Collision Attacks on AES-based MAC: Alpha-MAC

Alex Biryukov[1], Andrey Bogdanov[2], Dmitry Khovratovich[1] and Timo Kasper[2]

[1] University of Luxemburg, Luxemburg
[2] Chair for Communication Security, Ruhr-University Bochum, Germany
{alex.biryukov,dmitry.khovratovich}@uni.lu
{abogdanov,tkasper}@crypto.rub.de

Abstract. Message Authentication Code construction Alred and its AES-based instance Alpha-MAC were introduced by Daemen and Rijmen in 2005. We show that under certain assumptions about its implementation (namely that keyed parts are perfectly protected against side-channel attacks but bulk hashing rounds are not) one can efficiently attack this function. We propose a side-channel collision attack on this MAC recovering its internal state just after 29 measurements in the known-message scenario which is to be compared to 40 measurements required by collision attacks on AES in the chosen-plaintext scenario. Having recovered the internal state, we mount a selective forgery attack using new 4 to 1 round collisions working with negligible memory and time complexity.

Key words: Alpha-MAC, message authentication codes, MAC, AES, collision attack, side-channel attack, selective forgery

1 Introduction

A general Message Authentication Code construction Alred and its instance Alpha-MAC were introduced by Daemen and Rijmen in [1]. The Alpha-MAC is an iterative MAC function operating the state that is changed by consecutive "injections" of message blocks. The secret key of the Alpha-MAC is used as a key of two AES transformations, which are applied at the beginning and at the end of computation, respectively.

The Alpha-MAC is very likely to be used in embedded systems (e.g. smart card applications, etc.), since it suggests some significant benefits in this case. First, the Alpha-MAC can be easily implemented, if the AES algorithm is already on board, which has been becoming common practice in the real world. Moreover, the advantage of the Alpha-MAC over traditional MAC constructions such as CBC-MAC (where a message is encrypted using a block cipher in CBC mode and the last ciphertext block is output) is its performance. For CBC-MAC based on AES-128 the number of rounds per one processed 128-bit message block is 10, and for the Alpha-MAC it is $4 + \frac{20}{t}$, where t is the number of processed blocks.

That is, for sufficiently long messages the Alpha-MAC outperforms CBC-MAC by factor 2.5, which affects both the runtime and the power consumption.

Daemen and Rijmen provided several theorems that substantiate resistance of the Alpha-MAC against adaptively-chosen-message attacks. They proved that any forgery attack on the Alpha-MAC not involving internal collisions may be easily extended to an attack on the AES itself. Furthermore, they showed that any colliding messages of the same size have to be at least 5 blocks long. Due to the existence of keyed transformation both at the beginning and at the end of the Alpha-MAC the goal to construct such collisions seems intractable without any extra information except for the input-output pairs.

Recently Huanga et al. [2] have shown how to construct 5-block collisions given a complete internal state or a secret key. Unfortunately, they did not consider how to derive that information in real-world applications.

The contribution of our paper is two-fold: First we show how one can derive the internal state using side-channel collision attacks, first introduced in [3,4] for DES. Here we start with the assumption that the two keyed AES transformations of Alpha-MAC have perfect protection against side-channel attacks but that the inner rounds, which perform the bulk of the hashing and do not involve the secret key material, are not protected. Our technique is different from that of a collision attack on AES [5] since in Alpha-MAC the attacker controls only short 32-bit injections in each round, instead of a single 128-bit input. Nevertheless, we are able to recover a full internal state of Alpha-MAC in just 29 measurements in the *known* message scenario and without memory-intensive precomputations (compare it to 40 measurements in the *chosen* message scenario and 540 Mbytes of memory for a collision attack on AES [5]). The complexity of the offline part is about 2^{34} operations in $GF(2^8)$. As opposed to the standard collision attack on AES, we search for partial collisions over 4 consecutive rounds instead of a single round. Since DPA typically requires several hundred measurements, our side-channel collision attack is superior in terms of measurement complexity.

Secondly we show that instead of 5 block to 5 block or longer collisions one can construct 4 to 1 block collisions. Given an internal state (recovered by the side-channel collision attack) our algorithm constructs Alpha-MAC collisions with negligible time and memory complexity. The way we construct collisions allows to perform *selective forgery* attacks (arbitrary choice of messages to be authenticated except for a 128-bit suffix) on vulnerable implementations of Alpha-MAC. A remedy for the attacks shown in this paper would be to protect also the inner rounds of the Alpha-MAC implementation against power analysis. This agrees with the conjecture in [6] concerning the protection of the inner rounds of AES implementations against cache-based attacks, see also [7].

The paper is organized as follows. Section 2 gives a short description of the Alpha-MAC. In Section 3 we show how to obtain the internal state using side-channel collision attacks on the Alpha-MAC. Section 4 demonstrates a way to construct collisions in the Alpha-MAC using the knowledge of the internal state. Our experiments are described in Section 5, where we also briefly discuss possible countermeasures against our attacks. We conclude in Section 6.

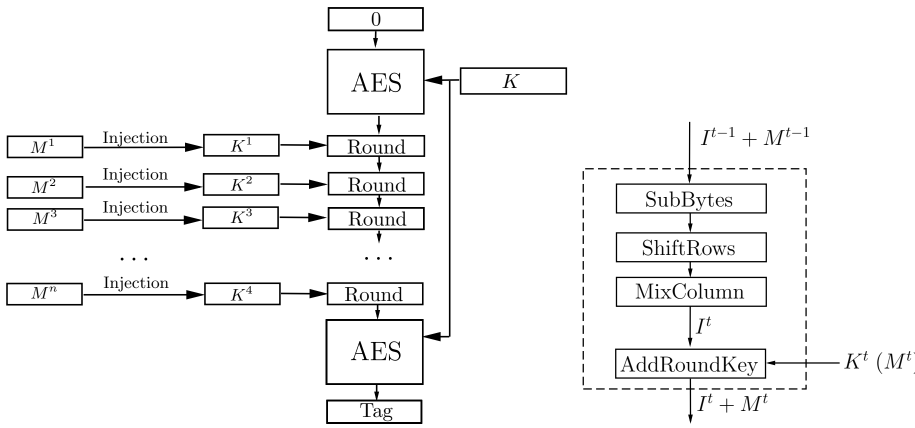

Fig. 1. The Alpha-MAC: Internal structure and the t-th round of message-dependent transformations

2 Description of the Alpha-MAC

Here we follow most of the notations from the original paper [1]. An Alpha-MAC structure is illustrated in Figure 1.

Let M be $4n$ byte long message. A computation of a MAC for M is a three step process. First AES with the key K (of length 16, 24, or 32 bytes) is used to encrypt a zero block. Then n *AES* rounds are applied with the subkey of the i-th round being of the form:

$$\textsc{Injection:} \quad K^i = \begin{pmatrix} m^i_{00} & 0 & m^i_{02} & 0 \\ 0 & 0 & 0 & 0 \\ m^i_{20} & 0 & m^i_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

where $(m^i_{00}, m^i_{02}, m^i_{20}, m^i_{22}) = M^i$ — the i-th block of M. After n such rounds the result is again AES encrypted with the same key K. Here and later we will denote an internal state I^t modified by $\textsc{AddRoundKey}(K^t)$ by $I^t + M^t$ instead of $I^t + K^t$ for convenience.

In [1] it was also allowed to add a Truncation block which crops some bytes from the final result. Truncation does not affect our attack so we omit it for simplicity.

3 Recovering the Internal State

In this section we show how to obtain the internal state of the Alpha-MAC using side-channel collision attacks and differential power analysis. Our attacks are aimed at the internal state rather than at K for the following reason: We attack implementations of the Alpha-MAC under the assumption that keyed AES transformations (before and after the message injections) are protected against side-channel attacks (e.g. through masking, etc.) and the internal injection rounds are not. It is a rather realistic assumption as developers in the real-world applications are likely to protect keyed transformations and keep the unkeyed rounds which perform the bulk of the hashing unprotected or weakly protected due to performance concerns. For instance, processing a (relatively short) 10-Kbyte message with the Alpha-MAC requires a total of 2580 AES-128 rounds, from which only 20 are keyed.

Another assumption is that the attacker can observe messages that are to be processed by the Alpha-MAC and is able to measure the power consumption of the computing device. Moreover, we require the observed messages to look random. This assumption is substantially different from that for the basic collision attack in [3], where the chosen-message possibility is required. Note also that during the attack we do not need the output of the MAC computation.

3.1 Basic Collision Attack on AES

Side-channel collision attacks were proposed for the case of the DES in [3] and enhanced in [4]. AES was attacked using collision techniques in [5]. This side-channel collision attack on AES is based on detecting internal one-byte collisions in the MixColumns transformation in the first AES round. The basic idea is to identify pairs of plaintexts leading to the same byte value in an output byte after the MixColumns transformation of the first round and to use these pairs to deduce information about some key bytes involved into the transformation.

Let $A = (a_{ij})$ with $i, j = \overline{0,3}$ and $a_{ij} \in GF(2^8)$ be the internal state in the first AES round after key addition, byte substitution and the ShiftRows operation. Let $B = (b_{ij})$ with $i, j = \overline{0,3}$ and $b_{ij} \in GF(2^8)$ be the internal state after the MixColumns transformation, $B = \text{MixColumns}(A)$, where the MixColumns transformation is defined for each column j as follows:

$$\begin{pmatrix} b_{0j} \\ b_{1j} \\ b_{2j} \\ b_{3j} \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \times \begin{pmatrix} a_{0j} \\ a_{1j} \\ a_{2j} \\ a_{3j} \end{pmatrix}. \tag{1}$$

Here all operations are performed over $GF(2^8)$. $P = (p_{ij})$ with $i, j = \overline{0,3}$, $p_{ij} \in GF(2^8)$, and $K = (k_{ij})$ with $i, j = \overline{0,3}$, $k_{ij} \in GF(2^8)$, denote the plaintext block and the first subkey, respectively, then b_{00} can be represented as:

$$\begin{aligned} b_{00} &= 02 \cdot a_{00} + 03 \cdot a_{10} + 01 \cdot a_{20} + 01 \cdot a_{30} = \\ &= 02 \cdot S(p_{00} + k_{00}) + 03 \cdot S(p_{11} + k_{11}) + \\ &\quad 01 \cdot S(p_{22} + k_{22}) + 01 \cdot S(p_{33} + k_{33}). \end{aligned} \tag{2}$$

For two plaintexts P and P' with $p_{00} = p_{11} = p_{22} = p_{33} = \delta$ and $p'_{00} = p'_{11} = p'_{22} = p'_{33} = \epsilon$, $\delta \neq \epsilon$, one obtains the following, provided $b_{00} = b'_{00}$:

$$\begin{aligned} &02 \cdot S(k_{00} + \delta) + 03 \cdot S(k_{11} + \delta) + 01 \cdot S(k_{22} + \delta) + 01 \cdot S(k_{33} + \delta) \\ &= 02 \cdot S(k_{00} + \epsilon) + 03 \cdot S(k_{11} + \epsilon) + 01 \cdot S(k_{22} + \epsilon) + 01 \cdot S(k_{33} + \epsilon). \end{aligned} \tag{3}$$

Let $C_{\delta,\epsilon}$ be the set of all key bytes $k_{00}, k_{11}, k_{22}, k_{33}$ that lead to a collision (3) with plaintexts (δ, ϵ). Such sets are pre-computed and stored for all 2^{16} pairs (δ, ϵ). Each set contains on the average 2^{24} candidates for the four key bytes. Actually, every set $C_{\epsilon,\delta}$ can be computed from the set $C_{\epsilon+\delta,0}$ using some relations between the sets. Due to some dependencies within the sets, this optimization reduces the required disk space to about 540 megabytes.

The attack on the single internal state byte b_{00} is then the following. The attacker generates random values (ϵ, δ) and inputs them to the AES module as described above. The power consumption curve for the time period, where b_{00} is processed, is stored. Then the attacker proceeds with other random values (ϵ', δ'), measures the power profile, stores it and correlates it with all stored power curves. And so on. One needs about 4 collisions (one in each output byte of a column) to recover the four bytes involved into the MixColumns transformation. The probability that after N operations at least one collision $b_{00} = b'_{00}$ occurs in a single byte is:

$$p_N = 1 - \prod_{l=0}^{N-1} (1 - l/2^8). \tag{4}$$

Actually, the attack can be parallelized to search for collisions in all four columns of B in parallel. In this case the attacker needs at least 16 collisions, 4 for each column of B, so $p_N^{16} \geq 1/2$ and $N \approx 40$. Once the required number of collisions was detected, he uses the pre-computed tables $C_{\epsilon+\delta,0}$ to recover all four key bytes for each column by intersecting the pre-computed key sets corresponding to the collisions (ϵ, δ) detected. Thus, on the average one has to perform about 40 measurements to get all 16 collisions needed and to determine all 16 key bytes. Note that since the cardinality of the intersections for the sets $C_{\epsilon,\delta}$ is not always 1, there are a number of key candidates to be tested using known plaintext-ciphertext pairs.

3.2 Our Enhanced Collision Attack on the Alpha-MAC

The collision attack on AES described above does not apply to the Alpha-MAC, since only 4 fixed bytes out of 16 input bytes K^i can vary. The other 12 bytes are zero. But the collision attack can be enhanced for the Alpha-MAC in a way that requires a reduced number of measurements (29 instead of 40) and requires no pre-computations. Note that our side-channel collision attack itself does not need the chosen-plaintext possibility. However, at the end there are about 2^8 state candidates, which have to be tested by constructing collisions and verifying them using access to the device computing the Alpha-MAC.

The ideas that we use are:

- Having detected several byte collisions, we treat them as a nonlinear system of equations over $GF(2^8)$. Then we solve these systems by brute-force. This allows not to use pre-computations and memory.
- We look for collisions in three consecutive injection rounds instead of working with only a single round. This is possible due to the fact that no entropy is introduced in the injection rounds. We show that this method requires less measurements than that of Schramm et al..

Let $I^1 = (i^1_{rs})$, $r, s = \overline{0,3}$, denote the internal state of the Alpha-MAC directly before the first injection. I^1 is $E_K(0)$ after the SubBytes, ShiftRows and MixColumns transformations of the first injection round. We also similarly define I^2, I^3, and I^4. The goal of our attack is to find I^1. The initial internal state $E_K(0)$ can be then easily computed from I^1, since all AES round transformations are bijective. We denote the i-th injection by K^i:

$$K^i = \begin{pmatrix} k^i_{00} & k^i_{01} & k^i_{02} & k^i_{03} \\ k^i_{10} & k^i_{11} & k^i_{12} & k^i_{13} \\ k^i_{20} & k^i_{21} & k^i_{22} & k^i_{23} \\ k^i_{30} & k^i_{31} & k^i_{32} & k^i_{33} \end{pmatrix} = \begin{pmatrix} m^i_{00} & 0 & m^i_{02} & 0 \\ 0 & 0 & 0 & 0 \\ m^i_{20} & 0 & m^i_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{5}$$

for $i = \overline{1,4}$.

Our side-channel collision attack on the Alpha-MAC works as follows. We treat the states $I^j + K^j$ for $j = \overline{2,4}$ and try to detect collisions in some of the column bytes for a number of messages. Note that this differs from the basic collision attack, where one looks for collisions directly after the MixColumns transform and before key addition. Collisions in $I^2 + K^2$ and $I^3 + K^3$ reveal eight bytes of the internal state I^1. Collisions in $I^4 + K^4$ recover eight linear state-dependent relations over $GF(2^8)$. These are linearly independent and can be uniquely solved, which reveals the remaining eight bytes of I^1.

After the MixColumns transform and message addition in the second injection round we have 8 bytes in $I^2 + K^2$ that can collide: $i^2_{r,0} + k^2_{r,0}$ and $i^2_{r,2} + k^2_{r,2}$, $r = \overline{0,3}$. For instance, if a collision occurs in $i^2_{00} + k^2_{00}$, one has the following relation:

$$02 \cdot S(i^1_{00} + m^1_{00}) + S(i^1_{22} + m^1_{22}) + m^2_{00} = 02 \cdot S(i^1_{00} + z^1_{00}) + S(i^1_{22} + z^1_{22}) + z^2_{00}, \tag{6}$$

where M^1, Z^1 and M^2, Z^2 are message blocks injected into the first and second injection rounds, respectively, which result in this collision. Note that the other bytes do not depend on the message and, thus, cancel out. in this equation. After a further collision of type (6) has been detected in another byte of the 0th column, one has two nonlinear equations over $GF(2^8)$ with two binary variables $i^1_{00}, i^1_{22} \in GF(2^8)$. These equations are solved by brute-force. One gets similar equations by detecting two one-byte collisions in the second column. These can be solved in the same way and yield i^1_{02} and i^1_{20}.

Next, we detect collisions after the MixColumns transform and message addition in the third injection round. Note that four bytes of I^1 are already known. If a collision is detected in $i^3_{00} + k^3_{00}$, the following relation holds:

$$\begin{gathered} 02 \cdot S(03 \cdot S(i^1_{11}) + S(i^1_{33}) + c_1 + m^2_{00}) \\ +S(S(i^1_{13}) + 03 \cdot S(i^1_{31}) + c_2 + m^2_{22}) + m^3_{00} = \\ 02 \cdot S(03 \cdot S(i^1_{11}) + S(i^1_{33}) + c'_1 + z^2_{00}) \\ +S(S(i^1_{13}) + 03 \cdot S(i^1_{31}) + c'_2 + z^2_{22}) + z^3_{00} \end{gathered} \tag{7}$$

for some injected blocks Z^2, M^2, Z^3, M^3 and known constants[3] $c_1, c_2, c'_1, c'_2 \in GF(2^8)$. Two collisions in two bytes of the 0th column in the third injection round give two relations of type (7) which are solved with respect to $03 \cdot S(i^1_{11}) + S(i^1_{33})$ and $S(i^1_{13}) + 03 \cdot S(i^1_{31})$. Two further collisions in the second column of the same round deliver two other relations which yield $03 \cdot S(i^1_{13}) + S(i^1_{31})$ and $S(i^1_{11}) + 03 \cdot S(i^1_{33})$. These four relations can be uniquely solved with respect to i^1_{11}, i^1_{33}, i^1_{13} and i^1_{31}.

At the time one arrives at the MixColumns transform in the fourth injection round, 8 bytes of I^1 are known. Let us again focus on the 0th column. Its state before the MixColumns operation is as follows:

$$\begin{pmatrix} S(03 \cdot S(f_2) + S(g_4) + c_{00} + m^3_{00}) \\ S(S(f_1) + 03 \cdot S(g_3) + c_{10}) \\ S(S(g_2) + 03 \cdot S(f_4) + c_{20} + m^3_{22}) \\ S(S(g_1) + 03 \cdot S(f_3) + c_{30}) \end{pmatrix}, \tag{8}$$

where

$$\begin{gathered} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \times \begin{pmatrix} S(i^1_{01}) \\ S(i^1_{12}) \\ S(i^1_{23}) \\ S(i^1_{30}) \end{pmatrix} \\ \text{and} \\ \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \times \begin{pmatrix} S(i^1_{03}) \\ S(i^1_{10}) \\ S(i^1_{21}) \\ S(i^1_{32}) \end{pmatrix}. \end{gathered} \tag{9}$$

The constants c_{00}, c_{10}, c_{20} and c_{30} in (8) depend on the message injections K^1 and K^2 as well as on the previously recovered state bytes. These constants can be seen random. Thus, if a collision is detected in $i^4_{00}+k^4_{00}$, the following equation

[3] These constants depend on known message injections bytes and on already recovered bytes of I^1.

holds:

$$\begin{array}{c} 02 \cdot S(03 \cdot S(f_2) + S(g_4) + c_{00} + m^3_{00}) + \\ 03 \cdot S(S(f_1) + 03 \cdot S(g_3) + c_{10}) + \\ S(S(g_2) + 03 \cdot S(f_4) + c_{20} + m^3_{22}) + \\ S(S(g_1) + 03 \cdot S(f_3) + c_{30}) + m^4_{00} \\ = \\ 02 \cdot S(03 \cdot S(f_2) + S(g_4) + c'_{00} + z^3_{00}) + \\ 03 \cdot S(S(f_1) + 03 \cdot S(g_3) + c'_{10}) + \\ S(S(g_2) + 03 \cdot S(f_4) + c'_{20} + z^3_{22}) + \\ S(S(g_1) + 03 \cdot S(f_3) + c'_{30}) + z^4_{00} \end{array} \tag{10}$$

for some appropriate Z^3, K^3, Z^4, K^4. The attacker can obtain the variables $03 \cdot S(f_2) + S(g_4)$, $S(f_1) + 03 \cdot S(g_3)$, $S(g_2) + 03 \cdot S(f_4)$ and $S(g_1) + 03 \cdot S(f_3)$, if all four bytes of the column collide. Then he has four equations of type (10) that can be solved by brute force (2^{32} operations). Another 4-byte collision in column 1 will lead to the values $02 \cdot S(f_1) + S(g_3)$, $02 \cdot S(g_2) + S(f_4)$, $S(g_1) + 02 \cdot S(f_3)$ and $S(f_2) + 02 \cdot S(g_4)$. These reveal f_j, g_j, $j = \overline{1,4}$. Note that this occurs for any pair of columns in this injection round. Thus, the attacker needs two 4-byte collisions in any of the four columns, a single 4-byte collision for a column meaning four 1-byte collisions in the same column which can occur in different messages.

After f_j, g_j, $j = \overline{1,4}$ have been recovered, the linear systems of equations (9) can be solved and uniquely deliver the rest of the variables i^1_{01}, i^1_{12}, i^1_{23}, i^1_{30}, i^1_{03}, i^1_{10}, i^1_{21} and i^1_{32}, since one deals with the invertible MixColumns transform. Thus, the whole state I^1 is recovered.

Our thorough simulations show that two collisions in a column for rounds 2 and 3 do not allow for a unique solution of the two involved unknown bytes, even if these collisions occur in different column bytes. In this case one has about two solutions, averaged over all pairs of unknown bytes for different random injections. In round 4 each of the two 4-byte collisions deliver approximately 2^3 candidates for the intermediate variables. Thus, the attacker has in average 2^8 candidates for I^1 at the end. The correct one can be identified in the next step by trying to construct a collision as described in Section 4.

Now the number of needed measurements is estimated. In injection rounds 2 and 3 at least two collisions are needed in the 0th and second columns (collisions in some two bytes of each of the both columns). In a single column two collisions have to be detected. Thus, the following probability needs to be computed:

$$\begin{array}{c} P = \Pr\{\text{at least two collisions in 4 column bytes}\} = \\ = 1 - \Pr\{A = \text{no collisions in all 4 bytes}\} - \\ - \Pr\{B = \text{exactly one collision in one of the 4 bytes}\} = \\ = 1 - P_A - P_B. \end{array} \tag{11}$$

If p is the probability that no collisions occurred in a specific column byte after N measurements, then:

$$p = \prod_{j=0}^{N-1} \left(1 - \frac{j}{256}\right) \text{ and } P_A = p^4. \tag{12}$$

Let q_i, $i = 2, \ldots, N$, be the probability that exactly one collision is detected in the i-th measurement and no collisions occurred in all the remaining measurements. Then:

$$q_i = \frac{i-1}{256} \prod_{j=0}^{N-2} \left(1 - \frac{j}{256}\right) \text{ and } P_B = 4p^3 \sum_{i=2}^{N} q_i. \tag{13}$$

In injection round 4 any two columns have to yield a 4-byte collision each. This probability can be calculated using p_N from (4) as follows:

$$P' = \binom{4}{2} p_N^8 (1 - p_N^4)^2 + \binom{4}{3} p_N^{12} (1 - p_N^4) + \binom{4}{4} p_N^{16}. \tag{14}$$

Thus, $P^4 P' \geq 1/2$ must be fulfilled for a successful attack, which is achieved for $N = 29$ with $P^4 P' \approx 0.560$. This is to be compared to 40 measurements needed for the key-recovery in [5].

4 Constructing Collisions

In this section we show how we can exploit the knowledge of the internal state in order to construct collisions for Alpha-MAC.

4.1 Some Properties of the AES Round Function

Let us take another look at Figure 1, more precisely, at the AES/Alpha-MAC round function. MixColumns is the only transformation that provides diffusion [8]. It is a linear transformation over $GF(2^8)^{16}$ and acts on groups of 4 bytes, so it may be considered as four linear transformations over $GF(2^8)^4$. Remind that SubBytes acts on individual bytes, and ShiftRows is just a permutation. As a result, we can divide the state $I^{t-1} + K^{t-1}$ into 4 groups of bytes (denote them by A_i, $i = \overline{1,4}$) and the state I^t into other 4 groups (B_i, respectively). Then the following property holds.

Observation 1 *B_i bytes are linear combinations over $GF(2^8)$ of* SubBytes-*transformed A_i bytes.*

This may be illustrated on the following scheme:

$$\begin{array}{ccc} I^{t-1} + K^{t-1} & & I^t \\ A_1 : a_{00}\ a_{11}\ a_{22}\ a_{33} & \xrightarrow{L_1 \circ S} & b_{00}\ b_{01}\ b_{02}\ b_{03} : B_1 \\ A_2 : a_{10}\ a_{21}\ a_{32}\ a_{03} & \xrightarrow{L_2 \circ S} & b_{10}\ b_{11}\ b_{12}\ b_{13} : B_2 \\ A_3 : a_{20}\ a_{31}\ a_{02}\ a_{13} & \xrightarrow{L_3 \circ S} & b_{20}\ b_{21}\ b_{22}\ b_{23} : B_3 \\ A_4 : a_{30}\ a_{01}\ a_{12}\ a_{23} & \xrightarrow{L_4 \circ S} & b_{30}\ b_{31}\ b_{32}\ b_{33} : B_4 \end{array} \tag{15}$$

Here S denotes the S-box transformation and L_i — the i-th subfunction of MixColumns. For example, byte 7 in I^t is a linear over $GF(2^8)$ function of S-boxed bytes 4, 9, 14, and 3 of $I^{t-1} + K^{t-1}$.

The following property also holds.

Observation 2 *Given any 4 bytes of a row of (15) one may compute the other 4 bytes of the row fast (4 S-box transformations and* ≈ 20 *additions and multiplications in the field).*

The proof may be easily derived by the substitution of $S(x)$ into x for all x from A_i.

4.2 How to construct a 4-1 collision

The collision attack (Section 3) gives us the internal state I^1. As a result, for every 4-byte M' we can compute J — the internal state after one round. The authors of the Alpha-MAC proved that any 128-bit J can be reached from any other internal state after 4 rounds with an appropriate choice of $M = (M^1, M^2, M^3, M^4)$. More formally, we use the following lemma.

Lemma 1 ([1]). *Given* I^1*, the state value before iteration* 1*, the map*

$$s : (M^1, M^2, M^3, M^4) \to I^5$$

from the sequence of 4 message blocks (M^1, M^2, M^3, M^4) *to the state value before iteration 5 is a bijection.*

This lemma was proved in a non-constructive way so we show how to compute the message $M = (M^1, M^2, M^3, M^4)$. A brief scheme of computing M is presented in Figure 2. Now we show how to compute unknown state values step by step.

Step 0. We already computed $I^5 = J$.

Step 1. A_2 and A_4 bytes are not modified by K^1 (the result of the injection M^1) so we can compute B_2 and B_4 bytes. Analogously for M^4.

Remark 1. Let us notice that $I^5 = J$ implies $I^4 + K^4 = I^1 + K'$ so known bytes of I^4 are exactly bytes of I^1.

Step 2. Given 12 bytes of I^4 we compute 8 bytes of I^3.

Step 3. Now we know 8 bytes of I^2 (and thus the same bytes of $I^2 + K^2$) and 8 bytes of I^3. One can easily check that we know 2 bytes from each A_i and 2 bytes from each B_i. Then we use Observation 2 and obtain all bytes of I^3 and 12 bytes of I^2.

Step 4. We use Observation 2 to derive values of last unknown bytes in I^2 and I^4.

Step 5. First we compute M^1 and M^2 using pairs (I^1, I^2) and (I^2, I^3). Secondly, we do the same to obtain M^3 and M^4.

As a result we have a 4–1 collision: $M^1||M^2||M^3||M^4$ collides with M'.

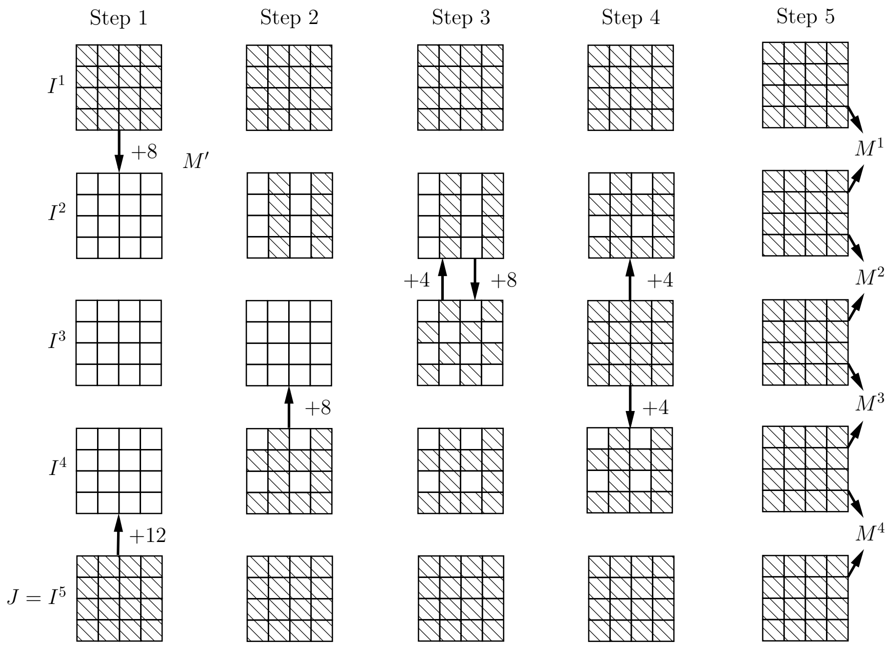

Fig. 2. Computation of the collision: $I^1 \xrightarrow[M^1||M^2||M^3||M^4]{M'} I^5$

On N–1 collisions. Collisions of type 3–1 and 2–1 may theoretically exist. Given I^1 one can reduce our construction to the search of such collisions. Due to Remark 1 we do not need to know M' to detect whether a collision exist. As a result, only some linear computations are required to check whether short collisions are possible.

The collisions that we showed in this section were not considered by designers or by [2]. It seems that in the case of AES it may be difficult to construct such collisions in less than 2^{64} steps without the knowledge of the internal state. However formally proving it seems challenging. Note also that in the case of Alred used with Feistel ciphers (ex. Triple-DES, proposed in [1]) one has to be more careful, since involutional structure of Feistel-ciphers makes it easier to find partial fixed points required in such collisions.

4.3 Selective forgery of Alpha-MAC

We have shown very efficient way to construct 4 to 1 block collisions in Alpha-MAC, if the internal state is known. We estimate the complexity of finding collisions as 2^{11} operations in $GF(2^8)$.

The way we construct these collisions allows to perform selective forgery attack on Alpha-MAC. The attack would be as follows: the attacker obtains measurements of authentication for 29 arbitrary known messages. From these he derives 2^8 candidates for the 128-bit internal state. He then picks arbitrary victim message-tag pair (M, σ). The attacker picks a message M' that he would like to authenticate (with exception of a suffix δ of 16 bytes). For each candidate of the internal state (and thus for each candidate of $E_K(0)$) and M' the attacker can evaluate the internal states I and I' of Alpha-MAC after the injection of M and M', respectively. He then computes the 16 injection bytes δ which transform I' into I. Then a pair $((M'||\delta), \sigma)$ would be a properly authenticated message-tag pair.

4.4 Implications for Pelican-MAC

Daemen and Rijmen have also proposed another AES-based MAC called Pelican [9], which is very similar to Alpha-MAC. The main difference is that in Pelican there is a single 128-bit message injection every 4 rounds instead of a 32-bit message injection every round. Pelican is also sensitive to attacks shown in this paper. The adversary first learns the internal state at some unprotected round, if such round exists. From that point the attacker has full control of the internal state: i.e. he can create arbitrary meaningful colliding messages by calculating a proper 128-bit injection at the end. Thus, one should mask the internal rounds of Pelican-MAC to hamper such attacks.

5 Experiments

As a proof of concept, we have implemented Alpha-MAC on an 8-bit microcontroller, representing the typical architecture found on current low-cost smart-cards. We opted for a PIC16F687 low power CMOS microcontroller [10], clocked at 4 MHz employing its internal oscillator and needing four internal clock cycles (1 μs) to execute an instruction, which is in between the twelve internal cycles of a standard i8051 controller and the single clock cycle an Atmel AVR processor needs for carrying out one command. Communication takes place via a serial port, while the actual power consumption is detected single-ended by means of a 240 Ω resistor inserted between the supply ground and the ground pin of the microcontroller. Care has to be taken to avoid ground loops and assure stable power supply and proper shielding of the measurement setup, for reduction of noise, as measurements with a high accuracy are required to detect collisions of bytes being processed, compared to only observing their Hamming weight.

Our implementation of the Alpha-MAC is written in PIC assembly language using ideas from [11]. Due to the RAM in the PIC being restricted to only 128 bytes, we stored the 256 bytes of the S-box in the program memory, as proposed in [12]. One pin of the microcontroller is used as a trigger output, thus easing the alignment of the data. While the PIC is computing the MAC, power traces are acquired by the 8-Bit ADC (Analog to Digital Converter) of an Agilent Infinium 54832D oscilloscope at a sampling rate of 4 GSa/s. The data is stored and evaluated on a PC with Matlab.

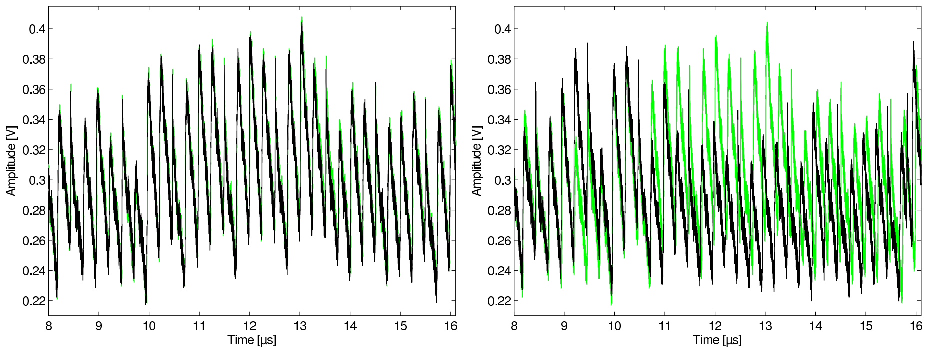

Fig. 3. Power consumption curves for equal (left) and different (right) bytes

In order to detect the collisions at the end of the i-th injection round, we identified four distinct sequences of instructions which looked valuable for a power analysis in the relevant parts of the $(i{+}1)$-th injection round. The following code uses the target byte directly:

```
...
;SubBytes, target byte in accu
  movf   A1,w           ;
  call   SBOX           ;
  movwf A1              ;
...
;ShiftRows (for rows 1,2 and 3 only)
  movf   A1,w           ;
  movwf A2              ;
...
;MixColumns
  movf   A2,w           ;
  xorwf R1,w            ;
...
  movf   A2,w           ;
  xorwf R2,f            ;
...
```

Note that the code between the listed fragments delivers random states of the accumulator register before each sequence. Moreover, values R1 and R2 in the MixColumns transform can be seen as uncorrelated with the target byte.

The similarity of two power traces $P_a(t)$ and $P_b(t)$ was detected for all discrete points in time t_i belonging to the above mentioned code fragments by finding a minimum of $\sum (P_a(t_i) - P_b(t_i))^2$.

The result is presented in Figure 3, where two power traces out of the section involving the S-box lookup are compared. On the left side, the indices for the table lookups are equal to each other and a difference between the power consumption is almost not noticeable, while on the right side two distinct table lookups exhibit an obvious difference.

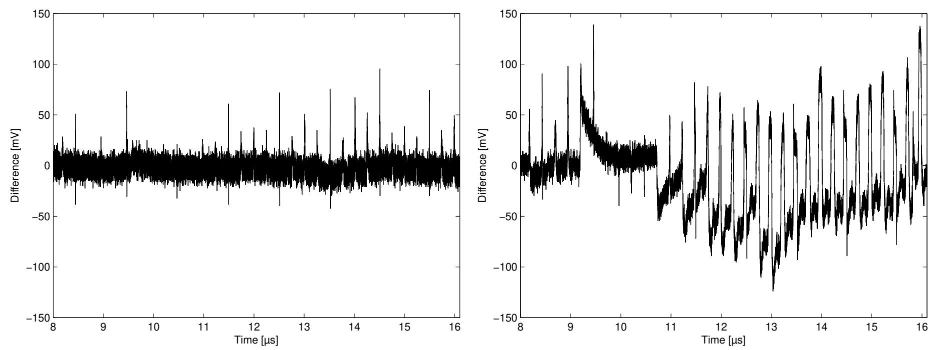

Fig. 4. Difference curves in case a collision is detected (left) and no collision is detected (right)

The left part of Figure 4 shows the difference curve, i.e., $P_a(t) - P_b(t)$, when a correlation is detected. The right half of Figure 4 depicts the difference curve in case the measured power consumption curves do not correlate.

Note that our collision attack, as any other power analysis attack, can be significantly hampered or even made impossible by minimizing the signal-to-noise ratio, using sound masking techniques [13], [14] or advanced clock randomizing methods [15]. However, the collision attack is likely to break through basic time randomization countermeasures such as simple random wait states, which can be detected using SPA or alignment techniques.

6 Conclusions

In this paper we showed that the Alred construction and its AES-based instance Alpha-MAC can be very efficiently attacked using side-channel collision attacks, even if the keyed rounds are masked. We are able to determine the whole Alpha-MAC internal state with just 29 measurements in the *known* message scenario instead of 40 measurements in the *chosen* plaintext scenario by mounting an enhanced side-channel collision attack. Moreover, since the internal hash of Alpha-MAC is not collision resistant, from the knowledge of internal state one can construct collisions in Alpha-MAC with negligible time and memory complexity. The way we construct these collisions allows to perform selective forgery of arbitrary messages with an exception of a 128-bit suffix which is calculated to create a collision. We describe a new 4 to 1 block collision finding algorithm. Our attacks demonstrate that the internal unkeyed rounds of Alpha-MAC should be also protected against power analysis.

Acknowledgments. We would like to thank Christof Paar and Kerstin Lemke-Rust for fruitful discussions and the anonymous referees for constructive suggestions which helped us to improve the paper. Dmitry Khovratovich is supported by PRP "Security & Trust" grant of the University of Luxembourg.

References

1. Daemen, J., Rijmen, V.: A new MAC construction Alred and a Specific Instance Alpha-MAC. In: FSE'05. Volume 3557 of LNCS., Springer (2005) 1–17
2. Huanga, J., Seberry, J., Susilo, W.: On the internal structure of Alpha-MAC. In: Vietcrypt'06. Volume 4341 of LNCS., Springer (2006)
3. Schramm, K., Wollinger, T., Paar, C.: A New Class of Collision Attacks and Its Application to DES. In: FSE'03. Volume 2887 of LNCS., Springer (2003) 206–222
4. Ledig, H., Muller, F., Valette, F.: Enhancing Collision Attacks. Volume 3156 of LNCS., Springer (2004) 176–190
5. Schramm, K., Leander, G., Felke, P., Paar, C.: A Collision-Attack on AES: Combining Side Channel- and Differential-Attack. In: CHES'04. Volume 3156 of LNCS., Springer (2004) 163–175
6. Osvik, D.A., Shamir, A., Tromer:, E.: Cache Attacks and Countermeasures: The Case of AES. In Pointcheval, D., ed.: Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006. Volume 3860 of LNCS., Springer-Verlag (2006)
7. Handschuh, H., Preneel, B.: Blind differential cryptanalysis for enhanced power attacks. In: SAC'06. LNCS, Springer-Verlag (2006)
8. Daemen, J., Rijmen, V.: The Design of Rijndael: AES — The Advanced Encryption Standard. Springer (2002)
9. Daemen, J., Rijmen, V.: The Pelican MAC Function. Available at `http://eprint.iacr.org/2005/088.pdf` (2005)
10. Microchip: PIC16F687 Microcontroller Data Sheet (2007)
11. Lechner, J., Tatzgern, M.: Efficient implementation of the AES Encryption Algorithm for Smart-cards. Available at `www.iaik.tugraz.at` (2004)
12. D'Souza, S.: AN556 - Implementing a Table Read. Technical report, Microchip Technology Inc. (2000) Application Note.
13. Oswald, E., Mangard, S., Pramstaller, N., Rijmen, V.: A Side-Channel Analysis Resistant Description of the AES S-box. In: FSE'05. Volume 3557 of LNCS., Springer (2005)
14. Oswald, E., Schramm, K.: An efficient masking scheme for aes software implementations. In: WISA'05. Volume 3786 of LNCS., Springer (2006)
15. Herbst, C., Oswald, E., Mangard, S.: An AES Implementation Resistant to Power Analysis Attacks. In: ACNS'06. Volume 3989 of LNCS., Springer (2006)