An Experimental Security Analysis of Two Satphone Standards

BENEDIKT DRIESSEN, Ruhr-University Bochum
RALF HUND, Ruhr-University Bochum
CARSTEN WILLEMS, Ruhr-University Bochum
CHRISTOF PAAR, Ruhr-University Bochum
THORSTEN HOLZ, Ruhr-University Bochum

General purpose communication systems such as GSM and UMTS have been in the focus of security researchers for over a decade now. Recently also technologies that are only used under more specific circumstances have come into the spotlight of academic research and the hacker scene alike. A striking example of this is recent work [Driessen et al. 2012] that analyzed the security of the over-the-air encryption in the two existing ETSI satphone standards GMR-1 and GMR-2. The firmware of handheld devices was reverse-engineered and the previously unknown stream ciphers A5-GMR-1 and A5-GMR-2 were recovered. In a second step, both ciphers were cryptanalized, resulting in a ciphertext-only attack on A5-GMR-1 and a known-plaintext attack on A5-GMR-2.

In this work, we extend the afore-mentioned results in the following ways: First, we improve the proposed attack on A5-GMR-1 and reduce its average case complexity from 2^{32} to 2^{21} steps. Second, we implement a practical attack to successfully record communications in the Thuraya network and show that it can be done with moderate effort for approx. \$5 000. We describe the implementation of our modified attack and the crucial aspects to make it practical. Using our eavesdropping setup, we recorded 30 seconds of our own satellite-to-satphone communication and show that we are able to recover Thuraya session keys in half an hour (on average). We supplement these results with experiments designed to highlight the feasibility of also eavesdropping on the satphone's emanations.

The purpose of this paper is threefold: Develop and demonstrate more practical attacks on A5-GMR-1, summarize current research results in the field of GMR-1 and GMR-2 security, and shed light on the amount of work and expertise it takes from setting out to analyze a complex system to actually break it in the real world.

Additional Key Words and Phrases: Mobile Security; Satellite Phone Systems; Cryptanalysis; Binary Analysis; Real-world Attack; Stream Cipher

ACM Reference Format:
Driessen, B., Hund, R., Willems, R., Paar, C., Holz, T.. 2012. An Experimental Security Analysis of Two Satphone Standards. ACM Trans. Info. Syst. Sec. 16, 3, Article 10 (November 2013), 30 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Mobile communication systems have revolutionized the way we interact with each other. Instead of depending on landline connections with fixed locations, we can talk to other people wherever we are and also transmit data from (almost) arbitrary locations. Especially the *Global System for Mobile Communications* (GSM) has evolved into an extremely large-scale system; with more than four billion subscribers in 2011, it is the

This work has been supported in part by the Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia (Grant IV.5-43-02/2-005-WFBO-009).

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

most widely deployed standard for cellular networks. Many other cellular network standards like *Universal Mobile Telecommunications System* (UMTS), *CDMA2000* (also known as *IMT Multi-Carrier* (IMT-MC)), or *3GPP Long Term Evolution* (LTE) exist and are continuously enhanced to meet growing customer demands.

Cellular mobile networks require a so-called *cell site* to create a cell within the network. The cell site provides all the infrastructure necessary for exchanging radio signals between mobile handsets and the provider network. For example, a typical cell site contains one or more sets of transmitters/receivers, antennas, digital signal processors to perform all computations, a GPS receiver for timing and other control electronics. The cells within a network have only a limited operating distance and, thus, a certain proximity to a cell site is always necessary to establish a connection to the mobile network.

In practice, however, it is not always possible to be close to a cell site and there are many use cases in which no coverage is provided. Workers on an oil rig or on board of a ship, researchers on a field trip in a desert or near the poles, aid workers in remote areas or areas that are affected by a natural disaster, journalists working in politically unstable areas, or certain military and governmental undertakings are a few of many uses cases where terrestrial cellular networks are not available. To overcome this limitation, satellite systems were introduced that provide telephony and data services based on telecommunications satellites. In such systems, the mobile handset (typically called *satphone*) communicates directly with satellites in orbit. Thus, coverage can be provided without the need of a highly interconnected infrastructure on the Earth's surface.

There are two major satphone protocol families, standardized by the European Telecommunications Standards Institute (ETSI), that were both developed in the past few years:

— *Geostationary Earth Orbit (GEO) Mobile Radio Interface* (better known as GMR-1) is a family of ETSI standards that were derived from the terrestrial cellular standard GSM. In fact, the specifications of GMR are an extension of the GSM standard, where certain aspects of the specification are adjusted for satphone settings. This protocol family is supported by several providers (e.g., Thuraya, SkyTerra, TerreStar) and has continuously undergone several revisions to support a broader range of services.

— The *GMR-2* family is even closer to GSM. It deviates from the GMR-1 specifications in numerous ways, most notably the network architecture is different.

The specification documents of GMR-1 and GMR-2 are available online, but do not provide any information about implementation details of security aspects. More precisely, it was not publicly known which encryption algorithms are actually used to secure the communication channels between a satphone and a satellite. Since an attacker can easily eavesdrop on the radio signals between satphone and satellite, even at some distance, it is obvious that weak encryption would be a serious threat to confidentiality. At this point, it was thus unclear what effort would be needed by an attacker to actually intercept telephony and data services for common satphone systems.

In this paper, we build on our previous work [Driessen et al. 2012], which reverse-engineered the stream ciphers A5-GMR-1 and A5-GMR-2, used in the respective standards. In contrast to the original publication, we focus less on the process of reverse-engineering itself, but rather collect all relevant information regarding security and configuration aspects of the system: We describe the network architecture of satellite telecommunication systems and, to some degree, how they operate on the physical and protocol level. We describe the architecture of the satphones themselves, the ciphers we found for GMR-1 and GMR-2 and our results of cryptanalyzing them. We improve the complexity of attacking A5-GMR-1 by a factor of 2^{11} due to targeting a different channel and exploiting the fact that frame numbers and initial states in A5-GMR-1 are linearly related. Latter property allows us to mount an attack that uses multiple speech data frames (instead of only one control channel frame), which leads to less guessing when the attack is performed. This considerable improvement enables us to mount an attack on the Thuraya network, for which we proceed to describe the hardware and software requirements for an actual attack. We discuss crucial aspects when implementing and executing an eavesdropping attack, experimentally establish the feasibility of direct uplink interception and ultimately show that GMR-1 privacy is practically non-existent.

Effectively, we thus demonstrate that current satphone systems are vulnerable to eavesdropping attacks; the results of this paper can be used to build an interceptor for satellite telecommunication systems.

2. BACKGROUND AND RELATED WORK

We now introduce the background information necessary to understand the basics of satellite telephone systems (with a focus on GMR-1), their security mechanisms and the architecture of the mobile handsets. More information about these topics can be found in the literature [Wright 1995; Matolak et al. 2002; ETSI 2001a; Maral and Bousquet 2009; Jim Geovedi and Raoul Chiesa 2011]. Furthermore, we discuss related work in this area.

2.1. Network Layout

Thuraya implements the GMR-1 standard and provides satellite telephony for most of Europe, the Middle East, North, Central and East Africa, Asia and Australia. To achieve this coverage, the network consists of two overlapping regions, each handled by a different satellite. Thuraya satellites are operating in Geosynchronous Orbit (GSO), where they do not stay on a position but follow a fixed movement pattern, typically an analemma. Currently, there are two operational[1] satellites named Thuraya-2 and Thuraya-3. The former is relevant here, since it is centered on the Middle East and supplies most of Europe as well as a large portion of the African continent with connectivity, see Figure 1.

Thuraya offers a diverse range of products for fixed installations, handhelds (i.e., satphones) and even solutions for the maritime environment. With the help of Thuraya, voice, fax and IP-based data can be transmitted where “traditional” infrastructures (e.g., GSM, UMTS, WLAN, etc.) are not available. In addition to the satellites, a set of terrestrial gateways and one primary gateway (located in Sharjah, United Arab Emirates) handle the entire network as depicted in Figure 2. Gateway stations provide the connectivity to tethered networks, e.g., telephone calls to a landline are forwarded to the Public Switched Telephone Network (PSTN) and enable maintenance and configuration purposes. For this so-called *ground segment*, conventional wavelength ($3.400 - 3.625$ GHz and $6.425 - 6.725$ GHz) signals are used. The *user segment* operates on L-band carriers assigned to spotbeams, which are Thuraya’s equivalent to cells in GSM (albeit covering far more area). In the L-band, the frequency band from 1.525 to 1.559 GHz is assigned for downlink (space-to-earth) communication while the uplink (earth-to-space transmissions) operates between 1.6265 and 1.6605 GHz. Uplink and downlink are divided into 1087 paired carrier frequencies with a spacing of 31.25 KHz.

[1]Thuraya-1 has ceased to operate in May 2007 and has been moved to “junk orbit” [TBS 2012].

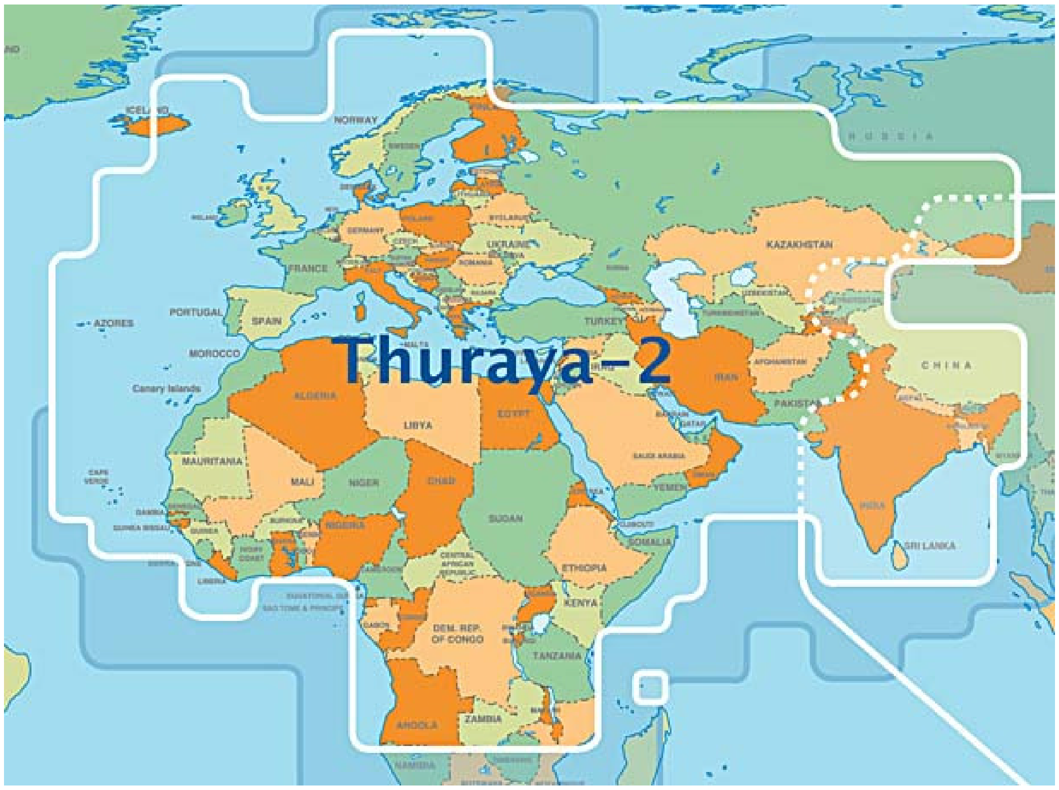

Fig. 1. Network coverage of the Thuraya-2 satellite, [Peter 2013]

2.2. Channels

Just like in GSM, the Time Division Multiple Access (TDMA) time slot architecture is employed which partitions a carrier frequency into disjunct timeslots of a fixed length. Figure 3 shows how a TDMA frame (middle) is split into 24 timeslots (bottom) of $\frac{5}{3}$ ms each. 16 TDMA frames are grouped together into one multiframe (top), which is 640 ms long. Furthermore, multiframes are consolidated into a superframe, of which $4\,496$ comprise a hyperframe. It should be noted that each TDMA frame has a 19-bit TDMA frame number; numbering starts at 0, the number is incremented with each new frame.

Several *logical channels* (called *channels* from now on) can share a carrier frequency by being mapped on different timeslots. Due to this architecture, a channel is uniquely determined by a frequency and a sequence of Timeslot Numbers (TN). There are different types of channels, but all are either Traffic Channels (TCH) for voice, fax or IP-based data, or Control Channels (CCH). Data is sent over these channels in the form of frames (i.e., blocks of consecutive bits), that are encoded (cf. Section 2.3) by adding redundancy to protect against transmission failures. Frames are enumerated by their respective TDMA frame numbers, which we simply call *frame numbers* from now on. For some channels, the encoded data is subsequently encrypted, see Section 2.3. The encoded (and encrypted) data is finally modulated before it is transmitted via the phone's antenna. The encoding scheme differs from channel to channel and is dependent on the respective reliability requirements as defined in the various documents of the standard.

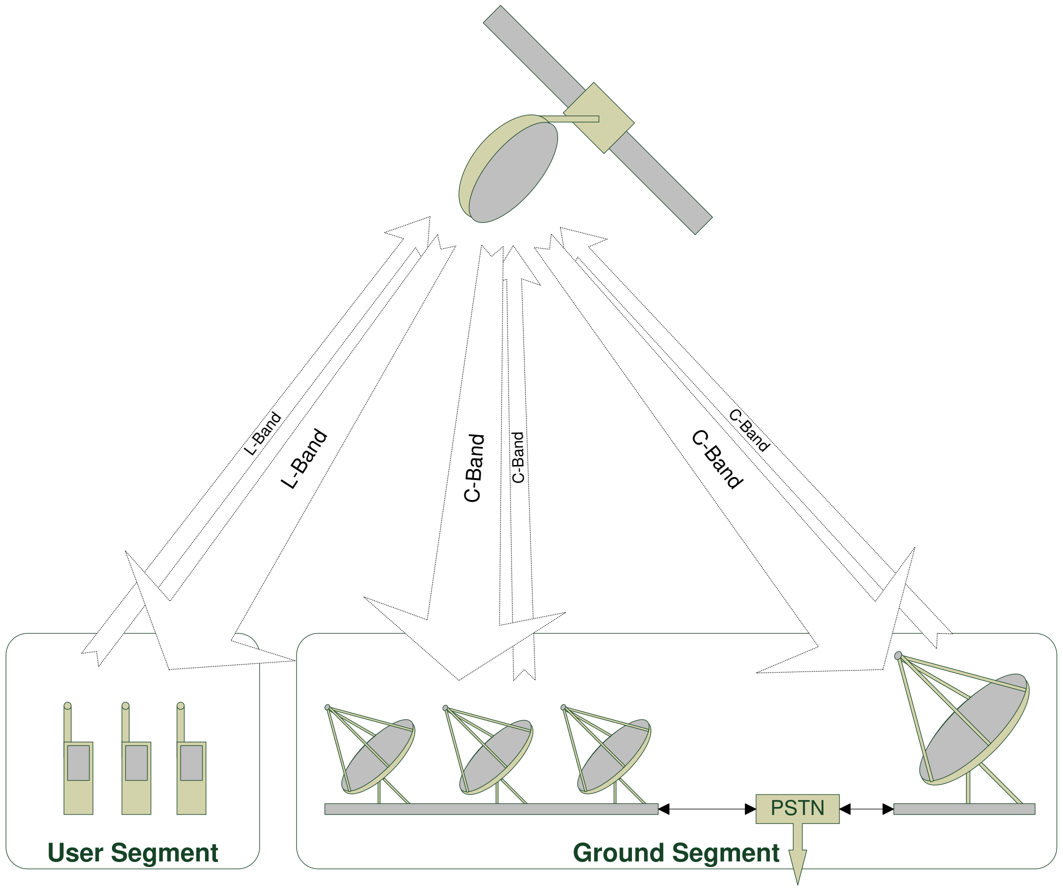

Fig. 2. Layout of the Thuraya network

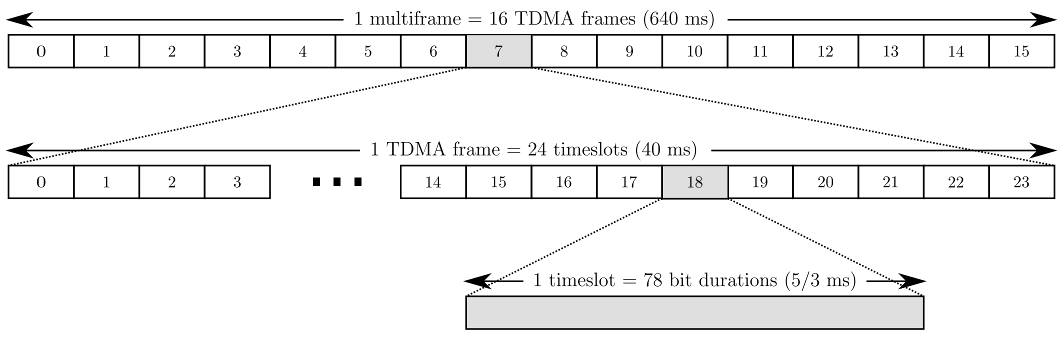
1 multiframe = 16 TDMA frames (640 ms)
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
1 TDMA frame = 24 timeslots (40 ms)
0
1
2
3
14
15
16
17
18
19
20
21
22
23
1 timeslot = 78 bit durations (5/3 ms)

Fig. 3. TDMA architecture of GMR-1 networks

Specific channels relevant for our attack are the Frequency Correction Channel (FCCH), the Common Control Channel (CCCH) and the Traffic Channel-3 (TCH3). The FCCH is initially (e.g., after power up) used by the satphone to determine its relative time and frequency error in order to synchronize with the satellite. The CCCH is used to send information to the phone when a new channel (e.g., TCH3) needs to be established[2]. These assignment messages contain an Absolute Radio-Frequency Channel Number (ARFCN) and a TN, which is, as explained above, all that is required to use the channel. After TCH3 has been set up on the uplink and downlink, it can be used to transmit speech data.

In Section 5.1 we will go more into the process of translating ARFCNs into frequencies and how we can actually tune to the TCH3 channel.

2.3. Encoding and Encryption

As mentioned before, all data has to be encoded before it is sent to travel the distance of around $36\,000$ Km between ground and satellite.

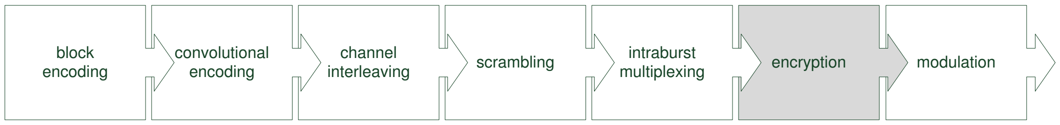

Fig. 4. Generic encoding (and encryption) scheme for information in the GMR-1 system

Encoding always increases the size of the encoded data, thus adding redundancy which allows error detection and possibly correction. Figure 4 shows the multiple encoding steps of which not all are always applied to data, depending on the channel it is sent on. The general encoding procedure is as follows:

> Each channel uses the following sequence and order of operations:
> — The information bits are encoded with a systematic block code, i.e., CRC, building words of information and parity bits;
> — these information and parity bits are encoded with a convolutional code, building the coded bits;
> — the coded bits are reordered and potentially interleaved over multiple bursts;
> — the interleaved bits are scrambled and, in some cases, multiplexed with other bits (before or after encryption);
>
> —[ETSI 2002, p. 11]

To protect against eavesdropping of data sent over the air, the encoded bits are encrypted with a proprietary cipher. However, doing it the way it is done in GMR-1 leads to a property we exploit for our real-world attack (cf. Section 4.5).

Encryption in GMR-1 is enabled on a per-session basis, i.e., for the duration of one call a session key Kc is established (see Figure 5 for a sketch of the respective protocol). This key is derived from a challenge $RAND$ sent by the network and a long-term key Ki, known only to the satphone and network. In the specifications, the key derivation algorithm is denoted as A8, which serves the same[3] role as the A8 function in GSM. On the handheld side it is implemented on the phone's SIM card, where also the corresponding long-term key is stored.

With the help of the session key, data can be encrypted with an algorithm denoted as A5 (shorthand for A5-GMR-1). This algorithm is a stream cipher that encrypts data

[2]TCH3 is typically established at the beginning of a call.
[3]It is possible to use GSM SIM cards for Thuraya's network.

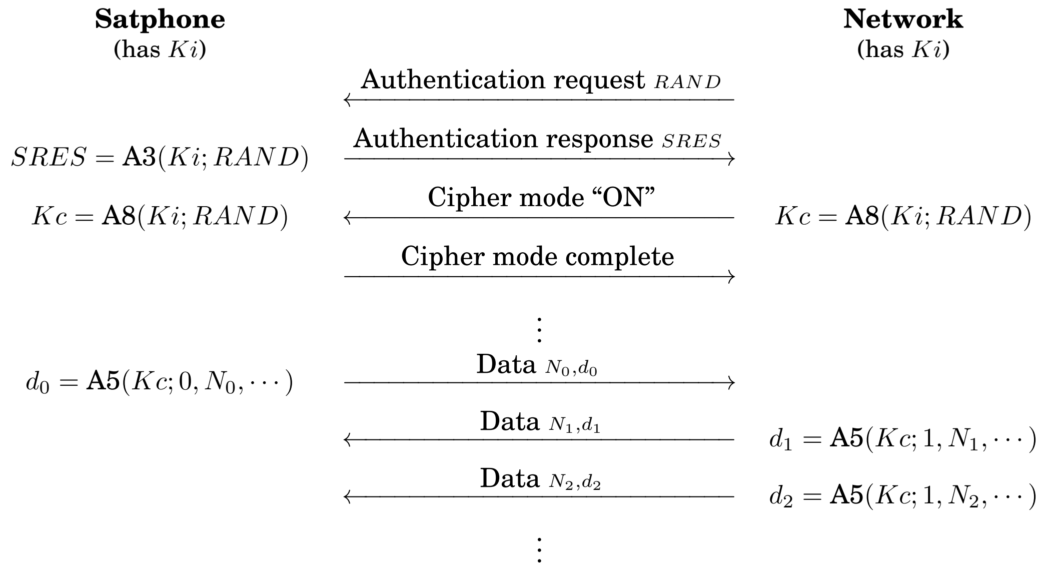
Satphone
(has Ki)
Network
(has Ki)
Authentication request $RAND$
Authentication response $SRES$
$SRES = \mathrm{A3}(Ki; RAND)$
Cipher mode "ON"
$Kc = \mathrm{A8}(Ki; RAND)$
$Kc = \mathrm{A8}(Ki; RAND)$
Cipher mode complete
Data N_0,d_0
$d_0 = \mathrm{A5}(Kc; 0, N_0, \cdots)$
Data N_1,d_1
$d_1 = \mathrm{A5}(Kc; 1, N_1, \cdots)$
Data N_2,d_2
$d_2 = \mathrm{A5}(Kc; 1, N_2, \cdots)$

Fig. 5. Protocol for establishing a session key Kc between satphone and provider network

based on the session key and its TDMA frame number and direction (i.e., whether it is received or sent by a satphone). A second property of the protocol is that it simultaneously authenticates the phone against the network—with the help of the A3 algorithm. Overall, this protocol is strikingly similar to what is implemented in GSM.

2.4. Satellite Telephone Architecture

We now briefly introduce the general architectural structure of satellite phones and the hardware typically found in such devices. In a later section, we provide more details on the specific phones we studied during our analysis, including a discussion of the actual hardware.

In general, the architecture of satellite phones is similar to the architecture of cellular phones [Welte 2010]. Both types of handsets have to perform a lot of processing of speech and signal data, thus they typically ship with a dedicated digital signal processor. Consequently, the afore mentioned, computationally intensive operations are done by the DSP. More relevant for our purpose are the facts that DSPs are also suitable for executing cryptographic algorithms and that encryption is part of the encoding process, which makes DSP code a prime candidate for holding GMR cipher code.

Nevertheless, the core of a satphone is a standard microprocessor (usually an ARM-based CPU) that serves as the central control unit within the system. This CPU initializes the DSP during the boot process. Furthermore, both processors share parts of the main memory or other peripheral devices to implement inter-processor communication. To understand the flow of code and data on a phone, we thus needed to analyze the communication between the two processors.

The operating system running on the main CPU is typically a highly specialized system that is designed with respect to the special requirements of a phone system (e.g., limited resources, reliability, real-time constraints, etc.). All of the software is deployed as one large, statically linked firmware binary which typically contains ARM code mixed with DSP code. For our analysis, we were especially interested in the inter-processor communication functionality provided by the operating system as well as the

DSP initialization routine. This is due to the fact that we needed to extract and transform the DSP code, as it is found in the firmware, into the format actually executed by the DSP.

2.5. Related Work

Satellite telecommunication systems are related to terrestrial cellular systems since the GMR standards are derived from the GSM standard. We can thus leverage work on the analysis of cellular systems for our security analysis as discussed in the following. Briceno *et al.* published in 1999 an implementation of the GSM A5/1 and A5/2 algorithms, which they apparently obtained by reverse engineering a GSM handset [Briceno et al. 1999]. However, no details about the analysis process were ever published and it remains unclear how the algorithms were actually derived. Our analysis is also based on actual implementations of the ciphers; we discuss the general approach in Section 3 and provide analysis details in later sections.

There has been much work on the security analysis of the ciphers used within GSM, e.g., [Golic 1997; Biham and Dunkelman 2000; Biryukov et al. 2000; Ekdahl and Johansson 2003; Bogdanov et al. 2007; Nohl and Paget 2009; Dunkelman et al. 2010]. The cipher used in GMR-1 is related to the A5/2 algorithm, but the configuration of the cipher is different. Our attack on this algorithm builds on existing ideas for A5/2 [Petrovic and Fuster-Sabater 2000; Barkan et al. 2008], which we extended to enable a time-ciphertext trade-off.

3. GENERAL APPROACH

In this section, we outline the general methodology we used for identifying and extracting encryption algorithms from satellite phones. Furthermore, we also discuss the assumptions that helped us during the reverse engineering phase.

We analyzed two representative phones that operate according the two different standards we are interested in. More precisely, we analyzed the firmwares of the following two phones:

— the Thuraya SO-2510 satphone that implements the GMR-1 specification
— the Inmarsat IsatPhone Pro satphone that implements the GMR-2 specification

The starting point of our analysis was the publicly available firmware upgrade of each of these two devices. The entire analysis was performed purely statically since we initially did not have a real satellite phone at our disposal that we could instrument to perform a dynamic analysis. Furthermore, we did not have access to a whole device simulator that enables debugging of arbitrary firmware image, thus we had to develop our own set of analysis tools. However, the ARM code for the main microprocessor (used by both phones) can be partially executed and debugged in a CPU emulator such as QEMU.

The approach we followed to analyze both satphones can be separated into the following five phases:

(1) Obtain the firmware update package and the respective update program (usually a Windows executable).
(2) Extract the firmware image from the package.
(3) Reconstruct the correct memory mappings of the code and data sections in the firmware image.
(4) Identify the DSP initialization procedure in order to extract the DSP code/mapping.
(5) Search for the encryption algorithms in the DSP code using specific heuristics as well as control and data flow analysis techniques.

Several steps can be automated, but some manual analysis is nevertheless required. We successfully applied this method to the two phones we analyzed. In addition, we speculate that also other kinds of satphones can be analyzed in this way.

These basic assumptions helped us to find the relevant pieces of code in a shorter amount of time:

(1) The length of the session key is known.
(2) The length of the keystream, as generated by the cipher, is equal to the length of the encrypted frame.
(3) Since the GMR standards are derived from GSM, the ciphers bear at least some resemblance to the well-known, LFSR-based A5 algorithms.

Actual lengths for first two assumptions can be derived from the publicly available parts of the GMR specifications [ETSI 2001b; 2001c]. These assumptions enabled us to decrease the search space of potential code. The last assumption is rather speculative, but helped us in finding one of the two algorithms (but did not hold for the second cipher).

4. SECURITY ANALYSIS OF GMR-1

We used the Thuraya SO-2510 phone as an example for a handset that operates according to the GMR-1 standard. This decision was solely driven by the fact that the firmware of this satphone is publicly available from the vendor's website. In fact, we did not analyze any other GMR-1 satellite phone, but since the protocol is standardized we are confident that our analysis results apply to all other GMR-1 phones as well.

4.1. Hardware Architecture

The Thuraya SO-2510 runs on a Texas Instruments OMAP1510 platform. The core of the platform is an ARM-925 CPU along with a TI TMS320C5000 signal processor. This information can be deduced from corresponding strings in the binary and from pictures of the actual components soldered on the circuit board [OsmocomGMR 2012]. Figure 6 provides a high-level overview of the architecture.

Both processors can communicate with each other using a special shared peripherals bus. Furthermore, they share the same RAM and can access additional memory (e.g., SRAM or Flash) on equal terms. Initially, DSP code or data has to be loaded by the ARM CPU into the specific memory regions of the DSP. The DSP code can be located in either the on-chip SARAM (which holds 96 Kb of memory) or in the SRAM, which is accessed through the memory interface controller (MIC). Writes to the SARAM region of the DSP are especially interesting for extracting the corresponding DSP code. The official OMAP1510 documents suggest pre-defined memory regions to be used by the ARM-MMU for mapping this memory area [Texas Instruments 2012]. During our analysis, we could confirm that the firmware uses exactly the same mappings.

4.2. Finding the Cipher

We were able to find the cipher A5-GMR-1 in the firmware of a Thuraya SO-2510 phone with the help of IDA Pro by ranking functions in the DSP code according to their percentage of XOR and SHIFT operations. The four topmost functions in this ranking turned out to implement the different linear feedback shift registers (LFSR) of the cipher . The interested reader is referred to the original publication [Driessen et al. 2012] for more details.

4.3. Structure of the Cipher

The cipher used in GMR-1 is a typical stream cipher. Its design is a modification of the A5/2 cipher [Petrovic and Fuster-Sabater 2000; Barkan et al. 2008], which is used

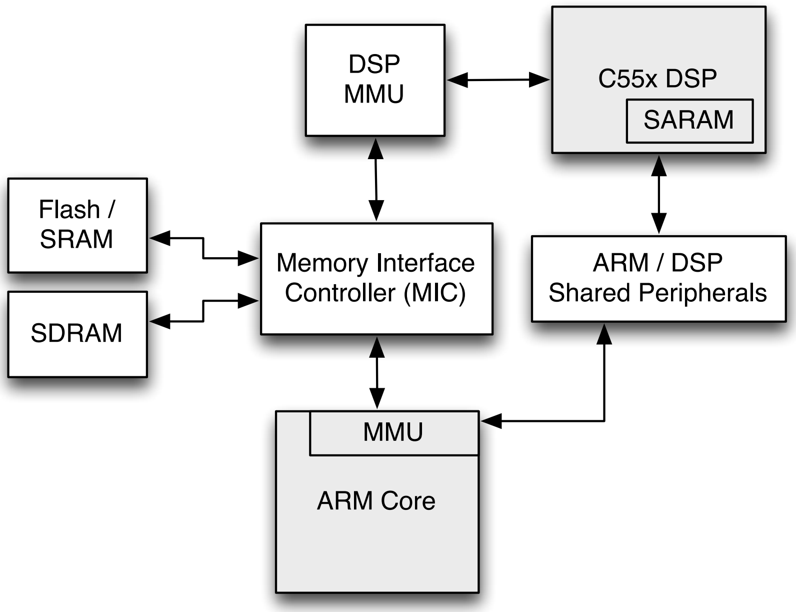
DSP MMU
C55x DSP
SARAM
Flash / SRAM
SDRAM
Memory Interface Controller (MIC)
ARM / DSP Shared Peripherals
MMU
ARM Core

Fig. 6. The OMAP1510 platform [Texas Instruments 2012]

in GSM networks. Similar to A5/2, the cipher uses four LFSRs which are clocked irregularly. We call these LFSRs R_1, R_2, R_3 and R_4, see Figure 7 for a schematic of the structure.

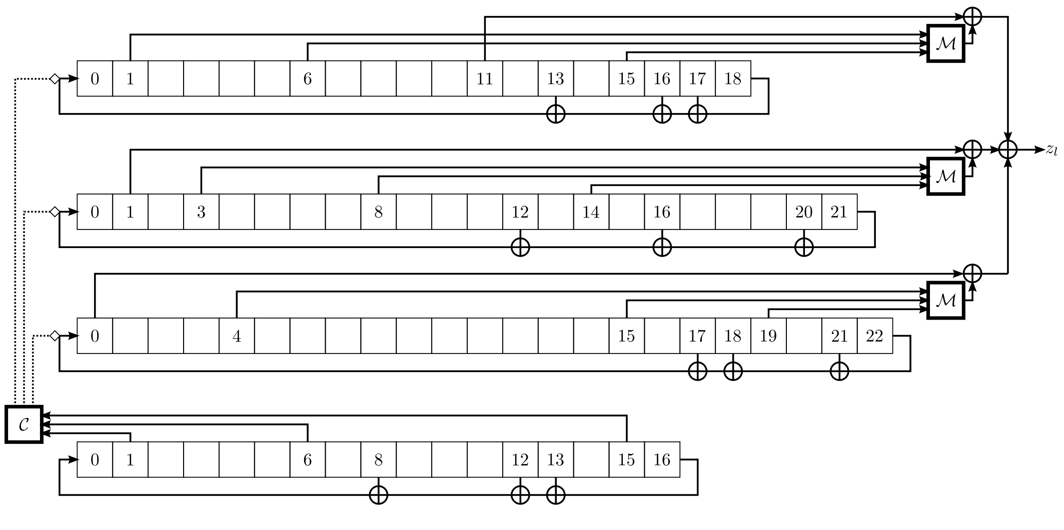

Fig. 7. The A5-GMR-1 cipher

Comparing A5/2 and A5-GMR-1, we see that for most registers the feedback polynomials and also the selection of input taps for the non-linear majority-function $\mathcal{M}$

Table I. Configuration of LFSRs in A5-GMR-1 and A5/2

		A5-GMR-1			A5/2		
	Size	Feedback polynomial	Taps	Final	Feedback polynomial	Taps	Final
R_1	19	$x^{19}+x^{18}+x^{17}+x^{14}+1$	1,6,15	11	$x^{19}+x^5+x^2+x+1$	12,14,15	18
R_2	22	$x^{22}+x^{21}+x^{17}+x^{13}+1$	3,8,14	1	$x^{22}+x+1$	9,13,16	21
R_3	23	$x^{23}+x^{22}+x^{19}+x^{18}+1$	4,15,19	0	$x^{23}+x^{15}+x^2+x+1$	13,16,18	22
R_4	17	$x^{17}+x^{14}+x^{13}+x^9+1$	1,6,15	-	$x^{17}+x^5+1$	3,7,10	-

with

$$\begin{aligned}\mathcal{M} : \{0,1\}^3 &\mapsto \{0,1\}\\ (x_2, x_1, x_0)_2 &\mapsto x_2x_1 \oplus x_2x_0 \oplus x_0x_1\end{aligned}$$

were changed. Also, the positions of the bits that are XORed with the respective outputs of the majority functions are different. All feedback-polynomials have five monomials, which is not the case for A5/2, as shown in Table I.

4.4. Mode of Operation

Next we focus on the mode of operation. Clocking a single LFSR means evaluating its respective feedback polynomial and using the resulting bit to overwrite the leftmost position of the LFSR, after shifting its current state by one bit to the right. When the cipher is clocked for the l-th time with irregular clocking active, the following happens:

(1) The irregular clocking component $\mathcal{C}$ evaluates all taps of R_4, the remaining registers are clocked accordingly, i.e.,
  (a) Iff $\mathcal{M}(R_{4,1}, R_{4,6}, R_{4,15}) = R_{4,15}$, register R_1 is clocked.
  (b) Iff $\mathcal{M}(R_{4,1}, R_{4,6}, R_{4,15}) = R_{4,6}$, register R_2 is clocked.
  (c) Iff $\mathcal{M}(R_{4,1}, R_{4,6}, R_{4,15}) = R_{4,1}$, register R_3 is clocked.

(2) The taps of R_1, R_2 and R_3 are evaluated and one bit of keystream is output accordingly, i.e.,

$$\begin{aligned}z_l = &\mathcal{M}(R_{1,1}, R_{1,6}, R_{1,15}) \oplus \mathcal{M}(R_{2,3}, R_{2,8}, R_{2,14}) \oplus\\ &\mathcal{M}(R_{3,4}, R_{4,15}, R_{3,19}) \oplus R_{1,11} \oplus R_{2,1} \oplus R_{3,0}\end{aligned}$$

is generated.

(3) R_4 is clocked.

The cipher is operated in two modes, *initialization* and *generation* mode. Running the cipher in former mode includes setting the initial state of the cipher, which is done in the following way:

(1) All four registers are set to zero.

(2) A 64-bit initialization vector $\alpha = (\alpha_0, ..., \alpha_{63})$ is computed by XORing the bits of the 19-bit frame number N and 64-bit session key K, i.e.,

$$\begin{aligned}\alpha = \mathcal{F}(K, N) = (&K_0, K_1, K_2, K_3 \oplus N_6, K_4 \oplus N_7,\\ &K_5 \oplus N_8, K_6 \oplus N_9, K_7 \oplus N_{10},\\ &K_8 \oplus N_{11}, K_9 \oplus N_{12}, K_{10} \oplus N_{13},\\ &K_{11} \oplus N_{14}, K_{12} \oplus N_{15},\\ &K_{13} \oplus N_{16}, K_{14} \oplus N_{17}, K_{15} \oplus N_{18},\\ &K_{16}, K_{17}, ..., K_{21}, K_{22} \oplus N_4,\\ &K_{23} \oplus N_5, K_{24}, ..., K_{59}, K_{60} \oplus N_0,\\ &K_{61} \oplus N_1, K_{62} \oplus N_2, K_{63} \oplus N_3)\end{aligned}$$

(3) The bits of α are re-ordered to α' with

$$\alpha' = (\alpha_{15}, \alpha_{14}, ..., \alpha_0, \alpha_{31}, \alpha_{30}, ..., \alpha_{16}, \alpha_{47}, ..., \alpha_{32}, \alpha_{63}, ..., \alpha_{48})_2$$

and clocked into all four registers in this order. To clock one bit of α' into R_1, its feedback polynomial is evaluated and the resulting bit then clocked into R_1, *after* XORing it with the α' bit. The same bit of α' is also clocked into R_2, R_3 and R_4. Then, the second bit of α' is clocked into all four registers in this manner and so on. While doing this, irregular clocking is deactivated, i.e., all registers are clocked for each bit of α'.

(4) The least-significant bits of all four registers are set to 1, i.e., $R_{1,0} = R_{2,0} = R_{3,0} = R_{4,0} = 1$.

We denote the whole initialization process by

$$(\underbrace{\beta_0, ..., \beta_{18}}_{R_1}, \underbrace{\beta_{19}, ..., \beta_{40}}_{R_2}, \underbrace{\beta_{41}, ..., \beta_{63}}_{R_3}, \underbrace{\beta_{64}, ..., \beta_{80}}_{R_4}) = \mathcal{G}(K, N),$$

where β is a 81-bit string, comprised of the consecutive bits of the four initialized registers. After all registers are initialized, irregular clocking is activated and the cipher is clocked 250 times. The resulting output bits are discarded.

Now the cipher is switched into generation mode and clocked for $2 \cdot m$ times, generating one bit of keystream at a time. Here, m is the length of an encrypted frame. Depending on the direction[4] bit, either the first half or the second half of the $2 \cdot m$ keystream bits is used for encryption/decryption. We denote the l-th keystream bit by $z_l^{(N)}$, where $0 \le l < 2 \cdot m$ is the number of irregular clockings (after warm-up) and N the frame number that was used for initialization. Since our cryptanalysis will focus on the downlink, we denote the continuous keystream for frames $N, N+1, ...$ (as decrypted by the phone) by z, where

$$z = \left(z_0^{(N)}, z_1^{(N)}, ..., z_{m-1}^{(N)}, z_0^{(N+1)}, ..., z_{m-1}^{(N+1)}, z_0^{(N+2)}, ...\right)_2$$

is the concatenation of the first halves of $z^{(N)}, z^{(N+1)}, ...$ respectively. The choice of m depends on the type of channel for which data is encrypted or decrypted. For the TCH3 channel, each frame has a length of $m = 208$ bits.

4.5. Cryptanalysis

The attack we present in the following is a variant of the ciphertext-only attack originally presented in [Driessen et al. 2012], which itself was inspired by previous attacks [Petrovic and Fuster-Sabater 2000; Barkan et al. 2003] on A5/2. Please note that we treat bit strings as column vectors and vice versa. We now briefly review the proposed attack that exploits several weaknesses which are either due to the design of the cipher or due to the use of the cipher in GMR-1:

(1) Given R_4, the clocking behavior of A5-GMR-1 is uniquely determined.
(2) Since the inputs to each majority-component are only from one register, one bit of keystream can always be expressed as an easy to linearize quadratic equation over $GF(2)$.
(3) In GMR-1, encryption is applied after encoding (which is entirely linear in $GF(2)$) and scrambling[5].
(4) For each two keystreams generated by the same session key but different frame numbers, the respective initial states are linearly related by the XOR-differences of the frame numbers.

Due to the first and second observation and given enough keystream bits for a particular frame N we can guess R_4, clock the entire cipher for several times and generate

[4]The first m bits are used on the handset's side for decryption, on the provider network side for encryption
[5]While encoding adds redundancy, scrambling is used to "[...] randomize the number of 0s and 1s in the output bit stream." [ETSI 2002].

a linearized system of equations over GF(2), i.e.,

$$\mathbf{A}x = z^{(N)},$$

describing keystream bits as linear combinations of terms which either are individual bits or products of two bits from the initial state of R_1, R_2 and R_3. If we guess R_4 correctly and $\mathbf{A}$ has full rank, solving the equation system gives the correct initial state which can easily be used to obtain the session key. Please note that, even if the session key is fixed, for different frame numbers not only the keystream but also the initial state and the matrix describing its relation to the keystream will be different. The required number of linearly independent equations, and hence the minimum[6] number of known keystream bits, is denoted as v with

$$v = \binom{18-k_1}{2} + \binom{21-k_2}{2} + \binom{22-k_3}{2} + (18-k_1) + (21-k_2) + (22-k_3).$$

Here, the numbers $18, 21$ and 22 are due to the sizes of registers R_1, R_2 and R_3 respectively (one bit is subtracted due to the fixed 1 per LFSR, cf. Subsection 4.4). By k_1, k_2 and k_3 we denote the number of bits we may additionally guess for each of these LFSRs. Fixing variables helps to decrease the size of the equation systems and the number of required keystream bits, but also increases the average amount of bits to guess for the whole attack to $2^{15+k_1+k_2+k_3}$.

We now use the principle we have outlined above (and the fact that encryption is applied to encoded data) for a ciphertext-only attack which explicitly targets the TCH3 channel in GMR-1. Encoding, scrambling and encrypting a 160-bit speech-frame $d^{(N)}$ with frame number N can be expressed as

$$c^{(N)} = \mathbf{G}^t d^{(N)} \oplus s \oplus z^{(N)},$$

where $\mathbf{G}^t$ is the transpose[7] of the 160×208 generator matrix $\mathbf{G}$ of the code, s is a 208-bit pseudo-random scrambling sequence, $z^{(N)}$ the keystream generated for this frame and $c^{(N)}$ the resulting 208-bit codeword. $\mathbf{G}$ and s can be derived from the specification of TCH3 (cf. Subsection 5.2), additionally a parity-check matrix $\mathbf{H}$ can be derived from $\mathbf{G}$ with $\mathbf{H}c = 0$ iff $c = \mathbf{G}^t d$. Due to this property, if we invert scrambling for a codeword $c^{(N)}$, we get

$$\mathbf{H}\left(c^{(N)} \oplus s\right) = \mathbf{H}\left(\mathbf{G}^t d^{(N)} \oplus z^{(N)}\right) = \mathbf{H}z^{(N)}.$$

Given a syndrome (i.e., a bit vector indicating whether decoding was completed without errors, potentially enabling error correction) vector $r^{(N)} = \mathbf{H}\left(c^{(N)} \oplus s\right)$, we can, again, set up an equation system in variables $x_0, x_1, ..., x_{v-1}$ of the initial state by guessing R_4, clocking the cipher 250 times (to account for the warm-up phase) and another 208 times, i.e.,

$$\mathbf{H}(\mathbf{A}x) = \mathbf{S}x = r^{(N)}.$$

Here, $\mathbf{A}$ is the $208 \times v$ matrix that describes the linear relation between x and the bits $z_0^{(N)}, z_1^{(N)} ..., z_{207}^{(N)}$ generated by the cipher. Please note that $\mathbf{H}$ is a 48×208 matrix and subsequently $\mathbf{S}$ is a $48 \times v$ matrix which implies that for $v > 48$ this system is not uniquely solvable. In order to obtain an equation system where $\mathbf{S}$ has full rank, we

[6]We need at least as many keystream bits as we have variables and thus equations. However, since not all equations we obtain by clocking the cipher based on R_4 are necessarily linearly independent, we may need even more keystream bits.

[7]We deviate from the traditional notation of encoding via $c = d\mathbf{G}$ and decoding via $s = \mathbf{H}c^t$ for the sake of clarity, since we consider column vectors instead of row vectors.

need to generate and collect equations from several encrypted frames for consecutive frame numbers $N, N+1, \ldots$. For a fixed session key, the initial states for different frame numbers are linearly related by the XOR-differences of the frame numbers. Taking these differences into account when generating equations allows to build a uniquely solvable equation systems and solving this equation system gives a potential initial state which could have generated $z^{(N)}$.

Now we describe the actual steps of our attack for which we assume that we are in possession of n 48-bit syndromes

$$r^{(N_0)} = \mathbf{H}\left(c^{(N_0)} \oplus s\right), \ldots, r^{(N_{n-1})} = \mathbf{H}\left(c^{(N_{n-1})} \oplus s\right)$$

which correspond to TCH3 downlink data encrypted under the same session key. Our attack is parameterized by n, k_1, k_2, k_3 and N_0 and recovers the initial state $\beta = \mathcal{G}(K, N_0)$. Before we proceed, we need to introduce the helper-function $\mathcal{V}(\cdot)$ which can be applied to extract certain bits of the 81-bit state of A5-GMR-1. Depending on the configuration of the attack, $\mathcal{V}(\cdot)$ will extract a bitstring which corresponds to these positions of the overall state whose bits we have guessed (i.e., R_4 and parts of the other registers).

(1) Systematically guess the bitstring γ which has $20 + k_1 + k_2 + k_3$ bits (also incorporating the fixed bit per LFSR). For each syndrome $0 \leq i < n$ do the following:
   (a) Compute the 81-bit difference $\delta = \mathcal{G}(0, N_0) \oplus \mathcal{G}(0, N_i)$ in the initialization state for frame number N_0 and N_i.
   (b) Modify γ by XORing it with the corresponding positions of δ, i.e., $\gamma' = \gamma \oplus \mathcal{V}(\delta)$.
   (c) Based on γ' and δ generate a linearized $458 \times v$ matrix $\mathbf{B}$ (and vector y for the one constant per equation) which describes the linear relation between the initial state for N_0 and the 458 keystream bits generated for $r^{(N_i)}$.
   (d) Take the warm-up phase into account by discarding the first 250 rows of $\mathbf{B}$ to obtain a $208 \times v$ matrix $\mathbf{B}'$ and also discarding the first 250 elements of y to obtain y'.
   (e) Compute the $48 \times v$ matrix $\mathbf{S}'$ and vector r' such that

$$\mathbf{S}' = \mathbf{H}\mathbf{B}' \qquad \text{and}$$
$$r' = \mathbf{H}y' \oplus r^{(N_i)} = \mathbf{H}\left(y' \oplus c^{(N_i)}\right)$$

   and add those rows of $\mathbf{S}'$ (and the corresponding bits from r') to the equation system $\mathbf{S}x = r$, which are linearly independent from all previously existing rows of $\mathbf{S}$.
   (f) Abort if $\mathbf{S}$ has full rank.

(2) Solve the equation system by computing $x = \mathbf{S}^{-1}r$ and combine the guessed bits and x appropriately to obtain the 81-bit initialization state candidate β.

(3) Initialize A5-GMR-1 with β and clock it to obtain 208 bits of keystream $z'^{(N_0)}$ for frame number N_0 and test whether

$$\mathbf{H}\left(c^{(N_0)} \oplus s \oplus z'^{(N_0)}\right) = 0.$$

If this equation holds, applying the obtained keystream produced a valid codeword. This implies we have produced the correct keystream and therefore (most likely) the correct initial state.

Once we have $\beta = \mathcal{G}(K, N_0)$, we can set up another equation system

$$\mathbf{L}\alpha = \beta \oplus \epsilon \qquad \text{with} \qquad \alpha = \mathbf{L}^{-1}(\beta \oplus \epsilon) = \mathcal{F}(K, N_0)$$

where $\mathbf{L}$ describes the process of clocking α into all four LFSRs (and setting the lowest bit per LFSR to 1 which is expressed by ϵ). Solving the equation system resolves the initialization vector α from which we can easily derive the session key $K = \mathcal{F}(\alpha, N_0)$.

5. A REAL-WORLD ATTACK ON THURAYA

In this section we describe the details of our real-world attack on the TCH3 channel in the Thuraya network.

5.1. Recording TCH3 Data

Executing the attack described in Section 4.5 requires acquiring and setting up appropriate hardware to generate and receive real-world data in the Thuraya network. Here, we describe our hard- and software setup that allows us to record speech data frames.

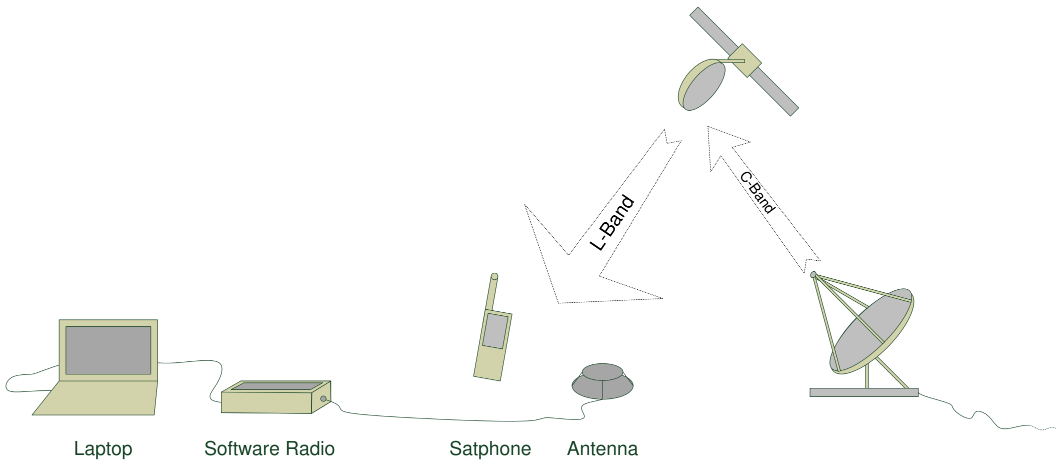

Fig. 8. A schematic of the attack setup

Figure 8 provides a schematic overview of our attack setup: We use a satphone to establish a call in the Thuraya network and place an antenna nearby, thus receiving all downlink transmissions. Attached to the antenna is a Software Defined Radio (SDR) system. With the help of the SDR hardware and some software running on the laptop, we can demodulate and decode received transmissions. It is important to note here that we only receive the downlink and not the uplink. We focus on this part of the communication for two reasons:

(1) Demodulation of downlink transmissions is (mostly) readily available as part of OsmocomGMR, while this is not true for the uplink.
(2) The downlink can be received (at least) in the entire area which is assigned to one spotbeam (see below).

Furthermore, if we can decrypt the downlink, we can also decrypt the uplink – both share the same session key for encryption.

The software we use here is based on the OsmocomGMR project [Munaut 2012], which is a subproject of Osmocom and is maintained by Sylvain Munaut. The aim of the Osmocom project family is to establish open-source implementations of a wide range of communication standards, e.g., GSM, TETRA and even GMR-1. Although the implementation of OsmocomGMR is still in its infancy it is evolved enough for our purposes; we were able to use it with only a few tweaks—although not in a completely automated fashion.

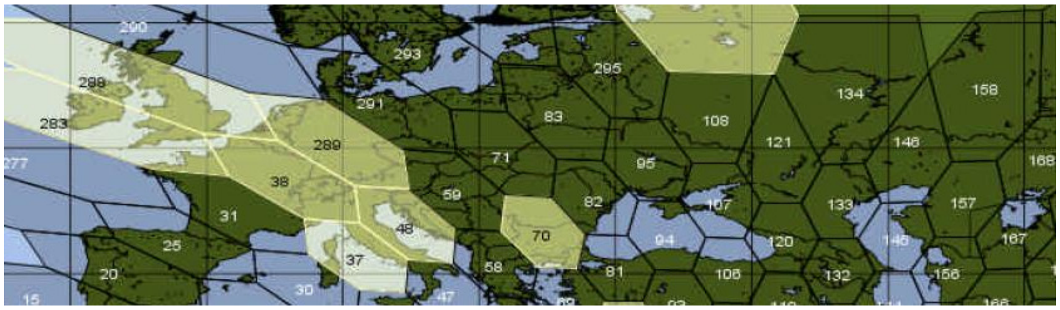

Fig. 9. Thuraya spotbeams (with numeric IDs) over central Europe, [Munaut 2012]

In addition to software, the OsmocomGMR project also provides[8] information regarding the configuration of the Thuraya network. As shown in Figure 9, there are two spotbeams assigned to Germany, they have the IDs 289 and 291. Since our experiments were performed in Bochum, we picked the spotbeam with the former ID, which is assigned the ARFCN 1007. Given the fact that the downlink frequency band is divided into 1087 physical channels (starting at 1.525 GHz) with a spacing of 31.25 KHz, ARFCN 1007 translates into a radio frequency of

$$f_D = 1.525 \text{ GHz} + \frac{31.25}{2} \text{ KHz} + 1007 \cdot 31.25 \text{ KHz} = 1.556484375 \text{ GHz}$$

for the downlink of the CCH channel.

To obtain real-world data for our attack, we used our Thuraya satphone and established some calls to a landline. By simultaneously tuning the SDR to f_D, we were able to intercept TCH3 assignments which were sent to our phone via the CCH (cf. Section 2.2). After some experimentation we found that TCH3 is typically assigned to one of these three ARFCNs: 1008, 1009 and 1011, a fact that is now also documented on the website of OsmocomGMR. Upon observing the ARFCN assignment, we could tune to the newly assigned frequency and capture most of the encrypted downlink speech data (missing only a fraction at the beginning of the call). All subsequent data (including frame numbers) was stored on a harddisk and could directly be used in our cryptanalysis.

At this point, it should be noted that we only target the downlink, which can be received easily and does not require immediate proximity to an eavesdropping target. However, downlink data only gives half of a communication, which is why we have also investigated the possibility of receiving uplink transmissions. The Thuraya SO-2510 has a small, helical antenna which not only radiates where the satphone is pointed at, but also to the sides (although with lower power). We have determined this level of power by establishing a call on the roof of the university while holding the phone strictly vertical and measuring the side radiation from a fixed distance. Using a signal analyzer and a horizontally polarized antenna with 5.85 dB gain, the uplink signal was detected at 1.65 GHz and determined to have a power of -33 dBm at a distance of 15 m. This implies that, assuming a free path loss of 110 dB and further propagation losses

[8]See `http://gmr.osmocom.org/trac/wiki/Thuraya_Beams`.

of 20–30 dB, it is entirely possible to directly receive the uplink signal at distances of 5 Km and more[9], given a direct line of sight.

We speculate that a more indirect approach might exploit the fact that Thuraya-2 and Thuraya-3 operate according to the “bent pipe” principle. In this setup, the satellite just acts as a redirector of incoming data, i.e., uplink data sent by a satphone is simply redirected to the ground segment (although shifted to a different frequency band). This implies that uplink data of mobile devices in the user segment can be intercepted by placing a satellite dish “closely” to the central gateway. However, implementing this method of interception is hindered by the lack of public specifications for the C-Band, which is used to transmit data between ground segment and satellite. Another difficulty stems from the fact that several concurrent communications are sent over this link to the gateway in parallel, therefore L-Band downlink and C-Band uplink data need to be matched. To us, it seems nonetheless reasonable to assume that this can be done when considering that Thuraya handles “only” $13\,750$ calls simultaneously, up- and downlink share frame numbers (and we can get frame numbers from the downlink) and timing of uplink data in the C-Band can probably be predicted quite accurately.

5.2. Encoding and Parity-Check Matrix

As stated in the previous sections, a key step to move from a known-plaintext to a ciphertext-only scenario is collapsing all linear encoding steps into a single matrix $\mathbf{G}$ and deriving the respective parity-check matrix $\mathbf{H}$. Obtaining $\mathbf{G}$ is straightforward: All relevant encoding steps for the TCH3 channel can be found in the respective document of the specification [ETSI 2001d]. These steps include:

(1) Block encoding
(2) Convolutional encoding
(3) Interleaving
(4) (Scrambling)
(5) Multiplexing

Each step—except for scrambling—can modeled as multiplication of an information vector with an appropriately constructed matrix $\mathbf{M}_i$ with $0 \leq i \leq 3$. Given these matrices, their product is the 160×208 encoding matrix

$$\mathbf{G} = \prod_{i=0}^{3} \mathbf{M}_i.$$

The corresponding parity-check matrix $\mathbf{H}$ with

$$\mathbf{H}(\mathbf{G}^t d) = 0 \quad \text{for all} \quad d \in \{0,1\}^{160}$$

can be obtained with these steps:

(1) Use Gaussian elimination to find a permutation matrix $\mathbf{P}$ with

$$\mathbf{LGP} = \mathbf{G}' = (\mathbf{I}_{160}|\mathbf{T})$$

for some $\mathbf{L}$, where the left hand side of $\mathbf{G}'$ is the 160×160 identity matrix. Here, $\mathbf{G}'$ is the *systematic form* of the encoding matrix.

(2) From the systematic form of $\mathbf{G}$, $\mathbf{H}'$ can be obtained easily, i.e.,

$$\mathbf{H}' = \left(\mathbf{T}^t|\mathbf{I}_{48}\right).$$

The result is a 48×208 matrix, which is appropriate for code words encoded with $\mathbf{G}'$.

[9]Using a high gain antenna and/or a more sensitive receiver should significantly boost reception levels.

(3) From $\mathbf{H}'$ the parity-check matrix $\mathbf{H}$ for the actual form of $\mathbf{G}$ can be obtained via another matrix multiplication, i.e.,

$$\mathbf{H} = \mathbf{H}'\mathbf{P}^{-1}.$$

The resulting matrix is given (rows encoded hexadecimally) in Figure 10.

```
2008020200802000028000a0202020a080800000800000000000
0410014010001000040501444501400441044440400000000000
080200808020080000a000280808082820200000200000000000
4415044000400405005000044454441400401000100000000000
8020080020080200022000888080808808080000080000000000
0004010050440100001050544000101454541040040000000000
4014040040500005000000104050400010501000000000000000
8028080080a0000a0000002080a0800020a02000000000000000

a0280802002020020080a0a02080008080800000008000000000
4400054010401004004050140410144004104400004000000000
280a0200800800802028280820002020200000000020000000000
4414040010400404001014005454501004541400001000000000
8822008020800208020088888008000808080000000800000000
0405054050000104000014544044505454541040000400000000
0000080000800008802088000880808808000008000000000000
000002000020000a0200a000802020a080000080000000000000

880a0a82800808088280a00088a8a82000080000000080000000
0000000000000400004000504050004010105040000040000000
a0220a80a080020a00a02800a0a8a80800800000000020000000
0411000100400105005014540454000454405400000010000000
28280a82202020028220880028a8a88000200000000008000000
4405014140441000001014140400141054541440000004000000
0411044100040000001044400004444444444004000000000000
200a0200202800028000000820282000082808000000000000000

a802088000a828008280200088282820808800000000000800000
0410004050001401004040145440544014001440000000400000
a8200a0000a80a0800a00800a088880820a00000000000200000
4004044110040501005004444414500454004400000000100000
a808028000a822020220800028a0a08008280000000000080000
0004050150441104001010540040141010145440000000040000
0411004100440004400000400444040040444000000000000000
0000008000080002808028002008082820000020000000000000

282a00828088000a800020008828a82000080000000000008000
4404054110000000100001014145004101440140000000004000
882a0800a0a0000a80000800a088a80800800000000000002000
0010004110440004000004501450100450545000000000001000
a02a02022028000a8000800028a0a88000200000000000000800
4010000100000004000040040444444004040400000000000400
5400054150000000000000545400540054005400000000000000
0411000100400000500401440144000145044401000000000000

0008000280802800020020 20a08000008000000000000000000080
4010050100001400000040005000505010501040000000000040
00020000a0200a00080080828200000200000000000000000020
4015040010400505004004145440400000501000000000000010
0020000220082200002080808808000008000000000000000008
0010000110041100001040404404000004000000000000000004
4014050040400000010050100040505050501040000000000000
0822008200880008800000800888080080888000000000000000
```

Fig. 10. The (encoded) $\mathbf{H}$ matrix of the TCH3 channel in Thuraya

5.3. Parameterization

As described in Section 4.5, our ciphertext-only attack on TCH3 is parameterized by the tuple (n, k_1, k_2, k_3). To actually execute the attack, we have experimentally established a working set of parameters.

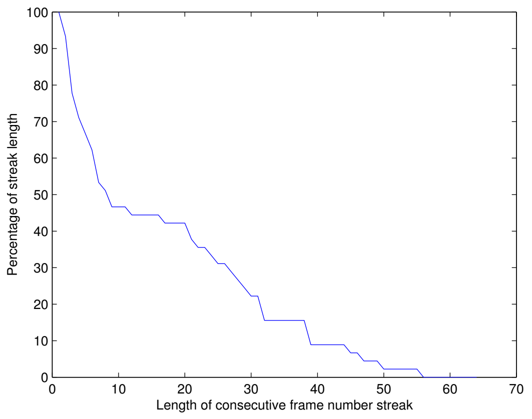

Fig. 11. Percentage of observed frame number streak lengths obtained from several short calls

First of all, we have determined how many streaks of TCH3 frames with consecutive frame numbers of a certain length we can expect to obtain with our eavesdropping setup. By *streak* we denote a set of received frames with numbers $N_0, N_1, N_2, ..., N_{n-1}$ with $N_i = N_{i-1} + 1$ for all $0 < i < n$, where n denotes the length of a streak. We have analyzed the TCH3 data of several 10 second calls and plotted the percentage of observed streak lengths in Figure 11. The longest streak we have observed consists of 56 TCH3 frames, which serves as an upper bound for the next step.

Table II. Guessed bits of the LFSRs R_1, R_2 and R_3

k_1	k_2	k_3	#Variables	#Frames (avg.)	#Frames (max.)
0	2	4	532	12.42	24
0	3	3	532	12.73	25
1	2	3	533	13.01	25

In order to determine how many bits of the LFSRs R_1, R_2 and R_3 we need to guess[10] (in addition to completely guessing R_4), we have performed experiments: We have systematically evaluated all combinations of k_1, k_2 and k_3 with the aim to minimize $k_1 + k_2 + k_3$. We found that we need to guess at least 6 bits of R_1 to R_3, in order to always achieve full rank from 56 TCH3 frames. Of the 28 possibilities, only three were found to always guarantee full rank of the obtained matrices, see Table II. Looking at the results, we picked $k_1 = 0, k_2 = 2, k_3 = 4$ because the average and maximum number of frames required to achieve full rank is lowest. Also, we can be certain that 33% of the frame number streaks (cf. Figure 11) are at least 24 frames long. This is helpful, because now we can pick multiple, different subsets of 24 TCH3 frames for our attack, which allows to validate any session key we may find.

In contrast to the original attack [Driessen et al. 2012], which targets the FACCH3, uses only one frame and has an average case complexity of 2^{32}, we now achieve a complexity of guessing and solving

$$2^{15+k_1+k_2+k_3} = 2^{15+0+2+4} = 2^{21}$$

[10]Please note that, for simplicity, we always guess the LSBs of each LFSR. It is certainly an interesting question which bits should be guessed for optimal performance: guessing different positions might lead to a higher percentage of linearly independent equations, which in turn could to reduce the number of guessed bits and thus improve the performance of the attack.

equation systems. This is due to the fact that we use multiple TCH3 frames to collect linearly independent equations. In this way, by gaining more equations we gain more information about the initial state of the cipher and thus have to guess less bits; it was this considerable improvement that made the attack practical, as shall be described in the following.

5.4. Implementation

With the parameters we have established in the previous section, we need to generate and solve on average 2^{21} equation systems with matrices of dimension 532×532 and consequentially test as many state candidates (by decoding via another matrix operation) to obtain one session key. To speed up the actual attack, we exploit the fact that the matrices which describe the linear relation between internal state of the cipher and keystream depend only on the bits we guess. These matrices are simply multiplied with $\mathbf{H}$, which is fixed and static (cf. Section 2.3 and Section 5.2). Thus, once we have fixed a set of parameters we are going to use in the attack, we need to generate and store the resulting S matrices only once. This effectively splits the execution of the attack into a *pre-computation* and *recovery* phase:

— In the pre-computation phase, a parameter set is chosen; all matrices, as result of the bits we guess, are generated and stored. This step has to be done only once.
— In the recovery phase, the generated matrices are read from disk and subsequently used to build and solve equation systems for actual TCH3 channel data. This step has to be repeated for every new GMR-1 TCH3 session.

To further optimize the execution time of the recovery phase, we apply two more tricks:

(1) In the pre-computation phase, we already test the matrices for linear dependencies and bring them in upper triangular form. Since we have to apply the resulting row operations also to the syndromes (i.e., the results of multiplying ciphertexts with $\mathbf{H}$) in the recovery phase, we have track and store them, too. While this requires some more storage space, not having to do linearity testing in the recovery phase and knowing that the lower tridiagonal part of each matrix is zero is a considerable improvement in terms of computation time and storage space.
(2) We use a very fast implementation of the Lempel-Ziv (LZ) compression algorithm [Ziv and Lempel 1977] in order to minimize the required storage space and the performance impact of reading and writing matrices from to a standard hard-disk. Although compression introduces some computational overhead in the pre-computation phase, decompression is fast enough to significantly speed-up our attack (when compared to an attack using uncompressed files).

Although our attack only uses TCH3 data, our implementation is also able to generate and handle mixtures of FACCH3, TCH3 and keystream frames for an attack, which makes it considerably more complex. However, more details on the implementation are not relevant here, which is why the results of our attack will be presented next.

5.5. Results

In this section we shortly subsume the results of executing our proposed attack and the hardware we have used. Figure 12 shows the components we have used to perform the attack:

(1) An accessory[11] antenna (typically used for installing satphones in cars) was used to receive Thuraya traffic.

[11]Earlier attempts at building such an antenna failed; assembling helical antennas is a very delicate process and requires experience and very high precision.

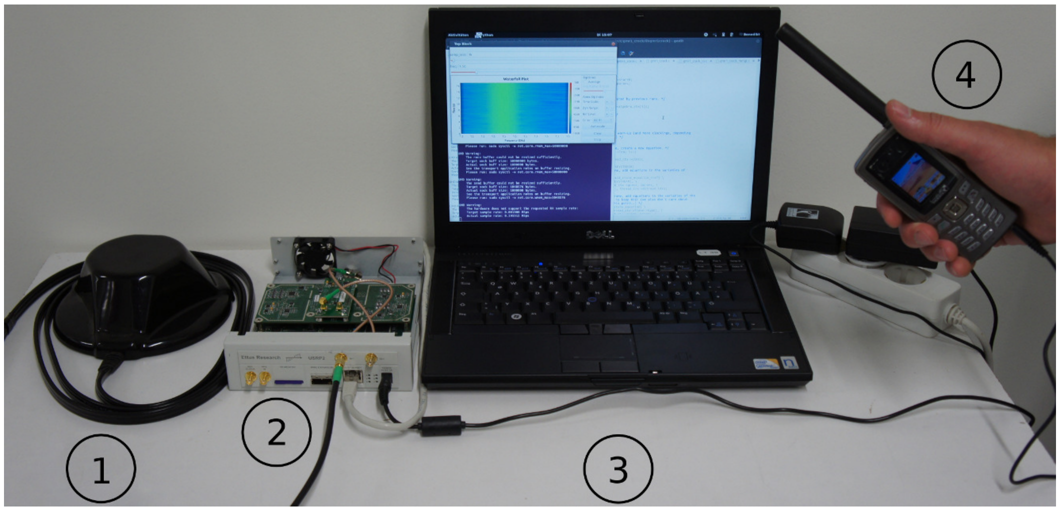

Fig. 12. The attack setup: (1) antenna, (2) software radio, (3) laptop, (4) satphone

(2) An Ettus USRP-2 device was used to digitize received transmissions (coming from the antenna) and send them to an attached laptop.
(3) A laptop is used to control the USRP-2, apply demodulation steps and execute the implemented attack.
(4) A Thuraya SO-2510 satphone was used as handheld device for communicating over the Thuraya network.

In the pre-computation phase, we have generated approx. 400 GB of system matrices in a negligible amount of time. We have executed a 30 second call between satphone and landline, of which more than 27 seconds of TCH3 data could be saved to disk. Given the eavesdropped data and pre-computed matrices, we were able to find the session key for multiple subsets of 24 frames in 32.1 minutes (on average).

It must be stressed here, that—since the speech codecs of Thuraya still have not been reverse-engineered—actually listening to a conversation is not possible for us. However, since the codecs can be reverse-engineered too (potentially even by applying similar heuristics as used by us) this is no real obstacle.

6. SECURITY ANALYSIS OF GMR-2

To obtain the code responsible for implementing the cipher according to the GMR-2 standard, we analyzed the latest publicly available firmware image of the Inmarsat IsatPhone Pro, which was released in June 2010. Only Inmarsat handsets support the GMR-2 standard at this point and we are confident that our analysis results apply to all of these satphones.

6.1. Hardware Architecture

The Inmarsat IsatPhone Pro runs on an Analog Devices LeMans AD6900 platform. The core of the platform is an ARM-926EJ-S CPU, which is supplemented by a Blackfin DSP (see Figure 13 for a schematic overview). This architecture can be deduced from plain text strings within the firmware image. We identified an operating system function that returns information on the underlying hardware of the system and this function returns the platform name as a static string.

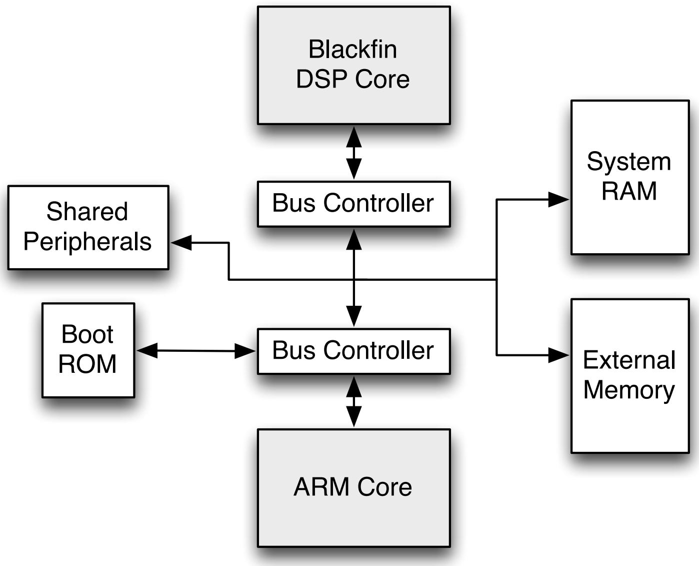

Fig. 13. The LeMans AD6900 platform [Jose Fridman, Analog Devices]

Both processors connect to the same bus interface, which is attached to the system RAM, any external memory that might be present as well as the shared peripherals (e.g., SIM card, keypad, SD/MMC slots, etc.). The system is initialized by the boot ROM code of the ARM CPU. The ARM CPU then has the task to initialize the DSP for further operations.

6.2. Finding the Cipher

We were able to reverse-engineer the A5-GMR-2 cipher from a firmware update for the Inmarsat IsatPhone Pro. We used IDA to analyze the ARM portion of the firmware and a custom disassembler to disassemble the code running on the Blackfin DSP. Starting from a function which XORs keystream and plaintext data, we traced the usage of the keystream buffer back to its origin and from there to the cipher which fills it. This process was mostly done manually, although it only became feasible after applying a trick to narrow down the search space. For more details, please read the original publication [Driessen et al. 2012].

6.3. Structure of the Cipher

After having obtained the cipher's assembler code, we had to find a more abstract description in order to enhance intuitive understanding of its way of functioning. We arbitrarily chose to split the cipher into several distinct components which emerged after examining its functionality. Note that, for the sake of symmetry, we denote the cipher as A5-GMR-2, although it shows no resemblance to any of the A5-type ciphers and is called GMR-2-A5 in the respective specification [ETSI 2001c].

The cipher uses a 64-bit encryption-key and operates on bytes. When the cipher is clocked, it generates one byte of keystream, which we denote by Z_l, where l represents the number of clockings. The cipher exhibits an eight byte state register $S = (S_0, S_1, ..., S_7)_{2^8}$ and three major components we call $\mathcal{F}, \mathcal{G}$ and $\mathcal{H}$. Additionally, there is a 1-bit register T that outputs the so-called "toggle-bit" (which alternates between 1 and 0 for each clock of the cipher) and a 3-bit register C that implements a counter (counting from zero to seven) which is incremented for each clock of the cipher. Figure 14 provides a schematic overview of the cipher structure. In the following, we detail the inner workings of each of the three major components.

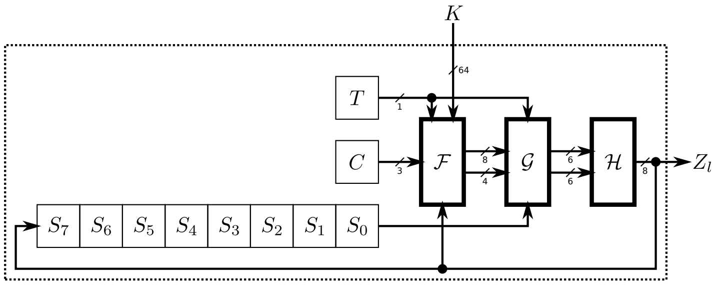

Fig. 14. The A5-GMR-2 cipher

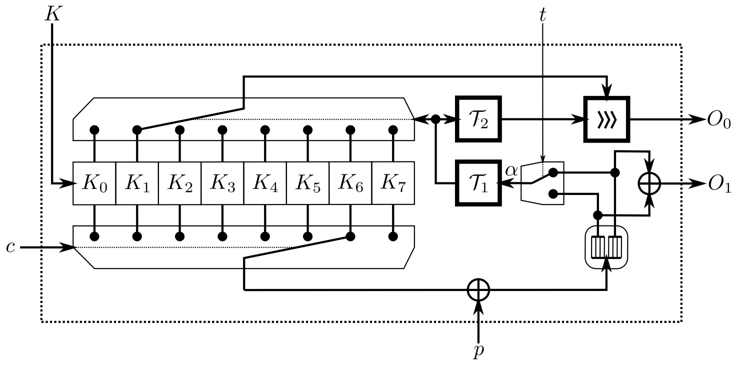

Fig. 15. $\mathcal{F}$-component of A5-GMR-2

We begin with the $\mathcal{F}$-component, which is certainly the most interesting part of this cipher—Figure 15 shows its internal structure. On the left we see another 64-bit register split into eight bytes $(K_0, K_1, ..., K_7)_{2^8}$. The register is read from two sides, on the lower side one byte is extracted according to the value of c, i.e., the output of the lower multiplexer is K_c. The upper multiplexer outputs another byte, but this one is determined by a 4-bit value we will call α. On the right side, two smaller sub-components

$$\mathcal{T}_1 : \{0,1\}^4 \mapsto \{0,1\}^3$$
$$\mathcal{T}_2 : \{0,1\}^3 \mapsto \{0,1\}^3$$

are implemented via table-lookups (see Table III).

The input of $\mathcal{T}_1$ is determined by p, K_c and the toggle-bit t. Note that we use $p = Z_{l-1}$ as a shorthand for the byte of keystream that was generated in the preceding clock. We model the behavior of the small vertical multiplexer by $\mathcal{N}(\cdot)$, which we define as

$$\mathcal{N} : \{0,1\} \times \{0,1\}^8 \mapsto \{0,1\}^4$$
$$(t, x_7, x_6, ..., x_0) \mapsto \begin{cases} (x_3, x_2, x_1, x_0)_2 & \text{if } t = 0, \\ (x_7, x_6, x_5, x_4)_2 & \text{if } t = 1. \end{cases}$$

Table III. $\mathcal{T}_1$ and $\mathcal{T}_2$ as lookup-table

x	$\mathcal{T}_1(x)$	$\mathcal{T}_2(x)$	$\mathcal{T}_2(\mathcal{T}_1(x))$	
$(0,0,0,0)_2$	2	4	6	
$(0,0,0,1)_2$	5	5	3	
$(0,0,1,0)_2$	0	6	4	*
$(0,0,1,1)_2$	6	7	2	
$(0,1,0,0)_2$	3	4	7	
$(0,1,0,1)_2$	7	3	1	
$(0,1,1,0)_2$	4	2	4	*
$(0,1,1,1)_2$	1	1	5	
$(1,0,0,0)_2$	3	-	7	
$(1,0,0,1)_2$	0	-	4	*
$(1,0,1,0)_2$	6	-	2	
$(1,0,1,1)_2$	1	-	5	
$(1,1,0,0)_2$	5	-	3	
$(1,1,0,1)_2$	7	-	1	
$(1,1,1,0)_2$	4	-	4	*
$(1,1,1,1)_2$	2	-	6	

With the help of $\mathcal{N}$, which returns either the higher or lower nibble of its second input, the following holds for the output of the mentioned multiplexer

$$\alpha = \mathcal{N}(t, K_c \oplus p) = \mathcal{N}(c \bmod 2, K_c \oplus p).$$

The output of the upper multiplexer is rotated to the right by as many positions as indicated by the output of $\mathcal{T}_2$, therefore the 8-bit output O_0 and the 4-bit value O_1 are of the following form,

$$\begin{aligned} O_0 &= (K_{\mathcal{T}_1(\alpha)} \ggg \mathcal{T}_2(\mathcal{T}_1(\alpha)))_{2^8} \\ O_1 &= (K_{c,7} \oplus p_7 \oplus K_{c,3} \oplus p_3, \\ &\quad K_{c,6} \oplus p_6 \oplus K_{c,2} \oplus p_2, \\ &\quad K_{c,5} \oplus p_5 \oplus K_{c,1} \oplus p_1, \\ &\quad K_{c,4} \oplus p_4 \oplus K_{c,0} \oplus p_0)_2. \end{aligned}$$

The $\mathcal{G}$-component gets the outputs of the $\mathcal{F}$-component as inputs, i.e., $I_0 = O_0, I_1 = O_1$. Additionally, the one byte S_7 of the state is used as input. As can be seen in Fig-

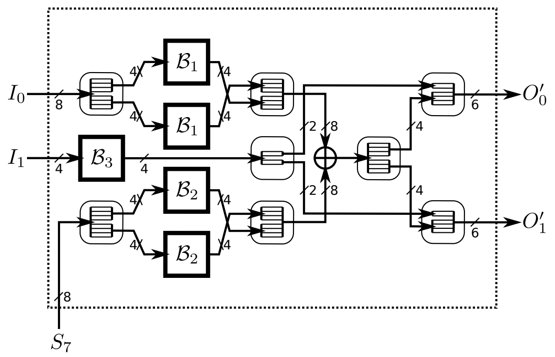

Fig. 16. $\mathcal{G}$-component of A5-GMR-2

ure 16, three sub-components, denoted as $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, are employed—again, they are implemented in the form of lookup-tables. Each of these components works on 4-bit inputs and equally returns 4-bit. After analyzing the tables, we found that all three simply implement linear boolean arithmetic, i.e.,

$$\begin{aligned}\mathcal{B}_1 : \{0,1\}^4 &\mapsto \{0,1\}^4\\ (x_3,x_2,x_1,x_0)_2 &\mapsto (x_3 \oplus x_0, x_3 \oplus x_2 \oplus x_0, x_3, x_1)_2,\\ \mathcal{B}_2 : \{0,1\}^4 &\mapsto \{0,1\}^4\\ (x_3,x_2,x_1,x_0) &\mapsto (x_1, x_3, x_0, x_2)_2,\\ \mathcal{B}_3 : \{0,1\}^4 &\mapsto \{0,1\}^4\\ (x_3,x_2,x_1,x_0) &\mapsto (x_2, x_0, x_3 \oplus x_1 \oplus x_0, x_3 \oplus x_0)_2.\end{aligned}$$

Since these sub-components and the XORs are linear and all other operations on single bits just amount to permutations, the $\mathcal{G}$-component is entirely linear. Therefore, we can write the 6-bit outputs O_0', O_1' as linear functions of the inputs I_0, I_1 and S_7, i.e.,

$$\begin{aligned}O_0' = (&I_{0,7} \oplus I_{0,4} \oplus S_{7,5}, I_{0,7} \oplus I_{0,6} \oplus I_{0,4} \oplus S_{7,7}, I_{0,7} \oplus S_{7,4}, I_{0,5} \oplus S_{7,6},\\ &I_{1,3} \oplus I_{1,1} \oplus I_{1,0}, I_{1,3} \oplus I_{1,0})_2,\\ O_1' = (&I_{0,3} \oplus I_{0,0} \oplus S_{7,1}, I_{0,3} \oplus I_{0,2} \oplus I_{0,0} \oplus S_{7,3}, I_{0,3} \oplus S_{7,0},\\ &I_{0,1} \oplus S_{7,2}, I_{1,2}, I_{1,0})_2.\end{aligned}$$

Finally, the $\mathcal{H}$-component gets $I_0' = O_0'$ and $I_1' = O_1'$ as input and constitutes the non-linear "filter" of the cipher (see Figure 17). Here, two new sub-components

$$\begin{aligned}\mathcal{S}_2 : \{0,1\}^6 &\mapsto \{0,1\}^4\\ \mathcal{S}_6 : \{0,1\}^6 &\mapsto \{0,1\}^4\end{aligned}$$

are used and implemented via lookup-tables. Interestingly, these tables were taken from the DES, i.e., $\mathcal{S}_2$ is the second S-box and $\mathcal{S}_6$ represents the sixth S-box of DES. However, in this cipher, the S-boxes have been reordered to account for the different addressing, i.e., the four most-significant bits of the inputs to $\mathcal{S}_2$ and $\mathcal{S}_6$ select the S-box-column, the two least-significant bits select the row. Note that this is crucial for the security of the cipher. The inputs to the S-boxes are swapped with the help of two

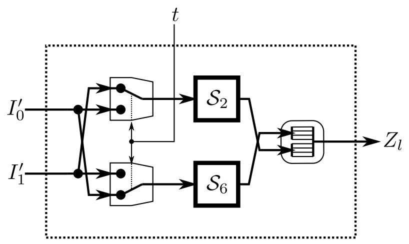

Fig. 17. $\mathcal{H}$-component of A5-GMR-2

multiplexers, depending on the value of t. Given the inputs I_0', I_1' and t we can express the l-th byte of keystream as

$$Z_l = \begin{cases}(\mathcal{S}_2(I_1'), \mathcal{S}_6(I_0'))_{2^4} & \text{if } t = 0,\\ (\mathcal{S}_2(I_0'), \mathcal{S}_6(I_1'))_{2^4} & \text{if } t = 1.\end{cases}$$

6.4. Mode of Operation

Next we describe the mode of operation. When the cipher is clocked for the l-th time, the following happens:

(1) Based on the current state of the S-, C- and T-register, the cipher generates one byte Z_l of keystream.
(2) The T-register is toggled, i.e., if it was 1 previously, it is set to 0 and vice versa.
(3) The C-register is incremented by one, when 8 is reached the register is reset to 0.
(4) The S-register is shifted by a byte to the right, i.e., $S_7 := S_6, S_6 := S_5$ etc. The previous value of S_7 is fed into the $\mathcal{G}$-component, the subsequent output Z_l of $\mathcal{H}$ is written back to S_0, i.e., $S_0 := Z_l$. This value is also passed to the $\mathcal{F}$-component as input for the next iteration.

The cipher is operated in two modes, *initialization* and *generation*. In the initialization phase, the following steps are performed:

(1) The T- and C-register are set to 0.
(2) The 64-bit encryption-key is written into the K-register in the $\mathcal{F}$-component.
(3) The state-register S is initialized with the 22-bit frame number N, this procedure is dependent on the "direction bit" but not detailed here as it is irrelevant for the remainder of this paper.

After C, T and S have been initialized, the cipher is clocked eight times, but the resulting keystream is discarded.

After initialization is done, the cipher is clocked to generate and output actual keystream bytes. By $Z_l^{(N)}$ we denote the l-th ($0 \leq l \leq 14$) byte of keystream generated after initialization (and warm-up) with frame number N. In GMR-2, the frame number is always incremented after 15 bytes of keystream, which forces a re-initialization of the cipher. Therefore, the keystream that is actually used is the concatenation of these 15-byte blocks. We denote continuous keystream for frames $N, N+1, ...$ by Z, with

$$Z = \left(Z_0^{(N)}, ..., Z_{14}^{(N)}, Z_0^{(N+1)}, ..., Z_{14}^{(N+1)}, Z_0^{(N+2)}, ...\right)_{2^8}.$$

6.5. Cryptanalysis

In this section, we present a known-plaintext attack that is based on several observations that can be made when carefully examining the $\mathcal{F}$-component (and the starred rows in Table III):

(1) If $\alpha \in \{(0,0,1,0)_2, (1,0,0,1)_2\}$ then $\mathcal{T}_1(\alpha) = 0$ and $\mathcal{T}_2(\mathcal{T}_1(\alpha)) = 4$, thus $O_0 = (\mathcal{N}(0, K_0), \mathcal{N}(1, K_0))_{2^4}$.
(2) If $\alpha \in \{(0,1,1,0)_2, (1,1,1,0)_2\}$ then $\mathcal{T}_1(\alpha) = 4$ and $\mathcal{T}_2(\mathcal{T}_1(\alpha)) = 4$, thus $O_0 = (\mathcal{N}(0, K_4), \mathcal{N}(1, K_4))_{2^4}$.
(3) If $\mathcal{T}_1(\alpha) = c$, both multiplexers select the same key-byte. We call this a *read-collision* in K_c.

In the following, we describe how to obtain K_0 and K_4 with high probability, which is then leveraged in a second step in order to guess the remaining 48 bits of K in an efficient way.

The key idea to derive K_0 is to examine keystream bytes $(Z_i, Z_{i-1}, Z_{i-8})_{2^8}$ with $i \in \{8, 23, 38, ...\}$ in order to detect when a read-collision in K_0 has happened during the generation of Z_i. Please note that due to our choice of i this

$$Z_8 = Z_8^{(N)}, Z_{23} = Z_8^{(N+1)}, Z_{38} = Z_8^{(N+2)}, ...$$

holds, i.e., for each i we already know that the lower multiplexer has selected K_0. In general, if the desired read-collision has happened in the $\mathcal{F}$-component, the outputs of the $\mathcal{F}$-component are

$$\begin{aligned} O_0 &= (p_3 \oplus \alpha_3, p_2 \oplus \alpha_2, p_1 \oplus \alpha_1, p_0 \oplus \alpha_0, K_{0,7}, K_{0,6}, K_{0,5}, K_{0,4})_2, \\ O_1 &= (K_{0,7} \oplus p_7 \oplus \alpha_3, K_{0,6} \oplus p_6 \oplus \alpha_2, K_{0,5} \oplus p_5 \oplus \alpha_1, K_{0,4} \oplus p_4 \oplus \alpha_0)_2, \end{aligned}$$

and the subsequent outputs of $\mathcal{G}$ are

$$\begin{aligned} O'_0 = (&p_3 \oplus \alpha_3 \oplus p_0 \oplus \alpha_0 \oplus S_{7,5}, p_3 \oplus \alpha_3 \oplus p_2 \oplus \alpha_2 \oplus p_0 \oplus \alpha_0 \oplus S_{7,7}, \\ &p_3 \oplus \alpha_3 \oplus S_{7,4}, p_1 \oplus \alpha_1 \oplus S_{7,6}, \\ &K_{0,7} \oplus p_7 \oplus \alpha_3 \oplus K_{0,5} \oplus p_5 \oplus \alpha_1 \oplus K_{0,4} \oplus p_4 \oplus \alpha_0, \\ &K_{0,7} \oplus p_7 \oplus \alpha_3 \oplus K_{0,4} \oplus p_4 \oplus \alpha_0)_2, \\ O'_1 = (&K_{0,7} \oplus K_{0,4} \oplus S_{7,1}, K_{0,7} \oplus K_{0,6} \oplus K_{0,4} \oplus S_{7,3}, \\ &K_{0,7} \oplus S_{7,0}, K_{0,5} \oplus S_{7,2}, \\ &K_{0,6} \oplus p_6 \oplus \alpha_2, \\ &K_{0,4} \oplus p_4 \oplus \alpha_0)_2. \end{aligned}$$

Considering the $\mathcal{H}$-component, we also know that

$$Z_i = (\mathcal{S}_2(O'_1), \mathcal{S}_6(O'_0))_{2^4}$$

holds.

In order to determine K_0, we examine the inputs and outputs of $\mathcal{S}_6$ and $\mathcal{S}_2$ in the $\mathcal{H}$-component, starting with $\mathcal{S}_6$. Due to the reordering of the DES S-boxes, the column of $\mathcal{S}_6$ is selected by the four most-significant bits of O'_0. If we assume a collision in K_0 has happened while generating Z_i, we can compute these most-significant bits due to the fact that

$$S_7 = Z_{i-8} \quad \text{and} \quad p = Z_{i-1}$$

are also known for all of our choices of i. If, for $\alpha \in \{(0,0,1,0)_2, (1,0,0,1)_2\}$ the lower nibble of Z_i is found in the row with index β, a collision may indeed have happened and the lower two bits of O'_0 must be $(\beta_1, \beta_0)_2$, which implies

$$\begin{aligned} K_{0,7} \oplus K_{0,5} \oplus K_{0,4} &= \beta_1 \oplus p_7 \oplus \alpha_3 \oplus p_5 \oplus \alpha_1 \oplus p_4 \oplus \alpha_0, \\ K_{0,7} \oplus K_{0,4} &= \beta_0 \oplus p_7 \oplus \alpha_3 \oplus p_4 \oplus \alpha_0. \end{aligned}$$

Here we gain "some" information about the bits of K_0, $K_{0,5}$ can even be computed. We can then use the output of $\mathcal{S}_2$ to verify whether a collision has happened for the particular α we used above. Due to the structure of the S-box, there are only four 6-bit inputs γ with

$$\mathcal{S}_2(\gamma) = (Z_{i,7}, Z_{i,6}, Z_{i,5}, Z_{i,4})_2.$$

Due to our partial knowledge about $(K_{0,4}, K_{0,5}, K_{0,7})_2$ we can test for each γ whether the following relations hold:

$$\begin{aligned} \gamma_5 &\overset{?}{=} \beta_0 \oplus p_7 \oplus \alpha_3 \oplus p_4 \oplus \alpha_0 \oplus S_{7,1}, \\ \gamma_4 \oplus \gamma_1 &\overset{?}{=} \beta_0 \oplus p_7 \oplus \alpha_3 \oplus p_4 \oplus \alpha_0 \oplus S_{7,3} \oplus p_6 \oplus \alpha_2, \\ \gamma_3 \oplus \gamma_0 &\overset{?}{=} \beta_0 \oplus p_7 \oplus \alpha_3 \oplus S_{7,0}, \\ \gamma_2 \oplus \gamma_5 &\overset{?}{=} \beta_1 \oplus p_7 \oplus \alpha_3 \oplus p_5 \oplus \alpha_1 \oplus p_4 \oplus \alpha_0 \oplus S_{7,1} \oplus S_{7,2}. \end{aligned}$$

If all of these relations hold for one γ, we can be sure with sufficiently high probability that a read-collision has indeed happened. A probable hypothesis for K_0 is now given by

$$(\gamma_3 \oplus S_{7,0}, \gamma_1 \oplus p_6 \oplus \alpha_2, \gamma_2 \oplus S_{7,2}, \gamma_0 \oplus p_4 \oplus \alpha_0, p_3 \oplus \alpha_3, p_2 \oplus \alpha_2, p_1 \oplus \alpha_1, p_0 \oplus \alpha_0)_2.$$

Our method detects all read-collisions, but there may also be false positives, therefore the process described above must be iterated for a few times for different portions of the keystream. Typically, over time, one or two hypotheses occur more often than others and distinguish themselves quite fast from the rest. Experiments show that about a dozen key-frames are usually enough so that the correct key-byte is among the first two hypotheses. The principle we outlined above not only works for K_0, it also allows to recover the value of K_4 when $\alpha \in \{(0,1,1,0)_2, (1,1,1,0)_2\}$, $i \in \{12, 27, 42, ...\}$ are chosen appropriately.

In the following we assume that we have obtained a set of hypotheses for K_0—we might also have K_4, but this improves the efficiency of the remainder of the attack only slightly. Based on these hypotheses, starting with the most plausible one, we can brute-force the remaining key-bytes separately. Please note that the following process will only produce the correct key, if our hypothesis for K_0 was correct. To obtain $K_1, ..., K_7$ we examine a few keystream-bytes for a second time, while focusing on the $\mathcal{F}$-component. For each K_j with $j \in \{0, 1, ..., 7\}$ for which we already have a hypothesis, we can use the corresponding key-stream bytes $(Z_{i+j}, Z_{i+j-1}, Z_{i+j-8})_{2^8}$ with $i \in \{8, 23, 38, ...\}$ to compute

$$\alpha = \mathcal{N}(j \bmod 2, K_j \oplus Z_{i+j-1}).$$

If we do not already have a plausible hypothesis for K_k with $k = \mathcal{T}_1(\alpha)$, we can simply try out all possible values $\delta \in \{0, 1, ..., 255\}$ and compute the output of the cipher. If we find for one value that the output equals Z_{i+j} we keep δ as hypothesis for K_k. This can be repeated for a few different i until a hypothesis for the full key has been recovered. Since the validity of the full hypothesis solely depends on the correctness of K_0, we must verify each key candidate by generating and comparing keystream.

The overall complexity of this attack depends on how many hypotheses for K_0 are used to derive the remaining key. Given $15 - 20$ key-frames, the correct byte for K_0 is usually ranked as best hypothesis so deriving the complete key means testing

$$(7 \cdot 2^8)/2 \approx 2^{10}$$

single byte hypotheses for the missing bytes (on average). Clearly, a keystream/time trade-off is possible: The more key-frames are available to test hypotheses for K_0, the more the right hypothesis distinguishes itself from all others. As a matter of fact, the most extreme trade-off is simply trying all 2^8 possible values for K_0 (without even ranking them), which reduces the required amount of known keystream to about 400–500 bits but increases the computational complexity to

$$(7 \cdot 2^8 \cdot 2^8)/2 \approx 2^{18}$$

guesses on average.

7. CONCLUSION AND IMPLICATIONS

In this paper, we merge prior work [Driessen et al. 2012] on the analysis of ETSI's satellite communication standards GMR-1 and GMR-2 with a practical attack on the Thuraya network. Reverse-engineering the firmware of two satphones revealed that the ciphers used in these standards are surprisingly weak. While A5-GMR-1 is a modification of the A5/2 cipher (used in GSM), the second stream cipher we found is an entirely proprietary design. Both ciphers could be broken by us; A5-GMR-1 in a ciphertext-only setting, while we require a handful of keystreams for A5-GMR-2. The recovered algorithms as well as the presented attacks were validated experimentally.

Compared to our prior work, we improve the ciphertext-only attack on GMR-1 by a factor of 2^{11}. We do this by targeting a different channel and adapting the attack in order to obtain more equations (thus guessing less bits of R_1, R_2 and R_3) from multiple, consecutive TCH3 frames. We describe hard- and software requirements, as well as the necessary information regarding the configuration of the Thuraya network, which allowed us to execute the proposed attack. We demonstrate the practicability of our attack by obtaining session keys from Thuraya downlink data within half an hour (on average). Thus, for GMR-1, we have documented all steps from performing black-box analysis of the system to finally executing a very efficient real-world attack. The entire process took approx. 6 months, the cost of the required equipment is $\$5\,000$, while storage requirements (400 GB for precomputed data) and computational costs are negligible. These facts, together with our discussion of uplink interception at a distance of 5 Km (or by indirect means in the C-Band), show that interception of GMR-1-based communication is entirely within reach of any attacker with only modest financial means but sufficient dedication.

This work does not improve our attack on GMR-2, which uses (in the configuration optimized for keystream) a handful of keystream frames and requires 2^{18} guesses. Very recently, Li *et al.* presented a different attack [Li et al. 2013], trading computation time for keystream. As a result, only one frame of keystream is required and the complexity increased by a factor of 2^{10}. This variant is still a known-plaintext attack and validating the feasibility of obtaining keystream from a real network (such as Inmarsat's network) is the next logical step. To perform this task, access to (and understanding of) live data from such a network is required. A presentation [Ortega and Muniz 2012] indicates that a framework to obtain this data exists. The authors were so kind to provide us with sample data, but the setup used to acquire it proved to introduce some errors on its own, which hampers analysis. However, given some time to improve the analysis framework as well as understand the structure of occuring plaintexts, real-world attacks building on these tools are very likely to occur in the future.

ACKNOWLEDGMENTS

Understanding existing, closed-source communication infrastructures, their security aspects and implications is inherently important and only made possible by dedicated individuals. Therefore, we gratefully acknowledge that implementing and carrying out our attack would not have been possible (in the given timeframe) without the availability of the OsmocomGMR [Munaut 2012] project.

Furthermore, we would like to thank Rainer Kronberger and Stephan Werker of the High Frequency Laboratory of Cologne University of Applied Science for supporting our work with measuring and approximating the uplink reception distance, thus greatly helping to establish a better understanding of the risk of direct uplink interception.

Finally, we want to thank the anonymous reviewers for their insightful comments, which helped to improve this work.

REFERENCES

BARKAN, E., BIHAM, E., AND KELLER, N. 2003. Instant Ciphertext-Only Cryptanalysis of GSM encrypted communication. In *International Crytology Conference (CRYPTO)*. 600–616.

BARKAN, E., BIHAM, E., AND KELLER, N. 2008. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. *Journal of Cryptology 21*.

BIHAM, E. AND DUNKELMAN, O. 2000. Cryptanalysis of the A5/1 GSM Stream Cipher. In *Indocrypt*.

BIRYUKOV, A., SHAMIR, A., AND WAGNER, D. 2000. Real Time Cryptanalysis of A5/1 on a PC. In *Fast Software Encryption (FSE)*.

BOGDANOV, A., EISENBARTH, T., AND RUPP, A. 2007. A Hardware-Assisted Realtime Attack on A5/2 Without Precomputations. In *Cryptographic Hardware and Embedded Systems (CHES)*.

BRICENO, M., GOLDBERG, I., AND WAGNER, D. 1999. A pedagogical implementation of the GSM A5/1 and A5/2 “voice privacy” encryption algorithms. Originally published at http://www.scard.org, mirror at `http://cryptome.org/gsm-a512.htm`.

DRIESSEN, B., HUND, R., WILLEMS, C., PAAR, C., AND HOLZ, T. 2012. Don't Trust Satellite Phones: A Security Analysis of Two Satphone Standards. In *IEEE Symposium on Security and Privacy*. 128–142.

DUNKELMAN, O., KELLER, N., AND SHAMIR, A. 2010. A Practical-Time Related-Key Attack on the KASUMI Cryptosystem Used in GSM and 3G Telephony. In *International Crytology Conference (CRYPTO)*.

EKDAHL, P. AND JOHANSSON, T. 2003. Another Attack on A5/1. *IEEE Transactions on Information Theory 49,* 1.

ETSI. 2001a. ETSI TS 101 376-3-2 V1.1.1 (2001-03); GEO-Mobile Radio Interface Specifications; Part 3: Network specifications; Sub-part 2: Network Architecture; GMR-1 03.002. Tech. rep.

ETSI. 2001b. ETSI TS 101 376-3-9 V1.1.1 (2001-03); GEO-Mobile Radio Interface Specifications; Part 3: Network specifications; Sub-part 9: Security related Network Functions; GMR-1 03.020. Tech. rep.

ETSI. 2001c. ETSI TS 101 377-3-10 V1.1.1 (2001-03); GEO-Mobile Radio Interface Specifications; Part 3: Network specifications; Sub-part 9: Security related Network Functions; GMR-2 03.020. Tech. rep.

ETSI. 2001d. ETSI TS 101 377-5-3 V1.1.1 (2001-03); GEO-Mobile Radio Interface Specifications; Part 5: Radio interface physical layer specifications; Sub-part 3: Channel Coding; GMR-2 05.003. Tech. rep.

ETSI. 2002. ETSI TS 101 376-5-3 V1.2.1 (2002-04); GEO-Mobile Radio Interface Specifications; Part 5: Radio interface physical layer specifications; Sub-part 3: Channel Coding; GMR-1 05.003. Tech. rep.

GOLIC, J. D. 1997. Cryptanalysis of alleged A5 stream cipher. In *Proceedings of the 16th annual international conference on Theory and application of cryptographic techniques*. EUROCRYPT'97. Springer-Verlag, 239–255.

JIM GEOVEDI AND RAOUL CHIESA. 2011. Hacking a Bird in the Sky. In *HITBSecConf, Amsterdam*.

JOSE FRIDMAN, ANALOG DEVICES. How to optimize H.264 video decode on a digital baseband processor.

LI, R., LI, H., LI, C., AND SUN, B. 2013. A Low Data Complexity Attack on the GMR-2 Cipher Used in the Satellite Phones. In *International Workshop on Fast Software Encryption (FSE)*.

MARAL, G. AND BOUSQUET, M. 2009. *Satellite Communications Systems: Systems, Techniques and Technology* 5 Ed. John Wiley & Sons.

MATOLAK, D., NOERPEL, A., GOODINGS, R., STAAY, D., AND BALDASANO, J. 2002. Recent progress in deployment and standardization of geostationary mobile satellite systems. In *Military Communications Conference (MILCOM)*.

MUNAUT, S. 2012. OsmocomGMR.

NOHL, K. AND PAGET, C. 2009. GSM: SRSLY? 26th Chaos Communication Congress.

ORTEGA, A. AND MUNIZ, S. 2012. Satellite baseband mods: Taking control of the InmarSat GMR-2 phone terminal. ekoparty Security Conference. `http://www.groundworkstech.com/blog/ekoparty2012satellitebasebandmods`.

OSMOCOMGMR. 2012. Thuraya SO-2510.

PETER. 2013. Airborne satellite weather data.

PETROVIC, S. AND FUSTER-SABATER, A. 2000. Cryptanalysis of the A5/2 Algorithm. Tech. rep. `http://eprint.iacr.org/2000/052`.

TBS. 2012. The Satellite Encyclopedia.

TEXAS INSTRUMENTS. 2012. The OMAP 5910 Platform.

WELTE, H. 2010. Anatomy of contemporary GSM cellphone hardware.

WRIGHT, D. 1995. Reaching out to remote and rural areas: Mobile satellite services and the role of Inmarsat. *Telecommunications Policy 19,* 2, 105 – 116.

ZIV, J. AND LEMPEL, A. 1977. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory 23,* 3, 337 – 343.

Received December 2012; revised May 2013; accepted November 2013