Formal Analysis of Security Procedures in LTE - A Feasibility Study

Noomene Ben Henda and Karl Norrman

Ericsson Research Stockholm, Färögatan 6 16480, SWEDEN
noamen.ben.henda@ericsson.com
karl.norrman@ericsson.com

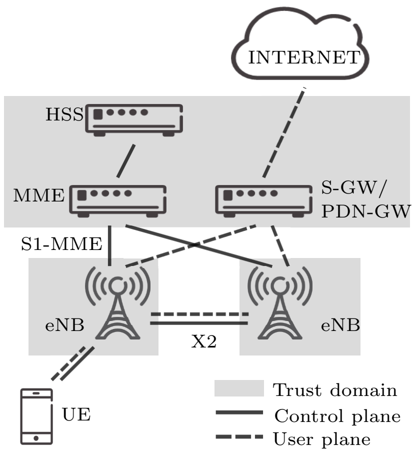
Abstract. The only part of the Long Term Evolution (LTE) security standard that has been formally analyzed is the Authentication and Key Agreement (AKA) procedure. It is not clear how well existing security related verification tools can handle other types of procedures. In this work, we use ProVerif to analyze the procedures related to session management and mobility. Our analysis has shown that most of the secrecy and agreement properties hold which was expected. However, we had difficulties proving stronger injective agreement properties.

Keywords: Formal verification, Telecom, LTE, security

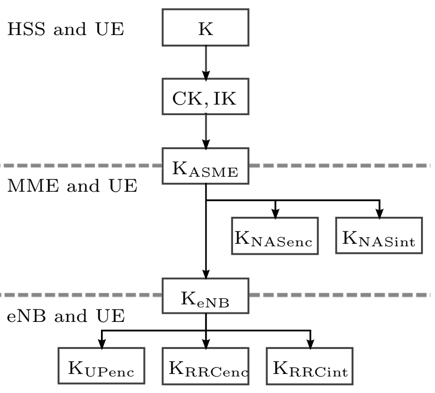
1 Introduction

Background. Long Term Evolution (LTE), a 4th Generation (4G) mobile communication system, is the most recent standard developed by the 3rd Generation Partnership Project (3GPP) [1]. Among the objectives of LTE is to provide higher data rates, enhanced quality of service and equal or better security compared to previous generations [1] (TS 22.278). One such improvement is that LTE introduces very granular key separation. LTE mandates the use of different session keys for specific protocols and purposes between the terminal and the nodes in the network. Those keys are organized in a hierarchy (see Fig. 1b). At the root of the hierarchy is a key that is shared between the Home Subscriber Server (HSS) (see Fig. 1a) and the terminal, or User Equipment (UE) in the 3GPP specifications, where it is securely kept in a smartcard. During initial attachment of the UE to the network, mutual authentication between them is achieved by running the Authentication and Key Agreement protocol (AKA) [1] (TS 33.401). The authentication is based on the root key. The other keys are subsequently derived from keys that are closer to the root in the hierarchy than themselves.

Each key in the hierarchy is shared between the UE and a particular node in the network. For example the K_{ASME} key is shared with the Mobility Management Entity (MME); the K_{eNB} key is shared with the Evolved Node B (eNB). The LTE standard defines specific procedures for the establishment of each key. For instance, the K_{ASME} key is established by the AKA protocol which runs between the UE and the HSS, and then provisioned to the target MME node. The K_{eNB} is initially established by a combination of procedures involving the MME, eNB, and the UE. The UE and MME use the K_{ASME} to agree on a K_{eNB}. The MME then provides this key to the eNB which finally activates the security between the UE and the eNB based on the K_{eNB}. Key establishment procedures like these typically have to satisfy at least the following security properties: *agreement*, *secrecy* and *freshness*. Agreement is the property that guarantees that the involved parties obtain the same key at the end of the run; otherwise, the key would be useless. Secrecy guarantees that no one, other that the involved parties (who are assumed to not leak the key to outsiders), has the key. If secrecy is not guaranteed, confidentiality protection, among other cryptographically based services, is not achievable. The last property of freshness prevents key re-use and thus, for example, situations were a plain text is encrypted twice using the same key.



(a) Simplified architecture of LTE



(b) Key hierarchy in LTE

Fig. 1: LTE overview

In a running system, the key establishment procedures and procedures making use of the keys can be interleaved, repeated and run simultaneously by several UEs and network nodes. They can as well be used as building blocks in more complex compound procedures such as the ones handling mobility. The security procedures are dependent on each other. For instance, the establishment of a K_{eNB} key requires the existence of a K_{ASME} key and thus any procedure using the K_{eNB} cannot be executed in a pure LTE system unless an AKA run has taken place earlier. The procedures might also rely on other type of context information, such as message counters and global parameters of the system. State-based formal verification tools like SPIN [18] can model this context information and capture the effect of reruns and interleaving. However state-based approaches are not effective to model cryptographic functions that usually rely on advanced computations. Other symbolic approaches that abstract away the implementation details of cryptographic primitives have been more effective. In general, formal analysis of security protocols is usually done against the symbolic Dolev-Yao intruder (or attacker) model [16]. In this model, the attacker has full control of the communication medium. In addition, cryptography is assumed perfect so that the attacker cannot decrypt messages unless he has the required key, hash functions are collision free, etc.

Contribution. In this paper, we present our work with ProVerif [8] which we used to model and verify security properties of different key establishment procedures in LTE. This work is part of a feasibility study whose aim is to bring and put to use tools like ProVerif in an industrial context such as that of the 3GPP standardization process. Our main contribution consists in providing formal models of the LTE protocols in the input language of the ProVerif. Our implementation preserves the trust model of 3GPP. Furthermore, to the best of our knowledge, the security procedures related to mobility and session management have not been previously subject to formal analysis. Another contribution consists in showing how to model and verify different security properties. Our analysis results confirm all secrecy and most of the weak agreement properties. However, stronger agreement properties are more challenging to prove, for several reasons that we later discuss and explain. Our analysis approach using ProVerif is simple and generic and thus can be easily adapted to other case studies.

Related Work. Although LTE security has received much scrutiny during the design process, it has been less studied in the research community. In particular, the research community has mainly focused on analyzing AKA [29, 28, 17, 30], which is largely the same authentication and key agreement protocol used for Wideband Code Division Multiplex Access (WCDMA), a 3G access. AKA as used in WCDMA was formally analyzed using BAN logic in [1] (TS 33.902). AKA is re-used exactly as is in LTE to boot strap the key hierarchy. Therefore, all analysis results on AKA as used in WCDMA carries over to LTE. A study of privacy aspects of WCDMA is presented in [5]. Although it does not study LTE, it looks at other procedures than AKA, namely the paging procedure. The study in [24] contains an analysis of a proposed, but not standardized, system for handovers between different types of radio access systems. It does not provide any analysis of LTE itself.

Research on formal verification of security protocols has been ongoing for two decades. Current state of the art tools like Scyther [13] and ProVerif can verify protocols for unbounded number of sessions and agents. Case studies by Scyther include the analysis of the Naxos protocol [14] and the IPsec exchange protocols IKEv1 and IKEv2 [15]. Other applications include the analysis of the privacy and key management protocol [27] and the handover schemes [26] in WiMAX networks. Case studies by ProVerif include the analysis the Bluetooth device pairing protocol [19], the just-fast-keying protocol [3], a secure file sharing protocol [9], authentication in 3G where both GSM and WCDMA access is used [28], and the privacy study on WCDMA mentioned earlier [5]. We note that [28] does not contain any analysis of mobility between radio access networks, but rather considers the case of authentication over the GSM/EDGE access network, when used to access a 3G network. There is a long list of similar tools from which we cite the following ones: The Tamarin tool [23] has been used for the verification of group key agreement protocols [25]. The AVISPA [6] has been used for the analysis of key management in hierarchical group protocols [12]. Other relevant tools are NRL [22], LySa [11] and Casper [21].

Outline. In the next section, we give an overview of the LTE architecture to put in context the protocol models we provide. In Section 3, we describe ProVerif and use AKA as an example illustrating our modeling approach. In Section 4, we describe the security procedures related to session management, provide the corresponding formal models and discuss the verification results. In Section 5, we present our work on security procedures in mobility events. Finally in Section 6, we conclude by a summary discussions and future work. For shortage of space, the full versions of the models that can be used to reproduce our results are not included. They are available on demand. In the description of the LTE procedures, many aspects not related to security have been omitted and thus we refer to [1] for the detailed specifications.

2 Overview of LTE

LTE provides 4G mobile broadband access service to terminals. More precisely, the service consists of providing a terminal with IP connectivity using a stable IP address, while the terminal moves throughout the LTE network.

2.1 Architecture

LTE [1] (TS 23.401) consists of a Radio Access Network (RAN) and a core network (see Fig. 1a). The radio access network consists of a set of base stations, the eNBs. The terminals connect to the eNB via the radio air interface. The eNB is connected to two nodes in the core network: the MME and the Serving Gateway (S-GW). The first node (MME) handles the control plane traffic for mobile terminals connected to the eNB. The control plane for a terminal is used to manage the terminal sessions, mobility and security. The second node (S-GW) handles the user plane traffic to and from the internet, and other operator services. Subscriber information such as authentication credentials, location, subscription preferences, etc. is kept in the HSS.

2.2 Trust Model

During the security design in 3GPP a trust model for the network is assumed. More precisely, the network is divided into two main types of trust domains: the core network trust domain, and the RAN one. The standards have a more granular concept of trust domains, but these two are sufficient for the protocols considered in this paper. The data traffic flows between nodes in different domains over an IP transport network.

The core network domain, which contains nodes like the HSS, MME, S-GW, etc. is assumed to be a physically secure one. This means that attackers do not have access to nodes in this domain other than what can be obtained remotely via the network interfaces of the nodes.

The RAN trust domain contains only the eNBs. Since such nodes may be deployed in physically insecure locations, such as on the wall of a shopping mall, or in a hotel corridor, etc. the security model in LTE is built to handle the situation where the eNBs are deployed in untrusted locations. In the standard [1] (TS 33.401), it is required that each eNB implements its security processing inside a secure environment. The purpose is to prevent attackers to gain access to any data in the eNB by physically tampering with the device. Furthermore, the IP transport network that connects nodes across different domains is to be protected using IPsec [1] (TS 33.210) unless it can be trusted.

2.3 Session and Mobility Management

The terminal maintains two control connections with the network, one with the MME managed by the Non Access Stratum (NAS) protocol, and one with the eNB managed using the Radio Resource Control (RRC) protocol. The MME keeps track of the terminal location even when it is idle, i.e., it is not exchanging user plane data. The location is defined by an area served by possibly several eNBs. The terminal keeps the MME updated of any area changes as it moves.

In case of incoming data, the MME pages the terminal on all eNBs in its last known area. In response to the paging, the terminal requests a user plane data connection from the MME. It is only then that the eNB, which the terminal uses to access the network, becomes aware of the terminal presence. The MME provides the eNB with initial state information to communicate with the terminal. The terminal can then become active sending and receiving data. Afterwards, it can become idle again. In such case, the serving eNB releases all the associated resources and is no longer aware of the terminal's presence.

2.4 Key Hierarchy

Once security is activated, the NAS protocol between the terminal and the MME becomes both integrity protected and encrypted. The same holds for the RRC protocol between the terminal and the eNB. The user plane traffic is encrypted in two hops. First the radio link between the terminal and the eNB is encrypted. The eNB terminates the encryption of uplink traffic inside its secure environment and forwards it to the S-GW through an IPsec tunnel. Downlink traffic is handled in a similar manner.

Security for NAS, RRC and user plane traffic relies on separate encryption and integrity session keys (see Fig. 1b). The keys for protecting RRC and user plane traffic are derived from the K_{eNB} which in turn, is derived from the K_{ASME}. The keys for the NAS protocol are also derived from the K_{ASME} key.

2.5 Initial Key Establishment

At start up, the terminal needs to register with the network. This is achieved by the attach procedure. In connection to the attach procedure, the terminal and network also run an AKA procedure. The outcome of AKA is the establishment of the K_{ASME} session key between the terminal and the serving MME.

Figure 2 contains a simplified chart of the message exchange related to AKA and that we briefly explain as follows: First, the UE sends its identifier IMSI and security capabilities to the MME in an attach request. The MME then stores the capabilities and forwards the IMSI to the HSS. The HSS uses the identifier to retrieve the secret subscriber key K, generates a nonce RAND and computes the K_{ASME} key together with other authentication parameters. The authentication data is then sent to the MME which uses it to authenticate the UE.

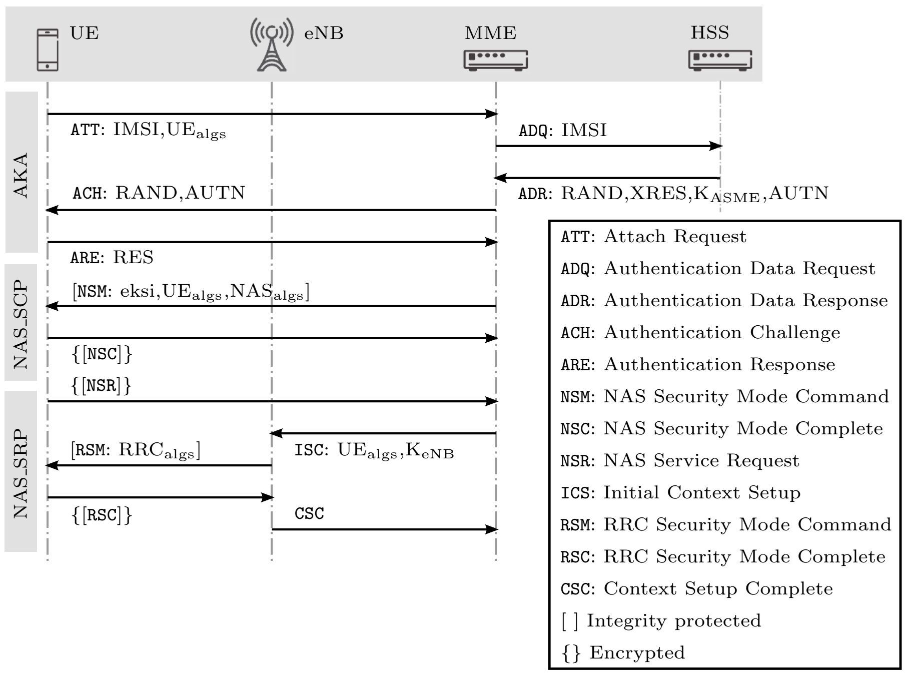

UE
eNB
MME
HSS
AKA
NAS_SCP
NAS_SRP
ATT: IMSI,UE_algs
ADQ: IMSI
ACH: RAND,AUTN
ADR: RAND,XRES,K_ASME,AUTN
ARE: RES
[NSM: eksi,UE_algs,NAS_algs]
{[NSC]}
{[NSR]}
[RSM: RRC_algs]
ISC: UE_algs,K_eNB
{[RSC]}
CSC
ATT: Attach Request
ADQ: Authentication Data Request
ADR: Authentication Data Response
ACH: Authentication Challenge
ARE: Authentication Response
NSM: NAS Security Mode Command
NSC: NAS Security Mode Complete
NSR: NAS Service Request
ICS: Initial Context Setup
RSM: RRC Security Mode Command
RSC: RRC Security Mode Complete
CSC: Context Setup Complete
[ ] Integrity protected
{} Encrypted

Fig. 2: AKA, NAS security control and service request procedures

3 ProVerif Overview

Before we present our work with the security procedures, we first describe ProVerif, the tool we use. We will be using AKA as a supporting example to show how protocols can be modeled and analyzed with it.

3.1 ProVerif

The tool takes formal models of the protocols together with a set of security properties as input. The input language is a typed variant of the applied pi calculus [4]. In this language, messages are modeled as terms. Relationships between cryptographic primitives are captured by rewrite rules or an equational theory. The complete specification can be found in the user manual [10].

ProVerif can prove reachability properties and correspondence assertions [7]. Reachability properties allow checking which information is in the possession of the attacker, i.e. secrecy. Correspondence properties are of the form "if some event is executed, then another event has previously been executed", and can be used for checking various types of authentication [20].

3.2 Input Language

Figure 3 shows an AKA model in the ProVerif language. In general, a protocol model can be divided in three parts: the declarations (lines 1-9), the process macros (10-31) and the main process (32). The declarations include the user types, the functions that describe the cryptographic primitives, and the security properties. The process macros consist of sub-process definitions. Each subprocess is a sequence of events. Finally, the main process is defined using those macros. In this particular example, it is defined as the parallel composition (denoted by |) of the unbounded replication (denoted by !) of three process macros representing a UE (line 10), an MME (18) and an HSS (24) node.

Declarations. Besides the built-in types: *channel*, *bitstring* and *bool*; additional user types can be declared as in line 2. Free names are introduced as in line 1 where two channels with names `pubch` and `secch` are declared. Free names are by default accessible to the attacker unless qualified by [**secret**]. In the example, the private channel is used for secure communication such as within a trusted domain or over an IPsec tunnel.

Constructors are functions used to build terms. They are declared by specifying their names, the types of the arguments and the return value (see lines 4-7). By default, functions are one-way; that is, the attacker cannot derive the arguments from the return value, unless qualified by [**data**] . Destructors (line 8) are special functions that are used to manipulate terms. Combined together, constructors and destructors are used to capture the relationship between cryptographic primitives. In the model of Fig. 3, the first three declared functions are

```
1  free pubch: channel. free secch: channel [private].
2  type key. type id. type msgheader.
3  const ATT, ADR, ADQ, ACH, ARE: msgheader.
4  fun kdf(bitstring, key): key.
5  fun autn(bitstring, key): bitstring.
6  fun res(bitstring, key): bitstring.
7  fun senc(bitstring, key): bitstring.
8  reduc forall x: bitstring, y: key; sdec(senc(x, y), y) = x.
9  table db(id, key).
10 let UE() =
11     new imsi: id; new k: key;(* key provisionning *)
12     insert db(imsi, k);(* key activation *)
13     out(pubch, (ATT, imsi));(* attach request *)
14     in(pubch, (=ACH, r: bitstring, a: bitstring));
15     if a = autn(r, k) then
16        let kasme: key = kdf(r, k) in
17        out(pubch, (ARE, res(r, k))).(* authentication response *)
18 let MME() =
19     in(pubch, (=ATT, imsi: id));
20     out(secch, (ADQ, imsi));(* authentication data request *)
21     in(secch, (=ADR, kasme: key, a: bitstring, xr: bitstring, n: bitstring));
22     out(pubch, (ACH, n, a));(* authentication request *)
23     in(pubch, (=ARE, =xr)).
24 let HSS() =
25     in(secch, (=ADQ, imsi: id));(* authentication data request *)
26     new n: bitstring;
27     get db(=imsi, k: key) in
28     let kasme: key = kdf(n, k) in
29     let a: bitstring = autn(n, k) in
30     let r: bitstring = res(n, k) in
31     out(secch, (ADR, kasme, a, r, n)).(* authentication data response *)
32 process ((!UE()) | (!MME()) | (!HSS()))
```

Fig. 3: AKA model

used to derive the authentication parameters in the HSS process (lines 28-30). The last two are used to model a shared key encryption and decryption scheme.

Constants (line 3) are 0-arity functions that together with types can be used to improve the clarity of the model and can help reducing the number of valid traces during the analysis. This is also common behavior of implementations, i.e., a protocol implementation typically reject messages of unexpected types. In addition, we use the constants to identify the different exchanged messages so that they can be easily mapped in the corresponding chart (Fig. 2).

The language provides support for tables for persistent storage. In line 9, a table modeling the subscriber database is declared. Lines 11-12 model the process of registering a new subscriber; and line 27 models the process of retrieving the pre-shared secret key of a subscriber (variable `k`) given its identity (`imsi`).

Process Macros. Messages are represented by terms. A term can be a name, a variable, a tuple of terms, a constructor or destructor application. In addition, the language has support for some common Boolean functions ($=, \&\&, ||, <>$) that use the infix notation. Pattern matching is used for term evaluation of message inputs. The pattern $x : t$ matches any term of type t and binds it to x. For a term M, the pattern $=M$ matches any term N such that $M = N$. A pattern tuple $(T_1, T_2, \ldots, T_n)$ matches any term tuple $(M_1, M_2, \ldots, M_n)$ where pattern matching is applyed recursively to each term M_i against pattern T_i. For example, the pattern $(= \texttt{ATT}, \texttt{imsi} : \texttt{id})$ in line 19 matches any term pair where the first one is the constant `ATT` and the second one is of type `id`.

Processes are defined as sequences of events. The name restriction event (line 26) creates a fresh name of a specific type and binds it inside the following events. The communication event $\textbf{out}(M, N); P$ (13), sends the term N on channel M and continue as the process P. The communication event $\textbf{in}(M, T); P$ (25), awaits a message matching pattern T on channel M and continues as P. The conditional **if** M **else** P **then** Q (15) continues as the process P if the term M evaluates to *true*, continues as the process Q if M evaluates to another value, or stops if M evaluation fails. The statement **let** $T = M$ **in** P **else** Q (28) tries to match the term M with pattern T, continues as the process P if there is a match, or continues as the process Q otherwise.

3.3 Security Properties

Security properties are declared with the keyword **query**. In our example of AKA, one of the goals is to establish the shared session key $\mathrm{K}_{\mathrm{ASME}}$ between the MME and the UE. In order to check this, we consider the following properties.

```
1 event ueReachable(). event mmeReachable(). event hssReachable().
2 query event(mmeReachable()); event(hssReachable()); event(ueReachable()).
3 free secret: bitstring [private].
4 query attacker(secret).
5 event ueRunning(key). event ueCommit(key). event mmeRunning(key). event
      mmeCommit(key).
6 query k: key; event(ueCommit(k)) ==> event(mmeRunning(k)).
7 query k: key; event(mmeCommit(k)) ==> event(ueRunning(k)).
8 query k: key; inj-event(ueCommit(k)) ==> inj-event(mmeRunning(k)).
9 query k: key; inj-event(mmeCommit(k)) ==> inj-event(ueRunning(k)).
```

The first two declarations are used for sanity checks. The "reachability" events of line 1 are intended to be executed each at the end of the corresponding process macro. Events are special extension to the process grammar that do neither affect the attacker knowledge, nor the execution of the processes. When analyzing the query of line 2, ProVerif attempts to falsify its claims by generating traces that reach those events. This is useful to check that the processes can be fully executed and that there are no blocking events for example due to a constantly failing pattern matching. The declarations in lines 3-4 are used to check secrecy of the established key. The **attacker** (line 4) is a built-in predicate that can be used to check which terms are compromised.

The last declarations are correspondence assertions used for checking mutual agreement between the UE and the MME on the key. The syntax to query a basic correspondence assertion uses the **event** keyword (lines 5-7). Correspondence assertions where a one-to-one mapping is required between events, use the **inj-event** keyword instead. In our case, we recall Lowe's definitions of weak and injective agreement [20] and use the special "running" and "commit" events declared in line 5 together with the correspondence assertions of lines 6-7 to check for agreement on the established key between the MME and UE processes. In general, a commit event is added in the end of each “responder” process, to which another “initiator” process is trying to authenticate. Then for each commit, a running event is added in the “initiator” process before the last send operation.

3.4 Analysis and Discussion

ProVerif is able to solve all the properties except one of the reachability queries of line 2 and the injective correspondence assertion query of line 9. The remaining queries are solved as expected. More precisely, the correspondence and secrecy ones are proved to hold and the reachability queries are falsified.

The unresolved reachability query can be solved by restricting the attacker model. This is can be done by a special feature of ProVerif for setting internal configuration parameters. One of those parameters representing the attacker capability can be set to either *passive* or *active* directly in the model. Setting the attacker to passive has the effect of reducing the number of traces (improving the chances for termination) but the analysis is no longer sound. This is not a problem for reachability because a trace that reaches the target event in the restricted model is also valid in the non restricted one. ProVerif is then able to solve the unanswered reachability claim. Another way for achieving the same effect consists in declaring all communication channels as private. Intuitively, the goal is to check whether the protocol can be run at all in a secure environment by honest agents.

For the unresolved correspondence assertion, while further experimenting with the model we observed the following. When strengthening the claim by including an additional `id` parameter (see below), executing the corresponding commit and running events with the additional argument set to the `imsi`, and setting the attacker model to passive, then ProVerif is able to find an attack trace even for the corresponding non-injective assertion.

```
query i: id, k: key; event(mmeCommit(i, k)) ==> event(ueRunning(i, k)).
```

The attack trace is due to the ProVerif approximation [8]. In the following we describe intuitively the effects of this approximation. First, a send operation on private channels is never blocking even in case of none matching operation. This does not correctly model communication in the real system as it might be reliable (for example transport over TCP). Second, the private channel is a shared broadcast one. In our case, this is problematic as what is really needed is a tunnel-like model of communication that simulates peer-to-peer (secure) channels. The model provided by ProVerif is too broad allowing even honest agents to read and use messages not destined to them. Therefore, false attack traces sometimes appear.

4 Session Management

We consider now the procedures that take place after the terminal and MME have established the K_{ASME} by AKA (see Fig. 2). Observe that the terminal has also informed the MME about which security capabilities it supports (the `ATT` message). The security capabilities include lists of encryption and integrity protection algorithms that the terminal supports. As a consequence, when analyzed separately, some initialization steps are needed in the protocol models in order to set up the required security context assumed to be established by AKA.

4.1 NAS Security

NAS security is enabled by a simple request-response procedure [1] (TS 24.301) that we refer to as the NAS Security Control Procedure NAS_SCP (see Fig. 2). The procedure is initiated by the MME sending a security mode command message (`NSM`) to the terminal. This message indicates the security algorithms chosen by the MME. The message includes a special identifier eksi indicating which K_{ASME} to use as the basis for the key derivation. For various reasons there may be more than one K_{ASME} known simultaneously to the terminal and network [1] (TS 33.401). The message also contains the list of security capabilities provided earlier by the terminal.

In response, the terminal verifies that the received security capabilities are consistent with what the terminal supports. If the verification fails, the terminal rejects the command thus preventing bidding-down attacks. If the verification succeeds, the terminal sends an encrypted and integrity protected completion message (`NSC`). All NAS messages are protected from replay attacks by inclusion of a sequence number (omitted in our models).

Model Description. Figure 4 shows a ProVerif model of the NAS_SCP protocol. Compared to the AKA model, the novelty in the declaration part consists in the use of predicates and clauses to model capability sets (lines 6-9). Predicates are declared like constructors and clauses are needed in order to define the meaning of the predicates. In our case, we declare a capability set constructor together with a constant representing the empty set in line 6. Then we use the predicate of line 7 to model the set membership test function which is defined below in the clauses of lines 8-9.

Furthermore, the functions used for the shared encryption scheme (lines 4-5) have been modified in order to take into account an additional parameter representing the algorithm to be used.

The main process executes some initialization events then expands and forks in parallel unbounded number of sessions of two process macros representing a UE (line 13) and an MME (20). The initialization steps consist in creating a capability set of two arbitrary algorithms (lines 31-32), disclosing it to the attacker (33), and finally creating a secret K_{ASME} key (34). The key is supposed to have been created earlier during an AKA run, while the capabilities should have been sent by the UE at startup in an attach request. Both parameters are used as input arguments to the process macros.

The use of predicates is illustrated in line 23. This particular event binds the variable `a : alg` to a value that satisfies the predicate `mem(a, uecaps)` in

```
free pubch: channel. free secch: channel [private].
type key.type alg. type caps. type id. type msgheader.
const NSM, NSC: msgheader. const NASINT, NASENC: bitstring.
fun psenc(alg, bitstring, key): bitstring.
reduc forall a: alg, x: bitstring, y: key; psdec(a, psenc(a, x, y), y) = x.
fun consset(alg, caps): caps [data]. const emptyset: caps.
pred mem(alg, caps).
clauses forall x: alg, y: caps; mem(x, consset(x, y));
        forall x: alg, y: caps, z: alg; mem(x, y) -> mem(x, consset(z, y)).
fun kdf(bitstring, key): key. fun ksi(key): id.
fun pmac(alg, bitstring, key): bitstring.
free secret: bitstring [private].
let UE(uecaps: caps, kasme: key) =
    in(pubch, (=NSM, =ksi(kasme), =uecaps, a: alg, nasmac: bitstring));
    let knasint: key = kdf(NASINT, kasme) in
    if mem(a, uecaps) && nasmac = pmac(a, (NSM, ksi(kasme), uecaps, a),
        knasint) then
       let knasenc: key = kdf(NASENC, kasme) in
       let msg: bitstring = (secret, pmac(a, (NSC, secret), knasint)) in
       out(pubch, (NSC, psenc(a, msg, knasenc))).(* security mode complete *)
let MME(uecaps: caps, kasme: key) =
    let eksi: id = ksi(kasme) in
    let knasint: key = kdf(NASINT, kasme) in (* integrity protection *)
    let a: alg suchthat mem(a, uecaps) in
    let nasmac: bitstring = pmac(a, (NSM, eksi, uecaps, a), knasint) in
    out(pubch, (NSM, eksi, uecaps, a, nasmac));(* security mode command *)
    in(pubch, (=NSC, payload: bitstring));
    let knasenc: key = kdf(NASENC, kasme) in (* confidentiality *)
    let (=secret, nasmacr: bitstring) = psdec(a, payload, knasenc) in
    if nasmacr = pmac(a, (NSC, secret), knasint) then 0.
process
        new a1: alg; new a2: alg;
        let uecaps = consset(a1, consset(a2, emptyset)) in
        out(pubch, uecaps);
        new kasme: key;
        ((!UE(uecaps, kasme)) | (!MME(uecaps, kasme)))
```

Fig. 4: NAS security establishment model

the rest of the process. Intuitively, this models the MME choosing an algorithm among the ones supported by the UE. During the analysis ProVerif considers all possible choices.

Analysis and Discussion. The goal of NAS_SCP is to establish the encryption and integrity keys, K_{NASenc} and K_{NASint}, that are to be used for the NAS protocol between the UE and the MME. In addition to the secrecy and sanity queries, we consider the following correspondence assertions in order to check agreement on the established keys and the chosen algorithm.

```
event ueRunning(alg, key, key). event ueCommit(alg, key, key).
event mmeRunning(alg, key, key). event mmeCommit(alg, key, key).
query a: alg, k1: key, k2: key;
   event(mmeCommit(a, k1, k2)) ==> event(ueRunning(a, k1, k2)).
query a: alg, k1: key, k2: key;
   inj-event(mmeCommit(a, k1, k2)) ==> inj-event(ueRunning(a, k1, k2)).
query a: alg, k1: key, k2: key;
   event(ueCommit(a, k1, k2)) ==> event(mmeRunning(a, k1, k2)).
query a: alg, k1: key, k2: key;
   inj-event(ueCommit(a, k1, k2)) ==> inj-event(mmeRunning(a, k1, k2)).
```

ProVerif is able to solve all the properties. The reachability queries are all falsified. The secrecy query and the basic correspondence assertions are proven to hold. However ProVerif reports attack traces on the injective assertions. This is not surprising as there is nothing in the protocol model that binds the runs to unique names (no creation of fresh names within the replicated processes). In fact the traces show that the attacker can falsify injection simply by duplicating and dropping messages to obtain a run between multiple parallel instances of MMEs against a single session of a UE and viceversa.

Modifying the model by moving the $\mathrm{K_{ASME}}$ key creation within the UE process and making the MME process read the key from a table leads to ProVerif proving that one of the direction holds. Intuitively, in the new model each run of the UE process is bound to a unique fresh key. Observe that this is a different system model since each replication of UE represents a new device rather than just a rerun of the same one. Furthermore, ProVerif is still able to report an attack trace for the other direction. This is expected as the modifications cannot prevent running in parallel multiple instances of MMEs that use the same $\mathrm{K_{ASME}}$ key and that can be matched against a single UE session. Since the $\mathrm{K_{ASME}}$ can only be present in one MME at a time, namely the one in which the UE is registered, it is not possible that two well behaved MMEs would be running NAS_SCP procedures simultaneously. Neither would a well behaved MME run two NAS_SCP procedures simultaneously by itself.

In fact well behaved agents would run the procedures sequentially. This we could not express in ProVerif. Even if we can express this sequential behavior, the injective agreement property will not hold. More precisely, assume the MME has sent two security mode command messages in separate sequential sessions, then it will not be able to distinguish to which session a reply belongs. This is because there is no information in the messages that tie them together, like for example a transaction identifier. It should be pointed out, that if an MME sends the same information repeatedly in different sessions, then regardless of which reply reaches the MME, the outcome of the whole procedure (algorithm negotiation and necessary key derivation) will be the same. From this perspective, injective agreement may not be necessary for this particular procedure.

4.2 RRC Security

Establishment of RRC security is achieved as follows: First, in order to send or receive data, the terminal needs to establish bearers to carry it. This is achieved by running a NAS Service Request Procedure with the network [1] (TS 24.301) and to which we refer by NAS_SRP (see Fig. 2). The terminal initiates the procedure by sending a service request (`NSR`) to the MME via the eNB. The radio channel between the UE and the eNB is not secured at this point, but this is not a problem since the NAS protocol provides its own security.

Upon reception of the request, the MME derives a $\mathrm{K_{eNB}}$ from the currently active $\mathrm{K_{ASME}}$ and the message sequence number associated with the NAS message. The latter parameter ensures that a fresh key is generated every time the procedure is run. This is necessary to prevent key stream re-use and replay attacks against the RRC protocol. The MME transfers the K_{eNB} together with the terminal's security capabilities to the eNB. The eNB sends a command message (`RSM`) to the terminal. This command includes the chosen algorithms and is integrity protected to prevent modification of the algorithm selection [1](TS 36.331). When the terminal receives the command, it derives the necessary keys and replies to the eNB with an encrypted and integrity protected completion message (`RSC`). From this point on, all RRC messages are integrity protected and encrypted, and all user plane traffic is encrypted.

Model and Analysis. A ProVerif model of the NAS_SRP is provided in Fig. 5. The declaration part has been removed as it is identical to that of the NAS_SCP model (see Fig. 4) except for some of the message headers and the constants used in the key derivation function, easily found in the model.

```
1  let UE(uecaps: caps, kasme: key) =
2      new nasulcount: bitstring; out(pubch, nasulcount);
3      out(secch, (NSR, nasulcount));(* initial service request *)
4      in(pubch, (=RSM, a: alg, rrcmac: bitstring));
5      let kenb: key = kdf(nasulcount, kasme) in
6      let krrcint: key = kdf(RRCINT, kenb) in
7      if mem(a, uecaps) && rrcmac = pmac(a, (RSM, a), krrcint) then
8         let krrcenc: key = kdf(RRCENC, kenb) in
9         out(pubch, (RSC, psenc(a, (secret, pmac(a, (RSC, secret), krrcint)),
              krrcenc))).
10 let MME(uecaps: caps, kasme: key) =
11     in(secch, (=NSR, nasulcount));
12     let kenb: key = kdf(nasulcount, kasme) in
13     out(secch, (ISC, uecaps, kenb));(* initial context setup *)
14     in(secch, =CSC).
15 let eNodeB() =
16     in(secch, (=ISC, uecaps: caps, kenb: key));
17     let krrcint: key = kdf(RRCINT, kenb) in (* integrity protection *)
18     let a: alg suchthat mem(a, uecaps) in
19     let rrcmac: bitstring = pmac(a, (RSM, a), krrcint) in
20     out(pubch, (RSM, a, rrcmac));(* security mode command *)
21     let krrcenc: key = kdf(RRCENC, kenb) in (* confidentiality *)
22     in(pubch, (=RSC, payload: bitstring));(* security mode complete *)
23     let (=secret, rrcmacr: bitstring) = psdec(a, payload, krrcenc) in
24     if rrcmacr = pmac(a, (RSC, secret), krrcint) then
25        out(secch, CSC).(* initial context setup response *)
26 process
27         ...
28         ((!UE(uecaps, kasme)) | (!eNodeB()) | (!MME(uecaps, kasme)))
```

Fig. 5: RRC security establishment model

The main process (line 26) executes some initialization steps then forks in parallel an unbounded number of sessions of three processes representing an UE (1), an MME (10) and an eNB (15) node. The initialization steps (omitted in the figure) are required to set up the parameters established earlier which are the user capabilities and the K_{ASME} key in a similar manner to how it is done in the model of Fig. 4. The additional in the model parameter denoted by `nasulcount` represents the NAS protocol message counter. This counter is used for deriving the K_{eNB} key (lines 5 and 12) that is to be provisioned to the eNB (13 and 16). It is incremented for each message exchange between the UE and MME. For example, this would be the effect of the send and matching receive operations of lines 3 and 11.

We model the counter by a fresh variable that we disclose (line 2) and make sure that it is synchronized by including it in the first NAS message (line 11). According to the specification [1](TS 33.401), when the counter, which is bounded, is about to wrap around then a new AKA run can be triggered in order to generate a new K_{ASME} key and thus preventing a K_{eNB} key reuse.

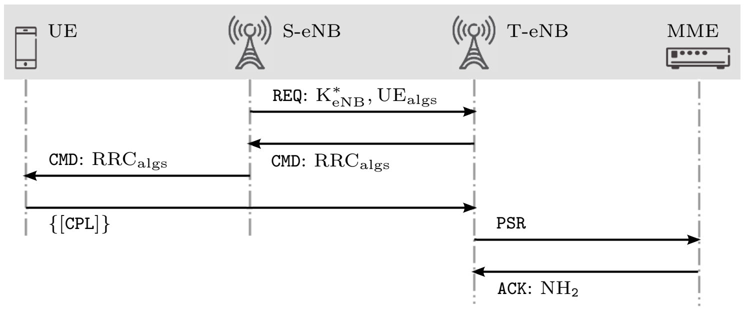
For the security properties, we consider the secrecy and sanity queries in a similar manner to the previous models. For the correspondence assertions, we focus on the agreement on the established K_{eNB} key and the chosen algorithm between the UE and the eNB. ProVerif solves all the queries as expected except one of the injective correspondence assertions (see Table 1).

5 Mobility Management

An eNB may detect that another eNB is better suited to serve an active terminal, for example because of better radio conditions. The source or serving eNB (denoted by S-eNB) hands over the terminal to the target eNB (denoted by T-eNB). There are two compound procedures to perform a handover. The first is a core network assisted handover that is called S1 handover (HO_S1). The second is a handover without core network assistance called X2 handover (HO_X2). The names come from the primary network interfaces used during the execution of the handovers.

5.1 X2 Handover

Handovers can be performed after the terminal has completed all necessary procedures so that RRC and NAS security has been activated. The X2 handover (Fig. 6) is initiated by the S-eNB calculating a K^*_{eNB} key from the currently active K_{eNB} and sending it together with the terminal security capabilities to the T-eNB in a handover request message (`REQ`). The T-eNB replies with the required configuration information for the terminal connection. This information includes the chosen algorithms that the T-eNB and the terminal shall use (`CMD`). The S-eNB then forwards the reply to the terminal, which confirms the handover with a completion message (`CPL`). In the last step, the T-eNB retrieves a new key called the Next Hop key (NH) from the MME. The NH which is derived from the K_{ASME} is to be used as a basis for the K^*_{eNB} calculation in the next handover event [1] (TS 33.401).

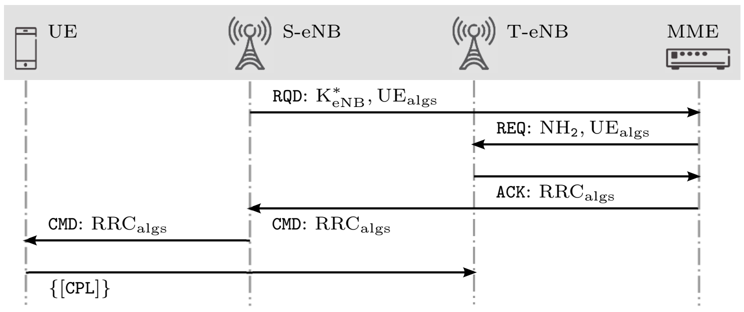


REQ: Handover Request	ACK: Path Switch Acknowledge
CMD: Handover Command	[ ] Integrity protected
CPL: Handover Complete	{} Encrypted
PSR: Path Switch Request	

Fig. 6: X2 handover

5.2 S1 Handover

In an S1 handover (Fig. 7), the S-eNB and target T-eNB are not directly connected. Instead, the S-eNB sends a handover required message (RQD) to the MME containing the security capabilities of the terminal. The MME then derives the NH key and sends it to the target node, together with the UE capabilities. The T-eNB uses the NH key to derive the K_{eNB} for communication with the terminal, and sends a handover command (CMD) containing the chosen algorithms to the source node. Finally, the S-eNB forwards the message to the terminal which replies to the T-eNB by a handover completed message (CPL).



RQD: Handover Required	PSR: Path Switch Request
REQ: Handover Request	ACK: Path Switch Acknowledge
CMD: Handover Command	[ ] Integrity protected
CPL: Handover Complete	{} Encrypted

Fig. 7: S1 handover

5.3 Formal Models and Analysis

Both handover procedures involve four agents: a UE, a source S-eNB, a target T-eNB and an MME. The procedures are very similar but provide slightly different security guarantees. The ProVerif models of the protocols are provided in Fig. 8 and Fig. 9. The declaration parts have been omitted as they are very similar to previous models except for some types and constants.

```
1  let UE(uecaps: caps, kenb: key, cellid: bitstring) =
2      in(secch, (=CMD, a: alg));
3      if mem(a, uecaps) then
4         let kenbstar: key =  kdf(cellid, kenb) in
5         out(pubch, (CPL, senc((a, mac((CPL, a), kenbstar)), kenbstar))).
6  let MME(nh_2: key) =
7      in(secch, =PSR);
8      out(secch, (ACK, nh_2)).
9  let SeNodeB(uecaps: caps, kenb: key, cellid: bitstring) =
10     let kenbstar: key =  kdf(cellid, kenb) in
11     out(secch, (REQ, kenbstar, uecaps)).
12 let TeNodeB() =
13     in(secch, (=REQ, kenbstar: key, uecaps: caps));
14     let a: alg suchthat mem(a, uecaps) in
15     out(secch, (CMD, a));
16     in(pubch, (=CPL, msg: bitstring));
17     let (=a, rrcmac: bitstring) = sdec(msg, kenbstar) in
18     if rrcmac = mac((CPL, a), kenbstar) then
19        out(secch, PSR);
20        in(secch, (=ACK, nh_2: key)).
21 process
22         new a1: alg; new a2: alg;
23         let uecaps = consset(a1, consset(a2, emptyset)) in
24         out(pubch, uecaps);
25         new kasme: key; new nasulcount: bitstring; out(pubch, nasulcount);
26         let kenb: key = kdf(nasulcount, kasme) in
27         let nh_1: key = kdf(tobitstring(kenb), kasme) in
28         let nh_2: key = kdf(tobitstring(nh_1), kasme) in
29         new cellid: bitstring; out(pubch, cellid);
30         ( (!UE(uecaps, kenb, cellid)) | (!SeNodeB(uecaps, kenb, cellid)) |
31           (!TeNodeB()) | (!MME(nh_2)) )
```

Fig. 8: X2 handover model

In the model of X2 handover (Fig. 8), the main process performs the required initialization steps and forks unbounded sessions of the processes defined in lines 1, 6, 9 and 12 representing respectively a UE, an MME, a S-eNB and a T-eNB. The initialization steps include defining the UE capabilities (lines 22-24), the K_{ASME} key shared between the UE and the MME (25), and the K_{eNB} shared between the UE and the S-eNB (26). The remaining additional steps are needed to establish some parameters required for deriving the current and future handover keys such as the next hop key (27-28) and the cell identifier for the T-eNB (29).

The model of S1 handover (Fig. 9) requires almost the same initialization steps (omitted here) except that the additional cell identifier `cellid` parameter of HO_X2 is no longer needed. The main difference between the models is in the

```
1  let UE(uecaps: caps, nh_2: key) =
2      in(secch, (=CMD, a: alg));
3      if mem(a, uecaps) then
4          out(pubch, (CPL, senc((a, mac((CPL, a), nh_2)), nh_2))).
5  let MME(uecaps: caps, nh_2: key) =
6      in(secch, =RQD);
7      out(secch, (ACK, nh_2, uecaps)).
8  let SeNodeB(kenb: key) =
9      out(secch, RQD).
10 let TeNodeB() =
11     in(secch, (=ACK, nh_2: key, uecaps: caps));
12     let a: alg suchthat mem(a, uecaps) in
13     out(secch, (CMD, a));
14     in(pubch, (=CPL, msg: bitstring));
15     let (=a, rrcmac: bitstring) = sdec(msg, nh_2) in
16     if rrcmac = mac((CPL, a), nh_2) then 0.
17 process
18           ...
19         ( (!UE(uecaps, nh_2)) | (!SeNodeB(kenb)) |
20           (!TeNodeB()) | (!MME(uecaps, nh_2)) )
```

Fig. 9: S1 handover model

MME role. In HO_S1, it is the MME that computes the key to be used in the T-eNB which is NH_2. However, in HO_X2 it is the S-eNB that computes the target's key which is the K^*_{eNB}. In addition, the MME provides its key (NH_2) in the last steps of the protocols (lines 7-8 and 20-21) for use in the next handover.

In both handover models, we consider the same type of sanity queries in order to check correctness. Recall that such queries are for special reachability events executed at each end of the process macros. ProVerif is not able to prove all of them. Nevertheless, the attack traces of the queries that ProVerif was able to falsify, show that the reachability events for the unresolved queries are executed as well. For the correspondence assertions, the aim is to prove agreement on the received handover key (K^*_{eNB} or NH_2) and the chosen algorithm between the UE and the T-eNB. ProVerif is able to prove all the assertions for the HO_X2 model except one (see Table 1). For the HO_S1 model, ProVerif is unable to prove the injective assertions, but proves the basic ones.

6 Conclusion

We have presented our work on security protocols in LTE. We have used ProVerif to formalize and verify the protocols. Our analysis has shown that all the secrecy and weak agreement properties hold which was expected. However, we had difficulties proving stronger agreement properties. All our results are summarized in Table 1. To the best of our knowledge, the security command procedures and handover procedures have not been previously analyzed in this manner. During the modeling process, our aim was to remain as faithful as possible to the 3GPP specifications [1] of the protocols and their trust model. One important aspect that is lacking in ProVerif is the support for modeling local state information. As we explain later, support for that would relieve protocol designers from the

Table 1: Analysis Results

Property	AKA	NAS_SCP	NAS_SRP	HO_X2	HO_S1
secrecy	true	true	true	true	true
weak-agree UE $\Longrightarrow \ldots$	true	true	true	true	true
weak-agree $\ldots \Longrightarrow$ UE	true	true	true	true	true
inj-agree UE $\Longrightarrow \ldots$	true	false	true	true	unresolved
inj-agree $\ldots \Longrightarrow$ UE	unresolved	false	unresolved	unresolved	unresolved

common and tedious tasks of ensuring uniqueness of inputs to key derivations functions.

ProVerif verifies the models in the order of seconds on a regular laptop and the runtime is hence adequate for practical use. The concepts of the ProVerif input language matches the concepts used in the 3GPP specifications. Therefore, constructing the models does not take any significant time assuming familiarity with ProVerif and the system under study.

Future Work. We have been experimenting with other formal verification tools. For this first work, we thought that ProVerif offers a good compromise for ease of use and expressiveness of the input language. Nevertheless, we are planning to conduct a more thorough evaluation and comparison of similar tools. Furthermore, we believe that in an industrial context, such as for proposals to standardization processes, several tools should be used in combination in order to overcome their shortcomings. This does not necessarily mean that a process as defined by a standardization organization must formally mandate the use of such tools. In fact, as we explain below, there are few aspects in the protocols that we could not handle properly with ProVerif.

In general, freshness is achieved by guaranteeing uniqueness of the derived keys in each session. This can be implemented by using nonces, like in AKA for the derivation of the K_{ASME} key. However, all the derivation of lower level keys rely on other protocol parameters such as counters, cell identifiers, etc. Counters are part of the protocol state that is continuously being updated. We believe that any issue in the considered protocols would be most likely related to this aspect, especially when different protocols are interleaved and used arbitrarily in other more complex compound procedures. State-based formal verification tools can be better suited to model check such features. However in order to improve efficiency of such (usually exhaustive) state search, one can assume a weaker attacker model. As a future work, we are investigating to which extent the attacker model can be simplified and still be able to find attacks.

Other continuations of this work include performing similar analysis of the protocols for inter-operability between LTE and other types of radio access networks (GSM and WCDMA), and updating our models to handle different scenarios. For example, in the AKA model of Fig. 3, each run of the UE process represents a new device because in every run, a new fresh pair of $\mathrm{IMSI}, \mathrm{K}$ is created and used. The model can be changed by adding another similar UE process which instead, gets the pair from the table being filled by the original process. This is how one can model arbitrary reruns of AKA by the same UE. In addition, the mobility models can be enhanced to include multi hop handovers. This can be used to verify further security properties, e.g. two hop forward security.

References

1. 3GPP The Mobile Broadband Standard. http://www.3gpp.org/specifications/.
2. *10th Computer Security Foundations Workshop (CSFW '97), June 10-12, 1997, Rockport, Massachusetts, USA*. IEEE Computer Society, 1997.
3. Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just Fast Keying in the Pi Calculus. In David A. Schmidt, editor, *ESOP*, volume 2986 of *Lecture Notes in Computer Science*, pages 340–354. Springer, 2004.
4. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In Chris Hankin and Dave Schmidt, editors, *POPL*, pages 104–115. ACM, 2001.
5. Myrto Arapinis, Loretta Ilaria Mancini, Eike Ritter, Mark Ryan, Nico Golde, Kevin Redon, and Ravishankar Borgaonkar. New privacy issues in mobile telephony: fix and verification. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 205–216. ACM, 2012.
6. Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.
7. Bruno Blanchet. Automatic verification of correspondences for security protocols.
8. Bruno Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *CSFW*, pages 82–96. IEEE Computer Society, 2001.
9. Bruno Blanchet and Avik Chaudhuri. Automated Formal Analysis of a Protocol for Secure File Sharing on Untrusted Storage. In *IEEE Symposium on Security and Privacy*, pages 417–431. IEEE Computer Society, 2008.
10. Bruno Blanchet, Ben Smyth, and Vincent Cheval. *ProVerif 1.88: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*.
11. Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Automatic validation of protocol narration.
12. Mohamed Salah Bouassida, Najah Chridi, Isabelle Chrisment, Olivier Festor, and Laurent Vigneron. Automated verification of a key management architecture for hierarchical group protocols. *Annales des Télécommunications*, 62(11-12):1365–1387, 2007.
13. Cas J. F. Cremers. The Scyther Tool: Verification, Falsification, and Analysis of Security Protocols. In Aarti Gupta and Sharad Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008.
14. Cas J. F. Cremers. Session-state Reveal Is Stronger Than Ephemeral Key Reveal: Attacking the NAXOS Authenticated Key Exchange Protocol. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 20–33, 2009.

15. Cas J. F. Cremers. Key Exchange in IPsec Revisited: Formal Analysis of IKEv1 and IKEv2. In Vijay Atluri and Claudia Díaz, editors, *ESORICS*, volume 6879 of *Lecture Notes in Computer Science*, pages 315–334. Springer, 2011.
16. Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.
17. Jiexiang Fang and Rui Jiang. An analysis and improvement of 3GPP SAE AKA protocol based on strand space model. In *Network Infrastructure and Digital Content, 2010 2nd IEEE International Conference on*, pages 789–793, Sept 2010.
18. Gerard J. Holzmann. The Model Checker SPIN. *IEEE Trans. Software Eng.*, 23(5):279–295, 1997.
19. Markus Jakobsson and Susanne Wetzel. Security Weaknesses in Bluetooth. In *Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, CT-RSA 2001, pages 176–191, London, UK, UK, 2001. Springer-Verlag.
20. Gavin Lowe. A Hierarchy of Authentication Specification. In *CSFW* [2], pages 31–44.
21. Gavin Lowe. Casper: A Compiler for the Analysis of Security Protocols. In *CSFW* [2], pages 18–30.
22. Catherine Meadows. The NRL Protocol Analyzer: An Overview. *J. Log. Program.*, 26(2):113–131, 1996.
23. Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.
24. Naïm Qachri, Olivier Markowitch, and Jean-Michel Dricot. A formally Verififed Protocol for Secure Vertical Handovers in 4G Heterogeneous Networks. *International Journal of Security and Its Applications*, 7(6), 2013.
25. B. Schmidt, R. Sasse, and D. Basin. Automated Verification of Group Key Agreement Protocols. In *IEEE Symposium on Security and Privacy*, page to appear, 2014.
26. A.M. Taha, A.T. Abdel-Hamid, and S. Tahar. Formal analysis of the handover schemes in mobile WiMAX networks. In *Wireless and Optical Communications Networks, 2009. WOCN '09. IFIP International Conference on*, pages 1–5, April 2009.
27. A.M. Taha, A.T. Abdel-Hamid, and S. Tahar. Formal Verification of IEEE 802.16 Security Sublayer Using Scyther Tool. In *Network and Service Security, 2009. N2S '09. International Conference on*, pages 1–5, June 2009.
28. Chunyu Tang, David A. Naumann, and Susanne Wetzel. Symbolic Analysis for Security of Roaming Protocols in Mobile Networks - [Extended Abstract]. In Muttukrishnan Rajarajan, Fred Piper, Haining Wang, and George Kesidis, editors, *SecureComm*, volume 96 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 480–490. Springer, 2011.
29. Joe-Kai Tsay and Stig Fr. Mjølsnes. A Vulnerability in the UMTS and LTE Authentication and Key Agreement Protocols. In Igor V. Kotenko and Victor A. Skormin, editors, *MMM-ACNS*, volume 7531 of *Lecture Notes in Computer Science*, pages 65–76. Springer, 2012.
30. Muxiang Zhang and Yuguang Fang. Security analysis and enhancements of 3gpp authentication and key agreement protocol.