An Attack Against Message Authentication in the ERTMS Train to Trackside Communication Protocols

Tom Chothia
University of Birmingham
Birmingham, UK
t.p.chothia@cs.bham.ac.uk

Mihai Ordean
University of Birmingham
Birmingham, UK
m.ordean@cs.bham.ac.uk

Joeri de Ruiter
Radboud University
Nijmegen, NL
joeri@cs.ru.nl

Richard J. Thomas
University of Birmingham
Birmingham, UK
r.j.thomas@cs.bham.ac.uk

ABSTRACT

This paper presents the results of a cryptographic analysis of the protocols used by the European Rail Traffic Management System (ERTMS). A stack of three protocols secures the communication between trains and trackside equipment; encrypted radio communication is provided by the GSM-R protocol, on top of this the EuroRadio protocol provides authentication for a train control application-level protocol. We present an attack which exploits weaknesses in all three protocols: GSM-R has the same well known weaknesses as the GSM protocol, and we present a new collision attack against the EuroRadio protocol. Combined with design weaknesses in the application-level protocol, these vulnerabilities allow an attacker, who observes a MAC collision, to forge train control messages. We demonstrate this attack with a proof of concept using train control messages we have generated ourselves. Currently, ERTMS is only used to send small amounts of data for short sessions, therefore this attack does not present an immediate danger. However, if EuroRadio was to be used to transfer larger amounts of data trains would become vulnerable to this attack. Additionally, we calculate that, under reasonable assumptions, an attacker who could monitor all backend control centres in a country the size of the UK for 45 days would have a 1% chance of being able to take control of a train.

1. INTRODUCTION

The European Rail Traffic Management System (ERTMS) Standard provides a suite of protocols used to deliver next-generation train management and signalling.[1] This standard is designed with the intention to enable trains to interoperate across borders and optimise the running operation of railways. At present, the system is being rolled out across Europe and also on high-speed lines around the world.

ERTMS is formed of three core communication layers: GSM-R, EuroRadio and the Application Layer protocol (see Figure 1). The EuroRadio and the Application Layer protocols form ETCS, the European Train Control System. The lowest layer of the stack, GSM-R, is a rail-specific variant of the GSM protocol, used for communications between the train and trackside infrastructure. EuroRadio, the middle layer, provides authentication and integrity of messages sent between the train and track side components using cryptographic MACs. The Application Layer protocol is the highest layer of the stack; this is a stateful protocol that includes timestamps and message acknowledgements to prevent the replay of messages and ensure successful communication.

In this paper, we present the results of a cryptographic analysis of the ERTMS communication stack, in which we determined whether arbitrary, unauthorised messages can be sent to trains or trackside equipment. We show that one such attack is possible and give details on the exact circumstances which would allow it to happen. We also propose several solutions to mitigate this vulnerability.

The MAC algorithm used in EuroRadio is a modified version of the ISO 9797-1 MAC Algorithm 3 [13, 23], a standard which was introduced in 2011. The ISO algorithm is a CBC-MAC that uses a single DES transformation for all but the last block, which is encrypted using triple DES (3DES) using two different keys. The use of only two keys for the 3DES operation is a potential weakness, so EuroRadio uses triple DES (3DES), with 3 distinct keys, for this final block.

As with any 64-bit MAC, it is possible for collisions to occur, i.e., two different messages may have the same MAC for a particular key. Such a collision is unlikely (requiring 2^{29} messages for a 1% chance) however an attacker that can wait long enough will eventually observe one. A well designed protocol should not be vulnerable to an attacker that observes colliding MACs. However, we show in Section 4 that in the case of EuroRadio that such a collision can be used to retrieve the first of the three DES keys using brute-force.

Establishing one of the keys used by the MAC should not pose an immediate threat to the integrity of the protocol, as the final transformation when generating a MAC involves a 3DES encryption with three distinct keys. An attacker, therefore, cannot simply generate a valid MAC

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

AsiaCCS '17, April 02 - 06, 2017, Abu Dhabi, United Arab Emirates.

ISBN 978-1-4503-4944-4/17/04…$15.00.

DOI: http://dx.doi.org/10.1145/3052973.3053027

for any ERTMS message to send to a train. However, the ERTMS specification allows a number of messages to be sent with optional packets. By using these optional packets, we show that it is possible to carefully craft a forged message with a valid MAC, e.g. a Movement Authority allowing a train to proceed further than it was safe to proceed, which would be accepted by a train.

Taking the UK train network as an example we look at the likelihood of a collision to occur, required to perform the attack. We find that due to current data speeds and session lengths, this is very unlikely for a single train. However, if EuroRadio was used to transfer larger amounts of data (for instance, transmitting train diagnostic data) it could be more likely broken. We go on to consider an attacker that is capable of monitoring the entire ERTMS backbone of a country the size of the UK and find that, when monitoring the network for 45 days, an attacker has a 1% chance of observing the collision needed to be able to take control of a train. We present a number of mitigations, both short-term and long-term based, against the presented attack.

In summary, our main contributions are as follows:

- **Security analysis of cryptography used in ERTMS.** We perform a security analysis of the cryptography implemented in each layer of the ERTMS standard (GSM-R, EuroRadio and Application Layer protocol).
- **A message forging attack against the EuroRadio protocol.** We implement an attack whereby a message can be forged to match a valid MAC by exploiting several limitations of the EuroRadio MAC algorithm, and weaknesses in the Application Layer protocol.
- **Mitigation and fixes.** We assess potential mitigations and fixes to the attack discovered, and propose alternative MAC algorithms suitable for use in EuroRadio.

In the next section, we provide an overview of the ERTMS system at a high level, and in Section 4, we describe how it is possible to retrieve one of the keys used in the EuroRadio MAC algorithm. In Section 5, we show how it is possible to create valid MACs for train control messages by an unauthorised party. The implementation of the MAC collision and DES key recovery methods is presented in Section 6. We then discuss the time needed to find the collisions required for our attack in Section 7, discussing fixes to the underlying protocols, analysis of their collision resistance and performance impact in Section 8, concluding in Section 9.

Related work. The EuroRadio MAC is similar to ISO 9797-1 MAC Algorithm 3 [13]. Mitchell [17] and Preneel et al. [21, 12] have previously analysed the ISO 9797-1 MAC Algorithm 3 design and found collision-based key recovery attacks, requiring $2^{32.5}$ known plaintext-MAC pairs and $3 \cdot 2^{56}$ offline operations. The EuroRadio MAC replaces the use of two key 3DES in ISO 9797-1 MAC Algorithm 3 with the use of three distinct keys, so these attacks are no longer feasible. Franekova et al. [8, 9], discuss similar attacks against the EuroRadio MAC, however the use of 3DES means that the attacks they discuss will not work in practice.

Pépin and Vigliotti [18] discuss the cryptographic mechanisms used in ERTMS key distribution, consider standard attacks against 3DES, such as a *related-key attack* and quantify the cost and resources required to break 3DES, which currently is not practical. As far as we are aware, our paper presents the first feasible attack against the cryptography used by ERTMS.

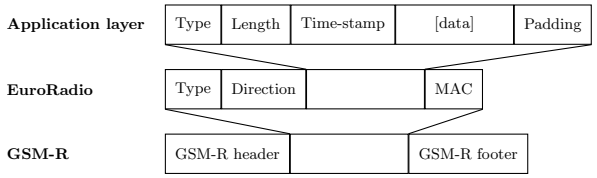


Figure 1: Communication layers in ERTMS. The Application layer contains the train signalling protocol with ETCS Application Messages. This protocol is encapsulated in EuroRadio, which provides authentication and integrity verification for the application layer messages through a MAC. EuroRadio is finally encapsulated in a GSM-R transport protocol which provides encryption through the A5/1 GSM cipher.

At a more abstract level, De Ruiter et al. [6] present a formal analysis of the security of the EuroRadio key establishment protocols, however they abstract away from the cryptographic details. Bloomfield et al. [3] provide a high-level security analysis of ERTMS, but did not find the vulnerability in the EuroRadio MAC presented in this paper.

2. ERTMS OVERVIEW

In this section, we present a high-level overview of the communication systems in ERTMS, focussing on some of the more complex components and provide context to the functioning of ERTMS.

Within ERTMS, two important components are *Radio Block Centres* (RBCs) and *balises*.

RBCs are entities which are responsible for trains in a specific area. They take care of the sending and receiving of signalling and train control information to and from trains under their control. A single RBC is typically responsible for a geographical radius of approximately 70 kilometres[2]. 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 control. Trains are authorised to operate on particular sections of track using Movement Authority (MA) commands. These are Application Layer messages which contain relevant information sent from the RBCs to the trains. They include details such as, for example, the maximum speed a train may operate at and the distance that the train is allowed to move under this command.

The RBC determines the positions of the trains in its area through position reports sent by the trains. Based on this information, signalling commands are provided, if the track ahead is clear. The train determines its position through *balises*, RFID-like units which are embedded into the trackbed, typically in pairs. As a train passes over a 'balise group', it may then determine the direction of travel, and relevant data is emitted to the train, for example, line speed limits, distance to the next balise group, balise group

[2]http://old.fel.zcu.cz/Data/documents/sem_de_2008/AnsaldoSTS_08.pdf

ID and other appropriate operational information such as track gradient and profile.

Communication between the various components of the trackside infrastructure is specified in the ERTMS standard. However, it should be noted that ERTMS, whilst interoperable, does not have a requirement for open operation within the signalling subsystems, which may be proprietary.

Currently, there are three operational levels to ERTMS:

- *Level 1*, is simply an overlay to the existing national signalling systems. Balises may optionally be used to provide movement authorities to trains operating on equipped lines. Lineside signals at this level remain mandatory, dependent on the national system in place.
- *Level 2* removes the need for lineside signals, where balises at this level provide only static data, including position data. Movement authorities are provided to the train over GSM-R by the trackside RBC. Lineside signals may optionally be retained, however, are not mandatory.
- *Level 3* uses a similar principle to *Level 2*, however integrity of safe operation is solely controlled through onboard means (i.e. track-based detection equipment is no longer required), therefore, allowing moving block operation of the trains, to increase utilisation of previously constrained capacity on the rail network.

2.1 GSM-R

GSM-R is the lowest level communication protocol used in ERTMS between the train and trackside [11, 10]. However, it is not used for communications between trains and balises. Based on the GSM Mobile Communications Standard[3], GSM-R uses different frequency ranges based on national spectrum availability, and provides additional rail-specific functionality. The additional functionality includes support for multi-party communication between drivers, emergency calling functionality, and priority-based pre-emption (i.e. active calls may be terminated if a higher priority call is received). ERTMS command and control messages are sent at the highest priority in GSM-R, and cannot be pre-empted. In comparison to the cell-based GSM standard, GSM-R uses an alternative network cell layout, where base stations are located along, or near railway lines, and an overlap is provided to ensure redundancy should one cell fail.

In the United Kingdom, a nationwide rollout of GSM-R for voice was completed by the Infrastructure Manager, Network Rail in 2014 to replace the outdated Cab Secure Radio system, with GSM-R data support rolled out on the Cambrian line in 2010 during the rollout of ERTMS on the line.

Cryptography used in GSM-R. GSM-R uses the A5 suite of cryptographic ciphers, more specifically A5/1, a stream cipher based on Linear Feedback Shift Registers (LFSRs). Optionally, the block cipher, A5/3, may be supported. These ciphers are used for encryption of the communication between mobile stations (e.g. trains and handheld devices) and a GSM-R Base Station, providing confidentiality during transmission. Mobile handsets and devices are authenticated onto the network, however, the base station is never authenticated to the handset.

[3]http://www.etsi.org/technologies-clusters/technologies/mobile/gsm

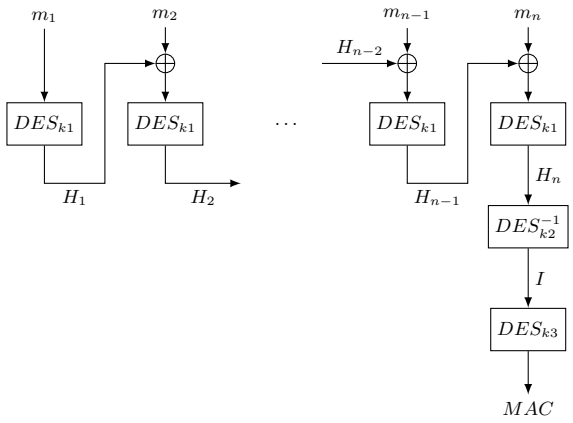


Figure 2: The MAC algorithm used by EuroRadio is comprised of $n-1$ rounds of DES followed by a final round of 3DES.

2.2 EuroRadio

The EuroRadio protocol is a middle layer protocol that sits between the GSM-R and Application Layer protocols, providing authentication and integrity verification for messages sent and received between the train and backend. EuroRadio's main purpose is to provide guarantees that messages received are authentic, by adding cryptographic MACs. It relies on GSM-R to provide encryption between the train and mobile base station. EuroRadio includes provisions for an algorithm negotiation phase during the handshake, however, currently there is only one option for this.

The message authentication algorithm used is based on the ISO 9797-1 MAC Algorithm 3 (also known as ANSI X9.19 Optional Procedure 1) [23, 13, 1]. This algorithm computes a message's MAC using a combination of the DES and 3DES ciphers by dividing a message into n blocks of 64 bits, padded appropriately, which are fed into a CBC circuit which uses DES for the first $(n-1)$ blocks of a EuroRadio payload and 3DES for the final block of this payload (see Figure 2). The MAC is computed on the length of the entire message, the ETCS ID of the recipient, the message being sent and optional padding to ensure appropriate length for the MAC computation [23]. The prefix consists of the message length and recipient ETCS ID, which is 40 bits long.

The MAC algorithm used in EuroRadio uses three distinct keys in the final 3DES computation, whereas the ISO-specified algorithm uses only two keys in the final round. EuroRadio uses *Padding Method 1*, specified in ISO 9797: 0s are used as padding such that the size of the data becomes a multiple of the block size. If the data is already a multiple of the block size, no padding is added.

The keys used by EuroRadio to generate the MACs are *session keys*, derived from a pre-shared 3DES key and nonces exchanged between the parties during the protocol handshake phase.

In addition to ensuring validity and integrity of messages sent to and from a train and backend equipment, this layer offers the ability to set the priority of messages to either 'normal' or 'high'. All messages which have a 'normal' priority set, must have a MAC computed and added to the payload. 'High' priority messages do not contain a MAC and bypass verification by the EuroRadio layer but can only be used to send a very restricted set of commands, such as emergency stop messages.

2.3 Application layer protocol

Once mutual party authentication in EuroRadio has completed, the train and trackside equipment can start to exchange data using the Application Layer protocol. This protocol is used to specify messages used for communication between train and trackside. A number of message types are defined in the ERTMS Standard [7].

Messages contain a header, which may be dependent on whether the message was generated by a train or trackside equipment. Messages will additionally contain a *timestamp*, indicating the time at which the message was sent. This time, however, is relative to the train (i.e. a counter) and is not represented by a universally-set time source, e.g. UTC. After authentication, the trackside equipment will synchronise its clock upon receipt of the first message from the train. The EuroRadio and Application Layer protocols have previously been formally analysed [6], where it was shown that while both layers prevent an attacker from learning the secret key, there are flaws in the protocol, for instance allowing unauthenticated 'Unconditional Emergency Stop' messages being sent by an attacker. There is no existing work to our knowledge, however, which attempts to fuzz the EuroRadio or Application Layer protocols at this time.

Typically, messages sent from train to trackside have the following format:

msg type	msg length	time-stamp	train ID	pos. report	other (opt.)	padding (opt.)

Similarly, a message sent from the trackside to the train will typically have this format:

msg type	msg length	time-stamp	ack.	LRBG ID	other (opt.)	padding (opt.)

Both messages contain their respective type, length and relative timestamp. Messages sent by the train contain a position report and, dependent on the message being sent, additional data is formatted in packets, as prescribed in [7]. The position report contains the distance and direction travelled respective to the Last Relevant Balise Group (LRBG), an upper and lower interval based on some tolerance for the odometer onboard, and the LRBG's location accuracy, which is part of data received from the balises. To ensure the message has a whole byte length, the message may be optionally padded with '0'.

Emergency messages.

Messages may carry a priority marker, where emergency messages, set as 'high' priority, are treated differently to other types of message. As a result, most of the integrity and authentication verification that takes place within EuroRadio is bypassed where the message may be immediately passed up to the Application Layer. This may only take place once an authenticated session is established. Currently, only two message types may be sent with 'high' priority: *unconditional emergency stop* and *conditional emergency stop* [7]. All emergency stop messages must be acknowledged by the train, where a different message type is used, rather than the standard acknowledgement message. Acknowledgements and revocations for emergency stops, however, are not sent as 'high' priority data, and must satisfy any verification that takes place in the EuroRadio layer.

3. EURORADIO ATTACK ROADMAP

This section will provide an outline of our attack on ERTMS to forge train control messages, where we present known vulnerabilities and show how the attack can be deployed. Whilst the attack mainly relies on breaking the cryptographic implementation of the MAC in EuroRadio, we might also need to circumvent the security of GSM-R. This can easily be done using *commercial off-the-shelf* (COTS) equipment, for example the $100 HackRF and NVIDIA GTX Titan X GPU for $1,400.

The stages of the attack presented in this paper are:

- Obtain unencrypted GSM-R traffic
- Observe a collision in the EuroRadio MAC
- Recover the first of the three 3DES keys, k_1
- Forge an Movement Authority message and send it

Obtaining and Decrypting GSM-R traffic. The GSM-R protocol uses the same cryptographic mechanisms as GSM to protect messages from tampering and eavesdropping. Therefore, any vulnerability that exists against the current GSM standard may be exploitable within GSM-R as well. A channel jamming attack using COTS equipment would affect all operations within range of the equipment for ERTMS. A more sophisticated attack would be to intercept communications in order to eavesdrop, or, insert messages into the communications channel. Inserted messages could cause a train to stop for an unpredictable amount of time to something more significant, for example, allowing a train to enter an unsafe situation e.g. allowing two trains to occupy the same section of track. Messages sent over GSM only provide authentication that the communicating party also possesses the A5 key being used for encryption, and not that the message being conveyed is actually from a genuine, honest entity.

The weakness of GSM encryption has been heavily evaluated, where recently, it was shown that the A5/1 cipher used in both GSM and GSM-R could be broken in as little as 9 seconds using commodity hardware [16]. For this a time-memory trade-off attack was used, with a probability for success of 81%. The setup time for this attack, however, was approximately two months, producing tables that may be reused over different attacks. An alternative is the use of rainbow tables, used by Nohl and others [15, 22, 14], requiring approximately 1.6TB of storage capacity.

An export variant of A5/1, A5/2, was also shown to be weak and is vulnerable to real-time attacks, which can break the cipher in less than a second [2, 19]. Barkan et al. [2] describe a method to perform a man-in-the-middle attack, which forces the GSM protocol to fall back to the A5/2 cipher, allowing real-time attacks to be carried out. As the key used for the A5/2 cipher is usually the same as the one used by the A5/1 cipher, this provides a means to attack the A5/1 cipher. Once the A5 key has been established, it is then possible to recover traffic sent over the GSM-R link, including train control messages.

Another form of attack is through so-called 'IMSI Catchers' [4], where the mobile station is tricked into connecting to a dishonest base station in two phases. The first phase is to obtain the TMSI of the victim train, known as 'Identification'. Whilst train operating timetables and (in the United Kingdom) train positions are publicly available, the TSMI and some ERTMS values (for example train ID) are not publicly documented. In the second phase, 'Camping',

the attacker captures traffic between the train and trackside infrastructure where the A5/1 key is then obtained.

The attacker then controls all traffic through to and from the mobile station.

Recovering k_1 from the EuroRadio MAC algorithm. Assuming that the attacker has access to the communication between the train and the backend, it is now a matter of observing two messages which have the same MAC at the EuroRadio layer.

The current EuroRadio MAC generation and verification process depends on the DES and 3DES encryption ciphers. The way the DES cipher is used in the EuroRadio MAC algorithm makes it likely for collisions to occur due to its small block sizes (i.e. 64 bits) [20].

In a well designed protocol, the existence of a collision would not lead to a vulnerability, however we show in Section 4 that flaws in the MAC combined with the short key-length of DES, makes it susceptible to successful brute-force attacks on k_1.

Forging Movement Authority messages. Finding just one of the three DES keys would not normally lead to an attack against a secure cryptographic system, however we show in Section 5 that additional flaws in EuroRadio allow us to create a forged message that has the same MAC as an observed valid message, using just k_1.

ERTMS messages from trackside to the train can include optional packets, where one of these can be used to send a text message, which is displayed to the driver. This packet also specifies the conditions under which the message should be displayed. This means that random bytes may be included in a message. An attacker may take advantage of this to construct a properly-formatted application layer message which would be accepted by the train. Additionally, this message can be designed by the attacker such that it will not be displayed, thus, the driver is not aware of the message having been received by the train.

4. EURORADIO KEY RECOVERY

In this section we will present a method to recover k_1, one of the three keys used by the MAC algorithm in the EuroRadio protocol. As stated before, our attack leverages the small block size of the DES and 3DES ciphers and relies on the observation of a MAC collision. In Section 5, we will go on to show how this key, together with a weakness in the Application Layer protocol, can be used to forge valid EuroRadio messages. Our method of recovering k_1 exploits two limitations: (1) if two messages have the same EuroRadio MAC then the input into the final 3DES block is identical; and (2) only k_1 is used for the single DES rounds and can be brute-forced.

As described in Section 3, EuroRadio uses a combination of the DES and 3DES ciphers, in a mode similar to CBC, to compute the MAC for an Application Layer protocol message m as follows (see also Figure 2). First, the message m is split into 64 bit blocks, $m = (m_1, \ldots, m_n)$. If the last block m_n is not 64 bits long it is padded with *0*s. Then, for each block $m_i,\ i \in \{1, \ldots, n-1\}$, H_i is computed as:

$$H_i = DES_{k_1}(H_{i-1} \oplus m_i) \tag{1}$$

where $H_0 = 0$ and $H_{i-1} \oplus m_i$ is the result of the XOR operation on H_{i-1} and m_i. The final MAC of the message

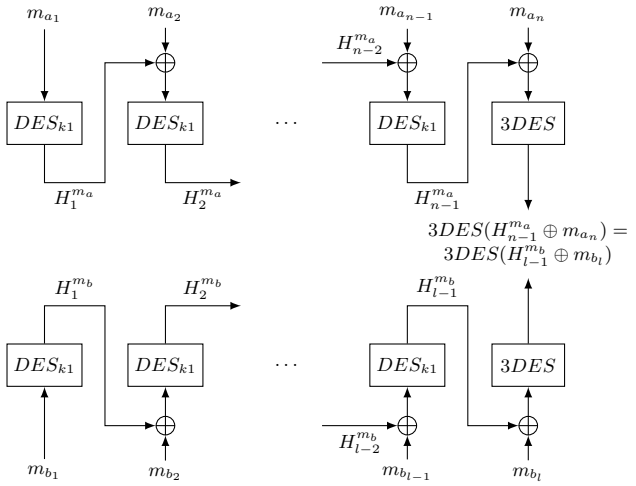


Figure 3: Recovering the DES k_1 key using message collisions.

is the result of the final 3DES encryption:

$$MAC(m) = 3DES_{k_1,k_2,k_3}(H_{n-1} \oplus m_n) \tag{2}$$

The likelihood of at least one MAC collision occurring between two messages out of M observed (message, MAC) pairs for N possible MAC values, is [5]:

$$\begin{aligned} P &= 1 - \prod_{i=1}^{M-1} \left(1 - \frac{i}{N}\right) \\ &\approx 1 - e^{-M(M-1)/(2N)} \\ &\approx \left(1 - \frac{1}{N}\right)^{M(M-1)/2} \end{aligned} \tag{3}$$

In the case of DES and 3DES $N = 2^{64}$.

We argue that, for rail infrastructure, a 1% probability of compromise is significant and using Equation (3) the number of (message, MAC) pairs needed to detect a collision with a probability of 1% is $M = 6.1 \times 10^8$. We discuss the feasibility of collecting this data in Section 7.

Once a collision is found between two messages m_a and m_b, consisting of n and l blocks respectively, we know that the input to the 3DES functions is identical for both messages (see Figure 3), namely:

$$H_{n-1}^{m_a} \oplus m_{a_n} = H_{l-1}^{m_b} \oplus m_{b_l} \tag{4}$$

We can use this information to perform a brute-force search for k_1. Both initial input messages m_a and m_b are known, so we can compute $H_{n-1}^{m_a}$ and $H_{l-1}^{m_b}$ for every possible key k_1'. If we find k_1' for which Eq. (4) holds then we have a possible candidate for key k_1. We discuss the time it takes to perform this brute-force attack in Section 6.

Example. For $n = l = 3$, Eq. (4) becomes $H_2^{m_a} \oplus m_{a_3} = H_2^{m_b} \oplus m_{b_3}$. We can expand this to $DES_{k_1}(H_1^{m_a} \oplus m_{a_2}) \oplus m_{a_3} = DES_{k_1}(H_1^{m_b} \oplus m_{b_2}) \oplus m_{b_3}$. The final expansion is:

$$\begin{aligned} &DES_{k_1}(DES_{k_1}(m_{a_1}) \oplus m_{a_2}) \oplus m_{a_2} \\ &= DES_{k_1}(DES_{k_1}(m_{b_1}) \oplus m_{b_2}) \oplus m_{b_3} \end{aligned}$$

As we know $m_{a_1}, m_{a_2}, m_{a_3}, m_{b_1}, m_{b_2}$ and m_{b_3} the only un-

Variable	Length (bits)	Description	Example
Message 146 (Acknowledgement)			
NID_MESSAGE	8	Message type	146 (Acknowledgement)
L_MESSAGE	10	Length of message (bytes)	10
T_TRAIN	32	Train timestamp	53088208
NID_ENGINE	24	Train ID	1
T_TRAIN	32	Timestamp being acknowledged	53088178

Table 1: Example of an Acknowledgement message sent by a train

known in this equality is k_1, therefore we may use this to find the key with a brute force guessing attack.

5. FORGING TRAIN CONTROL MESSAGES

In this section, we present our Application Layer message forging attack. The attack relies on several weaknesses in the ERTMS stack, allowing us to create Application Layer messages which have MACs identical to the ones of previously observed valid messages. Our attack does not require complete knowledge of all three keys used in the EuroRadio MAC algorithm, but only requires key k_1, used for the single DES cipher, to be known. In Section 4, we described how k_1 can be recovered when a MAC collision is observed.

For this attack, we assume that we have seen a valid message $m_v = (m_{v_1}, \ldots, m_{v_n})$ and the corresponding MAC, and we have obtained the key k_1. First, we show how a different message can be forged that has the same MAC as m_v.

Assuming we want to get a valid MAC for the forged message $m_f = (m_{f_1}, \ldots, m_{f_l}) \neq m_v$. Based on Equation (2) $MAC(m_f)$ will be identical to $MAC(m_v)$ only if:

$$H_{n-1}^{m_v} \oplus m_{v_n} = H_{l-1}^{m_f} \oplus m_{f_l} \tag{5}$$

Using k_1 we can compute valid intermediate MACs H for any message up to the last-but-one block (using Equation (1)), so we can compute $H_{n-1}^{m_v}$ and $H_{l-1}^{m_f}$. We also have knowledge of m_{v_n} as part of the observed message. Therefore we can compute $H_{n-1}^{m_v} \oplus m_{v_n}$ and $H_{l-1}^{m_f} \oplus m_{f_l}$, however, with a very high probability these will not be equal (Figure 4(a)).

In order to get a valid MAC for the message m_f we extend it with one additional block $m_{f_{l+1}}$ (see Figure 4(b)):

$$m_{f_{l+1}} = m_{v_n} \oplus H_{n-1}^{m_v} \oplus H_l^{m_f} \tag{6}$$

This block will force the input to the 3DES encryption in the MAC calculation to be the same for the new, forged message and the old, observed message. Therefore, even though we do not know the keys k_2 and k_3 we know that this new message will have the same MAC as m_v. The crafted block $m_{f_{l+1}}$ will, however, contain random data which most likely would not pass any message verification at the Application Layer. We now extend this approach to give us more control over the data we need to add to the message, and make it into an acceptable ERTMS message.

To be able to create a message that will be accepted by the train, we will leverage an additional feature of the Application Layer protocol, namely the ability to include optional data packets into the message. For the purposes of our attack we will include the packet for sending plaintext messages into the forged message. This packet is used to send messages of up to 255 characters that will be displayed on the driver's console. Intuitively, we want the random data contained in block $m_{f_{l+1}}$ to be the text message included in the additional packet. We make use of conditions that can be included to indicate when the message should be displayed to the driver. These conditions can be chosen such that the message will probably never be displayed.

According to the specification, the message in the text message packet needs to conform to the ISO 8859-1 encoding standard, which includes ASCII. It is not specified whether this is checked when receiving the message, when it is displayed or even at all. If the encoding is not checked, at this point, we can use the message for the random block exactly as described before. However, below we will assume the encoding is checked when the message is received and we show how to construct a forged message that includes a valid encoded text message.

In order to use the technique described previously to construct a forged message, we include a text message of 16 characters such that we have control over the last two full blocks in the MAC computation (see Figure 4(c)). The beginning of the message is again denoted by $m_f = (m_{f_1}, \ldots, m_{f_l})$. Two additional parts, that form the actual text message, will be appended in blocks $m_{f_{l+1}}$ and $m_{f_{l+2}}$. The addition of two blocks gives us enough flexibility in the plaintext to allow these blocks to conform to the ISO 8859-1 standard.

We start by computing the input to the 3DES block for the original message ($H_{n-1}^{m_v} \oplus m_{v_n}$) and the intermediate MAC of the fixed part of the forged message ($H_l^{m_f}$) using k_1. We then randomly generate the first half of the text message in the correct encoding, including it in block $m_{f_{l+1}}$. We continue by computing the value of the last message block $m_{f_{l+2}}$, such that m_f has the same MAC as the original message, using Eq. (6) and $H_{l+1}^{m_f} = DES_{k1}(m_{f_{l+1}} \oplus H_l^{m_f})$:

$$m_{f_{l+2}} = m_{v_n} \oplus H_{n-1}^{m_v} \oplus H_{l+1}^{m_f}$$

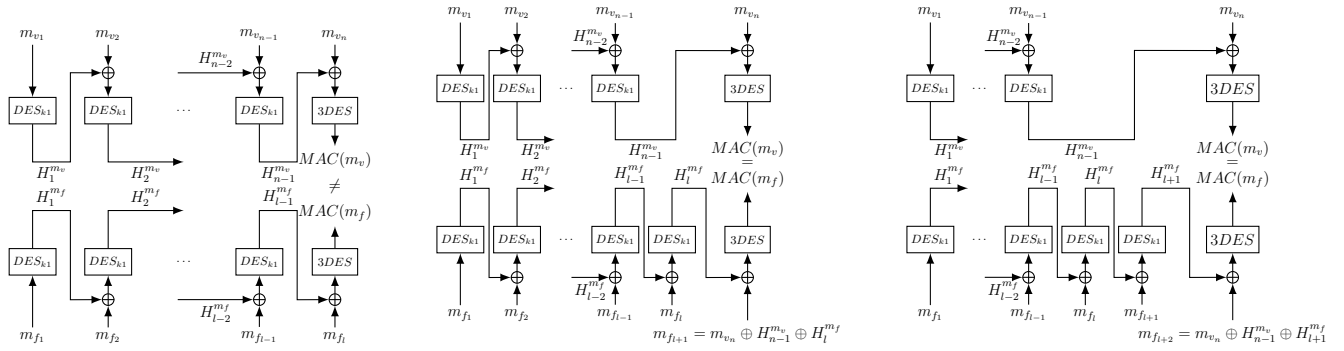
We then check if $m_{f_{l+2}}$ is a valid ISO 8859-1 encoded message. If this is not the case, we start over by generating another random first half of the text message $m_{f_{l+1}}$ and see if this results in a correctly encoded block $m_{f_{l+2}}$. Once we have a correctly encoded $m_{f_{l+2}}$ we have our forged message with the correctly encoded text message consisting of $m_{f_{l+1}}$ and $m_{f_{l+2}}$. In the next section we give an example of a forged Movement Authority message.

To determine the probability that we find a correctly encoded $m_{f_{l+2}}$, we assume its distribution is uniform. This should be the case due to the DES encryption used to compute $H_{l+1}^{m_f}$. As 65 byte values are not defined in ISO 8859-1, the probability that a random string of 8 bytes is correctly encoded according to ISO 8859-1 is:

Using the assumption that DES is a pseudo-random function, we therefore need 10 tries on average to find a correctly

Variable	Length (bits)	Description	Example
Message 3 (Movement Authority)			
NID_MESSAGE	8	Message type	3 (Movement Authority)
L_MESSAGE	10	Length of message (bytes)	51
T_TRAIN	32	Train timestamp	1327095428
M_ACK	1	Acknowledgement required	0 (Acknowledgement not required)
NID_LRBG	24	ID of Last Relevant Balise Group	1
Packet 15 (Movement Authority)			
NID_PACKET	8	Packet ID	15 (Movement Authority)
Q_DIR	2	Direction	2 (Both directions)
L_PACKET	13	Length of packet (bits)	113
Q_SCALE	2	Scale used for definition of resolution	2 (10m scale)
V_EMA	7	Maximum speed	40 (200km/h)
T_EMA	10	Validity time	1023 (unlimited)
N_ITER	5	Number of iterations	0 (No iterations)
L_ENDSECTION	15	Length of section in MA	5000 (50000m)
Q_SECTIONTIMER	1	Section timeout Qualifier	0
Q_ENDTIMER	1	Timer for end section in MA qualifier	0 (No information)
Q_DANGERPOINT	1	Indicates whether a danger point exists or release speed is to be specified	0 (No information)
Q_OVERLAP	1	Indicates whether overlap exists or release speed is to be specified	1 (Overlap information to follow)
D_STARTOL	15	Distance from overlap timer start to end of MA	0
T_OL	10	Validity period for overlap	0
D_OL	15	Distance from the end of the MA to end of overlap	0
V_RELEASEOL	7	Release speed for overlap	126 (Use calculated onboard speed)
Packet 72 (Plain Text Message)			
NID_PACKET	8	Packet ID	72 (Plain Text Message)
Q_DIR	2	Direction	0 (Reverse)
L_PACKET	13	Length of packet (bits)	220
Q_SCALE	2	Scale used for definition of resolution	2 (10m)
Q_TEXTCLASS	2	Class of Message to be displayed	0 (Auxiliary)
Q_TEXTDISPLAY	1	Display message if one/all events fulfilled (start/end events relation)	0 (as soon as one event fulfilled)
D_TEXTDISPLAY	15	Distance at which text is displayed (start event)	32767 (327670m)
M_MODETEXTDISPLAY	4	Operating mode for text display (start event)	9 (System Failure)
M_LEVELTEXTDISPLAY	3	Operating level for text display (start event)	0 (Level 0)
L_TEXTDISPLAY	15	Length the text is to be displayed for (end event)	0 (0m)
T_TEXTDISPLAY	10	Time the text is to be displayed for (end event)	0 (0 seconds)
M_MODETEXTDISPLAY	4	Operating mode for text display (end event)	9 (System Failure)
M_LEVELTEXTDISPLAY	3	Operating level for text display (end event)	0 (Level 0)
Q_TEXTCONFIRM	2	Confirmation required	0 (Not required)
L_TEXT	8	Length of text message	16 (16 chars)
X_TEXT	variable	Contents of text message	...

Table 2: The structure of a Movement Authority message. The message contains a Movement Authority packet and a Plain Text Message packet. We also show, as an example, the values we used in our forged message.



(a) Messages m_v and m_f have different MACs.

(b) Messages m_v and m_f are made to have the same MAC by adding block $m_{f_{l+1}} = m_{v_n} \oplus H^{m_v}_{n-1} \oplus H^{m_f}_l$.

(c) A properly encoded Plain Text Message requires two blocks $m_{f_{l+1}}$ and $m_{f_{l+2}}$ to be added. These blocks will be properly encoded according ISO 8859-1 as described in Section 5.

Figure 4: Creating forged message m_f by extending it with an additional block(s) to match the MAC of the valid message m_v.

encoded block and 50 tries to have a success rate of 99%. If we limit ourselves to ASCII encoding, the probability drops to $\left(\frac{95}{256}\right)^8 \approx 0.0004$. In this case, we still only need 2780 tries on average to find a correctly ASCII encoded block.

6. PROOF OF CONCEPT

In this section, we present the implementation of our attack and detail the process in which we identified collisions and implemented the DES key recovery process.

As a message to forge, we use a potentially damaging Movement Authority (MA) message with an attached Plain Text Message (PTM) packet using the details outlined in Table 2. This MA changes the maximum allowed train speed to 200 km/h for almost 328 km. The values in the message are chosen such that, by including optional variables, the text message exactly lines up with the last two message blocks in the computation of the MAC: a 40 bits prefix is added by the MAC algorithm and the message header (75 bits) together with the MA packet (113 bits) and Plain Text Message, without the actual text message, (92 bits) is 280 bits. This fixed part of the message is exactly contained in the five message blocks, so the text to be displayed will start at the beginning of the sixth block.

In order to find a MAC collision, we developed our own program to generate ERTMS messages. One of the most common messages sent between train and trackside is message 146: the Acknowledgement Message (see Table 1) to confirm the receipt of 'normal' priority messages. Messages sent from the backend to a train can request acknowledgements to be provided. This acknowledgement message contains the current timestamp, based on the train clock, as well as the timestamp for the message being acknowledged. Whilst the timestamp is 32 bits in length, they are relative to the onboard clock, wrapping around by design and therefore is not an issue, as is it is unlikely to happen. We computed the MAC value for many acknowledgement messages. using fixed keys, and looked for collisions. We parameterise the timestamps, where a defined offset was used between the timestamp being acknowledged and current time. This offset was specified to be intervals of 10, 20 and 30 ms, that is, the simulated train would acknowledge messages within the defined offset period. The remainder of the message (ETCS ID and Message ID) were left static, to simulate one train acknowledging all messages.

We implemented this algorithm in Java and generated approximately 12.9 billion $(3 * (2^{32} - 1))$ values. Equation 3, tells us that this leads to a 99.999% chance of a collision.

Collisions were detected using the standard Unix utilities `sort` and `uniq`. The system used to generate the MACs contained an Intel Xeon E5-2420 CPU, running at 2.2GHz. The complete process finished within two days, resulting in the discovery of 8 collisions, e.g. the two acknowledgement messages:

```
00120000020A9203A2105E0480000062105DFD0000000000
00120000020A9203AAE360078000006AE360000000000000
```

both have the same intermediate MAC H_n:

```
80B7557F31566DBB
```

for k_1 = `01020407080B0D0E`. The other collisions are given in Appendix A. Details on the probabilities of a collision and the time needed for an attacker to find a collision are discussed in the next section.

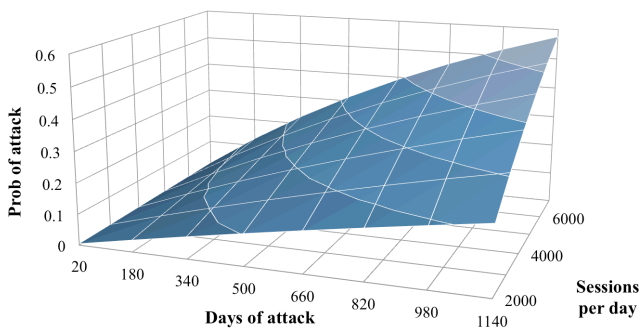
Verifying a single key guess using these collisions requires six DES encryptions (three for each message to calculate the inputs to the 3DES block). The fastest DES cracker we could find in the literature was

the RIVYERA, which uses dedicated hardware and takes slightly less than a day to crack a single DES key[4]. The cost of building the system is approximately $140,000.

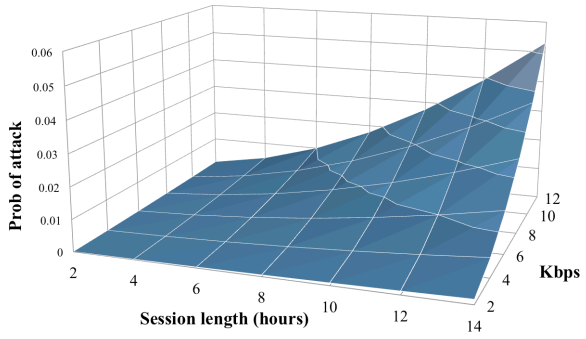
This is too slow to break the key in time to attack a train, so we investigate the cost and speed of cracking DES using Amazon's cloud services. We rented an Amazon EC2 instance with 16 NVIDIA Tesla K80 GPUs, (`p2.16xlarge`) and used the state of the art password cracker `hashcat`[5] to benchmark the number of possible DES operations per second. Scaling this up to many machines we found that we could brute-force the k_1 DES key in 30 minutes for $48,960,

[4]https://www.voltage.com/breach/the-state-of-the-art-in-key-cracking/

[5]https://hashcat.net/



(a) Days of attack vs Sessions per day vs Prob of attack



(b) Days of attack vs Sessions per day vs Prob of attack

Figure 5: Attack probabilities for a range of different session lengths and data speeds

and even faster for proportionally more money. A full breakdown of this calculation, and the benchmarking, is provided in Appendix B.

Using the collision above, we constructed a forged message with the same MAC. We used a Python script to create a forged message containing the Movement Authority, given in Table 2. Given k_1, this script finds a valid text message using the approach discussed in Section 5. It can find valid text messages either for the full ISO 8859-1 encoding or only ASCII. In the case of the messages above this process took 0.209s on a normal laptop containing an Intel Core i7-4550U CPU to give us the following ASCII encoded text to complete our forged message:

```
Z|1MB%<w*RRf)8n/
```

This leads to the ERTMS message:

```
030cd3c677a100000021f01c651ff809c4080000000007e4801
b90fffd2000000120105a7c314d42253c772a52526629386e2f
```

which we constructed without the second and third DES keys, and has the same intermediate MAC as the acknowledge messages above (`80B7557F31566DBB`), and therefore also the same final MAC. This forged movement authority message would now be accepted by a train, and is broken down in Table 2.

We are making the Python scripts available anonymously. Please see the link in the footnote[6].

When the attack is deployed, the attacker would need to actively jam the GSM-R uplink to ensure that acknowledgement messages sent by the train are not received by the RBC. Likewise, once deployed, the downlink would have to be jammed so that the train does not receive any conflicting movement authorities from the RBC which could shorten it. The RBC, may therefore, through interlocking identify two trains occupying the same section of track, and attempt to intervene by issuing an emergency stop or modified movement authority to the 'victim' train. That said, an attacker who jams the GSM reception would prevent the RBC commands being received. As the frequencies are the same for GSM-R voice, it would also prevent the signalling centres contacting the train driver via voice. For a fixed attacker at an RBC, or base station, it would be a case of dropping the messages which could restrict the movement authority or make the train go against the attacker's wishes.

[6]http://pastebin.com/Ge6gM1Qm

7. DATA CAPTURE

In this section, we show how much data an attacker would need to see, how long they would need to collect traffic for, and what the probability of a successful attack would be. In order to find a collision, traffic between the train and backend needs to be captured; one way to collect the traffic needed to find a collision would be to intercept GSM-R data between a train and mobile base station. Alternatively, data could be captured from the network infrastructure which connects the base stations, RBCs and control centres. Capturing data from this network infrastructure leads to much more data and so a high chance of finding a collision, but the network is more difficult to compromise. We consider both of these possible attacks below.

7.1 Monitoring a single train

In this subsection, we consider an attacker who monitors a single train, perhaps while travelling on it. The GSM protocol has been shown to be weak against an attacker, given sufficient pre-computation [16], therefore the attacker can easily decrypt this.

We consider a 1% chance of an attacker taking over a train as unacceptable, and in Section 5, we showed that an attacker gets this chance of a collision from 6.1×10^8 messages. Assuming an average message length of 32 bytes this would require 19.5 GB of data. GSM-R has a maximum data speed of 14Kbps, however, this connection is not always used at full capacity. If we assumed that a train sends data at 10Kbps, it would take a train 23 days to send 19.5GB. We note that there is nothing in any of the specification that enforces a limited life on a session key; in practice the session key is renegotiated whenever a train starts up, or when it starts communicating with a new RBC, therefore it seems unlikely that a single train will currently use a session key long enough for this to become a problem.

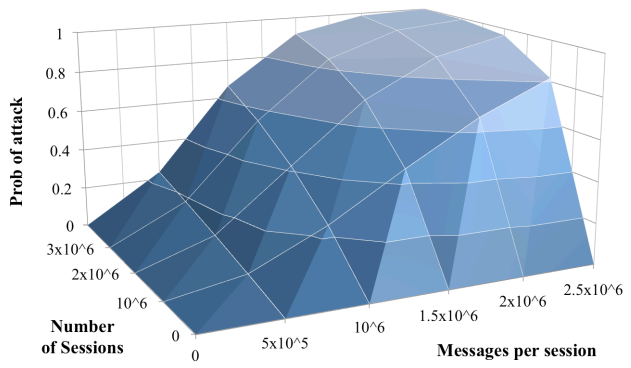


Figure 6: Attack probabilities for the number of messages and sessions observed

7.2 Multi-session traffic capture

Rather than monitoring a single train, an attacker could monitor a large number of trains at the same time, by for instance, tapping into the wired connections between the GSM-R base stations and the RBCs. These cables are typically buried in the ground or carried overhead along train tracks, therefore, accessing them on mass would involved considerable time and effort for an attacker, but would be possible. Base stations are located in open spaces, unlike public GSM stations, where unauthorised access to these may not be detected. Access to the underlying infrastructure, such as controllers may be carried out by an inside attacker, however.

We can adapt Eq. (3) to give us the probability P of finding one of more collisions in S sessions each of which contains M messages:

$$P = 1 - \prod_{i=1}^{M-1} \left(1 - \frac{i}{N}\right)^S \approx 1 - e^{-M(M-1)\cdot S/(2N)} \quad (7)$$

We plot the range of possible values for this in Figure 6. To make this more concrete, we consider a country the size of the UK which has in the region of 4000 trains running everyday. We additionally assume an average data speed of 10Kbps, sessions of 10 hours and message sizes of 32 bytes. The data speed, session lengths and message size then give us the number of messages per session and the number of session an attacker can monitor would then be 4000 times the number of days they were willing to wait. Using Eq. 7 we calculate that an attacker would have a 1% chance to find a collision, and so take control of a train, within 45 days and a 50% chance in 8 years. These figures suggest that, while difficult and expensive to pull off, this attack is within the realm of possibility.

Our figures are estimates of typical usage – high speed trains that move between RBC areas may have shorter sessions and local trains that spend a whole day in an area may have longer sessions. Data may be sent at a higher or lower speed and the number of trains in service will continue to grow. Figures 5(a) and 5(b) show how the likelihood of an attack varies as we change some of these estimates. Figure 5(a) shows how the success probability grows for an attacker that is willing to wait longer and for different sizes of train network. Figure 5(b) assumes a 45 day attack and shows how the success probability changes with the session length and data speed. We note that for short sessions and low data speeds the protocol is safe, however this quickly becomes unsafe as the amount of data send increases. We discuss what is a safe limit to the data sent over EuroRadio in the next section.

As discussed in section 2.1, GSM Capture is not a new concept, however, we outline an example process which may be undertaken. We use the HackRF software-defined radio, `airprobe` and `gnuradio` (Fig. 7) Linux tools to capture traffic on the Cambrian Line, an ERTMS Level 2-fitted line on the North Wales coastline (see Appendix C). For other software-defined radios, alternative tools, for example, OsmocomBB, may be used. Using the radio, the base station frequency must be identified - the `kalibrate` tool scans for GSM-R base stations, where `airprobe` can then capture on that frequency and decode into Control Channel and encrypted traffic. The capture files can be passed to `kraken` to retrieve the A5/1 key. The TMSI is exposed in the 'Paging Request' (Fig. 8) messages which may then be used in the input to establish the key. Another method, impersonating an existing base station can be achieved through open-source tools for example OpenBTS, OsmoBTS and Yate.

8. MITIGATIONS

In this section, we present the proposed mitigations that reduce the exposure of this attack, as well as other proposals to further enhance the security of train to trackside communication for the future. Whilst recommendations to promote and introduce immediate solutions may be given, we also must consider the impact and cost to infrastructure manager and operators to implement such changes. We consider the following mitigations, ranked in order of highest compatibility and lowest implementation cost to lowest compatibility and highest implementation cost:

1. Restricting EuroRadio use in high-bandwidth Applications
2. Forcing sessions to be renegotiated often in order to ensure that the probability of finding collisions is $P \leq 10^{-6}$.
3. Implementation of an alternative MAC Algorithm
4. Combining EuroRadio and Application Layers to provide a combined, secure message sending/receiving service

Although EuroRadio currently is safe for use in current ERTMS command and control applications against an attacker that targets a single train, our recommendation is that EuroRadio should not be considered for use in future high-bandwidth applications, for example streaming applications, for example remote condition monitoring. High-bandwidth applications reduce the time required to get collisions (see Figure 6) and thus session key recovery becomes more likely. As a result, the MAC algorithm in EuroRadio should not be considered for such applications.

An alternative short-term solution is to force sessions to be renegotiated by tearing down the existing session and forcing a new key to be used. The threshold number of messages $M = 6.1 \times 10^8$ discussed in Section 4 assumes a collision probability $P = 1\%$. However, as previously stated, this probability is to high. We consider $P = 10^{-6}$ a more

MAC algorithm	Avg. time (ns)	Time impact vs. EuroRadio MAC (%)	QCR
EuroRadio MAC (current)	10276.89	-	×
3DES Patch	13155.255	28% worse	×
AES-256-CBC MAC	8589.98	12% better	✓
HMAC-SHA-256	4558.64	55% better	✓

Table 3: Performance Summary of MAC Algorithms under Assessment

acceptable probability [24]. Using Eq. (7) we can compute the number of messages per session for the fixed probability P as:

$$M \approx \sqrt{\frac{2N}{S} \cdot ln\Big(\frac{1}{1-p}\Big)} \tag{8}$$

For $N = 2^{64}$ and $S = 1825000$ sessions, assuming an attacker that can monitor 5000 sessions per day for 365 days, the number of messages per session that result in a collision probability of $P = 10^{-6}$ is $M = 4496$. We recommend using a counter that forces the session to be re-established once this number of messages is reached.

For infrastructure managers and manufacturers, such a bound could be enforced in a range of places: either at the RBC where all trains coming into an area of control are subject to this message count bound, or as an onboard software update, although this limits the mitigation. Having the bound enforced by the RBC is a stronger proposition, as there are fewer operating in a country, reducing costs to the infrastructure manager to implement, and can be built and tested as a software update in a shorter timeframe than updating onboard software. For the 'enforcing' entity, a counter simply is maintained, in addition to the clock for messages sent and received, incremented each time a message is sent or received to or from a train. We recommend counting the number of messages send and received from the train, rather than relying on the existing clock, as one no longer has to consider data speeds. This solution is fully backwards-compatible with no identified limitations, where the current EuroRadio specification already has support for the train and RBC terminating sessions.

Another problem is that EuroRadio still relies on the security of the DES and 3DES ciphers. Whilst DES has been shown to be feasibly broken, 3DES does not yet have a full key recovery attack, however it is estimated that it will be feasible to brute-force 3DES by 2030[7]. Therefore, longer-term solutions, like changing the MAC scheme, will require significant changes to software and, for some ciphers, even extending key length. For infrastructure managers, updating the derivation key on every train and RBC is not a simple process, where all trains and RBCs must be updated at the same time to support such changes. This would likely result in significant implementation and deployment cost. While we believe the current setup is no immediate threat to rail infrastructure, these longer-term solutions should be considered as part of the ratification process of updated ERTMS standards as alternative safety features.

As alternative MAC schemes, AES and HMAC-based MACs were considered, where efficiency and long-term viability were considered. Any proposed MAC changes should be considered to be quantum resistant to prevent key recovery in a 'post-quantum world'. In order to evaluate the performance impact and resistance to collisions, we carried out the same collision detection code under the different MAC algorithms. Where key size under proposed algorithms was too short, the same prefix was used and extended with distinctly different bits. These computations were also timed to measure relative performance against the current MAC algorithm, which was used as a baseline. Our results are given in Table 8.

We show that the impact of using a DES-based MAC algorithm has a significant impact on the generation of the MAC, compared to the proposed alternatives. The theoretical performance improvements, however, do not directly translate to real-life functionality improvements. For example, the train's stopping distance is not significantly affected by changing the MAC algorithm as the time to generate a MAC under the current scheme can be quantified to be at 200km/h to be 0.05cm. Improvements, therefore are minimal in terms of distance travelled.

Finally, if some ERTMS Specifications were to be combined, the EuroRadio and Application Layers, which themselves provide independent defences, may be combined as a unified layer which provides authenticity, integrity and replay protection. This, however, requires significant changes to the underlying specifications to support this change, with cost, development and ratification involved.

9. CONCLUSION

We have presented the results of our analysis of the EuroRadio MAC algorithm, the algorithm used to secure communication between trains and backend equipment. On the ERTMS stack EuroRadio provides the safety-critical function of message authentication. We assessed each layer to determine potential weaknesses that could be exploited. This allowed us to develop a key recovery attack by leveraging collisions in the MAC for different messages. By recovering one of the keys used in the generation of the MACs, it is then possible to exploit cryptographic weaknesses in EuroRadio. We combined this with a second vulnerability in the Application Layer protocol to forge a Movement Authority with a valid MAC. The forged messages would be accepted by a train, potentially placing it in an unsafe situation. We have discussed the risk this poses to train networks and we have propose possible mitigations, such as restricting the number of messages per session.

We have discussed our attack with Maria Grazia Vigliotti at the UK's Rail Safety and Standards Board (RSSB), and Network Rail, they have agreed that EuroRadio is not safe to be used with a transport protocol faster than GSM-R. They also agree with the fact that monitoring the whole rail backbone is theoretically possible. However, they consider this to be more difficult than breaking into a key management centre and/or bribing train staff. RSSB has no objection to us publishing our results.

Acknowledgements

Funding for this paper was provided by the UKs 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.

[7]https://www.keylength.com/en/3/

10. REFERENCES

[1] ANSI. ANSI X9.19:1998 Financial Institution Retail Message Authentication. Technical report, ANSI, 1998.

[2] E. Barkan, E. Biham, and N. Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In D. Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer Berlin Heidelberg, 2003.

[3] 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.

[4] A. Dabrowski, N. Pianta, T. Klepp, M. Mulazzani, and E. Weippl. Imsi-catch me if you can: Imsi-catcher-catchers. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 246–255, New York, NY, USA, 2014. ACM.

[5] A. DasGupta. The matching, birthday and the strong birthday problem: a contemporary review. *Journal of Statistical Planning and Inference*, 130(1):377–389, 2005.

[6] J. de Ruiter, R. J. Thomas, and T. Chothia. A formal security analysis of ERTMS train to trackside protocols. In A. R. Thierry Lecomte, Ralf Pinger, editor, *Reliability, Safety and Security of Railway Systems: Modelling, Analysis, Verification and Certification. International Conference, Paris, France, June 28-30, 2016, Proceedings*, Lecture Notes in Computer Science, 2016.

[7] ERA. SUBSET-026: System requirements specification, version 3.5.0. Technical report, 2015.

[8] M. Franekova and P. Chrtiansky. Key Management System in ETCS. *Transport System Telematics*, 2009.

[9] 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.

[10] GSM-R Functional Group. EIRENE Functional Requirements Specification, version 7.4.0. Technical report, 2014.

[11] GSM-R Functional Group. EIRENE System Requirements Specification, version 15.4.0. Technical report, 2014.

[12] H. Handschuh and B. Preneel. Minding your MAC algorithms. *Information Security Bulletin*, 9(6):213–221, 2004.

[13] ISO/IEC. ISO/IEC 9797-1:2011 – Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher. Technical report, ISO/IEC, 2011.

[14] M. Kalenderi, D. Pnevmatikatos, I. Papaefstathiou, and C. Manifavas. Breaking the GSM A5/1 cryptography algorithm with rainbow tables and high-end FPGAS. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 747–753. IEEE, 2012.

[15] L. Karstensen. GSM A5/1 rainbow tables in Oslo, Norway. Available: https://lassekarstensen.wordpress.com/2013/08/08/gsm-a51-rainbow-tables-in-oslo-norway/, 2015. Online.

[16] J. Lu, Z. Li, and M. Henricksen. Time-Memory Trade-off Attack on the GSM A5/1 Stream Cipher Using Commodity GPGPU. In *13th International Conference on Applied Cryptography and Network Security (ACNS 2015)*, 2015.

[17] C. J. Mitchell. Key recovery attack on ANSI retail MAC. *Electronics Letters*, 39(4):361–362, 2003.

[18] F. Pépin and M. G. Vigliotti. *Risk Assessment of the 3Des in ERTMS*, pages 79–92. Springer International Publishing, Cham, 2016.

[19] S. Petrovic and A. Fùster-Sabater. CRYPTANALYSIS OF THE A5/2 ALGORITHM. Cryptology ePrint Archive, Report 2000/052, 2000. http://eprint.iacr.org/.

[20] B. Preneel and P. Van Oorschot. On the security of iterated message authentication codes. *Information Theory, IEEE Transactions on*, 45(1):188–199, Jan 1999.

[21] B. Preneel and P. C. van Oorschot. Key recovery attack on ANSI X9. 19 retail MAC. *Electronics Letters*, 32(17):1568–1569, 1996.

[22] SR Labs. Decrypting GSM phone calls. Available: https://srlabs.de/decrypting_gsm/, 2010. Online.

[23] UNISIG. SUBSET-037 - EuroRadio FIS, version 3.2.0. Technical report, 2015.

[24] J. Wolff. What is the value of preventing a fatality? In T. Lewens, editor, *Risk: Philosophical Perspectives*. Routledge, 2007.

APPENDIX

A. COLLISIONS

As part of our analysis of collision resistance of the MAC algorithm used in EuroRadio, we found 8 separate collisions for the Acknowledgement Message (Message 146). This was only for the intermediate MAC value (i.e. H_n), which is sufficient to detect collisions, as the 3DES transformation is deterministic. The collisions are shown in Table 4.

Intermediate MAC (H_n)	Plaintexts
365CA0E4D4901E85	00120000020A9203A2105E0480000062105DFF8000000000 00120000020A9203AAE360078000006AE360028000000000
410F1B9C2C09E958	00120000020A9203970598C5C00000570598C34000000000 00120000020A9203B04EA8D7C00000704EA8D54000000000
4BBDFBABD9757A38	00120000020A9203A9D9B5FDC0000069D9B5FB4000000000 00120000020A9203AC38CEEA8000006C38CEE58000000000
7A3D01D36BE88B21	00120000020A920385CCD6F280000045CCD6EB0000000000 00120000020A920386E4CFBCC0000046E4CFB7C000000000
80B7557F31566DBB	00120000020A9203A2105E0480000062105DFD0000000000 00120000020A9203AAE360078000006AE360000000000000
A7A3AD4FA4C6D433	00120000020A9203A16580E0400000616580DDC000000000 00120000020A9203A34C8FAF400000634C8FAA4000000000
BE23849D77705C72	00120000020A920398952D5AC0000058952D534000000000 00120000020A9203B553FC6480000007553FC5D0000000000
F813AED5FE3D445F	00120000020A9203A16580E0400000616580DB4000000000 00120000020A9203A34C8FAF400000634C8FACC000000000

Table 4: Pairs of messages which result in the same MAC under key $k_1 =$ 01020407080B0D0E

B. BENCHMARKING

The Amazon EC2 instance which was hired to perform the `hashcat` benchmarking was a `p2.16xlarge` instance. This type of instance is designed for high performance GPU computation[8] and costs $14.40 per hour to use. This instance comes with 64vCPUs, 734GiB of local RAM and, for our setup, an 8GB SSD-backed storage facility on the same network for minimal latency.

Each `p2` instance is equipped with an NVIDIA Tesla K80 GPU, with 5000 CUDA cores, 24GB of GDDR5 RAM. Our benchmarking instance was fitted with 16 K80 GPUs. At the time of benchmarking, the latest version of NVIDIA GPU Drivers and `hashcat` source were compiled and installed.

`hashcat` is optimised for OpenCL, allowing the GPUs to be leveraged for GPU-accelerated computation, and using the GPUs on board, our results are provided in the following section.

`hashcat` results

`hashcat` supports a benchmarking mode, allowing it to state the number of hashes, or values it is able to produce per second. The argument set and results presented below are broken down:

- `-m 1500` : Message Type: descrypt, DES(Unix), Traditional DES
- `-b` : Benchmark Mode
- `-w 4` : Workload Profile 4 – Extreme
- `-powertune-enable` : Enable automatic power tuning option on GPU

We use the *descrypt* message type, as this is the closest family of algorithm to simple DES encryption. *descrypt* works by taking a password as a 56-bit key, taking a 64-bit zeroed data input block, and encrypts this 25 times, where the hash is the output of this process. Thus, we state that the output speed from the `2.16xlarge` instance has a 25x factor improvement in speed, due to *descrypt* carrying out 25 rounds of DES encryption.

```
$ hashcat -m 1500 -b -w 4 --powertune-enable
hashcat (v3.10) starting in benchmark-mode...

OpenCL Platform #1: NVIDIA Corporation
======================================
- Device #1: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #2: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #3: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #4: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #5: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #6: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #7: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #8: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #9: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #10: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #11: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #12: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #13: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #14: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #15: Tesla K80, 2859/11439 MB allocatable, 13MCU
- Device #16: Tesla K80, 2859/11439 MB allocatable, 13MCU

Hashtype: descrypt, DES(Unix), Traditional DES

Speed.Dev.#1:   176.5 MH/s (482.39ms)
Speed.Dev.#2:   174.6 MH/s (482.68ms)
Speed.Dev.#3:   176.2 MH/s (483.11ms)
Speed.Dev.#4:   175.4 MH/s (485.09ms)
Speed.Dev.#5:   174.4 MH/s (483.27ms)
Speed.Dev.#6:   175.3 MH/s (480.53ms)
Speed.Dev.#7:   175.9 MH/s (483.79ms)
Speed.Dev.#8:   175.9 MH/s (483.84ms)
Speed.Dev.#9:   175.1 MH/s (481.23ms)
Speed.Dev.#10:   177.5 MH/s (479.46ms)
Speed.Dev.#11:   177.4 MH/s (479.99ms)
Speed.Dev.#12:   174.8 MH/s (486.82ms)
Speed.Dev.#13:   177.6 MH/s (484.14ms)
Speed.Dev.#14:   175.8 MH/s (484.19ms)
Speed.Dev.#15:   176.8 MH/s (481.54ms)
Speed.Dev.#16:   175.0 MH/s (481.54ms)
Speed.Dev.#*.:  2814.1 MH/s
```

We estimate the cost of breaking the EuroRadio MAC on a `p2.16xlarge` instance as follows: the instance produces 2,814,100,000 outputs per second. As this involves 25 rounds of DES, the actual indicative speed is 70,352,500,000 per second. To break DES on a single instance in terms of time can be calculated by the below equation:

$$\frac{2^{56}}{(70,352,500,000 * 60 * 60)} \approx 284 \; hours$$

On 400 `p2.16xlarge` instances, this would take $\sim$ 42 hours.

As the EuroRadio MAC involves 6 DES encryptions (32 byte messages with 3 DES encryptions each, totalling 6 to test the key), we can use the below equation to give a time of 30 minutes on 3,400 simultaneous instances:

$$\frac{6 \cdot 2^{56}}{(70,352,500,000 * 60 * 3400)} \approx 30 \; mins$$

As a `p2.16xlarge` instance costs $14.40 per hour to rent, this would cost (for 3,400 instances) $48,960.

[8] https://aws.amazon.com/ec2/details/

C. DATA CAPTURE

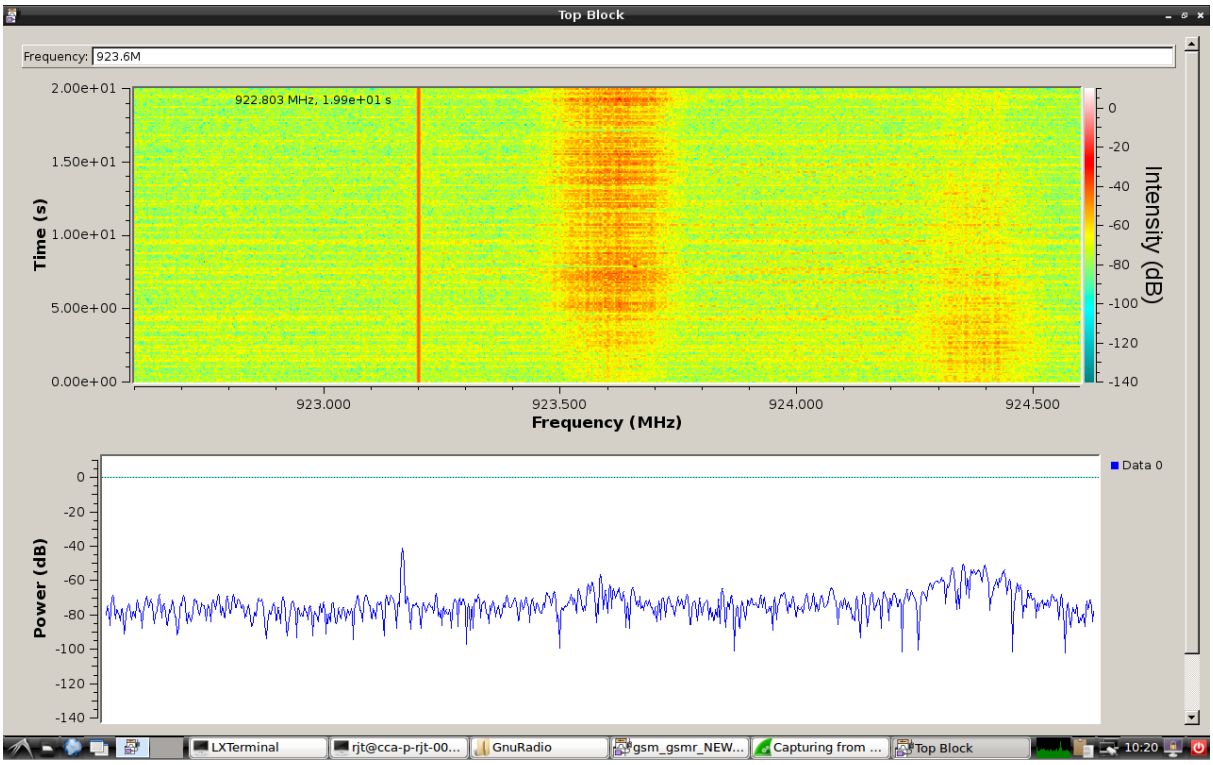


Figure 7: RF waterfall graph output from `gnuradio`. Potential GSM-R data (red areas) is shown on frequencies 923.6MHz and 924.4MHz.

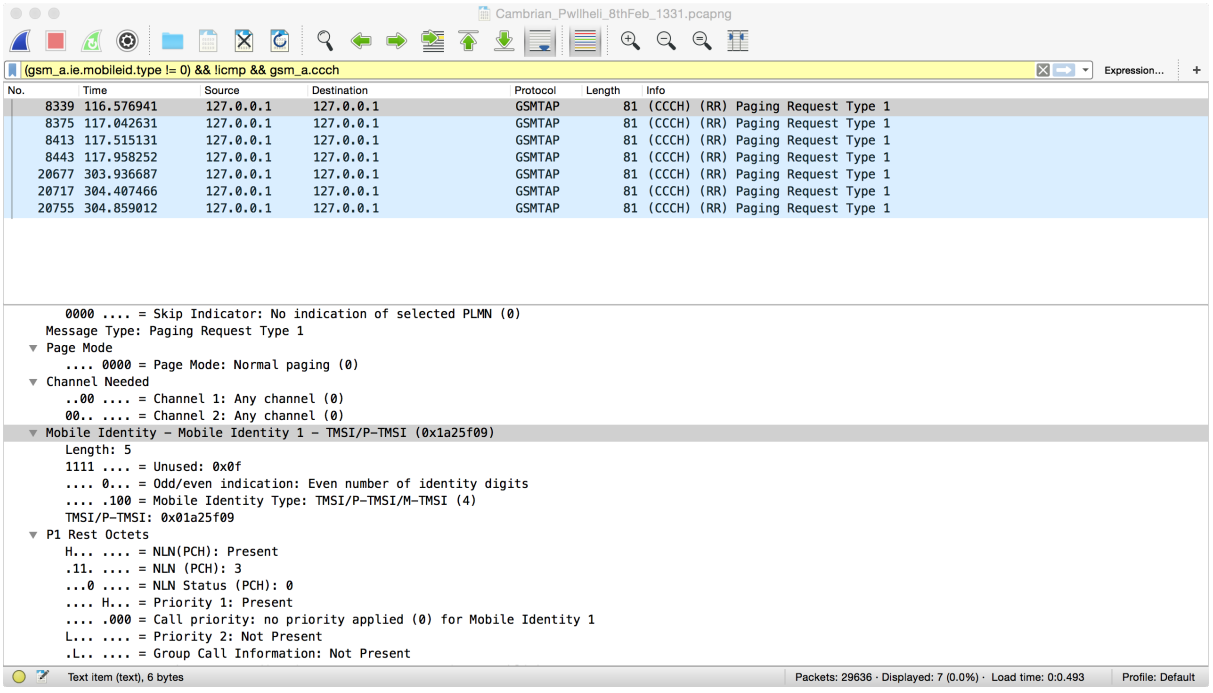


Figure 8: Wireshark screenshot showing captured GSM-R packets. The packets captured allow recovery of the TMSI value.