A Formal Security Analysis of ERTMS Train to Trackside Protocols

Joeri de Ruiter, Richard J. Thomas, and Tom Chothia

School of Computer Science
University of Birmingham
United Kingdom

Abstract. This paper presents a formal analysis of the train to trackside communication protocols used in the European Railway Traffic Management System (ERTMS) standard, and in particular the EuroRadio protocol. This protocol is used to secure important commands sent between train and trackside, such as movement authority and emergency stop messages. We perform our analysis using the applied pi-calculus and the ProVerif tool. This provides a powerful and expressive framework for protocol analysis and allows to check a wide range of security properties based on checking correspondence assertions. We show how it is possible to model the protocol's counter-style timestamps in this framework. We define ProVerif assertions that allow us to check for secrecy of long and short term keys, authenticity of entities, message insertion, deletion, replay and reordering. We find that the protocol provides most of these security features, however it allows undetectable message deletion and the forging of emergency messages. We discuss the relevance of these results and make recommendations to further enhance the security of ERTMS.

1 Introduction

The European Railway Traffic Management System (ERTMS) is a European standard for next-generation train management and signalling. It is intended to make it easier for trains to cross borders and optimise the running of the railway. Currently the system is being rolled out across Europe, and on high-speed lines across the world. By the end of 2014, over half of the 80,000 kilometres of tracks that were equipped with ERTMS were located in Asia.[1]

Within this wholly-digitised system, a number of protocols are employed to provide functionality to the ERTMS platform. For example, the EuroRadio protocol is used to ensure that messages exchanged between entities are genuine and have not been forged by an attacker, or to handover trains from one system responsible for a stretch of track to another. Moving from a largely analogue, manual or semi-automatic system to a digital, fully supervised system may expose it to threats which were not previously possible. These threats require

[1] `http://www.ertms.net`

appropriate analysis to ensure that the replacement system protects the underlying infrastructure and vehicles from attacks. In such a safety-critical system, it is key that the train is never allowed to be influenced externally to enter an unsafe state or perform in a manner which is not expected.

In this paper we perform a formal analysis of the EuroRadio protocol and parts of the ERTMS application protocol using the applied pi-calculus [1] and the ProVerif analysis tool [4, 5]. The applied pi-calculus provides an expressive, powerful framework to model protocols; functions can be used to define new cryptographic primitives. The ProVerif tool can automatically check a wide range of security properties including the secrecy of particular values, equivalence between processes and correspondence assertions between modeller-defined events. ProVerif uses a theorem proving method to establish if these queries hold, therefore it is able to establish if secrecy properties hold even in the face of an active attacker, for an unlimited number of protocol runs and arbitrary attacker behaviour. However, it may not always terminate and it makes the usual Dolev-Yao assumptions: i.e., the cryptography is unbreakable, the attacker cannot learn key material by other means than observing communication and interacting with the protocol participants, etc. The applied pi-calculus's expressiveness and the powerful checking methods of ProVerif have led to them being used to analyse a wide range of security properties for many important systems.[2]

We model the EuroRadio protocol in the applied pi-calculus, with one process representing the train side of the communication and another process representing the Radio Block Controller (RBC) which receives messages from the train. Our model allows for an arbitrary number of trains and RBCs running at the same time, and, using standard ProVerif methods, we can check if EuroRadio keeps its keys secret and successfully authenticates the trains and the RBC. After the EuroRadio protocol finishes, we model the application level sending three messages. These application level messages sent over EuroRadio use a counter-style timestamp to help ensure freshness and stop attacks, where we introduce new functions to model this. We tag our model with events indicating each party starting a run of EuroRadio, finishing a run of EuroRadio, sending messages and receiving messages. We then come up with novel correspondence assertions between these events which let us check if messages can be deleted, inserted, reordered or replayed.

Checking our correspondence assertions in ProVerif, we find that the protocol succeeds in most of its security goals, i.e., an attacker cannot learn the secret keys in use, or pretend to be a train or a RBC. Furthermore, after successfully completing a run of the EuroRadio protocol both sides will have securely established a secret session key. However, we also find that the attacker may delete/jam messages without this being detected, they can inject emergency stop messages into a communication between an train and an RBC, and that an attacker may change the "safety feature" in a communication, possibly downgrading security. These issues could be looked at as moderately security critical, we do not believe

[2] A collection of such studies can be found at `http://prosecco.gforge.inria.fr/personal/bblanche/proverif/proverif-users.html`

that they require immediate fixes but that designers and train operators should be aware of them.

Related Work: Some past work has also looked at the EuroRadio protocol: Esposito et al. [10] and Franekova et al. [11] use UML, Zhang et al. [18] use the SPIN model checker and Hongjie et al. in [14] use Petrinets. However, all of these analyses only look at single runs of the protocol, they do not consider an active attacker, and they do not try to test the security properties we focus on in this paper, rather they look at general correctness issues such as deadlock detection. A generic analysis of ERTMS was performed by Bloomfield et al. [6], however the paper itself gives a high-level overview of the process involved, and does not specify exact issues and mitigations. Our methodology in this paper is similar to our previous work that has included looked at modelling EMV protocols [8] and e-passports [3] in the applied pi-calculus. Other work looks at complex models of time in the applied pi-calculus (e.g. [7, 15]) - our novel modelling of counter-style timestamps provides an abstract model of time which is much similar than these, but still expressive enough to model ERTMS.

The contributions of this paper are:

- Formal analysis of the EuroRadio protocol using the ProVerif tool.
- Introduction of a light-weight notion of counter-style timestamps in ProVerif.
- Showing how it is possible to use ProVerif to check if the attacker can delete, insert and re-order messages.
- Identification of potential issues in the ERTMS protocols, with appropriate recommendations.

In Section 2, we describe ERTMS and the EuroRadio protocol. We describe our formal model in Section 3 and then analyse this model in Section 4. We discuss the implications of our results in Section 5, and then we conclude in Section 6.

2 ERTMS Communication

In this section, we present a high-level overview of the components within ERTMS that are used for communication between the train and trackside equipment.

During its journey, a train communicates with a *Radio Block Centre (RBC)*, which provides commands to the train. RBCs are responsible for a specific geographical area of approximately 70 kilometres [2]. They authorise trains to drive on particular parts of the track using Movement Authorities, which also include maximum speeds. Every RBC is connected to a fixed network in order to hand over trains to the next RBC when a train leaves its area of responsibility.

Within ERTMS, several layers are used for communications between the train and trackside (see Figure 1), where each layer provides some services and security features to upper layers.

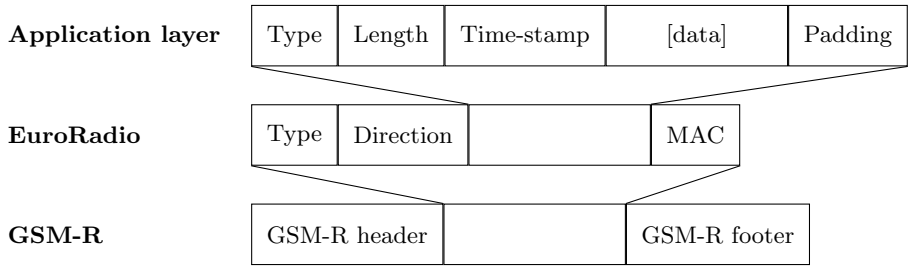
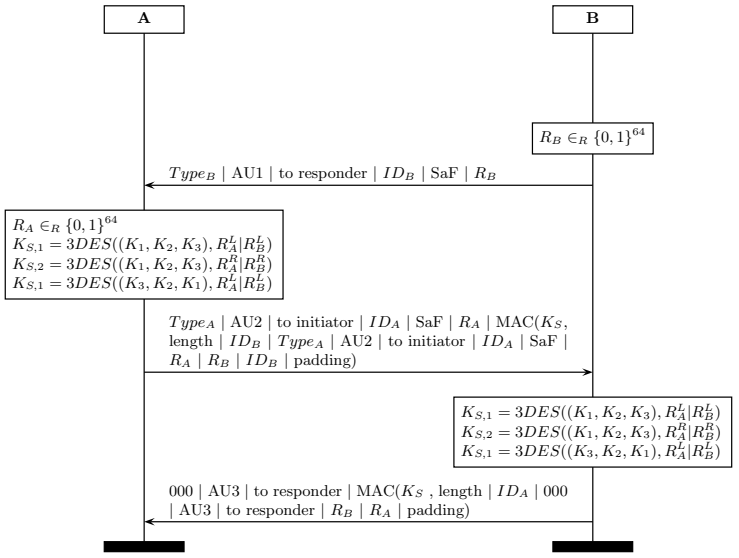


Fig. 1. Overview of the different communication layers in ERTMS.

GSM-R is the lowest layer for the communication between trains and the back-end specified in ERTMS [13, 12]. It is based on the original GSM specification, but provides additional rail-specific functionality and makes use of different frequencies. The additional functionality includes emergency calls and communications involving multiple drivers. Also, pre-defined short messages are included in the specification, which may be sent by driver and signaller, for example, 'standing at signal' [16], where the signaller may also send a message to the train at any time informing the driver that they must contact the signaller.

The EuroRadio Protocol is used on top of GSM-R and added to provide additional authentication and integrity protection to the communication [17]. EuroRadio uses the GSM-R communication layer to send messages between the base station and the train. When a connection is set up, an authentication protocol is used to provide mutual authentication for the train and back-end (see Figure 2). The two parties exchange nonces and compute a shared symmetric session key based on this and a unique train key. This session key is then used to compute a MAC to prove knowledge of the session key to the other party. Once the authentication protocol has completed successfully, the application layer can use the communication channel and the EuroRadio layer will add a MAC to all messages that have a normal priority. Exactly which MAC algorithm is used is indicated in a data field called the *Safety Feature*.

The Application Layer builds on top of the EuroRadio layer and is described in [9]. As not all threats are taken care of by the lower layers, the application layer has to provide protection against replay and deletion attacks. A 32 bit timestamp is added to the messages. Every received message needs to contain a timestamp that is greater than that of the previous message, the exact value of the time stamp is not important, therefore it acts partly as a counter. If the timestamp is not greater than the last message received, the new message will be discarded. In order to synchronise the time between the train and the RBC, the RBC will maintain multiple clocks and sets them based on the time from the train.



Key	Description
AU_x	Authentication Message x
ID_x	ETCS Entity ID of party x
R_x	Nonce generated by party x
SaF	Safety Feature selected
To Initiator/Responder	Flag to indicate in which direction the message is being sent
$Type_x$	ETCS Entity ID type (e.g. RBC or train)

Fig. 2. Authentication protocol used by EuroRadio. A and B share a symmetric key K. R^L and R^R are used to indicate the leftmost and rightmost 32 bits respectively. A 3DES key K consists of three single DES keys: $K = (K_1, K_2, K_3)$.

3 Formal Modelling in ProVerif

We performed our formal analysis of the EuroRadio key establishment protocol and part of the application level protocol using the applied pi-calculus [1] and the ProVerif automated verifier [4]. This protocol analysis framework can be used to identify potential leakage of information or other flaws in the protocols. The ProVerif syntax for the applied pi-calculus is given in Figure 3.

This language allows us to specify processes that perform inputs and outputs, run in parallel and replicate. The calculus also allows processes to declare new, private names which can be used as private channels or nonces [5]. Functions in the applied pi-calculus can be used to model a range of cryptographic primitives, e.g. MACs, signing and key generation. These functions abstract any implementations, which are therefore considered to be cryptographically perfect. In our analysis, we focus only on the protocol, rather than any weaknesses and

$M, N ::=$	terms
x, y, z	variables
a, b, c, k, s	names
$f(M_1, \ldots, M_n)$	constructor application
$D ::= g(M_1, \ldots, M_n)$	destructor application
$P, Q ::=$	processes
0	nil
out $(M, N).P$	output N on channel M
in $(M, x).P$	input x from channel M
$P \mid Q$	parallel composition
$!P$	replication
$\nu\, a.P$	create new name
let $x = D$ in P else Q	term evaluation
$event(x)$	execute event

Fig. 3. Syntax of the applied pi calculus

exposure as the result of cryptographic schemes used. The "let" statement can be used to check that two terms that used these equations are equal and branch on the result. This can be used to encode "if" statements, and conditional inputs *in* $(c, =a).P$ which inputs a value from channel c and proceeds only if the value received equals a (see e.g. [5]). For the verification, events can be added to a model. These events can be used to identify critical points of the protocol, and may be parameterised with variables from the model. Currently, the ProVerif tool is able to make guarantees for soundness, where if no attack is found, it is correct, however, it is not complete and might return false attacks [5].

ProVerif supports several types of queries to check security properties of protocols. The most basic type is to check for secrecy, i.e. whether the attacker is able to learn specific values. This can be used to verify whether cryptographic keys do not leak in a protocol. Another type of query is correspondence assertions, which can be used to check that if a particular event is executed, another event was executed before. Two types of correspondence assertions can be checked by ProVerif: non-injective and injective. An example of a non-injective query is `ev:event1(vars) ==> ev:event2(vars)`, which holds if `event2` was executed at some point before `event1`. For an injective assertion to hold, say `evinj:event1(vars) ==> evinj:event2(vars)`, for every execution of `event1` there must have been a *unique* execution of `event2`.

Our Model of EuroRadio Our models of the two parties in the EuroRadio protocol are given in figures 4 and 5. For the analysis of normal priority messages, these processes are followed by the ones in figures 6 and 7 respectively. In these last processes three messages are sent and received, where MACs and timestamps are added and checked. Another model was constructed to check high-priority messages. This model is almost the same as for the normal priority messages,

```
let Train =
    (* Set up a new session for the model *)
    (* Create a fresh session identifier used to link different events
        in the model *)
  new session;
  (* Get the identity of the RBC the train wants to communicate with *)
  in(id, rbc_etcs_id);
  (* Start of the actual authentication protocol *)
  (* T-CONN.request -- Au1 SaPDU *)
  new trainNonce;
  event trainStartSession(rbc_etcs_id, train_etcs_id, trainNonce, SAF);
  out(c, (TRAIN_ETCS_ID_TYPE, AU1, DF_SEND, train_etcs_id, SAF,
      trainNonce));
  (* T-CONN.confirmation -- Au2 SaPDU *)
  in(c, (=RBC_ETCS_ID_TYPE, =AU2, =DF_RESP, in_rbc_etcs_id, rbcSaF,
      rbcNonce, inMAC));
  (* Generate the session key *)
  let trainKS = genSessionKey(trainNonce, rbcNonce, getKey(
      in_rbc_etcs_id, train_etcs_id)) in
  (* Output encrypted secret to check secrecy of keys *)
  out(c, encrypt(SECRET, trainKS));
  out(c, encrypt(SECRET, getKey(in_rbc_etcs_id, train_etcs_id)));
  (* Verify whether the received MAC is correct *)
  if inMAC = mac(trainKS, ((PAYLOAD_LENGTH, train_etcs_id,
      RBC_ETCS_ID_TYPE, AU2, DF_RESP, in_rbc_etcs_id, rbcSaF), rbcNonce,
       trainNonce, train_etcs_id)) then
  (* T-DATA.request -- Au3 SaPDU *)
  event trainFinishSession(in_rbc_etcs_id, train_etcs_id, trainNonce,
      rbcSaF, rbcNonce, trainKS);
  out(c,(ZEROS, AU3, DF_SEND, mac(trainKS, (PAYLOAD_LENGTH,
      train_etcs_id, ZEROS, AU3, DF_SEND, trainNonce, rbcNonce))))
```

Fig. 4. The ProVerif model of the calling party in the EuroRadio protocol

except no MACs are added to messages in the application layer and timestamps are not checked. All models are available online.[3]

The expressive language of ProVerif allows us to define processes which are run by the verifier in a number of ways. In our model, we instantiate both models for the RBC and train as replicating processes using the '!' command, which may be nested, i.e. an arbitrary number of trains and RBCs can be run in parallel. This allows the verifier to provide a thorough examination of the protocol, giving the attacker in ProVerif the opportunity to reuse variables it has previously observed in previous protocol runs. ProVerif is then able to assess whether the properties defined hold, or it provides a trace if an attack is found.

Next we will discuss the models in figures 4 and 5. To represent the EuroRadio protocol, we must first introduce a session value which allows us to perform additional verification on the protocol for the reordering and replay of messages. Additionally, during the setup process, the train and RBC are sent the identity of the RBC. This allows us to assert that the train knows the identity of the RBC it is connecting to. The session, as specified by the EuroRadio specifications then starts, where the nonces and identities are exchanged, with the appropriate derivation of the session key to use. We generate and output some secret value encrypted with the negotiated session key. The confidentiality of this secret value

[3] `http://www.cs.bham.ac.uk/~rjt195/rssrail2016`

```
let RBC =
    (* Set up a new session for the model *)
  (* Get an RBC identity *)
  in(id, rbc_etcs_id);
  (* Start of the actual authentication protocol *)
  (* T-CONN.indication -- Au1 SaPDU *)
  new rbcNonce;
  in(c, (sent_ETCS_ID_TYPE, =AU1, =DF_SEND, in_train_etcs_id, trainSaF,
      trainNonce));
  event rbcStartSession(rbc_etcs_id, in_train_etcs_id, rbcNonce,
      trainSaF, trainNonce);
  (* Generate the session key *)
  let rbcKS = genSessionKey(trainNonce, rbcNonce, getKey(rbc_etcs_id,
      in_train_etcs_id)) in
  (* Output encrypted secret to check secrecy of keys *)
  out(c, encrypt(SECRET, rbcKS));
  out(c, encrypt(SECRET, getKey(rbc_etcs_id, in_train_etcs_id)));
  (* T-CONN.response -- Au2 SaPDU *)
  out(c, (RBC_ETCS_ID_TYPE, AU2, DF_RESP, rbc_etcs_id, trainSaF,
      rbcNonce, mac(rbcKS, ((PAYLOAD_LENGTH, in_train_etcs_id,
      RBC_ETCS_ID_TYPE, AU2, DF_RESP, rbc_etcs_id, trainSaF), rbcNonce,
      trainNonce, in_train_etcs_id))));
  (* AU3 SaPDU *)
  in(c,(=ZEROS, =AU3, =DF_SEND, inMAC));
  (* Verify whether the received MAC is correct *)
  if inMAC = mac(rbcKS, (PAYLOAD_LENGTH, in_train_etcs_id, ZEROS, AU3,
      DF_SEND , trainNonce, rbcNonce)) then
  event rbcFinishSession(rbc_etcs_id, in_train_etcs_id, rbcNonce,
      trainSaF, trainNonce, rbcKS)
```

Fig. 5. The ProVerif model of the called party in the EuroRadio protocol

is checked to verify that the attacker is not able to establish the session key. At each stage of messages being received, we verify the MAC prior to proceeding with protocol execution. This simulates the process that is in use within EuroRadio. After this, the EuroRadio link is established. We then are able to use one of two different variants of the model - for normal or high-priority messages.

Figures 6 and 7 show the application messages sent through EuroRadio, including the use of timestamps. Once the session is established, we generate some value for a timestamp, and proceed to use it when sending messages. Each time, we use a light-weight notion of time which we discuss below. The RBC then verifies the timestamps were greater than that of the previous received

```
  (* Send three messages from the train to the RBC *)
  new time;
  let msg1 = (DT, time, MESSAGE_1) in
  event DataSent1(session, msg1);
  out(c, (msg1, mac(trainKS, msg1)));
  let msg2 = (DT, inc(time), MESSAGE_2) in
  event DataSent2(session, msg2);
  out(c, (msg2, mac(trainKS, msg2)));
  let msg3 = (DT, inc(inc(time)), MESSAGE_3) in
  event DataSent3(session, msg3);
  out(c, (msg3, mac(trainKS, msg3)))
```

Fig. 6. The ProVerif model of the application layer to send messages with normal priority

```
(* Receive messages from the train *)
in(c, ((=DT, timeA, msgA), macA));
(* Check the MAC of the received message *)
if macA = mac(rbcKS, (DT, timeA, msgA)) then
event DataReceived1((DT, timeA, msgA));
in(c, ((=DT, timeB, msgB), macB));
(* Check the MAC and timestamp of the received message *)
if macB = mac(rbcKS, (DT, timeB, msgB)) then
if greater: timeB, timeA then
event DataReceived2((DT, timeB, msgB));
event MessagesReceived2((DT, timeA, msgA), (DT, timeB, msgB));
in(c, ((=DT, timeC, msgC), macC));
(* Check the MAC and timestamp of the received message *)
if macC = mac(rbcKS, (DT, timeC, msgC)) then
if greater: timeC, timeB then
event DataReceived3((DT, timeC, msgC));
event MessagesReceived3((DT, timeA, msgA), (DT, timeB, msgB), (DT,
    timeC, msgC))
```

Fig. 7. The ProVerif model of the application layer to receive messages with normal priority

message, and if it is, it will accept the message and execute the appropriate event to indicate that it was received in the context of that session. We include the session to verify that an attacker cannot combine messages from different sessions.

Modelling Counter-style Timestamps To support the checking of timestamps, we add a minimal notion of time to our model. In the application layer, it is checked whether the timestamp on a message is greater than on the previous message. For the time a counter on the train is used. In our model, we therefore only modelled relative time: time can increase and we can compare different timestamps that are based on the same initial timestamp. This mean we have no notion of how much time actions take, but our model proves to be sufficient for its purpose. In Figure 8, our model of time can be found. A timestamp can be increased using *inc*, and two timestamps can be compared using the predicate *greater*.

```
data inc/1.
pred greater/2.
clauses
  greater: inc(x),x;
  greater: x,y -> greater: inc(x),y.
```

Fig. 8. The ProVerif model for counter-style timestamps

4 Analysis of ERTMS Protocols

Using ProVerif, we can check that the protocol keeps the keys secret and that an attacker cannot disrupt the agreement process. These checks are standard

ProVerif queries. Next, we wish to check if an attacker can insert, reorder, replay or delete messages without being noticed. Our methods of doing this are new, and a contribution of this paper. We perform these checks by making the train send three messages to the RBC and tagging each of these with a particular event. We also use events to tag the three messages send by the train, and their order, and the three messages received by the RBC and their order. We then check for insertion, reordering, replay and deletion using queries on these events.

Secrecy of Keys We check if the EuroRadio protocol keeps the long term RBC/train key and session key secret from an active attacker. This check is performed by creating a new private value 'SECRET', encrypting this value using these keys and publicly broadcasting the encryption. If the attacker can then learn the value 'SECRET' it means the keys have been learnt. We checked this using the query `attacker:SECRET`. This verifies whether the attacker is able to establish the value 'SECRET'. Private values are not disclosed to the attacker and 'SECRET' is only output on the public communication encrypted using the long term and session key. Therefore, if the attacker is able to learn the value 'SECRET' this means at least one of the keys was compromised.

Running ProVerif, we find that the attacker cannot learn the value 'SECRET', this means that the EuroRadio protocol succeeds in its main goal of keeping the cryptographic keys secure from an active Delov-Yao attacker. The theorem proving method of ProVerif, further tells us that this holds for an unlimited number of runs of the protocol.

Agreement on Shared Session Key Even if attackers cannot learn the session key, they may still be able to interfere with the key establishment process. To check if any such attacks are possible, we use the injective ProVerif correspondence assertions:

```
evinj:trainFinishWithKey(ks) ==> evinj:rbcUsing(ks)
evinj:RBCFinishWithKey(ks) ==> evinj:trainUsing(ks)
```

These queries will only hold if, whenever the train believes it has successfully completed the EuroRadio protocol having established the key ks, then there is a single RBC that has also run the protocol and believes the established key is ks, and vice versa. ProVerif tells us that these queries hold, therefore the EuroRadio protocol succeeds in its second major goal of security and successfully setting up a key between a train and a RBC.

Mutual Authentication: Agreement on All Shared Values To check if it is possible for an attacker to interfere with any other parts of the protocol we extend our queries with all the key values used by the train and the RBC, i.e., the nonces, the trains and RBC identities and the safety feature (SaF):

```
evinj:trainFinishSession(rbc_id,train_id,train_nonce,saf,
  rbc_nonce,ks) ==>
evinj:rbcStartSession(rbc_id,train_id,rbc_nonce,saf,train_nonce)

evinj:rbcFinishSession(rbc_id,train_id,rbc_nonce,saf,train_nonce,
  ks) ==>
evinj:trainStartSession(rbc_id,train_id,train_nonce,saf)
```

While the first correspondence assertion holds, the second fails. Looking at the attack trace produced by ProVerif, we see that it is possible for the attacker to redirect the messages from the train to a second, different RBC as the train does not verify whether the returned ID is the same as the expected one. While implemented systems might add a check of the RBC ID, the protocol specification does not specify that the train explicitly checks it, or what to do if it is incorrect. Second, we see that it is possible for an attacker the change the SaF used in the communication as, again, this is not properly checked. We discuss the relevance of these findings in the section below.

Ability to Insert Attacker Messages We use the event `DataSent‘i’(m)` to mean that message `m` was the i-th message sent by the train, and the event `DataSent‘i’(m)` to mean that message `m` was the i-th message received by the RBC. We can check if an attacker can insert a message into the communication phase of the protocol by checking that all message `m` received by the RBC where send by the train either as its first, second or third message:

```
ev:DataReceived1(m) ==>
 (ev:DataSent1(s2, m) | ev:DataSent2(s2, m) | ev:DataSent3(s2, m))
ev:DataReceived2(m) ==>
 (ev:DataSent1(s2, m) | ev:DataSent2(s2, m) | ev:DataSent3(s2, m))
ev:DataReceived3(m) ==>
 (ev:DataSent1(s2, m) | ev:DataSent2(s2, m) | ev:DataSent3(s2, m))
```

This holds, showing that the attacker cannot insert their own messages.

Ability to Replay Messages The above correspondence assertions show that an attacker cannot insert their own messages, but they may still be able to replay an old message, tricking the receiver into thinking it is fresh. We test for replay attacks with a similar correspondence assertion, but this time we require the correspondence to be injective, i.e., for each receive event there must exist a *single, unique* send event:

```
evinj:DataReceived1(m) ==>
 (evinj:DataSent1(s,m)|evinj:DataSent2(s,m)|evinj:DataSent3(s,m))
evinj:DataReceived2(m) ==>
 (evinj:DataSent1(s,m)|evinj:DataSent2(s,m)|evinj:DataSent3(s,m))
evinj:DataReceived3(m) ==>
 (evinj:DataSent1(s,m)|evinj:DataSent2(s,m)|evinj:DataSent3(s,m))
```

These correspondence all hold showing that the attacker cannot replay messages.

Ability to Reorder Messages Another way in which an attacker could interfere with the communication would be to reorder the message, for instance causing disruption by swapping the order of a go and stop message. As this would not require additional messages, or replaying a message, it would not be detected by the two correspondence assertions above.

The `MessagesReceived3(m1, m2, m3)` event indicates that the messages `m1, m2, m3` were received in that order. We check reordering using this event, and an injective correspondence assertion on the order of the three messages sent by the train:

```
evinj:MessagesReceived3(m1, m2, m3) ==>
(evinj:DataSent1(s,m1)&evinj:DataSent2(s,m2)&evinj:DataSent3(s,m3))
```

We find that this correspondence assertion holds. In our model the attacker may also block messages, therefore even though this correspondence assertion holds it may still be possible for an attacker to block one message and reorder the other two (so meaning that the `MessagesReceived3(m1, m2, m3)` event is never reached. Therefore, we also check the possible reordering of two messages:

```
evinj:MessagesReceived2(m1, m2) ==>
  ((evinj:DataSent1(s, m1) & evinj:DataSent2(s, m2)) |
   (evinj:DataSent1(s, m1) & evinj:DataSent3(s, m2)) |
   (evinj:DataSent2(s, m1) & evinj:DataSent3(s, m2)))
```

This correspondence assertion also holds showing that reordering is not possible.

Ability to Delete Messages without the Receiver Knowing While the attacker can stop any message from being delivered we would like the protocol to allow this to be detected. For example, the receiver should not accept a message if the message sent before it did not arrive. We can check this with the following correspondence assertions:

```
evinj:DataReceived1(m) ==> evinj:DataSent1(s, m)
evinj:DataReceived2(m) ==> evinj:DataSent2(s, m)
evinj:DataReceived3(m) ==> evinj:DataSent3(s, m)
```

These correspondence assertions checks to see if deletion *or* reordering is possible, but as we have already shown that reordering is not possible this correspondence assertion will only hold if deletion is impossible and only fail if messages can be deleted.

We find that these correspondence assertions fail to hold, in particular, as the counter-style timestamp can be any value greater than the previous message. There is no simple method for the receiver to detect the absence of a message, however, this can be partly mitigated by acknowledgements messages and timeouts, as we discuss in the next section.

Analysis of Emergency Messages As described earlier, the application level protocol does not use MACs to verify the emergency stop messages. To see what effect this has, we run each of the test described above on our second model, which includes the sending of messages with no accompanying MAC. We find that, as before, the secrecy of keys and authentication and agreement on the key hold. However, message insertion, deletion, reordering and replay fail to hold. This means that the attacker still cannot pretend to be a train or a RBC, and it is still only possible to set up a communication between a genuine train and RBC. However, once such a session has been set up it is possible for the attacker to insert a stop message, which will be accepted by the train. We discuss the relevance of this finding below.

5 Discussion and Recommendations

In this section, we present recommendations regarding the different issues that were discovered in our analysis.

5.1 Inserting High-priority Messages

Our analysis showed it is possible to insert messages with high-priority as there is no protection provided over these messages. Therefore, anyone with access to the EuroRadio communication layer can insert emergency stop messages and trigger a train to brake. Though this might not directly lead to incidents with trains colliding, it can cause serious disruptions, for example, due to displaced crew and rolling stock. These disruptions can have a higher impact on the network if the emergency stop is carefully timed. For example, this happens when a train is in a GSM-R radio hole with no reception. In this case, the RBC will not know what has happened as it will not be able to communicate with the train and therefore will not be able to cancel the emergency stop. The driver of the train will need to follow special procedures until GSM-R coverage is available again. It would then take even longer than usual to recover from the emergency stop, which could seriously affect other traffic in the system as well.

To prevent unauthorised emergency stop messages from taking effect, high-priority messages should be authenticated using MACs as is the case with regular priority messages. They can still be given priority over the other messages when checking the MAC. A concern might be that keys could become corrupted, in which case it should still be possible to fall back to voice communication (as is used in most current systems). The application of a MAC to the high-priority message would prevent misuse by an external actor by stopping them from being able to successfully inject messages in the communication between a train and RBC. Although, of course, an attacker could still cause disruption by other means, such as jamming signals.

5.2 Deletion of Messages

The EuroRadio protocol does not protect against deletion of messages. This needs to be taken care of by the application layer. The timestamps that are added by the application layer do not protect against this. The sender of a message can request an acknowledgement for the message from the recipient. This is not the default though and needs to be done explicitly. Moreover, recipients have no way to determine whether it had not successfully received messages. In the worst case, an attacker could prevent reception (i.e. delete) emergency stop messages, after which a train might enter a danger point, a stretch of track, where the safety of the train may be compromised.

Though it is hard to prevent deletion of messages as an attacker could jam all communication between two parties, it is possible to detect the deletion of single messages. A simple way to do this is by adding a counter to all messages. If this counter skips between two messages, you know a message was missed. This would require changes to the current specifications to change both the message format and add procedures what to do in case a missed message is detected.

The specification already has a measure that can help in the case of a jamming attack. It is possible to let a train make an emergency stop if no messages are received within a specific timeout period. This timeout and the action to be taken if it expires are set using nationally set parameters, respectively `T_NVCONTACT` and `M_NVCONTACT`. The possible actions to take are to trigger the normal brakes, trip the onboard systems, including an immediate application of the emergency brakes or perform no action. The default value, as set out in SUBSET-026 of the ERTMS specifications are set to 'no reaction' with an infinite amount of time specified, i.e. there is no timeout, for safe messages to be received. Using this measure might result in problems with GSM-R black spots, i.e. if a train spends too much time within a black spot the brakes would be automatically triggered. However, the standard provides ways to inform the train of GSM-R black spots and therefore this should not be a problem.

5.3 Disagreement over RBC Identity and Safety Feature

One issue identified that is not specifically covered in the EuroRadio specifications is that of ensuring that when a train commences a EuroRadio session, the RBC that it establishes the session with is not only genuine, but also the correct one to handle the train. When a train tries to set up a session it doesn't always know the identity of the RBC it will be talking to. In this case the traffic could be redirected to another RBC. At the 'start of mission', the train may invalidate the RBC ID and phone number, for example, if it is recovered, in the case of the train breaking down, or it loses state following a system reboot. The specifications [9] allow the last RBC ID and number to be reused, however they allow the use of the EIRENE shortcode to use location-based addressing to contact the most appropriate RBC for the area the train is connected to via GSM-R. Finally, the driver may alternatively enter the number manually. The latter two

options allow the connection to an RBC which is not directly in the area that the train should connect to.

When the train does know the identity of the RBC it wants to communicate with, the standard does not specify what needs to be done if the expected identity is different than the one received during the authentication protocol. It is not even specified whether this should be checked. To make things less ambiguous, we recommend to explicitly include in the protocol description that the RBC identity needs to be checked, if known, and the connection should be aborted if this check fails.

A similar issue involves the safety feature that is used to indicate which MAC algorithm is to be used. The initiator of the protocol chooses a safety feature and sends it to the recipient in the first message, after which the recipient returns it in the second message. The standard does not specify what to do if the safety features do not match. In the official specification, only one safety feature is currently supported, but for future versions, where different safety features might be supported, it is crucial to add this. It should be enforced that the selected safety feature is either equal to or more secure than what was sent by the initiator.

6 Conclusions

We have presented a security analysis of ERTMS's EuroRadio protocol and parts of the application layer protocol. To do this, we developed a novel representation of counter-style timestamps, and new correspondence assertions to test for message insertion, deletion, reordering and replay. We found that EuroRadio defends the security of its key and authenticates the parties involved against an active Dolev-Yao attacker. However, it failed in some of the additional properties we would liked to have seen, such as message deletion and insertion of emergency messages. We discussed the relevance of these findings in the previous section. Our results on messages are tested for the train sending three messages to the RBC, as future work we would like to find a way of testing these results for an arbitrary number of messages sent in either direction, and any interleaving of normal and high-priority messages. While our analyses finds that the protocols do not protect from the insertion of high-priority emergency messages, inserting packets into a GSM-R data stream may be difficult and merits further investigation. Our analysis also makes the assumption that the cryptographic primitives used in ERTMS are secure, as future work we would like to examine these primitives and test this belief.

Acknowledgements: We would like to thank Maria Vigliotti and Florent Pepin from the UK's Rail Safety and Standards Board (RSSB) for helpful discussion regarding the security of ERTMS. Funding for this paper was provided by the UK's Centre for the Protection of National Infrastructure (CPNI) and Engineering and Physical Sciences Research Council (EPSRC) via the SCEPTICS: A SystematiC Evaluation Process for Threats to Industrial Control Systems project.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Symposium on Principles of Programming Languages (POPL)*, 2001.
2. Ansaldo STS Group. Product portfolio and ERTMS/RTCS projects of Ansaldo Segnalamento Ferroviario, 2008. `http://old.fel.zcu.cz/Data/documents/sem_de_2008/AnsaldoSTS_08.pdf`.
3. M. Arapinis, T. Chothia, E. Ritter, and M. Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010*, pages 107–121, 2010.
4. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Computer Security Foundations Workshop (CSFW)*, pages 82–96. IEEE, 2001.
5. B. Blanchet, B. Smyth, and V. Cheval. ProVerif 1.88: Automatic cryptographic protocol verifier, user manual and tutorial, 2013.
6. R. Bloomfield, R. Bloomfield, I. Gashi, and R. Stroud. How secure is ERTMS? In F. Ortmeier and P. Daniel, editors, *Computer Safety, Reliability, and Security*, volume 7613 of *Lecture Notes in Computer Science*, pages 247–258. Springer Berlin Heidelberg, 2012.
7. V. Cheval and V. Cortier. Timing attacks in security protocols: symbolic framework and proof techniques. In *Proceedings of the 4th Conference on Principles of Security and Trust (POST'15)*, Lecture Notes in Computer Science, 2015.
8. T. Chothia, F. Garcia, J. de Ruiter, J. van den Breekel, and M. Thompson. Relay cost bounding for contactless EMV payments. In R. Bhme and T. Okamoto, editors, *Financial Cryptography and Data Security*, volume 8975 of *Lecture Notes in Computer Science*, pages 189–206. Springer Berlin Heidelberg, 2015.
9. ERA. SUBSET-026: System requirements specification, version 3.5.0. Technical report, 2015.
10. R. Esposito, A. Lazzaro, P. Marmo, and A. Sanseviero. Formal verification of ERTMS EuroRadio safety critical protocol. *Proceedings 4th symposium on Formal Methods for Railway Operation and Control Systems (FORMS'03)*, 2003.
11. M. Franekova, K. Rastocny, A. Janota, and P. Chrtiansky. Safety Analysis of Cryptography Mechanisms used in GSM for Railway. *International Journal of Engineering*, 11(1):207–212, 2011. `http://annals.fih.upt.ro/pdf-full/2011/ANNALS-2011-1-34.pdf`.
12. GSM-R Functional Group. EIRENE Functional Requirements Specification, version 7.4.0. Technical report, 2014.
13. GSM-R Functional Group. EIRENE System Requirements Specification, version 15.4.0. Technical report, 2014.
14. L. Hongjie, C. Lijie, and N. Bin. Petrinet based analysis of the safety communication protocol. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 11(10):6034–6041, 2013.
15. L. Li, J. Sun, Y. Liu, M. Sun, and J. S. Dong. A formal specification and verification framework for timed security protocols. In *TSE in Submission*, 2015.
16. RSSB. GSM-R User Procedures, issue 7.1. Technical report, 2015.
17. UNISIG. SUBSET-037 - EuroRadio FIS, version 3.2.0. Technical report, 2015.
18. Y. Zhang, T. Tang, K. Li, J. M. Mera, L. Zhu, L. Zhao, and T. Xu. Formal verification of safety protocol in train control system. *Science China Technological Sciences*, 54(11):3078–3090, 2011.