IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 3, pp. 217–238. DOI:10.46586/tosc.v2022.i3.217-238

Breaking HALFLOOP-24

Marcus Dansarie[1,2], Patrick Derbez[3], Gregor Leander[4] and Lukas Stennes[4]

[1] Swedish Defence University, Stockholm, Sweden
marcus.dansarie@fhs.se
[2] University of Skövde, Skövde, Sweden
[3] Univ Rennes, Centre National de la Recherche Scientifique (CNRS), Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Rennes, France
patrick.derbez@irisa.fr
[4] Ruhr University Bochum, Bochum, Germany
gregor.leander@rub.de, lukas.stennes@rub.de

Abstract. HALFLOOP-24 is a tweakable block cipher that is used to protect automatic link establishment messages in high frequency radio, a technology commonly used by government agencies and industries that need highly robust long-distance communications. We present the first public cryptanalysis of HALFLOOP-24 and show that HALFLOOP-24, despite its key size of 128 bits, is far from providing 128 bit security. More precisely, we give attacks for ciphertext-only, known-plaintext, chosen-plaintext and chosen-ciphertext scenarios. In terms of their complexities, most of them can be considered practical. However, in the real world, the amount of available data is too low for our attacks to work. Our strongest attack, a boomerang key-recovery, finds the first round key with less than 2^{10} encryption and decryption queries. In conclusion, we strongly advise against using HALFLOOP-24.

Keywords: HF Radio · ALE · HALFLOOP · Boomerang

1 Introduction

Protocols for automatic link establishment (ALE) were developed during the 1980s to simplify communications via high frequency (HF) radio. The current ALE protocols are described in the US standards MIL-STD-188-141 and FED-STD-1045 and by NATO in STANAG 4538. To prevent spoofing and to protect the transmitted data, ALE includes an optional linking protection mode. Second (2G) and third generation (3G) ALE use the same block cipher, named SoDark, which operates on 24- and 48-bit states under a 56-bit key [JKF+12]. In [Dan21], Dansarie described several practical key-recovery attacks against the 8-round version of this cipher, corresponding to the exact number of rounds standardized for 2G ALE. However, a key length of 56 bits is way too short to offer a decent security level against modern computers and GPUs. For that reason, a replacement for SoDark has been specified since 2017. This new block cipher, named HALFLOOP, has been standardized in the latest revision of MIL-STD-188-141 with a key size of 128 bits and block sizes of 24, 48 and 96 bits [DoD17].

A Brief Introduction to HF Radio and ALE Modern technologies such as wireless networking, mobile telephony and satellite-based internet services make it possible to transmit data quickly and easily from virtually any place on earth. Despite this, HF radio, a century-old technology, is still in active use in many applications. HF radio, also known as shortwave radio, uses frequencies between 3 and 30 MHz. These frequencies enable skywave propagation, where the radio signals are reflected by electrically charged

Received: 2022-06-01 Accepted: 2022-08-01 Published: 2022-09-09

particles in the upper atmosphere. Through this effect, two radios can communicate across very large distances without any external infrastructure. This makes HF radio attractive to users who need communications to work even when conventional infrastructure is unavailable, such as after disasters, in war and in the polar regions where geostationary satellites can not be reached. Examples of users include the military, diplomatic services, disaster management agencies, humanitarian non-governmental organizations, and the air and maritime industries. Large HF antennas are a common sight on embassies throughout the world.

Skywave propagation is heavily dependent on a number of constantly changing factors, such as season, time of day and space weather. Additionally, transmitter and receiver locations as well as technical characteristics of the radio equipment also affect propagation. For that reason, HF radio has historically required trained and experienced operators. The first attempts at reducing this dependence on operators were made during the 1980s, when the first ALE systems were developed. Since different HF radio manufacturers developed their own standards, they were not interoperable. This was solved with the introduction of 2G ALE, which became a US military standard in 1988. 3G ALE was introduced in 1999. Recently, in 2017, a fourth generation, known as Wideband ALE, was introduced. Despite the introduction of later generations, 2G ALE still remains in active use and, due to its age and proliferation, is the de facto world standard for interoperable ALE communications.

An ALE-enabled radio establishes a link to another radio by selecting a suitable frequency according to a propagation model and then transmitting a call frame. If the frequency is good, the other radio receives the frame and the two radios perform handshaking to set up a link. Once set up, the link can be used for voice communication or data transmission. Radios that are not in active use continuously scan a list of assigned frequencies in order to detect incoming calls. Apart from call frames, ALE radios can also transmit and receive frames for other purposes. For example, radios can perform soundings, whereby they regularly transmit frames on different frequencies so that other radios in the network can measure and record the current propagation characteristics. There is also a facility for orderwire communications, i.e. transmission of short text messages between operators [JKF$^+$12]. Section 4 contains a more in-depth description of the 2G ALE linking process and the plaintexts involved that are of interest for the attacks presented here.

The encryption of ALE frames is known as linking protection. It only protects ALE frames sent between two radios and not the established link itself. It is primarily meant to protect unauthorized users from establishing links with radios in a network or interfering with established links, for example by disconnecting them [Joh16]. While traditional jamming can obtain the same results, it requires orders of magnitude more transmission power than simply sending forged disconnection frames. Additionally, encryption of ALE frames also protects the network from certain types of traffic analysis, which is the analysis of operating data such as network structure, frequencies, callsigns and schedules [Cal89]. The first ALE standard did not specify a cipher, but specified how to integrate a stream cipher with ALE [DoD88]. Later standards introduced the Lattice/SoDark cipher, which is now recommended to be replaced with HALFLOOP whenever possible [DoD17].

Our contribution In this paper we present attacks on the full HALFLOOP-24 cipher. The main weakness of HALFLOOP-24 comes from its key schedule. While it is common that block ciphers have small internal states compared to their keys [DR02, BJK$^+$16], such constructions require special attention in the round-key generation process to ensure that all the key bits of the master key are well mixed with the message. Otherwise the security of the cipher can be much lower than expected. For instance, in [BK09], Biryukov and Khovratovich described related-key attacks against the full versions of both AES-192 and AES-256, relying on the low diffusion of their respective key schedules. The 24-bit version of HALFLOOP actually suffers from the exact same problem. Furthermore, since

Table 1: Summary of attacks on HALFLOOP-24 presented in this paper.

Setting	Time	Data	Memory	#Recovered Key Bits	Section
Ciphertext only	2^{87}	2^{38}	2^{63}	128	4.4
Known-plaintext	2^{56}	2^{37}	2^{16}	128	3
Chosen-plaintext	2^{56}	2^{18}	2^{16}	128	3
Chosen-ciphertext	2^{10}	2^{10}	1	24	5

HALFLOOP is a tweakable block cipher and the tweak is simply xored to the key, there is a generic related-tweak key-recovery attack. For HALFLOOP-24, this means that we can recover the key with time, data and memory complexity 2^{64}. To do so, we would simply query a message for all 2^{64} tweaks, store the resulting ciphertexts in a table and then brute force the 64 bits of key not influenced by the tweak. However, as we show in this work, there are significantly more powerful attacks against HALFLOOP-24.

Our attacks rely on differential cryptanalysis which is one of the most powerful cryptanalysis techniques. It was proposed by Biham and Shamir in [BS91] and has generated much attention since then. The aim of differential cryptanalysis is to study the propagation of differences through a cipher to highlight unexpected behaviors compared to a random permutation. Typically it concerns the existence of a differential characteristic with a high probability but we may also search for impossible transitions [BBS99]. In particular, HALFLOOP-24 is very weak against boomerang attacks [Wag99] and there exists a related-tweak boomerang distinguisher of probability 1 against its full version.

Results The attacks presented in this paper break HALFLOOP-24. However, because the amount of data is extremely limited in the real world, we do not expect that these attacks can be used in practice. Table 1 shows a summary of the attacks and their complexities. Note that the complexity of our ciphertext-only attack heavily depends both on how callsigns are assigned and on the traffic intensity in the attacked network. The complexities we provide in Table 1 assume that callsigns are randomly assigned and that the rate of traffic is very high. The complexities improve significantly as the randomness of callsigns decreases. Ignoring the actual complexities, ciphertext-only attacks are the most simple ones since an attacker only needs a radio to eavesdrop ciphertexts. In the case of ALE, plaintexts consist mostly of callsigns which can be obtained with moderate effort. Chosen-plaintext and chosen-ciphertext attacks require (temporary) access to a radio that holds the desired key. All attacks are significantly more efficient than brute force. Therefore, we strongly advise against the use of HALFLOOP-24. We implemented and verified our chosen-plaintext and chosen-ciphertext attacks. The source code is available at `https://doi.org/10.5281/zenodo.7043329`.

Structure of the paper The next section gives the HALFLOOP-24 specification and clarifies related notation. In Section 3, we give an attack against HALFLOOP-24 in a theoretical model and in Section 4 we describe how we can adapt this attack for the real world. After that, Section 5 presents our boomerang key-recovery before we conclude the paper in Section 6.

2 Description of HALFLOOP-24

Here, we briefly describe the tweakable block cipher HALFLOOP-24. HALFLOOP-24 operates on 24-bit blocks and features a 128-bit key and 64-bit tweak. We write

$c = \text{HL-24}_k(p, t)$ to denote the encryption of a plaintext $p \in \mathbb{F}_2^{24}$ under the key $k \in \mathbb{F}_2^{128}$ and the tweak $t \in \mathbb{F}_2^{64}$ which results in the ciphertext $c \in \mathbb{F}_2^{24}$.

HALFLOOP-24 borrows many operations from AES. The state consists of three bytes, arranged in a 3×1 matrix (instead of 4×4 for AES). The SBox is the same as for AES. Instead of `ShiftRows`, the second byte is rotated by six and the third byte is rotated by four bits to the left. This operation is called `RotateRows`. For `MixColumns`, we multiply the state with $c(x) = x^2 + 2x + 9$ modulo $x^3 + 1$, which can be seen as a 3×3 MDS matrix over $\mathbb{F}_{2^8}$. Note that [DoD17] gives $x^4 + 1$ as modulus which apparently is a typing error since the test vectors indeed work for $x^3 + 1$. Finally, as for AES-128, the number of rounds is ten and the last round does not involve the `MixColumns` operation.

Key Schedule The key schedule is the same as for AES-128 with one crucial difference. For HALFLOOP-24, there is a 64-bit tweak t (called seed) that is xored to the master key k before the key schedule takes place. Furthermore, we only need 24-bit round keys and hence a far shorter expansion. We depict the key schedule in Fig. 1. The internal function g is the same as for AES, i.e. it applies the AES SBox to all four input bytes, rotates the results by one byte and adds a round constant to the most significant byte. For more details, we refer to [DR02]. Note that, if we consider two tweaks t_1 and t_2 for the same key k, then we obtain different round keys. However, the differences in the round keys depends only on the difference $\Delta t = t_1 \oplus t_2$ except for the last byte of rk_{10}. Hence, we often transform a round key for a tweak t to the round key we would obtain for the all-zero tweak and call this a normalized round key $\widehat{rk}$.

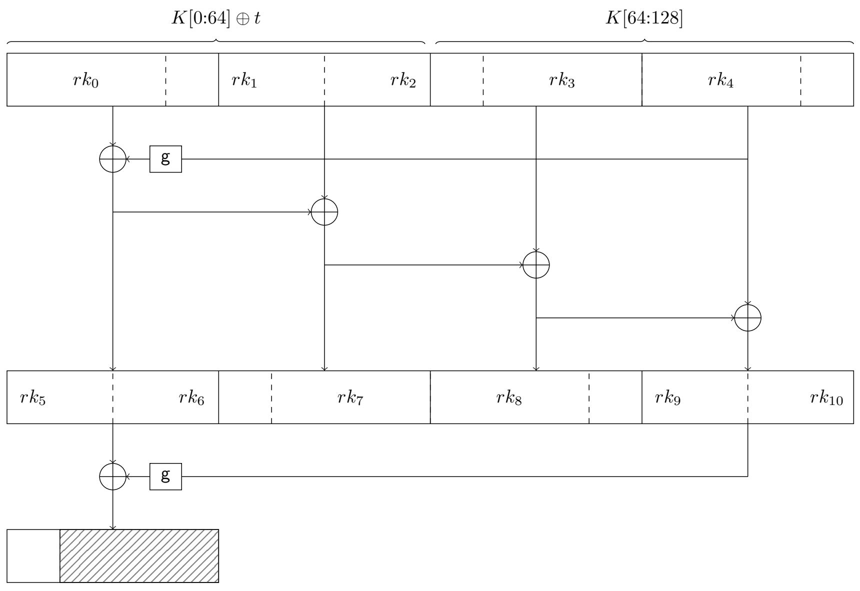


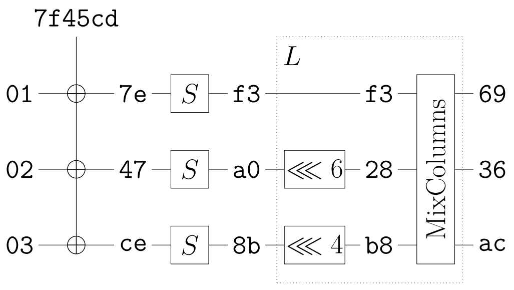
Figure 1: HALFLOOP-24 Key Schedule.

Example To make the notation, especially the ordering of bytes clear, we give a test vector from [DoD17] as an example. Consider key k, tweak t and plaintext p as follows:

$$k = \texttt{0x2b7e151628aed2a6abf7158809cf4f3c}$$
$$t = \texttt{0x543bd88000017550}$$
$$p = \texttt{0x010203}$$

The first round key is `0x7f45cd`. When we arrange p into the state matrix, `0x01` is the first, i.e. topmost entry. We often refer to it as $p[0{:}8]$ or the most significant byte of p.

Analogously, $p[8{:}16] = $ `0x02` and $p[16{:}24] = $ `0x03`. We use the same notation for (round) keys and tweaks, i.e. we always count bits from the left-hand side, inspired by the notation in the Python programming language. Below we give the first steps of HL-$24_k(p, t)$, i.e. the encryption of plaintext p with key k and tweak t. For convenience, we combine the `RotateRows` and `MixColumns` steps into a single linear layer L in the following.



3 A Related-Tweak Attack

Our attack builds upon some simple observations. First, the state size of HALFLOOP-24 is only 24 bits. Therefore, it is not too unlikely that multiple round key differences cancel each other. Second, except for the least significant byte of the last round key, a difference in the tweak only has linear influence on the round key differences. Hence, we can easily craft tweaks which skip multiple rounds in a differential attack. Lastly, 10 rounds seem not adequate given that there is a simple meet-in-the-middle attack against the full cipher. This is possible because guessing 5 round keys is easier than guessing the complete key.

We visualize the influence of the tweak difference on the round keys in Fig. 2. There, an empty white block indicates that the round key is not influenced by this part of the tweak. A gray block shows that the difference in the byte given in the box is given by the byte of the tweak difference in the head of that column. When there is a light and a dark gray block with the same byte number, the round key difference is the xor of both. Notice, we can use Fig. 2 in two ways. On the one hand, we can see that e.g.

$$\Delta rk_6 = \Delta t[16{:}24] || \Delta t[24{:}32] || (\Delta t[32{:}40] \oplus \Delta t[0{:}8]).$$

On the other hand, the visualization makes clear that differences in the least significant byte of the tweak only influence $\Delta rk_2, \Delta rk_7, \Delta rk_9$ and Δrk_{10}. But beware that the difference in the least significant byte of rk_{10} is not visualized there. As depicted in Fig. 1, the last byte of the round key is derived as the xor of the second byte of rk_5 and the first output byte of the g function applied to the bytes of rk_9 and rk_{10}. Hence, it does not only depend on the difference in the tweak, but on the concrete values of the tweaks as well as on the value of the ninth round key $rk_9[16{:}24]$. Since this itself depends on the choice of the tweak, to avoid ambiguity, we use $\widehat{rk}_9$ to denote the ninth round key obtained by running the key schedule with the all zero tweak. Then, for two tweaks t_1, t_2, we have

$$\begin{aligned} rk_{9,t_1}[16{:}24] &= \widehat{rk}_9[16{:}24] \oplus t_1[8{:}16] \oplus t_1[40{:}48] \\ rk_{9,t_2}[16{:}24] &= \widehat{rk}_9[16{:}24] \oplus t_2[8{:}16] \oplus t_2[40{:}48] \\ \Delta rk_{10}[16{:}24] &= \Delta rk_5[8{:}16] \oplus S(rk_{9,t_1}[16{:}24]) \oplus S(rk_{9,t_2}[16{:}24]). \end{aligned} \tag{1}$$

Now, more concretely, consider a plaintext $x \in \mathbb{F}_2^{24}$, a tweak $t \in \mathbb{F}_2^{64}$ and a difference $\delta \in \mathbb{F}_2^8$. When we encrypt x with tweak t under a key $k \in \mathbb{F}_2^{128}$ and do the same for $x' = x \oplus 0^{16}||\delta$ and $t' = t \oplus 0^{16}||\delta||0^{40}$, we observe the following differences in the round

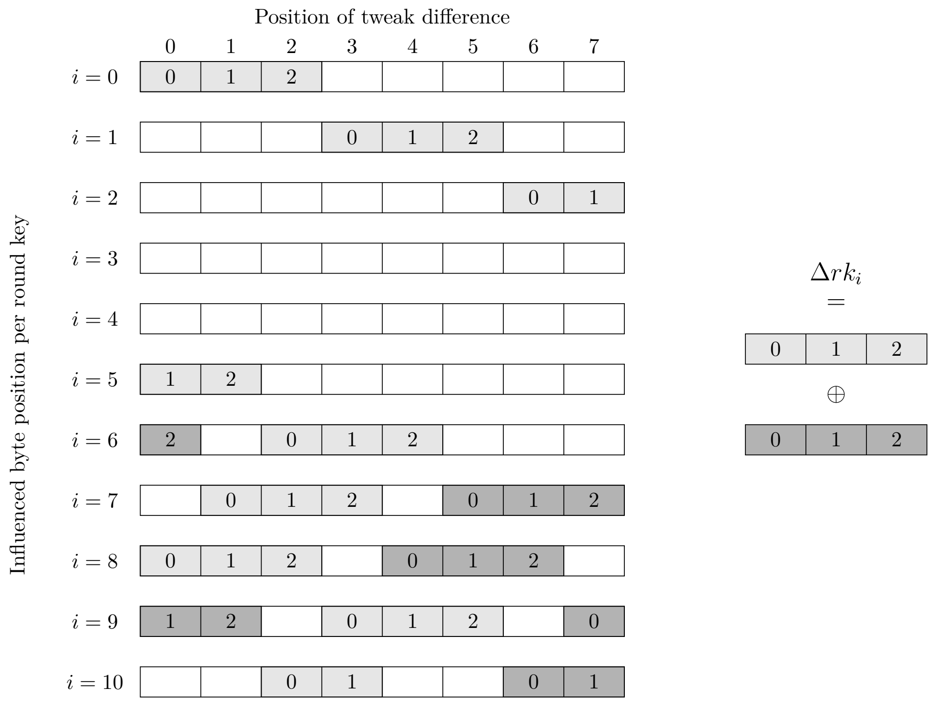


Figure 2: Visualization of the round key differences as a function of the tweak difference.

keys.

$$
\begin{array}{llll}
\Delta rk_0 = 0^{16}||\delta & \Delta rk_1 = 0 & \Delta rk_2 = 0 & \Delta rk_3 = 0 \\
\Delta rk_4 = 0 & \Delta rk_5 = 0 & \Delta rk_6 = \delta||0^{16} & \Delta rk_7 = 0^8||\delta||0^8 \\
\Delta rk_8 = 0^{16}||\delta & \Delta rk_9 = 0 & \Delta rk_{10} = \delta||0^{16} &
\end{array}
$$

The differences in the plaintexts and in rk_0 cancel and hence, there is no state difference after the addition of rk_0. It stays like this until we add rk_6. Now, our hope is that the differences in the state introduced by the difference in rk_6 is canceled by the differences in rk_7 and rk_8 so that after round eight there is no state difference again. The linear layer L is MDS. Hence, the one-byte difference δ in rk_6 is transitioned to a three-byte difference. The same holds in the backward direction for the difference in rk_8. Now, after the addition of rk_7 and the SBox layer those differences must be the same for the cancellation to take place. Therefore, the probability for cancellation is about 2^{-24}. Further notice, this cancellation happens if and only if we observe the ciphertext difference $\delta||0^{16}$ because there is no difference in rk_9 and difference $\delta||0^{16}$ in rk_{10}. Pairs that lead to such a cancellation can only stem from a rather small set of middle states, which eventually allows us to restore the last three round keys.

3.1 Overview of the Attack

We divide our attack into four steps, which we list below. Each step is then discussed in more detail in the following subsections. Fig. 3 gives an overview of the attack. We implemented our attack and executed it in a lab setting. We report the results in Section 3.6.

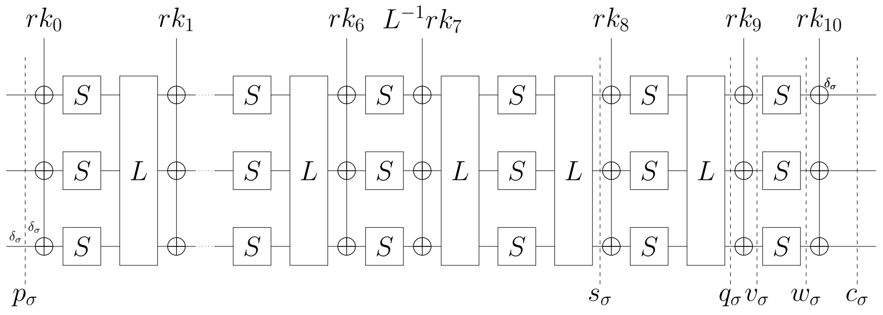


Figure 3: Overview of the attack.

1. **Gather Data:** Gather three tuples $(p_\sigma, t_\sigma, \delta_\sigma, c_\sigma) \in \mathbb{F}_2^{24} \times \mathbb{F}_2^{64} \times \mathbb{F}_2^{8} \times \mathbb{F}_2^{24}$, $\sigma \in \{x, y, z\}$ such that $\delta_\sigma \neq 0$ and the ciphertexts
$$c_\sigma = \text{HL-24}(p_\sigma, t_\sigma)$$
$$c'_\sigma = \text{HL-24}(p_\sigma \oplus 0^{16}||\delta_\sigma, t_\sigma \oplus 0^{16}||\delta_\sigma||0^{40})$$
fulfill
$$\Delta c_\sigma = c_\sigma \oplus c'_\sigma = \delta_\sigma||0^{16}.$$
2. **Build Tables:** We build two tables. T_L stores all middle states (s_x, s_y, s_z) that can lead to a difference-less state after round eight and the corresponding candidates for rk_7. T_R contains the two most significant bytes of the states (v_x, v_y, v_z) immediately before the last SBox layer together with corresponding choices of $rk_{10}[0{:}16]$.
3. **First Enumeration:** We enumerate all possible rk_8 and $(s_x, s_y, s_z) \in T_L$ to compute forward to (q_x, q_y, q_z). Then, we check for hits in T_R to restore rk_9 and rk_{10}.
4. **Second Enumeration:** We brute force the remaining bits of the key.

3.2 Gathering Data

Our attack needs three tuples $(p_x, t_x, \delta_x, c_x)$, $(p_y, t_y, \delta_y, c_y)$ and $(p_z, t_z, \delta_z, c_z)$. For each $\sigma \in x, y, z$, this constitutes a plaintext pair (p_σ, p'_σ), a tweak pair (t_σ, t'_σ) and a ciphertext pair (c_σ, c'_σ) with corresponding differences
$$\Delta p_\sigma = 0^{16}||\delta_\sigma \qquad \Delta t_\sigma = 0^{16}||\delta_\sigma||0^{40} \qquad \Delta c_\sigma = \delta_\sigma||0^{16}.$$

Depending on whether we are in a known-plaintext or chosen-plaintext setting, there are different approaches to obtain these. Here, we give rough theoretical estimation for the amount of data needed. In Section 4, we discuss what data we would use in practice and how long it would take to obtain it.

In the known-plaintext setting, two random plaintexts fulfill the input difference $0^{16}||\delta$ with probability 2^{-16} since δ is arbitrary. The tweak difference is fulfilled with probability 2^{-64} but, as we discuss in more detail later, 30 bits of the tweak can be assumed to be constant for a given frequency. As described above, the spontaneous cancellation happens with probability roughly 2^{-24} and if and only if the ciphertexts have difference $\delta||0^{16}$. Hence, the overall probability that a random pair is good is 2^{-74}. Therefore, considering birthday effects, our attack needs about 2^{37} random known plaintexts.

In the chosen-plaintext setting, we could simply query random plaintexts and tweaks and then add the right input difference. Thereby, only the output difference must be conformed which happens with probability 2^{-24} and hence we need around $3 \cdot 2^{24}$ queries on average. Notice that we can improve this using birthday effects again. For a random plaintext p and a random tweak t, we query $(p \oplus 0^{16} || \delta, t \oplus 0^{16} || \delta || 0^{40})$ for all $\delta \in \mathbb{F}_2^8$ to obtain 2^8 data and therefore 2^{15} pairs that have good input differences. Hence, the probability to obtain at least one pair that also has the correct output difference is approximately 2^{-9}. Repeating this process $2^{10.42}$ times, i.e. $2^{18.42}$ queries, the probability of obtaining at least three good pairs is around 50%, assuming a binomial distribution.

3.3 Building Tables

Building the Left-Hand Table T_L Recall, except for the first and last round, the only round key differences are $\Delta rk_6, \Delta rk_7$ and Δrk_8. In particular, $\Delta rk_9 = 0$. Consequently, the observed output difference $\Delta c_\sigma = \delta_\sigma || 0^{16}$ implies that there is no difference after the addition of rk_8. That is, the differences in rk_6, rk_7 and rk_8 spontaneously canceled each other. Now, consider Fig. 4. It is clear that there must be difference $0^{16} || \delta_\sigma$ before we add rk_8 since this is the difference Δrk_8 for t_σ and t'_σ. Further, we swap the addition of rk_7 and the linear layer and hence transform the round key difference accordingly as $L^{-1}(0^8 || \delta_\sigma || 0^8)$ which we refer to as $\alpha_\sigma || \beta_\sigma || \gamma_\sigma$ as depicted in Fig. 4. Observe that there is no difference in the two least significant bytes of the input to round seven. Hence, when we compute backward, β_σ and γ_σ must cancel the differences induced by the difference in the least significant byte of s_σ. Similarly, $L^{-1}(rk_7)[0{:}8]$ must be such that we obtain difference δ_σ in front of the most significant SBox.

Now, to construct T_L, we first construct three sub tables. For each pair $\sigma \in \{x, y, z\}$, we enumerate all middle states $s_\sigma \in \mathbb{F}_2^{24}$ and check if the differences in round seven in front of the two least significant SBoxes are canceled by β_σ and γ_σ respectively. If so, we compute all possible values of the most significant byte of $L^{-1} rk_7$ such that we obtain difference δ_σ in front of the most significant SBox. We store these in a table $[(\widetilde{rk}_7, s)]_\sigma$ where $\widetilde{rk}_7$ is the normalized most significant byte of $L^{-1} rk_7$. Here, normalized means that we remove the influence of the tweak, i.e., we compute the value that we would obtain with an all-zero tweak. We sort the table by $\widetilde{rk}_7$. All three tables are of size about 2^8 since we enumerate 2^{24} middle states, check a 16-bit condition and on success store the solutions of the equation

$$S^{-1}(u_\sigma \oplus \widetilde{rk}_7) \oplus S^{-1}(u_\sigma \oplus \widetilde{rk}_7 \oplus \delta'_\sigma) = \delta_\sigma$$

where u_σ is the most significant state byte derived by computing s_σ back and hence all values but $\widetilde{rk}_7$ are given. S is the AES SBox, for which it is well-known that this equation has either 0, 2 or 4 solutions. On average, there is one solution and hence the size of roughly 2^8. Now, we combine the three sub tables into T_L. To do so, we iterate all $\widetilde{rk}_7 \in \mathbb{F}_2^8$ and look up s_x, s_y and s_z in the previously generated tables. For all hits, we add an entry $(s_x, s_y, s_z, \widetilde{rk}_7)$ to the table T_L. The table T_L is also of size around 2^8. This is because we are enumerating 2^8 values for $\widetilde{rk}_7$ and the average number of hits is roughly one, since we check an 8-bit condition on 2^8 entries per small table.

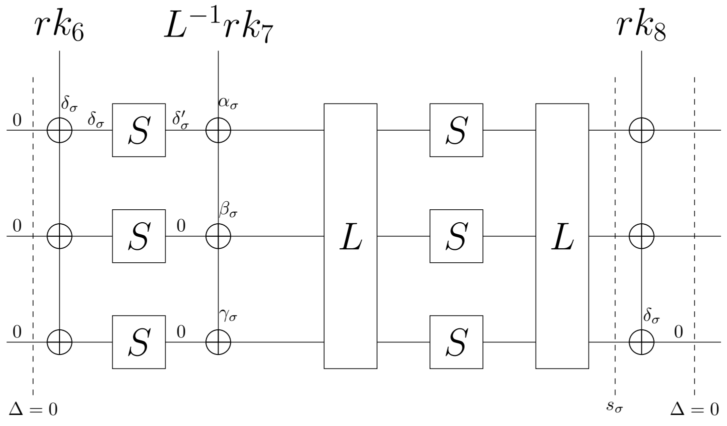


Figure 4: Spontaneous cancellation of differences.

Building the Right-Hand Table T_R The second table T_R is generated as

$$T_R = [((v_x \oplus v_y)[0{:}16], (v_y \oplus v_z)[0{:}16], \widehat{rk}_{10}[0{:}16], v_x[0{:}16]) \mid \forall \widehat{rk}_{10}[0{:}16] \in \mathbb{F}_2^{16}].$$

T_R is sorted by the first two components. Here, v_σ is the state before the last round. Since we are only interested in the two most significant bytes, and the round differences there do only depend on the tweak differences, we only need to enumerate 2^{16} candidates for $\widehat{rk}_{10}$ to build T_R:

$$v_\sigma[0{:}8] = S^{-1}(c_\sigma[0{:}8] \oplus \widehat{rk}_{10}[0{:}8] \oplus t_\sigma[16{:}24] \oplus t_\sigma[48{:}56])$$
$$v_\sigma[8{:}16] = S^{-1}(\mathrm{ROR}(c_\sigma[8{:}16] \oplus \widehat{rk}_{10}[8{:}16] \oplus t_\sigma[24{:}32] \oplus t_\sigma[56{:}64], 6))$$

Here, $\mathrm{ROR}(a, b)$ denotes bitwise right rotation of a by b. Notice, we cannot compute $v_\sigma[16{:}24]$ likewise, since there the round key difference depends also on the least significant byte of rk_9.

3.4 First Enumeration

Now, we connect T_L and T_R by enumerating rk_8. More precisely, we enumerate all $(\widehat{rk}_8, (s_x, s_y, s_z, \widetilde{rk}_7)) \in \mathbb{F}_2^{24} \times T_L$. For each of those tuples, we compute q_σ and search for $((q_x \oplus q_y)[0{:}16], (q_y \oplus q_z)[0{:}16])$ in T_R. For a hit, we restore the candidate for the two most significant bytes of $\widehat{rk}_{10}$ from T_R and compute the candidate for $\widehat{rk}_9[0{:}16]$ as

$$\widehat{rk}_9[0{:}16] = (q_x \oplus v_x)[0{:}16] \oplus t_x[24{:}40] \oplus (t_x[56{:}64] || t_x[0{:}8]).$$

Restoring the least significant byte of rk_9 and rk_{10} is only slightly more complicated. Recall that the difference in the least significant byte of rk_{10} depends on the tweak difference and the value of the least significant byte of rk_9. Hence, we enumerate all possible $\widehat{rk}_9[16{:}24]$ and compute the corresponding differences in $rk_{10}[16{:}24]$. We denote these as Δ_{xy} and Δ_{yz}. Then, we compute all $q_\sigma[16{:}24]$ forward to w_σ and check

$$w_x \oplus w_y \stackrel{?}{=} c_x \oplus c_y \oplus \Delta_{xy}$$
$$w_y \oplus w_z \stackrel{?}{=} c_y \oplus c_z \oplus \Delta_{yz}.$$

If both equations are fulfilled, we restore the least significant byte of the candidate for rk_{10} by normalizing $w_x \oplus c_x$ as in Eq. (1), i.e.

$$\widehat{rk}_{10}[16{:}24] = w_x \oplus c_x \oplus t_x[8{:}16] \oplus S(\widehat{rk}_9[16{:}24]) \oplus S(\widehat{rk}_9[16{:}24] \oplus t_x[8{:}16] \oplus t_x[40{:}48]).$$

Table 2: Average execution times and other metrics averaged over 10 runs on a 16-core computer. The steps refer to the numbered list in Section 3.1. #Queries is the number of chosen-plaintext queries in step 1. #Candidates is the number of key candidates generated in step 3.

Step 1	Step 2	Step 3	Step 4	#Queries	$\lvert T_L \rvert$	#Candidates
0.2 s	2.5 s	693 s	18766 s $\approx$ 5.2 hours	500506 $\approx 2^{18.9}$	280	286

Now we store the candidate $(\widetilde{rk}_7, \widehat{rk}_8, \widehat{rk}_9, \widehat{rk}_{10})$ in a list.

The total cost of this enumeration is $|T_L| \cdot 2^{24} \approx 2^{32}$. The table T_R is of size 2^{16} and we check a 32-bit condition, i.e. we encounter about $2^{24+8+16-32} = 2^{16}$ hits in T_R. For these, we enumerate one more byte and then check a 16-bit condition. Therefore, we expect $2^{16+8-16} = 2^8$ candidates for 80 bits of the key. In other words, we learn 72 bits of the key using time 2^{32} and memory 2^{16}.

Note that we could run the same procedure, i.e. Steps 1-3 of our attack, with two or four instead of three tuples. Then, the expected number of wrong key candidates is $2^{8+24+16-16+8-8} = 2^{32}$ for two and $2^{8+24+16-48+8-24} = 2^{-16}$ for four tuples respectively.

3.5 Second Enumeration

We repeat this step for all candidates from the last step, i.e. around 2^8 for three tuples (2^{32} times for two tuples, and only once for four tuples). The least significant bytes of rk_{10} and rk_9 allow us to compute $rk_5[8{:}16]$. Hence, we are only missing 48 bits of the second 128-bit block of the key schedule. We simply enumerate all possibilities to find the correct key. So, in total, the complexity of this step is 2^{56} (2^{80} and 2^{48} respectively) and hence dominates the costs of our attack.

Notice, a straightforward time-memory trade-off does not seem possible because the missing key bits are, informally speaking, too scattered. Nevertheless, we leave it to future work to find more clever ways to recover the remaining bits of the key. We do not, since 2^{56} is a reasonable complexity, especially for large-scale adversaries, and also this approach does not require any more data. Notice that in total we use six plaintexts and six ciphertexts (of course many more are simply ignored) which is nearly optimal since each pair contains roughly 24 bits of information about the key.

3.6 Experimental Verification

We wrote a bitsliced [Bih97] implementation of the attack for processors with the AVX instruction set and verified our chosen-ciphertext attack in a lab setting. The output of one example run is given in Appendix A. We executed our attack 10 times, each time using four randomly generated good plaintext–ciphertext–tweak tuples, on a 16-core computer to obtain the averaged metrics presented in Table 2. Step 4 clearly dominates the overall complexity. However, since that step is a simple brute force search, parallelization is trivial and we expect it to be at least an order of magnitude faster when implemented on modern GPUs. Hence, we conclude that the cost of the attack is practical even for a small-scale adversary with access to four good tuples.

4 From Theory to Practice

4.1 Plaintexts in 2G ALE

Radio stations in 2G ALE can transmit a number of different frames between them. The most fundamental of these are the frames involved in making a call, i.e. setting up a transmission channel between two radios, since this is the primary purpose of the ALE standards. An ALE network will typically have a number of different assigned frequencies and the radios will scan them sequentially while listening for calls. The typical ALE call scenario is illustrated in Fig. 7. An operator or computer that wishes to start a voice call or transmit a message initiates the call (3). The calling radio then selects a suitable frequency based on a radio propagation model (4) and transmits a three-word call frame on the selected frequency (5). When the called radio receives the call frame, it transmits an equivalent frame in return (6). The three-way handshake is completed by a second frame from the calling radio (7). After this step, both radios will be in the linked state and the radio channel is available for higher-level protocols to use. Either station can end the call. This is done using a single three-word frame (11), after which both radios return to the unlinked state and resume scanning frequencies. Notice that, in all frames, the contents of the first and second words are identical.

All 2G ALE words are 24 bits long. The three words used in the typical calls shown here are TO, TIS and TWAS. They all have the same structure, shown in Fig. 5, and start with three bits that identify the word type. This is followed by three seven-bit characters that represent a station callsign. Instead of the full ASCII character set, 2G ALE only uses the BASIC 38 character set which consists of the uppercase characters A-Z, numbers 0-9, @ and ?. The latter two characters are the null and wildcard characters which are not used in ordinary three-letter callsigns.

1 3	4 10	11 17	18 24
Type	7-bit char	7-bit char	7-bit char

Figure 5: Structure of 2G ALE TO, TIS and TWAS words. The type bits are 010 for TO, 101 for TIS and 011 for TWAS words.

The tweak used in the encryption of any ALE word is generated from the date, time, message word number and transmission frequency, as shown in Fig. 6. All these properties are immediately observable by anyone capable of intercepting frames. This is deliberate, since the ALE standard does not have a method for transferring the tweak. The required tweak differences for the attack described in Section 3 occur when two transmitted words are encrypted with tweaks that differ only in the third byte, i.e. in bits 17–24. This byte contains the four least significant bits of the coarse time (minutes since midnight) and the four most significant bits of the fine time (seconds). This means that, in every 16-minute time window, any call frames that are sent during the same number of seconds modulo 4 will provide three word pairs with the required tweak difference. For example, the first word of a frame transmitted at 1755 kHz on May 8 at 15:57:34 will have the tweak `54 3b d8 80 00 01 75 50` and the first word of a frame transmitted at 1755 kHz on May 8 at 15:59:58 will have the tweak `54 3b fe 80 00 01 75 50`. Similarly, the second and third words in the two frames will have the same differences.

4.2 Gathering Data

As mentioned in Section 3.2, the probability that a pair of words will have the required ciphertext difference is 2^{-24}. If all captured frames have three words, i.e. the lowest possible number of words per frame, the probability that a pair of two frames with the

1 4	5 9	10 20	21 26	27
Month	Day	Coarse time	Fine time	Word

34		37 40	41 44	45 48	49 52	53 56	57 60	61 64
	Zero	100 MHz	10 MHz	1 MHz	100 kHz	10 kHz	1 kHz	100 Hz

Figure 6: Tweak format for HALFLOOP as used in 2G and 3G ALE. The coarse time field holds minutes since midnight and the fine time fields holds seconds in the current minute. The four bit frequency fields are used to encode the frequency as a binary coded decimal number. Bits 35 and 36 are always zero.

required tweak difference also has at least one pair of words with the required ciphertext difference is

$$1 - \left(1 - 2^{-24}\right)^3 \approx 2^{-22.4}.$$

We assume that the lower two bits of the second a radio transmits a message is uniformly random. Then, the number of frames with each of the four possible numbers of seconds modulo 4 will follow a multinomial distribution. An approximation of the expected number of pairs in a 16-minute time window with n captured frames that have the required tweak difference is

$$\left\lfloor 4 \cdot \frac{\left(\frac{n}{4}\right)\left(\frac{n}{4} - 1\right)}{2} \right\rfloor = \left\lfloor \frac{n^2}{8} - \frac{n}{2} \right\rfloor,$$

each with probability $2^{-22.4}$ of having the required ciphertext difference in at least one of the three words. A day has 90 such 16-minute bins.

For example, a 16-minute bin with 96 captured frames (one every 10 seconds), corresponding to very high intensity traffic, is expected to have about 1104 pairs with the required tweak difference. The probability that at least one of those pairs has the required ciphertext difference is $1 - \left(1 - 2^{-22.4}\right)^{1104} \approx 2^{-12.3}$.

4.3 Plaintext differences

Having the required difference in ciphertext and tweak is not enough. The two callsigns in the plaintext must also have the same difference in the last eight bits of the word, i.e. in the last character and the least significant bit of the middle character, as in the ciphertexts and tweaks. The callsigns must be identical in the other 13 bits. The probability of this ultimately depends on how callsigns are assigned in the particular radio network. A network could have no callsigns that differ only in the last eight bits. This would mean that the required plaintext difference can never appear. This would however significantly limit the number of possible callsigns in a network. In the case of randomly assigned callsigns, the probability of a plaintext pair with the same difference as the tweak is $36^{-1} \cdot 18^{-1} \cdot 2^{-8} \approx 2^{-17.3}$. In practice, the probability may actually be higher for a number of reasons. For example, the number of stations in a given network are often not close to the maximum number of possible callsigns, callsigns may be assigned sequentially according to organizational structure (as opposed to randomly), and stations that are close to each other organizationally are more likely to communicate with each other [Cal89].

A document available online [ALE21] contains, among other things, a large list of observed callsigns in a number of unencrypted ALE networks, mostly belonging to government agencies in the United States and Europe. While the data is of uncertain origin, it is the only source available that provides any insight in how callsigns are assigned in ALE networks. We sorted all valid three letter callsigns from the list by network name and enumerated all possible pairs of three-character callsigns for each network. We then calculated how many possible pairs in each network that differed only in the least significant

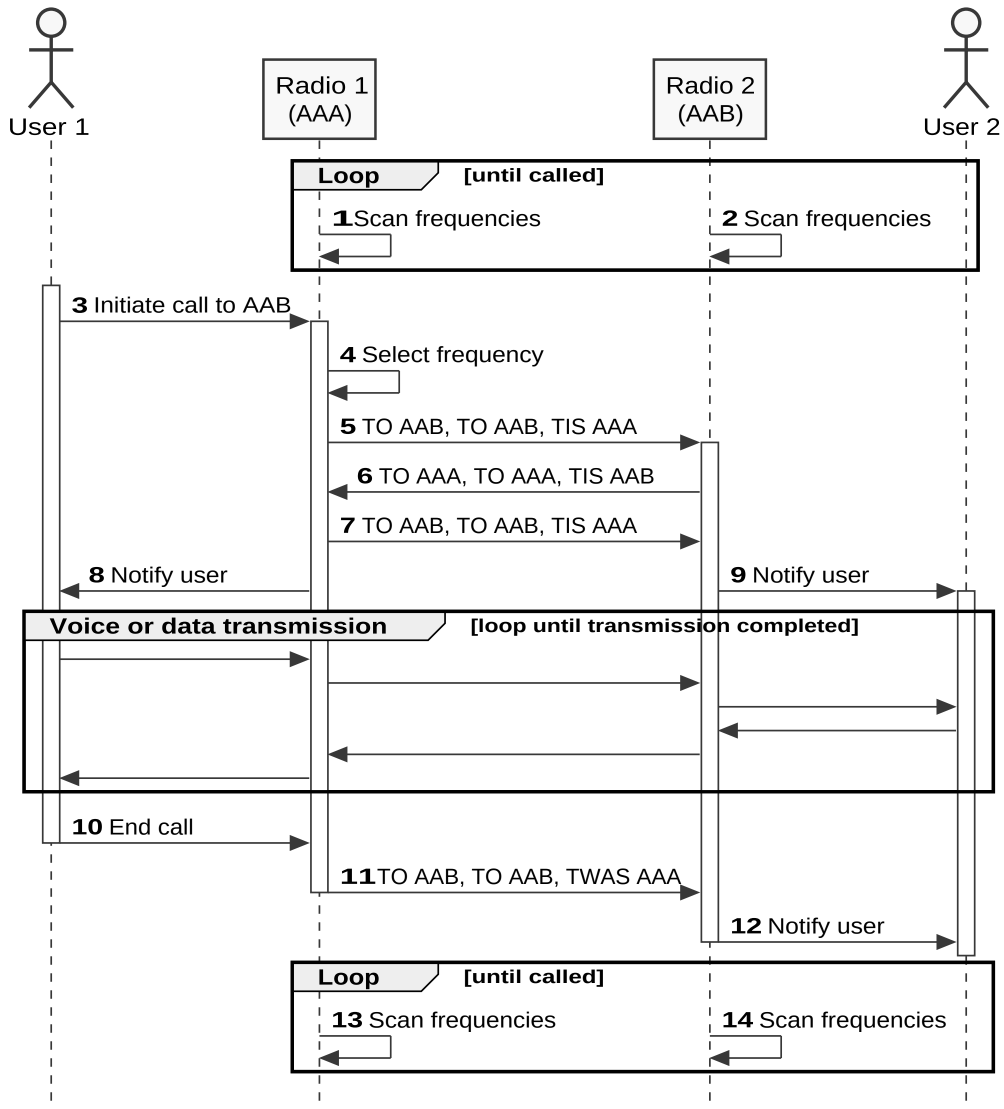


Figure 7: Sequence diagram of a typical 2G ALE call between two radio stations with callsigns AAA and AAB. The users can be either humans or computers, depending on the application.

Table 3: Possible call sign pairs in networks with more than 20 observed three-letter callsigns. The data is adapted from the list provided in [ALE21]. It is of uncertain origin and the names assigned to the networks may not reflect their actual operators. (LSB = Least Significant Byte)

Network name	Observed callsigns in network	Observed 3-character callsigns	Fraction of pairs with difference only in LSB
USCG	269	262	2.5 %
FBI	82	68	0.9 %
CBP	72	68	3.3 %
Sweden	64	64	21.5 %
Romania	56	54	1.7 %
US 3-Letter Net	46	45	0.6 %
USAF	73	38	1.1 %
COTHEN	46	31	0.2 %
Saudi Arabia	29	28	9.0 %
UK	22	22	10.8 %
Venezuelan Navy	52	21	7.6 %
US Army National Guard	50	21	0.5 %

byte. The results for the networks in the list with more than 20 observed callsigns are presented in Table 3. All but one of the networks have significantly more possible pairs of callsigns that differ in only the least significant byte than what would be expected if the callsigns were randomly assigned using all 36 available characters.

Combining the assumption of 96 messages per 16-minute window and with the highest observed fraction of plaintext pairs that differ only in the LSB from Table 3 results in a probability of

$$1 - \left(1 - 2^{-22.4} \cdot 0.215 \cdot 2^{-8}\right)^{\left\lfloor \frac{96^2}{8} - \frac{96}{2} \right\rfloor} = 2^{-22.5}$$

that a given 16-minute window contains at least one ALE word with good plaintext, ciphertext and tweak differences. This can be considered the best-case scenario for an attacker only capable of recording transmitted frames and their corresponding plaintexts. Since three good pairs are needed to mount the attack, an attacker would have to collect frames for

$$\frac{3 \cdot 2^{22.5}}{90 \cdot 365.25} \approx 541 \text{ years}$$

to have a 50% probability of success. (Two pairs can be leveraged for a complexity 2^{80} attack. This would lower the expected time by one third.)

In other words, although getting pairs of messages with a good tweak difference is fairly easy, the low probability of getting the same difference in ciphertext and plaintext, even under favorable conditions, makes the attack unlikely to work in practice. Fig. 8 shows the probability of success as a function of time for three different traffic intensities, representing high to very high network usage. In the high-traffic scenarios, the probability of success after a year may still be higher than what is acceptable for some users. In the worst case presented, it is 1.5% after a year.

4.4 A Ciphertext-only Attack

By leveraging knowledge of the structure of the plaintexts, the attack described in Section 3 can be adapted into a ciphertext-only attack that is performed as follows.

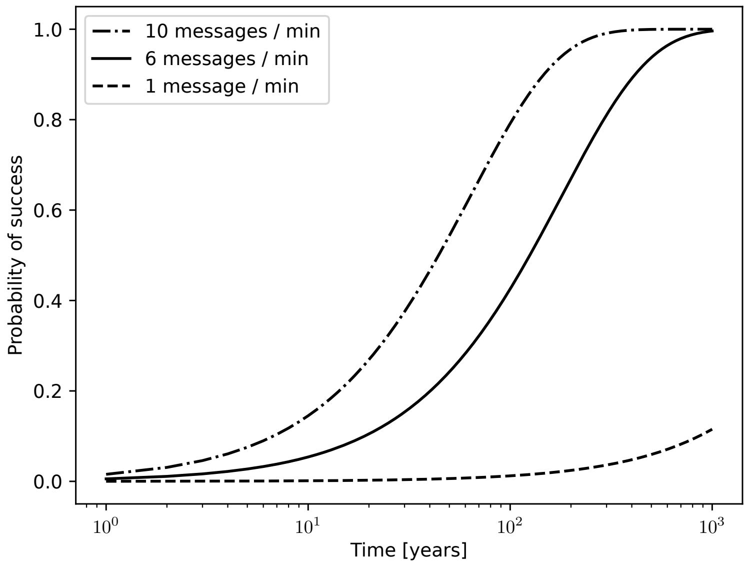


Figure 8: Probability of finding three pairs with good differences in plaintexts, ciphertexts and tweaks as a function of time in a best-case scenario.

1. **Gather Data:** In each 16-minute window, we collect all observed ciphertexts and store all pairs that have the required ciphertext and tweak differences. We do this until we have a sufficiently high probability of having collected four pairs with good plaintexts. The probability p_{PT} that a stored pair will have the required plaintext difference varies between networks as described in Section 4. The probability of having found at least four good pairs after n collected ciphertexts is

$$1 - \sum_{k=0}^{3} \binom{n}{k} (1-p)^{n-k} p^k,$$

   where $p = 2^{-8} \cdot p_{PT}$ is the probability that the plaintexts have the correct difference (see Section 4.3).

2. **Enumerate candidate keys:** For each possible combination of three collected pairs, we build the left and right tables as described in Section 3.3 and perform the first enumeration as described Section 3.4. We store all candidate keys in a sorted list. For n ciphertext pairs with the required ciphertext and tweak differences, this means performing the enumeration step for $\binom{n}{3}$ choices of three ciphertext pairs. Each search for candidate keys has a time complexity proportional to 2^{32} and will yield approximately 2^8 random 80-bit candidate keys, meaning that approximately $m = \binom{n}{3} \cdot 2^8$ candidate keys will be generated in total. The correct candidate appears at least four times since there are four combinations of three good pairs whereas the number of wrong key candidates appearing four times is roughly $m^4 \cdot 2^{-240}$, assuming they are distributed randomly. Thus, it is easy to identify the correct key candidate. With a time complexity of $\binom{n}{3} \cdot 2^{32}$, this step dominates the overall time complexity of the attack.

3. **Full key search:** We run the full key search as described in Section 3.5. Instead of known plaintexts, we verify that the computed plaintexts have the correct type

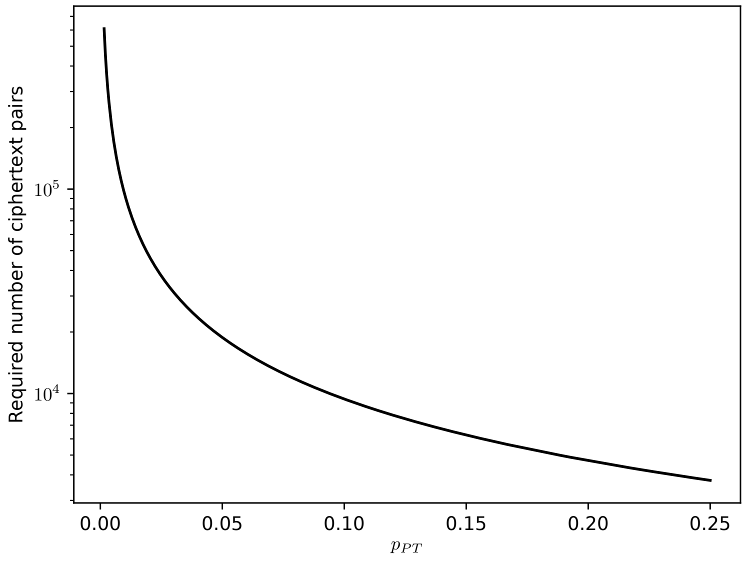


Figure 9: Required number of good ciphertext pairs for 50% probability of success in the ciphertext-only attack as a function of the probability p_{PT} that two plaintexts differ only in the least significant eight bits.

Table 4: Time, data and memory complexities of the ciphertext-only attack for some values of p_{PT}. The data complexity values are dependent on traffic intensity. Here, 6 messages per minute are assumed.

p_{PT}	n	Time	Data	Memory
$2^{-9.3}$	$609151 \approx 2^{19.2}$	$2^{87.1}$	$2^{38.2}$	$2^{63.1}$
0.02	$47003 \approx 2^{15.5}$	$2^{76.0}$	$2^{34.5}$	$2^{52.0}$
0.215	$4372 \approx 2^{12.1}$	$2^{65.7}$	$2^{31.1}$	$2^{41.7}$

bits, contain only allowed callsign characters, that words one and two of each frame are identical, and that the plaintext pairs have the expected differences. The time complexity of this step is 2^{48}.

Fig. 9 shows the number of required good ciphertext pairs that are required for a 50% probability of success as a function of p_{PT}. We list the time, data and memory complexity for some values of p_{PT} in Table 4.

A Note on 3G ALE In addition to its use in 2G ALE, HALFLOOP-24 is also used for encrypting 26-bit robust link setup (RLSU) frames, called protocol data units (PDU), in 3G ALE. This is achieved by transmitting the two most significant bits unencrypted and encrypting only the least significant 24 bits. There are five types of RLSU PDUs. All have linear checksums in the least significant bits. In three cases, the checksums are eight bits long, meaning that plaintexts that differ in only the least significant byte are impossible. The other two, the call PDU and notification PDU, have four bit checksums. The notification PDU is, among other things, used for regular sounding calls [JKF+12]. In those cases, ten of the 24 bits will be known with certainty to an outside observer. Since four of the remaining bits are a checksum, the transmitter's ten-bit identity is the only uncertain part of an intercepted 3G ALE RLSU sounding. Three of those bits are in the least significant byte, meaning that a pair of intercepted such PDUs will have probability 2^{-7} of differing only in that byte, if the identities are distributed uniformly random. This could be leveraged for ciphertext-only attacks against 3G ALE RLSU networks with complexities similar to those for 2G ALE.

5 A boomerang Attack

In this section, we describe a boomerang attack, i.e. a chosen-ciphertext attack, against HALFLOOP-24. This requires an encryption and also decryption oracle which for example could be achieved by gaining physical (but only temporary) access to a radio that stores the desired key.

Regard Fig. 10 where we restore the first key byte. To do so, we utilize a boomerang that returns with probability one if the plaintext difference δ cancels the tweak-induced difference β in the first round key. More specifically, consider two plaintexts p_0, p_1 with $p_0 \oplus p_1 = \delta || 0^{16}$ and two corresponding tweaks t_0, t_1 with $t_0 \oplus t_1 = 0^{24} || \tilde{\beta} || 0^{16}$ where $\tilde{\beta}$ is such that $L^{-1}\tilde{\beta} = \beta || 0^{16}$. For convenience, we assume $t_0 = 0$ in the following, so that we can omit the normalization. We swap the first linear layer with the addition of rk_1. Then, the only state difference after the addition of $L^{-1}rk_1$ is in the most significant byte and its value is

$$S(p_0[0{:}8] \oplus rk_0[0{:}8]) \oplus S(p_0[0{:}8] \oplus rk_0[0{:}8] \oplus \delta) \oplus \beta. \tag{2}$$

Detecting that this difference is zero for specific values of δ and β enables us to restore rk_0. Now recall Fig. 2 and notice that there is no round key difference until rk_6. Hence, if Eq. (2) is zero, then the state difference before adding rk_6 is zero too. After adding rk_6, the difference of the middle state s, is therefore $0 || \tilde{\beta}[8{:}16]$. Now consider the resulting ciphertexts c_0 and c_1. For the backward part of our boomerang, we do not change these, i.e. we have $c_0' = c_0$ and $c_1' = c_1$. The tweaks are changed as follows:

$$t_0' = t_0 \oplus 0^{16} || \gamma || 0^{24} || \gamma || 0^8$$
$$t_1' = t_1 \oplus 0^{16} || \gamma || 0^{24} || \gamma || 0^8$$

where $\gamma \in \mathbb{F}_2^n \setminus \{0\}$ is arbitrary. Now, we decrypt (c_0', t_0') and (c_1', t_1'). The difference of t_0 and t_0' is chosen such that it does not influence the last four round keys and since $c_0 = c_0'$, we obtain the same middle state $s_0' = s_0$. The same holds for $s_1' = s_1$. Hence, the difference $s_0' \oplus s_1'$ is again $0 || \tilde{\beta}[8{:}16]$ and therefore canceled by the addition of rk_6 which has the same difference as before as $t_0' \oplus t_1' = t_0 \oplus t_1$. There is no difference for the round keys rk_5, rk_4, rk_3, rk_2 and so we end up with the difference β induced by the addition of $L^{-1}rk_1$ in the most significant byte again. Finally, we obtain two plaintexts p_0' and p_1' with $p_0' \oplus p_1' = \delta' || 0^{16}$ such that Eq. (2) is also fulfilled if we replace δ with δ'.

Summing up, we need at most 2^8 encryption and decryption queries to find the first key byte. In the same way, we can find the second and third key byte. Notice, for the third key byte we have to add γ to the input of both SBoxes in Eq. (2) for the back direction. This of course already breaks the security of 128 bits since only 104 unknown key bits remain, but it is also not hard to see that one can recover these more efficiently than brute force. We leave the details to future work.

Experimental Verification We implemented our boomerang attack and tested it in a lab setting. The results are depicted in Appendix A. Executing our attack takes less than a second on a modern laptop. In other words, the time needed to run the attack essentially only depends on the rate of encryption and decryption queries. Therefore, we only average the number of needed queries. For 1000 runs, we obtain an average number

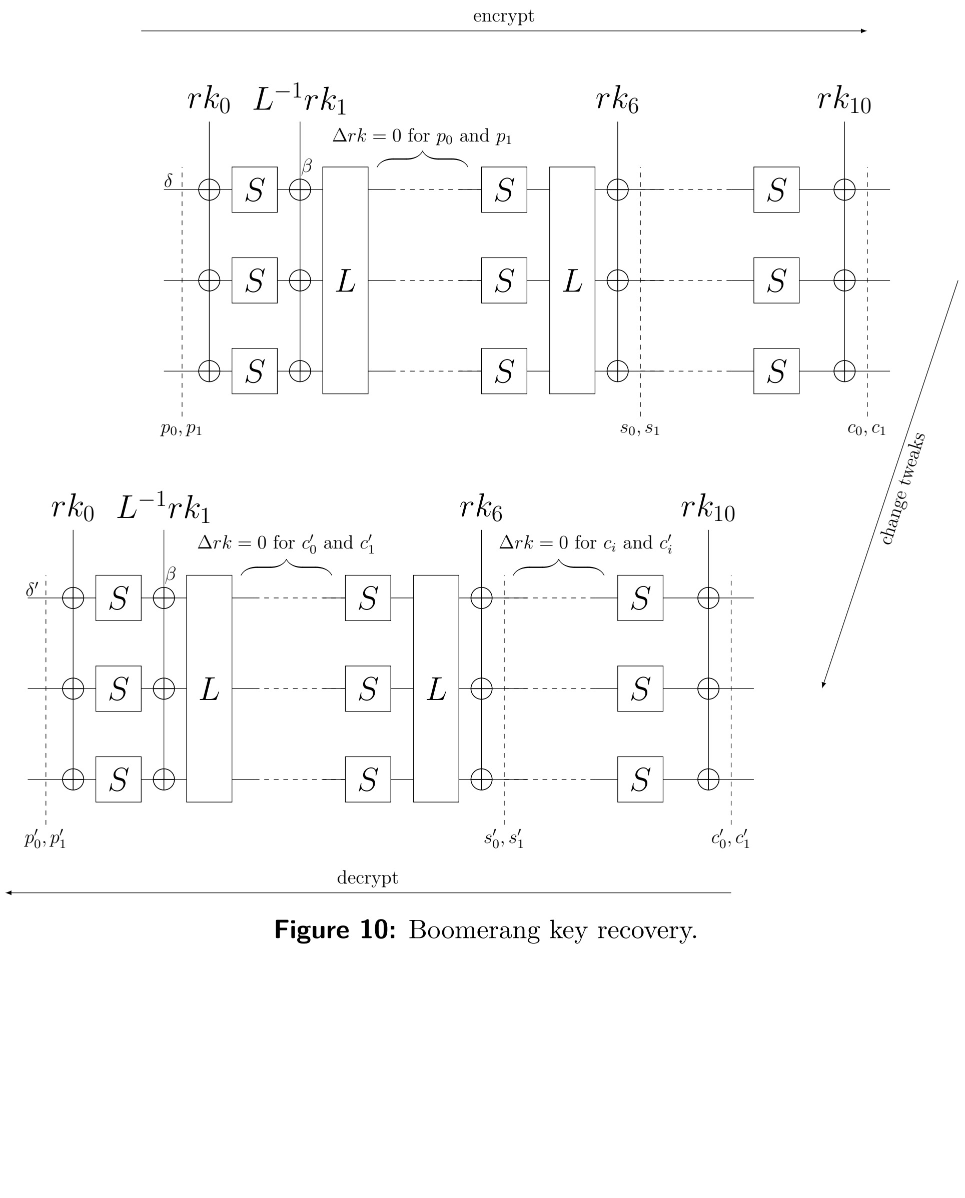


Figure 10: Boomerang key recovery.

of 385 encryption plus 385 decryption queries to restore the first round key which is in line with our theoretical estimation of a maximum of $3 \cdot 2^8 = 768$ needed queries per round key.

6 Conclusion

We have presented theoretical and practical attacks against the HALFLOOP-24 cipher which is used to protect the automatic link establishment in HF radio. We revealed that HALFLOOP-24 is far from providing 128 bits of security, although it is not known whether that is indeed the intended security level. In fact, the design requirements for HALFLOOP that have been published [Joh16] mostly focus on the requirement that the encryption should not decrease ALE performance, along with a note that the US government has considered the use of AES-128 in approved implementations sufficient to protect classified information. In any case, it is apparent that HALFLOOP-24 was not subjected to a thorough security analysis before its introduction. This, as well as the fact that its predecessor SoDark is only a 56-bit cipher, leaves many open questions about the design goals, decisions and the process of the authorities in charge.

Surely, one could try to mitigate the presented weakness, e.g. by increasing the number of rounds as was done when the number of rounds in SoDark was increased from eight in 2G ALE to sixteen in 3G ALE. Alternatively, one could avoid using the 2G ALE or 3G ALE RLSU modes which use HALFLOOP-24, instead using only modes that use HALFLOOP-48 or HALFLOOP-96. However, the security of the latter has not yet been studied and is therefore unknown. We leave this as future work. In the meantime, we advise against the use of any HALFLOOP variant. Instead, security should rely on ciphers that have undergone rigorous public security analysis.

Acknowledgments

We would like to thank Schloss Dagstuhl, as this work was discussed at Dagstuhl-Seminar 22141 (symmetric cryptography).

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.

References

[ALE21] List of automatic link establishment agencies and frequencies, 2021. http://www.ominous-valve.com/ale-list.txt.

[BBS99] Eli Biham, Alex Biryukov, and Adi Shamir. Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 12–23. Springer, 1999.

[Bih97] Eli Biham. A fast new DES implementation in software. In Eli Biham, editor, *FSE'97*, volume 1267 of *LNCS*, pages 260–272. Springer, Heidelberg, January 1997.

[BJK$^+$16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016*

- *36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

[BK09] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In Mitsuru Matsui, editor, *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[Cal89] Lambros D. Callimahos. *Traffic Analysis and the Zendian Problem : An Exercise in Communications Intelligence Operations.* Aegean Park Press, Laguna Hills, CA, 1989.

[Dan21] Marcus Dansarie. Cryptanalysis of the SoDark cipher for HF radio automatic link establishment. *IACR Trans. Symmetric Cryptol.*, 2021(3):36–53, 2021.

[DoD88] Interoperability and performance standards for medium and high frequency radio systems. United States Department of Defense Interface Standard MIL-STD-188-141A, 1988.

[DoD17] Interoperability and performance standards for medium and high frequency radio systems. United States Department of Defense Interface Standard MIL-STD-188-141D, 2017.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Information Security and Cryptography. Springer, 2002.

[JKF$^+$12] Eric E Johnson, Erik Koski, William N Furman, Mark Jorgenson, and John Nieto. *Third-generation and Wideband HF Radio Communications.* Artech House, Norwood, MA, 2012.

[Joh16] Eric E. Johnson. Wideband ALE – the next generation of HF. In *2016 Nordic HF Conference, HF 16*, Fårö, Sweden, 2016.

[Wag99] David A. Wagner. The boomerang attack. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

A Example Output of Our Attack Implementations

Here, we give example outputs for our attacks described in Sections 3 and 5.

Listing 1: Generation of four chosen plaintext–ciphertext–tweak pairs as described in Section 3.2.

```
Key: 71915d837a45a05689e33d745c2b7b38
4 pairs generated in 0.2 seconds.
Number of chosen plaintext queries: 561152
```

Listing 2: Output of one run of the attack described in Section 3.

```
[06:02:49] Initializing HALFLOOP-24 library.
[06:02:49] Loading tuples from data.txt.
[06:02:49] Loaded 8 tuples.
[06:02:49] Found 4 good pairs.
[06:02:49] Searching for 80-bit candidate keys.
[06:02:51] Left table size: 300
[06:08:52] Found 323 candidate keys.
[06:08:53] Left table size: 266
[06:14:12] Found 282 candidate keys.
[06:14:12] 1 candidate key remaining.
[06:14:12] Time spent building left tables:
           0 minutes and 2.8 seconds.
[06:14:12] Time spent building right tables:
           0 minutes and 0.0 seconds.
[06:14:12] Time spent enumerating candidate keys:
           11 minutes and 20.1 seconds.
[06:14:12] Searching for remaining 48 bits for key
           81 0f 7216c7eb2e3dbcd3 (1/1).
[06:14:12] Spawning 16 threads.
[06:20:25] 1% done 7558585494 keys/second.
[06:26:35] 2% done 7607707818 keys/second.
[06:32:49] 3% done 7512880168 keys/second.
[06:38:59] 4% done 7604368012 keys/second.
[06:45:13] 5% done 7524366063 keys/second.
[06:51:26] 6% done 7560786600 keys/second.
[06:57:37] 7% done 7587583319 keys/second.
[07:03:51] 8% done 7511394037 keys/second.
[07:10:08] 9% done 7484862010 keys/second.
[07:16:18] 10% done 7602523998 keys/second.
[07:22:27] 11% done 7616353731 keys/second.
[07:28:40] 12% done 7560130350 keys/second.
[07:34:50] 13% done 7608953536 keys/second.
[07:40:59] 14% done 7608874360 keys/second.
[07:47:15] 15% done 7502177912 keys/second.
[07:53:25] 16% done 7607708047 keys/second.
[07:59:08] Found key: 71915d837a45a05689e33d745c2b7b38
[07:59:19] Time spent searching for key:
           1 hour, 45 minutes and 6.5 seconds
```

Listing 3: Output of one run of the boomerang attack described in Section 5.

```
[10:28:11] Initializing HALFLOOP-24 library.
[10:28:11] Key:        89681b1a281e9bd07851f91f7578dd5e
[10:28:11] Seed:       3cbae7385a8b9626
[10:28:11] Plaintext:  dcb26a
[10:28:11] Ciphertext: cc6930
[10:28:11] Beta:       c0
[10:28:11] Gamma:      ef
[10:28:11] Key byte 0: 89 (d = 32 d' = db).
[10:28:11] Key byte 1: 68 (d = 1c d' = d8).
[10:28:11] Key byte 2: 1b (d = fa d' = 2c).
[10:28:11] Performed 329 encryptions and 329 decryptions.
```