        !Ltk(MeNB, SeNB, pid, Kx2)]
                --[Reset(pid, scc)]->
        [Out(senc{<scc, ~n2>}Kenb),
        OneTime_Offload_UE(pid, scc, ~n2), //prevent replay
        Out(senc{<SKenb, ~n2>}Kx2),
        OneTime_Offload_SeNB(pid, scc, ~n2),
        Counter(pid, '1')]

rule UE_Offload_Cmd:
let
        SKenb = KDF(<Kenb, scc>)
in
        [OneTime_Offload_UE(pid, scc, n),
        !Ltk(MeNB, UE, pid, Kenb),
        In(senc{<scc, n>}Kenb)]
                --[SessionKey(UE, SKenb, pid)]->
        []

rule SeNB_Offload_Cmd:
        [!Ltk(MeNB, SeNB, pid, Kx2),
        OneTime_Offload_SeNB(pid, scc, n),
        In(senc{<SKenb, n>}Kx2)]
                -->
        []

rule KeyReveal:

        [!Pid(pid),
        !Ltk(A, B, pid, k)]
                --[LtkReveal()]->
        [Out(k)]



lemma counters_linear_order[use_induction]:
  "All x y #n #m s.
      Inc(s, x) @ n & Inc(s, y) @ m
      ==> (Ex z. x + z = y) | (Ex z. y + z = x) | y = x"
```

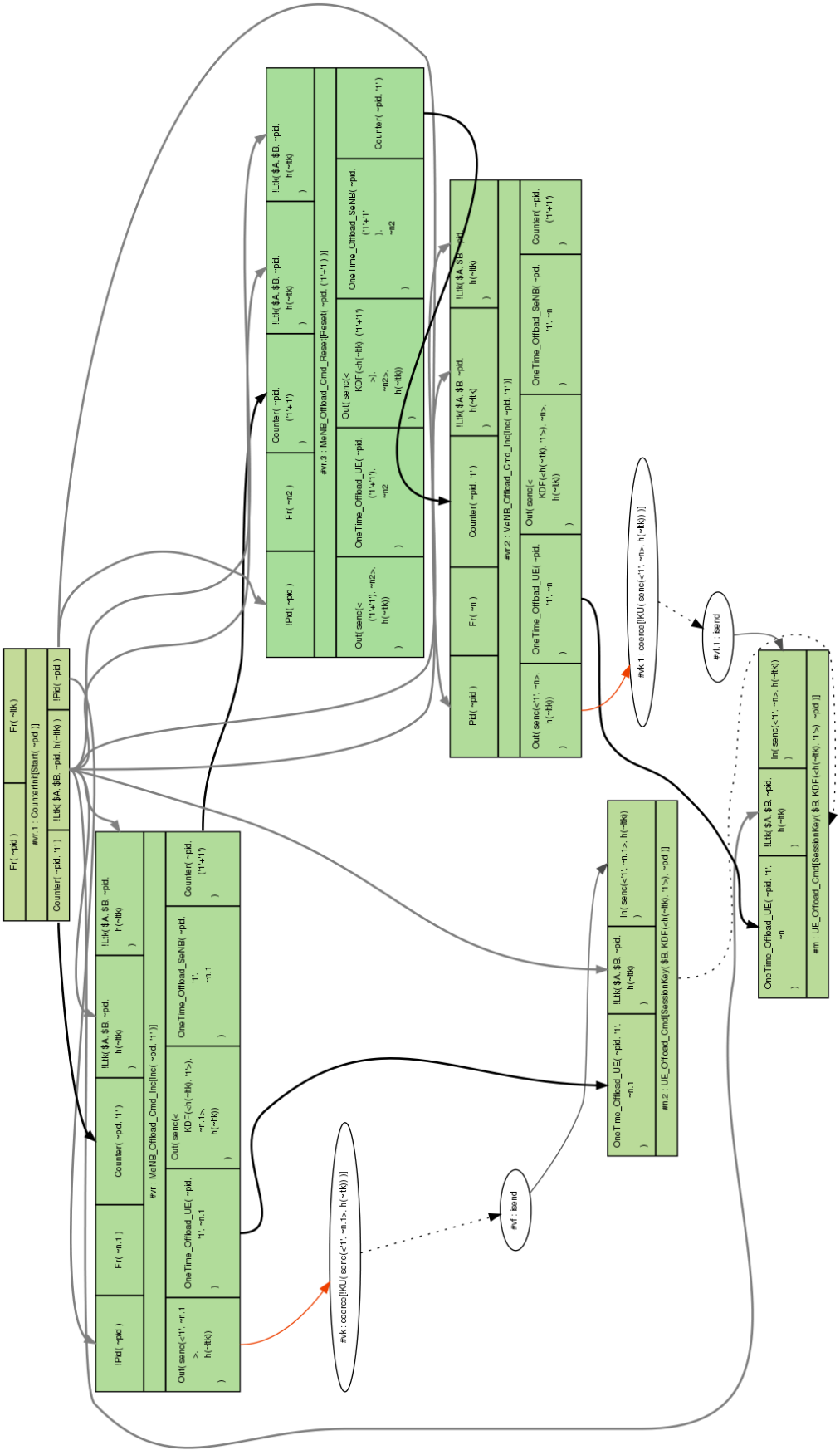
```
lemma counter_reset:
  exists-trace
  "Ex s x #n #m.
      Start(s) @ n & Reset(s, x) @ m & n < m"

lemma counter_increases:
  exists-trace
  "Ex s x #n #m.
      Start(s) @ n & Inc(s, x) @ m & n < m"


lemma key_freshness:
  "not(Ex a k pid #n #m.
        SessionKey(a, k, pid) @ n &
        SessionKey(a, k, pid) @ m & n < m)"

end
```

B.2 Tamarin Counterexample (Input File 1)



B.3 Tamarin Input File 2: UE Release before SCC Wrap-Around

```
theory MyDualConnectivity

begin

functions: KDF/1
builtins: multiset, hashing, symmetric-encryption


// Counter creation and Provisioning of symmetric keys
rule CounterInit:

        [Fr(~pid),
        Fr(~ltk)]
                --[Start(~pid)]->
        [Counter(~pid, '1'),
        !Ltk($A, $B, ~pid, h(~ltk)),
        !Pid(~pid)]


// Offload start (scc counter increment)
rule MeNB_Offload_Cmd_Inc:

let
        SKenb = KDF(<Kenb, scc>)
in
        [!Pid(pid),
        Fr(~n),
        Counter(pid, scc),
        !Ltk(MeNB, UE, pid, Kenb),
        !Ltk(MeNB, SeNB, pid, Kx2)]

                --[Inc(pid, scc)]->

        [Out(senc{<scc, ~n>}Kenb),
        OneTime_Offload_UE(pid, scc, ~n), //prevent replay
        Out(senc{<SKenb, ~n>}Kx2),
        OneTime_Offload_SeNB(pid, scc, ~n),
        Counter(pid, scc + '1')]
```

```
rule UE_Offload_Cmd:

let
        SKenb = KDF(<Kenb, scc>)
in
        [OneTime_Offload_UE(pid, scc, n),
        !Ltk(MeNB, UE, pid, Kenb),
        In(senc{<scc, n>}Kenb)]
                --[SessionKey(UE, SKenb, pid, n)]->
        []

rule SeNB_Offload_Cmd:
        [!Ltk(MeNB, SeNB, pid, Kx2),
        OneTime_Offload_SeNB(pid, scc, n),
        In(senc{<SKenb, n>}Kx2)]
                -->
        []

rule KeyReveal:
        [!Pid(pid),
        !Ltk(A, B, pid, k)]
                --[LtkReveal()]->
        [Out(k)]


lemma counters_linear_order[use_induction]:
  "All x y #n #m s.
      Inc(s, x) @ n & Inc(s, y) @ m
      ==> (Ex z. x + z = y) | (Ex z. y + z = x) | y = x"

lemma counter_increases:
  exists-trace
  "Ex s x #n #m.
      Start(s) @ n & Inc(s, x) @ m & n < m"

lemma key_freshness:
  "not(Ex a k pid nonce #n #m.
        SessionKey(a, k, pid, nonce) @ n &
        SessionKey(a, k, pid, nonce) @ m & n < m)"

end
```

TRITA-ICT-EX-2014:94