A practical attack on *patched* MIFARE Classic

Abstract

MIFARE Classic is the world's most widely deployed RFID (radio-frequency identification) technology, and it is supposed to be protected by the proprietary Crypto-1 stream cipher. However, it proved inadequate after weaknesses in the design and implementation of Crypto-1 and MIFARE Classic started surfacing since late 2007 [6, 7, 11–15].

Some users reacted by upgrading to more secure alternatives such as MIFARE DESFire. However, many (especially in Asia) opted to "patch" MIFARE Classic instead. Their reasoning might have gone as follows:

"The most serious threat comes from efficient card-only attacks, where the attacker only needs an off-the-shelf reader and a PC to tamper a target tag. All efficient card-only attacks depend on certain implementation flaws. Ergo, if we just fix these flaws, we stop the most serious attacks without an expensive infrastructure upgrade."

One such prominent case is the "EasyCard 2.0," today accepted in Taiwan as a means of electronic payment not only in public transportation but also in convenient stores, drug stores, restaurants, cafes, supermarkets, book stores, movie theaters, etc.

We believe that the whole "patching" approach is questionable because Crypto-1 is fundamentally still a weak cipher. In support of our proposition, we present a new card-only attack based on state-of-the-art algebraic differential cryptanalytic techniques [1]. Still using the same cheap reader as previous attacks, it takes 2–15 minutes of computation on a PC to recover a secret key of EasyCard 2.0 after 10–20 hours of data collection. We hope the new attack makes our point sufficiently clear, and urge all MIFARE Classic users with important transactions — such as electronic payment — to upgrade to more secure alternatives soon.

1 Introduction

MIFARE Classic, a brand owned by the NXP Semiconductors, is the most widely used RFID technology in today's world, with billions of chips sold worldwide. It is used in many public-transportation ticketing systems in, e.g., Beijing, Chongqing, Guangzhou, Boston, the Netherlands, London, Seoul, Taipei, etc. In recent years, it has even found its way into electronic payment systems in several Asian countries including China and Taiwan.

The proprietary Crypto-1 stream cipher is designed to provide cryptographic protection to MIFARE Classic. NXP Semiconductors has never made public the detailed algorithm of Crypto-1. Nevertheless, starting from late 2007 in a series of papers, the specifications and several weaknesses of the cipher have been found via reverse engineering and cryptanalysis [6, 7, 11–15]. As Courtois *et al* concluded: "The security of this cipher is therefore close to zero" [7]. Users of MIFARE Classic around the world responded differently to this incident. Some kept silent, while others promptly announced plans to migrate to more secure technologies such as MIFARE DESFire.

In this paper, we shall investigate in detail one such replacement being deployed in Taipei, an early adopter and aggressive user of MIFARE Classic. Branded under the name "EasyCard," more than 35 million cards have been issued in Taipei since the official release in 2002, with more than 4.6 million transactions per day in 2012. Starting from 2010, the card is also accepted as a means of electronic payment by almost all convenient store chains, as well as drug stores, eateries, cafes, supermarkets, book stores, movie theaters, etc. Similar use of MIFARE Classic is reported in several cities in China including Beijing, Chongqing, and Guangzhou.

In a nutshell, not only does Crypto-1 use way too short a key (48 bits) by today's standards, its cipher structure also allows very easy recovery of its internal state (and hence the secret key) if the attacker learns a small number of contiguous keystream bits [11]. This allows a sniffer to recover the secret key if it is placed in proximity when a pair of legitimate reader and tag are in a transaction.

There are two serious implementation flaws which also cause weaknesses: (i) Parity bits are *encrypted along with the original plaintext*, which leaks information on keystream bits; (ii) the 32-bit *tag nonces used in the authentication satisfy a degree-16 linear recurrence relation and can be controlled* (by appropriately timing the authentication attempts). Together, they allow extremely efficient attacks even when the attacker only has access to the tag [12].

Compared with sniffer-based attacks, these efficient card-only attacks are arguably much more serious because of the low entry barriers: all an attacker needs is a PC and a cheap, off-the-shelf reader, so any ordinary person can launch such an attack in private by downloading the appropriate software from the internet.

In late 2012, the EasyCard Corporation rolled out the so-called "EasyCard 2.0," a dual-interface smart card that is compatible with existing EasyCard readers, yet with all implementation flaws *fixed*. The tag nonces seem random and unpredictable, and the tag responses are indistinguishable whether the parities sent by the reader are rm correct or not. This renders all existing efficient card-only attacks [6, 7, 11–13] *ineffective*, as we have verified through experiments. This doesn't stop, of course, brute-force attacks, which are arguably less threatening because it takes years of computation on an ordinary PC. The attacker would need to have access to expensive supercomputers, e.g., GPU or FPGA clusters, in order to recover the keys within a reasonably short amount of time. As a result, the EasyCard Corporation seems confident that EasyCard 2.0 can be "reasonably secure," as the computational power required by brute-force attacks is way beyond the reach of an ordinary person.

In this paper, we will show that such a sense of security is *false*. Namely, we will present a new card-only attack based on state-of-the-art algebraic differential cryptanalytic techniques [1]. The attack is highly practical: it uses the same cheap reader as previous attacks [6, 7, 11–13] and takes 2–15 minutes on a PC to recover the secret key of EasyCard 2.0 or other similar implementations of MIFARE Classic. The extra price one needs to pay for the new attack is a slightly longer time for data-collection, typically 10–20 hours. We note that this is not atypically long for differential attacks. Overall, this is a significant improvement over the brute-force attacks, which would take about 4 years on the same PC.

The rest of this paper is organized as follows. In Section 2, we will first give some background information on the cipher itself and the cryptanalytic techniques we have used to attack it. We will then present our new attack in Section 3 and empirical resultss in Section 4. Finally, we will discuss the implications and conclude this paper in Section 5.

2 Background and Related Work

2.1 Crypto-1 and the MIFARE Classic Authentication Protocol

Crypto-1 is a stream cipher used to provide cryptographic protection to MIFARE Classic tags and contactless smart cards. For more than a decade, its design was kept secret by NXP, along with the rest of MIFARE Classic. After the details of MIFARE Classic was reverse-engineered in 2007 [11, 14, 15], many weaknesses have been discovered, and with them many attacks. These attacks vary greatly in efficacy. The first few key-recovery attacks exploit the weaknesses of the cipher and gather the required information either by direct communication with a legitimate reader or by eavesdropping a communication session. Although some system vendors argue even today that these attacks are impractical, the cipher itself was by then considered cryptographically broken.

A few months later, better, card-only attacks were published [12]. These exploit several properties in the authentication protocol of MIFARE Classic as well as flaws in generating tag nonces.

For the sake of completeness, we include a brief description of Crypto-1 and its use in the authentication protocol of MIFARE Classic. Crypto-1 uses a 48-bit linear feedback shift register (LFSR) with nonlinear output filter [11]. The feedback function of the LFSR is $F(s_0, s_1, \ldots, s_{47}) := s_0 \oplus s_5 \oplus s_9 \oplus s_{10} \oplus s_{12} \oplus s_{14} \oplus s_{15} \oplus s_{17} \oplus s_{19} \oplus s_{24} \oplus s_{25} \oplus s_{27} \oplus s_{29} \oplus s_{35} \oplus s_{39} \oplus s_{41} \oplus s_{42} \oplus s_{43}$. With every tick of the clock, 20 bits from the LFSR are fed into the function f to generate one new bit of the keystream. Then the LFSR shifts one bit to the left, and the new rightmost bit is filled by the output of F — or (if the operational phase calls for inputs) F XORed with an input bit. F is *primitive*: the LFSR has a period of $2^{48} - 1$ clock cycles, the maximum possible.

The function f or *output filter* consists of two layers of nonlinear functions. The first layer is a mixed combination of two 4-input nonlinear functions f_a and f_b, and the second layer is a 5-input function f_c. Here, $f_a = \texttt{0x2c79}$,

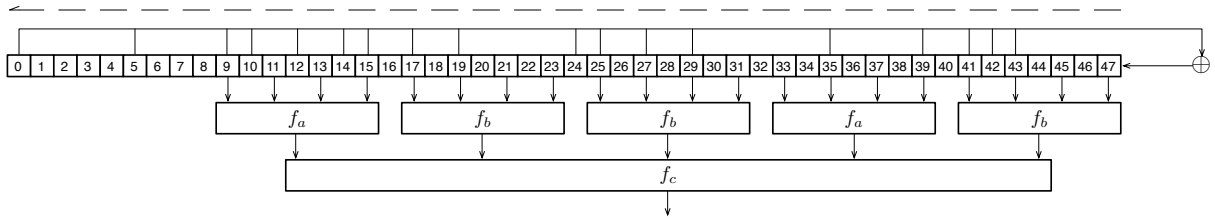

Figure 1: The structure of the Crypto-1 stream cipher

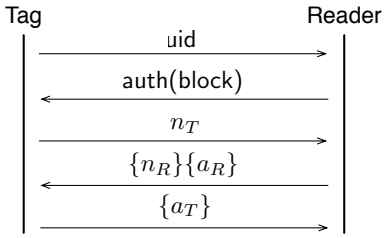

Figure 2: Authentication Protocol in MIFARE Classic.

$f_b = \texttt{0x6617}$, $f_c = \texttt{0x7907287b}$ in "table form" (collating the output bits as the input goes lexicographically over its range), and f can then be expressed as

$$\begin{aligned} f(s_0,\ldots,s_{47}) := f_c(&f_a(s_9,s_{11},s_{13},s_{15}),\\ &f_b(s_{17},s_{19},s_{21},s_{23}),f_b(s_{25},s_{27},s_{29},s_{31}),\\ &f_a(s_{33},s_{35},s_{37},s_{39}),f_b(s_{41},s_{43},s_{45},s_{47})). \end{aligned} \quad (1)$$

Note that each has an equal number of 0 and 1 bits, and hence outputs 0 or 1 each with probabily 1/2 if input bits are independently and uniformly distributed over $\mathbb{F}_2$ [12].

On being powered up by the reader's electromagnetic field, the tag sends its uid to the reader to start the anticollision phase. The reader may then request to authenticate a specific block. The tag sends back a challenge nonce n_T in plaintext. Meanwhile, $n_T \oplus \text{uid}$ is shifted into the LFSR state of Crypto-1. All subsequent communication is encrypted with the keystream, and we will use the notation $\{X\}$ to represent the ciphertext of X, i.e., $X \oplus$ keystream. Next, the reader picks its challenge n_R, which will also be shifted into the cipher, and sends $\{n_R\}$ followed by the answer $\{a_R\}$ to the tag's challenge. Finally, the tag replies with its answer $\{a_T\}$ to conclude the authentication procedure (see Fig. 2). If the tag and the reader used the same secret key for the initial state of their ciphers, this authentication procedure should bring the ciphers on either side to the same internal state, and the two keystreams generated by both ends will be henceforth in sync.

2.2 Existing card-only attacks against MIFARE Classic

The best known attacks have been summarized by Garcia *et al.* [12], which we will recapitulate here for the sake of completeness. The card-only attacks mainly exploited the following weaknesses.

1. The communication of MIFARE Classic follows the ISO 14443-A standard, which requires that a parity bit be sent after every 8 bits of transmission. However, in MIFARE Classic, these parity bits are computed over the plaintext, and the keystream bit used to encrypt the parity bits is reused to encrypt the next bit of plaintext. Furthermore, during authentication, the tag would not reply anything if the received messages have incorrect parity, i.e., the tag checks the authenticity of the reader's answer only if the parity bits are correct.

2. If all parity bits are correct but the encrypted answer $\{a_R\}$ to the tag's nonce cannot be correctly verified, the tag responds with an encrypted 4-bit error code. Since the error code is fixed, this leaks 4 keystream bits.

3. The 32-bit tag nonce is actually generated by a 16-bit LFSR which runs in a deterministic cycle after it powers up. Therefore, controlling or measuring when the reader sends every authentication request basically gives us control or a very good guess to the next tag nonce.

4. When a reader is already communicating with a tag (i.e., having authenticated to certain sector), the protocol of a subsequent authentication for a new sector differs slightly from the initial one in that the tag nonce will be encrypted by the new sector key

before transmitted to the reader. Since the first tag nonce was sent in plaintext, and the timing between two authentication attempts is known, an attacker can guess the second tag nonce and recover 32 bits of keystream with high accuracy.

Taking advantage of the weakness in the parity bits, the attacker can ask to authenticate for a sector of the tag at hand and answer the tag's challenge with random $\{n_R\}$ and $\{a_R\}$ (totally 8 bytes) accompanied with 8 random parity bits. On average, one out of 256 trials will the attacker receive the encrypted error code from the tag. Each such trace reveals 12 bits of information (8 parity bits and 4 error code bits) on the secret key. In practice, six traces are enough for the offline brute-force check of the secret key. It takes $6 \cdot 256 = 1536$ trials on average to gather these traces and can be accomplished within a minute. The offline part of this attack is to check which key out of the 2^{48} possible keys generates all "correct" parity and error code bits in these traces, and the computing time depends on the implementation realized by the attacker. Pessimistically, the run-time of checking on a powerful FPGA cluster like COPACOBANA is around half an hour.

Two other attacks try to trade online communication for the offline computing time using the weakness that tag nonces n_T can be controlled precisely by timing the authentication requests. The attacker may substantially reduce offline search space by fixing either tag or reader nonce while varying the other, and look for specific properties.

In the second card-only attack [12], the tag nonce n_T is fixed. The attacker searches for a reader nonce n_R such that flipping the last bit in each byte of n_R also flips the following encrypted parity bit, which averages around 28500 authentication attempts or 15 minutes. Such a n_R let us cut approximately 15 bits from the offline search space, enabling a standard desktop to finish the computation in around 1 minute.

For the third attack [12], the attacker fixes response of the reader to $\{n_R\} = 0 = \{a_R\} = 0$, and searches for an n_T such that the tag responds with the desired encrypted error code. For example, it might be desired that the encryption bits for the code are all zero, which means that the ciphertext would be identical to the plaintext. Such search takes 4096 attempts on average since we need 12 bits (8 parity plus 4 keystream bits) to be exactly zero. The direct offline search in a huge precomputed table (with around 2^{36} entries) of the cipher states that could lead to such pattern may take about one day. However, with some further attempts to find the parity bits that correspond to the same n_T but different n_R and a_R (e.g., $\{n_R\} = \{a_R\} = \texttt{0xffffffff}$), one can split the table into 4096 parts. This not only makes it easier to store and read the table but also speeds up the offline search significantly.

A fourth attack [12] tries to derive from a known sector key 32 keystream bits generated by a sector key using another unknown one. That, plus the parity-bit weakness in the parity bits, and Crypto-1 being strucutured such that the stream can be separated into odd and even-numbered bits, let us further reduce the search space. As a result, around only three authentication attempts and less than a second of offline computation would be needed to determine the second sector key.

Impact and current countermeasures. The last attack is particularly critical as it takes less than one second per additional key, making is feasible to "pickpocket" a card wirelessly if a deployed system leaves unused sectors with default keys or does not diversify keys. Therefore, several countermeasures have been implemented in newer versions of MIFARE Classic cards, such as Easy-Card 2.0, that are still compatible with legacy systems.

First and most importantly, the generator of the tag nonce is replaced by a better random number generator (RNG) such that it is no longer possible to control or predict n_T. Also, it is a true 32-bit RNG instead of having a period of 2^{16}. This improvement breaks almost all card-only attacks depicted in this section except the brute-force attack since all the techniques to reduce the search space make use of the flaw in tag nonce generation. Furthermore, these new cards now always reply with encrypted error code if the authentication fails, whether the parity bits are correct or not. It is no longer possible to gather the required information for the brute-force attack described above.

2.3 Algebraic differential cryptanalysis

Algebraic cryptanalysis brings the concept of applying algebraic techniques to attack various cryptographic primitives, e.g., block ciphers, stream ciphers, and public key systems. It is usually done in two major steps. First, a set of multivariate polynomial equations over a finite field is constructed to describe the cryptographic scheme. This system of equations is formulated in a way that its solutions correspond to certain secret information of the cryptographic scheme. The second step is then to solve the system using techniques such as SAT solvers or Gröbner-basis algorithms. As a result, the efficiency of this category of attacks is strongly related to the quality

of the constructed equations as well as the performance of the system-solving technique in use.

The idea of algebraic cryptanalysis is not new. Back in 1949, Shannon already noted the relationship between breaking a good cipher and solving a complex system of equations [16]. Shannon probably was thinking about how to build a good cipher, but this concept gives us a hint of checking possible weaknesses of cryptosystems using algebraic techniques. However, it was not until the huge progress in the efficiency of system solving, especially the solving of multivariate polynomial systems, that people started to consider system-solving as legitimate attacks. The invention of XL, F_4, and F_5 [5, 9, 10] and their variants greatly boosted the speed of solving multivariate polynomial systems. Also, the substantial advances in the performance of SAT solvers provides us with an alternative, namely to transform problems into boolean formulas and search for solutions.

Differential cryptanalysis exploits information leaked by special pairs of input and output differences, called *differentials*, in a block cipher to distinguish its output from random or to recover (some of) its key bits [2]. Such an attack is statistical in nature and usually requires a large number of plaintext-ciphertext pairs, especially in the context of known-plaintext or ciphertext-only attacks, for which the attacker cannot freely choose the plaintexts. Even before its publication, differential cryptanalysis has played a very important role in cipher design. It is so successful that today's standard procedures for designing a new cipher include checking differential immunity.

In the recent seminal work [1], Albrecht and Cid tried to incorporate the information obtained from differential characteristics into algebraic attacks. They proposed three methods, labeled simply as Attack A, B, and C, of obtaining and using such information. It happens that Attack B works for us even though it was ineffective against the PRESENT cipher [17]. The reason is that we have a much better distinguisher to be detailed in Section 3.

2.4 SAT solvers

Satisfiability (SAT) is the problem of deciding if a boolean formula is *satisfiable*, i.e., if there exists an assignment to the variables such that this boolean formula evaluates to `true`. Usually the boolean formula is read in conjunctive normal form (CNF), which is a conjunction (logical `AND`) of clauses formed by disjunctive (logical `OR`) literals. A literal is either a variable or its negation. A complete SAT solver eventually either find a satisfying assignment to the variables or proves the problem unsatisfiable.

Most modern complete SAT solvers are based on the DPLL algorithm [8], which is simply a branch-and-backtracking search algorithm with an additional technique called unit propagation to speed up the search procedure. All efficient solvers also incorporate conflict analysis techniques to learn new clauses, whuch help pruning the search tree, with the difference mainly in policies for deciding the next branching literal and their schemes for clause learning.

SAT solvers let us solve problems with large number of variables within reasonable time and have been extensively used in electronic design automation (EDA). Recently, they are also adopted in other fields such as artificial intelligence, model checking, bioinformatics, and cryptanalysis. SAT solvers can be used as a solving engine in a larger solver, i.e., as a subroutine for finding solutions to suitably formulated boolean SAT subproblems. Alternatively, it can also be extended to take advantage of the structure in the target problems. For example, the award-winning CryptoMiniSat adds the ability to deal with `XOR` clauses, which play an important role in problem instances arising from cryptanalysis. Since an `XOR` clause with l literals will expand to 2^{l-1} `OR` clauses, incorporating such an ability not only saves memory usage to a great extent, but also significantly accelerates the search speed if the majority of the clauses in the problem are `XOR` clauses.

3 Our attacks

As mentioned in Section 2.1, the introduction of improved MIFARE Classic cards, e.g., EasyCard 2.0, blocks all existing efficient card-only attacks while maintaining compatibility with legacy readers. In this section, we illustrate several potential attacks using modern cryptanalysis techniques against these new cards.

Since the tag now replies with an encrypted error code to every authentication failure (including parity error), it leaks four keystream bits per authentication attempt. We call the data collected in one failed authentication attempt a *trace* and derive a set of algebraic equations for the four keystream bits as the following. Let $\mathbf{x} = (x_0, \ldots, x_{47})$ denote the initial state of the LFSR, i.e., the secret key. The new state of the LFSR after inputting an n-bit sequence $\mathbf{i}$ can be written in a form like:

$$\mathbf{A_i}(\mathbf{x}) = \mathbf{L}^n\mathbf{x} + \mathbf{v_i}, \tag{2}$$

where $\mathbf{L}$ is a linear transformation that depends only on the LFSR's feedback function F, and $\mathbf{v_i}$ is a 48-bit vector

that depends on the input $\mathbf{i}$. Then the keystream bit generated by the nonlinear filter right after the input $\mathbf{i}$ can be obtained by

$$a_{\mathbf{i}} = f(\mathbf{A}_{\mathbf{i}}(\mathbf{x})). \tag{3}$$

Both uid and n_T are transmitted in plaintext. It is then easy to express the LFSR state after inputting $\mathsf{uid} \oplus n_T$ in terms of the unknowns $x_0, \ldots, x_{47}$ using Eq. (2). Although only the *encrypted* reader nonce is available (in fact, it is generated by the attacker in card-only attacks), it is still possible to *decrypt* $\{n_R\}$ using the keystream bits obtained by Eq. (3) and derive subsequent LFSR states and keystream bits in the form of polynomials of $x_0, \ldots, x_{47}$. By equating the 4 keystream bits to their corresponding polynomials, we get 4 equations per trace of failed authentication session. It is then theoretically possible to collect sufficiently many equations (≥ 12) and solve the resulted system using Gröbner-basis or SAT solvers.

In practice, however, the main difficulty of the algebraic attack described above lies in the last step, namely, solving the resulted polynomial system. In fact, the degree of such systems saturates due to the nonlinearity introduced by the recurrent decryption of $\{n_R\}$. In order to speed up the solving procedure, we try to extract more information from the traces using algebraic differential cryptanalytic techniques.

According to Eq. (2), we know that the difference of two LFSR states that descend from a common ancestor is $\mathbf{A}_{\mathbf{i}}(\mathbf{x}) \oplus \mathbf{A}_{\mathbf{j}}(\mathbf{x}) = \mathbf{v}_{\mathbf{i}} \oplus \mathbf{v}_{\mathbf{j}}$, where $\mathbf{i}$ and $\mathbf{j}$ are two input bit streams of the same length. It means that we can *know* the LFSR state difference after two different n_T's, even though we *cannot control* it. This is easy to circumvent, however, as one can keep authenticating with a card at hand, hoping that the desired differences will eventually show up. For example, we are interested in those pairs which have only one bit difference in the LFSR state after inputting $\mathsf{uid} \oplus n_T$, especially when the different bit lies at the leftmost possible position. More specifically, let $\mathbf{y}_{\mathbf{i}}$, $\mathbf{y}_{\mathbf{j}}$ denote the LFSR states after inputting $\mathbf{i}$, $\mathbf{j}$ that correspond to two different tag nonces, then our target would be the pairs such that $\Delta\mathbf{y} = \mathbf{y}_{\mathbf{i}} \oplus \mathbf{y}_{\mathbf{j}} = \mathsf{0x000080000000}$. Since n_T has only 32 bits, one bit difference at position 16 (cf. Fig. 1) is the furthest we can get. Thanks to the birthday paradox, it does not take too long to gather sufficiently many such pairs.

Once such a pair is obtained, we can try to manipulate the n_R of the second trace so that the difference of the LFSR state after inputting n_R is 0. Because n_R is transmitted in ciphertext, we will need to *guess* the difference in keystream bits whenever there is a difference in inputs to the nonlinear filter function. Such a cancellation of input difference could be examined by checking whether the tag responds with an identical encrypted error code. In other words, the four keystream bits obtained after each authentication failure are used as an oracle for confirming whether our guesses in $\{n_R\}$ successfully produce the desired differential or not. In practice, there are false positives due to collision. We will discuss more about this in Section 4.1 along with the success rate of guessing the difference bits.

More specifically, our goal in this stage is to keep pushing bits with zero difference into the LFSR. Since there is only one bit of difference at position 16 of the LFSR state at the beginning of this stage, we only need to keep our eyes on it. When it is shifted to a position that is not part of input to the nonlinear filter function f, the difference in keystream bit is 0. In this case, we know exactly what the difference in the corresponding bits of $\{n_R\}$ should be by inspecting the feedback function of LFSR. However, for positions 15, 13, 11, and 9 (cf. Fig. 1), we need to guess the output keystream bits of the nonlinear function f. If all four guesses are correct, then we get a pair of traces with identical LFSR state after shifting in n_R. Let $\mathbf{z}_k$, $\mathbf{z}'_k$ be the LFSR states of the pair after shifting in k bits. Then the following four equations, or *differential relations*, should hold.

$$f(\mathbf{z}'_k) \oplus f(\mathbf{z}_k) = f(\mathbf{z}_k \oplus \mathbf{e}_{48-k}) \oplus f(\mathbf{z}_k) = \frac{\partial f}{\partial z_{48-k}}(\mathbf{z}_k) = \delta_k, \tag{4}$$

where $\mathbf{e}_k$ is the 48-bit vector with 1 in the k-th position and 0 elsewhere, and δ_k is the guessed difference, for $k = 33, 35, 37, 39$.

In addition to Eq. (4), by taking a closer look at state $\mathbf{z}_{33}$, we devise the following formula as a filter to reduce the search space of our attack.

$$\left(\frac{\partial f}{\partial z_{15}}(\mathbf{z}_{33}) \oplus \delta_{33}\right)\left(\frac{\partial^2 f}{\partial z_{15}\partial z_{47}}(\mathbf{z}_{33}) \oplus 1\right) = 0. \tag{5}$$

Any state assignment that does not satisfy Eq. (5) would result in $\frac{\partial f}{\partial z_{15}}(\mathbf{z}_{33}) \neq \delta_{33}$ and $\frac{\partial^2 f}{\partial z_{15}\partial z_{47}}(\mathbf{z}_{33}) = 0$ at the same time. This means that, no matter what the value of the newly input bit (z_{47}) is, the output of f would not be equal to our guessed value, which contradicts with the fact that we have already reached the same LFSR state in both traces at the end of the authentication session. As a result, we have an additional equation for each successful pair of traces, which is expected to work as a filter that eliminates 1/4 of the solution space. In practice, there is a high degree of dependency among the traces, but it does not take too long to collect sufficiently many pairs such

that only a few candidate solutions can pass all filters. Based on our experience, these filters help tremendously in solving the nonlinear system.

4 Empirical results and Discussion

4.1 Dealing with false positives

In practice, the oracle used in our attack is not 100% correct. Because we can only observe four keystream bits, it is possible for two traces to have the same keystream bits yet different internal states. In our experiments, around 26% of the cases where the four keystream bits agree are actually false positives. As a result, not all the collected differential relations, i.e., Eq. (4) and (5), can be incorporated in the final system to solve. Alternatively, we can use the following method to reduce the search space in real implementation.

We note that only 18 bits $(z_9, z_{11}, z_{13}, z_{17}, z_{19}, \ldots, z_{45})$ might have an effect on the evaluation of Eq. (5), which we call the *filter equation* here. Random assignments to these 18 bits should be in the solution space of the filter equation with probability $q = 3/4$, given that the filter function f is unbiased. Additionally, the correct assignment should be a solution to those filter equations collected from the true positive results. If we collect sufficiently many pairs and rank all 2^{18} possible assignments by their number of correct evaluations to the collected filter equations, the correct assignment should be very close to the top of the list with a high probability. Therefore, the list serves as a good guide for guessing the 18 bits in the resultant system of equations. We can substitute the 18 variables with the bit assignments according to the list and try solving the system using SAT solvers. According to our empirical results, it takes around 2 to 15 minutes for CryptoMiniSat to solve the system if the 18 bits are assigned with correct values.

The next question is how many pairs are *sufficient* to put the correct assignment at the top of the list with high probability. Assume that in total N such differential pairs are collected, among which $\tilde{N}$ are true positives. The number of filter equations that the correct assignment would evaluate to true, denoted by N_1, should have the following probability mass function.

$$Pr[N_1 = n] = \binom{N-\tilde{N}}{n-\tilde{N}} q^{n-\tilde{N}} (1-q)^{N-n}, n = \tilde{N}, \ldots, N. \tag{6}$$

Furthermore, if we denote the rank of the correct assignment in the list by ρ, then we have

$$Pr[\rho = k | N_1 = n] = \binom{M-1}{k-1} a(n)^{M-k} [1-a(n)]^{k-1}, \tag{7}$$

where $M = 2^{18}$ and $a(n) = \sum_{i=0}^{n-1} \binom{N}{i} q^i (1-q)^{N-i}$ is the probability that an incorrect assignment evaluates less than n filter equations to true. Using Eq. (6) and (7), it is straightforward to compute the probability function of the rank of the correct assignment by

$$Pr[\rho = k] = \sum_{n=\tilde{N}}^{N} Pr[\rho = k | N_1 = n] Pr[N_1 = n]. \tag{8}$$

We compute the percentiles of the rank of the correct bit assignments for various numbers of filter equations and summarize the most useful results in Table 1. This gives us an estimate of how many pairs would be sufficient to substantially reduce the expected number of trials we have to perform before finally solving the system. For example, given 150 filter functions collected, we are able to solve the system in less than 7 trials with a probability of 99%. This is a very good result because in most cases, we just need to repeat the computation a few times before we can recover the key.

Number of filter equations (N)	Percentile	Rank(ρ)
90	75	11
110	90	8
130	95	4
150	99	6

Table 1: The percentiles of ρ ($\tilde{N}/N = 74\%$)

4.2 The complete attack

We first summarize our attack procedure as follows.

1. Initiate (failing) authentication sessions with the target tag and record in a database each n_T received, $\mathbf{v}_{n_T}$, and four keystream bits $\mathbf{s}$ used to encrypt the returned error code.

2. For each $(n_T, \mathbf{v}_{n_T}, \mathbf{s})$ received, check whether $\mathbf{v}_{n_T} \oplus$ 0x000080000000, matches any $\mathbf{v}_{n'_T}$ already recorded. If so go to Step 3, having found a pair of n_T's that produce the state difference $\Delta\mathbf{y} =$ 0x000080000000. Otherwise repeat Step 1.

3. Guess four δ_k's and manipulate $\{n_R\}$ accordingly. Check whether we see the same four keystream

bits. If so, record the four differential relations (cf. Eq. (4)) thus found.

4. Repeat Steps 1-3 until we have collected enough differential relations (about 600–1000, or 150 to 250 successful attempts), then we use the method from Section 4.1 to remove the false positives.

5. Feed the differential relations, along with (i) some equations on keystream bits, to a Gröbner-basis or SAT solver, and (ii) the 18-bit solution to the filter equations (cf. Section 4.1) as hint bits, to the (SAT-) solver and solve for the key. Empirically, we need about 1 keystream bit equation for every 4–5 differential relations.

We note that the differential relations, as a system of equations, tend to be highly redundant and have multiple solutions. It is to avoid ending up with such a wrong solution, that in step 5 we must add a few equations on keystream bits in order to obtain a unique solution with high probability.

Also, we have tried several different solvers including the built-in Gröbner-basis solver in Maple, as well as PolyBoRi [3]. Empirically, CryptoMiniSat outperforms the other solvers by a large margin. Hence we only report the timings obtained using CryptoMiniSat for the rest of the paper. The results also show that the hint bits are extremely helpful to CryptoMiniSat, usually resulting in a tremendous speed-up.

A submarine patch. We had bought a fair number of EasyCards on the streets of Taipei between 2009 and 2012 trying to track there were different editions of EasyCards. Surprisingly, we discovered that EasyCard 2.0 was actually not the first "patch" attempted by the EasyCard Corporation. There is actually another different kind of EasyCard, that we shall refer to as EasyCard 1.5, which has been surreptiously in circulation since late 2010 or early 2011.

Although to all outward appearances EasyCard 1.5 is identical to EasyCard 1.0, it has a better RNG which makes n_T neither predictable nor controllable based on timing. This already defeats some (but not all) existing card-only attacks, even though EasyCard 1.5 performs otherwise identically to the original. For example, since the parities attack relies on the capability of controlling n_T, such an improved RNG already makes the attack time much longer if still possible at all. We represent this fact using a question mark in Table 3, in which we summarize the time required to carry out various attacks. It is perhaps surprising that the EasyCard Corporation managed to resist the temptation of announcing a security upgrade and kept this modification under wraps for so long. The differences among the three types of EasyCards are summarized in Table 2.

Card type	Parities checked	n_T generation
EasyCard 1.0	Yes	Predictable
EasyCard 1.5	Yes	Somewhat random
EasyCard 2.0	No (always `0x0`)	Random

Table 2: Types of EasyCards attacked in our experiments

From Table 3, it is clear that our attack is the most practical one among the effective attacks against EasyCard 2.0 in the sense that our attack can be carried out by an ordinary person in private with an off-the-shelf reader and a PC.

In Table 3, the GPU result is taken from Chih *et al.* [4], while all other experiments are all carried out on a PC with 2.3 GHz AMD CPU. The data collection, on the other hand, is performed on a laptop PC with 2.0 GHz Intel CPU. For CPU brute-force attack, we obviously have not run it to completion but extrapolate based on the timing result of a partial run instead. We use open-source software whenever possible, but we have also implemented and optimized some of the attacks. We are in the stage of cleaning up our software, which will be released as open-source.

5 Concluding remarks

In this paper, we have demonstrated a highly practical attack against the EasyCard 2.0, which is marketed as having patched the vulnerabilities of previous implementations of MIFARE Classic. By applying algebraic differential cryptanalysis techniques, our card-only attack can recover the secret key of EasyCard 2.0 within one day. This includes the time for online data collection and offline computation, both of which can be carried by a working platform that costs no more than a few hundreds of US dollars and is affordable even to the least wealthy attacker. This again shows the weakness of the Crypto-1 cipher, and highlights the unfortunate the fact that "security" protocols based on unsound ciphers, such as MIFARE Classic, is not suitable for important transactions such as electronic payment.

It is noteworthy that although our attack takes advantage of some specific weaknesses of Crypto-1, the algebraic differential techniques we have developed in this work could also apply to other ciphers or even hash func-

Attack type	Online time	Compute time	1.0	1.5	2.0
Sniffing attack	2 sec.	$<$ 2 sec.	$\surd$	$\surd$	$\surd$
GPU brute-force [4]	5 sec.	14 hours	$\surd$	$\surd$	$\surd$
CPU brute-force	5 sec.	4 years	$\surd$	$\surd$	$\surd$
Parities attack	$>$ 3 min.	$<$ 30 sec.	$\surd$	?	
Nested authentications	15–75 sec.	25–125 sec.	$\surd$	$\surd$	
Our attack	10–20 hours	2–15 min.			$\surd$

Table 3: Timing comparison of all known attacks

tions. All we need is a way to collect algebraic (differential) equations from the target cipher and send the resulted system to Gröbner-basis or SAT solvers. Currently, we believe that such a technique, combined with the advances in the solvers, is quite general and powerful. The next step of our research is to investigate and analyze in full the strengths, shortcomings, and performance of this technique.

References

[1] ALBRECHT, M., AND CID, C. Algebraic techniques in differential cryptanalysis. In *Proceedings of the 16th International Workshop on Fast Software Encryption* (Berlin, Heidelberg, 2009), O. Dunkelman, Ed., FSE 2009, Springer-Verlag, pp. 193–208.

[2] BIHAM, E., AND SHAMIR, A. Differential cryptanalysis of DES-like cryptosystems. In *Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, UK, 1991), CRYPTO '90, Springer-Verlag, pp. 2–21.

[3] BRICKENSTEIN, M., AND DREYER, A. PolyBoRi: A framework for Gröbner-basis computations with boolean polynomials. *J. Symb. Comput. 44*, 9 (Sept. 2009), 1326–1345.

[4] CHIH, M.-Y., SHIH, J.-R., YANG, B.-Y., DING, J., AND CHENG, C.-M. MIFARE Classic: Practical attacks and defenses. In *19th Cryptology and Information Security Conference (CISC 2010)* (Hsinchu, Taiwan, May 2010).

[5] COURTOIS, N., KLIMOV, A., PATARIN, J., AND SHAMIR, A. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Proceedings of the 19th international conference on Theory and application of cryptographic techniques* (Berlin, Heidelberg, 2000), EUROCRYPT '00, Springer-Verlag, pp. 392–407.

[6] COURTOIS, N. T. The dark side of security by obscurity and cloning MiFare Classic rail and building passes anywhere, anytime. Cryptology ePrint Archive, Report 2009/137, 2009. `http://eprint.iacr.org/`.

[7] COURTOIS, N. T., NOHL, K., AND O'NEIL, S. Algebraic attacks on the Crypto-1 stream cipher in MiFare Classic and Oyster cards. Cryptology ePrint Archive, Report 2008/166, 2008. `http://eprint.iacr.org/`.

[8] DAVIS, M., LOGEMANN, G., AND LOVELAND, D. A machine program for theorem-proving. *Communications of ACM 5*, 7 (July 1962), 394–397.

[9] FAUGÈRE, J. C. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of pure and applied algebra 139*, 1 (1999), 61–88.

[10] FAUGÈRE, J. C. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *Proceedings of the 2002 international symposium on Symbolic and algebraic computation* (New York, NY, USA, 2002), ISSAC '02, ACM, pp. 75–83.

[11] GARCIA, F. D., KONING GANS, G., MUIJRERS, R., ROSSUM, P., VERDULT, R., SCHREUR, R. W., AND JACOBS, B. Dismantling MIFARE Classic. In *Proceedings of the 13th European Symposium on Research in Computer Security: Computer Security* (Berlin, Heidelberg, 2008), ESORICS '08, Springer-Verlag, pp. 97–114.

[12] GARCIA, F. D., ROSSUM, P. V., VERDULT, R., AND SCHREUR, R. W. Wirelessly pickpocketing a Mifare Classic card. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2009), S&P '09, IEEE Computer Society, pp. 3–15.

[13] KONING GANS, G., HOEPMAN, J.-H., AND GARCIA, F. D. A practical attack on the MIFARE Classic. In *Proceedings of the 8th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Applications* (Berlin, Heidelberg, 2008), CARDIS '08, Springer-Verlag, pp. 267–282.

[14] NOHL, K., EVANS, D., STARBUG, S., AND PLÖTZ, H. Reverse-engineering a cryptographic RFID tag. In *Proceedings of the 17th conference on Security symposium* (Berkeley, CA, USA, 2008), Security '08, USENIX Association, pp. 185–193.

[15] NOHL, K., AND PLÖTZ, H. Mifare: Little security, despite obscurity. In *24th Chaos Communication Congress* (Berlin, Germany, Dec. 2007), 24C3.

[16] SHANNON, C. E. Communication theory of secrecy systems. *Bell system technical journal 28*, 4 (1949), 656–715.

[17] WANG, M., SUN, Y., MOUHA, N., AND PRENEEL, B. Algebraic techniques in differential cryptanalysis revisited. In *Proceedings of the 16th Australasian conference on Information security and privacy* (Berlin, Heidelberg, 2011), ACISP '11, Springer-Verlag, pp. 120–141.

Notes