Cryptanalytic Attacks on MIFARE Classic Protocol

Jovan Dj. Golić

Security Lab, Telecom Italia IT
Via Reiss Romoli 274, 10148 Turin, Italy
{jovan.golic}@it.telecomitalia.it

Abstract. MIFARE Classic is the most widely used contactless smart card in the world. It implements a proprietary symmetric-key mutual authentication protocol with a dedicated reader and a proprietary stream cipher algorithm known as CRYPTO1, both of which have been reverse engineered. The existing attacks in various scenarios proposed in the literature demonstrate that MIFARE Classic does not offer the desired 48-bit security level. The most practical scenario is the card-only scenario where a fake, emulated reader has a wireless access to a genuine card in the on-line stage of the attack. The most effective known attack in the card-only scenario is a differential attack, which is claimed to require about 10 seconds of average on-line time in order to reconstruct the secret key from the card. This paper presents a critical comprehensive survey of currently known attacks on MIFARE Classic, puts them into the right perspective in light of the prior art in cryptanalysis, and proposes a number of improvements. It is shown that the differential attack is incorrectly analyzed and is optimized accordingly. A new attack of a similar, differential type is also introduced. In comparison with the optimized differential attack, it has a higher success probability of about 0.906 and a more than halved on-line time of about 1.8 seconds.

Keywords: RFID, NFC, smart card attacks, key reconstruction attacks, stream ciphers, repeated nonce attacks, inversion atacks, resynchronization attacks, differential attacks.

1 Introduction

There are a number of proprietary algorithms and protocols used for data encryption and device authentication in RFID (Radio Frequency IDentification) and NFC (Near Field Communication) systems. The MIFARE Classic smart card, by NXP Semiconductors, is claimed to be the most widely used contactless smart card in the world, especially for access control to buildings and public transport. According to [6,7], this smart card covers more than 70% of the market share for access control worldwide. It is a memory card with several extra functionalities. It is capable of implementing a proprietary symmetric-key mutual authentication protocol and a proprietary encryption algorithm (stream cipher) known as CRYPTO1. They are also implemented on the dedicated contactless card reader. CRYPTO1 uses a preshared 48-bit secret key for encrypting messages between a card and a reader, including challenges and responses in the mutual authentication protocol. Data integrity on the RFID or NFC channel is not provided for. The protocol and CRYPTO1 are both reverse engineered in [14,7] and then analyzed in a number of scientific publications [14,12,6,7,4]. The proposed cryptanalytic attacks in various attack scenarios demonstrate that the MIFARE Classic smart card does not offer the 48-bit security level. Note that on a standard CPU, the brute-force attack on the 48-bit key can take several years, but less than an hour on the FPGA board COPACOBANA [13], at a cost of about 10000 USD. NXP Semiconductors has also introduced other smart cards for replacing MIFARE Classic, which use stronger authentication protocols and encryption algorithms, e.g., MIFARE Plus in 2008, based on 128-bit AES. According to [6], there are more than a billion of MIFARE and about 200 million of MIFARE Classic smart cards in use worldwide.

Attack scenarios considered include a passive scenario AttP and active scenarios AttAT and AttAR or their combinations. In AttP scenario, the attacker intercepts and records traces of valid transactions between a genuine card/tag and a genuine reader, and has the objective to decrypt some of the traces by cryptanalysis. There are a number of active scenarios where the attacker can initiate fake transactions between a tag and a reader. In AttAT scenario, the attacker uses a fake, emulated reader to access a genuine tag and, possibly in combination with AttP scenario, has the objective to perform an illegitimate transaction on the genuine tag, e.g., reading or modifying the stored data. In particular, in AttP or AttAT scenarios, the attacker may have the objective to reconstruct the genuine tag key. In AttAR scenario, the attacker uses a fake, emulated tag to access a genuine reader and has the objective to reconstruct the genuine reader key corresponding to the unique ID of the emulated tag. AttAT is the easiest scenario to implement in practice and is also called the tag-only or card-only scenario. In all the scenarios, the ProxMark instrument [15], with the open-source specification, programmed to handle the standard ISO/IEC 14443-A, can be used to emulate tags and readers and eavesdrop on valid transactions.

The reconstruction of the key totally breaks the system. The attacker can then perform any transaction on a genuine tag by using a fake reader with the genuine key. In particular, it can read or modify the stored data (e.g., read sensitive data or modify valuable data). Alternatively, the attacker can clone a tag, i.e., produce a fake tag with the genuine key or emulate a genuine tag and thus force a genuine reader into legitimate transactions or actions (e.g., access to building or access to any event requiring a ticket).

The main objectives of this paper are to critically analyze all known attacks on MIFARE Classic protocol in various scenarios and put them into the right perspective with respect to the prior art in cryptanalysis, propose their improvements, and introduce a novel attack, which appears to be the most effective currently known attack in the tag-only scenario, where the attacker uses a fake reader for a contactless access to a targeted genuine tag. For comparison of tag-only attacks, see Table 1. Note that the main practical limitation factor of these attacks is the on-line stage, which requires real-time access to the tag. The topic and the results are very interesting in practice, due the worldwide usage of MIFARE Classic smart cards and dedicated readers.

The MIFARE Classic protocol including the authentication protocol, the encryption algorithm CRYPTO1, and the error detection code are described in Section 2. The attack [12] in the combined AttP and AttAT scenario, which is independent of the structure of CRYPTO1, is discussed in Section 3. Two attacks [6] that work in AttP or AttAR scenario, namely, the time-memory-data tradeoff attack and the inversion attack are presented in Section 4 and Appendix A. Section 5 is dedicated to five attacks in AttAT, i.e., tag-only scenario. Three of them [7] are only outlined in Appendix B, as they are not very practical. The practical differential attack [4] is explained, critically analyzed, and optimized in Section 5 and Appendix C. A novel differential attack is proposed in Section 5 and Appendix C. All these cryptanalytic attacks relate to the mutual authentication protocol for one sector of the MIFARE Classic smart card. The attacks [6,7] aiming at reconstructing the keys for multiple sectors are presented in Section 6 and Appendix D. Conclusions are given in Section 7.

2 Description of MIFARE Classic Protocol

The EEPROM memory of the MIFARE Classic tag is divided into sectors, which are further divided into blocks of 16 bytes each. The last block of each sector contains two 48-bit secret keys and access conditions for the sector. The basic operations that can be performed on the memory data include read, write, increment, and decrement the stored value. The reader can access data in a given block only upon successful authentication for the sector containing that block, where the access conditions determine which of the two keys should be used and define the operations allowed for the sector. The first block of the first sector is a read-only block that contains special data including the unique 32-bit identifier (ID or UID) of the card, the parity byte computed on ID, and the manufacturer data. Secret keys stored on the tag can be specific to the tag or shared among a number of tags. In the latter case, a particular tag is identified by its unique ID stored in the read-only block. In the former case, in order to avoid storing all the keys in the reader memory, a key specific to the tag can be derived from a group master key and the unique ID.

The three-pass symmetric-key authentication protocol is of the challenge-response type, where the 32-bit challenge nonces n_T and n_R used by the tag and the reader are generated by the respective pseudorandom number generators. The tag nonce, n_T, is generated by a 16-bit linear feedback shift register (LFSR), which implies that it contains only 16 bits of entropy if the LFSR state is assumed to be uniformly random. The LFSR starts from the same state after powering up, has period 65535, and shifting its state every $9.44\,\mu$s it repeats its state after about 618 ms. The only randomness factor is thus a variable time when the tag nonce is produced. On the other hand, the reader pseudorandom number generator starts from the same state after every restart and produces a nonce n_R only upon invocation by the authentication protocol. In this case, the only randomness factor is a variable number of invocations after the restart. In principle, fresh nonces are important in order to avoid fake authentication by the replay attacks. However, if the nonces are treated as random in the protocol, but are in fact easily repeatable or predictable, then cryptanalytic attacks with a fake reader or a fake tag may be possible.

The tag and reader nonces serve for mutual authentication as well as for the initialization of CRYPTO1 for encrypting the data to be exchanged. CRYPTO1 is a stream cipher (keystream generator) with a structure of a nonlinear filter generator using a 48-bit LFSR and a nonlinear 20-bit Boolean function applied to the LFSR state to produce the keystream bits. The ciphertext bits are produced by XORing the plaintext and keystream bits. CRYPTO1 is initialized during the authentication process by using the challenge nonces n_T and n_R, which are bitwise XORed into the feedback path of the LFSR. Fresh nonces are important to ensure that the keystream is not repeated for the same sector key.

The main steps of the three-pass mutual authentication protocol are depicted in Fig. 1. In the preliminary two steps (which in reality include more auxiliary steps), the tag sends its unique ID and the reader sends back the index of the block (including that of the corresponding sector) to which it wishes to communicate to. The tag and the reader then select the key to be used for that sector, according to the access conditions. In the first pass, the tag sends n_T in the clear form. In the second pass, the reader sends back its response containing n_R and the answer a_R encrypted with two successive 32-bit keystream segments ks_1 and ks_2, respectively. In the third pass, the tag sends its answer a_T encrypted with the subsequent 32-bit keystream segment ks_3. We use the notation as in [7], i.e., $\{n_R\} = n_R \oplus ks_1$, $\{a_R\} = a_R \oplus ks_2$, and $\{a_T\} = a_T \oplus ks_3$. The answers to the challenges are defined by $a_R = \text{suc}^2(n_T)$ and $a_T = \text{suc}^3(n_T)$, where "suc" denotes the successor function associated with the 16-bit LFSR used for generating n_T, which maps 32 successive LFSR sequence bits into the next (non-overlapping) 32 successive LFSR sequence bits.

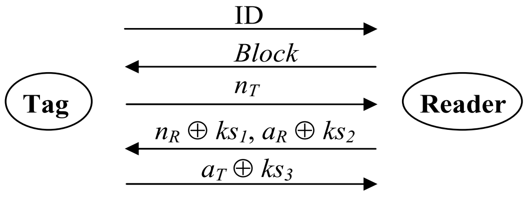

Fig. 1. MIFARE Classic authentication protocol

In the process, the same keystream is generated both by the tag and the reader, with the only difference that the reader uses n_R directly, whereas the tag first decrypts $\{n_R\}$ into n_R. This is achieved in the bitwise manner since ks_1 also depends on n_R, but in a specific way. Namely, as the current bit of ks_1

used to encrypt the current bit of n_R depends only on the previous bits of n_R, where the first bit of ks_1 does not depend on n_R at all, the bits of n_R can be recovered by the tag one at a time and then XORed into the feedback path of the LFSR. The LFSR in CRYPTO1 is first initialized with the 48-bit key as the initial state. Then, the LFSR is clocked 32 times during which $n_T \oplus \mathrm{ID}$ is XORed in the feedback loop of the LFSR, one bit at a time. Then, the LFSR is clocked another 32 times during which n_R is XORed in the feedback loop of the LFSR, one bit at a time. The first bit of ks_1 is generated after the insertion of the last bit of $n_T \oplus \mathrm{ID}$ and before the insertion of the first bit of n_R, whereas the last bit of ks_1 is generated before the insertion of the $32nd$ bit of n_R. The keystream segments ks_2 and ks_3, which are successively generated after ks_1, as well as the subsequent keystream bits are generated by clocking the LFSR autonomously, and thus depend on the key and all the bits of $n_T \oplus \mathrm{ID}$ and n_R.

Let $k = k_0k_1 \ldots k_{47}$, $n_T = n_{T,0}n_{T,1} \ldots n_{T,31}$, $n_R = n_{R,0}n_{R,1} \ldots n_{R,31}$, and $u = u_0u_1 \ldots u_{31}$ denote the 48-bit key and 32-bit tag and reader nonces and unique tag ID, respectively. Let $s = s_0s_1s_2 \ldots$ denote the LFSR sequence and let $S_i = s_is_{i+1} \ldots s_{i+47}$ and $L(S_i)$ denote the LFSR state and the feedback bit at time $i \geq 0$, respectively. The keystream sequence $z = z_0z_1z_2 \ldots$ is defined by

$$s_i = k_i, \quad 0 \leq i \leq 47, \tag{1}$$

$$s_{48+i} = L(S_i) \oplus n_{T,i} \oplus u_i, \quad 0 \leq i \leq 31, \tag{2}$$

$$s_{80+i} = L(S_{32+i}) \oplus n_{R,i}, \quad 0 \leq i \leq 31, \tag{3}$$

$$s_{112+i} = L(S_{64+i}), \quad i \geq 0, \tag{4}$$

$$z_i = f(S_i), \quad i \geq 0, \tag{5}$$

where f denotes the filter function applied to the LFSR state as a whole. The three keystream segments used in the protocol are $ks_1 = z_{32}z_{33} \ldots z_{63}$, $ks_2 = z_{64}z_{65} \ldots z_{95}$, and $ks_3 = z_{96}z_{97} \ldots z_{127}$. Effectively, f is defined as a balanced 20-bit Boolean function being a composition of five 4-bit Boolean functions and a 5-bit Boolean function. The tap positions forming inputs to f are taken from odd-indexed LFSR stages $9, 11, \ldots, 47$, so that we can write $f(S_0)$ as $f(s_9, s_{11}, \ldots, s_{47})$. It effectively depends on all 20 input bits and is linear neither in the first not the last input bit, i.e., it does not satisfy the sufficient condition [8] for the output sequence to preserve pure randomness of the input sequence to the filter function. In Section 5, it will be shown that this weakness opens a door for the most effective known attacks in the tag-only scenario.

If the additive inputs to the LFSR, $n_T \oplus u$ and n_R, are known, then it follows that the LFSR can also be clocked backwards, i.e., starting from the LFSR state at any time, one can determine all previous LFSR states, including the initial state defined by the key. As shown in [6,7], this so-called LFSR rollback also holds if the encryption $\{n_R\}$ is known instead of n_R, as a simple consequence of the fact that the filter function $f(S_0)$ does not effectively depend on the first, i.e., leftmost input bit s_0. We note that even when $f(S_0)$ effectively depends on s_0, the LFSR rollback is still feasible by the branching inversion/reversion attack [8,10]. Accordingly, the key can be easily recovered from any reconstructed internal state of CRYPTO1 and known transcripts of the authentication protocol.

The rationale for the mutual authentication is as follows. The correct encrypted answer $\{a_R \oplus ks_2\}$ can be produced by a genuine reader knowing the key and the nonces n_T and n_R. The correct encrypted answer $\{a_T \oplus ks_3\}$ can be produced by a genuine tag knowing the key and the nonces n_T and n_R, by first recovering n_R by decryption. The tag recovers a_R by decryption and verifies if it is equal to $\mathrm{suc}^2(n_T)$, thus authenticating the reader. The reader recovers a_T by decryption and verifies if it is equal to $\mathrm{suc}^3(n_T)$, thus authenticating the tag.

The mutual authentication protocol described above relates to the authentication for one sector. For authenticating access to multiple sectors of the same tag, the protocol is repeated by using new nonces n_T and n_R, with the only difference that, for each new sector, the new authentication command (Block) is encrypted with the previous sector key and the new n_T is bitwise encrypted by using the new key while it is bitwise inserted into the feedback path of the LFSR in the same way as n_R. The encrypted tag nonce sent by the tag is thus $n_T \oplus ks_0$, where $ks_0 = z_0 z_1 \dots z_{31}$ denotes the preceding 32-bit keystream segment. The attacker thus knows n_T only for the first sector.

According to the standard ISO/IEC 14443-A, every plaintext byte sent is followed by the parity bit for error detection, computed as the binary complement of the XOR of all 8 bits in the byte. The parity bits in MIFARE Classic communication protocol are thus computed over the plaintext and, importantly, each parity bit is then encrypted with the same keystream bit subsequently reused to encrypt the next bit of the plaintext. The fact that the parity bits are computed over the plaintext instead of the ciphertext implies that the ciphertext itself reveals linear relations among the keystream bits (one relation per 8+1 ciphertext bits), which may be useful for ciphertext-only cryptanalytic attacks. The fact that the keystream bits are repeatedly used implies that the ciphertext also reveals linear relations among the plaintext bits other than those determined by the parity bits (one relation per pair of ciphertext bits corresponding to the parity bit of the current byte and the first bit of the next byte). In particular, this weakness can be used for reducing the uncertainty of new tag nonces used for authenticating access to multiple sectors, because such nonces are sent in the encrypted form and need to be guessed, as pointed out above. For example, the entropy of n_T given $n_T \oplus ks_0$ is thus reduced to 13=16-3 bits.

3 Attacks on Genuine Session and Genuine Tag

The attack [12] works in the combined AttP and AttAT scenario which requires access to a genuine authentication session and a genuine tag, by a fake reader. It neither uses knowledge of the encryption algorithm CRYPTO1 nor aims at reconstructing the key. It only aims at reconstructing portions of the keystream, which are repeatedly used for encrypting data in fake sessions initiated by a fake reader, on the condition that the tag can be forced to use the same tag nonce n_T as the one from the intercepted genuine session. This can be achieved by using either the periodic query or the reset query technique, due to the weakness of the tag pseudorandom number generator, where the only randomness factor is a variable time when the tag nonce is produced. Both techniques are also important for other attacks with fake readers or fake tags considered in this paper.

The periodic query technique is made possible by the short period of the tag pseudorandom number generator, which repeats its state roughly every 680 ms. Accordingly, the same value of n_T can be verifiably obtained (since n_T is transmitted in the clear), by forcing the genuine tag into several fake authentication sessions, called queries to the tag. The attacker can thus initiate fake authentication sessions periodically, with a precise timing, and thus ideally get the same n_T every 680 ms. In practice, it would take several attempts to get the same n_T each time. Alternatively, the reset query technique [14,6] is based on the fact that the tag pseudorandom number generator always starts shifting from the same initial state, after resetting, which can be achieved by switching-off the field and powering-up the passive tag by the fake reader. The attacker thus initiates the queries at a fixed time after resetting the tag and repeats this operation as fast as possible until the same value of n_T is obtained. According to [6], the required number of attempts to get the same n_T does not exceed ten each time. The reset query technique is significantly faster than the periodic query technique.

In fake sessions with repeated n_T, it is also needed to repeat the same reader nonce n_R, which is unknown to the fake reader. To this end, the fake reader simply replays the second pass of the authentication protocol from the recorded transcripts of the intercepted genuine session, in each fake session with the repeated n_T. The reader will thus be successfully authenticated to the tag and the tag will reproduce the same n_R and automatically generate the same keystream.

Since the repeated tag nonce enables the fake reader to successfully authenticate itself in the fake session by replaying the corresponding transcripts of the genuine session, the genuine tag will then proceed with the session as it were authentic. The desired keystream portions can be recovered from known plaintext portions obtained either from the genuine session or fake sessions, e.g., from the tag ID and manufacturer data, access conditions for the sector, and known tag responses to modified encrypted reader commands enabled by malleable bitwise XOR encryption. The recovered keystream portions, for a fixed value of the tag nonce and a fixed response of the fake reader in a genuine session, can then be used to perform illegitimate transactions on the tag, such as reading and modifying data stored in memory blocks. More precisely, any known or partially known 16-byte keystream block for any sector enables the attacker to read and modify the data at the same bit positions in any other block from the same sector.

4 Attacks on Genuine Session or Genuine Reader

The attacks [6] work in AttP scenario requiring interception of a genuine authentication session or in AttAR scenario requiring access to a genuine reader by a fake tag. In the latter case, the authentication session cannot be terminated, because the fake tag cannot be successfully authenticated to the reader as it does not know the key. The objective is to reconstruct the key for a single sector from the known keystream segments obtained from the recorded transcripts of the authentication protocol by using the known tag nonces n_T and the derived values

$a_R = \mathrm{suc}^2(n_T)$ and $a_T = \mathrm{suc}^3(n_T)$. Each known n_T gives rise to 64 keystream bits in AttP scenario and 32 or 64 keystream bits in AttAR scenario, where 64 bits correspond to ks_2 and ks_3 and 32 bits to ks_2. The keystream segment ks_3 can be recovered also in AttAR scenario, regardless of the fact that the fake tag does not send anything in the third pass of the protocol, if the reader proceeds by sending back either an encrypted "halt" command (because the tag authentication failed) or an encrypted "read" command (as if the tag authentication were successful). According to [6], this happens for most readers in practical use.

The structure of the stream cipher CRYPTO1 is needed in the attacks. The attacks first recover an internal state of CRYPTO1, e.g., the 48-bit LFSR state S_{64} at a time immediately after all the bits of n_T and n_R have been fed into the LFSR. This is the state from which the first bit of ks_2 is produced. The secret key is then reconstructed by the LFSR rollback, as explained in Section 2. The two attacks proposed in [6] include a time-memory-data tradeoff (TMDT) attack and an inversion attack. The former works in AttAR scenario and uses keystream segments from multiple (fake) authentication sessions with a genuine reader, while the latter works in AttP or AttAR scenario and uses the keystream segments from one or two authentication sessions.

Both the attacks are essentially known from previous publications, which is not mentioned in [6]. The TMDT attack can be regarded as an adaptation of the generic TMDT attack [1,9], for a stream cipher with 2^{48} states. More precisely, if the stored states correspond to a special form of tag nonces chosen by the fake tag, then the attack succeeds with probability 1 instead of a high probability typical of TMDT attacks. The required number of keystream segments are obtained from fake authentication sessions using variable n_T and random n_R produced by the genuine reader. We point out that the number of authentication sessions can be reduced at the expense of increasing the precomputation time, while keeping the same computation time and memory, by applying the TMDT attack [2]. The inversion attack can be regarded as an adaptation of the inversion attack with the decimation technique [8,10] to decimated keystream segments shorter than the LFSR length. Recall that the generic inversion attack on a nonlinear filter generator with the input memory size M and the greatest common divisor of the pairwise differences between the tap positions to the filter function being equal to d takes about $2^{M/d}$ steps (in CRYPTO1, $M = 38$ and $d = 2$). The adapted attack takes about 0.05 seconds and 8 MByte of memory on a standard CPU to recover the key. For comparison, it is claimed in [3] that the key can be reconstructed in about 12 seconds on the same CPU by an algebraic attack. More detailed descriptions of the attacks are given in Appendix A.

5 Attacks on Genuine Tag

The easiest attacks to implement are the tag-only or card-only attacks in AttAT scenario, with a fake reader having access to a targeted genuine tag. The fake reader forces the tag into multiple fake partial authentication sessions, called queries, in which it cannot be successfully authenticated to the tag. The objective is to reconstruct the key stored in the tag by using the transcripts of the partial authentication sessions and the known structure of CRYPTO1.

It would be impossible to obtain needed portions of known keystream from the tag, if it were not for a peculiar property of the authentication protocol, which can rightfully be called a bug or even a deliberately inserted weakness [7,4]. Namely, in the protocol, upon receiving a tag nonce in the clear, the fake reader sends two 32-bit ciphertexts, one standing for the encrypted reader nonce, $\{n_R\} = n_R \oplus ks_1$, and the other for the encrypted answer to the tag nonce, $\{a_R\} = a_R \oplus ks_2$. The fake reader also sends 8 encrypted parity bits $\{p\}$, corresponding to the 8 bytes sent. The attacker produces all the ciphertext directly, in a random or chosen manner depending on the attack to be conducted, without knowing the secret key. The reader's answer, n_R, decrypted by the tag will be wrong and the authentication will fail. The bug is that in this case, if all 8 decrypted parity bits in p happen to be correct, then the tag sends back to the reader a 4-bit ciphertext of a fixed 4-bit error message, encrypted with the first four bits of ks_3. This is a serious weakness, because it reveals to the attacker that the 8 parity bits are all correct and discloses 4 keystream bits. Moreover, the correct parity bits disclose to the attacker 8 independent linear combinations of keystream bits, which are here referred to as the keystream parity bits. Altogether, in the case of a successful query, the attacker thus gets 12 bits of information or entropy regarding the key, in the form of 4 keystream bits and 8 keystream parity bits. This can then be used for mounting key reconstruction attacks in the tag-only scenario, which are described in the sequel. Surprisingly, according to [4], unlicensed clone MIFARE Classic cards used in some countries always send out the encrypted error message, regardless of the values of the parity bits. This then greatly facilitates the attacks.

In the on-line stage of the attacks, the fake reader makes a number of queries to the genuine tag, in order to achieve a sufficient number of successful queries (with all 8 parity bits correct). The queries can use random or fixed tag nonces, where the latter, realized by the reset query technique, take about 50 times more time than the former. It is thus claimed in [7] that the attacker can perform about 1500 queries per second with a random tag nonce and about 30 queries per second with a fixed tag nonce realized by the reset query technique. In the off-line stage of the attacks, the collected keystream data is analyzed in order to reconstruct the key. The on-line stage is thus a practical bottleneck of the attacks and the required numbers of queries of one or the other type, together with some auxiliary, simple computations, determine the on-line complexity of the attacks. In addition, the attacks may also require significant precomputation time and storage in the setup stage.

For random tag nonces, the keystream is also random and cannot be controlled by the fake reader. Therefore, the best strategy for the fake reader to get a successful query is to choose randomly $\{n_R\}$, $\{a_R\}$, and $\{p\}$ until a successful query occurs, i.e., until all 8 parity bits in p are correct. On average, this requires 256 queries with random n_T. For fixed tag nonces, if $\{n_R\}$ is kept fixed by the fake reader, then the keystream and n_R will also be fixed. Further, if $\{a_R\}$ is also kept fixed, then a_R will be fixed and, hence, all 8 parity bits in p will be fixed too. To get a successful query, the best strategy for the fake reader is then to choose different instead of random values of $\{p\}$. The average number of queries is thus reduced to 128=256/2.

Each of the three attacks proposed in [7] has serious practical limitations, as shown in Table 1. Namely, the first attack has huge off-line time, the second attack has very large on-line time, and the third attack has huge precomputation time. Concise descriptions of the attacks, denoted as Attack 1, 2, and 3, are given in Appendix B, where it is shown that by using the queries with random tag nonces to obtain different tag nonces, the on-line time of Attack 3 can be reduced from about 2 minutes to about 7 seconds. We now concentrate on the fourth attack, proposed in [4] and denoted here as Attack 4. It overcomes all the limitations of the three attacks from [7]. In spirit, this attack is similar to Attack 2 from [7], but takes better advantage of differential properties of the nonlinear filter function f applied to the LFSR sequence. As a consequence, it requires a significantly smaller number of queries in the on-line stage and has a significantly smaller off-line time complexity. However, it is shown below that the analysis of the attack given in [4] is incorrect and that the attack can be simply improved by better usage of the queries with random and fixed tag nonces. As a result, the optimized attack, denoted as Attack 4* in Table 1, has a better performance.

The main idea of the differential attack [4] is for the fake reader to first get one successful query for some $\{n_T\}$, $\{n_R\}$, $\{a_R\}$, and $\{p\}$. Then, the fake reader performs a number of modifications of $\{n_R\}$ and, for each modification, performs further queries with fixed $\{n_T\}$, $\{n_R\}$, and $\{a_R\}$ and different $\{p\}$ in order to get a new successful query. The 32-bit encrypted reader nonce $\{n_R\}$ is modified by changing its last 3 bits and then fixed. For each of 7 possible changes, $\{a_R\}$ is randomly chosen and then fixed, and only the last 5 bits of $\{p\}$, which are (randomly) affected by the change of $\{n_R\}$ and $\{a_R\}$, are varied. On average, 16=32/2 such queries with a fixed nonce tag are needed for a successful query to occur. As a result of the on-line stage, the attacker thus obtains 8 successful queries yielding the known keystream data. The problem is that this keystream depends on unknown values of n_R. This can be overcome with a high probability, by using differential properties of f when shifted along the LFSR sequence.

Namely, it is claimed in [4] that with probability about 0.75, the 3 keystream bits used for the decryption of the last 3 bits of $\{n_R\}$ are independent of these 3 bits. As a consequence, each nonzero 3-bit change δ_3 of $\{n_R\}$ will result in the same change of the last 3 bits of n_R itself. Since the LFSR sequence depends linearly on n_R, this implies that for each value of δ_3, the subsequent LFSR sequence will change linearly in a way that depends only on this value. This means that it can be expressed as the bitwise XOR of the LFSR sequence corresponding to $\{n_R\}$ and a binary sequence that depends only on the known value of δ_3. For each of 8 values of δ_3, including the all-zero value, the attacker can use the 4 keystream bits resulting from the corresponding successful query. Then, in the off-line stage, an adapted variant [4] of the well-known resynchronization attack [5,11], where the IV corresponds to δ_3, can be used to obtain about 2^{16}

candidates for the LFSR state at the time when the last of the 4 keystream bits is generated. The relation with the resynchronization attack is not noticed in [4]. A detailed description is given in Appendix C. The 2^{16} candidate keys resulting from the LFSR rollback are then tested on other keystream data already collected in the on-line stage (i.e., 64 keystream parity bits), to produce the correct key. The total off-line time complexity of about 2^{16} steps takes practically zero time on a standard CPU. According to [4], the attack succeeds with probability about 0.75. In order to reconstruct the key, both on-line and off-line stages of the attack need to be repeated about $4/3 \approx 1.33$ times on average.

Our criticism of the differential attack [4] concerns the probability of the exploited differential property of f and the way the queries with random or fixed tag nonces are performed in the on-line stage.

Consider a general case where the last m, $m \leq 32$, bits of $\{n_R\}$ are changed. Then the last m bits of n_R obtained by the bitwise decryption of $\{n_R\}$ will change in the same way if the corresponding m keystream bits used for the decryption do not change. The fact overlooked in [4] is that the first (leftmost) of these m keystream bits does not change necessarily, because it depends only on the previous bits of $\{n_R\}$, which are not changed. The remaining $m-1$ keystream bits are generated as $m-1$ successive output bits of the filter function shifted along the LFSR sequence, i.e., by the $(m-1)$-bit augmented filter function of all the bits contained in $m-1$ (overlapping) successive LFSR states, which form the input to the augmented filter function. Therefore, any change of the last m bits of $\{n_R\}$ will result in the same change of the last m bits of n_R if and only if the $(m-1)$-bit output of the $(m-1)$-bit augmented filter function is independent of the last $m-1$ input bits. Let π_{m-1} denote this probability, over the uniformly distributed inputs. In the attack [4], $m = 3$ and the relevant probability is then π_2. In [7], it is proved that $\pi_1 = 29/32 \approx 0.906$, where the 1-bit augmented filter function is f expressed as a function of all 48 LFSR state bits. Since the input bits to f that are effectively used for generating two successive output bits are distinct, it follows that $\pi_2 = (29/32)^2 \approx 0.821$. This is the correct probability for the differential attack [4], not 0.75.

It is interesting to note that $\pi_m = 0$ would hold for all m if f were linear in the first or the last input variable, i.e., it f satisfied the sufficient condition [8] for pure randomness of the output sequence. In other words, if f had satisfied this condition, then the considered differential attack would have been impossible. On the other hand, $\pi_m = 1$ would hold if f did not effectively depend on the last m LFSR state bits. It is fair to say that in CRYPTO1, f effectively depends on the last LFSR state bit and, hence, $\pi_m < 1$ holds for all m.

In the first phase of the on-line stage of the attack, to get one successful query, the fake reader can perform queries with a fixed or random tag nonce, where, on average, 128 queries are required in the former case and 256 in the latter. However, as mentioned above, the difference in the timings is significant. If q_r and q_f denote the timings of the queries using random and fixed tag nonces, respectively, then, according to [7], $q_f \approx 50q_r$, $1500q_r \approx 1\,\mathrm{sec}$, and $30q_f \approx 1\,\mathrm{sec}$. Accordingly, $256q_r \approx 5.12q_f \ll 128q_f$. Surprisingly, in [4], it is proposed to use queries with a fixed tag nonce. Moreover, if the attack fails in the first run, then, in order to reduce the number of queries in repeated runs, it is suggested in [4] to keep n_T and the first two bytes of $\{n_R\}$ the same as in the first run, which implies that the first two bits of $\{p\}$ will also be the same. In this case, on average, 32=64/2 instead of 128 queries with a fixed tag nonce are required in the repeated runs. The average on-line time of the attack is thus estimated to be about $256q_f$ in the first run and $160q_f$ in the repeated runs. Since the off-line time is negligible, the average time required for the success is estimated to be $(128 + 32/3 + (4/3)(8 \cdot 16))q_f \approx 310q_f \approx 10.33\,\mathrm{sec}$.

It follows that this differential attack can be simply optimized by using the queries with a random tag nonce in the first phase of all the runs. Also, in other phases of the attack, further 16 queries with a fixed tag nonce need to be performed not 8 times, as proposed in [4], but only 7 times, for each nonzero 3-bit modification δ_3 of $\{n_R\}$. The average on-line time of each run then becomes $256q_r + (7 \cdot 16)q_f \approx 117.12q_f \approx 3.9\,\mathrm{sec}$. The average time required for the success of the optimized attack Attack 4* is then about $117.12q_f/\pi_2 \approx 4.75\,\mathrm{sec}$, since the attack needs to be repeated $1/\pi_2$ times on average.

We now propose a novel key reconstruction attack, denoted as Attack 5 in Table 1. It is in spirit similar to the differential Attack 4*, but requires a considerably smaller number of queries in the on-line stage, which is a bottleneck of tag-only attacks, and has a considerably higher probability of success. This is achieved at the cost of increasing the off-line time complexity, which nevertheless remains practically low. Our main insight making the tradeoff possible is that m can be reduced from 3 to 2 bits, according to the general considerations given above. The success probability then increases from $\pi_2 \approx 0.821$ to $\pi_1 \approx 0.906$.

The first phase of the on-line stage of the attack is the same as in Attack 4*. The fake reader on average makes 256 queries with random $\{n_T\}$, $\{n_R\}$, $\{a_R\}$, and $\{p\}$, in order to get one successful query. The fake reader then proceeds by making new queries using fixed n_T, modified and then fixed $\{n_R\}$, randomly chosen and then fixed $\{a_R\}$, and different $\{p\}$, where the first 3 bits of $\{p\}$ are kept the same and only the last 5 bits are varied. The modification of $\{n_R\}$ consists in changing its last $m = 2$ bits. On average, 16=32/2 such queries with a fixed n_T are needed for a successful query to occur. Since this has to be repeated 3 times, for all 3 nonzero 2-bit changes δ_2 of $\{n_R\}$, a total of $48 = 3 \cdot 16$ queries with a fixed n_T are needed on average. The average on-line time of the attack is then $256q_r + (3 \cdot 16)q_f \approx 53.12q_f \approx 1.77\,\mathrm{sec} \approx 1.8\,\mathrm{sec}$, which is more than 2 times smaller than in Attack 4*. Each of the 4 successful queries achieved in the on-line stage provides 12 bits of information about the 48-bit key in the form of 4 keystream bits and 8 keystream parity bits, to be used in the off-line stage.

With probability $\pi_1 \approx 0.906$, each nonzero 2-bit change δ_2 of $\{n_R\}$ will result in the same change of the last 2 bits of n_R. This means that the subsequent LFSR sequence can then be expressed as the bitwise XOR of the LFSR sequence corresponding to $\{n_R\}$ and a binary sequence that depends only on the known value of δ_2. Then, in the off-line stage, about 2^{32} candidates for the LFSR state at the time when the last of the 4 keystream bits resulting from each successful query is generated can be obtained by a variant of the resynchronization attack [5,11], where the IV corresponds to δ_2. A detailed description is given in Appendix C. The 2^{32} candidate keys resulting from the LFSR rollback are then tested on the $32 = 4 \cdot 8$ keystream parity bits collected in the on-line stage, to produce the correct key or a very small number of candidates. The total off-line time complexity of about 2^{32} steps takes about 5 minutes on a standard CPU. The attack thus succeeds with probability about $\pi_1 \approx 0.906$.

For reducing the number of candidate keys to only 1, the attacker may also make a small number of additional queries in the on-line stage in order to collect more bits of information about the key. For example, to obtain additional 12 bits of information, the attacker needs another successful query, which may be achieved in the random n_T scenario with an average of 256 additional queries. This effectively increases the on-line time to $58.24q_f \approx 1.94$ sec and is still about 2 times smaller than in Attack 4*. Alternatively, the attacker can find the unique key at any later time in AttAR scenario, by accessing a genuine reader and then testing the candidate keys on the keystream segment ks_2 reconstructed from n_T and $\{a_R\}$. More precisely, for each assumed key, the attacker generates a 32-bit keystream segment by using n_T and $\{n_R\}$ and then compares it with ks_2.

If the attack fails, then the strategy of repeating the whole attack (on-line and off-line stages) as many times as needed until the key is reconstructed may not be practical, due to the off-line time of about 5 minutes. In any case, the attack would need to be repeated only about $\pi_1 \approx 1.1$ times on average. It is hence preferable for the attacker to decide in advance on the success probability, repeat the on-line stage a number of times that guarantees this probability, and only then perform the off-line stage using all the collected data. To obtain the success probability of at least 0.99, the on-line stage needs to be performed only twice, which takes about 3.5 seconds. In this case, additional successful queries are not needed for testing the candidate keys, because the on-line stage performed twice already provides the sufficient keystream. The off-line stage takes about 5 minutes with probability 0.906 or about 10 minutes otherwise, which is still about 5 minutes on average and thus remains very practical. For comparison, to achieve the success probability of at least 0.99, Attack 4* needs to be repeated three times, which takes about 11.7 seconds, whereas the off-line stage can be done practically instantly. The on-line stage in the new Attack 5, with $m = 2$, is thus more than 3 times faster than in the optimized Attack 4*, with $m = 3$.

Properties of the five presented tag-only attacks are summarized in Table 1. Times are given both in appropriate steps (clear from the context) and in time units, where off-line times relate to a standard CPU. TLU denotes a table lookup operation and f_{ev} denotes one evaluation of f.

6 Multiple Sector Attacks

The cryptanalytic attacks described in previous sections relate to the mutual authentication protocol for one sector. In this case, the tag nonce n_T is sent

Table 1. Summary of tag-only attacks

	Attack 1 [7]	Attack 2 [7]	Attack 3 [7]	Attack 4* [4]	**Attack 5** this paper
Setup time	0	0	2^{48}	0	0
Setup memory	0	0	$48 \cdot 2^{36}$ bit 384 GByte	0	0
On-line time	$1280q_r$ 1 sec	$28500q_f$ 15 min	$4230q_r + 128q_f$ 7 sec	$3(256q_r + 112q_f)$ 11.7 sec	$2(256q_r + 48q_f)$ 3.5 sec
Off-line time	$5 \cdot 2^{48}$ 3 year	$2^{32.8}$ 10 min	2^{24} TLU 20 sec	$2^{16} + 2^{26}f_{ev}$ ≈ 0	$2^{32} + 2^{25}f_{ev}$ 5 min
Off-line memory	≈ 0	≈ 0	≈ 0	168 Byte	42 KByte
Success rate	100%	100%	100%	99.4%	99.1%

in the clear form, which can be used for known keystream attacks (except in the tag-only scenario). As explained in Section 2, for authenticating access to multiple sectors of the tag in the same session, the same protocol is repeated with a difference that, for each new sector, the new authentication command (Block) is encrypted with the previous sector key and the new n_T is encrypted with the preceding keystream segment ks_0 generated from the new key. Therefore, n_T has to be guessed by the attacker.

Assume that the key for the first sector has been recovered by the attacker by one of the attacks described in Sessions 4 and 5, in AttP, AttAR, or AttAT scenario. The objective of the attacker is then to reconstruct the secret keys for other sectors. To this end, it is required to gather data from an authentication session for multiple sectors. This is possible in AttP and AttAT scenarios. AttAR scenario is not useful for the attacker, because a fake tag cannot encrypt a new n_T with the correct key for the next sector and the attacker cannot then obtain any correct keystream segment generated from the key for the next sector.

In AttP scenario [6], the attacker intercepts and records a genuine authentication session for multiple sectors, between a genuine tag and a genuine reader. Since the key for the first sector is already reconstructed, the attacker determines the next sector by decrypting the new authentication command. In AttAT scenario [7], a fake reader initiates a fake authentication session for multiple sectors with a genuine tag and repeatedly uses the previously reconstructed key for the first sector. The attacker is then successfully authenticated for the first sector and then proceeds with the authentication for the next sector. This is achieved by sending the authentication command (Block) encrypted with the key for the first sector, which is then successfully decrypted by the tag. In either case, the attacker obtains a correct keystream segment generated from the key for the next sector, on the condition that n_T can be effectively guessed. In AttAT scenario, the keystream segment length is 32 bits, from $n_T \oplus ks_0$, while in AttP scenario, the keystream segment length is 96 bits, from $n_T \oplus ks_0$, $a_R \oplus ks_2$, and $a_T \oplus ks_3$. The key for the next sector can then be easily reconstructed by using the techniques described in Section 4. The keys of other sectors can be reconstructed analogously, by proceeding one sector at a time. In [6], this attack is called the nested authentication attack.

The tag nonce can be guessed easily not only because it contains at most 16 bits of entropy and its effective entropy depends only on the imprecision of timing, but also because of the way the parity bits are encrypted, as pointed out in Section 2. Namely, any pair of ciphertext bits, one bit corresponding to the parity bit of the current byte and the other to the first bit of the next byte, reveals a linear relation among the plaintext bits due to the fact that the two keystream bits are the same. Each such relation reduces the plaintext uncertainty by 1 bit. More details can be found in Appendix D.

7 Conclusions

Attacks on the MIFARE Classic protocol are made possible by repeatable and predictable tag nonces, the weak structure of the nonlinear filter generator in CRYPTO1, the way the parity bits for error detection are genarated, and the fact that the tag can sent out an encrypted response even if the authentication of the reader fails. It is shown that the TMDT attack [6] in AttAR scenario can be regarded as an adaptation of the generic TMDT attack [1,9], whereas the inversion attack [6] in AttP or AttAR scenario can be regarded as an adaptation of the inversion attack with the decimation technique proposed in [8,10] for attacking nonlinear filter generators. The easiest attacks to implement are the tag-only attacks, in AttAT scenario, where a fake, emulated reader has a wireless access to a genuine tag in the on-line stage of the attack. Their main limitation factor in practice is the on-line time required. It is pointed out that each of the three attacks from [7] has serious practical limitations, namely, Attack 1 has huge off-line time, Attack 2 has very large on-line time of about 15 min, and Attack 3 has huge precomputation time. It is shown that by using the queries with random tag nonces to obtain different tag nonces, the on-line time of Attack 3 can be reduced from about 2 minutes to about 7 seconds.

The best known attack in the tag-only scenario is the differential attack [4], Attack 4, which is claimed to take about 10 seconds of average on-line time in order to reconstruct the secret key for one sector of the card. A correct analysis of this attack demonstrates that it can be optimized into Attack 4*, to reduce the average on-line time to about 4.75 seconds. On the basis of the conducted analysis, a new attack of a similar, differential type, denoted as Attack 5, is also proposed. It achieves a success probability of about 0.906 with the on-line time of about 1.8 seconds, whereas the optimized differential attack, Attack 4*, has the success probability of about 0.821 with the on-line time of about 3.9 seconds. The success probability and the on-line time of two independent runs of Attack 5 are about 0.991 and 3.5 seconds, respectively. For three independent runs of Attack 4*, they are about 0.994 and 11.7 seconds, respectively. This significant improvement is achieved at the cost of increasing the off-line time to about 5 minutes, which still remains very practical. It is explained that the off-line stage of both differential attacks can be regarded as an adaptation of the resynchronization attack [5,11].

As the worldwide MIFARE Classic infrastructure cannot be changed easily, the most effective countermeasure [4] against the tag-only attacks is putting the cards in electromagnetic-shield covers.

References

1. Babbage, S.: A space/time tradeoff in exhausting search attacks on stream ciphers. In: Proc. European Convention on Security and Detection, IEE Conference Publication No. 408, pp. 161–166 (May 1995)
2. Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)
3. Courtois, N.T., Nohl, K., O'Neil, S.: Algebraic attacks on the Crypto-1 stream cipher in MiFare Classic and Oyster cards. Cryptology ePrint Archive, Report 2008/166 (2008)
4. Courtois, N.T.: The darkside of security by obscurity - and cloning MiFare Classic rail and building passes, anywhere, anytime. In: Proc. Secrypt 2009, pp. 331–338 (2009)
5. Daemen, J., Govaerts, R., Vandewalle, J.: Resynchronization Weaknesses in Synchronous Stream Ciphers. In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 159–167. Springer, Heidelberg (1994)
6. Garcia, F.D., de Koning Gans, G., Muijrers, R., van Rossum, P., Verdult, R., Wichers Schreur, R., Jacobs, B.: Dismantling MIFARE Classic. In: Jajodia, S., Lopez, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 97–114. Springer, Heidelberg (2008)
7. Garcia, F.D., van Rossum, P., Verdult, R., Wichers Schreur, R.: Wirelessly pickpocketing a Mifare Classic card. In: Proc. 30th IEEE Symposium on Security and Privacy, Oakland, pp. 3–15 (2009)
8. Golić, J.Dj.: On the Security of Nonlinear Filter Generators. In: Gollmann, D. (ed.) FSE 1996. LNCS, vol. 1039, pp. 173–188. Springer, Heidelberg (1996)
9. Golić, J.Dj.: Cryptanalysis of Alleged A5 Stream Cipher. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 239–255. Springer, Heidelberg (1997)
10. Golić, J.Dj., Clark, A., Dawson, E.: Generalized inversion attack on nonlinear filter generators. IEEE Trans. Comput. C-49, 1100–1109 (2000)
11. Golić, J.Dj., Morgari, G.: On the Resynchronization Attack. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 100–110. Springer, Heidelberg (2003)
12. de Koning Gans, G., Hoepman, J.-H., Garcia, F.D.: A Practical Attack on the MIFARE Classic. In: Grimaud, G., Standaert, F.-X. (eds.) CARDIS 2008. LNCS, vol. 5189, pp. 267–282. Springer, Heidelberg (2008)
13. Kumar, S., Paar, C., Pelzl, J., Pfeiffer, G., Schimmler, M.: Breaking Ciphers with COPACOBANA –A Cost-Optimized Parallel Code Breaker. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 101–118. Springer, Heidelberg (2006)
14. Nohl, K., Evans, D., Starbug, Plötz, H.: Reverse-engineering a cryptographic RFID tag. In: Proc. USENIX Security 2008, pp. 185–193 (2008)
15. Proxmark III instrument, HW and SW, `http://cq.cx/proxmark3.pl`

A TMDT Attack and Adapted Inversion Attack from [6]

TMDT attack uses a precomputed table with 2^{48-x} entries (S_{64}, ks_2, ks_3), sorted by (ks_2, ks_3), and 2^x keystream segments, either (ks_2, ks_3) or only ks_2, obtained from 2^x (partial) authentication sessions with the genuine reader. For $x = 12$, the precomputation time and memory complexities are thus both proportional to 2^{36}. In the on-line stage, 4096 authentication sessions are needed for collecting the keystream data, which takes about 10 min. In the off-line stage, the key can be reconstructed by 4096 table lookup operations, on a hard disk, and by the LFSR rollback, i.e., almost instantly, provided that 64-bit keystream segments are used. With 32-bit keystream segments, the 4096 table lookups will, on average, yield 2^{16} $(2^{48}/2^{32})$ candidate states S_{64}, instead of only one. To get the correct key from the resulting 2^{16} candidate keys, one more authentication session is needed. We note that the required number of authentication sessions can be reduced at the expense of increasing the precomputation time, while keeping the same computation time and memory, by applying the TMDT attack [2].

Let M be the input memory size of a nonlinear filter generator defined as the difference between the last and the first tap position to the filter function (in CRYPTO1, $M = 38$). Then the time complexity of the generic inversion attack is 2^M steps. Further, let d be the greatest common divisor of the pairwise differences between the tap positions (in CRYPTO1, $d = 2$). Then the input memory size can be reduced d times by looking at d-decimations of d successive phase shifts of the LFSR sequence.

In order to deal with short keystream segments corresponding to decimated sequences, which are shorter than the LFSR length, one has to consider different phase shifts of the LFSR sequence simultaneously. This can be achieved by the technique [6] based on partitioning the LFSR recursion into $d = 2$ linear terms corresponding to 2-decimations of two successive phase shifts of the LFSR sequence. The values of these terms are computed along the decimated LFSR segments obtained by the inversion attack and then stored in two sorted tables, each containing about 2^{19} entries. The matches between the entries in the two tables then yield the reconstruced LFSR segments by interleaving. For 64-bit keystream segments, the time complexity is thus at most 2^{17} steps. For 32-bit keystream segments, the time complexity is at most 2^{20} steps and the number of candidate keys obtained is about 2^{16}, so that one more authentication session is needed to get the correct key. Altogether, it takes about 0.05 seconds and 8 MByte of memory on a standard CPU to recover the key.

B Three Tag-Only Attacks from [7]

Attack 1 is a brute-force attack that exhaustively tests all 2^{48} keys on the keystream data obtained from 4-5 successful queries with a random tag nonce, which provide 48-60 bits of entropy for the 48-bit key. To get one successful query, it is on average needed to perform 256 queries with a random nonce tag. The on-line stage thus takes less than 1 second to perform about 1280 queries with a random tag nonce.

Attack 2 uses a relatively large number of queries with a fixed tag nonce, i.e., on average, about 28500 such queries. The on-line stage thus takes about 15 minutes. The objective is to find a special value of the 32-bit encrypted reader nonce by adaptively choosing its values for a given value of the tag nonce. This is achieved by an iterative process with backtracking, each time changing the last bit of one byte of the encrypted reader nonce and checking a condition on the corresponding encrypted correct parity bits, upon obtaining the respective two successful queries by repeatedly changing the encrypted parity bits. The special value of the encrypted reader nonce, satisfying the condition on all four bytes, significantly reduces the number of possible LFSR states at a given time and thus reduces the number of candidate keys to about $2^{32.8}$. They are then checked in the off-line stage by using the keystream data already collected in the on-line stage, to find the correct key. The off-line stage approximately requires 10 minutes on a standard CPU.

Attack 3 uses a large partitioned table of about 384 GByte, stored on a hard disk, which is precomputed in time equivalent to that of the brute-force attack. It stores about 2^{36} LFSR states at a given time that result in the all-zero 32-bit encrypted reader nonce, all-zero 32-bit encrypted answer to the tag nonce, all 8 encrypted parity bits equal to zero, as well as 4 keystream bits encrypting the error message, all equal to zero. In the on-line stage, the attacker should make about 4096 queries with different tag nonces in order to satisfy the conditions for the LFSR state to belong to the precomputed table of 2^{36} entries. In [7], it is claimed that this requires about 2 minutes of on-line time, which is true if these queries use a fixed tag nonce. However, in view of the fact that the tag nonce can take 2^{16} different values, it follows that only 4230 queries with random tag nonces are on average sufficient to get 4096 different tag nonces. In addition, the attacker then makes about 128 queries with a fixed tag nonce in order to further restrict the LFSR state to a subtable of 2^{24} entries. The on-line stage can thus take only about 7 seconds, not 2 minutes as claimed in [7]. In the off-line stage, the resulting 2^{24} candidate keys are then tested on the keystream data already collected in the on-line stage, to produce the correct key. The off-line complexity of the attack is thus determined by 2^{24} table lookup operations, which takes less than about 20 seconds on a standard CPU.

C Adapted Resynchronization Attacks

The resynchronization attack [5,11] is applicable to stream ciphers with a linear next-state function, a nonlinear Boolean output function f depending on a relatively small or moderately large number of bits, n, and a linear reinitialization algorithm combining a k-bit secret key with an IV (initialization vector). Due to the linearity, the input to the output function at any given time can be linearly decomposed into two components, one depending on the key and the other depending on the IV. The attack then essentially consists in solving the corresponding equations of the form $z_t^i = f(X_t \oplus C_t^i)$, for a number of time instants t, where X_t is a linear function of the key and C_t^i is a linear function of the IV^i.

For each t, the solution is found by the exhaustive search. Accordingly, the key can be reconstructed from about n different IVs, by observing the outputs of f at about k/n time instants, and by evaluating f about $k2^n$ times.

Consider first the off-line stage of the differential Attack 4 [4] or Attack 4*. The 4 keystream bits resulting from each of 8 successful queries are the first 4 keystream bits of ks_3, i.e., z_i, $96 \leq i \leq 99$. Accordingly, for each such keystream bit z_i, the following 8 equations hold simultaneously with probability π_2: $z_i(\delta_3) = f(S_i \oplus \Delta_{3,i})$, where δ_3 is varied over all 8 possible values, $\Delta_{3,i}$ is a 48-bit vector depending only on i and δ_3 ($\Delta_{3,i} = 0$ if $\delta_3 = 0$), and S_i is the LFSR state at time i for $\delta_3 = 0$, for given n_T and $\{n_R\}$. Due to the fact [4] that f depends only on 20 bits of the state and that after 2 steps, f depends on the same 19 bits and 1 new bit, z_{96} and z_{98} (as well as z_{97} and z_{99}) depend on 21 LFSR bits and we have 16=8+8 nonlinear equations involving these 21 bits. As in the resynhcronization attack, by the exhaustive search over these 21 bits, each time evaluating f 16 times, the attacker finds and stores all 21-bit inputs that are consistent with these 16 equations. On average, the number of possible values for the 21 bits is thus $2^5 = 2^{21-16}$. In the same way, the attacker gets and stores about 2^5 possibilities for the other 21 LFSR sequence bits determining z_{97} and z_{99}. The two tables of 2^5 entries are thus computed in 2^{22} steps, each consisting of 16 evaluations of f, that is, almost instantly on a standard CPU.

Altogether, the attacker thus obtains about 2^{10} possibilities for the corresponding 42 successive bits of the LFSR sequence. By guessing the remaining 6 bits, the attacker thus obtains 2^{16} possibilities for the LFSR state S_{99}. By exhaustively examining all 2^{16} found states S_{99}, for given n_T and $\{n_R\}$, 2^{16} candidate keys are thus recovered via the LFSR rollback. The 2^{16} candidate keys are then tested on $64 = 8 \cdot 8$ keystream parity bits already collected in the on-line stage of the attack to yield the correct key. In fact, only 24 keystream parity bits obtained from 3 successful queries are sufficient. This can be done almost instantly on a standard CPU. The total off-line time of the attack is thus practically zero. The attack succeeds with probability $\pi_2 \approx 0.821$, and fails if no candidate key survives the testing.

Consider now the off-line stage of the new differential Attack 5. For each such keystream bit z_i, $96 \leq i \leq 99$, the following 4 equations hold simultaneously with probability π_1: $z_i(\delta_2) = f(S_i \oplus \Delta_{2,i})$, where δ_2 is varied over all 4 possible values, $\Delta_{2,i}$ is a 48-bit vector depending only on i and δ_2 ($\Delta_{2,i} = 0$ if $\delta_2 = 0$), and S_i is the LFSR state at time i for $\delta_2 = 0$, for given n_T and $\{n_R\}$. Now, similarly as in Attack 4, since there are 4 instead of 8 linear equations for each of these 4 keystream bits, the attacker then finds and stores $2^{13} = 2^{21-8}$ possibilities for each of the two interleaved 21-bit parts of the state S_{99}, in 2^{22} steps, each step consisting of 8 evaluations of f. The total memory required is then 42 KByte. The off-line time of this part of the attack, requiring 2^{25} evaluations of f, is practically zero. The attacker then exhaustively examines $2^{32} = 2^{26+6}$ possibilities for the LFSR state S_{99}, for given n_T and $\{n_R\}$, by recovering 2^{32} candidate keys via the LFSR rollback. The 2^{32} candidate keys are then tested on $32 = 4 \cdot 8$ keystream parity bits already collected in the on-line stage of the attack. This should suffice to reduce the number of candidate keys to only 1 or a very small number. The required 2^{32} steps of the off-line stage can be performed in about 5 minutes or less on a standard CPU. The attack succeeds with probability $\pi_1 \approx 0.906$, and fails if no candidate key survives the testing.

D Nested Authentication Attacks from [6,7]

It is here explained how the attacker can effectively guess a correct value of n_T used in the authentication protocol for the next sector, in the nested authentication attacks [6,7], where [6] deals with AttP scenario and [7] with AttAT scenario. There are only 2^{16} values of n_T to start with, due to the 16-bit LFSR used for generating 32-bit tag nonces. In fact, as pointed out in [6], the uncertainty is much smaller, because n_T is generated almost deterministically from the previously reconstructed value of the tag nonce, where the only residual uncertainty relates to the imprecision of timing. Another weakness which further reduces the uncertainty is that any pair of ciphertext bits encrypting the parity bit of the current byte and the first bit of the next byte reveals a linear relation among the plaintext bits due to the fact that the two keystream bits are the same. Each such relation reduces the plaintext uncertainty by 1 bit. Therefore, the initial uncertainty of n_T given $n_T \oplus ks_0$ is 13 (16-3) bits.

In AttP scenario, the uncertainty of n_T, given $n_T \oplus ks_0$, $a_R \oplus ks_2$, and $a_T \oplus ks_3$ is only 6 (16-10) bits, so that the attacker needs to check at most 64 values of n_T. This number is effectively much smaller in practice, as pointed out above. For any guessed value of n_T, the key for the next sector can be reconstructed from the corresponding 96-bit keystream segment (ks_0, ks_2, ks_3) by the techniques from Section 4, provided that the guess is correct. If the guess is incorrect, no 48-bit key can be reconstructed, because the corresponding 96-bit keystream segment will be incorrect. The keys of other sectors can be reconstructed analogously from already recovered keys of the previous sectors.

In AttAT scenario, as pointed out in [7], the attack proceeds along similar lines, with a difference that only the encryption $n_T \oplus ks_0$ can be used. This implies that at most 2^{13} values of n_T need to be checked, which may be significantly reduced by using the measured timing between two successive authentication rounds. For each guessed value of n_T, about 2^{16} candidate keys for the next sector can be reconstructed from the corresponding 32-bit keystream segment ks_0 by the techniques described in Section 4. More precisely, the techniques need to be slightly adapted to the fact that during the generation of ks_0, $n_T \oplus ID$ is shifted in the LFSR. The correct key for the next sector is then found by using another fake authentication session for multiple sectors initiated by the fake reader (or at most two such sessions), by computing the intersections between the obtained sets of candidate keys. With a high probability, only the correct guesses of the tag nonces will result in a unique value of the key, whereas for incorrect guesses, the intersections will be empty. The keys of other sectors can be reconstructed analogously. This means that in AttAT scenario, the keys of other sectors can be reconstructed much easier than for the first sector.