All cops are broadcasting: TETRA under scrutiny

Carlo Meijer
Midnight Blue
c.meijer@midnightblue.nl

Wouter Bokslag
Midnight Blue
w.bokslag@midnightblue.nl

Jos Wetzels
Midnight Blue
j.wetzels@midnightblue.nl

Abstract

This paper presents the first public in-depth security analysis of TETRA (Terrestrial Trunked Radio): a European standard for trunked radio globally used by government agencies, police, prisons, emergency services and military operators. Additionally, it is widely deployed in industrial environments such as factory campuses, harbor container terminals and airports, as well as critical infrastructure such as SCADA telecontrol of oil rigs, pipelines, transportation and electric and water utilities. Authentication and encryption within TETRA are handled by secret, proprietary cryptographic primitives. This secrecy thwarts public security assessments and independent academic scrutiny of the protection that TETRA claims to provide.

The widespread adoption of TETRA, combined with the often sensitive nature of the communications, raises legitimate questions regarding its cryptographic resilience. In this light, we have set out to achieve two main goals. First, we demonstrate the feasibility of obtaining the underlying secret cryptographic primitives through reverse engineering. Second, we provide an initial assessment of the robustness of said primitives in the context of the protocols in which they are used.

We present five serious security vulnerabilities pertaining to TETRA, two of which are deemed critical. Furthermore, we present descriptions and implementations of the primitives, enabling further academic scrutiny. Our findings have been validated in practice using a common-off-the-shelf radio on a TETRA network lab setup.

More than a year ago, we started to communicate our preliminary findings through a coordinated disclosure process with several key stakeholders. During this process we have actively supported these stakeholders in the identification, development and deployment of possible mitigations.

1 Introduction

Terrestrial Trunked Radio (TETRA) is a European standard for trunked radio used globally by government agencies, emergency services and critical infrastructure. Apart from networks dedicated to military operators and emergency services (such as C2000 in the Netherlands, ASTRID in Belgium, BOSNET in Germany and RAKEL in Sweden), TETRA is also widely used in industrial environments such as factory campuses, harbor container terminals and airports, as well as for SCADA telecontrol of oil rigs, pipelines, transportation and electric and water utilities.

TETRA authentication and encryption are handled by secret, proprietary cryptographic primitives. Authentication, key derivation and distribution is implemented in the *TETRA standard Authentication Algorithm set 1 (TAA1)*. Four *Key Stream Generator (KSG)* functions (called *TETRA Encryption Algorithm 1 to 4*, or, TEA1 to TEA4[1]) generate keystream for encryption of air interface traffic. TEA1 is approved for general use worldwide; TEA2 is approved for European use by emergency services; TEA3 is approved for extra-European emergency services and, TEA4 is once again intended for general use (but has hardly seen any adoption). While the TETRA specification is largely public [8, 12, 13], all the crucial cryptographic components required for assessment of the system remain secret, available only to select parties under strict NDAs. This runs counter to both the spirit of open technologies and Kerckhoffs's principle.

The potential consequences have previously been illustrated by the fate of A5/1 [2], A5/2 [14] and their GMR variants [5] in cellular and satellite communications, as well as the Crypto AG / Hagelin sensitive communications equipment [26], allowing backdoored or practically exploitable ciphers to fester in public and critical infrastructure for decades. The potential implications of security issues for confidentiality and integrity of critical voice and data communications, as well as telecontrol of critical infrastructure, are significant.

In an attempt to prevent the secret cryptographic primitives used in TETRA from becoming public, serious con-

[1] Not to be confused with Tiny Encryption Algorithm [35]

straints have been imposed, both technically and regulatory. However, with millions of TETRA devices being deployed around the world, the primitives are bound to fall into the hands of adversarial parties, through either reverse engineering efforts or leaked/stolen documents.

Acquisition of TETRA cryptographic primitives We distinguish three methods of acquiring the TETRA cryptographic primitives. First, the official route through ETSI, involving extensive eligibility criteria and non-disclosure agreements. Second, theft of the official specification or source code from any ETSI-approved party. Third, reverse engineering of a firmware or hardware implementation.

The ETSI standards body is the official institution governing access to descriptions of TETRA cryptographic primitives. They can be obtained under a 'Non disclosure and restricted usage license', restricting copying of the specification and requiring implementation of countermeasures against reverse engineering in end products. Approval is given to 'bona fide' manufacturers of TETRA radio equipment, and a record is kept in a register [9,10]. For TEA2, additional requirements apply, such as geographic restrictions and a license requirement for parties involved in installing, repairing and/or destroying TEA2 capable equipment [11].

Clearly, the restrictions imposed by ETSI prevent academic discussion. It is worth noting that the scientific community is likely considerably more impacted by the restrictions than a malicious actor with clandestine goals and fewer inclinations towards legal compliance. While the possibility of theft (through hacking, coercion or otherwise) is self-evident, estimating the feasibility of reverse engineering is less straightforward. Such an endeavor is nontrivial, due to the aforementioned ETSI-mandated countermeasures against reverse engineering. We successfully recovered the TAA1 suite of primitives and the TEA1, TEA2 and TEA3 stream ciphers from a Motorola MTM5400 (a common off-the-shelf radio) and its associated firmware images, using software exploitation techniques. With our research, we demonstrate the feasibility of the reverse engineering approach, while complying with the legal framework and remaining at liberty to share our findings with the public.

Uncovered issues Having obtained the primitives through reverse engineering, we find ourselves in the unique position to study the security of TETRA in a more in-depth fashion than any previously published work in the scientific literature. We found several serious issues, pertaining to air interface encryption, identity encryption and the authentication. Ranging from trivial key recovery through brute-force and meet-in-the-middle attacks to keystream recovery by an active adversary, our work proves the long-standing reputation of TETRA as a highly secure system to be unjustified.

Surprisingly, one of the most severe issues (Section 5.1) could have been identified without access to the cryptographic primitives. We speculate the closed nature of TETRA security has dissuaded the usual public research from taking place.

1.1 Related work

The little prior work on TETRA security that exists consists mostly of high-level comparative surveys [20,28], end-to-end encryption and SIM card implementation proposals [22,36], as well as generic threats to open or misconfigured networks [27,30]. In addition, offensive electromagnetic activities (EMA) such as jamming and radio localization have received historical attention [6,24,29,31]. At the level of protocol security, some prior work has been done limited to authentication protocol issues affecting network availability [7] and malicious terminal cloning [23]. However, to the best of our knowledge, there is no comprehensive prior work on TETRA security that takes evaluation of the underlying security primitives into account, likely as a result of their confidentiality up until this work. Finally, while there are some open-source SDR-based TETRA decoder implementations[2], none of these include the required security primitives for the same reasons.

1.2 Our contribution

Our contribution is fourfold.

First, we are the first to make the cryptographic primitives that underpin the security of TETRA available for public scrutiny, after a period of over two decades of secrecy. This enables cryptographers, security researchers, system integrators, and users of TETRA to investigate and form an informed opinion about the actual security guarantees that TETRA offers, rather than those advertised.

Second, we are the first to provide a public in-depth analysis on the security of TETRA, wherein we identify several weaknesses which demonstrate that the actual security guarantees offered are considerably less than claimed.

Third, while numerous previous examples exist, we demonstrate once again that attempting to keep cryptographic primitives confidential, while simultaneously distributing them in millions of end-user devices around the world, is a strategy that eventually collapses. This in itself need not be fatal for the security of the system as long as the secrecy is not relied upon for security. However, in the case of TETRA, the TEA1 stream cipher is deliberately weakened in a way so evident that anyone with knowledge of the algorithm would be able to passively intercept TEA1 encrypted traffic with very little resources. This runs directly counter to the Kerckhoffs's principle. The issue is

[2]See https://osmocom.org/projects/tetra and https://www.rtl-sdr.com/tetra-kit-a-new-open-source-tetra-decoder/

aggravated by the fact that the non-disclosure agreements and the strict conditions under which the TEA1 specification is provided causes system integrators and end users to remain uninformed, while adversarial actors (state sponsored or otherwise) likely have access to the specification and need not resort to reverse engineering.

Fourth, we release all resources and tools developed during the research trajectory that we deem of potential public interest[3]. Additionally, we have provided patches bringing cryptography support to *OsmocomTETRA*, an open-source tool for demodulating and interpreting TETRA traffic. Furthermore, we release our suite of tools developed for the Motorola MTM5000 series of radios to the public as well. This includes a disassembler plug-in for IDA Pro covering the full TI C674x instruction set, disassembly support for the architecture in the *Capstone* framework, and support for the architecture in the *RetDec* decompiler. Lastly, we also publicly release our tools for unpacking Motorola firmware packages on an MTM5000 series radio, as well as utilities for instrumenting, debugging, monitoring and packet injection. The goal of these releases is to turn the MTM5000 series into a development platform for researchers, hopefully allowing security research into TETRA to gain momentum, and eventually hold TETRA security to a standard on par with ones that have evolved through public scrutiny.

2 Recovering cryptographic primitives

Although this paper's main focus is on the security of TETRA itself, we believe it is valuable to give an impression of the reverse engineering process, in particular because some may argue that the secrecy of the cryptographic primitives is a part of the security of TETRA as a whole.

2.1 Target device selection

As a preliminary step, in order to decide which TETRA device to acquire, we investigated firmware images for several devices that can be found online on amateur radio forums. As expected, none of them contain the primitives in unprotected form. It is worth noting that some firmwares contain a presumably encrypted, high-entropy data section, which we expect to embed the TETRA cryptographic primitives. In virtually all cases where such a high-entropy section is absent, the device itself and its SoC are produced by the same manufacturer, suggesting that the algorithm may be implemented in hardware in the SoC. Since hardware reverse engineering is considerably more time and resource intensive, we decided to avoid these devices.

Ultimately, we opted for the Motorola MTM5400[4]. It is built around the Texas Instruments OMAP-L138 SoC[5], which houses an ARM core and a TI C6748 DSP[6], offering secure boot and a *Trusted Execution Environment (TEE)* in which confidential code can be loaded and executed without revealing the implementation. We found through static analysis that the TEE is indeed used by Motorola and that the high-entropy section contained within the firmware is in fact provided to the TEE as a loadable protected module. Code related to TEE invocation referenced error messages pertaining to TETRA cryptography, providing the final confirmation the TEE module indeed embeds the TETRA cryptographic primitives, albeit in an encrypted form. As such, we now have a clear path forward. We will first gain code execution on the ARM core, then move laterally to obtain a similar foothold on the DSP. There, we will either have to break the TEE implementation or exploit a flaw in the TEE module code.

2.2 Initial foothold

Since the SoC supports secure boot, gaining arbitrary code execution is not trivial. However, the device has an AT modem command interface over a serial link, exposing a reasonably large attack surface. Through static analysis we identified a format string vulnerability (CVE-2022-26941), which we successfully exploited to subsequently gain arbitrary code execution on the ARM core.

We then needed to extend our control over the device to include DSP code execution. While the L138 has *Memory Protection Units (MPUs)* and *I/O Protection Units (IOPUs)* that can be configured to prevent writes from one core to the other's designated memory areas, these units were left unconfigured, not enforcing any restrictions (CVE-2022-27813). Hence, gaining code execution on the DSP core was as straightforward as overwriting parts of the DSP firmware in RAM.

The C6748 DSP has a notion of a *privilege* level: code runs in either *user* or *supervisor* mode. Orthogonal to privilege level, there is the notion of a *security* level. Code runs in either *non-secure* or *secure* mode, where non-secure code is prevented from accessing secure code and memory [18]. This two-dimensional privilege space allows for compartmentalization of confidential code and/or key material.

The DSP main firmware runs in *non-secure supervisor* mode. In case an action is to be performed that requires secure mode execution, a *secure kernel* API call is invoked. This transfers control to the secure kernel (contained in ROM and provided by TI), which runs in *secure supervisor* mode. The secure kernel handles the request and then passes back control to the non-secure code.

As stated, the TEE allows for run-time loading of mod-

[3] See: https://github.com/MidnightBlueLabs/TETRA_burst

[4] See: https://www.motorolasolutions.com/en_xu/products/tetra/devices/mtm5000_series.html#resources

[5] See: https://www.ti.com/product/OMAP-L138

[6] See: https://www.ti.com/product/TMS320C6748

ules. This mechanism is performed through the SK_LOAD[7] secure kernel API call. The provided module is then decrypted, signature checked, and copied to a secure portion of the address space (i.e. inaccessible in non-secure mode). Subsequently, code contained within a loaded module can be invoked through the SK_ALGOINVOKE API call. The sought-after TETRA cryptographic primitives are loaded and invoked through this mechanism. With the previously obtained ability to run arbitrary code in non-secure supervisor mode, we can make direct invocations of SK_LOAD and SK_ALGOINVOKE.

The limited public documentation covering the module loading mechanism states that the module is encrypted with AES-128, which gives us a foothold for analysis of the SK_LOAD primitive.

2.3 Cache-timing side channel

Primitives Caches may introduce timing differences in AES computations which results in a potentially exploitable side channel, as was shown in [1]. The documentation teaches us that the C6748 offers fine-grained cache control functionality: one can force the eviction of any chosen memory range from cache with 64-byte granularity [16, 17]. It turns out that this primitive can be used in non-secure supervisor mode, but also affects secure portions of the address space. On top of that, the cache can be *'frozen'*, i.e. a cache hit results in a speed advantage, but a cache miss will not result in the cache being updated. Both primitives combined offer a solid foundation for a cache-timing side channel attack.

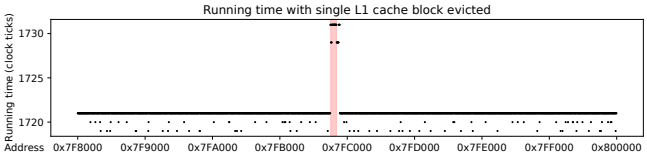


Figure 1: Impact on average SK_LOAD running time while evicting single lines of the protected ROM area from cache. The inverse AES S-box location is highlighted in red.

The secure kernel and AES implementation reside in a secure ROM page, and cannot be read in non-secure mode. In order to carry out the key recovery attack described below, we first need to locate the AES S-box in this secure page.

To do so, we feed arbitrary data to SK_LOAD, which results in an error but not without first decrypting the provided module header using AES. We do so repeatedly while varying which block to evict from cache. For each block, we measure the average time it takes to execute our request in number of clock cycles. Figure 1 depicts the results. Clearly, the large consecutive area that incurs running time penalties when evicted is where the S-box is located.

Now that the location of the S-box is known, we can use our cache primitives to partially evict the S-box, and subsequently freeze the cache. It turns out that the start of the S-box is located halfway a 64-byte eviction block. As such, we can conveniently use this property to evict the first 32 bytes of the S-box from cache.

Key recovery Before describing the attack in detail, we briefly introduce some notation. rk denotes the first round key used during the AES decryption. Likewise, ct denotes the ciphertext under our control. Furthermore, rk_0 denotes the first round key byte, i.e. $rk = \{rk_0, rk_1, \ldots, rk_{15}\}$. Finally, $\oplus$ denotes the bitwise XOR operator.

In the scenario outlined above, each S-box lookup that falls within the first 32 entries induces a running time penalty, whereas the other lookups do not. This property can be used to learn information about rk. The first step of the AES decryption is AddRoundKey, which takes $ct_0 \oplus rk_0$ and stores the result in its internal state. The next step is InverseShiftRows, which changes the order of the internal state bytes. Since we are measuring the running time in aggregate over the entire AES decryption, the order in which operations occur has no effect on our measurements. Finally, we reach the InverseSubBytes step, in which the S-box lookup is performed. Thus, we encounter a running time penalty in case $ct_0 \oplus rk_0 < 32$. Of course, since in total 176 S-box lookups are performed, we must ensure that our lookup of interest 'stands out' from the others. We achieve this by taking a statistical approach: when targeting rk_0, we randomize all bytes of ct except ct_0 for each SK_LOAD invocation, and measure the average running time for each choice of ct_0. This way, running time penalties do occur during S-box lookups beyond the one of our interest, but they do not occur *consistently*, as whether a penalty is incurred ultimately depends on random data. As such, the average running time for these lookups converges to a baseline value.

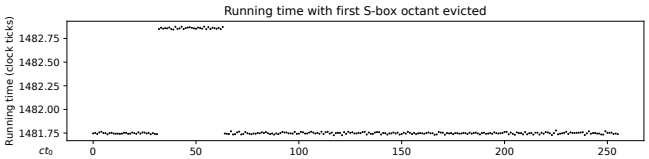


Figure 2: Running time per value for ct_0. Other bytes of ct are randomized for each run. Averaged over 8192 runs.

Bringing the above into practice: we feed random ciphertexts to the SK_LOAD API function and compute the average running time over 256 groups of ciphertexts,

[7] Some hints can be found on the module data format, such as on github: https://github.com/alexeicolin/sysbios/blob/master/packages/ti/sk/sk.h

where each group is characterized by the value of ct_0. We obtain the pattern shown in Figure 2. The running times are all centered around a baseline value, except for 32 consecutive values for ct_0, in this case, $32 \leq ct_0 < 64$. Hence, for our example, we obtain the following predicate:

$$ct_0 \oplus rk_0 < 32 \text{ iff } 32 \leq ct_0 < 64$$

Which is equivalent to:

$$(ct_0 \oplus rk_0) \ \& \ 0\times E0 = 0 \text{ iff } ct_0 \ \& \ 0\times E0 = 32$$

This further simplifies to knowledge on the round key:

$$rk_0 \ \& \ 0\times E0 = 32$$

Thus, we have successfully recovered the 3 most significant bits of rk_0. We can apply the same methodology to the remainder of rk as well, allowing us to recover a total of 48 bits of the 128-bit rk.

We subsequently developed a more complex attack targeting the S-box lookup in the second decryption round, that allows for recovery of the full rk. However, since we have already illustrated the feasibility of a key recovery attack, we omit this extended attack from this paper. Having recovered rk, we can easily recover the actual AES key [3]. The vulnerability was registered under CVE-2022-25332.

2.4 Exfiltrating cryptographic primitives

It turns out that distinct keys are used for decrypting the module header and body. The latter key was recovered by mounting the attack once more, this time targeting the body decryption step. Finally, there is one last layer of obfuscation, which is removed by applying a bitwise XOR over each 16-byte block with the cumulative XOR of its predecessors.

At this stage, we have access to the module's protected contents and we are able to study the cryptographic primitives in the form of C674x assembly instructions and write equivalent implementations in C. The process is greatly aided by the fact that we can load the instructions into an L138 development board and invoke each function to generate known-good test vectors. Each MTM5000 series firmware contains at most a single stream cipher. We found firmware images online containing TEA1, TEA2 and TEA3, and hence we recovered all three ciphers. The TAA1 suite was also recovered, since it is needed for TETRA authentication and is present in all MTM5x00 firmwares. The TEA4 algorithm was not recovered, since adoption is low and a TEA4 capable firmware is not offered by Motorola.

3 Background on the TETRA standard

For convenience and readability, we provide a brief outline of several aspects of the TETRA standard relevant for this paper. Additional background relevant for each finding is provided in Section 5. Particularly, the authentication handshake and identity encryption are described in Section 5.4 and 5.3, respectively.

For the purpose of this paper, we assume a *Trunked Mode of Operation (TMO)* network, and distinguish two entities. First, *Mobile Stations (MSes)*, which typically come in the form of a portophone or car radio unit, and second, the *Switching and Management Infrastructure (SwMI)*, which consists of base stations (towers) and the backend network. Besides TMO, TETRA also supports *Direct Mode of Operation (DMO)*, where MSes communicate directly without the need for an SwMI. It typically serves as a fallback mode when the SwMI is unavailable. However, in this paper, we only concern ourselves with TMO.

3.1 Protocol layers

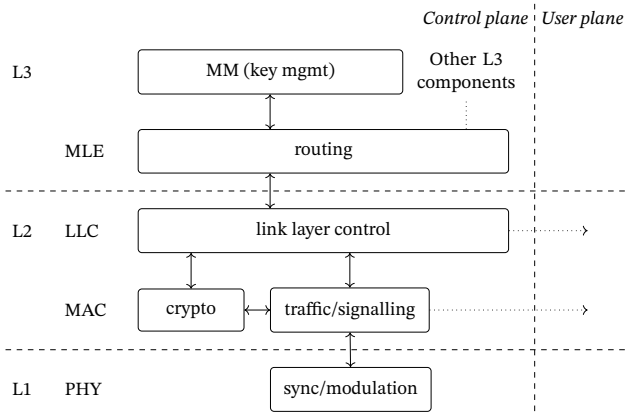


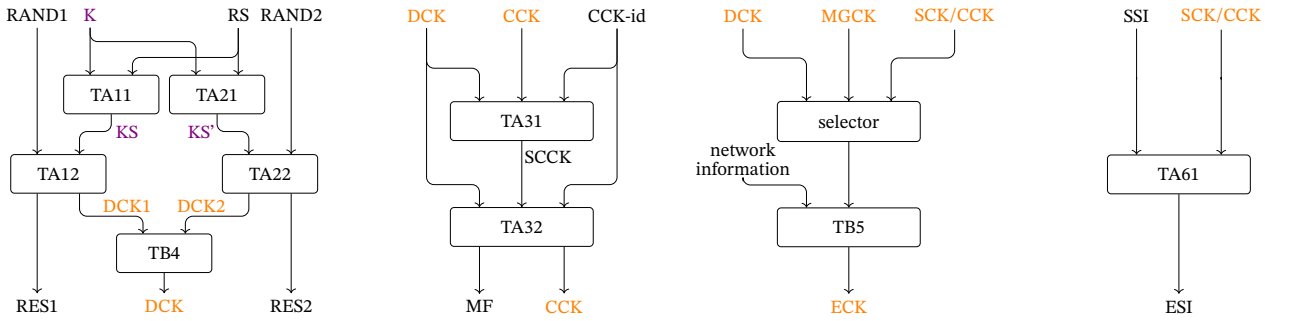
Figure 3: Simplified overview of TETRA components across the OSI model.

In Figure 3, we present a simplified overview of the *Control Plane* of the TETRA protocol, which implements the functional core of TETRA. (Voice) traffic and non-signaling data are transferred to user applications (referred to as the *User plane*) through various (out-of-scope) interfaces. Cryptographic keys are managed in the *Mobility Management (MM)* component on OSI layer 3, and the keys are used by the cryptographic primitives in the layer 2 lower MAC for encryption/decryption of signaling, traffic and identities.

3.2 Security classes and keys

TETRA employs a set of cryptographic keys, for various purposes. An overview of keys, and how they relate to the TETRA primitives, is shown in Figure 4.

Three security classes are defined, each characterized by a set of mandatory and optional features. An overview



(a) Authentication (b) CCK key sealing (c) ECK derivation (d) Identity encryption

Figure 4: Relations between keys and primitives. 128-bit keys in purple, 80-bit keys in orange. Adapted from [12].

is given in Table 1. Class 1 networks support no encryption and optional authentication. Class 2 networks encrypt signaling, voice and data using a single key: the *Static Cipher Key (SCK)*, whereas class 3 networks use individual session keys: the *Derived Cipher Key (DCK)* yielded by the authentication handshake (Section 5.4), and a shared *Common Cipher Key (CCK)*. Group conversations are supported through *talk groups*. Each talk group may optionally have a *Group Cipher Key (GCK)* associated with it. The GCK is not used directly. Rather, it is used as input to algorithm TA71 ([12] clause 4.2.2), together with the SCK (class 2) or CCK (class 3) to generate the *Modified Group Cipher Key (MGCK)*. In case the GCK for a group is not defined, the SCK or CCK is used instead. *Over The Air Re-keying (OTAR)* functionality, albeit largely out of scope, allows for new key material to be provided through the network. Confidentiality of key distribution is protected by means of *sealing*, i.e. encryption of the key with added redundant data to ensure integrity. Figure 4b illustrates this for a CCK update; transmitted as *Sealed Common Cipher Key (SCCK)* to the MS and subsequently unsealed.

As mentioned before, the actual air interface encryption is performed in the lower MAC layer by one of four KSGs, TEA1 through TEA4. The key is modified with public network-specific information through function TB5, yielding the *Encryption Cipher Key (ECK)* (Figure 4c) before being fed to the KSG.

Class	Auth	Encryption	OTAR	TEA Keys
1	Optional	Unavailable	Unavailable	-
2	Optional	Mandatory	Optional	SCK, MGCK
3	Mandatory	Mandatory	Mandatory	DCK, CCK, MGCK

Table 1: Security classes in TETRA, see [12] clause 1.1

3.3 End-to-end encryption

End-to-end encryption is an optional feature in TETRA that introduces an additional layer of encryption for voice and data, offering a secure channel between sender and recipient. Standardization is limited to design guidelines covering replay protection, IV synchronization and key/algorithm selection [8]. Several proprietary and often mutually incompatible solutions exist, with different form factors (SIM card, hardware add-on module, software), algorithms (at least AES and IDEA) and key management procedures. Since end-to-end encryption is not fully standardized by ETSI, and implementations are exceedingly hard to come by for independent review, we cannot make thorough assessments on security guarantees offered by such solutions.

4 Recovered cryptographic primitives

In this section we provide an overview of the recovered TETRA cryptographic primitives. We discuss the general structure and noteworthy design features.

4.1 The TAA1 suite of primitives

The TAA1 suite of algorithms is used for authentication, key derivation and OTAR[8]. After recovery and analysis of TAA1, we found that all primitives defined as TAxx in the standard rely on a proprietary block cipher called HURDLE, discussed in Section 4.2. Furthermore, primitives labeled TBx are non-cryptographic transformation functions.

4.2 The HURDLE block cipher

HURDLE is a balanced 16-round Feistel network with a 128-bit key and a 64-bit block size[9]. Each round key (96

[8] See: https://github.com/MidnightBlueLabs/TETRA_crypto/blob/main/taa1.c

[9] See: https://github.com/MidnightBlueLabs/TETRA_crypto/blob/main/hurdle.c

bits are used per round) is derived from its predecessor, through a simple linear function that wraps around in 16 rounds. This allows for on-the-fly round key generation in both the encryption and decryption direction, enabling efficient hardware implementations.

The round function consists of key-mixing, substitution, and permutation steps utilizing a single $\mathbb{F}_{2^8} \rightarrow \mathbb{F}_{2^8}$ S-box and a single $\mathbb{F}_{2^{32}} \rightarrow \mathbb{F}_{2^{32}}$ P-box. We managed to identify the S-box generator as an index-swapped version of the classic multiplicative inverse based Nyberg S-box [21].

4.3 Stream ciphers

As can be observed from the diagrams below, all three stream ciphers have a similar LFSR-based design. They consist of two registers: a *key register*, which is initialized with the key, and a *state register*, which is initialized with the IV. Both registers perform byte-wise shifting. The key register is fed only with data generated from the register itself, and thus always produces the same key-dependent output stream, independent of the IV. All three ciphers employ two non-linear filter functions (denoted as Fxx), generating one byte of output from two state bytes. One of these filter outputs is mixed in the middle of the state register, and the other with the state feedback. Furthermore, another state byte is fed to a bit-reordering function (denoted as Rx), and mixed in with the feedback as well. Finally, the leftmost state byte is taken directly as a keystream byte. Note that this happens only every 19 rounds, with an additional number of warm-up rounds for the first keystream byte. The KSGs all share the same overall structure, which to the best of our knowledge, is somewhat unconventional. S-boxes and filter functions differ but have similar structures and properties. We have not been able to find serious weaknesses in the KSGs, with the exception of the deliberately weakened TEA1 cipher.

The TEA1 stream cipher

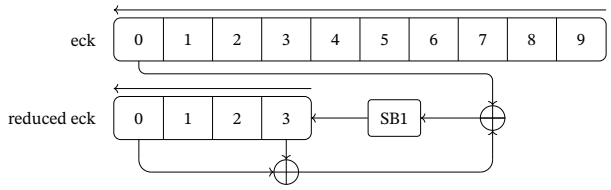


Figure 5: TEA1 key register initialization

Figure 6 presents a schematic overview of the TEA1 key stream generator[10]. Its design differs from the other KSGs in its key register, which consists of only 32 instead of 80 bits. The 80-bit key is fed through a compression function (Figure 5), the output of which initializes the TEA1 key register. This effectively reduces the TEA1 key entropy to 32 bits, and constitutes an obviously deliberate weakening of the cipher. The matter is discussed in more detail in Section 5.2.

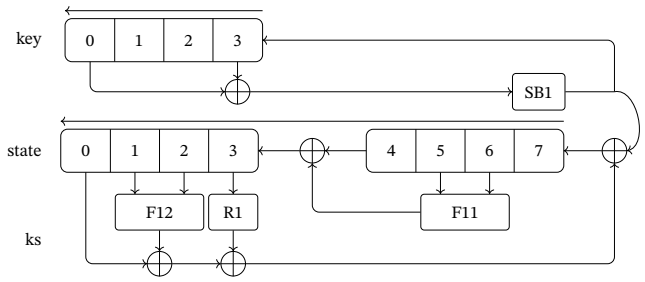


Figure 6: The TEA1 key stream generator, using the reduced key from Figure 5

The TEA2 stream cipher

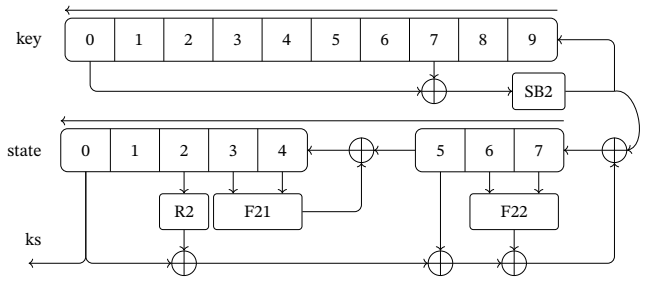


Figure 7: The TEA2 key stream generator

The TEA2 cipher resembles the general pattern described in the beginning of this section[11], except for one detail. It mixes state byte 5 into the state feedback byte.

The TEA3 stream cipher

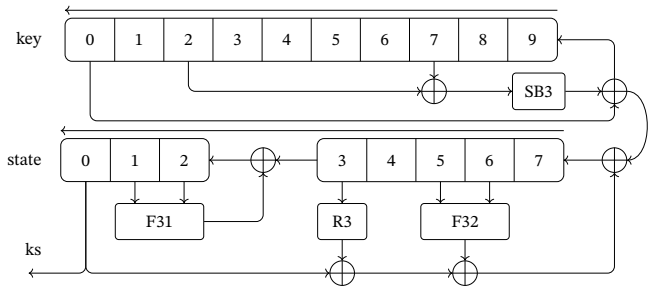


Figure 8: The TEA3 key stream generator

[10] See: https://github.com/MidnightBlueLabs/TETRA_crypto/blob/main/tea1.c

[11] See: https://github.com/MidnightBlueLabs/TETRA_crypto/blob/main/tea2.c

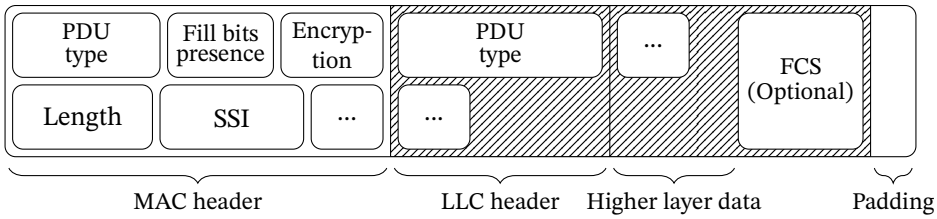


Figure 9: Data format for downlink air interface messages. The shaded portions are encrypted.

The TEA3 S-box is peculiar in the sense that it maps two distinct inputs to the same output[12], which we further discuss in Section 5.5. Interestingly, TEA3 has an additional property that is not shared with TEA1 and TEA2: the S-box output is mixed with a key register byte before being fed back. This effectively prevents key register state merges due to the duplicate S-box entry.

5 Findings

In this section, we discuss the findings identified throughout our research in detail. Section 5.1 explains that the IV provided to the keystream generator ultimately depends on broadcast, unencrypted and unauthenticated data elements. This can be abused to provoke keystream re-use regardless of KSG (CVE-2022-24401, CVE-2022-24404). Section 5.2 describes a real-time passive key recovery attack against the TEA1 stream cipher, which has been deliberately weakened by design (CVE-2022-24402). Section 5.3 describes a weakness in the identity encryption scheme, allowing one to de-anonymize MSes (CVE-2022-24403). Section 5.4 explains that, under certain circumstances, an attacker impersonating the infrastructure can successfully complete an authentication with an MS, establishing a session key (DCK) of all zeroes (CVE-2022-24400). Lastly, we mention a peculiarity regarding the TEA3 S-box in Section 5.5. Future research is required in order to assess whether this has detrimental effects on the security guarantees provided by TEA3.

5.1 Keystream re-use

Background

Data format Figure 9 depicts the data format for downlink messages over the air interface. The *Medium Access Control (MAC)* header is sent in cleartext and contains basic information such as the message's destination *Short Subscriber Identity (SSI)*, a bit indicating whether the message is encrypted, and the length of the message in bytes. Bit-granular message lengths can be achieved by toggling the *fill bits* flag, indicating that the message is padded. The *Logical Link Control (LLC)* header contains information such as the link layer type (see below), presence of the *Frame Check Sequence (FCS)*, a CRC32-based checksum, whether or not the message should be acknowledged, and whether the message is/contains an acknowledgment of a previous message. All these properties are encoded in the LLC PDU type (see [13] clause 21.2). Depending on the PDU type, additional information may be included in the LLC header as well. Note that the LLC PDU type is distinct from the MAC PDU type. If present and valid, the FCS is stripped away before the message is passed to handler functions pertaining to higher layers of the protocol. As such, seen from the perspective of these higher layer handler functions, presence or absence of the FCS is equivalent. As briefly mentioned, there is a distinction between link types. Two are supported, *Basic Link (BL)* and *Advanced Link (AL)*. The FCS is required under Advanced Link, but is optional under Basic Link. All services of our interest use the Basic Link, and as such, we will not concern ourselves with the Advanced Link.

Encryption While often cryptography is implemented at 'higher' protocol layers, in TETRA, the IV definition is bound to MAC-level frame counters and as such, air interface encryption needs to be performed at the MAC layer.

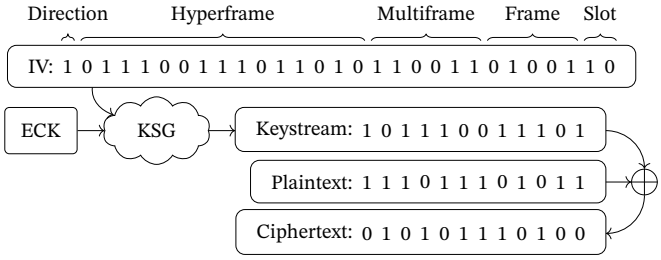


Figure 10: Encryption of signaling, voice and data.

The purpose of the IV is to provide distinct information to the stream cipher each time it is invoked, in order to produce a different keystream each time. As illustrated in Figure 10, the IV is comprised of a direction bit, indicating

[12] See: https://github.com/MidnightBlueLabs/TETRA_crypto/blob/main/tea3.c

a downlink or uplink message, a hyperframe, multiframe, frame and slot number. The numbers are all incremented sequentially (except the direction bit). Under normal circumstances, these numbers combined have a periodicity of about 23 days[13]. Re-keying should thus occur at least once during such a period, since keystream re-use may occur beyond that point.

Air Interface integrity guarantees The TETRA specification mandates the use of a 16-bit CRC on air interface frames. This checksum is computed over the ciphertext and is purely meant to detect transmission errors. Besides this, redundancy included with OTAR-sealed key material and the previously discussed FCS, no cryptographic integrity checks are present. Rather, integrity is established based on whether the data 'makes sense' after decryption.

Vulnerability assessment

Binding the IV to the temporal dimension should ensure each IV occurs only once, and thus, that each keystream is unique. We refer to a combination of hyperframe, multiframe, frame, and timeslot number as a *timestamp*

We may assume that properly configured networks update their key material within the aforementioned 23-day window. This alone, however, is insufficient in preventing keystream re-use, as it is also imperative that the synchronization of the timestamp between MS and/or SwMI cannot be affected by third parties. TETRA fails to enforce this. The hyperframe number is provided to the MS through the SYSINFO PDU, whereas the multiframe, frame and slot number are provided via the SYNC PDU. Once adopted by the MS, these numbers are maintained (i.e. incremented) internally by the MS. SYSINFO and SYNC PDUs are neither encrypted nor authenticated, and need to be frequently broadcast by the SwMI in order to allow for proper network selection and operation by mobile stations. ETSI's documentation covering the air interface [13] clause 23.6.2 and 23.6.3 states that the MS shall update its internal bookkeeping upon reception of a SYSINFO or SYNC PDU that conflicts with expected frame counter values.

Due to the direction bit incorporated in the IV, keystream recovered from the uplink cannot be used to decrypt downlink messages and vice versa. A means of recovering keystream in each direction is described below.

Recovering uplink keystream Suppose we would like to decrypt encrypted message $c = \mathrm{KSG}(t, \mathrm{ECK}) \oplus m$, which was previously sent at timestamp t. Assuming cryptographic keys have not been updated, we may impersonate the SwMI and trick the MS into re-using the IV that was employed at time t. We do so by sending SYNC and SYSINFO frames with a timestamp set to slightly precede t. At this point, when the MS transmits a message $c' = \mathrm{KSG}(t, \mathrm{ECK}) \oplus m'$ at the spoofed time t, this message is encrypted with the same keystream as the target message m. As such, any knowledge on the contents of m' directly translates to knowledge of m. Oftentimes, the length of m', together with the way it fits within an observed communication stream, will be very typical for certain types of communication. As such, we can quite often establish (parts of) m' with high confidence.

Recovering downlink keystream The perception of time of the SwMI is beyond our control. As such, obtaining downlink keystream is less straightforward than for the uplink case. As we will demonstrate, downlink keystream recovery is still possible by sending ciphertexts to the MS and discerning information based on its behavior in response.

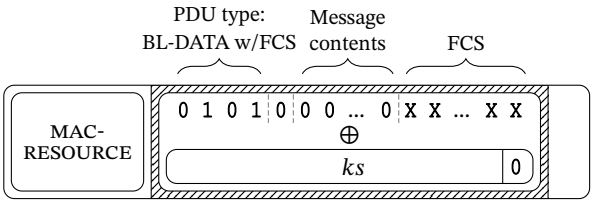


Figure 11: Message c as used in keystream expansion. FCS follows from message contents. Correctly guessing the last keystream bit triggers acknowledgment by the MS.

Suppose that we possess known keystream ks at time t, and that we would like to expand that knowledge. We refer to this concept as *keystream expansion*. To do so, we construct a message m, situated in the LLC layer (see Data format), with the same bit length of ks *plus one*. We choose *BL-DATA with FCS* as its PDU type, indicating presence of the FCS checksum and requiring acknowledgment upon correct reception. The contents of m itself are irrelevant and filled with zeroes. Finally, message m is concluded with the FCS checksum computed over its contents. We construct ks' by appending a zero bit to ks, i.e. $ks' = ks \parallel 0$. Furthermore, we construct encrypted message c by taking $c = m \oplus ks'$. A schematic view of c is given in Figure 11. By sending spoofed SYNC and SYSINFO frames, we set the timestamp of the MS such that it slightly precedes t, and send message c at exactly time t. Depending on whether the FCS is correct, and by extension whether the newly added zero bit in ks' is the correct keystream bit, the MS either emits an acknowledgment, or silently discards the message. In either case, we learn a keystream bit. By continuously repeating the procedure, we eventually learn the entire downlink keystream at time t.

A prerequisite for the above procedure is a known

[13] [13] clause 4.5.1 describes 14.167ms per time slot, 4 slots per frame, 18 frames per multiframe, 60 multiframes per hyperframe. Hence, 2^{15} hyperframes make for a periodicity of ~23.21 days

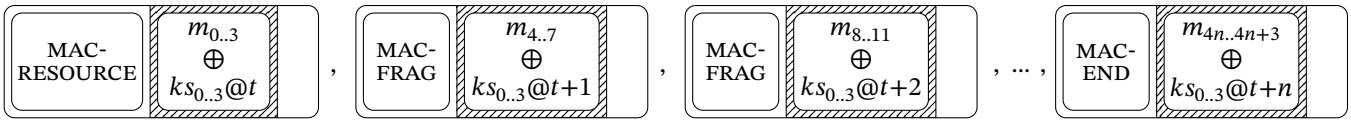


Figure 12: Single message m divided over $n+1$ four-bit fragments. $m_{i..j}$ denotes bits i to j of m.

keystream ks long enough to cover m, which must contain at least a 4-bit PDU type, a 1-bit *TL-SDU number* (which is replicated in the acknowledgment), and the 32-bit FCS. Hence, we need ks to be at least 37 bits long. However, as we discuss below, ks need not be a consecutive keystream segment. Rather, we can leverage TETRA MAC-layer fragmentation support and divide it into multiple fragments, where each fragment is encrypted with keystream belonging to a different timestamp.

Suppose that we send an encrypted LLC PDU to the MS of exactly 5 bits. The first four bits indicate the *PDU type*. Depending on this value, the MS expects a certain minimum PDU length, which in most cases exceeds 5 bits. All messages for which this is the case will be silently discarded. Among the leftover PDU types is *BL-DATA without FCS*, which expects a minimum length of 5 bits and requires acknowledgment upon reception. This is the only valid 5-bit PDU type that causes the MS to issue a response. As such, we iterate over all possible values of the first 4 bits, where, on each iteration, we reset the MS's timestamp to t by means of spoofed SYNC and SYSINFO frames, and determine which of these values triggers a response[14]. In the message causing the response to be issued, the first four bits are now known to decrypt to the value corresponding to the *BL-DATA without FCS* PDU type, and thus, we have just recovered the first four bits of keystream at time t.

Next, we use this strategy to recover the first 4 keystream bits for 10 consecutive frames. Once obtained, we divide the previously discussed message m over fragments of 4 bits each. The first fragment is encapsulated in a MAC-RESOURCE PDU, with the length field set to a special value indicating *start of fragmentation*. Due to absence of the actual message length, the MS expects it to cover the entire slot. However, the length can be cut down to 4 bits by setting the fill bits flag and padding the remainder of the slot. Subsequent 4-bit fragments are encapsulated in MAC-FRAG PDUs, whereas the last fragment is contained in a MAC-END PDU. Figure 12 depicts a diagram of the concept. At this point, upon reception of all the fragments, the MS decrypts and reconstructs m from the fragments. In case the FCS is determined to be correct, the MS will emit an acknowledgment. As such, we can now apply the keystream expansion technique described previously on any timestamp between t and t+10 to fully recover its corresponding keystream.

[14]The fifth bit (TL-SDU number) does not affect whether or not an acknowledgment is sent

Group communication Traffic destined for a talk group is not addressed to individual MSes by the SwMI. Rather, the talk group has its own subscriber identity, similar to the concept of multicast in IP networks. In addition to its own individual subscriber identity, an MS subscribed to a talk group also processes incoming data destined for a group subscriber identity, and decrypts it using the associated key. There is no principle argument pertaining to why the downlink keystream recovery attack outlined above should not work in the context of talk groups. However, the attack relies on frames being acknowledged by the MS, which normally never occurs in a group context.

Experimental results In order to validate the presented attack in practice, two viable approaches exist. Either, we have to implement a TETRA base station stack on a *Software Defined Radio (SDR)*, or, we leverage the stack on a real-world TETRA base station and instrument it to exhibit the required behavior. Since implementing a TETRA infrastructure stack is a tremendous engineering effort, we opted for the latter approach.

As such, we procured a Motorola MBTS TETRA base station. Unfortunately, it came without support for air interface encryption. We found that the included firmware contains all the necessary prerequisites, except for the actual stream cipher, which is implemented as an empty stub function. We proceeded by injecting arbitrary read/write/execute primitives in the firmware image and replacing the stub with the TEA1 stream cipher. Doing so is straightforward as the image is not cryptographically signed. Then, we built a small framework on top of these read/write/execute primitives allowing us to compile C code to an ELF executable, and load it as a module into the MBTS at runtime. Functionality for redirecting code flow from the firmware to a replacement implementation was also introduced. This served as an excellent foundation for experimentation. Notably, the framework was used to insert key material in the MBTS, which can normally only be done by means of a Motorola *Key Variable Loader (KVL)* device. The MBTS was set to use security class 2, and the same key material was provided to our MTM5400. Finally, the MTM5400 successfully registered with the network and operated as normal. Analysis of downlink frames with OsmocomTETRA confirmed that the traffic is indeed encrypted.

By overriding the procedure in the firmware that feeds data to the transmission hardware, we gained the ability to inject arbitrary messages. By injecting spoofed SYNC and

SYSINFO frames, we found that the MTM5400 can indeed be persuaded to arbitrarily update its internal frame counters, and by extension, to re-use keystream.

We proceeded by implementing the bootstrap and keystream expansion attack described earlier in this section in the form of a base station console command. As such, the MBTS acts as a usual TETRA base station, with normal progression of network time. Upon entering the console command, however, the attack code starts tampering with network time, sends out spoofed messages, and interprets acknowledgments sent by the MS (or absence thereof) in order to infer keystream. We confirmed that the attack works reliably in practice on the MTM5400, recovering the entire downlink keystream belonging to any desired timestamp. We found the MTM5400 also acknowledges frames destined for a talk group. Consequently, we tailored our attack to target group subscriber identities, and confirmed successful keystream recovery for group-encrypted traffic as well. We are not aware of any deviations between the MTM5400's implementation and the TETRA standard and therefore expect most if not all other MS models to be equally susceptible to the attack.

In a real-world scenario, an attacker will likely opt for an SDR implementation of the attack, rather than a repurposed TETRA base station. Furthermore, the attacker needs to perform a 'takeover' of the legitimate SwMI signal, for example, by overpowering it. The ability to overpower an existing TETRA network was not demonstrated experimentally. However, given the attacker's ability to use high-gain directional antennas in areas with low infrastructure reception, as well as evidence by prior research in this area [25], this should be relatively straightforward.

By carefully aligning with the legitimate SwMI's time slots, an MS should not be able to distinguish the attacker's signal from that of a legitimate SwMI. In the hypothetical situation where a signal takeover is infeasible without causing the MS to enforce a re-authentication, the authentication handshake can simply be relayed to the legitimate SwMI. In class 2 networks, this has no effect on the efficacy of the attack. In class 3 networks, this would trigger a change of the DCK, effectively preventing keystream recovery of unicast traffic. However, keystream pertaining to group-encrypted traffic can still be recovered.

Mitigations

In order to counter the keystream re-use vulnerability while maintaining compatibility with the existing infrastructure, MS firmwares may be updated to perform basic sanity checks over incoming frame counters, such as detecting repeated frame counter decrements. Depending on the strictness of such checks, the attack may be impeded beyond practical feasibility. In our communication with stakeholders, we have been informed that firmware updates have been developed that indeed implement such sanity checks. However, it must be noted that not all models will receive patches, and that until the last vulnerable MS in a network is either updated or replaced, the network as a whole very much remains vulnerable to this attack.

As highlighted in Section 3.3, end-to-end encryption is proprietary and hard to obtain and analyze, and we therefore cannot vouch for its security. However, it is likely to be an adequate mitigation strategy against the keystream re-use issue. Alternatively, in case the network predominantly carries packet data, less opaque and more generic solutions may also be opted for, such as VPN tunnels, or serial shield encrypters[15].

5.2 Weak TEA1 stream cipher

Background

According to the specification [12], all four air interface encryption algorithms (TEA1, TEA2, TEA3, TEA4) use 80-bit keys, DCK, SCK, CCK, or MGCK, modified with public network-specific information by function TB5 to yield an 80-bit *Encryption Cipher Key (ECK)* for use on the air interface (Figure 4c). The KSG IV is constructed from frame numbers and an uplink/downlink bit (see Section 5.1).

Vulnerability assessment

Although TEA1 takes an 80-bit ECK, the cipher's initialization compresses it to fit a 32-bit key register. Besides the IV, the generated keystream subsequently only depends on this value, and as such we refer to it as the *reduced ECK*.

The compression function is shown schematically in Figure 5. The attack outlined below can be carried out passively, and allows for full decryption of real-time, future and previously captured traffic, and allows for encryption of forged messages.

Validating a correctly guessed key While a brute-force attack is easily implemented, one still needs a means of determining whether or not a key guess is correct. The most obvious method of doing so involves gathering keystream by guessing (partial) cleartext message contents based on its context, such as call setup signaling, which generally precedes encrypted traffic streams. Since the reduced ECK holds 32 bits of entropy, knowledge of 32 bits of keystream is typically enough to uniquely determine the correct reduced ECK. However, there is a more systematic approach. In order to increase transmission reliability, certain signaling messages are transmitted several times, such as messages related to channel allocation. Analysis of live TETRA traffic shows that such retransmissions are frequent and easily

[15] A device that provides transparent encryption over serial link

identified by looking for four messages with the same destination, MAC header and length, transmitted at a constant 1-frame interval. Since the repeated messages are sent at different timestamps, their corresponding keystreams are different. However, assuming that the cleartext of these messages are identical, they should yield the same message after decryption. By computing keystreams for all repetitions of the message, we can distinguishing correct from incorrect key guesses by verifying whether the messages are indeed identical after decryption.

Recovering full keys A reduced ECK may be leveraged into obtaining the full 80-bit DCK, SCK, CCK or MGCK. Due to the modification through TB5, each channel (i.e. uplink/downlink frequency for a single cell) uses a different ECK, and thus a reduced ECK allows for encryption and decryption on a single channel only, whereas the full key is valid across the entire network. Furthermore, in a class 3 network, recovery of a DCK allows us to unseal intercepted CCK updates received from the SwMI, and to forge sealed CCK updates for an MS.

The compression function compresses the 80-bit ECK to 32 bits such that there are 2^{48} candidate ECKs for each reduced ECK. Due to reliance on linear operations, we can efficiently generate these 2^{48} pre-images. Furthermore, TB5 is trivially invertible given the (public) network information elements. As such, we can mount an attack with a complexity of 2^{48} if we have a means of efficiently distinguishing correct from incorrect key guesses. One such means is to recover three different reduced ECKs from different cells or carrier frequencies. For each of them, the network information passed to TB5 is different. The full 80-bit key is uniquely determined by iterating over each pre-image of the first reduced ECK, applying the inverse of TB5, and checking whether the resulting key yields the other two reduced ECKs after applying TB5 again (with different network information parameters) and the compression function.

Experimental results We implemented a proof-of-concept in OpenCL for both attacks presented in this section. Running on an NVIDIA GTX 1080 GPU, the search space is exhausted in approximately 52 seconds for the reduced ECK recovery, and 7 minutes for the full key recovery.

Mitigations

The TEA1 weakness cannot be resolved while retaining backwards compatibility. We would like to see ETSI lift their restrictions on the use of TEA2, and recommend TEA1 end users to migrate to TEA2. Given the fate of TEA1 paired with the fact that TEA1 and TEA4 share the same target audience designation, we strongly advise caution when considering migration to TEA4 prior to any public scrutiny.

Since migrating a network from one cipher to another requires either replacement or firmware updating of all devices, without option for a (secure) transition period, migration will prove infeasible in many scenarios. As with the keystream re-use issue, one could opt for end-to-end encryption instead. For networks mainly carrying packet data, a VPN tunnel or serial shield encrypters can also be considered.

Research collision *After successful extraction of the TETRA cryptographic primitives, we were informed that researchers from the Ruhr University Bochum had obtained TEA1 and TEA2 from an anonymous source. They had equally identified the issue in TEA1, but did not intend to publish.*

5.3 Weak identity encryption scheme

Background

In order to establish which MS traffic is intended for, downlink traffic includes a 24-bit destination *Short Subscriber Identity (SSI)*, analogous to the MSIN (part of the IMSI) in GSM terminology. This might be an *Individual Short Subscriber Identity (ISSI)*, or a *Group Short Subscriber Identity (GSSI)*, for individual and group-addressed traffic, respectively. Other identity types exist but are out of scope for this paper.

For all encrypted TETRA traffic, the destination address is protected by an identity encryption scheme based on algorithm TA61 [12] (Figure 4d). The primitive takes the 80-bit SCK (class 2 networks) or CCK (class 3), which we refer to as k, and the 24-bit SSI and generates a 24-bit *Encrypted Subscriber Identity (ESI)*.

First, k is transformed into a 64-bit *intermediate secret* c using the HURDLE block cipher in a construction resembling a one-way function. The 24-bit SSI is then encrypted with c using a minimalistic two-round encryption scheme, of which a diagram is given in Figure 13. Subfunction TRID represents a permutation function.

Publication restrictions *Due to a pending coordinated disclosure process (see Section 8), we currently refrain from publishing TA61 in full detail. We do, however, provide the outline of a meet-in-the-middle attack, in order to enable informed discussion on the security of the primitive. The derivation of* c *and the structure of* TRID *are deliberately left out.*

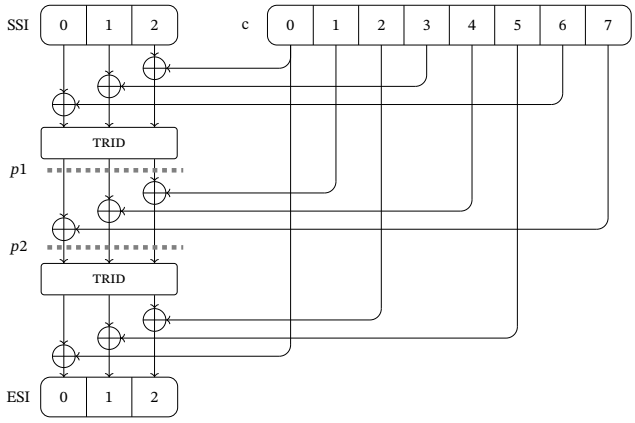


Figure 13: Transformation of identity SSI using c as TA61 intermediate secret. The points $p1$ and $p2$ represent the attack forward and backward phase products, respectively.

Vulnerability assessment

The HURDLE block cipher is used solely in transforming k into c, and the SSI is involved in the encryption process only *after* computation of c. Thus, TA61 does not fully benefit from the security guarantees offered by HURDLE.

We present a meet-in-the-middle attack[16] that recovers c given three valid (SSI, ESI) pairs. Meet-in-the-middle attacks are a well-understood [4,19,32] class of time-memory tradeoff cryptographic attacks against schemes that perform multiple encryption steps in succession, where a different (sub)key is used for each step. As can be seen in Figure 13, TA61 follows such a structure.

Our attack consists of a forward phase, taking an unencrypted SSI and computing $p1$ (Figure 13) for all 2^{24} valuations of c_0, c_3, c_6. For each of these, we perform a backward phase (involving TRID^{-1}) computing $p2$ for all values of c_2 and c_5, given the ESI. Byte c_0 follows from the forward guess.

For each (SSI, ESI) pair, we thus compute all possible valuations for $p1$ and $p2$. As can be seen in the figure, these relate to c_1, c_4, c_7 as follows:

$$(c_1, c_4, c_7) = (p1_2, p1_1, p1_0) \oplus (p2_2, p2_1, p2_0)$$

Guessing five bytes of c thus yields us the remaining three bytes. Given two (SSI, ESI) pairs, we obtain 2^{16} valuations for $(c_0, c_2, c_3, c_5, c_6)$ which yield identical, consistent values for (c_1, c_4, c_7): candidates for the actual value of c. With a third pair, we obtain a single candidate. As such, we can recover c using three (SSI, ESI) pairs, with a complexity of 2^{40}.

The forward step has a complexity of 2^{24}, while the backward step complexity is only 2^{16} due to the re-use of c_0. This yields a total complexity of 2^{40} with no significant memory requirement.

Note that c is re-used for identity encryption as long as k does not change, and as such, finding c allows us to encrypt and decrypt all identifiers until the SCK or CCK is changed.

Experimental results Identifiers are unencrypted during authentication and encrypted thereafter. As such, an (SSI, ESI) identity pair becomes known when we passively monitor an authentication and subsequently encounter an encrypted identity not previously seen. Hence, obtaining three identity pairs is straightforward and only takes minutes on a reasonably sized network. The process can also be sped up by actively interfering with signals, provoking MSes to perform a cell reselection and subsequently re-authenticate.

We implemented a proof-of-concept of the attack in OpenCL. The program takes three identity pairs and finds c. Running on an NVIDIA GTX 1080 GPU, the search space is exhausted in approximately 16 seconds.

Mitigations

Since identity encryption is an essential and mandatory part of the TETRA standard, modifying TA61 would break compatibility with current versions of the standard. One alternative mitigation could involve preventing one from obtaining any knowledge on (SSI, ESI) identity pairs. We see no viable non-protocol-breaking way of realizing such a mitigation. As such, modification of TA61 in a future TETRA standards revision seems to be the only way forward.

The design of TA61 suggests it was designed to support both encryption and decryption of identities, and avoid collisions. Using the unencrypted identity as (part of) the input for HURDLE would improve the cryptographic robustness. However, the 64-bit output would somehow need to be compressed down to 24 bits, for which we see no possibility without violating the stated requirements. Rather, we propose an amendment: first, we increase the size of c to 128 bits in order to have a stronger intermediate secret. Second, we expand the procedure to involve 6 rounds instead of the current 2. This way, a practical meet-in-the-middle attack is no longer possible, as the added entropy of c raises the computational complexity of deriving c from identity pairs beyond feasibility.

5.4 Session key pinning vulnerability

Background

The TETRA standard defines message flows for authentication in ETSI 300 392-7 clause 4.1 [12]. An MSC of the

[16] Not to be confused with a man-in-the-middle attack. A good introduction on the meet-in-the-middle attack was presented by Verdult in [33].

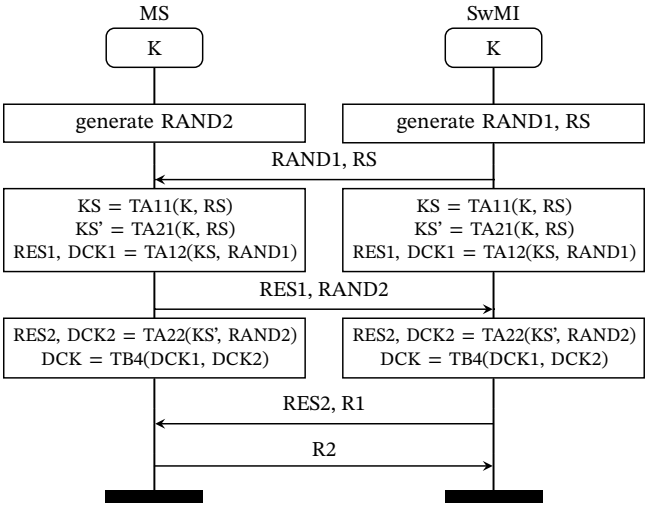


Figure 14: MSC of TETRA mutual authentication. The MS and SwMI share a secret key K. R1 and R2 are Boolean values indicating the handshake validity. On completion, SwMI and MS share session key DCK.

message flow for mutual authentication is given in Figure 14, while key derivations are depicted in Figure 4a. Since we focus on air interface communications, the distinction between SwMI and authentication centre is omitted for readability.

In order to authenticate the MS to the SwMI and vice versa, MS and SwMI share a secret key K. Random RS is generated by the SwMI, and both are fed to the TA11 and TA21 primitives, yielding KS and KS', respectively. These intermediate keys are combined with the challenges RAND1 and RAND2, generated by the SwMI and MS respectively, using algorithms TA12 and TA22. These algorithms yield the responses RES1 and RES2, proving knowledge of the secret K. Additionally, TA12 and TA22 yield partial session keys DCK1 and DCK2. The SwMI signals whether RES1 was accepted by setting bit R1, while the MS signals RES2's validity with bit R2. If both are accepted, the session is authenticated and session key DCK is derived from DCK1 and DCK2 using primitive TB4.

Vulnerability assessment

We can deduce a definition for session key DCK from the authentication mechanism as described above and visualized in Figure 4a. We obtain:

$$\begin{aligned}\text{DCK} \;=\; &\text{TB4}(\\ &\quad \text{TA12}(\text{TA11}(\text{K},\text{RS}),\text{RAND1}),\\ &\quad \text{TA22}(\text{TA21}(\text{K},\text{RS}),\text{RAND2})\\ &)\end{aligned}$$

However, having obtained the implementation of the TAA1 primitives, we can simplify the equation even further. We found that TB4 is simply a bitwise XOR operator, that TA12 and TA22 are identical, and that TA11 and TA21 are very similar; the difference being that TA21 reverses the byte order of RS prior to performing the same operation as TA11. Hence, the following definition also captures DCK:

$$\begin{aligned}\text{DCK} \;=\; &\text{XOR}(\\ &\quad \text{TA12}(\text{TA11}(\text{K},\text{RS}),\text{RAND1}),\\ &\quad \text{TA12}(\text{TA11}(\text{K},\text{reverse}(\text{RS})),\text{RAND2})\\ &)\end{aligned}$$

Suppose that we are impersonating the SwMI. We can thus freely choose RS, including a palindrome, i.e. a value for RS satisfying reverse(RS) = RS. Furthermore, suppose that we can accurately predict the value of RAND2 sent by the MS. Then, by choosing RAND1 = RAND2, the DCK will equal the following:

$$\begin{aligned}\text{DCK} \;=\; &\text{XOR}(\\ &\quad \text{TA12}(\text{TA11}(\text{K},\text{RS}),\text{RAND2}),\\ &\quad \text{TA12}(\text{TA11}(\text{K},\text{RS}),\text{RAND2})\\ &)\end{aligned}$$

Clearly, due to the XOR cancellation property, this simplifies to 0 and hence, we obtain an all-zero DCK. Also, under the portrayed circumstances, RES1 = RES2. Since we receive RES1 from the MS before having to provide RES2, we can send the correct response without knowing K.

Hence, the integrity of session key DCK depends entirely on the unpredictability of RAND2. However, this crucial dependency is not reflected in any of TETRA's specifications and only becomes apparent when details of the cryptographic primitives are revealed. Given the track record of the security of random number generation [15, 34] on embedded systems, it is likely that a significant share of radios deployed worldwide are vulnerable. In fact, by investigating the PRNG present the MTM5400, we found that it is among the set of vulnerable radios, as its random number generator solely relies on the clock tick register as its entropy source (CVE-2022-26943). Furthermore, the emitted RAND2 reveals information about the device's internal random pool, which can be used for a subsequent prediction attempt.

Carrying out this attack yields an authenticated session with an MS, using an all-zero DCK, without knowledge of K. Note that this does not allow the attacker to decrypt traffic between the MS and the legitimate SwMI.

Mitigations

It is worth noting that, if the authentication handshake were designed slightly differently, a degree of predictability of RAND2 need not have compromised the DCK. For instance, instead of using XOR, TB4 could have been based on an operator that does not have the cancellation property, such as addition with carry. Fortunately, mitigations can be applied that preserve compliance to the protocol. For

Section	CVE	Severity	Description
Section 5.1	CVE-2022-24401	Critical	Keystream re-use
Section 5.1	CVE-2022-24404	High	Malleability of messages
Section 5.2	CVE-2022-24402	Critical	Weak cryptographic primitive TEA1
Section 5.3	CVE-2022-24403	High	Weak identity encryption scheme
Section 5.4	CVE-2022-24400	Low	Session key pinning vulnerability
Section 5.5	-	Unclear	TEA3 S-box is not a permutation

Table 2: Summary of findings presented in this paper.

example, by ensuring that RAND1 = RAND2 never holds, e.g. by enforcing RAND2 to be generated anew if it does.

5.5 TEA3 S-box

Background

The TEA3 keystream generator is intended for public safety organizations that operate outside the geographical boundaries to which the use of TEA2 is restricted, and therefore is widely adopted in non-EU countries around the world such as India, China and Mexico.

Vulnerability assessment

Similar to TEA1 and TEA2, a separate key register is used to create a stream of key-derived bytes. The key register is updated by means of an S-box (see Section 4.3). Remarkably, and contrary to best practice, we found the TEA3 S-box is not a permutation. Instead, both index 0×14 and 0×9E map to the same value, 0×C2, while the value 0×D2 does not occur. Additionally, the duplicate and missing entries 0×C2 and 0×D2 only differ by a single bit. After performing preliminary cryptanalysis on the TEA1 and TEA2 S-boxes, we found that when the first 0×C2 entry in the TEA3 S-box is substituted by 0×D2 certain properties of the resulting S-box (such as algebraic degree, nonlinearity, and linear potential) more closely align with those of TEA1 and TEA2. This suggests that the duplicate entry may be considered an accidental bit-flip, with unclear implications. Whether this was by design or through error is currently unknown.

To the best of our knowledge, the property contrasts with all S-boxes found in scientifically scrutinized and as-of-yet unbroken ciphers. As such, the TEA3 peculiarity is to be taken seriously. We leave in-depth cryptanalysis of TEA3 as future work.

Mitigations

The TEA3 KSG is a part of the TETRA standard that cannot be changed without breaking compatibility with previous versions of the standard. As such, pending further reassurance on the security of TEA3, we believe use of the algorithm should be discouraged.

6 Conclusions

During the research that forms the basis of this paper, we set out to demonstrate the feasibility of recovering the secret cryptographic primitives underpinning TETRA through reverse engineering, and provide the first public in-depth analysis of these primitives and TETRA as a whole. As elaborated in the introduction (Section 1), the process of recovering the primitives is far from trivial, but possible nonetheless. The secrecy of TETRA's cryptographic primitives has resulted in a now widely adopted technology with severe weaknesses. The results uncovered are cause for serious concern. Table 2 contains an overview. In summary:

- Keystream re-use issues (Section 5.1) exist affecting all ciphers, resulting in loss of confidentiality and integrity. While the issues could have been inferred from publicly available materials, the proprietary and inaccessible nature of TETRA equipment significantly complicates practical attack validation and subsequent public disclosure.
- The TEA1 cipher (Section 5.2), used by commercial and public safety organizations around the world, is deliberately weakened to the extent that it offers little to no protection, resulting in loss of confidentiality and integrity. The secretive nature of the ciphers impedes proper assessments of the level of security.
- Flawed identity encryption and authentication protocol design issues (Sections 5.3 and 5.4) exist that could lead to de-anonymization of users, illegitimate authentication and session establishment with confidentiality implications. These attacks only become apparent when one is in possession of the cryptographic primitives.
- The S-box of the TEA3 cipher (Section 5.5), used by public safety organizations around the world, was found not to be a permutation, which is highly unusual. The precise implications are still unclear, but preliminary analysis suggests that it is not the result of a deliberate hardening effort. Again, this issue only becomes apparent when one is in possession of the TEA3 cipher.

We believe that the security posture of TETRA is significantly worse than what has been common perception until now. Particularly worrisome is that this situation has existed for multiple decades, with little to no means for security analysts and asset owners to either address or even become aware of such issues.

7 Future Work

Although several issues with high impact were uncovered, we believe TETRA is far from exhausted as a research area.

For example, a security analysis of currently deployed end-to-end encryption solutions would be a major contribution, especially since a considerable share of TETRA users will consider end-to-end encryption as a mitigation pathway for the issues described in this paper.

Furthermore, the structure of the uncovered ciphers is somewhat unconventional and additional in-depth analysis is needed. In particular TEA3 should be critically evaluated so that the discussion on whether its users should migrate can be put to rest. In addition, recovery and analysis of the TEA4 cipher may be valuable since, like TEA1, it is approved for worldwide commercial use, and thus may also offer less cryptographic strength than advertised.

Finally, ETSI has announced future TETRA security enhancements[17] including additional authentication and encryption algorithms which ought to resist cryptanalysis into the 2030s and beyond, indicating an envisioned multi-decade future for TETRA. These algorithms will again be considered confidential, perpetuating the use of secret cryptography in sensitive contexts. As such, we believe heightened scrutiny of these future enhancements is warranted.

8 Ethics and Responsible Disclosure

In December, 2021, we have informed the Dutch NCSC of our findings. Ever since, we have been in close collaboration with ETSI, manufacturers and government agencies in the context of a coordinated disclosure process. Due to the high impact and often sensitive nature of the communications, the embargo period was initially set at a year and later extended in agreement with key stakeholders in order to allow for large-scale patch deployment operations, mitigating some of the highlighted issues. While we always insisted on full publication of both the primitives and the vulnerabilities we uncovered, in the interest of some stakeholders, we have agreed to temporarily withhold publication of the TA61 identity encryption primitive. We will publish the full TA61 primitive in December 2023.

[17] See: https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=57516

Acknowledgments

This research was supported by the NLnet Foundation[18] with financial support from the European Commission's Next Generation Internet programme [19]. We would like to thank Christian Veenman and the Dutch NCSC for their extensive support and collaboration during the coordinated disclosure process.

References

[1] Daniel J Bernstein. Cache-timing attacks on AES. 2005.

[2] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In *International Workshop on Fast Software Encryption*, pages 1–18. Springer, 2000.

[3] William R. Cordwell. AES key recovery from round keys. 11 2008.

[4] Hüseyin Demirci and Ali Aydın Selçuk. A meet-in-the-middle attack on 8-round aes. In *Fast Software Encryption: 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers 15*, pages 116–126. Springer, 2008.

[5] Benedikt Driessen, Ralf Hund, Carsten Willems, Christof Paar, and Thorsten Holz. Don't trust satellite phones: A security analysis of two satphone standards. In *2012 IEEE Symposium on Security and Privacy*, pages 128–142. IEEE, 2012.

[6] Shuwen Duan. Security analysis of TETRA. Master's thesis, Institutt for telematikk, 2013.

[7] Shuwen Duan, Stig Frode Mjølsnes, and Joe-Kai Tsay. Security analysis of the terrestrial trunked radio (TETRA) authentication protocol. 2013.

[8] ETSI. Terrestrial trunked radio (TETRA); security; synchronization mechanism for end-to-end encryption. *ETSI ES 202 109 V1.1.1*, 2003.

[9] ETSI. Rules for the management of the TETRA standard authentication and key management algorithm set TAA1. *ETSI TS 101 052 V2.1.1*, 2016.

[10] ETSI. Confidentiality and restricted usage undertaking relating to the TAA1 algorithm. 2018.

[11] ETSI. Rules for the management of the TETRA standard encryption algorithms; part 2: TEA2. *ETSI TS 101 053-2 V2.5.1*, 2018.

[18] See: http://www.nlnet.nl

[19] See: https://www.ngi.eu

[12] ETSI. Terrestrial trunked radio (TETRA); voice plus data (v+d); part 7: Security. *ETSI EN 300 392-7 V3.5.1*, 2019.

[13] ETSI. Terrestrial trunked radio (TETRA); voice plus data (v+d); part 2: Air interface (ai). *ETSI EN 300 392-2 V3.8.1*, 2020.

[14] Ian Goldberg, David Wagner, and Lucky Green. The (real-time) cryptanalysis of A5/2. *Rump session of Crypto*, 99:16, 1999.

[15] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Presented as part of the 21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 205–220, 2012.

[16] Texas Instruments. OMAP-L138 C6000 DSP+ARM processor datasheet (rev. j) (SPRS586J), 2009.

[17] Texas Instruments. TMS320C674x DSP megamodule reference guide (SPRUFK5A), 2010.

[18] Texas Instruments. TMS320C6748/OMAP-L138 security user's guide (SPRUGQ9), 2011.

[19] Ralph C Merkle and Martin E Hellman. On the security of multiple encryption. *Communications of the ACM*, 24(7):465–467, 1981.

[20] Floriano Neto, Jorge Granjal, and Vasco Pereira. A survey on security approaches on PPDR systems towards 5G and beyond. *IEEE Access*, 2022.

[21] Kaisa Nyberg. Perfect nonlinear S-boxes. In *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, pages 378–386. Springer, 1991.

[22] Jonas Olofsson. Design and implementation of SIM functionality for TETRA-system on a smart card, 2012.

[23] Yong-Seok Park, Choon-Soo Kim, and Jae-Cheol Ryou. The vulnerability analysis and improvement of the TETRA authentication protocol. In *2010 The 12th International Conference on Advanced Communication Technology (ICACT)*, volume 2, pages 1469–1473. IEEE, 2010.

[24] Martin Pfeiffer, Jan-Pascal Kwiotek, Jiska Classen, Robin Klose, and Matthias Hollick. Analyzing TETRA location privacy and network availability. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 117–122, 2016.

[25] Christina Pöpper, Nils Ole Tippenhauer, Boris Danev, and Srdjan Capkun. Investigation of signal and message manipulations on the wireless channel. In *Computer Security–ESORICS 2011: 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings 16*, pages 40–59. Springer, 2011.

[26] Paul Reuvers and Marc Simons. Operation RUBICON) the secret purchase of Crypto AG by BND and CIA, 2020.

[27] Thomas Rudolph. *Analyzing Security-related Signals Using Software defined Radio*. PhD thesis, Citeseer, 2013.

[28] Security and InteroperabiLity in Next Generation PPDR CommUnication InfrastructureS. Security and privacy analysis of TETRA / TETRAPOL PPDR, LTE and Wi-Fi networks, 2014.

[29] Ray R Tanuhardja, Stefan van de Beek, Mark J Bentum, and Frank BJ Leferink. Vulnerability of terrestrial-trunked radio to intelligent intentional electromagnetic interference. *IEEE transactions on electromagnetic compatibility*, 57(3):454–460, 2015.

[30] Marek Sebera Tomáš Suchan. TETRA networks security, 2015.

[31] Stefan van de Beek and Frank Leferink. Robustness of a TETRA base station receiver against intentional EMI. *IEEE transactions on electromagnetic compatibility*, 57(3):461–469, 2015.

[32] Paul C Van Oorschot and Michael J Wiener. Improving implementable meet-in-the-middle attacks by orders of magnitude. In *Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings*, pages 229–236. Springer, 2001.

[33] Roel Verdult. *The (in) security of proprietary cryptography*. PhD thesis, [Sl: sn], 2015.

[34] J. Wetzels and A. Abbasi. Wheel of fortune - analyzing embedded OS random number generators. 33C3, 2016.

[35] David J Wheeler and Roger M Needham. TEA, a tiny encryption algorithm. In *Fast Software Encryption: Second International Workshop Leuven, Belgium, December 14–16, 1994 Proceedings 2*, pages 363–366. Springer, 1995.

[36] Zhi-Hui Zhang and Yi-Xian Yang. Research on end-to-end encryption of TETRA. *The Journal of China Universities of Posts and Telecommunications*, 13(2):70–73, 2006.