Fast Key Recovery Attack on ARMADILLO1 and Variants

Pouyan Sepehrdad, Petr Sušil, Serge Vaudenay

EPFL, Lausanne, Switzerland
{pouyan.sepehrdad, petr.susil, serge.vaudenay}@epfl.ch

Abstract. The ARMADILLO cryptographic primitive is a multi-purpose cryptographic primitive for RFID devices proposed at CHES'10. The main purpose of the primitive is to provide a secure authentication in a challenge-response protocol. It has two versions, named ARMADILLO (subsequently denoted by ARMADILLO1) and ARMADILLO2. However, we found a fatal weakness in the design which allows a passive attacker to recover the secret key in polynomial time, of ARMADILLO1 and some generalizations. We introduce some intermediate designs which try to prevent the attack and link ARMADILLO1 to ARMADILLO2. Considering the fact that the attack against ARMADILLO1 is polynomial, this brings about some concerns into the security of the second version ARMADILLO2, although it remains unbroken so far.

1 Introduction

ARMADILLO is a hardware oriented multi-purpose cryptographic primitive presented at CHES'10 [2]. It was built for RFID applications. It can be used as a PRF/MAC, e.g. for a challenge-response protocol as a MAC, and also as a hash function for digital signatures, or a PRNG for making a stream cipher. It has two versions, named ARMADILLO (subsequently denoted by ARMADILLO1) and ARMADILLO2.

During the review process of CHES'10 we found an attack against ARMADILLO1 and its variants. The ARMADILLO2 includes a quick fix to resist it. The attack and its variants are presented in this paper.

To fix the vulnerability of ARMADILLO1 and simultaneously shrink the design, we define multiple intermediate versions of ARMADILLO and we investigate their security with respect to the original attack and illustrate that they are still vulnerable to a key recovery or a forgery attack. Although our attack is not applicable against ARMADILLO2, the step by step approach in the design of other variants would give a concern behind the security of ARMADILLO2. These intermediate designs reveal that the security bounds on ARMADILLO2 might be insufficient.

We introduce a generalized version ARMADILLOgen and we explain when the key recovery or forgery attack is possible. Finally, we come to the definition of ARMADILLO2.

The attacks we show have always complexity polynomial in the size of input. Specifically, the attack against ARMADILLO1 has complexity $O(k^2 \log k)$ and it can be performed "by hand", as the actual key recovery algorithm is very simple.

1.1 Related work

In [1] the authors found an attack against ARMADILLO2 based on parallel matching. The key recovery attack against FIL-MAC application of ARMADILLO2-A and ARMADILLO2-E using single challenge-response pair is 2^7 and 2^{18} times faster than exhaustive search respectively. The techniques presented in our paper may help to reduce the time complexity if the attacker uses multiple samples.

2 Description of ARMADILLO

ARMADILLO relies on data-dependent bit transpositions. Given a bitstring x with bit ordering $x = (x_\ell \| \cdots \| x_1)$, fixed permutations σ_0 and σ_1 over the set $\{1,2,\ldots,\ell\}$, a bit string s, a bit $b \in \{0,1\}$ and a permutation σ, define $x_{\sigma_s} = x$ when s has length zero, and $x_{\sigma_{s\|b}} = x_{\sigma_s \circ \sigma_b}$, where x_σ is the bit string x transposed by σ, that is,

$$x_\sigma = (x_{\sigma^{-1}(\ell)} \| \cdots \| x_{\sigma^{-1}(1)})$$

The function $(s,x) \mapsto x_{\sigma_s}$ is a data-dependent transposition of x. The function $s \mapsto \sigma_s$ can be seen as a particular case of the general semi-group homomorphism from $\{0,1\}^*$ to a group G.

Notations. Throughout this document, $\|$ denotes the concatenation of bitstrings, $\oplus$ denotes the bitwise XOR operation, $\overline{x}$ denotes the bitwise complement of a bitstring x; we assume the little-endian numbering of bits, such as $x = (x_\ell \| \cdots \| x_1)$.

In this section, we give the description of two variants ARMADILLO1 and ARMADILLO2. Then, we introduce a common generalized version ARMADILLOgen and show how it relates to all versions. We show the attack against ARMADILLOgen for many different choices of parameters.

2.1 ARMADILLO1

ARMADILLO1 maps an initial value C and a message block U to two values (see Fig. 1)
.

$$(V_C, V_T) = \mathsf{ARMADILLO1}(C,U)$$

ARMADILLO1 works based on a register Xinter. By definition, C and V_C are of c bits, V_T as well as each block U_i are of m bits, Xinter is of $k = c+m$ bits. ARMADILLO1 is defined by integer parameters c, m, and two fixed permutations σ_0 and σ_1 over the set $\{1,2,\ldots,2k\}$. Concretely, we consider $m \geq 40$ and $k = c+m$. To initialize ARMADILLO1, Xinter is set to $C\|0^m$ where 0^m is a null padding block, and C is an initial value. ARMADILLO1 works as follows (Fig. 1).

1: in the i-th step, replace the rightmost m-bit block of Xinter by the block U_i;
2: set a $\ell = 2k$ bits register $x = \overline{\mathsf{Xinter}}\|\mathsf{Xinter}$;

3: x undergoes a sequence of bit permutations which we denote by P. The output of this sequence of bit permutations is truncated to the rightmost k bits, denoted S, by

$$S = \mathsf{tail}_k((\overline{\mathsf{Xinter}}\|\mathsf{Xinter})_{\sigma_{\mathsf{Xinter}}})$$

4: set Xinter to the value of $S \oplus \mathsf{Xinter}$.
5: after processing the last block U_n, take $(V_C\|V_T) = \mathsf{Xinter}$ as the output.

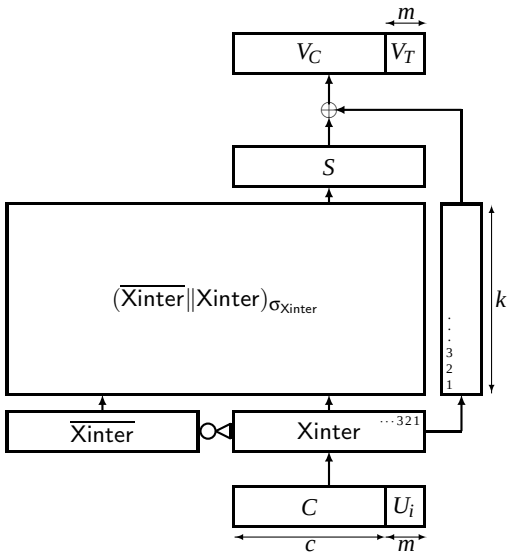

Fig. 1. Scheme of ARMADILLO1.

2.2 ARMADILLO2

For completeness, we now provide the description of ARMADILLO2 [2] here. The ARMADILLO2 is mostly based on ARMADILLO1b (be defined later) with an additional pre-processing mechanism. As the reader see later in the paper, the pre-processing prevents our attack. We note that the pre-processing step outputs a sequence of bits that defines the data dependent permutation and ensures that the data dependent permutation σ_{Xinter} cannot be easily controlled by the attacker (see Fig. 2).

1: in the i-th step, replace the rightmost m-bit block of Xinter by the block U_i;
2: set a $\ell = k$ bits register $x = \mathsf{Xinter}$;
3: x undergoes a sequence of bit permutations, σ_0 and σ_1 and a constant γ addition, which we denote by P. In fact, P maps a bitstring of m bits and a vector x of k bits into another vector of k bits as $P(s\|b,x) = P(s, x_{\sigma_b} \oplus \gamma)$, where $b \in \{0,1\}$ and x_{σ_b} is a permutation of bits of x (transposition). The output of this sequence of k bit permutations and constant addition is denoted $Y = P(U_i, x)$. We call this step pre-processing, since it is used to define the permutation for the consequent step.

4: x undergoes a sequence of bit permutations and constant addition P defined by Y. The output of this sequence of k bit permutations and constant addition is denoted $S = P(Y,x)$.
5: set Xinter to the value of $S \oplus$ Xinter.
6: after processing the last block U_n, take $(V_C \| V_T)$ as the output.

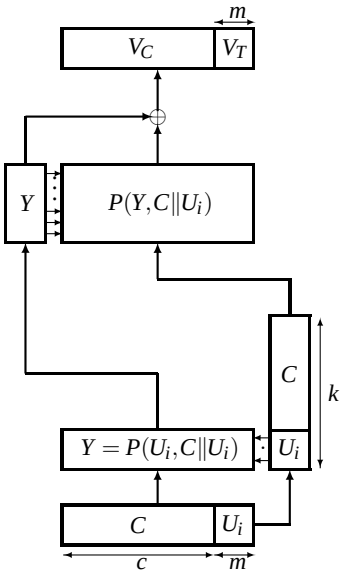

Fig. 2. Scheme of ARMADILLO2.

3 General ARMADILLOgen Algorithm

We define various intermediate versions of ARMADILLO. These intermediate versions show the relation between ARMADILLO1 and ARMADILLO2 and give a security concern on ARMADILLO2. We explain step by step how the weakness in the design of ARMADILLO1 relate to a possible weaknesses in design of ARMADILLO2.

All these versions are based on data-dependent permutation P. They all can be covered under ARMADILLOgen as a parametrized version of distinct variants, and by setting corresponding parameters we obtain ARMADILLO1, ARMADILLO1b, ARMADILLO1c, ARMADILLO1d and ARMADILLO2. We show an attack against ARMADILLOgen for some choices of parameters.

ARMADILLOgen is defined as

$$\text{ARMADILLOgen}(X) = T_4(P(T_1(X), T_2(X)), X)$$

where

$$P(s\|b,Y) = P(s, T_3(b,Y))$$
$$P(\lambda,Y) = Y$$

λ denotes the empty string, T_1, T_2, and T_4 are some linear functions, and T_3 in its most general form is

$$T_3(b,Y) = L(Y)_{\sigma_b} \oplus \gamma$$

where L is linear and γ is a constant.

Then, ARMADILLO1 is defined as ARMADILLOgen for

$$T_1(X) = X$$
$$T_2(X) = \overline{X}\|X$$
$$T_3(b,X) = X_{\sigma_b}$$
$$T_4(X,Y) = \mathsf{tail}_k(X) \oplus Y$$

ARMADILLO2 is defined as ARMADILLOgen for

$$T_1(X) = P(\mathsf{tail}_m(X), X)$$
$$T_2(X) = X$$
$$T_3(b,X) = X_{\sigma_b} \oplus \gamma$$
$$T_4(X,Y) = X \oplus Y$$

3.1 ARMADILLO1b: Shrinking the Xinter Register

The ARMADILLO1b is a compact version of ARMADILLO1 which prevents the preservation of Hamming weight by adding a constant. However, it does not prevent the attack against ARMADILLO1. According to [2], the ARMADILLO1 design prevents a distinguishing attack based on constant Hamming weight by having the double sized internal register and the final truncation, assuming the output of P transposition looks pseudo-random. We see later in this paper (see section 4) that this proof does not hold in standard attack model and ARMADILLO1 can be broken in polynomial time. First, we define ARMADILLO1b and then demonstrate an attack against this version and explain how the same attack can be used against ARMADILLO1.

ARMADILLO1b is defined as ARMADILLOgen for

$$T_1(X) = X$$
$$T_2(X) = X$$
$$T_3(b,X) = X_{\sigma_b} \oplus \gamma$$
$$T_4(X,Y) = X \oplus Y$$

In the design of ARMADILLO1b the state size is reduced to k bits to save more gates. So, there is only the register Xinter and not its complement, and there is no truncation. To avoid Hamming weight preservation, after each permutation there is an XOR of the current state with a constant γ (see Fig. 3).

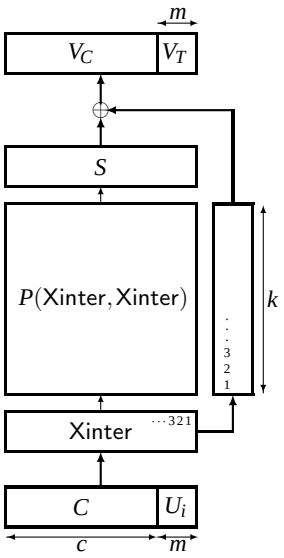

Fig. 3. Scheme of ARMADILLO1b.

3.2 ARMADILLO1c: Adding a Linear Layer in T_3

To investigate whether a more complex layer in T_3 can prevent the attack on ARMADILLO1b we define ARMADILLO1c. It is defined as ARMADILLOgen for

$$T_1(X) = X$$
$$T_3(b,X) = L(X)_{\sigma_b} \oplus \gamma$$

for a linear transformation L, with arbitrary linear T_2 and T_4.

3.3 ARMADILLO1d: Adding a Fixed Transposition in T_1

We will see later that ARMADILLO1c is still vulnerable. To prevent the attack on ARMADILLO1c, a fixed transposition was added in T_1 to mix the bits of the secret and the challenge. ARMADILLO1d is defined as ARMADILLOgen for

$$T_1(X) = X_\pi$$
$$T_3(b,X) = X_{\sigma_b} \oplus \gamma$$

for a fixed permutation π and with arbitrary linear T_2 and T_4.

4 Key Recovery Attack against ARMADILLO1 and ARMADILLO1b

In this section, we describe an attack against two versions of ARMADILLO. We first explain the attack on ARMADILLO1b and then setting $\gamma = 0$ and extending the initial

state to $(\overline{\mathsf{Xinter}}\|\mathsf{Xinter}) = (\overline{C\|U}\|C\|U)$, the same attack can be directly used against ARMADILLO1.

Since ARMADILLO has more than one applications, we just briefly explain how it is deployed in the challenge-response application. We refer the reader to [2] for more details. The objective is to have a fixed input-length MAC. Suppose that C is a secret and U is a one block challenge. The value V_T is the response or the authentication tag. We write

$$V_T = \mathsf{ARMADILLO}(C,U)$$

As can be seen from the description of the algorithm, there is no substitution layer. This means that for a fixed key C the permutation σ_C is fixed (but unknown). As we see later in the paper, it can be easily recovered. For the attack it suffices to recover the mapping σ_C of a single index, for instance we recover $\sigma_C(j) = n$ for some value j. If we can recover the mapping $\sigma_C(j)$, we than take challenges U_i so that j-th bit of $P(U_i,C\|U_i)$ contains different bits of the key. This allows us to recover the secret key from literally reading the key from the output of ARMADILLO1b. We also show that the attack can be extended to other scenarios, or can be changed to forgery attack if the key recovery is not possible. More precisely, we consider

$$\begin{aligned} T_1(X) &= X \\ T_3(b,X) &= X_{\sigma_b} \oplus \gamma \end{aligned}$$

with arbitrary linear T_2 and T_4. This includes ARMADILLO1 and ARMADILLO1b.

The attack is based on the fact that a bit permutation is linear with respect to XOR operation, i.e., for a permutation σ, X and Y be two vectors, we have $(X \oplus Y)_\sigma = X_\sigma \oplus Y_\sigma$.

Lemma 1. *For any T_3, C, and U, we have*

$$P(C\|U,C\|U) = P(C,P(U,C\|U))$$

Proof. We easily prove it by induction on the size of C. □

Lemma 2. *For $T_3(b,X) = X_{\sigma_b} \oplus \gamma$, there exists a function $f : 2^{|X|} \to 2^{|X|}$ such that for any $Y = (y_k\| \ldots \|y_1)$ and X, we have*

$$P(Y,X) = X_{\sigma_Y} \oplus f(Y)$$

Proof. Let rewrite

$$P(Y,X) = \left(\left(\left(X_{\sigma_{y_1}} \oplus \gamma \right)_{\sigma_{y_2}} \oplus \gamma \right)_{\sigma_{y_3}} \oplus \gamma \ldots \right)_{\sigma_{y_k}} \oplus \gamma$$

Let define the *prefix* of Y as

$$\text{prefix}(Y) = \{Y_j;\ Y_j = (y_k\| \ldots \|y_j), 1 \le j \le k\}$$

Thus, P can be rewritten as

$$P(Y,X) = (X \oplus \gamma)_{\sigma_Y} \oplus \gamma \oplus \bigoplus_{p \in \text{prefix}(Y)} \gamma_{\sigma_p} = X_{\sigma_Y} \oplus P(Y,0)$$

□

Now we apply the above results to ARMADILLOgen with $T_1(X) = X$ and $T_3(b,X) = X_{\sigma_b} \oplus \gamma$.

$$\begin{aligned}
\text{ARMADILLOgen}(C\|U) &= T_4(P(C\|U, T_2(C\|U)), C\|U) \\
&= T_4(P(C, P(U, T_2(C\|U))), C\|U) \\
&= T_4(P(C, (L_U(C)_{\sigma_U} \oplus f(U))), C\|U) \\
&= T_4\left((L_U(C)_{\sigma_U} \oplus f(U))_{\sigma_C} \oplus f(C), C\|U\right)
\end{aligned}$$

where $L_U(C) = T_2(C\|U)$ and $f(U)$ is given by Lemma 2. The first equality is coming from the definition, the second from Lemma 1 and the last two from Lemma 2. So, we can write

$$\text{ARMADILLOgen}(C\|U) = L\left((L_U(C)_{\sigma_U} \oplus f(U))_{\sigma_C} \oplus g(U) \oplus h(C)\right)$$

for some linear function L and some functions g and h. For all the variants we consider, L is either the identity function or consists of dropping a few bits. For ARMADILLO1b and ARMADILLO1 the function $h(C) = f(C) \oplus (C\|0^m)$, $g(U) = (0^c\|U)$. Similarly, $L(X) = X$ and $L(X) = \text{tail}_k(X)$ respectively.

In what follows, we consider an arbitrary i and take a vector e_i such that $e_i \cdot L(X) = X[i]$, i.e, the i-th bit of register X. So, we obtain

$$e_i \cdot \text{ARMADILLOgen}(C\|U) = (L_U(C)_{\sigma_U} \oplus f(U))_{\sigma_C^{-1}(i)} \oplus g(U)_i \oplus h(C)_i$$

Clearly, there exists a $j = \sigma_C^{-1}(i)$ such that

$$e_i \cdot \text{ARMADILLOgen}(C\|U) \oplus g(U)_i = L_U(C)_{\sigma_{U_t}^{-1}(j)} \oplus f(U)_j \oplus h(C)_i \qquad (1)$$

In chosen-input attacks against the PRF mode, we assume that the adversary can compute

$$e_i \cdot \text{ARMADILLOgen}(C\|U)$$

for a chosen U and a secret C. In the challenge-response application, we only have access to V_T, but in all considered variants, e_i has Hamming weight one, so we just need to select i so that this bit lies in the V_T window. We introduce an attack (see Fig. 4) which only needs this bit of the response for $n = k \log k$ queries. This algorithm has complexity $O(k^2 \log k)$ to recover the secret C (also see Fig. 5). In fact, the attacker can simply recover the permutation $Y = P(U_i, \text{Xinter})$, since she has control over U_i's. Now, her goal is to find out how $P(C,Y)$ maps the index j to i. The goal of the algorithm is to find this mapping and recovers C. It is exploiting the fact that fixing the i, then $h(C)_i$ is fixed for all challenges and the left side of Eq. (1) can be computed directly by the adversary. Then, it recovers C by solving an overdefined linear system of equations and check it has a solution. If so, it checks whether the recovered C is consistent with other samples.

1: Pick a random i from 1 to m.
2: **for** t from 1 to $n = k \log k$ **do**
3: collect challenge-response pair $(U_t, e_i \cdot \mathsf{ARMADILLOgen}(C \| U_t))$
4: compute $b_t = e_i \cdot \mathsf{ARMADILLOgen}(C \| U_t) \oplus g(U_t)_i$.
5: **end for**
6: **for** j from 1 to ℓ **do**
7: **for** each $\beta \in \{0,1\}$ **do**
8: set $h(C)_i = \beta$.
9: **for** t from 1 to n **do**
10: compute $L_{U_t}(C)_{\sigma_{U_t}^{-1}(j)} = b_t \oplus f(U_t)_j \oplus \beta$ for all c bits.
11: **end for**
12: solve the system of n linear equations $L_{U_t}(C)_{\sigma_{U_t}^{-1}(j)}$
13: **if** no solution **then**
14: **break**
15: **end if**
16: derive C
17: **if** C is consistent with samples **then**
18: output C.
19: **end if**
20: **end for**
21: **end for**

Fig. 4. The key recovery algorithm against ARMADILLO1 and ARMADILLO1b.

Attack complexity. The first **for** loop runs ARMADILLO algorithm $k \log k$ times . The second loop runs ℓ times where $\ell = 2k$ for ARMADILLO1 and $\ell = k$ for ARMADILLO1b. We perform up to $2k \log k$ simple arithmetic operations in the second loop to compute values $L_{U_t}(C)_{\sigma_{U_t}^{-1}(j)}$. Solving the system of n linear equation requires $O(n^3)$ in general case. However, in the case of ARMADILLO1 and ARMADILLO1b every line contains only one variable of secret C, which comes from the Lemma 2. As we have $k \log k$ equations in c variables, if the mapping $i \to j$ is not guessed correctly we have high probability to obtain contradiction on line 13. So overall, we have complexity of $O(k^2 \log k)$ for attacking both ARMADILLO1 and ARMADILLO1b.

Probability of success. We first choose randomly $k \log k$ challenges U_t and compute $\mathsf{ARMADILLOgen}(C \| U_t)$. That is because, according to *coupon collector problem* [3] the expected number of challenges so that every bit of C is mapped to i-th bit of output is $k \log k$. Therefore, among $k \log k$ challenges all the bits of challenge and all the bits of secret key are mapped to a single bit of the output. The attacker can derive equation for the j-th bit of $P(U_t, C \| U_t)$, and for $k \log k$ distinct challenges U_t the set of equations will have full rank. These equations do not change through the fixed mapping σ_C, only the constant term might change due to term $P(C,0)$. Therefore if the attacker guess $j \to i$ correctly, the set of $k \log k$ equations in c variables has a solution, otherwise the set of $k \log k$ equations in c variables has no solution with probability at least $1 - 2^{-n}$.

Failure of the previous attack. The previously mentioned attack would fail, if the permutations σ_0, σ_1 map bit indices in the set $[1,m]$ to the set $[1,m]$, i.e., $\sigma_0[1..m] = [1..m]$

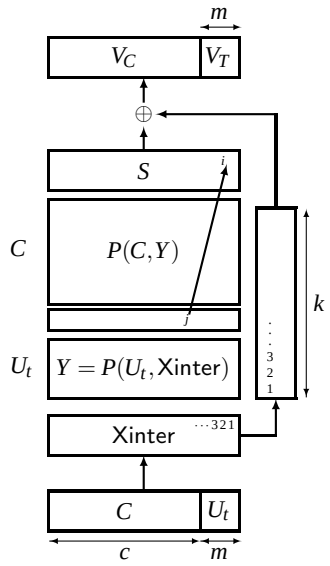

Fig. 5. Scheme of the key recovery algorithm against ARMADILLO1 and ARMADILLO1b.

and $\sigma_1[1..m] = [1..m]$. So, it might be speculated that picking such permutations in the design makes the cryptosystem secure. However, the ARMADILLO with such permutations is vulnerable to a simple forgery attack.

Let remind the decomposition

$$\begin{aligned}\mathsf{ARMADILLO1b}(C\|U) &= P(C\|U, C\|U) \oplus (C\|U)\\ &= P(C, P(U, C\|U)) \oplus (C\|U)\\ &= P(C, (L(C)_{\sigma_U} \oplus f(U))) \oplus g(U)\\ &= (L(C)_{\sigma_U} \oplus f(U))_{\sigma_C} \oplus g(U) \oplus h(C)\end{aligned}$$

Given m challenges which form a linearly independent system, we can compute the response by solving the set of these linear equations.

5 Attack Extension with Linear Layer in T_3 (ARMADILLO1c)

T_3 function is very simple in the previous versions. The first attempt to prevent the previous attack is to use a more complex layer but still linear and check whether it prevents the attack. We define another intermediate version and call it ARMADILLO1c. Then, we show that only adding a linear layer L in $T_3(b,X) = L(X)_{\sigma_b} \oplus \gamma$ would not prevent the attack.
Let L be a linear transformation. This attack requires $O(k)$ challenges and three Gaussian eliminations which require $O(k^3)$ operations. We define the new ARMADILLO1c as follows.

$$\mathsf{ARMADILLO1c}(C\|U) = T_4(P(C\|U, T_2(C\|U)), (C\|U))$$

where

$$P(s\|b, Y) = P(s, L(Y)_{\sigma_b} \oplus \gamma)$$
$$P(\lambda, Y) = Y$$

We build a system of equations, where

$$\begin{aligned}
\mathsf{ARMADILLO1c}(C\|U_j) &= T_4\left(P(C\|U_j, T_2(C\|U_j)), C\|U_j\right) \\
&= T_4\left(P(C, P(U_j, T_2(C\|U_j))), C\|U_j\right) \\
&= T_4\left(P(C, P(U_j, T_2(C\|0) \oplus T_2(0\|U_j))), C\|U_j\right) \\
&= T_4\left(P(C, P(U_j, T_2(C\|0)) \oplus P(U_j, T_2(0\|U_j))), C\|U_j\right) \\
&= T_4\left(L_C(L_{U_j}(C) \oplus \gamma_j) \oplus P(C, 0), C\|U_j\right)
\end{aligned}$$

where $L_{U_j}(C) = P(U_j, T_2(C\|0))$ and $\gamma_j = P(U_j, T_2(0\|U_j))$.

We use the fact that a set of $ck+1$ equations $L_{U_j}(C)$ is linearly dependent. Using this we can bypass the unknown mapping L_C. We can find a set J of equations whose sum is 0. Let ε be the parity of the cardinality of J. We obtain

$$\bigoplus_{j\in J} \mathsf{ARMADILLO1c}(C\|U_j) = T_4\left(L_C\left(\bigoplus_{j\in J}\gamma_j\right) \oplus \varepsilon P(C, 0), \varepsilon C\|\bigoplus_{j\in J} U_j\right)$$

Using the above expression with several J's, we can recover linear mapping L_C and $P(C, 0)$. Using the knowledge of $L_C(X)$ we recover $P(C, 2^i)$ for $0 < i < k$. We do this by Gaussian elimination on values $\left(\bigoplus_{j\in J}\gamma_j\right)$. Using $P(C, 2^i)$ for $0 \le i < k$, we can recover $L_{U_j}(C)$ as follow.

$$\begin{aligned}
\mathsf{ARMADILLO1c}(C\|U_j) &= T_4\left(L_C(L_{U_j}(C) \oplus \gamma_j) \oplus P(C, 0), C\|U_j\right) \\
&= T_4\left(L_C(L_{U_j}(C)), C\|0\right) \oplus T_4\left(L_C(\gamma_j) \oplus P(C, 0), 0\|U_j\right)
\end{aligned}$$

Therefore, we can compute

$$T_4\left(L_C(L_{U_j}(C)), C\|0\right) = \mathsf{ARMADILLO1c}(C\|U_j) \oplus T_4\left(L_C(\gamma_j) \oplus P(C, 0), 0\|U_j\right)$$

Let consider $U_i \neq U_j$, we have

$$\begin{aligned}
\Delta(U_i, U_j) =& T_4\left(L_C(L_{U_i}(C)), C\|0\right) \oplus T_4\left(L_C(L_{U_j}(C)), C\|0\right) \\
=& T_4\left(L_C(L_{U_i}(C) \oplus L_{U_j}(C)), 0\right) \\
=& \mathsf{ARMADILLO1c}(C\|U_i) \oplus T_4\left(L_C(\gamma_i) \oplus P(C, 0), 0\|U_i\right) \\
& \oplus \mathsf{ARMADILLO1c}(C\|U_j) \oplus T_4\left(L_C(\gamma_j) \oplus P(C, 0), 0\|U_j\right)
\end{aligned}$$

Hence, we obtain

$$L_{U_i}(C) \oplus L_{U_j}(C) = L_C^{-1}(T_4^{-1}(\mathsf{ARMADILLO1c}(C\|U_i) \oplus T_4(L_C(\gamma_i) \oplus P(C,0),0\|U_i) \\ \oplus \mathsf{ARMADILLO1c}(C\|U_j) \oplus T_4(L_C(\gamma_j) \oplus P(C,0),0\|U_j),0))$$

Since $L_{U_i}(C)$ is a known transformation linear in C we can recover the secret key by solving the set of linear equations.

6 Attack Extension with a Fixed Transposition in T_1 (ARMADILLO1d)

6.1 Case with no general T_2 and T_4

The previous attack can be prevented by using an S-box layer. However, if the underlying permutation P is not predictable then we can not apply the aforementioned attack. We used $P(U,C\|U)$ to generate linear equations, and then guess the mapping $P(C,Y)$. So, an attempt is to mix bits of C and U. But then, we show that even though we do not know the secret parts of permutation since these parts are fixed, we can guess them one by one. We design a version called ARMADILLO1d that first applies a fixed permutation π on the first register, i.e., T_1 is a transposition. Then, we show that setting T_1 to be a transposition does not prevent a forgery attack. ARMADILLO1d is defined as follows.

$$\mathsf{ARMADILLO1d}(C\|U) = T_4(P((C\|U)_\pi, T_2(C\|U)),(C\|U))$$

where

$$P(s\|b,Y) = P(s,Y_{\sigma_b} \oplus \gamma)$$
$$P(\lambda,Y) = Y$$

We first consider a simple case for T_1 when bits of challenge U form an interval (see Fig. 6), and T_2 is identity and $T_4(X,Y) = X \oplus Y$. Later, we extend the attack to a general transposition T_1, T_2 and T_4.

$$\mathsf{ARMADILLO1d}(C_1\|C_2\|U) = P(C_1\|U\|C_2,C_1\|C_2\|U) \oplus (C_1\|C_2\|U)$$

Let denote $X = (C_1\|C_2\|U)$. We have

$$\begin{aligned}\mathsf{ARMADILLO1d}(C_1\|C_2\|U) &= P(C_1\|U\|C_2,X) \oplus X \\ &= P(C_1\|U,P(C_2,X)) \oplus X \\ &= P(C_1,P(U,P(C_2,X))) \oplus X\end{aligned}$$

Concentrating on an arbitrary output bit n and using Lemma 2, we obtain

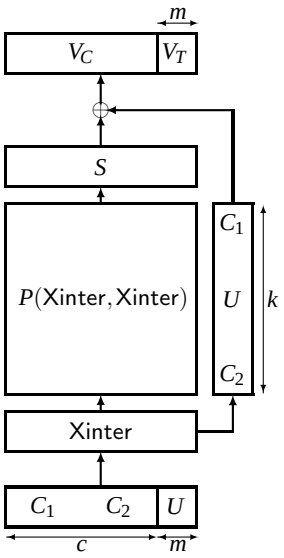

Fig. 6. The compression function of ARMADILLO1d

$$
\begin{aligned}
\mathsf{ARMADILLO1d}(C_1\|C_2\|U)[n] &= P(C_1,P(U,P(C_2,C_1\|C_2\|U)))[n] \\
&\quad \oplus (C_1\|C_2\|U)[n] \\
&\overset{t=\sigma_{C_1}^{-1}(n)}{=} P(C_1,0)[n] \oplus P(U,P(C_2,C_1\|C_2\|U))[t] \\
&\quad \oplus (C_1\|C_2\|U)[n] \\
&\overset{l=\sigma_{U}^{-1}(t)}{=} P(C_1,0)[n] \oplus P(U,0)[t] \oplus P(C_2,C_1\|C_2\|U)[l] \\
&\quad \oplus (C_1\|C_2\|U)[n] \\
&\overset{i=\sigma_{C_2}^{-1}(l)}{=} P(C_1,0)[n] \oplus P(U,0)[t] \oplus P(C_2,0)[l] \\
&\quad \oplus (C_1\|C_2\|U)[i] \oplus (C_1\|C_2\|U)[n]
\end{aligned}
$$

Re-arranging the above expression, for $a_l = P(C_2,0)[l]$ and $b_n = P(C_1,0)[n]$ we obtain

$$\mathsf{ARMADILLO1d}(C_1\|C_2\|U)[n] \oplus g(U)[t] \oplus X[n] \oplus X[i] = (a_l \oplus b_n) \tag{2}$$

We follow Fig. 7. for the attack scenario. Let $n = \sigma_{C_1}(t)$ and $l = \sigma_{C_2}(i)$, where both permutations are unknown. Set a value q to be determined later. We group $k.q$ distinct challenges as follows: we put the challenge U_j in group $\mathcal{G}_{l\to t}$ if $\sigma_{U_j}(l) = t$. In fact, a challenge appears in k groups. These are groups $\mathcal{G}_{1\to\sigma_{U_j}(1)}$, $\mathcal{G}_{2\to\sigma_{U_j}(2)}$, $\ldots$, $\mathcal{G}_{k\to\sigma_{U_j}(k)}$.

We obtain k^2 groups with approximately q challenges in each. The right hand side of equation (2) is fixed for all challenges in the same group, since $\sigma_U(l) = t$ for all $U \in \mathcal{G}_{l\to t}$. Hence the left hand side should be fixed for all challenges in the same group

as well, since neither a_l not b_n changes. We deploy this property and use the following algorithm to filter out "bad groups" and recover relations which allow us to forge the response. Since C_1 and C_2 are fixed, $\sigma_{C_2}(i)$ and $\sigma_{C_1}^{-1}(n)$ are fixed for (i,n) fixed. So, $\sigma_{C_2}(i)$ for $i \in [1,m]$ can only map to m distinct positions, therefore l can have only m possibilities out of k. The same is true for $\sigma_{C_1}^{-1}(n)$ which can only have m possibilities for t. Intuitively, what we mean by a "bad group" is a group which whether in the mapping $\sigma_{C_2}^{-1}(l)$ maps l out of the corresponding windows of size m or in the mapping $\sigma_{C_1}(t)$ maps t out of the corresponding windows of size m. Term "good groups" is used to recover $\sigma_{C_1}, \sigma_{C_2}$. We now define more precisely what we mean by a "bad group" and how they can be filtered.

Definition 1. *We call a group a "bad group" if the exists no pair $(i,n) \in [1,m]^2$, such that $\sigma_{C_2}(i) = l$ and $\sigma_{C_1}^{-1}(n) = t$.*

Lemma 3. *The group $\mathcal{G}_{l\to t}$ is bad if for every pair $(i,n) \in [1,m]^2$ there exists $U \in \mathcal{G}_{l\to t}$ such that the equation 2 is not satisfied.*

Proof. The equation 2 has to be satisfied for group $\mathcal{G}_{l\to t}$ only if we guess $\sigma_{C_2}(i) = l$ and $\sigma_{C_1}^{-1}(n) = t$ correctly. If $\sigma_{C_2}(i) \neq l$ or $\sigma_{C_1}(t) \neq n$, then given $U \in \mathcal{G}_{l\to t}$ we have $\frac{1}{2}$ probability that the equation 2 would be satisfied even if the group is chosen incorrectly.

Therefore, we drop out a "bad group" $\mathcal{G}_{l\to t}$ if there exists no pair $(i,n) \in [1,m]^2$ such that $(i \to l \to t \to n)$ for which equation (2) is satisfied for all elements (see Fig 9). Following this step, we also output a correct mapping $(i \to l \to t \to n)$ where $(l,t) \in [1,k]^2$ and $(i,n) \in [1,m]^2$. The probability to accept a given incorrect group is lower than $\frac{k^2}{2^q}$. So, for $k^4 \ll 2^q$ we keep no incorrect group for sure. That is, we need $q \approx 4\log_2 k$.

Now we have m^2 groups left after filtering and at least one mapping $(i_f \to l_f \to t_f \to n_f)$ for a group $\mathcal{G}_{l_f\to t_f}$. These m^2 groups correspond to all groups $\mathcal{G}_{l_i\to e_i}$ for $l_i \in \{\sigma_{C_2}[1],\ldots,\sigma_{C_2}[m]\}$ and $e_i \in \{\sigma_{C_1}^{-1}[1],\ldots,\sigma_{C_1}^{-1}[m]\}$ (see Fig. 8). Since $\sigma_{C_1}, \sigma_{C_2}$ are fixed, the correct groups correspond to all mappings between the set of m indices $\{\sigma_{C_2}[1],\ldots,\sigma_{C_2}[m]\}$ and the set of m indices $\{\sigma_{C_1}^{-1}[1],\ldots,\sigma_{C_1}^{-1}[m]\}$.

Now, we fix l to l_f. This way we reduce the number of groups to m. At this stage, we know the exact mapping is $i_f \to l_f$. Depending on which group $\mathcal{G}_{l_f\to t_g}$ we pick at this stage (we have a free choice of t_g), we have m distinct mappings from bit l_f. We pick one of these groups $\mathcal{G}_{l_f\to t_g}$ which maps l_f to t_g. i.e., the mappings on both ends of Fig. 7 are fixed. Then, we go through all m possibilities for n and check for all $U \in \mathcal{G}_{l_f\to t_g}$ whether $\mathsf{ARMADILLO1d}(C_1\|C_2\|U)[n] \oplus g(U)[t_g] \oplus X[n] \oplus X[i_f]$ is constant. If yes, we know that t_g maps to n. Using all m groups $\mathcal{G}_{l_f\to\cdot}$, we can recover permutation $\sigma_{C_1}^{-1}$ on $[1,m]$. We can fix t to t_f this time and perform the same procedure to recover σ_{C_2}.

Now we have all we need to forge a response for a new challenge. Let U' be a new challenge. We forge ARMADILLO1d bit by bit. Let consider bit n of the responce R'. We have recovered $\sigma_{C_1}^{-1}[1,m]$ and therefore we know $\sigma_{C_1}^{-1}(n)$ where $n \in [1,m]$ i.e., position

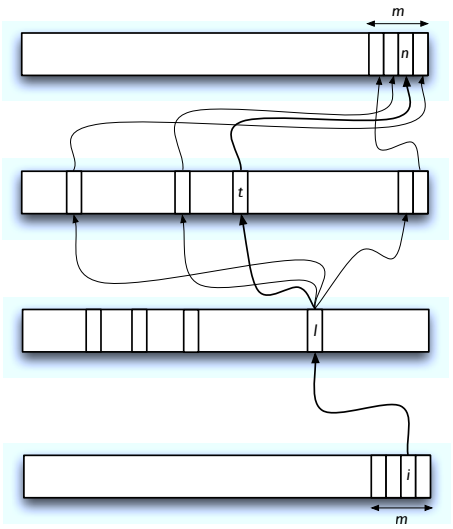

Fig. 7. ARMADILLO1d attack scheme

to which the n^{th} bit of the response R' is mapped. Now we select l [1] such that $U' \in \mathcal{G}_{l \to \sigma_{C_1}^{-1}(n)}$. If we find such l (i.e., we find the corresponding group) then we forge the $n = \sigma_{C_1}\sigma_U\sigma_{C_2}(i)$-th bit of the response as follows. Let $U \in \mathcal{G}_{l \to \sigma_{C_1}^{-1}(n)}$ be a representative of such a group. From equation (2) we have

$$\begin{aligned}\mathsf{ARMADILLO1d}(C_1\|C_2\|U)[n] \oplus g(U)[t] \oplus X[n] \oplus X[i] = (a_l \oplus b_n)\\ \mathsf{ARMADILLO1d}(C_1\|C_2\|U')[n] \oplus g(U')[t] \oplus X'[n] \oplus X'[i] = (a_l \oplus b_n)\end{aligned}$$

and therefore

$$\begin{aligned}\mathsf{ARMADILLO1d}(C\|U')[n] = \mathsf{ARMADILLO1d}(C\|U)[n] \oplus (X[i] \oplus X'[i]) \oplus (g(U)[t]\\ \oplus g(U')[t]) \oplus (X[n] \oplus X'[n])\end{aligned}$$

If there is no such l (i.e., we cannot find any corresponding group) we forge the $n = \sigma_{C_1}\sigma_U\sigma_{C_2}(i)$-th bit of response as follows. Compared to the previous case we only drop $(X[i] \oplus X'[i])$, since $X[i] = X'[i]$, since the bit is coming from the secret part.

$$\begin{aligned}\mathsf{ARMADILLO1d}(C\|U')[n] = \mathsf{ARMADILLO1d}(C\|U)[n] \oplus g(U)[t] \oplus g(U')[t]\\ \oplus X[n] \oplus X'[n]\end{aligned}$$

The complexity of the forgery attack is $O(qk^4)$ with qk challenges, where $q \approx 4\log_2 k$.

[1] Notice that this l can be in the "bad groups". We only use the "good groups" to recover the secret permutations σ_{C_2} and σ_{C_1}.

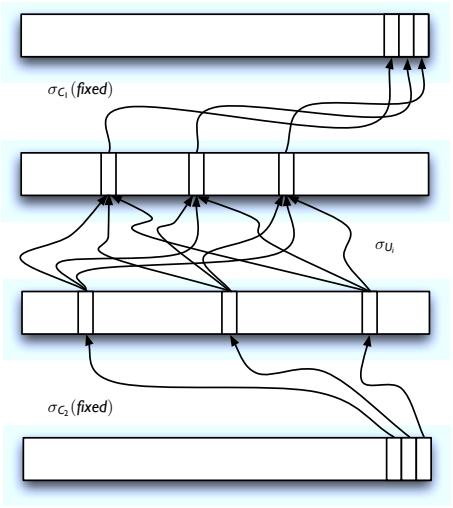

Fig. 8. ARMADILLO1d group filtering scheme

6.2 Extension with the T_4 and T_2 Transformations

Let now consider the definition for a general case of a T_2 and T_4 and $T_1(C_1\|C_2\|U) = (C_1\|U\|C_2)$, i.e., we assume that π keeps a large piece of consecutive bits of U together.

$$\mathsf{ARMADILLO1d}(C_1\|C_2\|U) = T_4(P(C_1\|U\|C_2, T_2(C_1\|C_2\|U)), (C_1\|C_2\|U))$$

The same steps as the previous attacks hold in the general case as well. In the case of T_2, since the secret is fixed it can be derived out as a constant in our computations. So, deploying the same grouping strategy, the attack still works. It is not difficult to see that the same method also holds even if we have the T_4 function.

6.3 Extension to a general π

Let now consider the definition

$$\mathsf{ARMADILLO1d}(X) = P(X_\pi, X) \oplus X$$

for $X = (C_1\| \ldots \|C_t\|U_1\| \ldots \|U_{t-1})$. In the algorithm above, the attacker decomposes the computation of ARMADILLO1d into several stages. Let suppose `wlog` that we permute the bits of secret using permutation π as follows

$$(C_1\| \ldots \|C_t\|U_1\| \ldots \|U_{t-1})_\pi = C_1\|U_1\|C_2\|U_2\| \ldots \|C_{t-1}\|U_{t-1}\|C_t$$

I.e. π mixes C and U bits together, without putting too many consecutive bits of U. We use t times the forgery algorithm on ARMADILLO1d to recover all mappings σ_{C_i}.

```
1: for all $\mathcal{G}_{l\to t}$, where $(l,t)\in[1,k]^2$ do
2:   $\omega = 0$
3:   for all $(i,n)\in[1,m]^2$ do
4:     $err = 0$
5:     pick an arbitrary $U_1 \in \mathcal{G}_{l\to m}$
6:     $\nu = \mathsf{ARMADILLO1d}(C_1\|C_2\|U_1)[n]\oplus g(U_1)[t]\oplus X[n]\oplus X[i]$
7:     for all $U \in \mathcal{G}_{l\to m}\backslash U_1$ do
8:       if $\nu \neq \mathsf{ARMADILLO1d}(C_1\|C_2\|U)[n]\oplus g(U)[t]\oplus X[n]\oplus X[i]$ then
9:         $\omega = \omega+1$
10:        $err = 1$
11:        break
12:      end if
13:    end for
14:    if $(err == 0)$ then
15:      output $(i\to l\to t\to n)$.
16:    end if
17:  end for
18:  if $(\omega == m^2)$ then
19:    drop $\mathcal{G}_{l\to t}$.
20:  end if
21: end for
```

Fig. 9. group filtering algorithm for ARMADILLO1d.

Let assume that $2^{|U_i|} \geq kq$ for every i. If this is not the case we can recover the mapping $\sigma_{C_{i-1}}\sigma_{U_i}\sigma_{C_i}$ for all evaluations of U_i in polynomial time. In both cases, we recover each σ_{C_i} and $\sigma_{C_{i+1}}$ recursively. To recover σ_{C_i} and $\sigma_{C_{i+1}}$, we fix $E_{i+1} = U_{i+1}\|C_{i+2}\|\ldots\|C_t$ by fixing $U_{i+1},\ldots,U_{t-1}$, since $C_{i+2},\ldots,C_t$ is already fixed. Then, the problem is reduced to the same situation as the previous attack, where we have a challenge part sandwiched between two intervals of the secret bits C. Finally, we obtain the mapping σ_{C_1} on set $[1,m]$, σ_{C_2}, …, $\sigma_{C_{t-1}}$ on set $[1,k]$ and $\sigma_{C_t}^{-1}$ on set $[1,m]$. Note that we can recover permutations σ_{C_2}, …, $\sigma_{C_{t-1}}$ on set $[1,k]$ by setting challenge bits to different values $\frac{k}{m}$ times. All we need is to describe an algorithm to recover all constants $P(C_i,0)$. Then we will have everything we need to forge the response of ARMADILLO1d to any challenge.

We now describe how to recover $P(C_1,0)$. The same method can be used to derive other $P(C_i,0)$ recursively. The same as before, Let fix all U_i's except U_1. We use Lemma 1. and Lemma 2. and rewrite

$$\begin{aligned}\mathsf{ARMADILLO1d}(X)\oplus X &= P(C_1\|U_1\|E_2,X)\\ &= P(E_2,P(U_1,P(C_1,X)))\\ &= P(C_1,X)_{\sigma_{U_1}\sigma_{E_2}}\oplus P(U_1,0)_{\sigma_{E_2}}\oplus P(E_2,0)\\ &= P(C_1,0)_{\sigma_{U_1}\sigma_{E_2}}\oplus X_{\sigma_{U_1}\sigma_{E_2}}\oplus P(U_1,0)_{\sigma_{E_2}}\oplus P(E_2,0)\end{aligned}$$

which gives us set of linear equations and we can vary σ_{U_1} as necessary to obtain a large system of equations and solve it.

Complexity. The general attack may iterate the algorithm in Fig. 9. up to k times. In some iterations, the requirement $2^{|U_i|} \geq k \log k$ does not need to be satisfied. In such case, we need to extend the interval U_i by guessing some bits of key. Such technique would require another $k \log k$ steps. Therefore, the complexity is bounded by $O(k \cdot k^4 q \cdot k \log k)$. We specified before that $q \approx 4 \log k$. Hence, the complexity of the offline stage of the attack is $O(k^6 \log^2 k)$ and the algorithm requires at most $k^3 \log^2 k$ queries.

6.4 Attack Impact on ARMADILLO2

We can see ARMADILLO2 as a successor of ARMADILLO1d with a pre-processing T_1 which is more elaborate than a simple transposition. Such preprocessing makes it resistant against our attack. Our attack is based on decomposition according to Lemma 1 and a guess of a constant value of function $f(U)$ from Lemma 2. The pre-processing phase protects against both the decomposition and the constant value of function $f(U)$. However, the attack we propose points out possible weaknesses in the design of ARMADILLO2.

7 Conclusion

We have shown a devastating key recovery attack against ARMADILLO1 and discussed a potential implication on ARMADILLO2. Although we did not find an attack on ARMADILLO2, we have illustrated that the non-linearity based on data-dependent permutations in both ARMADILLO1 and ARMADILLO2 is not sufficient. The results do not immediately apply on ARMADILLO2 but they allow for better understanding the design and they might be used to improve the attack in [1].

Acknowledgements. This work was sponsored by Oridao [2]. We would like to thank Nicolas Reffé for his kind support. The ARMADILLO algorithms are subject to patent number WO/2008/148784 [4].

References

1. M. Abdelraheem, C. Blondeau, M. Naya-Plasencia, M. Videau, and E. Zenner. Cryptanalysis of ARMADILLO2. In *proceeding of ASIACRYPT 2011*. Springer, 2011.
2. Stéphane Badel, Nilay Dağtekin, Jorge Nakahara, Khaled Ouafi, Nicolas Reffé, Pouyan Sepehrdad, Petr Sušil, and Serge Vaudenay. ARMADILLO: A Multi-purpose Cryptographic Primitive Dedicated to Hardware. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, chapter 27, pages 398–412–412. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2011.
3. John Kobza, Sheldon Jacobson, and Diane Vaughan. A Survey of the Coupon Collector's Problem with Random Sample Sizes. *Methodology and Computing in Applied Probability*, 9(4):573–584, December 2007.
4. N. Reffé. CRYPTOGRAPHIC METHODS AND DEVICES FOR THE PSEUDO-RANDOM GENERATION OF DATA ENCRYPTION AND CRYPTOGRAPHIC HASHING OF A MESSAGE, 12 2008.

[2] http://www.oridao.com/