Security Analysis of the Terrestrial Trunked Radio (TETRA) Authentication Protocol

Shuwen Duan, Stig Frode Mjølsnes, Joe-Kai Tsay

Department of Telematics, NTNU, Trondheim
shuwend@stud.ntnu.no,{sfm,joe.k.tsay}@item.ntnu.no

Abstract

We present a security analysis of the widely-deployed Terrestrial Trunked Radio (TETRA) authentication protocol, where we find that the TETRA authentication protocol suffers from flaws that allow an attacker to defeat basic authentication properties that are normally required in secure operations. In order to launch an attack, an adversary only needs to control the radio link communication (*e.g.*, by impersonating as a base station). This attack can be used to reduce the users' availability of the network access, which may cause serious consequences in an emergency scenario: a targeted mobile station may falsely show that it is connected to the network while, in fact, the mobile station is unable to receive network communications.

Based on this analysis, we propose a strengthened authentication protocol for the TETRA system, and formally verify security properties for our protocol proposal using the automated tool Scyther.

1 Introduction

Terrestrial Trunked Radio (TETRA) is an ETSI (European Telecommunications Standards Institute) standard, first issued in 1995, for a mobile communication system designed to be used by law enforcement, emergency and rescue service organizations, in public transportation organizations, and as a general national safety communication network. TETRA systems are widely deployed and in operation in more than 100 nations. Norway is in the roll-out phase of the TETRA-based emergency network "Nødnett", which is managed by the governmental Directorate for Emergency Communication (DNK) and developed and operated by Motorola Solutions Inc. [6].

While the GSM mobile system, also an ETSI standard first issued in 1987, has later evolved into the Universal Mobile Telecommunications System (UMTS) and now into the Long-Term Evolution (LTE) system specifications, there does not appear to exist any follow-up of TETRA despite the age of its specifications, and

This paper was presented at the NISK-2013 conference.

as such TETRA can be regarded as the state-of-the-art in interoperability standard for public safety use.

A trunked radio system allows many groups of users to time-share a few radio channels by the use of statistical multiplexing techniques. The main service of TETRA is voice communications. Some special features of this mobile communication system are: very short call setup time (<0.3s), push-to-talk group calling mode, and direct terminal-to-terminal radio transmission. TETRA provides cryptographic authentication protocols, radio link encryption, and end-to-end terminal encryption.

For emergency and safety communication networks it is imperative that they stay operational even in extraordinary situations. Therefore, it is first and foremost important that TETRA offers *availability* to authorized users, so that emergency service/safety personnel can give and receive messages from dispatchers and coordinators; for TETRA this is arguably more important than *confidentiality*. In general, a protocol that fails to offer some security requirement may not necessarily entail real-world consequences. For instance, an attacker who is able to impersonate as an authorized user but who is not able to learn the secret session key will not be able to obtain any read access to data, which is only available encrypted under this session key. However, a security weakness of an authentication protocol, even if it does not result in leakage of session keys (and therefore does not violate confidentiality), could be used to disrupt availability of service. We present here three technical attacks that defeat authentication in TETRA. In one attack an adversary can target mobile stations and lead them to falsely believe that they are connected to the network.

Related Work Roelofsen made a rather high level description of the TETRA security system in Ref [12], but his paper does not present or give any reference to security analysis performed. We are aware of only one published security analysis of the TETRA system, which is the recently published paper of Park et al. [11]. They assume a very strong attacker who is able to clone the secret key of the mobile station and knows the user identity (phone number). However, they fail to discover the weaknesses that we present in this work, nor do they analyze TETRA authentication formally. This paper is partly based on the results of Shuwen Duan's master thesis work submitted at Department of Telematics, NTNU in June 2013 [8].

Structure of this Work Section 2 gives a brief overview of TETRA's network architecture sufficient for our analysis, and then describes in detail the TETRA authentication protocol. Section 3 discusses the security weaknesses of the TETRA authentication protocol we discovered, as well as their consequences. Section 3.3 presents our strengthened protocol construction proposal. Section 4 presents the formal verification of the strengthened protocol using the tool Scyther.

2 Terrestrial Trunked Radio (TETRA)

2.1 Overview of the Network Architecture

For the analysis purposes in this paper, we model the TETRA system to comprise three main subsystems: Mobile Stations (*MS*) at the users, Base Stations (*BS*) as the radio access network, and the Switching and Management Infrastructure

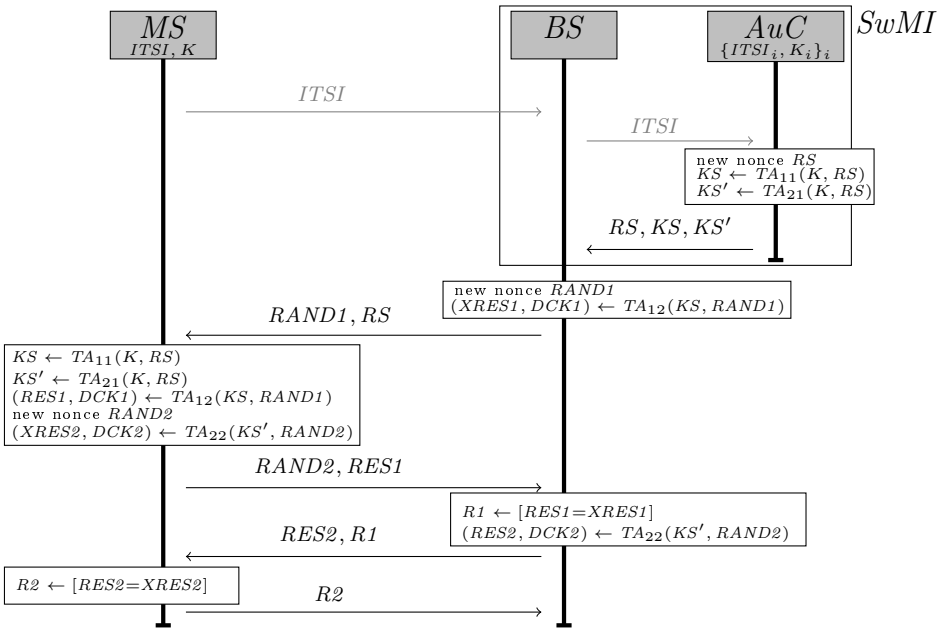


Figure 1: The TETRA Authentication Protocol; mutual authentication initiated by the network *SwMI*. *MS* and *BS* agree on session key $TB4(DCK1, DCK2)$.

(*SwMI*) as the core network, which includes the Authentication Centre (*AuC*). An *MS* comprises both the physical terminal equipment, identified with a TETRA Equipment Identity (*TEI*) unique to each device, and a Subscriber Identity Module (*SIM*). The SIM keeps a unique TETRA user/subscriber identity and an associated symmetric cryptographic key in tamper-resistant memory, and is able to execute the cryptographic primitives applied in the authentication protocol.

2.2 TETRA Authentication Protocol

The TETRA authentication protocol is specified in [9]. TETRA supports both unilateral and mutual authentications between terminals and the mobile network, which can be initiated by either one. In this paper we only discuss the complete mutual authentication protocol, as it is the strongest authentication property covered by the specifications. Furthermore, we concentrate on the procedure initiated by the infrastructure (Fig. 1). However, we can easily adapt attacks presented below to break authentication of the weaker unilateral authentication protocols as well as the mutual authentication initiated by the *MS* (Fig. 2). The parties involved in the authentication process are *MS*, *AuC* and *BS*. We assume the communication channel between *AuC* and *BS* to be secure and, therefore, treat the *AuC* and *BS* together as a single party *SwMI*.

Figure 1 shows the message diagram of the TETRA mutual authentication protocol initiated by the *SwMI*. After the *AuC* receives the Individual TETRA Subscriber Identity (*ITSI*) of the *MS*, it chooses the user's authentication key K matching its *ITSI*. We note that the delivery of the *ITSI* is not specified as part of the authentication protocol in [9], and, therefore, we do not include it in our analysis below (and depict it in gray in Fig 1). Given the *ITSI*, the *AuC* then generates a random seed *RS* and uses it with K to generate a pair of session keys *KS* and KS' through algorithms *TA*11 and *TA*21, respectively. *KS*, KS' and *RS* are forwarded to the *BS*, which generates a random number *RAND1* and computes the *expected*

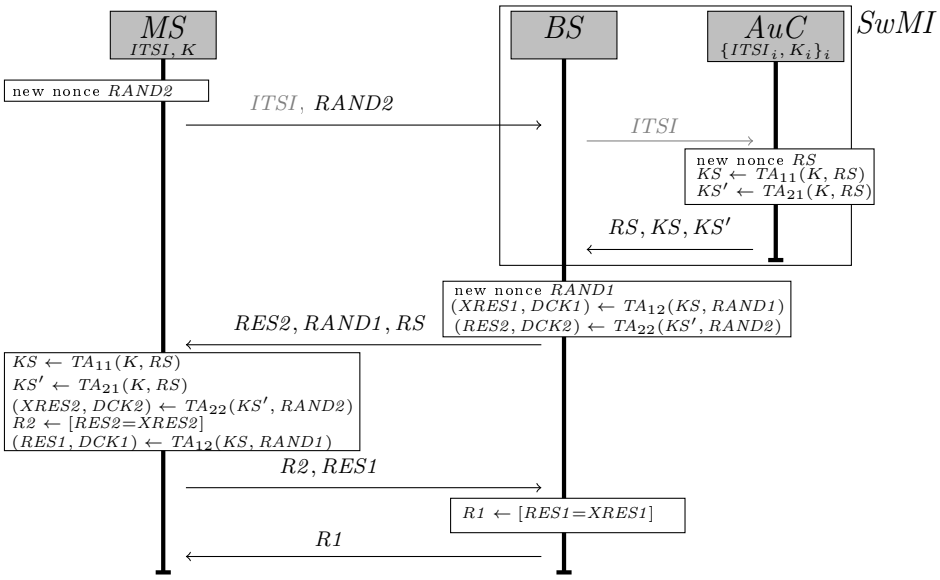


Figure 2: The TETRA Authentication Protocol; mutual authentication initiated by mobile station *MS*. *MS* and *BS* agree on session key $TB4(DCK1, DCK2)$.

response XRES1 and the derived cipher key *DCK1* from *KS* and *RAND1* under the algorithm *TA*12. Then *BS* sends *RAND1*, *RS* to *MS*. *MS* then also generates *KS* and KS' and computes the *response RES1* and *DCK1*, of which *RES1* is sent back to *BS*. The user also generates and sends a random number *RAND2* to *BS*. *BS* compares *RES1* with *XRES1*, and if the two values are equal it sets the value *R1* to `true`. Furthermore, *BS* computes the *response RES2* and the derived cipher *DCK2* from KS' and *RAND2* under the algorithm *TA*22. It then returns *RES2* and the authentication result *R1* ← `true`. Then *MS* compares *RES2* with *expected response XRES2*, and if they are equal the *MS* finally returns the value *R2* ← `true`. In case both parties complete their runs successfully, they both use *DCK1* and *DCK2* to compute under algorithm *TB*4 the shared session key *DCK*.

The algorithms *TA*11, *TA*12, *TA*21, *TA*22, and *TB*4 are not specified, rather it is up to the operator to choose a secure implementation. The specifications [9] rather vaguely demand that these functions satisfy some one-wayness properties[1].

3 Weaknesses in TETRA Authentication

3.1 Attacker Model and Security Properties

3.1.1 Attacker Model

We analyze the TETRA authentication protocol with respect to the so called *Dolev-Yao* security model [7]. In this model, the attacker is in full control of the network. He can intercept any sent message and decompose it. The attacker, who can act nondeterministically, may create and inject messages from information he learnt from intercepted messages and from an unbounded number of fresh constants. He can initiate an unbounded number of protocol sessions to be executed concurrently. However, cryptographic primitives are assumed to be perfectly secure: For instance,

[1]For instance, [9] states about *TA*11 that "the algorithm should be designed such that it is difficult to infer any information about Input 1 from the knowledge of Input 2 and the Output (even if the details of the algorithm are known)."

an attacker can only learn anything about the underlying plaintext of a given ciphertext if he knows the corresponding decryption key. There is no partial information of basic message components, *e.g.*, keys or nonces, available.

3.1.2 Security Properties

In our security analysis we are interested in investigating whether the following basic authentication and secrecy properties of two-party protocols are satisfied[2]:

Weak Agreement A protocol guarantees *weak agreement* to party A, if for all protocol runs that A completes with intended communication partner B, party B indeed executed a run with intended communication partner A.

Non-injective Agreement (Ni-Agree) A protocol guarantees *ni-agree* for party A, if for all protocol runs that A completes with intended communication partner B, there is a protocol run by B with intended partner A, so that the content of the received messages in these runs is equal to the content of the messages sent by the corresponding run of the intended partner.

Non-injective Synchronisation (Ni-Synch) A protocol guarantees *ni-synch* for party A, if for all protocol runs that A completes with intended communication partner B, there is a protocol run by B with intended partner A, so that the content of the received messages in these runs is equal to the content of the messages sent by the corresponding run of the intended communication partner, and, additionally, the corresponding send and receive actions are performed in the expected order.

Secrecy of Data A protocol guarantees *secrecy* for some data that party A holds at the end of a completed run, *e.g.*, the session key agreed upon in that run, if an attacker does not gain any information on that data.

3.2 Breaking TETRA Authentication

3.2.1 Attacks against the Protocol

There is a rather obvious weakness in the TETRA authentication protocol depicted in Figure 1: The *acknowledgement bits* R_1 and R_2 are not integrity protected! Therefore, an attacker controlling air-interface communication may flip these bits at will. A more subtle weakness is, in addition, that the response *RES1* sent by the *MS* to the *SwMI* is not bound to the random number *RAND2*. These weaknesses are exploited in the following attacks:

Attack 1: The attacker observes a TETRA authentication protocol execution and only intercepts and changes the last message *R2* sent by the involved *MS*, *i.e.*, the attacker changes *R2* from `true` to `false`. As a consequence, the *MS* believes that it correctly authenticated itself to the network and shares with it a session key, while the network believes that it completed a run, in which authentication was not successful. This violates Ni-Agree and Ni-Synch in the role of *MS*. Note that this is different from *cutting-the-last-message* attacks, where the attacker simply drops the last protocol message (– this is always possible) and its intended receiver never completes that run.

[2]These properties are formally defined in [10, 5] and are automatically verifiable with Scyther.

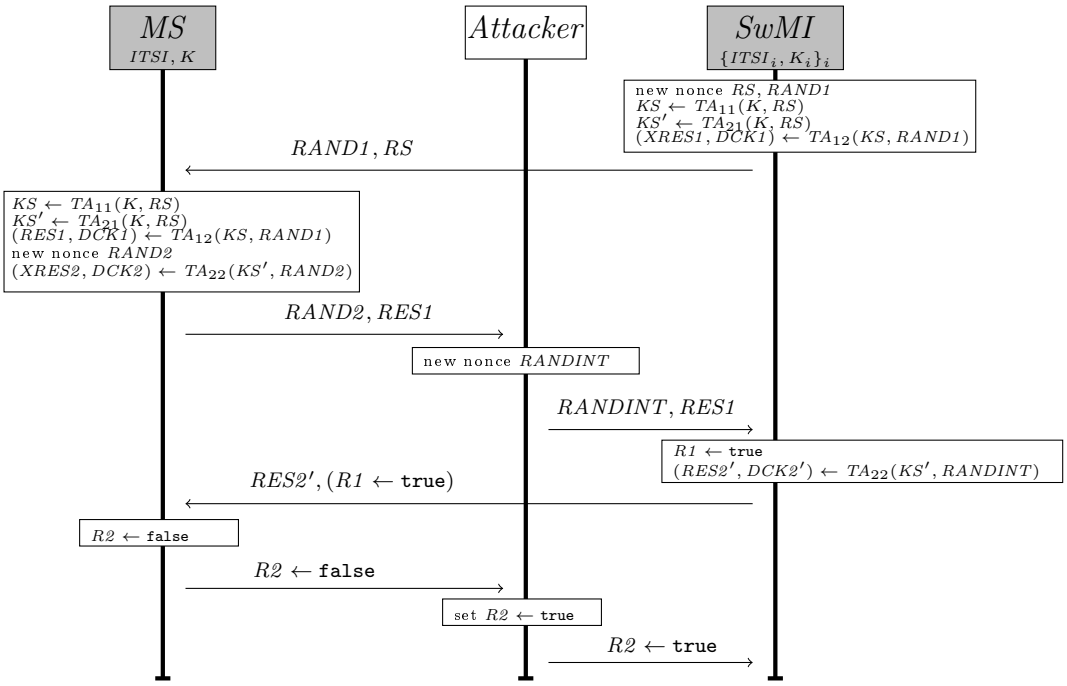


Figure 3: Attack 2 against TETRA mutual authentication initiated by *SwMI*.

Attack 2: In this attack, depicted in Figure 3, the attacker intercepts the message sent from the *MS* to the SwMI consisting of *RES1* and *RAND2*. The attacker selects *RES1* and replaces the value of *RAND2* with a newly generated nonce *RANDINT*. The *SwMI* believes the modified message was sent from the *MS* and returns a message that contains *RES2* computed from an incorrect random value and $R1 \leftarrow \texttt{true}$. Therefore, the comparison test of *RES2* and *XRES2* fails on the *MS* side and *R2* is set to `false`[3]. However, the attacker intercepts the last message and sends instead $R2 \leftarrow \texttt{true}$ to the *SwMI*. In the perspective of role *SwMI*, security properties Ni-Synch and Ni-Agree do not hold.

Attack 3: [4] In this attack, depicted in Figure 4, the *MS* starts two concurrent sessions both with the intended communication partner *SwMI*, while *SwMI* runs a single session with intended partner *MS*. The attacker eventually makes *SwMI* and *MS* falsely believe that they have agreed on the same session key. First, when *SwMI* sends $RAND1, RS$ to *MS*'s first session, the attacker selects *RS*, generates a fresh nonce *RANDINT* and sends $RANDINT, RS$ to *MS*'s second session. *MS*'s first session computes the correct response *RES1* and returns $RES1, RAND2$, with a fresh nonce *RAND2*, while *MS*'s second session computes the incorrect response $RES1'$ and returns $RES1', RAND2'$, with a fresh nonce $RAND2'$. Now the attacker chooses *RES1* from the first session and $RAND2'$ from the second session, and sends to *SwMI* the message $RES1, RAND2'$. As *RES1* equals *XRES1*, the infrastructure *SwMI* sets $R1 \leftarrow \texttt{true}$, computes from $RAND2'$ the response $RES2'$ and sends out $RES2', R1$. This message is then directed by the attacker to *MS*'s second session; while the first session is abandoned by the attacker. As $RES2'$ was computed from the random nonce $RAND2'$ generated in the second session of *MS*, it is indeed

[3] Alternatively, the attacker may directly change *RES2*, without replacing *RAND2*

[4] Unlike the first two attacks, we discovered this attack with the help of Scyther.

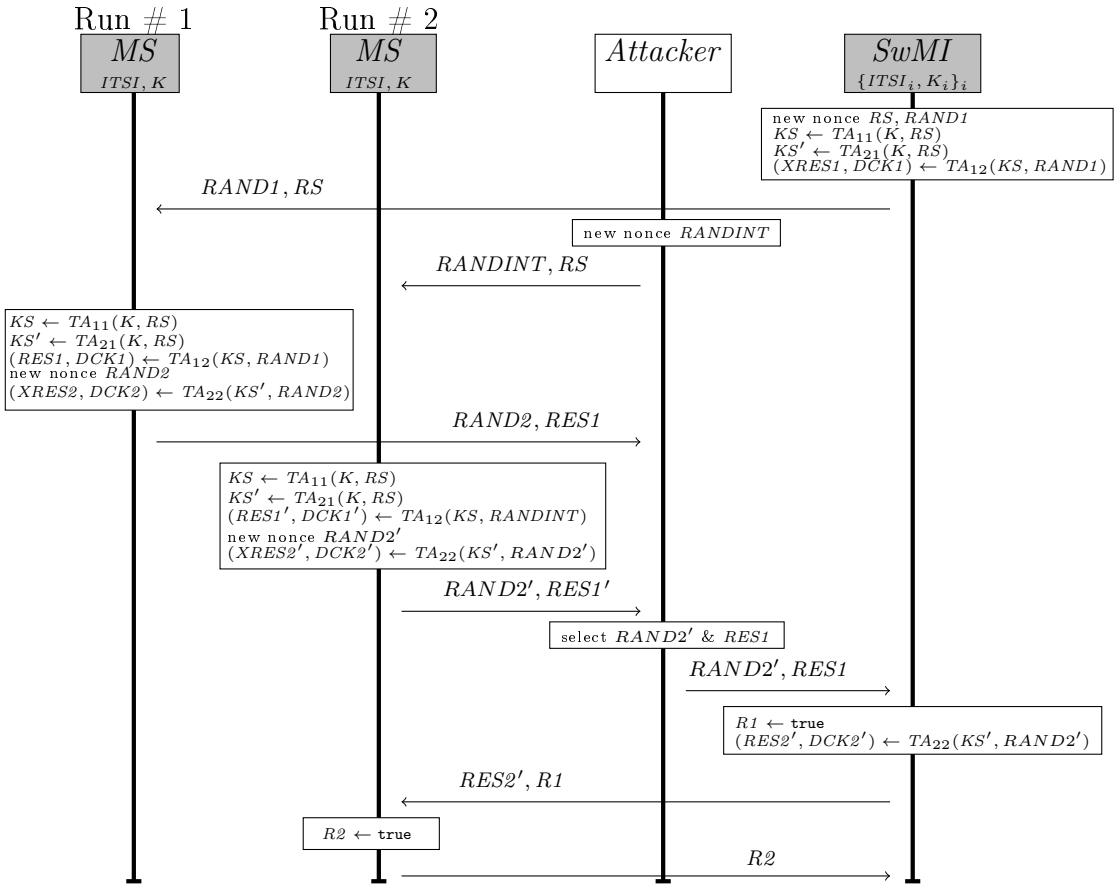


Figure 4: Attack 3 against TETRA mutual authentication initiated by *SwMI*.

equal to the expected response in the second session. Therefore, *MS*'s second session sends out $R2 \leftarrow \texttt{true}$. At this point, *MS* believes in its second session that it successfully completed a protocol run with *SwMI*, and *SwMI* believes that it successfully completed a protocol run with *MS*. However, not only are the received messages not equal to the sent messages of those sessions, the session keys computed are also not equal. Recall that the session key is computed from *DCK1* and *DCK2*. But the value of *DCK1*, is computed in *SwMI*'s session from *RAND1*, while in *MS*'s second session it is computed from *RAND1'*. This violates Ni-Agree and Ni-Synch in roles *MS* and *SwMI*.

3.2.2 Real-world Threats

In order to decide whether the three attacks presented in Section 3.2.1 have any practical relevance, we need to consider more details from the TETRA specifications.[5] In practice, the TETRA authentication protocol is running within the Location Update/Registration procedure, which is a Mobility Management (MM) service, or it is running if an application on the *MS* explicitly requests authentication via the Mobility Management Service Access Point (TNMM-SAP).

The registration procedure starts with the *MS* sending the `U-LOCATION UPDATE DEMAND` message to the infrastructure. The infrastructure may then (optionally) initiate the authentication process by sending the `D-AUTHENTICATION DEMAND`

[5]Alternatively, one could try to implement our attacks. However, we did not have access to a TETRA test network, nor did we consider it an option to harm TETRA networks in operation.

message to the *MS*, *i.e.*, the first message of the authentication protocol. After receiving the `U-AUTHENTICATION RESULT`, *i.e.*, the last authentication protocol message, the infrastructure returns `D-LOCATION UPDATE ACCEPT` message to the *MS*. The protocol data unit (PDU) of the `D-LOCATION UPDATE ACCEPT` message consists of the authentication result R1, the cipher key (CK) exchange information and TEI request flag indicates whether the infrastructure request TEI from the MS. For mutual authentication (*i.e.*, in security class 3), CK indicates the *Common Cipher Key* (CCK). Each Location Area (LA) uses a single CCK at any given time, *i.e.*, it is shared by all mobile stations in the same LA. The CCK is used to protect traffic and signalling sequences from the infrastructure to the mobile stations; either by using the CCK directly as encryption key or, in case of group calls, by using the CCK-derived corresponding modified group cipher key (MGCK).

Now we consider the feasibility of the three attacks from Section 3.2.1:

Attack 1 in Practice During location update/registration: After an attacker modifies the value of *R2* in the last protocol message from `true` to `false`, the network believes that authentication failed and sends a `D-LOCATION UPDATE REJECT` message. In order to make the *MS* believe that authentication/registration is completed successfully, the attacker needs to forge a `D-LOCATION UPDATE ACCEPT` message and substitute it for the `D-LOCATION UPDATE REJECT` message. According to the TETRA specifications, the `D-LOCATION UPDATE ACCEPT` PDU does not necessarily contain message parts that prevent such a forgery. There is the option of an *MS* requesting the CCK in the `U-LOCATION UPDATE DEMAND` message, in which case, the `D-LOCATION UPDATE ACCEPT` message must carry the CCK encrypted under the session key DCK [9](*e.g.*, page 56). This makes the forgery of the `D-LOCATION UPDATE ACCEPT` PDU infeasible, since the attacker cannot learn the DCK. However, we note that requesting the CCK in the `U-LOCATION UPDATE DEMAND` message is not mandatory. According to the specification [9] (on page 30), a once-registered *MS* may store one or more CCKs if it detaches from the network, and, therefore, it may not request the CCK during re-registration. In practice, an attacker may force re-registration by temporarily jamming mobile stations in a given cell, which will afterwards try to re-connect/-register, giving the attacker the opportunity to launch Attack 1. As a consequence, targeted mobile stations (and their users) may falsely believe that they are connected to the network and ready to receive messages. This can, at the very least, cause a loss of valuable time in which the holder of these mobile stations cannot communicate with their coordinator and colleagues.

During application requested authentication: Authentication requested explicitly by an application running on the *MS* is not followed by something like the `D-LOCATION UPDATE ACCEPT` message. As shown in the diagram in [9] (on page 48), right after the *MS* sends out the authentication result *R2* to the *SwMI*, the user application shall receive a `TNMM AUTHENTICATE` confirm message. This means that in this case the *MS* is not waiting for a message from the *SwMI* that is generated using the session key. Hence, Attack 1 should be applicable in a straight-forward way and the user should get a positive authentication results from its user application, leaving the user to believe that the terminal equipment is working fine, not realizing that he or she is under attack.

Attack 2 in Practice On a protocol level, this attack achieves that *SwMI* believes that authentication was successful, while *MS* believes that it failed, preventing an *MS* to communicate with the *SwMI*. However, in practice, *MS* will initiate cell re-selection if the check on *RES2* fails, as specified in [9] (on page 46), *i.e.*, *MS* will try to connect to the network through a different base station. This cell re-selection applies to both location update/registration and user application initiated authentication. This makes the execution of this attack difficult in practice as the attacker needs to control multiple base stations within the targeted location area. In an emergency situation, a rescue worker may miss critical instructions in the period, in which the *MS* is establishing re-connection while the *MS* appears already connected to the network (and the rescue coordinators).

Attack 3 in Practice Out of the three attacks presented in this work, Attack 3 appears to be the least feasible in practice. This attack requires an *MS* to run two concurrent sessions. However, according to specifications [9] (on page 44), an authentication sequence that has begun but has not yet completed is called "pending", and [9] states (on page 44) that "[if] a new authentication is started then any pending authentication shall be abandoned". Again, this applies to both location update/registration and user application initiated authentication. This seems to prevent an *MS* from running two concurrent sessions of the authentication protocol. Thus, this attack is not exploitable if a system is correctly implemented.

3.3 Fixing TETRA Authentication

It is straight-forward to see that the attacks against TETRA are possible because

(1.) lack of integrity protection on *R2*[6]

(2.) lack of binding of *RES1* and *RAND2*[7]

One can easily verify that (1) enables Attacks 1 & 2, while (2) enables Attack 3. Therefore, we propose to fix the TETRA authentication protocol by

(A) adding integrity protection of *R2* by adding the value of a message authentication function $TC1$ over *R2* and nonce *RAND2*, and keyed by KS''; *i.e.*, the last message sent by *MS* should be changed from *R2* to $R2, TC1(R2, RAND2, KS'')$.

(B) adding binding of *RES1* and *RAND2* by adding the value of message authentication function $TC1$ over *RES1* and nonce *RAND2*, keyed by KS''; *i.e.*, instead of $RES1, RAND2$ the *MS* sends $RES1, RAND2, TC1(RES1, RAND2, KS'')$.

In both 2 and A, the KS'' should be generated by *MS* and *AuC* from *RS* and their long-term key K using a key derivation function that is different from $TA11$ and $TA21$. Our proposed fixes take into account the network infrastructure in practice, where the mobile station initially shares secrets only with the *AuC* but none with the base stations. Our proposed fixes require the introduction of an additional key derivation function (for generating KS'', so none of KS, KS', KS'' is used for multiple cryptographic primitives) and an additional function $TC1$ for

[6] of *R1* in the protocol of Fig. 2

[7] of *RES2* and *RAND1* in the protocol of Fig. 2

```
    role MS  {
var RS: Nonce;
var RAND1:  Nonce;
fresh RAND2:  Nonce;
recv_1 (SwMI,MS, RAND1, RS);
send_2 (MS,SwMI, TA12(TA11(k(MS,SwMI),RS), RAND1), RAND2,
        TC1(TA12(TA11(k(MS,SwMI),RS), RAND1), RAND2, TAX1(k(MS,SwMI),RS)));
recv_3 (SwMI,MS, TA22(TA21(k(MS,SwMI),RS),RAND2), R1);
send_4 (MS,SwMI, R2, TC1(R2,RAND2,TAX1(k(MS,SwMI),RS)));
claim_MS1 (MS,Niagree);
claim_MS2 (MS, Weakagree);
claim_MS3 (MS,Nisynch);
claim_MS4 (MS,Secret,TB4(TA12b(TA11(k(MS,SwMI),RS), RAND1),
           TA22b(TA21(k(MS,SwMI),RS), RAND2)));
}
```

Figure 5: Example input to the Scyther tool

message authentication. There are no additional messages introduced and the additional computational overhead can be negligibly small. We note that, in mutual authentication, sending *R1* is redundant, as it is implicit by sending a challenge *RAND2*, that the check on *RES1* passed. Similarly, there is no need to send R2, only the resulting value $TC1(R2, RAND2, KS'')$.

4 Formal Verification of TETRA Authentication

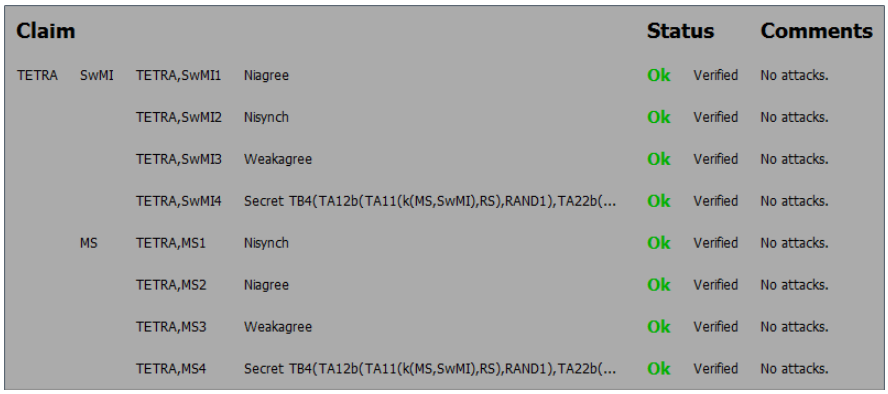
4.1 Scyther Basics

Scyther [4] is a tool for the automatic verification and falsification of security protocols. It is freely available at [4]. Scyther analyses protocols with respect to the Dolev-Yao intruder model. Its algorithm is guaranteed to terminate, at which point Scyther establishes unbounded verification (*i.e.* for an unbounded number of protocol session and freshly generated values, and an unbounded message size), falsification (by presenting an attack trace), or bounded verification (*i.e.* the assurance that no attacks exist within a certain bound) of a wide range of basic authentication and secrecy properties, including the properties presented in Section 3.1.2. A description of Scyther's patterns and pattern refinement algorithm, based on the approach of [13], is beyond the scope of this work. For a detailed description of Scyther's theoretical foundations, we refer to [2, 5].

Scyther has been deployed to analyze the standards of industrial security protocols IKE (v1 & v2) [3] and ISO/IEC 9798 [1]. In [3], it was used to verify more advanced security properties of authenticated key exchange security models.

4.1.1 Input language Basics

In general, messages are modeled as terms which can be constructed from atomic terms and constructor function symbols. An atomic term is any alphanumeric string, which represents, *e.g.*, a (global) constant, a nonce or a variable. Scyther comes with predefined constructors for pairing, hash functions, signing and encryption, of which the latter constructor can be used for both for symmetric and asymmetric encryption. Scyther's language is typed: There are predefined types including `Agent`



Claim				Status		Comments
TETRA	SwMI	TETRA,SwMI1	Niagree	Ok	Verified	No attacks.
		TETRA,SwMI2	Nisynch	Ok	Verified	No attacks.
		TETRA,SwMI3	Weakagree	Ok	Verified	No attacks.
		TETRA,SwMI4	Secret TB4(TA12b(TA11(k(MS,SwMI),RS),RAND1),TA22b(...	Ok	Verified	No attacks.
	MS	TETRA,MS1	Nisynch	Ok	Verified	No attacks.
		TETRA,MS2	Niagree	Ok	Verified	No attacks.
		TETRA,MS3	Weakagree	Ok	Verified	No attacks.
		TETRA,MS4	Secret TB4(TA12b(TA11(k(MS,SwMI),RS),RAND1),TA22b(...	Ok	Verified	No attacks.

Figure 6: Scyther Results for TETRA Authentication with proposed fixes of Sec. 3.3.

(for protocol participants) and `Nonce`. Protocols are modeled as sets of roles; for instance, in the case of TETRA one declares in the input file (after the declaration of user types, functions and global constants): `protocol TETRA(`*MS*`,` *SwMI*`){ role` *MS*`{...} role` *SwMI*`{ ...} }`. Each role is a set of events (mostly send and receive events, but also some claims and pattern matching).

The role `role` *MS* of the *MS* as in the network initiated authentication of Figure 1, is given in Figure 5. At the beginning, there are two variable declarations for nonces *RS* and *RAND1* that are received during a protocol run. Furthermore, a nonce *RAND2* is generated. Then the event `recv_1`$(SwMI, MS, RAND1, RS)$ denotes that the *MS* receives, supposedly from the network *SwMI*, the message consisting of the two nonces $RAND1, RS$. Then in event `send_2`$(MS, SwMI, \ldots)$ the *MS* sends to the intended receiver *SwMI* the message $TA12(TA11(k(MS, SwMI), RS), RAND1), RAND2$ together with the binding discussed in Section 3.3, where $TA12, TA11, TAX1$ and $TC1$ are globally declared as hash functions and $k(MS, SwMI)$ denotes the symmetric key shared between *MS* and *SwMI*. Here $TA11(k(MS, SwMI), RS)$ models *KS* and $TA12(TA11(k(MS, SwMI), RS), RAND1)$ models *RES1*. Then, in the next event `recv_3`$(\ldots)$, the *MS* receives a message that must equal $TA22(TA21(k(MS, SwMI), RS), RAND2)$ and the constant *R2*, *i.e.*, KS' equals $TA21(k(MS, SwMI))$. Again, *TA22* and *TA21* are modeled as hash functions. We note that in our model the constants *R1* and *R2* only represent the value `true`, *i.e.*, we are not modeling the traces in which *R1* $\leftarrow$ `false` or *R2* $\leftarrow$ `false` are explicitly sent by the honest protocol participants. The last send event in `role` *MS* is the sending of *R2* $\leftarrow$ `true` together with its integrity protection (see Section 3.3).

4.2 Verification Results of TETRA Authentication Protocol

We verified the security of our proposed fixes to TETRA authentication with Scyther.[8] For the *MS*, we asked Scyther to verify the properties *ni-agree*, *weak agreement* and *ni-synch* as well as the secrecy of the session key, by adding `claim_MS1(MS,Niagree)`, `claim_MS1(MS,Niagree)`, `claim_MS3(MS,Nisynch)` and `claim_MS4(MS,Secret,TB4(TA12b(TA11(k(MS,SwMI),RS),RAND1),TA22b(TA21(k(MS,SwMI),RS),RAND2)))` to the end of `role` *MS* (see Fig. 5). For the role of the network (`role` *SwMI*) we asked Scyther to verify the symmetric claims. Scyther's GUI output for the verification of these properties for both `role` *MS* and `role` *SwMI* is

[8] In order to obtain our Scyther input scripts, please contact one of the authors.

given in Figure 6. Scyther indeed verified all these security properties for the TETRA authentication protocol after fixing it (as proposed in Sec. 3.3). We also used Scyther to verify the corresponding claims of the original TETRA authentication protocol: it found that ni-agree and ni-synch were violated in both `role` *MS* and `role` *SwMI*.

5 Conclusion

We have analyzed the Terrestrial Trunked Radio (TETRA) authentication protocol. We show several novel protocol-level attacks against TETRA and discuss real-world attack scenarios, which allow an adversary to disrupt availability but which are prevented by the TETRA specifications. We propose fixes to the TETRA authentication protocol, which we verify with the automated tool Scyther with respect to the Dolev-Yao security model.

As the disruption of availability can be particularly harmful for emergency networks, our results may especially be interesting for authorities of those countries, where the roll-out of TETRA has not started or been completed yet, *e.g.*, in Norway.

Acknowledgement: We thank Cas Cremers for helpful comments on verifying authentication properties with Scyther.

References

[1] D. A. Basin, C. J. F. Cremers, and S. Meier. Provably repairing the ISO/IEC 9798 standard for entity authentication. In *POST*, pages 129–148. Springer, 2012.

[2] C. Cremers. Unbounded verification, falsification, and characterization of security protocols by pattern refinement. In *ACM Conference on Computer and Communications Security*, pages 119–128, 2008.

[3] C. Cremers. Key exchange in IPsec revisited: Formal analysis of IKEv1 and IKEv2. In *ESORICS*, pages 315–334, 2011.

[4] C. Cremers. The scyther tool, 2012. `http://people.inf.ethz.ch/cremersc/scyther/index.html`, accessed 2013-05-10.

[5] C. Cremers and S. Mauw. *Operational semantics and verification of security protocols*. Information Security and Cryptography. Springer, 2012.

[6] Direktoratet for nødkommunikasjon. Motorola solutions har overtatt totalansvaret for utbygging og drift av nødnett. `http://www.dinkom.no/DNK/Nyhetsarkiv/Motorola-har-overtatt-totalansvaret/`, 2012-02-27.

[7] D. Dolev and A. C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[8] S. Duan. Security Analysis of TETRA. Master's thesis, Norwegian University of Sciences and Technology, Trondheim, 2013.

[9] ETSI. EN 300 392-7 v3.3.1 (2012-07): Terrestrial trunked radio (TETRA); voice plus data; part 7: Security, 2012.

[10] G. Lowe. A hierarchy of authentication specifications. In *Computer Security Foundations Workshop*, pages 31–43. IEEE Computer Society Press, 1997.

[11] Y.-S. Park, C.-S. Kim, and J.-C. Ryou. The vulnerability analysis and improvement of the TETRA authentication protocol. In *IEEE ICACT*, volume 2, pages 1469–1473, 2010.

[12] G. Roelofsen. TETRA Security. *Inf. Sec. Techn. Report*, 5(3):44–54, 2000.

[13] D. X. Song. Athena: a new efficient automatic checker for security protocol analysis. In *In Proceedings of the Twelth IEEE Computer Security Foundations Workshop*, pages 192–202. IEEE Computer Society Press, 1999.