IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2023, No. 4, pp. 58–82. DOI:10.46586/tosc.v2023.i4.58-82

Cryptanalysis of HALFLOOP Block Ciphers

Destroying HALFLOOP-24

Gregor Leander[1], Shahram Rasoolzadeh[2] and Lukas Stennes[1]

[1] Ruhr University Bochum, Bochum, Germany
firstname.lastname@rub.de
[2] Radboud University, Nijmegen, The Netherlands
firstname.lastname@ru.nl

Abstract. HALFLOOP is a family of tweakable block ciphers that are used for encrypting automatic link establishment (ALE) messages in high frequency radio, a technology commonly used by the military, other government agencies and industries which require high robustness in long-distance communications. Recently, it was shown in [DDLS22] that the smallest version of the cipher, HALFLOOP-24, can be attacked within a practical time and memory complexity. However, in the real-word ALE setting, it turns out that this attack require to wait more than 500 years to collect the necessary amount of plaintext-tweak-ciphertext pairs fulfilling the conditions of the attack.

In this paper, we present real-world practical attacks against HALFLOOP-24 which are based on a probability-one differential distinguisher. In our attacks, we significantly reduce the data complexity to three differential pairs in the chosen-plaintext (CPA) setting which is optimal in the sense that even a brute force attack needs at least six plaintext-tweak-ciphertext pairs to uniquely identify the correct key. Considering the same ALE setting as [DDLS22], this translates to a reduction from 541 years to 2 hours worth of intercepted traffic.

Besides, we provide the first, non generic, public cryptanalysis of HALFLOOP-48 and HALFLOOP-96. More precisely, we present Demirci-Selçuk meet-in-the-middle attacks against full-round HALFLOOP-48 and round-reduced HALFLOOP-96 to recover the complete master key in a CPA setting. However, unlike the attacks on HALFLOOP-24, our attacks on the larger versions are only theoretical. Moreover for HALFLOOP-96 the known generic time-memory trade-off attack, based on a flawed tweak handling, remains the strongest attack vector.

In conclusion, we iterate what was already stated in [DDLS22]: HALFLOOP does not provide adequate protection and should not be used.

Keywords: HF Radio · ALE · HALFLOOP · Differential · DS-MITM · TDM-TO

1 Introduction

High frequency (HF) radio communication enables radio communication even in case there is no direct line-of-sight by using so-called skywave propagation, i.e., the fact that the signal is reflected back to earth by the ionosphere layer in the atmosphere. As such, HF radio communication can be used for intercontinental radio communications or in cases where the topography prevents direct line-of-sight communication and a redirecting network is not available. As a downside, HF radio has a rather small bandwidth.

In order to establish a connection between two communication parties, protocols for automatic link establishment (ALE) were developed during the 1980s, with military settings being one of the primary setups. The US military standard [DoD17] describes how to use dedicated tweakable block ciphers, HALFLOOP and its predecessor SoDark,

Received: 2023-09-01 Accepted: 2023-11-01 Published: 2023-12-08

in order to ensure confidentiality and authentication in the ALE protocol. We refer to [Dan21, DDLS22] for more background on ALE and its practical relevance.

The cipher SoDark was developed already in 1992 [Joh92]. However, it took nearly 30 years until Dansarie [Dan21], who showed weaknesses in SoDark, attracted the attention of the symmetric crypto community to the encryption algorithms used in ALE. By then, the successor algorithm HALFLOOP was already specified in [DoD17]. Its building blocks mostly imitate the operations of the AES. Dansarie, Derbez, Leander, and Stennes [DDLS22] presented a practical time attack on HALFLOOP-24 using a data complexity of 2^{38} known plaintexts. While this badly breaks the cipher in an academic setting, it turns out that this data complexity is actually impractical: As detailed in [DDLS22] even assuming a favorable situation with many ALE messages sent per day, it would require to wait about 541 years to collect plaintext-tweak-ciphertext pairs fulfilling the requirements of the attack.

Our Contribution Our main contribution are practical attacks against HALFLOOP-24. Besides that, we give the first public cryptanalysis of HALFLOOP-48 and HALFLOOP-96.

Our attack on HALFLOOP-24 is a rather classical differential attack with the important caveat that, unlike classical differential attacks, we focus on reducing the data complexity down to very few pairs as our primary goal. We achieve that by a rather heavy trial decryption phase. This in turn requires a careful fine-tuning of the filtering process for key candidates in order to keep the entire attack practical.

We significantly reduce the data complexity from 2^{18} to three differential pairs in the chosen-plaintext (CPA) setting. This is optimal in the sense that even a brute force attack needs at least six plaintext-tweak-ciphertext pairs to uniquely identify the correct key. When we consider the same ALE setting as [DDLS22], i.e., the same number of sent messages per minute etc., this translates to a reduction from 541 years to 2 hours worth of intercepted traffic. Hence, we consider our attack on HALFLOOP-24 *practical*.

We would like to highlight that we implemented all details of the attack in order to (i) verify the heuristic assumptions made and (ii) get accurate estimates of the running time on actual hardware.

Concerning the two larger variants of HALFLOOP, the only known attacks, mentioned in [DDLS22], are time-data-memory trade-off (TDM-TO) attacks based on the flawed handling of tweaks. For completeness, we present the details of this attack here. For instance, for HALFLOOP-96, this attack requires 2^{64} encryptions for each offline and online computations, 2^{64} chosen-plaintext-tweak-ciphertext pairs of data, and $3 \cdot 2^{69}$ bytes of memory.

Those attacks are generic in the sense that they do not depend on any internal structure of the round function or the key-scheduling itself. As such, the question remains if stronger attacks exist. Our second contribution is thus to initiate the non-generic cryptanalysis on the larger versions of HALFLOOP by providing the first non-generic cryptanalysis of these two ciphers.

More precisely, we apply Demirci-Selçuk Meet-in-the-Middle (DS-MITM) attack [DS08] against full-round HALFLOOP-48 and 7-round-reduced HALFLOOP-96 to recover the complete master key in a CPA setting. The complexity of the first attack is 13 chosen-plaintext-tweak-ciphertext pairs together with about 2^{121} encryptions and 2^{122} look-ups in a table which needs 2^{97} bytes of memory. Note that compared to the TDM-TO attack on full-round HALFLOOP-48, the time complexity of the DS-MITM attack is comparably higher, however, its data complexity is comparably lower which can be seen as a trade-off between time and data complexity.

The latter attack needs 15 chosen-plaintext-tweak-ciphertext pairs together with about $2^{113.3}$ encryptions and 2^{114} look-ups in a table with a size of 2^{105} bytes. Thus, for HALFLOOP-96 the generic attack remains the best attack vector for now.

We summarize our results together with the attacks from [DDLS22] in Table 1. For

Table 1: Overview of the attacks on HALFLOOP family of tweakable block ciphers. †: The CCA attack in [DDLS22] restores only the first round key.

Variant	Setting	Time	Data	Memory	Reference
HALFLOOP-24	ALE	2^{56} Enc.	**541 years**	2 MB	[DDLS22]
HALFLOOP-24	ALE	2^{48} Enc.	**2 hours**	5 GB	Section 3
HALFLOOP-24	CCA	2^{10} Enc.†	2^{10}	negligible	[DDLS22]
HALFLOOP-24	CPA	2^{56} Enc.	6	5 GB	Section 3
HALFLOOP-24	CPA	2^{48} Enc.+2^{48} LUT	8	5 GB	Section 3
HALFLOOP-48	CPA	2^{121} Enc.+2^{122} LUT	13	2^{57} TB	Section 5
HALFLOOP-48	CPA	2^{65} Enc.+2^{64} LUT	2^{65}	$3 \cdot 2^{29}$ TB	Section 4
HALFLOOP-96	CPA	2^{64} Enc.+2^{64} LUT	2^{64}	$3 \cdot 2^{29}$ TB	Section 4

completeness, we also include the boomerang attack from [DDLS22] which is in the chosen-ciphertext (CCA) setting where an attacker can choose tweaks and query encryptions and decryptions as they wish. But, in the real world ALE setting, this is rather unrealistic as the attacker needs full access to a radio with the key.

Structure of the Paper We clarify our notation and give a brief description of HALFLOOP in Section 2. In Section 3, we give our new and *practical* attack against full-round HALFLOOP-24. The theoretical TDM-TO attacks on all versions of HALFLOOP are described for completeness in Section 4. We present DS-MITM attacks on HALFLOOP-48 and 7-round-reduced HALFLOOP-96 in Section 5. We conclude our work in Section 6.

2 Preliminaries

We write $\mathbb{F}_2$ for the finite field with two elements $\{0,1\}$, $\mathbb{F}_{2^n}$ for the finite field with 2^n elements, and $\mathbb{F}_2^n$ for the vector space of dimension $n \in \mathbb{N}$ over $\mathbb{F}_2$. We use $\oplus$ to denote the component-wise addition in $\mathbb{F}_2^n$ which we also call XOR. For a set of vectors $A \subseteq \mathbb{F}_2^n$ and a vector $b \in \mathbb{F}_2^n$, we simply write $A \oplus b$ for the set $\{a \oplus b \mid a \in A\}$. For two vectors $x, x' \in \mathbb{F}_2^n$, we write Δx for the *difference* $\Delta x = x \oplus x'$. The concatenation of two vectors a and b is denoted as $a \,\|\, b$.

We refer to a tuple of plaintext, tweak, and ciphertext (p,t,c) as *plaintext-tweak-ciphertext pair* and call a pair of such pairs $((p,t,c),(p',t',c'))$, which is usually related in a differential way, a *differential pair*.

Recall the basic differential properties of the AES S-box $S\colon \mathbb{F}_2^8 \to \mathbb{F}_2^8$. For a non-zero input differences $\delta \in \mathbb{F}_2^8$ there are 127 possible output differences γ. For 126 of those, the equation $S(x) \oplus S(x \oplus \delta) = \gamma$ has two solutions. For each δ, there is one and only one γ such that the equation has four solutions. We denote a possible transition of δ to γ through S by $\delta \xrightarrow{S} \gamma$. When we apply S in parallel to a vector $x \in \mathbb{F}_2^{k \cdot 8}$, we write $\mathtt{SB}(x)$.

2.1 Description of HALFLOOP and Related Notation

The family of tweakable block ciphers HALFLOOP, i.e., the three variants HALFLOOP-24, HALFLOOP-48, and HALFLOOP-96 are specified in US military standard 188-141 [DoD17]. For the sake of self-completeness, we give a brief description of HALFLOOP.

For each variant, the trailing number is the block size in bits. All variants use a 128-bit key k, a 64-bit tweak t and consist of 10 rounds. Each round is built up of four steps: AddRoundKey, SubBytes, RotateRows and MixColumns. These components are very

similar to the building blocks of the AES and are explained in detail below. In the last round there is no MixColumns operation, and instead there is an extra round key addition.

State and AddRoundKey Similar to the AES, the state of HALFLOOP is represented as a matrix of 8-bit cells. For HALFLOOP-24, the 24-bit state is represented as a 3×1 matrix. For HALFLOOP-48, we use a 3×2 and for HALFLOOP-96 a 4×3 matrix respectively. For each variant, the round keys are represented in the same way as the state and the key addition is simply the bitwise XOR of state and key.

For convenience, throughout the paper, we denote the state immediately after the addition of the round key with the letter x, the state after the S-boxes by y and the state after the linear layer by z. Furthermore, we use the superscript to illustrate the round, e.g., $x^{(0)}$ denotes the XOR of the plaintext p and the very first round key $rk^{(0)}$ and $x^{(10)}$ corresponds to the ciphertext. At times, we are only interested in some bytes of a state or round key. Then, we use the subscript to denote the corresponding cell, i.e., byte, in the standard way depicted below, where (a state of) HALFLOOP-24 is depicted on the left, HALFLOOP-48 in the middle and HALFLOOP-96 on the right.

x_0
x_1
x_2

$x_{0,0}$	$x_{0,1}$
$x_{1,0}$	$x_{1,1}$
$x_{2,0}$	$x_{2,1}$

$x_{0,0}$	$x_{0,1}$	$x_{0,2}$
$x_{1,0}$	$x_{1,1}$	$x_{1,2}$
$x_{2,0}$	$x_{2,1}$	$x_{2,2}$
$x_{3,0}$	$x_{3,1}$	$x_{3,2}$

For convenience of the reader, we always use the variable names i and j to address different rows and columns of the cells in a state and r to address the round number. For denoting states in the data path and states of round keys, we show the two-dimensional array of a state as a vector of bytes by concatenating the columns. Thus, we denote the state x of HALFLOOP-24 with (x_0, x_1, x_2), and for HALFLOOP-48 and HALFLOOP-96 with $(x_{0,0}, x_{1,0}, x_{2,0}, x_{0,1}, x_{1,1}, x_{2,1})$ and $(x_{0,0}, x_{1,0}, \dots x_{2,2}, x_{3,2})$, respectively. Besides, for denoting the tweak state and two halves of the key, we always use an array of 16 bytes.

SubBytes We apply the 8-bit AES S-box S to each cell of the state.

MixColumns For the MixColumns step, each column is treated as a polynomial over $\mathbb{F}_{2^8}$. For HALFLOOP-96, we then multiply this polynomial with the fixed polynomial $c(\chi) = 3\chi^3 + \chi^2 + \chi + 2$ and reduce the result modulo $\chi^4 + 1$ which is the same as for AES. For HALFLOOP-24 and HALFLOOP-48, we multiply with $c(\chi) = \chi^2 + 2\chi + 9$ and reduce modulo $\chi^3 + 1$. For all three variants, these linear mappings are MDS.

RotateRows The RotateRows operation, which corresponds to ShiftRows in the AES, is essentially the same for all three variants. In contrast to ShiftRows, the rotation does not take place on the cell-level but on the bit-level. The first row of the state is not changed. The second row is rotated to the left by 6 bit positions and the third by 12 bit positions. For HALFLOOP-96, the fourth row is rotated to the left by 18 bit positions.

The linear layer of the round functions (excluding the last round) is the composition of the RotateRows and MixColumns operations. For the description of our attacks, we call the resulting linear layer LL.

Key Schedule First, the 64-bit tweak t is added to the first 64-bit of the key k. For ease of use, we split the 128-bit key k into two parts: k' the first half and k'' the second half, i.e., $k = (k' \,\|\, k'')$ and the tweak is XORed to k'. Then, the same Feistel network as in the AES-128 key schedule is used to derive the round keys. That is, we start with $(k' \oplus t \,\|\, k'')$ for the first 128 bits of the round keys. Then, we apply one round of the AES-128 key schedule to derive the next 128 bits and so forth. However, since the round keys are only of size 24, 48 or 96 bits, we do *not* need the full AES-128 key expansion. More concretely, we only need 264, 528 or 1056 round key bits. For HALFLOOP-24, this means that the tweak has only linear influence on the round keys except for the very last byte, which was already observed in [DDLS22].

3 Attack on HALFLOOP-24

Our new attack on HALFLOOP-24 builds up on the ideas of [DDLS22] and hence, before we present our new attack, we briefly recall the main concepts of their attack.

Recap of [DDLS22] Attack In [DDLS22], Dansarie, Derbez, Leander, and Stennes present a *nearly* practical known-plaintext attack against HALFLOOP-24. Their attack builds upon two observations. First, since the tweak is simply XORed to the master key and there are only 10 rounds and each round key is only 24 bits, related-tweaks allow for powerful attacks, similar to the related-key attacks on AES-192 and AES-256 [BK09]. More concretely, they show that for a differential pair of plaintexts, tweaks and ciphertexts (p, t, c) and (p', t', c') and a difference $\delta \in \mathbb{F}_2^8$ with

$$p \oplus p' = (0, 0, \delta)\,, \qquad t \oplus t' = (0, 0, \delta, 0, 0, 0, 0, 0)\,,$$

it holds that there is no difference in $z^{(5)}$, i.e., the state after five rounds. Namely, we have

$$\Delta rk^{(r)} = \begin{cases} (0, 0, \delta) & \text{if } r \in \{0, 8\} \\ (\delta, 0, 0) & \text{if } r \in \{6, 10\} \\ (0, \delta, 0) & \text{if } r = 7 \\ (0, 0, 0) & \text{else.} \end{cases}$$

This is visualized in Figure 1. The second observation is that we have $\Delta x^{(8)} = 0$ if and only if we observe $c \oplus c' = (\delta, 0, 0)$ which is not too unlikely, since the ciphertexts are only 24 bits in size. Given three such pairs, [DDLS22] shows how to recover the key in practical time. However, it turns out that the data complexity is too high in the real world. Our new attack improves on this as we do not rely on a special ciphertext difference.

Notice, to ease notation, throughout this section, we ignore what is called *round key normalization* in [DDLS22]. That is, when we consider the round keys for independent tweaks t and t', the round keys differ but the difference is almost only linear in the tweaks. This is also visible in Figure 1. No matter where the tweaks t and t' differ, the last 32-bit word of the key, corresponding to $rk^{(4)}$ and $rk_0^{(5)}$, is not affected. Hence, the first-round input to g, which is the only non-linear operation in the key schedule, is the same for every tweak. For the attacks on HALFLOOP, we consider multiple tweaks and hence it is handy to normalise the round keys by considering the round keys for the all zero tweak. Of course, we do so in our implementation of our attack. But for this write-up, we do this only implicitly. Notice that the influence of the tweak is only almost linear since it is non-linear on the last byte of $rk^{(10)}$. This is again visible in Figure 1 as the last byte of $rk^{(10)}$ depends on the output of the non-linear g function. However, this is not a problem for our new attack since this non-linearity only depends on $rk^{(9)}$ which we guess at the same time as $rk^{(10)}$.

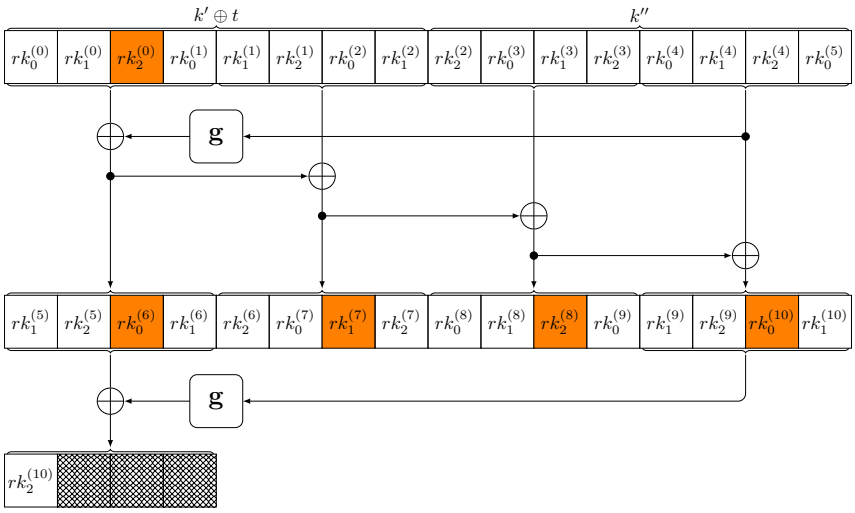


Figure 1: Key schedule of HALFLOOP-24 together with the effect of tweak difference $(0, 0, \delta, 0, 0, 0, 0, 0)$ adapted from [DDLS22]. The orange blocks depict the round key bytes that will be affected by the tweak difference. Note that the induced difference in each of these bytes are the same as δ.

3.1 Our New Attack

Naive Attack With Better Data Complexity To ease the understanding of our new attack, we present an intermediate attack first. That is, an attack that improves the data complexity but is not optimal regarding the time complexity. The attack is straightforward and uses a related-tweak differential with *probability-one*.

Consider Figure 2. We use d differential plaintext-tweak-ciphertext pairs $((p, t, c), (p', t', c'))$ with the aforementioned form of differences in the plaintext and tweak. Then, we guess 80 bits of the key, namely $rk^{(10)}, rk^{(9)}, rk^{(8)}$, and the first byte from the equivalent round key $\mathtt{LL}^{-1}(rk^{(7)})_0$. Notice that those 80 bits are sufficient to check $\Delta z^{(5)} \stackrel{?}{=} 0$, i.e., the difference before the key addition of $rk^{(6)}$. While for a correct guess, $\Delta z^{(5)}$ must be zero, for a wrong guess, we assume that the difference is randomly distributed over all 24-bit values.

We use $d = 3$ differential pairs, so, in total, we check for a $3 \cdot 24 = 72$-bit condition and hence expect 2^8 candidates for the 80 bits of the key. After this, we simply brute force the remaining $2^8 \cdot 2^{128-80} = 2^{56}$ bits which is the same as Step 4 of the attack in [DDLS22].

The time complexity is clearly dominated by the 2^{80} partial HALFLOOP-24 decryption. We want to emphasize that, in terms of data complexity, this attack is optimal in the chosen-plaintext setting. That is, even the trivial brute force attack that checks all 2^{128} possible keys needs six plaintext-ciphertext pairs, simply because a random key confirms a plaintext-tweak-ciphertext pair with probability 2^{-24} and hence confirms five plaintext-ciphertext pairs with probability 2^{-120} and so we would be left with about 256 candidates. We believe that obtaining a *differential* attack reaching *optimal* data complexity, is astonishing. But, of course, this is due to the *probability-one* differential.

Improving Our Time Complexity To improve the time complexity, we first notice that we do not have to guess the first byte of $\mathtt{LL}^{-1}(rk^{(7)})$ immediately. By guessing the last three round keys $rk^{(10)}, rk^{(9)}$ and $rk^{(8)}$, we partially decrypt the ciphertext pairs for three rounds to compute the $x^{(7)}$ state pairs. Without knowing any information about round key $rk^{(7)}$, we can compute $\Delta z^{(6)}$ and $\Delta y^{(6)}$. The current guess for the last three round

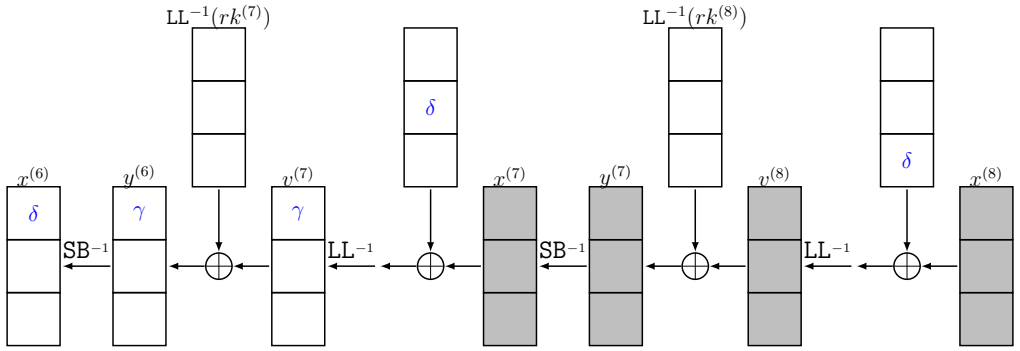


Figure 2: The main stage of our attack: seventh and eighth round of HALFLOOP-24. The corresponding difference value for each byte is written in blue, and the empty bytes are the ones with difference zero. Besides, the bytes filled with gray color have unknown difference values which will be determined by the partial decryption of the ciphertext pairs.

keys is acceptable only if the values for the second and third bytes of $\Delta y^{(6)}$ are zero.

Further, we precompute a lookup table for candidates of $rk^{(8)}$. That is, for all (observed) $\delta \in \mathbb{F}_2^8$ (from three differential pairs) and all $\Delta y^{(7)} \in \mathbb{F}_2^{24}$, we compute

$$T_\delta[\Delta y^{(7)}] = \{\mathtt{SB}(x^{(7)}) \in \mathbb{F}_2^{24} \mid \exists \gamma : \delta \xrightarrow{S} \gamma \text{ and } \mathtt{SB}(x^{(7)}) \oplus \mathtt{SB}(x^{(7)} \oplus \Delta x^{(7)}) = \Delta y^{(7)}\}$$

where $\Delta x^{(7)} = \mathtt{LL}(\gamma, 0, 0) \oplus (0, \delta, 0)$. Then, we get all the candidates for $\mathtt{LL}^{-1}(rk^{(8)})$, and equivalently for $rk^{(8)}$, by computing $T_\delta[\Delta y^{(7)}] \oplus v^{(8)}$ where $v^{(8)} = \mathtt{LL}^{-1}(x^{(8)})$ and we can compute it with only guessing $rk^{(10)}$ and $rk^{(9)}$.

As we show in the next section, on average there are about 2^7 candidates left for each differential pair. We assume that the resulting candidates are independent for each differential pair. Now, the idea is to simply intersect the candidates for each pair. That is, for each $(rk^{(10)}, rk^{(9)})$, the intersection of the three sets of candidates for $rk^{(8)}$ is roughly of size $2^{3\cdot 7 - 2\cdot 24} = 2^{-27}$, as we analyze this in more detail in next subsection.

For those set of candidates for $(rk^{(10)}, rk^{(9)}, rk^{(8)})$, we compute the set of all possible $\mathtt{LL}^{-1}(rk^{(7)})_0$, again for each pair and then compute the intersection. For each pair, we expect two candidates, on average. So, in total, we expect $2^{3\cdot 1 - 2\cdot 8} = 2^{-13}$ candidates for $\mathtt{LL}^{-1}(rk^{(7)})_0$ for every candidate of $(rk^{(10)}, rk^{(9)}, rk^{(8)})$. We add the resulting candidates to a set K which concludes the computation of all candidates for $\left(rk^{(10)}, rk^{(9)}, rk^{(8)}, \mathtt{LL}^{-1}(rk^{(7)})_0\right)$.

After we handled every $(rk^{(10)}, rk^{(9)})$, we expect $|K| \approx 2^8$ candidates for the 80 bits of the key. Therefore, there are only 56 bits to brute force and that is the final step of our attack. We capture the preceding description of our attack in Algorithm 1.

3.2 Analysis of the Attack

Here, we analyse the time, memory and data complexity of our attack. However, our intention here is not only to derive the specific complexity but also to deepen our understanding of the attack, as, in the next stage, we discuss practical improvements of our attack.

Time Complexity The time complexity of our attack clearly depends on the size $|T_\delta[\Delta y^{(7)}]|$ and the size $|K|$. As stated before, a rough estimate gives an average size of 2^7 for each $T_\delta[\Delta y^{(7)}]$. To see this, notice that for a fixed nonzero $\delta \in \mathbb{F}_2^8$, there are $127 \approx 2^7$ possible output differences γ with $\delta \xrightarrow{S} \gamma$ and those are mapped one-to-one to $\Delta x^{(7)}$. For each byte $\Delta x_j^{(7)}$, the probability that the difference goes through the single S-box to the

Algorithm 1 Attack on HALFLOOP-24

Input three differential pairs $((p,t,c),(p',t',c'))$ whose differences in the input are $\Delta p = (0,0,\delta_i)$ and $\Delta t = (0,0,\delta_i,0,0,0,0,0)$
Output the master key k used for encryption

for all $\Delta y^{(7)} \in \mathbb{F}_2^{24}$ and $\delta \in \{\delta_0,\delta_1,\delta_2\}$ **do** ▷ precomputation
  $T_\delta[\Delta y^{(7)}] \leftarrow \{\mathtt{SB}(x^{(7)}) \mid \exists \gamma : \delta \xrightarrow{S} \gamma \text{ and } \mathtt{SB}(x^{(7)}) \oplus \mathtt{SB}(x^{(7)} \oplus \Delta x^{(7)}) = \Delta y^{(7)}\}$
end for
$K \leftarrow \{\}$ ▷ list of key candidates
for all $rk^{(10)} \in \mathbb{F}_2^{24}$ **and all** $rk^{(9)} \in \mathbb{F}_2^{24}$ **do**
  compute $v^{(8)}$ and $\Delta y^{(7)}$ for each differential pair
  $RK^{(8)} \leftarrow \bigcap_{i=0}^{2} T_{\delta_i}[\Delta y^{(7)}] \oplus v^{(8)}$ ▷ 2^{48} times
  for all $\mathtt{LL}^{-1}(rk^{(8)}) \in RK^{(8)}$ **do**
    $RK^{(7)} \leftarrow$ compute candidates for $\mathtt{LL}^{-1}(rk^{(7)})_0$ ▷ $\approx 2^{21}$ times
    for all $\mathtt{LL}^{-1}(rk^{(7)})_0 \in RK^{(7)}$ **do**
      $K \leftarrow K \cup \{(rk^{(10)}, rk^{(9)}, \mathtt{LL}^{-1}(rk^{(8)}), \mathtt{LL}^{-1}(rk^{(7)})_0)\}$ ▷ $\approx 2^{8}$ times
    end for
  end for
end for
for all $(rk^{(10)}, rk^{(9)}, \mathtt{LL}^{-1}(rk^{(8)}), \mathtt{LL}^{-1}(rk^{(7)})_0) \in K$ **do**
  brute force remaining 48 bits of the key ▷ same as [DDLS22, Step 4]
  return if correct key is found
end for

(fixed) difference $\Delta y_j^{(7)}$ is $\frac{127}{256} \approx 0.5$ (ignoring that $\Delta x_1^{(7)}$ might be zero) but at the same time we expect two candidates for $y_j^{(7)}$ if it goes through. Hence, there are roughly 2^7 candidates for $rk^{(8)}$ as the XOR with $v^{(8)}$ does not change the number of key candidates.

We experimentally verified our estimates for $|T_\delta[\Delta y^{(7)}]|$. That is, we exhaustively computed $|T_\delta[\Delta y^{(7)}]|$. We give the results in Figure 3. There, we plot $\Pr[|T_\delta[\Delta y^{(7)}]| = \chi]$ with respect to χ, where the probability is taken over the uniform choice of a nonzero δ and $\Delta y^{(7)}$. The results confirm our theoretical estimates. However, there are interesting outliers, namely $\Pr[|T_\delta[\Delta y^{(7)}]| = 0] \approx 0.011$ and nonzero (but small) probabilities for 1024, 2048 or 4096 candidates for $rk^{(8)}$ (for clarity, we removed those from Figure 3). The three latter cases are due to the fact that for the middle byte there might be a zero difference which results in 256 candidates for that byte. Paired with 2 or 4 candidates for the first and last byte this gives 1024, 2048 or 4096 candidates in total. The first case corresponds to impossible differentials. For instance, consider the differences $\Delta y^{(7)} = (0,*,*)$. Those are clearly impossible because the input difference $\Delta x^{(7)}$ must be nonzero for the topmost S-box (as LL uses an MDS matrix) and so must be the output difference.

Coming back to the time complexity of our attack, we argue that the intersection over three pairs leaves us with 2^{-27} candidates for $rk^{(8)}$. To do so, we assume that the candidates for $rk^{(8)}$ are pairwise independent. Then, we estimate the number of left candidates after the first intersection as $2^7 \cdot 2^7 \cdot 2^{-24} = 2^{-10}$, i.e., we multiply the number of pairs of candidates for $rk^{(8)}$ with the probability that both candidates in the pair are the same. After the second intersection, we expect roughly $2^{-10} \cdot 2^7 \cdot 2^{-24} = 2^{-27}$ candidates, with the same reasoning.

For every pair and every candidate for $rk^{(8)}$, we expect two candidates for $\mathtt{LL}^{-1}(rk^{(7)})_0$. Again, we take the intersection. With the same reasoning as above, we are left with $2^{3\cdot 1 - 2\cdot 8} = 2^{-13}$ candidates. So, in total, we expect $2^{-27} \cdot 2^{-13} = 2^{-40}$ candidates for $(rk^{(8)}, \mathtt{LL}^{-1}(rk^{(7)})_0)$ for each $(rk^{(10)}, rk^{(9)})$. In other words, we expect $|K| \approx 2^8$ candidates

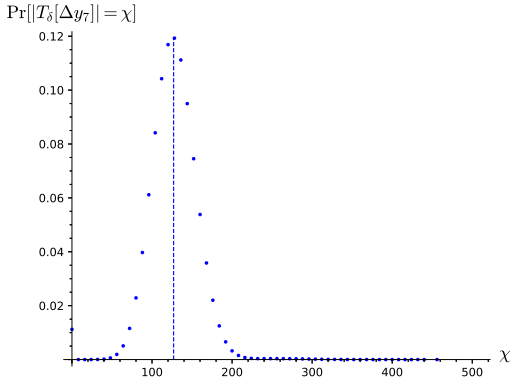


Figure 3: Plot of $\Pr[|T_\delta[\Delta y^{(7)}]| = \chi]$ where the probability is taken over the uniform choice of a nonzero $\delta \in \mathbb{F}_2^8$ and $\Delta y^{(7)} \in \mathbb{F}_2^{24}$. For clarity, there are no dots if the probability is zero and outliers at $\chi = 1024$, 2048 and 4096 were removed. The mean value $\mu = 127$ is marked with the dashed line.

for the 80 bits of the key given by $\left(rk^{(10)}, rk^{(9)}, rk^{(8)}, \mathtt{LL}^{-1}(rk_0^{(7)})\right)$. This matches the line of argumentation that we guess 80 bits of the key and filter with a $3 \cdot 24 = 72$-bit condition that we applied for our naive approach. Finally, for each of such candidate, we brute force the remaining 48 bits of the key and hence the overall time complexity is dominated by the 2^{56} evaluations of HALFLOOP-24 in the last step.

Memory Complexity The memory complexity is clearly dominated by the memory required to store T. We already studied the average size of $T_\delta[\Delta y^{(7)}]$ and so it is easy to estimate the size of T for three different differential pairs, as

$$3 \cdot 2^{24} \cdot 2^7 \cdot 3 \text{ bytes } = 18 \text{ GB}.$$

Data Complexity In contrast to [DDLS22], our attack does not rely on any special ciphertext difference. That is, we need three differential plaintext-tweak-ciphertext pairs $\big((p,t,c),(p',t',c')\big)$ where $\Delta p = (0,0,\delta)$ and $\Delta t = (0,0,\delta,0,0,0,0,0)$ for three nonzero δ and where c, c' are the corresponding ciphertexts, of course, encrypted under the same key. In a chosen-plaintext-tweak setting, this requires only six queries which is an improvement by a factor of about 2^{15} compared to [DDLS22]. This reduction of the data complexity is our main improvement. As we discuss in Subsection 3.4, this improvement transfers to a more practical known-plaintext scenario in the context of ALE and hence results in an attack that is applicable in the real world.

Time-Data Trade-Offs We briefly discuss how our attack performs with less data since, for HALFLOOP-24, the data complexity is what matters the most in the real world. As the dominating step for the time complexity is the final exhaustive enumeration, it is sufficient to consider the naive variant of our attack here.

If we have only one differential pair and four more arbitrary plaintext-tweak-ciphertext pairs that we need to uniquely identify the correct key in the final step, we get an attack with a time complexity of 2^{104} HALFLOOP-24 evaluations: we guess the 80 bits of the key, check a 24 bit condition and are left with 2^{56} candidates, i.e., we have to check $2^{56+48} = 2^{104}$ keys in the final step. The memory complexity is not affected since we do not have to store all the candidates but can check them immediately. Although this clearly is a valid attack, we rank the time complexity of 2^{104} evaluations of HALFLOOP-24

as unrealistic, even for large-scale adversaries. For a more detailed discussion on the practicality of attacks with enormous time complexity, we refer to [BG12].

If we have two differential pairs (and two more arbitrary plaintext-tweak-ciphertext pairs), we get a time complexity of 2^{80} as we can filter with a 48-bit condition and hence are left with 2^{32} candidates for each of which we have to check 2^{48} more. Again, the memory complexity is not affected. Of course, a time complexity of 2^{80} HALFLOOP-24 evaluations is still gigantic but we argue that this might be doable for large-scale adversaries such as state actors. For comparison, at time of writing, the hash rate of the Bitcoin network is in the ballpark of 2^{68} SHA-256 hashes per second.[1] Hence, it seems possible that a state actor could run our attack with only two differential pairs in the order of magnitude of a day (one day has roughly 2^{16} seconds). Surely, this is not a precise comparison, but it is still quite alarming, especially when we consider that a single key is used for an entire communication network and that it is not clear how often keys are changed in the fields.

Three pairs result in our main attack described above with a time complexity of 2^{56}. With four pairs, we are left with one unique candidate for the 80-bit partial key $\left(rk^{(10)}, rk^{(9)}, rk^{(8)}, \mathtt{LL}^{-1}(rk_0^{(7)})\right)$ and hence need only 2^{48} HALFLOOP-24 evaluations (and also 2^{48} set intersections in the first phase). Notice that there is no need to use the fourth pair in the intersection phase. We can simply use it to check the $\Delta z^{(5)} \stackrel{?}{=} 0$ conditions for approximating 2^8 candidates from the three-pair version. Thereby, we do not need any further precomputation and especially no additional memory accesses in the intersection phase.

3.3 Fine-Tuning the Attack

Here, we describe observations that allows for two separate approaches to improve our attack in practice. From a theoretical point of view, they do not change the time complexity of the attack. However, in practice, they might have a rather vast impact on the performance.

The observation, which is quite obvious once seen, is that T decomposes into unions of Cartesian products of three smaller sets (actually affine subspaces), one for each S-box. More precisely, let $\widetilde{\mathrm{DDT}}$ be similar to the DDT of the AES S-box S but instead of the cardinality of the set we consider the corresponding output values, i.e.,

$$\widetilde{\mathrm{DDT}}[\alpha][\beta] \coloneqq \{S(x) \in \mathbb{F}_2^8 \mid S(x) \oplus S(x \oplus \alpha) = \beta\}.$$

Then, for $\Delta x^{(7)} = \mathtt{LL}(\gamma, 0, 0) \oplus (0, \delta, 0)$, we have

$$\begin{aligned} T_\delta[\Delta y^{(7)}] &= \{\mathtt{SB}(x^{(7)}) \in \mathbb{F}_2^{24} \mid \exists \gamma : \delta \xrightarrow{S} \gamma \text{ and } \mathtt{SB}(x^{(7)}) \oplus \mathtt{SB}(x^{(7)} \oplus \Delta x^{(7)}) = \Delta y^{(7)}\} \\ &= \bigcup_{\substack{\gamma \\ \delta \xrightarrow{S} \gamma}} \bigtimes_{j=0}^{2} \widetilde{\mathrm{DDT}}[\Delta x_j^{(7)}][\Delta y_j^{(7)}]. \end{aligned}$$

In other words, we can replace the large precomputed table T by combining lookups to the significantly smaller table $\widetilde{\mathrm{DDT}}$. We present two approaches to make use of this.

First, we can enormously decrease the memory complexity by purging T. Consider Figure 2 again. For γ, $\Delta x_0^{(7)}$, $\Delta x_1^{(7)}$, and $\Delta x_2^{(7)}$, there is a pairwise one-to-one correspondence. This is because two differences at the input of LL are zero and so the correspondences are essentially multiplications in $\mathbb{F}_{2^8}$ with the constants from the MixColumns matrix. Using this and the observation above, it is possible to efficiently restore all possible $\Delta x^{(7)}$ from $\Delta y^{(7)}$ without the need for T. We know four one-byte differences, namely γ, $\Delta y_0^{(7)}$, $\Delta y_1^{(7)}$, and $\Delta y_2^{(7)}$, and, because of the one-to-one correspondences, each of those implies 127 (out

[1] See, e.g., https://www.blockchain.com/explorer/charts/hash-rate.

of 256) candidates for $\Delta x_0^{(7)}$ which in turn gives us all of $\Delta x^{(7)}$. When we intersect those, which we can efficiently do by ANDing 256-bit registers, we expect about $2^{8-4\cdot1} = 16$ candidates for $\Delta x^{(7)}$. For each S-box, we then expect two candidates for the corresponding byte of $\mathtt{LL}^{-1}(rk^{(8)})$ per candidate for $\Delta x^{(7)}$ and hence we expect $|RK^{(8)}| \approx 2^4 \cdot 2^3 = 2^7$. Of course, those are the same candidates that we stored in T before. With this technique, we reduce the memory complexity from about $3 \cdot 2^{24} \cdot 2^7 \cdot 3$ bytes = 18 GB for T to about $3 \cdot 3 \cdot 2^8 \cdot 256$ bits = 72 KB to map the known differences to candidates for $\Delta x^{(7)}$. For the change in run time, naively, we have to compare one (main memory) lookup to a couple of cached lookups and, on modern x86 CPUs, some AVX instructions. Although the specific changes surely depend on the concrete hardware, we assume that this would actually increase the run time on a single-core system. However, on a multi-core system (or even on dedicated hardware), where the memory for T would be shared across many computing units, it seems reasonable to assume that the latency of the many parallel main memory lookups is doomed to be a bottleneck of the attack.

Our second idea based on the observation above aims at improving the complexity of the intersection step. Recall that we have to compute (to avoid confusion with the byte index j, we omit the index i of the used differential pair for $\Delta y^{(7)}$ and $v^{(8)}$)

$$RK^{(8)} \leftarrow \bigcap_{i=0}^{2} T_{\delta_i}[\Delta y^{(7)}] \oplus v^{(8)}$$

for about 2^{48} times. That is, we repeatedly have to intersect three sets of 24-bit values. With the observation above this becomes

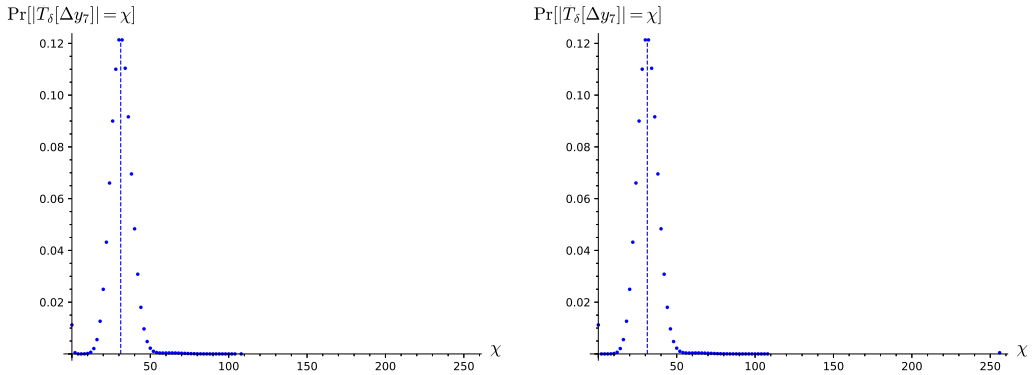
$$RK^{(8)} \leftarrow \bigcap_{i=0}^{2} \bigcup_{\delta_i \xrightarrow{S} \gamma} \bigtimes_{j=0}^{2} \left(\widetilde{\mathrm{DDT}}[\Delta x_j^{(7)}][\Delta y_j^{(7)}] \oplus v_j^{(8)} \right).$$

Implementation-wise, it would be preferable to do 8-bit intersection because then each set of 8-bit values can be represented by a 256-bit AVX register and the intersection is just an AND operation. Now, our key observation is that we can swap the Cartesian product and the union over the possible intermediate differences γ without loosing correctness of our attack. Thereby, we achieve the desired 8-bit intersection but, of course, the outcome of the intersection changes. That is, the now obtained set

$$\widetilde{RK}^{(8)} \leftarrow \bigtimes_{j=0}^{2} \bigcap_{i=0}^{2} \bigcup_{\delta_i \xrightarrow{S} \gamma} \left(\widetilde{\mathrm{DDT}}[\Delta x_j^{(7)}][\Delta y_j^{(7)}] \oplus v_{i_j}^{(8)} \right)$$

is a superset of the original candidates $RK^{(8)}$. Notice that $\widetilde{RK}^{(8)} \supset RK^{(8)}$ directly implies that our attack stays correct. Concerning the run time, to quantify the size of $\widetilde{RK}^{(8)}$, consider Figure 2 again. There, the nonzero δ can transition to 127 possible γ. The probability that γ transitions to a given difference $\Delta y^{(7)}$ is about 2^{-3} (0.5 for each S-box) and hence we only have to consider roughly $127 \cdot 2^{-3} \approx 16$ values for γ. Notice that we can either precompute a table $\widetilde{T}$ for those, using 4.5 GB of memory, or use an approach similar to the one presented above that eliminates the need for a large precomputation. Each γ is mapped one-to-one to $\Delta x^{(7)}$ and the lookup in $\widetilde{\mathrm{DDT}}$ yields (roughly) 2 candidates for one key byte. Hence, we expect approximately 32 candidates for each byte of $rk^{(8)}$ for each differential pair.

We validate this claim by exhaustively computing the number of candidates in Figure 4 in the same manner as we did for the joined candidates in Figure 3. The plot for the first and last byte are the same and given on the left hand side. They confirm our expectation



(a) Plot for bytes $j = 0$ and $j = 2$. $\mu \approx 31.14$.

(b) Plot for byte $j = 1$. $\mu \approx 31.26$.

Figure 4: Plot of $\Pr[|\widetilde{T}_\delta[\Delta y^{(7)}]| = \chi]$ where the probability is taken over the uniform choice of a nonzero $\delta \in \mathbb{F}_2^8$ and $\Delta y^{(7)} \in \mathbb{F}_2^{24}$. For clarity, there are no dots if the probability is zero. The mean values μ are marked with dashed lines.

of 32 key candidates per byte. The plot for the middle byte is given on the right hand side and is quite similar except for the outlier at $\chi = 256$ which again is explained by the fact that $\Delta x_1^{(7)} = 0$ is possible. Therefore, for the byte-wise intersections using three differential pairs, we expect $2^{3\cdot 5-2\cdot 8} = 2^{-1}$ candidates per byte and hence 2^{-3} in total. For those, we check that $\Delta y^{(7)}$ indeed is of the form $(*, 0, 0)$ and then continue as before. We implemented this variation of our attack and give the experimental results in Subsection 3.5.

For an attack using only three good differential pairs, the impact of this is limited, as the final brute force search with a complexity of 2^{56} HALFLOOP-24 evaluations dominates the run time. But, as discussed in the previous section, if we have access to a fourth differential pair, we can reduce this to 2^{48} evaluations. Then, we need as many intersection computations as HALFLOOP-24 evaluations and therefore it is sensible to also optimize the first phase.

A Note on an Integral Attack For completeness, we want to point out that, in terms of time complexity, there is an even stronger attack, namely an integral attack. For this, we consider plaintext-tweak structures with all possible δ instead of only pairs, i.e., plaintexts and tweaks of the form

$$\left\{ \big(p \oplus (0,0,\delta), t \oplus (0,0,\delta,0,0,0,0,0)\big) \mid \delta \in \mathbb{F}_2^8 \right\}$$

for some $p \in \mathbb{F}_2^{24}$, $t \in \mathbb{F}_2^{64}$. With this, we get a zero-sum property for each byte of $x^{(8)}$. That is, for all $j \in \{0, 1, 2\}$, we have (omitting index i for readability)

$$\bigoplus_i x_j^{(8)} = 0$$

and hence it is enough to guess 32 bits at once: all $rk^{(10)}$ and one byte of $\mathtt{LL}^{-1}(rk^{(9)})$. This way, it is also possible to recover the aforementioned 80 bits of the master key with a dominant time complexity of $3 \cdot 2^8 \cdot 2^{32}$ times looking up to the S-box table. However, it still remains to exhaustively search for the other remaining 48 bits of the key and more importantly it comes with a significantly increased data complexity. Therefore, we do not pursue this approach any further.

3.4 Practicality: Data Complexity in the Real World

We argue that the presented attack is practical in terms of time, memory and data complexity. That is, we are confident that our attack can be applied against HALFLOOP-24 used in ALE in the *real world*, provided that the adversary indeed *knows* plaintexts, i.e., that they hold some light intelligence on the victim's callsigns.

As discussed above, the time and memory complexity is very much practical: the memory-wise worst variation of our attack needs only about 18 GB of memory and with three good differential pairs we need only 2^{56} HALFLOOP-24 evaluations. Since the attack is trivial to parallelize, we claim that a large-scale, i.e., state-level adversary could run the attack easily in less than an hour.

This leaves the data complexity, which was also the main restriction in [DDLS22], as the limiting complexity. However, in contrast to [DDLS22], our attack does not rely on a special ciphertext difference and therefore needs far less data. Recall, from [DDLS22], that in a 16-minute window with n captured ALE frames there are about

$$\left\lfloor \frac{n^2}{8} - \frac{n}{2} \right\rfloor$$

differential pairs with a suitable tweak difference. Further, if we assume that on average m messages are send per minute and that the probability that the plaintext difference matches the tweak difference is ρ, the probability that a 16-minute window contains at least one good pair is

$$1 - (1-\rho)^{\lfloor \frac{(16m)^2}{8} - \frac{16m}{2} \rfloor} = 1 - (1-\rho)^{\lfloor 32m^2 - 8m \rfloor}.$$

As detailed in [DDLS22], the probability ρ that the plaintext difference is also fulfilled is highly dependent on the attacked network. For instance, if the network assigns callsigns uniformly at random, it holds that $\rho = 36^{-1} \cdot 18^{-1} \cdot 2^{-8} \approx 2^{-17.3}$. However, as [DDLS22, Table 3] demonstrates, at least in the case of unencrypted networks, there are examples where this probability is as high as $0.215 \cdot 2^{-8} \approx 2^{-10.22}$. In that case, if we assume a (high) number of $m = 6$ messages per minute, the probability for at least one good pair in a 16-minute window is

$$1 - (1 - 2^{-10.22})^{1104} \approx 0.60.$$

This in turn means that $0.6^{-1} \cdot 4$ windows of 16 minutes each, i.e., less than *two hours* of intersected traffic is enough to mount our four-pair attack with more than 50% success probability. In contrast, for the same setting, the attack in [DDLS22] needs around *541 years* worth of traffic. In conclusion, we double down on the conclusion in [DDLS22]: HALFLOOP-24 must not be used.

3.5 Experimental Results

We implemented the fine-tuned variation of our chosen-plaintext-tweak attack with three differential pairs in a lab setting. Our implementation is freely available online.[2] We executed the *full* attack on a server equipped with two AMD EPYC 7742 64-Core processors. We need about 50s for the precomputation steps and about 5 GB of memory. Checking all 2^{48} candidates for $(rk^{(10)}, rk^{(9)})$ took roughly 261 hours, i.e., slightly less than eleven days. As expected, there are some false positive candidates, namely 278, which fits our estimate of 2^8. Those could be eliminated using a fourth differential pair.

Furthermore, we use our implementation to experimentally validate our theoretical analysis of the steps of our attack. To do so, we execute five *partial* runs of the attack, i.e., we check only 2^{32} out of the 2^{48} candidates on our laptop. Regarding the number of

[2] See https://doi.org/10.5281/zenodo.10206087

candidates per byte of $rk^{(8)}$, our experiments confirm the estimate of about 32 candidates per byte. After computing the intersection, on average, 6.43% of the candidates for $(rk^{(10)}, rk^{(9)})$ are left. Notice that this is only half of what we predict in our analysis, i.e., our attack is better than initially assumed. However, the effect of this is only marginal, simply because the preceding steps dominate the run time anyway. Also, recalling our analysis, this slight inaccuracy is not unexpected. We assumed subsets of fixed size 32 and also independence which is too simple considering that, again assuming fixed size 32, our attack can only encounter

$$\frac{3 \cdot 255 \cdot 2^{24}}{\binom{256}{32}} \approx 2^{-101}$$

of all the possible subsets. For the surviving candidates, we check whether $\Delta y^{(6)} \stackrel{?}{=} (*, 0, 0)$ holds and thereby reduce the portion of left candidates to 0.056%. In this experiment, no candidate survived the final filter, i.e., there never was a candidate for $rk^{(7)}$. Considering the small number of tested keys, this is as expected.

Notice that we omit the exhaustive search of the remaining key bits which was already implemented in [DDLS22]. There the authors provide an implementation that finds the remaining 48 bits of the master key in roughly 5h on a 16-core machine. If only three differential pairs are available, this step has to be repeated about 256 times, but parallelizing this step is trivial.

4 Time-Data-Memory Trade-off Attack on HALFLOOP

In this section, we present a time-data-memory trade-off (TDM-TO) attack [DH77] which works on all three versions of full-round HALFLOOP. Note that previously, applying TDM-TO attacks on HALFLOOP ciphers was briefly mentioned in [DDLS22]. For completeness, we present the details of this attack here.

Recall that the tweak t is simply XORed to the first half of the key. Due the simple tweakey schedule, it is possible to use the following property: the encryption with key $(k' \,\|\, k'')$ and tweak t is the same as the encryption with $(k' \oplus \Delta \,\|\, k'')$ and $t \oplus \Delta$, for any 64-bit value Δ. This related-key related-tweak property of the cipher makes it possible to launch a related-tweak TDM-TO attack.

In the offline phase, the attacker randomly chooses d different plaintexts $p_1, \ldots, p_d$. Then for each 64-bit value of k'', he computes the corresponding ciphertexts $c_1, \ldots, c_d$ using the encryption with tweak 0^{64}, and key $(0^{64} \,\|\, k'')$. He saves the value of k'' in a hash table $\mathcal{T}$ at index $(c_1, \ldots, c_d)$. For an attack on HALFLOOP-b, that means that each index of the table contains $2^{64-d \cdot b}$ elements, on average.

In the online phase of the attack, for each 64-bit tweak t, he queries the corresponding ciphertexts for the same $p_1, \ldots, p_d$ plaintext values. We denote the corresponding ciphertexts as $c_1^*, \ldots, c_d^*$. By looking up to the index $(c_1^*, \ldots, c_d^*)$ of the hash table, he has candidates for the 128-bit key used for encryption. Namely, for each 64-bit k^* in $\mathcal{T}[c_1^*, \ldots, c_d^*]$, the 128-bit key value $(t \,\|\, k^*)$ is a candidate for the correct key.

Since on average there are $2^{64-d \cdot b}$ values in each index of the table, in total there will be $2^{64} \cdot 2^{64-d \cdot b} = 2^{128-d \cdot b}$ candidates for the 128-bit correct key. To find the exact 128-bit correct key, the attacker needs to query the corresponding ciphertexts for another $\lceil \frac{128-d \cdot b}{b} \rceil = \lceil \frac{128}{b} \rceil - d$ randomly chosen plaintext and tweak pairs. Then, he can do an exhaustive search on those $2^{128-d \cdot b}$ candidate keys to find the correct key for the encryption.

Attack Complexity In this attack, in the offline phase, the attacker needs $d \cdot 2^{64}$ encryptions to pre-compute the entries of the hash table. Then, in the online phase, he needs to query the ciphertexts for $d \cdot 2^{64}$ different plaintext-tweak values. Besides, the exhaustive search step of the attack requires computing about $2^{128-d \cdot b}$ encryptions.

The data complexity of the attack is $d \cdot 2^{64}$ chosen-plaintext-tweak data together with $\lceil \frac{128}{b} \rceil - d$ known-plaintext-tweak data, and its memory complexity is saving the hash table $\mathcal{T}$. In the case of $d \cdot b < 64$, the table requires saving about 2^{64} of 64-bit words which is equal to 2^{67} bytes. In the case of $d \cdot b > 64$, we can use the first 64 bits of the d ciphertext values as an index for the table and put the rest of it, together with the value of k'', in the content of the index which it is then used in the matching. In this case, on average, each index of the table contains one 64-bit value for the key and $(d \cdot b - 64)$-bit value for the remaining ciphertext(s) bits. This means saving the table requires memory of about 2^{64} $d \cdot b$-bit words.

For HALFLOOP-24, HALFLOOP-48, HALFLOOP-96, respectively by setting $d = 3$, $d = 2$, $d = 1$, the complexity of attack is about $3 \cdot 2^{64}$, 2^{65}, 2^{64} encryptions for each offline and online computations, $3 \cdot 2^{64}$, 2^{65}, 2^{64} of chosen-plaintext-tweak, and $9 \cdot 2^{67}$, $12 \cdot 2^{67}$, $12 \cdot 2^{67}$ bytes of memory.

In conclusion, we highlight that to build a tweakey schedule, even with applying a strong key schedule, simply XORing the tweak to (some part of) the master key is not a correct approach. This always makes it possible tho have a TDM-TO attack with a complexity of order $2^{|t|}$ chosen-plaintext-tweak-ciphertext pair of data, and order of $2^{|k|-|t|}$ memory cost and computations in each offline and online phases.

5 Meet-in-the-Middle Attack on HALFLOOP

In this section, we present Demirci-Selçuk Meet-in-the-Middle (DS-MITM) attacks [DS08] on full-round HALFLOOP-48 and reduced to 7-round HALFLOOP-96 block ciphers. By the nature of MITM attacks, our attacks in this section also focus on reducing the data complexity.

5.1 DS-MITM Attack on Full-Round HALFLOOP-48

Consider the key schedule of HALFLOOP-48 shown in Figure 5. Since the tweak t is simply XORed to the first half of the key, k', by inserting a difference in a single byte of the tweak, only some bytes of the round keys will be affected by this difference. For instance, for the difference shown in Figure 5, for the tweak difference $\Delta t = (0, 0, 0, 0, 0, \alpha, 0, 0)$, with α being a nonzero 8-bit value, only 21 bytes (out of 66) shown with orange and red colors of the round keys will be affected. The difference in 8 bytes of these 21 bytes is the same as α which are determined with orange color. Namely:

$$\begin{aligned}
\Delta rk^{(0)} &= (\ \ 0,\ \ 0,\ 0,\ 0,\ \ 0,\ \ \alpha\ ), & \Delta rk^{(1)} &= (\ 0,\ 0,\ 0,\ \ 0,\ 0,\ 0\ ),\\
\Delta rk^{(2)} &= (\ \ 0,\ \ 0,\ 0,\ 0,\ \ 0,\ \ 0\ ), & \Delta rk^{(3)} &= (\ 0,\ 0,\ 0,\ \alpha,\ 0,\ 0\ ),\\
\Delta rk^{(4)} &= (\ \ 0,\ \ \alpha,\ 0,\ 0,\ \ 0,\ \ \alpha\ ), & \Delta rk^{(5)} &= (\ 0,\ 0,\ \beta,\ 0,\ 0,\ 0\ ),\\
\Delta rk^{(6)} &= (\ \ \beta,\ \ \alpha,\ 0,\ 0,\ \ \beta,\ \ 0\ ), & \Delta rk^{(7)} &= (\ 0,\ 0,\ \beta,\ \alpha,\ 0,\ 0\ ),\\
\Delta rk^{(8)} &= (\ \ \gamma,\ \ 0,\ 0,\ \delta,\ \beta \oplus \gamma,\ \alpha\ ), & \Delta rk^{(9)} &= (\ 0,\ \delta,\ \gamma,\ \alpha,\ 0,\ \delta\ ),\\
\Delta rk^{(10)} &= (\ \beta \oplus \gamma,\ 0,\ 0,\ \delta,\ \ \gamma,\ \ 0\ ), & &
\end{aligned}$$

where

$$\begin{aligned}
\beta &= S(rk^{(4)}_{2,1}) \oplus S(rk^{(4)}_{2,1} \oplus \alpha),\\
\gamma &= S(rk^{(7)}_{0,1}) \oplus S(rk^{(7)}_{0,1} \oplus \alpha) \oplus \beta,\\
\delta &= S(rk^{(7)}_{2,0}) \oplus S(rk^{(7)}_{2,0} \oplus \beta).
\end{aligned}$$

Due to the nonlinearity of above equations, for a given value of α determining values of β, γ and δ requires knowing three round key bytes: $rk^{(4)}_{2,1}, rk^{(7)}_{2,0}$ and $rk^{(7)}_{0,1}$. This property of the key schedule in HALFLOOP-48 makes it possible to launch a DS-MITM attack as it is described in the following.

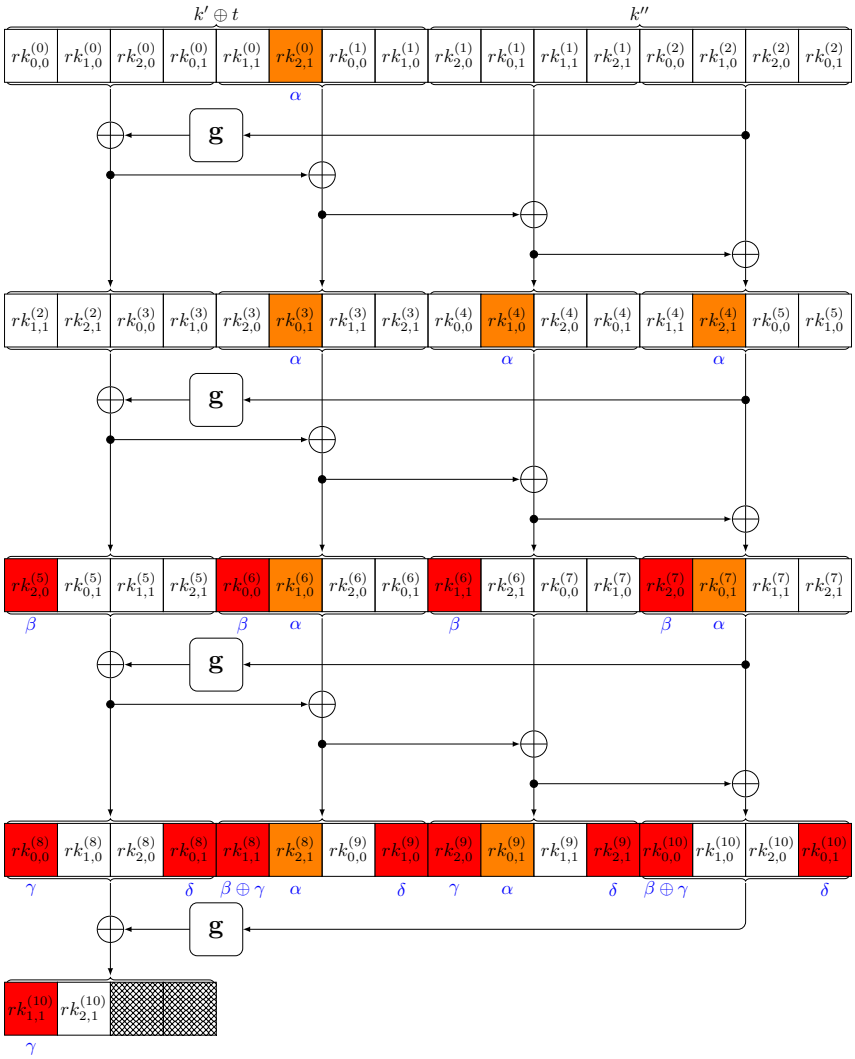


Figure 5: Key schedule of HALFLOOP-48 together with the effect of tweak difference $(0, 0, 0, 0, 0, \alpha, 0, 0)$. The orange and red blocks depict the round key bytes that will be affected by the tweak difference: The orange ones are the bytes with difference equal to α, but the red ones are the bytes whose difference is dependent on the value of some round key bytes. The induced difference in these bytes are written in blue.

Forward Direction For a given $\alpha \in \mathbb{F}_2^8$, we define the difference in plaintext and tweak states as below:

$$\Delta p = (0,0,0,0,0,\alpha)\,, \qquad \Delta t = (0,0,0,0,0,\alpha,0,0)\,.$$

This way it is possible to cancel the effect of tweak difference in the data path for the first three rounds. In other words, we have $\Delta x^{(r)} = \Delta y^{(r)} = \Delta z^{(r)} = (0,0,0,0,0,0)$ for $r \in \{0,1,2\}$. The difference in the data path propagates again by XOR of the difference in $rk^{(3)}$ where we have $\Delta x^{(3)} = (0,0,0,\alpha,0,0)$.

By guessing the value of $x^{(3)}_{0,1}$, we can compute the value for $\Delta x^{(4)}$:

$$\Delta x^{(4)} = \texttt{LL}\Big((0,0,0,S(x^{(3)}_{0,1} \oplus \alpha) \oplus S(x^{(3)}_{0,1}),0,0)\Big) \oplus (0,\alpha,0,0,0,\alpha)\,.$$

One step forward, by guessing the values for four bytes of $x^{(4)}_{i,j}$ with $(i,j) \in \{(1,0),(0,1),(1,1),(2,1)\}$, we can compute the values for five bytes of $\Delta x^{(5)}$:

$$\Delta y^{(4)}_{i,j} = \begin{cases} S(x^{(4)}_{i,j}) \oplus S(x^{(4)}_{i,j} \oplus \Delta x^{(4)}_{i,j}) & \text{if } (i,j) \in \{(1,0),(0,1),\,(1,1),(2,1)\}\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

$$\Rightarrow \quad \Delta x^{(5)} = \texttt{LL}(\Delta y^{(4)}) \oplus (0,0,0,\beta,0,0)\,.$$

Since, determining value of β requires knowledge about the value of $rk^{(4)}_{2,1}$, we cannot determine the value of $\Delta x^{(5)}_{0,1}$ by only knowing the values for aforementioned five bytes from $x^{(3)}$ and $x^{(4)}$.

As the last step in forward direction, by guessing the values for $rk^{(4)}_{2,1}$ and $x^{(5)}$, we can compute the value for $\Delta x^{(6)}$:

$$\Delta x^{(6)} = \texttt{LL}\big(\texttt{SB}(x^{(5)}) \oplus \texttt{SB}(x^{(5)} \oplus \Delta x^{(5)})\big) \oplus (\beta,\alpha,0,0,\beta,\alpha)\,.$$

Figure 6 depicts the approach of making a meeting point at $\Delta x^{(6)}_{0,0}$. The corresponding bytes which we know their value by guessing the aforementioned 12 bytes, are shown with orange color, and the bytes which we only know the difference in these bytes are shown with apricot color, but we leave the bytes with zero-difference with white color. We emphasis the bytes that we do not know the difference value in these positions, with gray hatch pattern. Some of the bytes that we know its exact value by guessing those aforementioned 12 bytes, are the bytes of $z^{(5)}$ that we will use this property in our attack.

Backward Direction To compute the value for $x^{(6)}_{0,0}$ from the ciphertext side, we need to know the values for whole $rk^{(10)}$ and $rk^{(9)}$ round keys, together with three bytes in the first column (the ones with $(i,0)$ indices) of the equivalent round key $\texttt{LL}^{-1}(rk^{(8)})$ and the first byte (the one with $(0,0)$ index) of the equivalent round key $\texttt{LL}^{-1}(rk^{(7)})$.

We also need to compute the value for $\Delta x^{(6)}_{0,0}$ from the ciphertext side for a given value of α. Therefore, it is necessary to determine the values for β, γ and δ which requires knowing $rk^{(4)}_{2,1}, rk^{(7)}_{2,0}$ and $rk^{(7)}_{0,1}$ round key bytes.

Considering the relation between the round key bytes based on the key schedule of HALFLOOP-48 shown in Figure 5, some of these round key bytes are dependent. Thereby, computing $x^{(6)}_{0,0}$ and $\Delta x^{(6)}_{0,0}$ only requires $rk^{(10)}$, $rk^{(9)}$, $rk^{(8)}_{2,1}$, $rk^{(4)}_{2,1}$ round key bytes together with extra $6+4$ bits from the equivalent round key $\texttt{LL}^{-1}(rk^{(8)})$. In total, we need to guess $(6+6+1+1)\cdot 8 + 6 + 4 = 122$ bits of the 128-bit master key.

In Figure 6, the corresponding bytes or the bits which we know their value by guessing the aforementioned 122 key bits, are shown with green color, and the bytes which we only

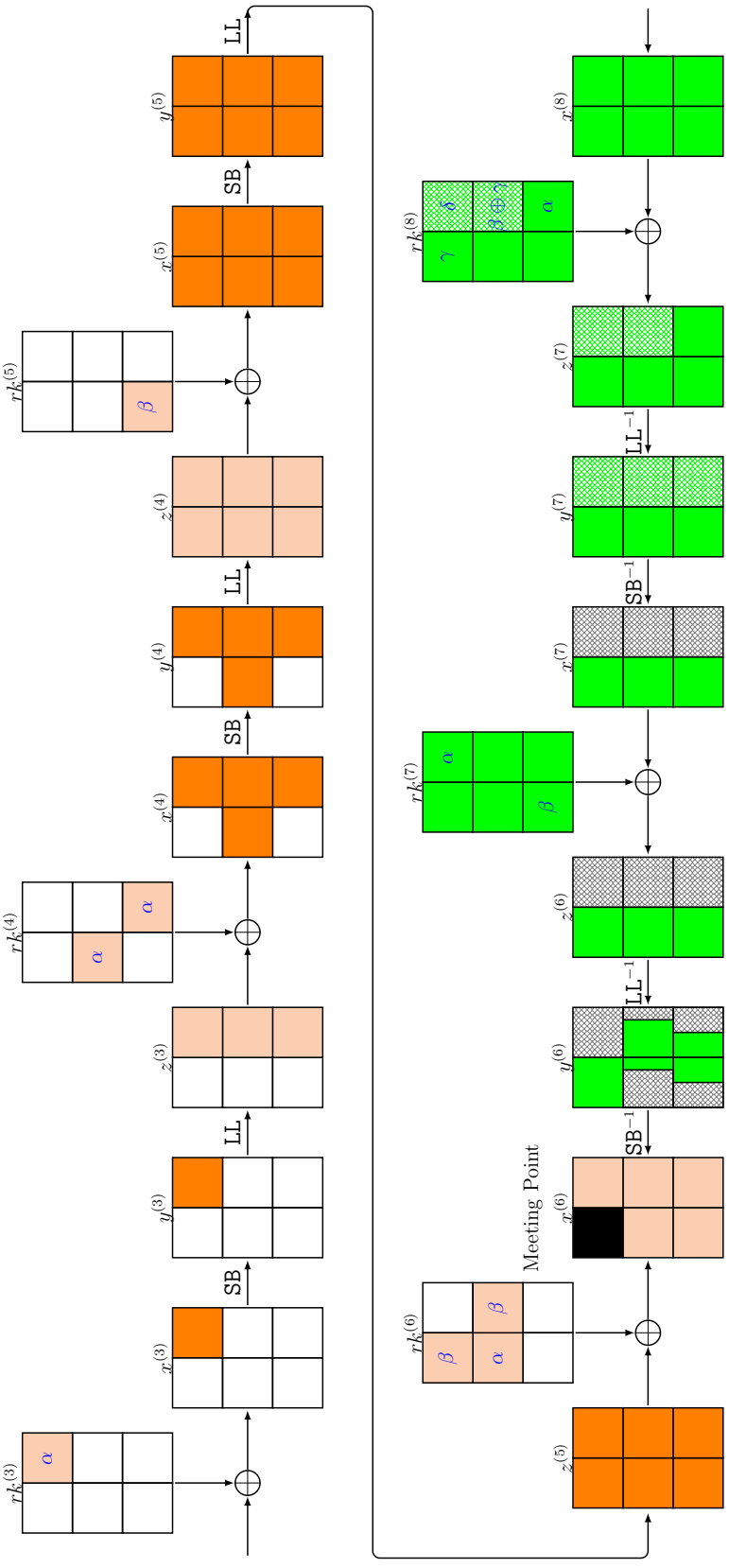


Figure 6: Demirci-Selçuk Meet-in-the-Middle attack on HALFLOOP-48. The bytes which we know their value in the forward direction are shown with orange color, and the bytes which we only know the difference in these bytes are shown with apricot color. The bytes which we know their value in the backward direction are shown with green color.

know linearly-dependent information of its bits are shown with green hatch pattern. We emphasis the bytes or the bits that we do not know the difference value in these positions, with gray hatch pattern.

Attack Procedure In the offline phase of the attack, we randomly choose d different nonzero 8-bit values for $\alpha_1, \ldots, \alpha_d$ where d is an integer greater than 12. We denote the value for $\Delta x^{(6)}_{0,0}$ when $\alpha = \alpha_i$ by $\Delta x^{(6)}_{0,0}\big|_{\alpha_i}$ for each i with $1 \leq i \leq d$.

For all values for 12 bytes of $x^{(3)}_{0,1}$, $x^{(4)}_{i,j}$ with $(i,j) \in \{(1,0),(0,1),(1,1),(2,1)\}$, $x^{(5)}_{i,j}$ for all i and j values, and $rk^{(4)}_{2,1}$, we compute the values of $\Delta x^{(6)}_{0,0}\big|_{\alpha_i}$ for each α_i and the value of $z^{(5)}_{0,0}$. We save these $(d+1)$ bytes together with the value of $rk^{(4)}_{2,1}$ in a hash table $\mathcal{T}$.

For ease of application, we can use the first 12 bytes (i.e., $\Delta x^{(6)}_{0,0}\big|_{\alpha_i}$ with $1 \leq i \leq 12$) as of the index for the hash table and put the remaining $(d-10)$ bytes (i.e., $\Delta x^{(6)}_{0,0}\big|_{\alpha_i}$ with $12 < i \leq d$, $rk^{(4)}_{2,1}$, and $z^{(5)}_{0,0}$) in the content of the table. This way, in average, each index of the table contains $(d-10)$ bytes.

In the online phase, we choose a random plaintext $p \in \mathbb{F}_2^{48}$, a random tweak $t \in \mathbb{F}_2^{64}$ and for each α_i chosen in the offline phase, we define $p|_{\alpha_i} = p \oplus \Delta p|_{\alpha_i}$ and $t|_{\alpha_i} = t \oplus \Delta t|_{\alpha_i}$. For each plaintext-tweak pair $(p|_{\alpha_i}, t|_{\alpha_i})$, we query the corresponding ciphertext which we denote it by $c|_{\alpha_i}$. We also query the ciphertext c which corresponds to the (p,t) plaintext-tweak pair.

For each 122-bit value for $rk^{(10)}$, $rk^{(9)}$, $rk^{(8)}_{2,1}$, $rk^{(4)}_{2,1}$ and the other 10 bits from $\mathtt{LL}^{-1}(rk^{(8)})$, we partially decrypt c and each $c|_{\alpha_i}$, to compute $x^{(6)}_{0,0}$ and $\Delta x^{(6)}_{0,0}\big|_{\alpha_i}$ for all $1 \leq i \leq d$. By looking up to the index $x^{(6)}_{0,0}\big|_{\alpha_1} \| \ldots \| x^{(6)}_{0,0}\big|_{\alpha_{12}}$ of the table $\mathcal{T}$, we check if the computed values for $\Delta x^{(6)}_{0,0}\big|_{\alpha_i}$ with $12 < i \leq d$ and $rk^{(4)}_{2,1}$ in the backward direction match with the saved values from the forward direction. This matching happens with probability of $2^{-8\cdot(d-12+1)}$ on average.

In the case of the first matching for $\Delta x^{(6)}_{0,0}\big|_{\alpha_i}$ bytes, we use the saved value for $z^{(5)}_{0,0}$ from the offline phase and the computed value for $x^{(6)}_{0,0}$ from the online phase to get the value for round key byte $rk^{(6)}_{0,0}$ by XORing those two bytes. Then, we check if this one round key byte matches with the guessed value for 122 bits of the key. This matching happens with a probability of $2^{128-122-8} = 2^{-2}$ on average.

In the case of the second matching, we have a candidate for the 128-bit master key. Using this key candidate, we compute the encryption for already-existing plaintext-tweak pairs (i.e., $(p|_{\alpha_i}, t|_{\alpha_i})$) to see if the computed value is the same as the queried ciphertext value (i.e., $c|_{\alpha_i}$).

Attack Complexity Applying this attack requires $(d+1)$ chosen-plaintext-tweak data and a memory of saving $2^{8\cdot 12}$ of $(d-10)$ bytes.

In the offline phase to compute the hash table $\mathcal{T}$, we need $(d+1)\cdot 2^{8\cdot 12}$ of partial encryptions. We optimize the cost of this partial encryption by splitting the guess of 10 bytes to several steps of guess-and-compute. This way, the cost of each partial encryption will be about one look up table to the S-box table.

In the online phase, we need $(d+1)\cdot 2^{122}$ times of partial decryption and 2^{122} times looking up to $\mathcal{T}$ table. Again, we optimize the cost of this partial decryption by splitting the guess to several steps of guess-and-compute. This way, the cost of each partial decryption will be about two look up table to the inverse of S-box table. Besides, in the exhaustive search step, there will be about $2^{128-8\cdot(d-11)-2}$ key candidates.

By setting $d = 12$, our attack needs about 2^{97} bytes of memory, together with about $\frac{1}{60}\cdot 13\cdot 2^{96}$ encryptions in the offline phase, $\frac{2}{60}\cdot 13\cdot 2^{122} + 2^{128-8-2}$ encryptions with

2^{122} look-up tables in the online phase. That means the time complexity of the attack is dominant by 2^{121} encryptions and 2^{122} look-up tables.

Note that compared to the TDM-TO attack on full-round HALFLOOP-48, the time complexity of the DS-MITM attack is comparably higher, however, its data complexity is comparably lower which can be seen as a trade-off between time and data complexity.

5.2 DS-MITM Attack on 7-Round-Reduced HALFLOOP-96

Figure 7 illustrates the key schedule of HALFLOOP-96 reduced to 7 rounds. Similar to the key schedule in the smaller versions of the cipher, by inserting a difference in a single byte of the tweak value, only some bytes of the round keys will be affected by this difference.

For the example shown in Figure 7 with the tweak difference $\Delta t = (\alpha, 0, 0, 0, \alpha, 0, 0, 0)$, with α being a nonzero 8-bit value, only 38 bytes (out of $8 \cdot 12 = 96$) shown with orange and red colors of the round keys will be affected. The difference in the 15 bytes of these 38 bytes is the same as α which are determined with orange color. Namely:

$$
\begin{aligned}
\Delta rk^{(0)} &= (\ \alpha,\ 0,\ \ 0,\ \ 0,\ \alpha,\ 0,\ 0,\ \ 0,\ \ 0,\ 0,\ \ 0,\ \ 0\ ),\\
\Delta rk^{(1)} &= (\ 0,\ 0,\ \ 0,\ \ 0,\ \alpha,\ 0,\ 0,\ \ 0,\ \ 0,\ 0,\ \ 0,\ \ 0\ ),\\
\Delta rk^{(2)} &= (\ 0,\ 0,\ \ 0,\ \ 0,\ 0,\ 0,\ 0,\ \ 0,\ \ \alpha,\ 0,\ \ 0,\ \ 0\ ),\\
\Delta rk^{(3)} &= (\ \alpha,\ 0,\ \ 0,\ \ 0,\ \alpha,\ 0,\ 0,\ \ 0,\ \ \alpha,\ 0,\ \ 0,\ \ 0\ ),\\
\Delta rk^{(4)} &= (\ \alpha,\ 0,\ \ 0,\ \ \beta,\ \alpha,\ 0,\ 0,\ \ \beta,\ \ \alpha,\ 0,\ \ 0,\ \ \beta\ ),\\
\Delta rk^{(5)} &= (\ \alpha,\ 0,\ \ 0,\ \ \beta,\ \alpha,\ 0,\ \gamma,\ \ \delta,\ \ 0,\ 0,\ \ \gamma,\ \ \beta\oplus\delta\ ),\\
\Delta rk^{(6)} &= (\ \alpha,\ 0,\ \ \gamma,\ \ \delta,\ 0,\ 0,\ \gamma,\ \beta\oplus\delta,\ \alpha,\ \epsilon,\ \ \zeta,\ \ \delta\ ),\\
\Delta rk^{(7)} &= (\ \alpha,\ \epsilon,\ \gamma\oplus\zeta,\ \beta,\ 0,\ \epsilon,\ \zeta,\ \beta\oplus\delta,\ 0,\ \epsilon,\ \gamma\oplus\zeta,\ \ 0\ ),
\end{aligned}
$$

where

$$
\begin{aligned}
\beta &= S(rk^{(3)}_{0,2}) \oplus S(rk^{(3)}_{0,2} \oplus \alpha)\ , & \gamma &= S(rk^{(5)}_{3,0}) \oplus S(rk^{(5)}_{3,0} \oplus \beta) \oplus \beta\ ,\\
\delta &= S(rk^{(5)}_{0,0}) \oplus S(rk^{(5)}_{0,0} \oplus \alpha)\ , & \epsilon &= S(rk^{(6)}_{2,1}) \oplus S(rk^{(6)}_{2,1} \oplus \gamma)\ ,\\
\zeta &= S(rk^{(6)}_{3,1}) \oplus S(rk^{(6)}_{3,1} \oplus \beta \oplus \delta) \oplus \gamma\,. & &
\end{aligned}
$$

Due to the nonlinearity of the equations above, for a given value of α determining values of $\beta, \gamma, \delta, \epsilon$ and ζ requires knowing five round key bytes: $rk^{(3)}_{0,2}, rk^{(5)}_{0,0}, rk^{(5)}_{3,0}, rk^{(6)}_{2,1}$ and $rk^{(6)}_{3,1}$. Applying this property of the key schedule in HALFLOOP-96, we present a DS-MITM attack on its reduced to 7-rounds version of the cipher in the following.

Forward Direction For a given $\alpha \in \mathbb{F}_2^8$, we define the difference in plaintext and tweak states as below:

$$\Delta p = (\alpha, 0, 0, 0, \alpha, 0, 0, 0, 0, 0, 0, 0)\,, \qquad \Delta t = (\alpha, 0, 0, 0, \alpha, 0, 0, 0)\,.$$

This way it is possible to cancel the effect of tweak difference in the data path for the first round: i.e., we have $\Delta x^{(0)} = \Delta y^{(0)} = \Delta z^{(0)} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$. The difference in the data path propagates again by XOR of the difference in $rk^{(1)}$ where we have $\Delta x^{(1)} = (0, 0, 0, 0, \alpha, 0, 0, 0, 0, 0, 0, 0)$.

By guessing the value of $x^{(1)}_{0,1}$, we can compute the value for $\Delta x^{(2)}$:

$$
\begin{aligned}
\Rightarrow\quad & \Delta y^{(1)} = \big(0, 0, 0, 0, S(x^{(1)}_{0,1} \oplus \alpha) \oplus S(x^{(1)}_{0,1}), 0, 0, 0, 0, 0, 0, 0\big)\\
\Rightarrow\quad & \Delta x^{(2)} = \mathtt{LL}(\Delta y^{(1)}) \oplus (0, 0, 0, 0, 0, 0, 0, 0, 0, \alpha, 0, 0, 0)\,.
\end{aligned}
$$

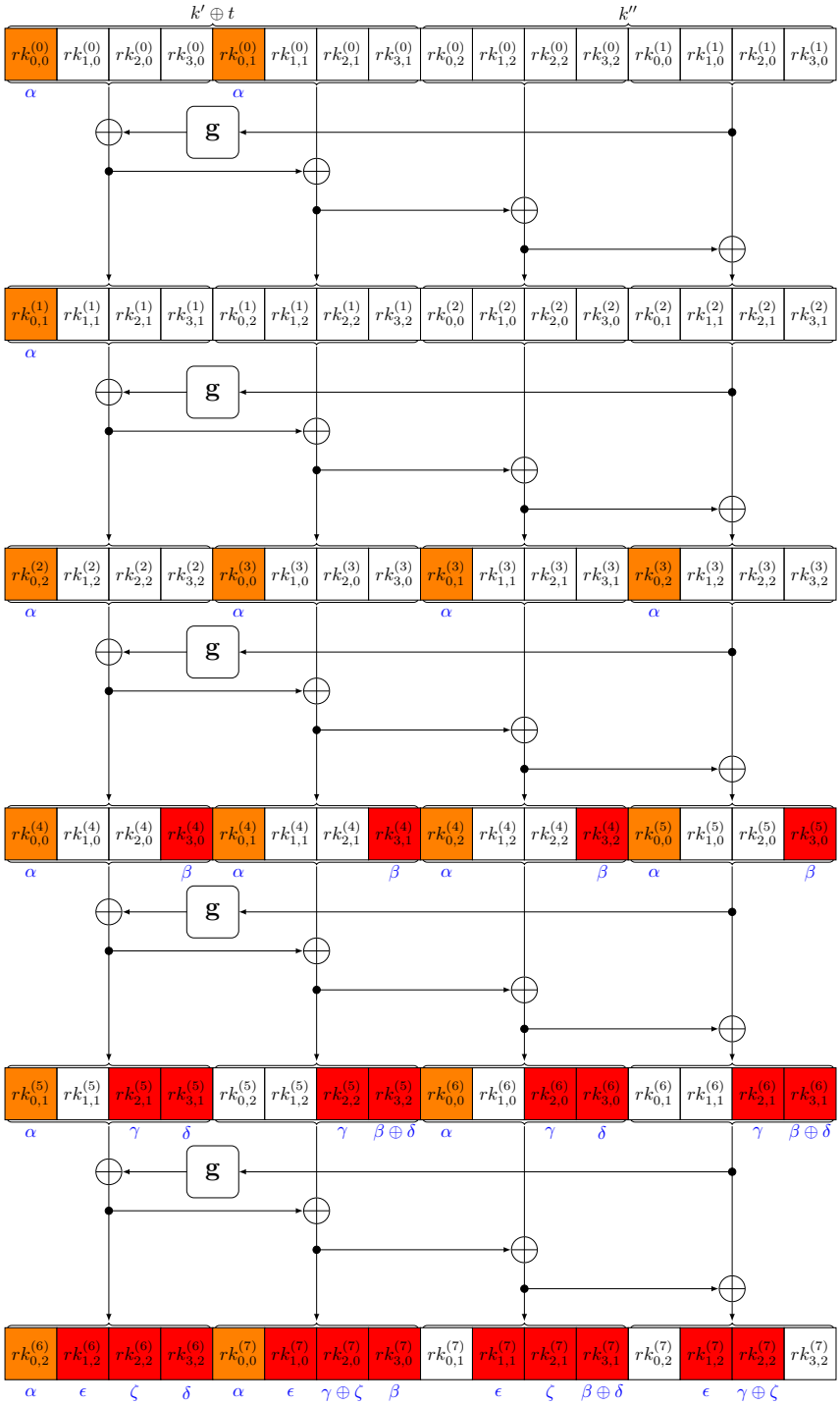


Figure 7: Key schedule of HALFLOOP-96 reduced to 7 rounds together with the effect of tweak difference $(\alpha, 0, 0, 0, \alpha, 0, 0, 0)$. The orange and red blocks depict the round key bytes that will be affected by the tweak difference: The orange ones are the bytes with difference equal to α, but the red ones are the bytes whose difference is dependent on the value of some round key bytes. The induced difference in these bytes are written in blue.

One step forward, by guessing the values for five bytes of $x^{(2)}_{i,j}$ with $(i,j) \in \{(0,1),(1,1),(2,1),(3,1),(0,2)\}$, we can compute $\Delta x^{(3)}$.

$$\Rightarrow \quad \Delta y^{(2)}_{i,j} = \begin{cases} S(x^{(2)}_{i,1}) \oplus S(x^{(2)}_{i,1} \oplus \Delta x^{(2)}_{i,1}) & \text{if } j = 1\,, \\ S(x^{(2)}_{0,2}) \oplus S(x^{(2)}_{0,2} \oplus \alpha) & \text{if } (i,j) = (0,2)\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

$$\Rightarrow \quad \Delta x^{(3)} = \mathtt{LL}(\Delta y^{(2)}) \oplus (\alpha,0,0,0,\alpha,0,0,0,\alpha,0,0,0)\,.$$

One more step forward, by guessing the values for seven bytes of $x^{(3)}_{i,j}$ with $(i,j) \in \{(0,0),(1,0),(3,0),(1,1),(2,1),(2,2),(3,2)\}$, we can compute the difference in the first column of the output for third round, i.e., $\Delta z^{(3)}_{i,0}$:

$$\Rightarrow \quad \Delta y^{(3)}_{i,j} = S(x^{(3)}_{i,j}) \oplus S(x^{(3)}_{i,j} \oplus \Delta x^{(3)}_{i,j}) \quad \text{for } (i,j) \text{ in the aforementioned set}\,,$$

$$\Rightarrow \quad \Delta z^{(3)}_{i,0} = \mathtt{LL}\big((\Delta y^{(3)}_{0,0}, \Delta y^{(3)}_{1,0}, 0, \Delta y^{(3)}_{3,0}, 0, \Delta y^{(3)}_{1,1}, \Delta y^{(3)}_{2,1}, 0, 0, 0, \Delta y^{(3)}_{2,2}, \Delta y^{(3)}_{2,3})\big)_{i,0} \ \forall i\,.$$

Note that not only the difference in these seven bytes can be computed, but also their exact value can be computed:

$$z^{(3)}_{i,0} = \mathtt{LL}\Big(\big(S(x^{(3)}_{0,0}), S(x^{(3)}_{1,0}), 0, S(x^{(3)}_{3,0}), 0, S(x^{(3)}_{1,1}), S(x^{(3)}_{2,1}), 0, 0, 0, S(x^{(3)}_{2,2}), S(x^{(3)}_{3,2})\big)\Big)_{i,0} \ \forall i\,.$$

Figure 8 illustrates our approach for making a meeting point at $z^{(3)}_{0,0}$ and $\Delta z^{(3)}_{0,0}$ bytes. The corresponding bytes which we know their value by guessing the aforementioned 13 bytes, are shown with orange color, and the bytes which we only know the difference in these bytes are shown with apricot color, but we leave the bytes with zero-difference with white color. We emphasis the bytes that we do not know the difference value in these positions, with gray hatch pattern.

Backward Direction To compute the value for $z^{(3)}_{0,0}$ from the ciphertext side, we need to know the values for whole $rk^{(7)}$ round key, four bytes in the first column (the ones with $(i,0)$ indices) of the equivalent round key $\mathtt{LL}^{-1}(rk^{(6)})$, the first byte (the one with $(0,0)$ index) of the equivalent round key $\mathtt{LL}^{-1}(rk^{(5)})$, and $rk^{(4)}_{0,0}$ round key byte.

We also need to compute the value for $\Delta z^{(3)}_{0,0}$ from the ciphertext side for a given value of α. Therefore, it is necessary to determine the values for $\beta, \gamma, \delta, \epsilon$ and ζ which requires knowing $rk^{(3)}_{0,2}, rk^{(5)}_{0,0}, rk^{(5)}_{3,0}, rk^{(6)}_{2,1}$ and $rk^{(6)}_{3,1}$ round key bytes.

Considering the relation between the round key bytes based on the key schedule of HALFLOOP-96 shown in Figure 7, some of these round key bytes are dependent. Thereby, computing $z^{(3)}_{0,0}$ and $\Delta z^{(3)}_{0,0}$ only requires knowing the values for whole $rk^{(7)}$ round key, $rk^{(3)}_{0,2}$ round key byte together with extra $6+4$ bits from the equivalent round key $\mathtt{LL}^{-1}(rk^{(6)})$. In total, we need to guess $(12+1)\cdot 8 + 6 + 4 = 114$ bits of the 128-bit master key.

In Figure 8, the corresponding bytes which we know their value by guessing the aforementioned 114 key bits, are shown with green color, and the bytes which we only know linearly-dependent information of its bits are shown with green hatch pattern. We emphasis the bytes or the bits that we do not know the difference value in these positions, with gray hatch pattern.

Attack Procedure The attack procedure is very similar to the attack on full-round HALFLOOP-48 in the previous subsection. In the offline phase of the attack, we set α_0 to be zero and randomly choose $(d-1)$ different nonzero 8-bit values for $\alpha_1, \ldots, \alpha_{d-1}$ where d is an integer greater than 13. We denote the value for $\Delta z^{(3)}_{0,0}$ when $\alpha = \alpha_i$ by $\Delta z^{(3)}_{0,0}\big|_{\alpha_i}$

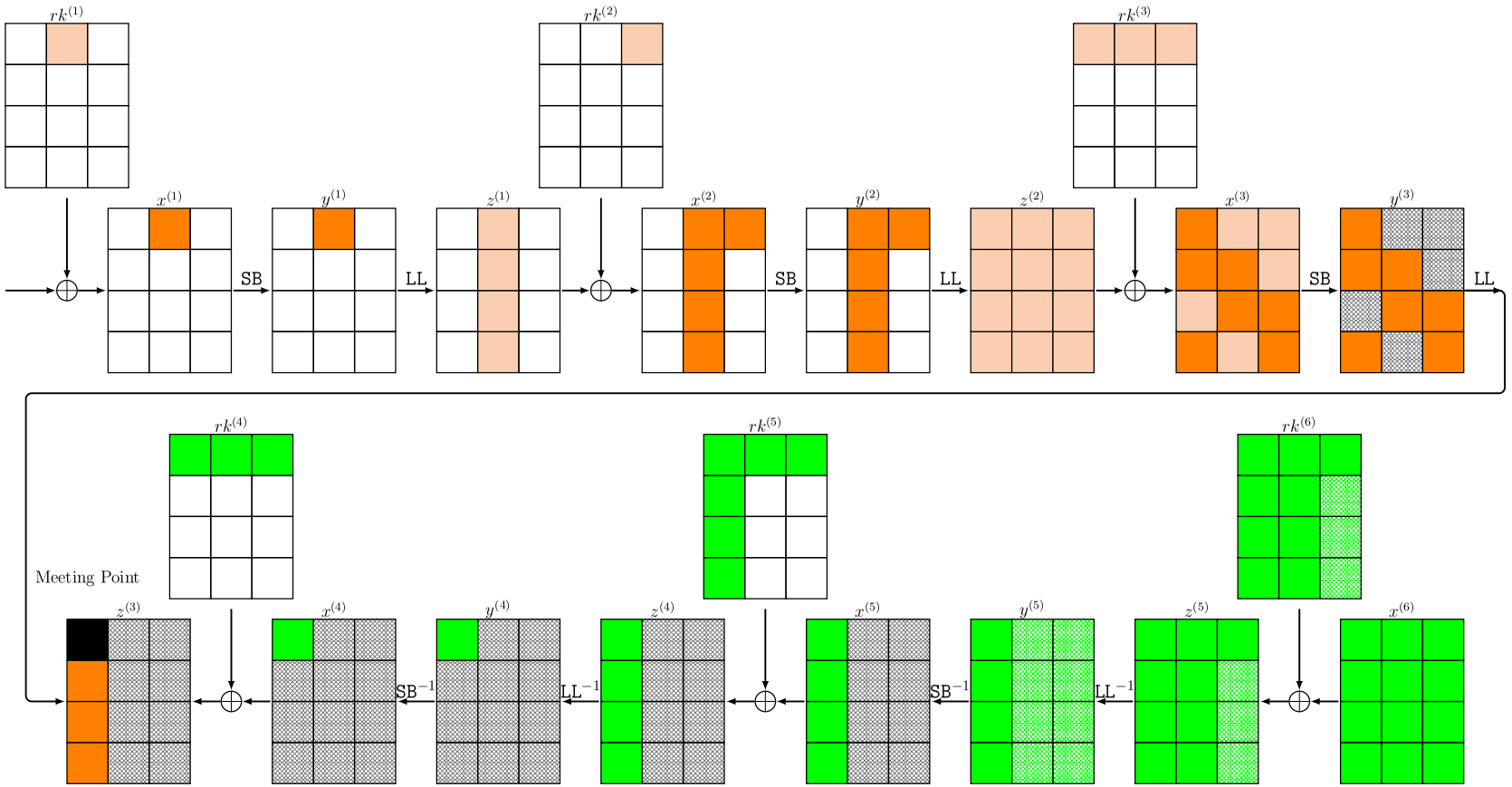


Figure 8: Demirci-Selçuk Meet-in-the-Middle attack on 7-round-reduced HALFLOOP-96. The bytes which we know their value in the forward direction are shown with orange color, and the bytes which we only know the difference in these bytes are shown with apricot color. The bytes which we know their value in the backward direction are shown with green color.

and also define $z^{(3)}_{0,0}\big|_{\alpha_i}$ with $z^{(3)}_{0,0}\big|_{\alpha_i} = z^{(3)}_{0,0} \oplus \Delta z^{(3)}_{0,0}\big|_{\alpha_i}$ for each i with $0 \leq i < d$. Note that $z^{(3)}_{0,0}\big|_{\alpha_0} = z^{(3)}_{0,0}$.

For all values for 13 bytes of $x^{(1)}_{0,1}$, $x^{(2)}_{i,j}$ with $(i,j) \in \{(0,1),(1,1),(2,1),(3,1),(0,2)\}$, and $x^{(3)}_{i,j}$ with $(i,j) \in \{(0,0),(1,0),(3,0),(1,1),(2,1),(2,2),(3,2)\}$, we compute the values of $z^{(3)}_{0,0}\big|_{\alpha_i}$ for each α_i. We save these d bytes in a hash table $\mathcal{T}$.

We use the first 13 bytes (i.e., $z^{(3)}_{0,0}\big|_{\alpha_i}$ with $0 \leq i \leq 12$) as of the index for the hash table and put the remaining $(d-13)$ bytes (i.e., $z^{(3)}_{0,0}\big|_{\alpha_i}$ with $12 < i < d$ in the content of the table. This way, in average, each index of the table contains $(d-13)$ bytes.

In the online phase, we choose a random plaintext $p \in \mathbb{F}_2^{96}$, a random tweak $t \in \mathbb{F}_2^{64}$ and for each α_i, we define $p|_{\alpha_i} = p \oplus \Delta p|_{\alpha_i}$ and $t|_{\alpha_i} = t \oplus \Delta t|_{\alpha_i}$. For each plaintext-tweak pair $(p|_{\alpha_i}, t|_{\alpha_i})$, we query the corresponding ciphertext which we denote it by $c|_{\alpha_i}$.

For each 114-bit value for $rk^{(7)}$, $rk^{(3)}_{0,2}$ and other 10 bits from $\mathtt{LL}^{-1}(rk^{(6)})$, we partially decrypt each $c|_{\alpha_i}$ to compute $z^{(3)}_{0,0}\big|_{\alpha_i}$. By looking up to the index $z^{(3)}_{0,0}\big|_{\alpha_0} \| \ldots \| z^{(3)}_{0,0}\big|_{\alpha_{12}}$ of the table $\mathcal{T}$, we check if the computed values for $z^{(3)}_{0,0}\big|_{\alpha_i}$ with $12 < i < d$ in the backward direction match with the saved value from the forward direction. This matching happens with probability of $2^{-8\cdot(d-13)}$ in average.

In the case of matching, we have a candidate for 114 bits from the 128-bit master key. Using this candidate, and guessing the remaining 14 bits we compute the encryption for already-existed plaintext-tweak pairs to see if the computed value is the same as the queried ciphertext value.

Attack Complexity Applying this attack requires d chosen-plaintext-tweak data and a memory for saving $2^{8\cdot 13}$ of $(d-13)$ bytes.

In the offline phase to compute the hash table $\mathcal{T}$, we need $d \cdot 2^{8\cdot 13}$ of partial encryptions. We optimize the cost of this partial encryption by splitting the guess of 13 bytes to several steps of guess-and-compute. This way, the cost of each partial encryption will be about one look up table to the S-box table.

In the online phase, we need $d \cdot 2^{114}$ times of partial decryption and 2^{114} times looking up to $\mathcal{T}$ table. Again, we optimize the cost of this partial decryption by splitting the guess to several steps of guess-and-compute. This way, the cost of each partial decryption will be about three look up table to the inverse of S-box table. Besides, in the exhaustive search step, there will be about $2^{128-8\cdot(d-13)}$ key candidates.

By setting $d = 15$, our attack needs about 2^{105} bytes of memory, together with about $\frac{1}{120} \cdot 15 \cdot 2^{104}$ encryptions in the offline phase, $\frac{3}{120} \cdot 15 \cdot 2^{114} + 2^{128-8\cdot 2}$ encryptions with 2^{114} look-up tables in the online phase. That means the time complexity of the attack is dominant by about $2^{113.3}$ encryptions and 2^{114} look-up tables.

6 Conclusion

We presented attacks on HALFLOOP-24 that are practical in all dimensions, in particular in terms of the data complexity. We therefore can only iterate what was already stated in [Dan21]: HALFLOOP does not provide adequate protection and should not be used.

This leaves the obvious question on how to fix the flaws in the design of HALFLOOP. Given our analysis, it is obvious that a different tweakey scheduling should be deployed in order to avoid the generic attacks presented in Section 4. Besides that, we actually think that the design of the round function itself is sound but obviously the number of rounds is not adequate. Thus a possible fix with limited effort is to use a tweakey scheduling along the lines of TWEAKEY framework [JNP14] and more rounds, e.g. 20 instead of 10 should

provide ample security margin even in the case of HALFLOOP-24. Detailing out the design and the analysis of such a cipher is left as possible future work.

Acknowledgments

The work described in this paper has been partially supported by the German Research Foundation (DFG) under Germany's Excellence Strategy EXC 2092 CASA - 390781972, by the Netherlands Organization for Scientific Research (NWO) under TOP grant TOP1.18.002 SCALAR, and also by the European Research Council (ERC) grant agreement no. 101097056 SymTrust.

References

[BG12] Alex Biryukov and Johann Großschädl. Cryptanalysis of the full AES using gpu-like special-purpose hardware. *Fundam. Informaticae*, 114(3-4):221–237, 2012.

[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.

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

[DDLS22] Marcus Dansarie, Patrick Derbez, Gregor Leander, and Lukas Stennes. Breaking HALFLOOP-24. *IACR Trans. Symmetric Cryptol.*, 2022(3):217–238, 2022.

[DH77] Whitfield Diffie and Martin E. Hellman. Special feature exhaustive cryptanalysis of the NBS data encryption standard. *Computer*, 10(6):74–84, 1977.

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

[DS08] Hüseyin Demirci and Ali Aydin Selçuk. A meet-in-the-middle attack on 8-round AES. In *Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers*, pages 116–126, 2008.

[JNP14] Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2014.

[Joh92] Eric E Johnson. A 24-bit encryption algorithm for linking protection. Technical report, Technical Report ASQB-OSO-S-TR-92-04, USAISEC, 1992.