ARMADILLO: a Multi-Purpose Cryptographic Primitive Dedicated to Hardware

Stéphane Badel[1], Nilay Dağtekin[1], Jorge Nakahara Jr⋆[1], Khaled Ouafi⋆⋆[1], Nicolas Reffé[2], Pouyan Sepehrdad[1], Petr Sušil[1], Serge Vaudenay[1]

[1] EPFL, Lausanne, Switzerland
[2] Oridao, Montpellier, France
{stephane.badel, nilay.dagtekin, jorge.nakahara, pouyan.sepehrdad, petr.susil, khaled.ouafi, serge.vaudenay}@epfl.ch, nicolas.reffe@oridao.com

Abstract. This paper describes and analyzes the security of a general-purpose cryptographic function design, with application in RFID tags and sensor networks. Based on these analyzes, we suggest minimum parameter values for the main components of this cryptographic function, called ARMADILLO. With fully serial architecture we obtain that 2 923 GE could perform one compression function computation within 176 clock cycles, consuming 44 μW at 1 MHz clock frequency. This could either authenticate a peer or hash 48 bits, or encrypt 128 bits on RFID tags. A better tradeoff would use 4 030 GE, 77 μW of power and 44 cycles for the same, to hash (resp. encrypt) at a rate of 1.1 Mbps (resp. 2.9 Mbps). As other tradeoffs are proposed, we show that ARMADILLO offers competitive performances for hashing relative to a fair Figure Of Merit (FOM).

1 Introduction

Cryptographic hash functions form a fundamental and pervasive cryptographic primitive, for instance, providing data integrity in digital signature schemes, and for message authentication in MACs. In particular, there are very few known hardware-dedicated hash function designs, for instance, Cellhash [6] and Subhash [5]. On the other hand, Bogdanov *et al.* [2] suggest block-cipher based hash functions for RFID tags using the PRESENT block cipher. Concerning block and stream ciphers, the most prominent developments include PRESENT [1], TEA [22], HIGHT [13], Grain [12], Trivium [4] and KATAN, KTANTAN family [3].

We propose a cryptographic function dedicated to hardware which can be used for several cryptographic purposes.[3] Such functions rely on data-dependent bit transpositions [16]. Given a bitstring $x = x_{2k}\|\cdots\|x_1$, fixed permutations σ_0 and σ_1 over the set $\{1,2,\ldots,2k\}$, 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

⋆ This work was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the SNF under grant number 5005-67322.

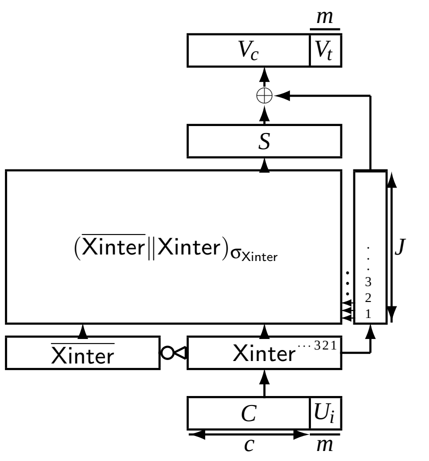
⋆⋆ Supported by a grant of the Swiss National Science Foundation, 200021-119847/1.

[3] The content of this paper is subject to a pending patent by ORIDAO http://www.oridao.com/.

is, $x_\sigma = x_{\sigma(2k)} \| \cdots \| x_{\sigma(1)}$. 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. It was already used in the Zemor-Tillich construction [21] for $G = \mathsf{SL}_2$ and in braid group cryptography [10]. We observe that when σ_0 and σ_1 induce an expander graph on the vertex set $v = \{1,\ldots,2k\}$, then $(s,x) \mapsto x_{\sigma_s}$ has good cryptographic properties.

This paper is organized as follows: Sect. 2 describes a general-purpose cryptographic function called ARMADILLO. In Sect. 3 we analyze ARMADILLO. Sect. 4 contains design criteria for the bit permutation components of ARMADILLO. Sect. 5 suggests parameter vectors. Sect. 6 presents an updated design, called ARMADILLO2. Sect. 7 provides implementation results. Sect. 8 compares hardware implementations of ARMADILLO with other well-known hash functions.

Notations. Throughout this document, $\|$ denotes the concatenation of bitstrings, $\oplus$ denotes the bitwise XOR operation, $\bar{x}$ denotes the bitwise complement of a bitstring x; we assume the little-endian numbering of bits, such as $x = x_{2k} \| \cdots \| x_1$.



m
V_c
V_t
$\oplus$
S
$(\overline{\mathsf{Xinter}} \| \mathsf{Xinter})_{\sigma_{\mathsf{Xinter}}}$
J
3
2
1
$\overline{\mathsf{Xinter}}$
$\mathsf{Xinter}^{\cdots 321}$
C
U_i
c
m

Fig. 1. The ARMADILLO function.

2 The ARMADILLO Function

ARMADILLO maps an initial value C and a message block U_i to two values

$$(V_c, V_t) = \mathsf{ARMADILLO}(C, U_i)$$

By definition, C and V_c are of c bits, V_t as well as each block U_i are of m bits, a register Xinter is of $k = c + m$ bits. ARMADILLO is defined by integer parameters c, m, $J = c + m$, and two fixed permutations σ_0 and σ_1 over the set $\{1,2,\ldots,2k\}$. $\mathsf{ARMADILLO}(C,U)$ works as follows (see Fig. 1)

```
1: set $\mathsf{Xinter} = C\|U$;
2: set a $2k$-bit register $x = \overline{\mathsf{Xinter}}\|\mathsf{Xinter}$;
3: $x$ undergoes a sequence of bit permutations, $\sigma_0$ and $\sigma_1$, which we denote by $P$. $P$
   maps a bitstring of $k$ bits and a vector $x$ of $2k$ bits into another vector of $2k$ bits.
   Assuming $J = k$, the output of this sequence of $J$ bit permutations is truncated to
   the rightmost $k$ bits, denoted $S$, by
       $S = P(\mathsf{Xinter}, x) = \mathsf{tail}_k((\overline{\mathsf{Xinter}}\|\mathsf{Xinter})_{\sigma_{\mathsf{Xinter}}})$
4: set $V_c\|V_t$ to the value of $S \oplus \mathsf{Xinter}$.
```

The security is characterized by two parameters S_{offline} and S_{online}. Concretely, the best offline attack has complexity $2^{S_{\mathsf{offline}}}$, while the best online one, with practical complexity, has success probability $2^{-S_{\mathsf{online}}}$. Typically, we aim at $S_{\mathsf{offline}} \geq 80$ and $S_{\mathsf{online}} \geq 40$. However, we can only upper bound S_{offline} and S_{online}.

Application I: FIL-MAC. For challenge-response protocols (e.g. for RFID tags [17]), the objective is to have a fixed input-length MAC. Suppose that C is a secret and U is a challenge. The value V_t is the response or the authentication tag. We write

$$V_t = \mathsf{AMAC}_C(U)$$

Additionally, the V_c output could be used to renew the secret in a synchronized way or to derive an encryption key for a secure messaging session as specified in [17]. The security of challenge-response protocols requires that an adversary cannot extract from the RFID tag enough information that allows it to impersonate the tag with high probability. In this FIL-MAC context, the C parameter can be recovered by exhaustive search with complexity 2^c, where $c = |C|$; so, $S_{\mathsf{offline}} \leq c$. In addition to this, the adversary can try to guess V_t online with probability 2^{-m}, so $S_{\mathsf{online}} \leq m$.

Application II: Hashing and digital signatures. For variable-length input messages hashing, we assume a strengthened Merkle-Damgård [7,15] construction (with padding using length suffix) for ARMADILLO, with V_c as chaining variable, U as message block and V_c as hash digest. The initial value (IV) can use the fractional part of the square root of 3 truncated to c bits, similar to the values adopted in SHA-2 hash function family [20]. We write

$$V_c = \mathsf{AHASH}_{\mathsf{IV}}(\mathsf{message}\|\mathsf{padding}).$$

Generic birthday attacks are expected to find collisions in ARMADILLO with complexity $2^{\frac{c}{2}}$. So, $S_{\mathsf{offline}} \leq \frac{c}{2}$ when collisions are a concern. Preimages and second preimages are expected with probability 2^{-c}, so, $S_{\mathsf{offline}} \leq c$. Sometimes, free-start collisions or free-start second preimage attacks matter. In this case, we refer to Application II'.

Application III: PRNG and PRF. For pseudorandom generation, we take the first t bits of $V_c\|V_t$ after at least r iterations. We define

$$\mathsf{APRF}_{\mathsf{seed}}(x) = \mathsf{head}_t(\mathsf{AHASH}_{\mathsf{seed}}(x\|\mathsf{cste}))$$

with an input x with length multiple of m and cste a $(r-1)m$-bit constant. A relevant property for this application is indistinguishability. Assuming a secret seed, ARMADILLO could be used as a stream cipher. The keystream is composed of t-bit frames where the ith frame is $\mathsf{APRF}_{\mathsf{seed}}(i)$. The index i can be synchronized, or sent in clear in which case we have a self-synchronous stream cipher. In this setting, the output should be indistinguishable from a truly random string when the key is random.

3 Dedicated Attacks

Key recovery. Suppose $V_c \| V_t$ and U are known, and we look for C. Since U is known, the $\mathsf{tail}_m(\mathsf{Xinter})$ are known. Guessing the $\mathsf{tail}_{J-m}(C)$ gives access to the $\mathsf{tail}_J(S)$, since U is known. This fact motivates a meet-in-the-middle attack to recover $\mathsf{tail}_{J-m}(C)$. Let us split these $J-m$ bits of Xinter into two pieces of sizes $\lceil \frac{J-m}{2} \rceil$ and $\lfloor \frac{J-m}{2} \rfloor$. The heading $\lceil \frac{J-m}{2} \rceil$ bits are used to compute backwards the P permutation from S, with $m + \lceil \frac{J-m}{2} \rceil = \lceil \frac{J+m}{2} \rceil$ bits known. The tailing $\lfloor \frac{J-m}{2} \rfloor$ bits of C, together with the m bits of U form $m + \lfloor \frac{J-m}{2} \rfloor = \lfloor \frac{J+m}{2} \rfloor$ bits of Xinter. The meet-in-the-middle consists in checking consistency of known bits in the middle of the P permutation. We expect to find a solution with complexity $O(2^{\lceil \frac{J+m}{2} \rceil})$ and a single U. Thus, $S_{\mathsf{offline}} \leq \lceil \frac{J+m}{2} \rceil$ in Application I, II and III.

Free-start collision. We look for a triplet (C, U, U') that causes a collision, that is, $\mathsf{AHASH}_C(U) = \mathsf{AHASH}_C(U')$. For this, we look for (C, U, U') such that

$$P(U, \overline{C} \| \overline{U} \| C \| U) \approx P(U', \overline{C} \| \overline{U'} \| C \| U')$$

with $\approx$ meaning that the Hamming weight of the difference is some low value w. Then, we hope that the next P permutation will move all w different bits outside the window of the $c+m$ bits which are kept in S. Since the probability for a vector to have weight w is $\binom{2c+2m}{w} 2^{-2c-2m}$, the number of solutions we get is $\binom{2c+2m}{w} 2^{-c}$ on average. The probability that a solution leads to a collision is the probability that w difference bits are moved outside a window of c bits. Finally, the expected number of collisions we can get is $\binom{c+m}{w} 2^{-c}$. We can now fix $w = w_{\mathsf{opt}}$ such that $\binom{c+m}{w_{\mathsf{opt}}} \geq 2^c$ so that we can find one solution with complexity $2^{w_{\mathsf{opt}}}$. To implement the attack, for all U and U' we enumerate all C's such that $P(U, \overline{C} \| \overline{U} \| C \| U) \approx P(U', \overline{C} \| \overline{U'} \| C \| U')$. The complexity is $2^{2m} + \binom{2k}{w_{\mathsf{opt}}} 2^{-c}$ which is dominated by 2^{2m}. So, $S_{\mathsf{offline}} \leq 2m$ in Application II'.

A distinguisher. Assuming that the J iterations in the P permutation output a random $2k$-bit vector of Hamming weight k, we have $\binom{2k}{k}$ possible vectors. By extracting a window of t bits we do not have a uniformly distributed string. Indeed, any possible string of weight w has a probability of $p(w) = \binom{2k-t}{k-w} / \binom{2k}{k}$. There exists a distinguisher to tell whether a t-bit window comes from a random output from P or a truly random string, with advantage

$$\frac{1}{2} \sum_{w=0}^{t} \binom{t}{w} \left| \frac{\binom{2k-t}{k-w}}{\binom{2k}{k}} - \frac{1}{2^t} \right|$$

For $t = k = 160$, this is 0.1658. Here, the distinguisher recognizes P when the Hamming weight w is in the interval $[75,\ldots,85]$, and a random string otherwise.

The final XOR hides this bias a bit but we can wonder by how much exactly. Assume that we hash a message of r blocks. The final output is the XOR of the initial value together with r outputs from P. Assuming that the initial value is known and that the P outputs are random and independent, we can compute the distribution of the final hash by convolution. Indeed, the probability that it is a given string x is $p_r(x)$ such that

$$p_r(x) = \sum_{x_1 \oplus \cdots \oplus x_r = x} p(\mathsf{wt}(x_1)) \cdots p(\mathsf{wt}(x_r))$$

Let us define the spectrum $\hat{p}_r(\mu)$ by $\hat{p}_r(\mu) = \sum_x (-1)^{\mu \cdot x} p_r(x)$. We have $\hat{p}_r(\mu) = (\hat{p}_1(\mu))^r$. We can now compute

$$\hat{p}_1(\mu) = \sum_{i=0}^{\mathsf{wt}(\mu)} \sum_{j=0}^{t-\mathsf{wt}(\mu)} \binom{\mathsf{wt}(\mu)}{i} \binom{t-\mathsf{wt}(\mu)}{j} (-1)^i p(i+j)$$

It only depends on $\mathsf{wt}(\mu)$ so we write $\hat{p}_1(\mathsf{wt}(\mu))$. Since $\sum_\mu \hat{p}_r(\mu)^2 = 2^t \sum_x p_r(x)^2$ we deduce that the Squared Euclidean Imbalance (SEI) of the difference of the hash of r blocks with the initial value is

$$\mathsf{SEI}_r = 2^t \sum_x \left(p_r(x) - 2^{-t}\right)^2 = \sum_{\mu \neq 0} (\hat{p}_r(\mu))^2 = \sum_{w=1}^{t} \binom{t}{w} (\hat{p}_1(w))^{2r}$$

We have $S_{\mathsf{offline}} \leq -\log_2 \mathsf{SEI}_r$, where r is the minimal number of blocks which are processed in Application III. The SEI expresses as

$$\mathsf{SEI}_r = \sum_{w=1}^{t} \binom{t}{w} \left(\sum_{i=0}^{w} \binom{w}{i} \sum_{j=0}^{t-w} \binom{t-w}{j} (-1)^i \frac{\binom{2k-t}{k-i-j}}{\binom{2k}{k}} \right)^{2r}$$

As an example, we computed SEI_r for four selections of $t = k$.

	$t = k = 128$	$t = k = 160$	$t = k = 200$	$t = k = 275$
r	SEI_r			
1	$2^{-2.70}$	$2^{-2.70}$	$2^{-2.70}$	$2^{-2.70}$
2	$2^{-18.99}$	$2^{-19.63}$	$2^{-20.28}$	$2^{-21.20}$
3	$2^{-34.98}$	$2^{-36.27}$	$2^{-37.56}$	$2^{-39.40}$
4	$2^{-50.96}$	$2^{-52.90}$	$2^{-54.81}$	$2^{-57.60}$
5	$2^{-66.95}$	$2^{-69.54}$	$2^{-72.12}$	$2^{-75.81}$
6	$2^{-82.94}$	$2^{-86.17}$	$2^{-89.40}$	$2^{-94.01}$
7	$2^{-98.93}$	$2^{-102.81}$	$2^{-106.68}$	$2^{-112.21}$

Given k and c, we look for r and t such that $\mathsf{SEI}_r < 2^{-c}$ and r/t is minimal.

4 Permutation-Dependent Attacks

In this section we present security criteria for the σ_0 and σ_1 permutations.

Another distinguisher. Consider a set I of indices from $V = \{1, \ldots, 2k\}$. Let $\mathsf{swap}_I(\sigma) = \#\{i \in I; \sigma(i) \notin I\}$ and $\mathsf{wt}_I(x) = \sum_{i \in I} x_i$. We assume that $s_b = \mathsf{swap}_I(\sigma_b)$ is low for $b = 0$ and $b = 1$ to see how much the low diffusion between inside and outside I would lead to a distinguisher on $P(s, \cdot)$ with a random s of J bits. In the worst case we can assume that all indices in I are in the same half of x so that the distinguisher can choose the input on P with a very biased $\mathsf{wt}_I(x)$.

A permutation σ_b keeps $\#I - s_b$ of the bits inside I and introduce s_b bits from outside I. Assuming that all bits inside and outside I are randomly permuted, we have the approximation

$$E(\mathsf{wt}_I(x_{\sigma_b})) \approx (\#I - s_b)\frac{E(\mathsf{wt}_I(x))}{\#I} + s_b \frac{k - E(\mathsf{wt}_I(x))}{2k - \#I}.$$

Thus,

$$E(\mathsf{wt}_I(x_{\sigma_b})) - \frac{\#I}{2} \approx \left(1 - \frac{s_b}{\#I} - \frac{s_b}{2k - \#I}\right)\left(E(\mathsf{wt}_I(x)) - \frac{\#I}{2}\right).$$

On average over the control bits, we have

$$\frac{E(\mathsf{wt}_I(P(s,x))) - \frac{\#I}{2}}{E(\mathsf{wt}_I(x)) - \frac{\#I}{2}} \approx \left(1 - \frac{s_0 + s_1}{2} \times \frac{2k}{\#I(2k - \#I)}\right)^J.$$

The best strategy for the distinguisher consists of having either $\mathsf{wt}_I(x) = 0$ or $\mathsf{wt}_I(x) = \#I$. In both cases we have

$$\left|E(\mathsf{wt}_I(P(s,x))) - \frac{\#I}{2}\right| \approx \frac{\#I}{2}\left(1 - \frac{s_0 + s_1}{2} \times \frac{2k}{\#I(2k - \#I)}\right)^J.$$

The number of samples to significantly observe this bias is

$$T = \left(1 - \frac{s_0 + s_1}{2} \times \frac{2k}{\#I(2k - \#I)}\right)^{-2J}. \tag{1}$$

So, $S_{\mathsf{offline}} \leq \log_2 T$. This expression relates to the theory of expander graphs. We provide below a sufficient condition which can be easily checked.

To compute the minimal value of $\frac{s_0+s_1}{2\#I}$ over all I we observe that if P_{σ_b} is the matrix of permutation σ_b and if x_I is the 0-1 vector whose coordinate of index in I are the ones set to 1, then

$$\frac{s_0 + s_1}{2\#I} = 1 - \frac{x_I \cdot \left(\frac{P_{\sigma_0} + P_{\sigma_1}}{2} x_I\right)}{x_I \cdot x_I}. \tag{2}$$

Let u be the vector with all coordinates set to 1. Clearly, the hyperplane $u^\perp$ orthogonal to u is stable by the matrix $M_0 = \frac{1}{2}(P_{\sigma_0} + P_{\sigma_1})$. Let $M = \frac{1}{2}(M_0 + M_0^t)$, where the superscript indicates the transpose matrix. We can easily see that $Mu = u$. Furthermore, we notice that $Mx = \lambda x$ with $x \neq 0$ implies $|\lambda| \leq 1$. Let λ be the second largest eigenvalue of M, or equivalently the largest eigenvalue of operator M restricted to $u^\perp$. Note that λ can be $\lambda = 1$ if the eigenvalue 1 has multiplicity higher than one. We can easily prove that $|\lambda| = 1$ and $Mx = \lambda x$ with $x \neq 0$ implies that x_i is constant for all $i \in I$, for all connected components I for the relation $i \sim j \iff \exists s \quad \sigma_{s_{|s|}} \circ \cdots \circ \sigma_{s_2} \circ \sigma_{s_1}(i) = j$. Hence, the only sets I which are stable by σ_0 and σ_1 at the same time are the empty one and the complete set if and only if eigenvalue 1 has multiplicity one. So, having $\lambda < 1$ is already a reasonable criterion but we can have a more precise one. We know that for any vector x orthogonal to u we have $\frac{x\cdot(M_0x)}{x\cdot x} \le \lambda$ with equality when x is an eigenvector for λ. Thus,

$$\frac{x\cdot(M_0x)}{x\cdot x} \le \frac{(1-\lambda)\frac{(x\cdot u)^2}{u\cdot u} + \lambda(x\cdot x)}{x\cdot x}$$

for any $x \neq 0$. For $x = x_I$ we obtain

$$\frac{x_I\cdot(M_0x_I)}{x_I\cdot x_I} \le (1-\lambda)\frac{\#I}{2k} + \lambda. \tag{3}$$

From (2), $\frac{s_0+s_1}{2\#I} = 1 - \frac{x_I\cdot(M_0\cdot x_I)}{x_I\cdot x_I} \ge 1-(1-\lambda)\frac{\#I}{2k}+\lambda = (1-\lambda)(1-\frac{\#I}{2k})$. Going back to the complexity (1) of our distinguisher we have $T \ge \lambda^{-2J}$. Hence, by having $\lambda \le 2^{-\frac{S_{\mathsf{offline}}}{2J}}$ for an offline complexity $2^{S_{\mathsf{offline}}}$, we make sure that the distinguisher has complexity $T \ge 2^{S_{\mathsf{offline}}}$. To conclude, if λ is the second largest eigenvalue of $M = \frac{1}{4}(P_{\sigma_0} + P^t_{\sigma_0} + P_{\sigma_1} + P^t_{\sigma_1})$ then we have an attack of complexity λ^{-2J}. So, $S_{\mathsf{offline}} \le -2J\log_2\lambda$.

Yet another distinguisher. We define the vector x of dimension k such that the ith coordinate of x is the probability that x_i is set to 1. If x is fixed, we can consider that x is equal to x by abuse of notation. If $y = x_\sigma$, we have that y is obtained by multiplying a permutation matrix P_σ by x. We have $(P_\sigma)_{j,i} = 1$ if and only if $j = \sigma(i)$. Clearly, for $y = x_{\sigma_b}$ we can write

$$\begin{aligned} y &= ((1-b)P_{\sigma_0} + bP_{\sigma_1}) \times x \\ &= \left(\frac{1}{2}(P_{\sigma_0}+P_{\sigma_1}) + \frac{(-1)^b}{2}(P_{\sigma_0}-P_{\sigma_1})\right) \times x \end{aligned}$$

We let $M_0 = \frac{1}{2}P_{\sigma_0} + \frac{1}{2}P_{\sigma_1}$ and $M_1 = \frac{1}{2}P_{\sigma_0} - \frac{1}{2}P_{\sigma_1}$. We have

$$\prod_{i=1}^{J}(M_0 + (-1)^{s_i}M_1) = \sum_{a_1=0}^{1}\cdots\sum_{a_k=0}^{1}(-1)^{a_1s_1+\cdots+a_ks_k}M_a = \hat{M}_s$$

where $M_a = M_{a_1}\times\cdots\times M_{a_k}$. So, the vector of $y = P(s,x)$ is $y = \hat{M}_s x$. The average $\langle\hat{M}_s\rangle$ of $\hat{M}_s$ over all $s_1,\ldots,s_k$ is M_0^k. We define a square matrix F in which all terms are equal to $\frac{1}{k}$. Clearly, if s is a uniformly distributed J-bit random string, the probability vector of $P(s,x)$ is $M_0^J \times x$. Since M_0 is a bi-stochastic matrix, we have $M_0 \times F = F \times M_0 = F$. Similarly, we have $M_1 \times F = F \times M_1 = 0$. We easily deduce that $(M_0 - F)^J = M_0^J - F$. Let θ be the second largest eigenvalue of $M_0^tM_0$, or equivalently, the largest eigenvalue of $M_0^tM_0 - F$. For any vector x such that $\sum x_i = w$ and $0 \le x_i \le 1$, we have $\|M_0^Jx - wu\|_2^2 = \|(M_0-F)^Jx\|_2^2 \le w\theta^J$, where $u = (1,\ldots,1)$. So, the cumulated squared Euclidean imbalance of each component of M_0^Jx is bounded by $2k\theta^J$. Thus, the complexity is $\frac{1}{2k\theta^J}$, and $S_{\mathsf{offline}} \le -J\log_2\theta - \log_2 2k$. The average $\left\langle\hat{M}_s\binom{\bar{s}}{s}\right\rangle$ of the image of

$\mathsf{Xinter} = s$ is

$$\sum_a M_a \left\langle (-1)^{a\cdot s} \begin{pmatrix} \overline{s} \\ s \end{pmatrix} \right\rangle.$$

For $a = 0$ the average is $(\frac{1}{2} \cdots \frac{1}{2})$. For a of weight at least 2, the average is zero. For a of weight 1, e.g. $a = (1, 0, \ldots, 0)$ the average is $(-\frac{1}{2}, 0, \ldots, 0, \frac{1}{2}, 0, \ldots, 0)$. We let e_i be the vector with coordinate 1 in its ith position and 0 elsewhere. We have

$$\left\langle \hat{M}_s \begin{pmatrix} \overline{s} \\ s \end{pmatrix} \right\rangle = \begin{pmatrix} \frac{1}{2} \\ \vdots \\ \frac{1}{2} \end{pmatrix} + \frac{1}{2} \sum_{i=1}^{k} M_0^{i-1} M_1 M_0^{k-i} (-e_i + e_{k+i}).$$

Let

$$b = \frac{1}{2} \sum_{i=1}^{k} M_0^{i-1} M_1 M_0^{k-i} (-e_i + e_{k+i}). \tag{4}$$

The complexity is $\frac{1}{2\|b\|_2^2}$, so, we have $S_{\mathsf{offline}} \leq -2\log_2 \|b\|_2 - 1$.

The parity of P. Let ε_i be the parity of σ_i. The $x \mapsto P(s,x)$ is a permutation whose parity is $\varepsilon_0^{|s|-\mathsf{wt}(s)} \varepsilon_1^{\mathsf{wt}(s)}$. If $\varepsilon_0 \neq \varepsilon_1$, an adversary with black-box access to $x \mapsto P(s,x)$ and knowing $|s|$ can thus easily deduce $\mathsf{wt}(s)$. We thus, recommend that $\varepsilon_0 = \varepsilon_1$.

5 Parameter Vectors

Here we suggest sets of parameters for four different applications, based on our analyzes. In all cases, we require $J = c + m$ and also that σ_0 and σ_1 have the same parity.

I: in a challenge-response application: $S_{\mathsf{offline}} \leq \min(c, \frac{J+m}{2})$ and $S_{\mathsf{online}} \leq m$

II: in a collision-resistance context: $S_{\mathsf{offline}} \leq \frac{c}{2}$

II’: in a free-start collision context: $S_{\mathsf{offline}} \leq \min\left(\frac{c}{2}, 2m\right)$

III: $S_{\mathsf{offline}} \leq -\log_2 \mathsf{SEI}_r$. If λ is the second largest eigenvalue of $M = \frac{1}{4}(P_{\sigma_0} + P_{\sigma_0}^t + P_{\sigma_1} + P_{\sigma_1}^t)$ then $S_{\mathsf{offline}} \leq -2J\log_2 \lambda$. For σ_0 and σ_1, the second largest eigenvalue of $M_0^t M_0$, called θ, $S_{\mathsf{offline}} \leq -J\log_2 \theta - \log_2 k - 1$. The bias b in (4) shall satisfy $S_{\mathsf{offline}} \leq -2\log_2 \|b\|_2 - 1$. Moreover, $S_{\mathsf{offline}} \leq \frac{J+m}{2}$.

To match the ideal security, we need these bounds to yield $S_{\mathsf{offline}} \leq c$ and $S_{\mathsf{online}} \leq m$ for Application I, $S_{\mathsf{offline}} \leq \frac{c}{2}$ for Application II and II’, and $S_{\mathsf{offline}} \leq c$ for Application III. So, we take $J = c + m$, $m \geq \frac{c}{2}$; r and t such that $\mathsf{SEI}_r \leq 2^{-c}$; σ_0 and σ_1 such that $-\log_2 \lambda \geq \frac{c}{2(c+m)}$, $-\log_2 \theta \geq \frac{c+\log 2k}{c+m}$, $-\log_2 \|b\|_2 \geq \frac{c+1}{2}$, and $\varepsilon_0 = \varepsilon_1$. Our recommendations for the parameter values of ARMADILLO are given in Table 1. Note that c is the key length for Applications I and III and also the digest length for Application II.

Table 1. Parameter vectors.

Vector	k	J	c	m	r	t
A	128	128	80	48	6	128
B	192	192	128	64	9	192
C	240	240	160	80	10	240
D	288	288	192	96	12	288
E	384	384	256	128	15	384

6 ARMADILLO2

Ever since the first version of ARMADILLO, we have developed an updated design, called ARMADILLO2, that is even more robust than the version presented in Fig. 1. In fact, ARMADILLO2 brings in a new compression function, called Q, which is not only more compact in hardware than P, but also addresses security concerns brought about during the continuous analyzes of ARMADILLO. For these reasons, ARMADILLO2 is our preferred design choice. Due to space limitations, further details about the security analysis of ARMADILLO2 are omitted. ARMADILLO2 is defined by

$$(V_c, V_t) = \mathsf{ARMADILLO2}(C,U) = Q(X, C\|U) \oplus X, \text{ where } X = Q(U, C\|U).$$

We call the new permutation Q, instead of P as in Fig. 1, to avoid confusion. The main novelties are:

- there is *no* complementation of the k-bit input $\mathsf{Xinter} = C\|U$ anymore; as a consequence, the σ_i permutations (and therefore Q) now operate on k-bit data $C\|U$, instead of $\overline{C}\|\overline{U}\|C\|U$, leading to a more compact design;
- a *new* permutation Q which interleaves σ_i's, $i \in \{0,1\}$, with an xor using the k-bit constant bitstring $\gamma = 1010\cdots10$; Q is defined recursively as $Q(s\|b, X) = Q(s, X_{\sigma_b} \oplus \gamma)$ and $Q(\emptyset, X) = X$, for $b \in \{0,1\}$ and bitstrings s and X;
- the outermost Q is controlled by a data-dependent value, $X = Q(U, C\|U)$, in contrast to simply $C\|U$ in Fig. 1;

In the new structure of Q, the output bias disappears and we can take $r = 1$ and $t = k$.

7 Hardware Implementation and Performance

There exist different demands on the implementation and the optimization meanings for various application scenarios. In this context, the scalability of ARMADILLO allows to deploy the implementation in a very wide realm of area and speed parameters, which constitutes the most essential trade-off in electronics circuits. The implementation of the P function, using the building block, is depicted in Fig. 2(b). It accepts an input vector of $2k$ bits and a key of J bits. It consists of a variable number N of permutation stages, all identical, and each stage essentially requires $2k$ multiplexers (Fig. 2(a)). One register of

$2k$ bits is needed to hold the input and/or intermediate data, as well as one J-bit register to hold the permutation key. At each cycle, these registers are either loaded with new data/key or fed back the output data/key for a new permutation round, depending on the state of the load signal. The number N of permutations executed in each cycle can be adjusted, the only restriction being that J be an integer multiple of N. The output data is the $2k$ bits vector resulting from the permutation round, and the output key is the $J-N$ bits remaining to be processed. This building block can be flexibly assembled into a

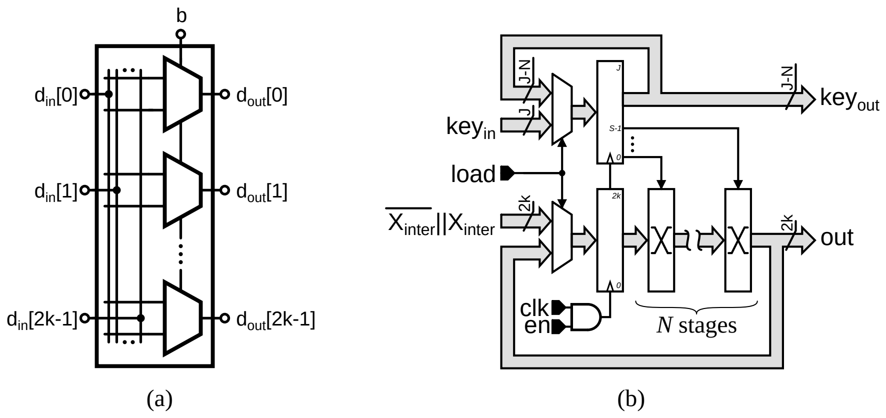


Fig. 2. Hardware implementation of the ARMADILLO function. (a) one permutation stage. (b) P function building block.

T-stage pipeline, where each stage performs a number $R = J/(N \cdot T)$ of permutation rounds (building blocks) before passing the results to the next stage and accepting new input from the previous stage. In that case, the throughput is $1/R$ items per cycle and the latency is J/N cycles, the parameters being linked by the equality $R \cdot N \cdot T = J$. The latency / throughput / cost trade-off can be adjusted, the two extreme cases being $R = 1$ (fully pipelined, resulting in a throughput of 1 item per cycle) and $T = 1$ (fully serial, resulting in a throughput of S/J items per cycle). Obviously, the more pipeline stages, the more hardware replication and therefore the higher the cost in area and power. To construct the complete hash function of Fig. 1, we essentially need to add a state machine (which is little more than a counter) around the permutation function block, and the final XOR operation.

Metrics for evaluating performance In order to compare different cryptographic functions, several metrics can be taken into account. The security is of course the primary concern. The silicon area, the throughput, the latency and the power dissipation are other metrics of interest, and can be traded-off for one another. For example, the power dissipation is nearly proportional to the clock frequency in any CMOS circuit, therefore, power can be reduced by decreasing the clock frequency and thus at the expense of throughput. Conversely, throughput can be increased by running at a faster clock frequency, up to a maximum clock frequency which is process- and implementation-dependent. Another example is serialization, where an operation is broken into several steps executed in series, allowing to reuse the same hardware, but again at the cost of a longer execution time. Through serialization, throughput and latency can be traded-off for area, down to a point where operations can not be broken into smaller operations anymore and we have reached a minimum area. Given this large design space, comparing the relative merit of different cryptographic functions is a challenging task.

The approach taken in [2] (and numerous other publications) includes comparing the area of synthesized circuits as reported in the literature or estimated by the authors in gate-equivalent (GE). It is notable though that the GE unit of measure, while being convenient because it is process-independent, is very coarse. For example, does the reported area after synthesis include the space needed for wiring? Typically, the utilization of a routed circuit can be in the range of 50%–80%, and is especially critical when using a limited number of metal layers for routing. A synthesis tool may report an estimated routing area, but in all cases it may vary to a large extent after physical implementation. Consider also that one design may have scan chains inserted while another may not, which may increase the register area by as much as 20–30% and require extra interconnections. Furthermore, different standard cells may be of varying area efficiency; as an illustration of this fact, a comparison of gate-equivalent figures from different standard-cell libraries can produce different results with a ratio up to $\frac{2}{3}$. For instance, a simple 2-input multiplexer can lead to 2.67 GE or 1.67 GE from one library to the other. Taking into account all these factors, it is clear that such a comparison can have a large margin of error, unless the circuits being compared have been implemented in the exact same conditions.

Besides comparing areas, the authors of [2] also use a metric called efficiency, which is defined as the ratio of the throughput (measured at a fixed clock frequency) over the area. It may seem at first sight that such a metric provides a more general measure of quality, since it may be fair to give up some area for a higher throughput, however it is flawed in that it does not consider the possibility of trading off throughput for power. Indeed, according to this metric, two designs A and B would be deemed of equal value if, for example, A's throughput and area were twice B's throughput and area, respectively. However, if B's power dissipation is half that of A at the same clock frequency, then by doubling B's operating frequency, its throughput can be made equal to that of A while consuming the same power and still occupying a smaller area. Clearly then, B should be recognized as superior to A, which can be captured by dividing the metric by the power dissipation, thus making it independent of the power/throughput trade-off. However, this does not come without its own problems, since the power dissipation is an extremely volatile quantity. Being subject to the same error factors as the area as described above, it also depends heavily on the process technology, the supply voltage, and the parasitic capacitances due to the interconnections. Furthermore, it can vary largely depending on the method used to measure it (i.e. gate-level statistical or vector-based simulation, or SPICE simulation). As if this were not enough, different standard-cell libraries also exhibit various power/area/speed trade-offs, for example, a circuit implemented with a high-density library is likely to result in a lower power figure than the same circuit implemented with a general-purpose library, for a similar gate count.

Nevertheless, a fairer figure of merit would need to include the influence of power dissipation. In order to keep process-independent metrics, we can assume that the power is proportional to the gate count.[4] This is reasonable since the dynamic power in CMOS circuits is proportional to the total switched capacitance, which correlates to the area. We propose therefore to use a figure of merit defined as $\text{FOM} = \text{throughput}/\text{GE}^2$. In practice, this is a coarse approximation, since it does not take into account switching activity or the influence of wire load; it is nevertheless fairer than not including power dissipation at all, since it tends to favor designs with smaller area (at equal throughput) which are very likely to dissipate less power.

Synthesis Results Table 2 presents the results of synthesis for the hash function described above in a $0.18\mu m$ CMOS process using a commercial standard-cell library, with the parameters given in Sect. 5. Synthesis was performed with Synopsys Design Compiler in topographical mode, in order to obtain accurate wire loads. The power consumption was evaluated with Synopsys Primetime-PX using gate-level vector-based analysis.

In RFID applications, the latency is constrained by the communication protocols (though the constraint is relatively easily satisfiable) but a high throughput is not necessary, designating a fully serial implementation as the ideal candidate. Therefore T is set to $T = 1$. The number N of permutations per clock cycle in the permutation function is set to $N = 1$, which is favorable to smaller area and power consumption for the tight power budget associated with RFID applications. The clock frequency is set to 1MHz, which is a representative value for the target application.

In hash mode we hash m bits per compression. In encryption mode we encrypt t/r bits per compression. The throughput values given in Table 2 correspond to hash mode.

Our goal for selecting $T = 1$ and $N = 1$ was to minimize the hardware. The area in the proposed implementation is roughly proportional to

$$(k_{\mathsf{reg}} * (2k + J) + k_{\mathsf{log}} * (2k(N+1) + J))T$$

for some constants k_{reg} and k_{log}.

To maximize the FOM with T given, we can show that we should in theory pick

$$N = \left(\frac{k_{\mathsf{reg}}}{k_{\mathsf{log}}} + 1\right)\left(1 + \frac{J}{2k}\right)$$

For $k_{\mathsf{reg}} \approx 2k_{\mathsf{log}}$ and $J = k$, this is $N = 4.5$. In practice, the best choice is to take $T = 1$ and $N = 4$ for ARMADILLO2 in context A, for which we would get an area of 4030 GE, 77 μW, and a latency of 44 cycles (1.09 Mbps for hashing or 2.9 Mbps for encryption).

8 Comparison

Table 3 shows a comparison of hardware implementations of ARMADILLO in the hash function setting, relative to other hash functions such as MD4, MD5, SHA-1, SHA-

[4] In the same spirit as the GE unit of measure, a more interesting metric would be to divide the power by $C_{unit} \cdot V_{DD}^2$, where C_{unit} is the input capacitance of an inverter. However, this is not applicable to compare other published implementations since these quantities are usually unknown

Table 2. Synthesis results at 1MHz.

Algorithm	N=1				N=4			
	Area (GE)	Power (μW)	Throughput (kbps)	Latency (cycles)	Area (GE)	Power (μW)	Throughput (kbps)	Latency (cycles)
ARMADILLO-A	3972	69	375	128	5770	133	1500	32
ARMADILLO-B	6598	117	333	192	9709	237	1333	48
ARMADILLO-C	8231	146	333	240	12217	300	1333	60
ARMADILLO-D	8650	177	333	288	14641	368	1333	72
ARMADILLO-E	13344	228	333	384	19669	513	1333	96
ARMADILLO2-A	2923	44	272	176	4030	77	1090	44
ARMADILLO2-B	4353	65	250	256	6025	118	1000	64
ARMADILLO2-C	5406	83	250	320	7492	158	1000	80
ARMADILLO2-D	6554	102	250	384	8999	183	1000	96
ARMADILLO2-E	8653	137	250	512	11914	251	1000	128

256, and MAME according to [2]. We computed the throughput in kbps at a clock rate of 100 kHz. We added the best FOM results for KATAN and KTANTAN with 64-bit blocks from [3]. Algorithms are categorized in terms of security by taking into account the digest size. In each category, we listed the algorithms by decreasing order of merit. To estimate the FOM we assumed that the power was proportional to the area. So, it is the speed divided by the square of the area. These figures show that different versions of ARMADILLO2 provide clear advantage for hashing, either in terms of area, or of throughput, or of overall merit.

Table 3. Implementation comparison for hash functions with throughput at 100 kHz.

Algorithm	Digest (bits)	Block (bits)	Area (GE)	Time (cycles/block)	Throughput (kb/s)	Logic (μm)	FOM (nanobit/cycle.GE^2)
ARMADILLO2-A	80	48	4030	44	109	0.18	67.17
ARMADILLO2-A	80	48	2923	176	27	0.18	31.92
H-PRESENT-128 [2]	128	128	4256	32	200	0.18	110.41
ARMADILLO2-B	128	64	6025	64	1000	0.18	27.55
MD4 [9]	128	512	7350	456	112.28	0.13	20.78
ARMADILLO2-B	128	64	4353	256	250	0.18	13.19
MD5 [9]	128	512	8400	612	83.66	0.13	11.86
ARMADILLO2-C	160	80	7492	80	100	0.18	17.81
ARMADILLO2-C	160	80	5406	320	250	0.18	8.55
SHA-1 [9]	160	512	8120	1274	40.18	0.35	6.10
ARMADILLO2-D	192	96	8999	96	100	0.18	12.35
C-PRESENT-192 [2]	192	192	8048	108	59.26	0.18	9.15
ARMADILLO2-D	192	96	6554	384	25	0.18	5.82
MAME [24]	256	256	8100	96	266.67	0.18	40.64
ARMADILLO2-E	256	128	11914	128	100	0.18	7.05
SHA-256 [9]	256	512	10868	1128	45.39	0.35	3.84
ARMADILLO2-E	256	128	8653	512	25	0.18	3.34

Table 4. Implementation comparison for encryption with throughput at 100 kHz.

Algorithm	Key (bits)	Block (bits)	Area (GE)	Time (cycles/block)	Throughput (kb/s)	Logic (μm)	FOM (nanobit/cycle.GE^2)
DES [18]	56	64	2309	144	44	0.18	83.36
PRESENT-80 [1]	80	64	1570	32	200	0.18	811.39
Grain [11]	80	1	1294	1	100	0.13	597.22
KTANTAN64 [3]	80	64	927	128	50	0.13	581.85
KATAN64 [3]	80	64	1269	85	75	0.13	467.56
ARMADILLO2-A	80	128	4030	44	291	0.18	179.12
Trivium [11]	80	1	2599	1	100	0.13	148.04
PRESENT-80 [19]	80	64	1075	563	11	0.18	98.37
ARMADILLO2-A	80	128	2923	176	73	0.18	85.12
mCrypton [14]	96	64	2681	13	500	0.13	684.96
PRESENT-128 [1]	128	64	1886	32	200	0.18	562.27
HIGHT [13]	128	64	3048	34	189	0.25	202.61
TEA [23]	128	64	2355	64	100	0.18	180.31
ARMADILLO2-B	128	192	6025	64	300	0.18	82.64
ARMADILLO2-B	128	192	4353	256	75	0.18	39.58
AES-128 [8]	128	128	3400	1032	12	0.35	10.73
ARMADILLO2-C	160	240	7492	80	300	0.18	53.45
ARMADILLO2-C	160	240	5406	320	75	0.18	25.66
DESXL [18]	184	64	2168	144	44	0.18	94.56
ARMADILLO2-D	192	288	8999	96	300	0.18	37.04
ARMADILLO2-D	192	288	6554	384	75	0.18	17.46
ARMADILLO2-E	256	384	11914	128	300	0.18	21.13
ARMADILLO2-E	256	384	8653	512	75	0.18	10.02

9 Conclusions

This paper suggested a new hardware dedicated cryptographic function design called ARMADILLO. Applications for ARMADILLO include MACs, hashing for challenge-response protocols, PRNG and as a stream cipher.

References

1. Bogdanov,A., Knudsen,L.R., Leander,G., Paar,C., Poschmann,A., Robshaw,M.J.B., Seurin,Y., Vikkelsoe,C.: Present: a Ultra-Lightweight Block Cipher. CHES'07, LNCS, vol. 4727, pp. 450–466. Springer (2007)
2. Bogdanov,A., Leander,G., Paar,C., Poschmann,A., Robshaw,M.J.B., Seurin,Y.: Hash Functions and RFID Tags: Mind the Gap. CHES'08, LNCS, vol. 5154, pp. 283–299. Springer (2008)
3. De Cannière,C., Dunkelman,O., Knežević,M.: KATAN & KTANTAN: a Family of Small and Efficient Hardware-Oriented Block Ciphers. CHES'09, LNCS, vol. 5747, pp. 272–288, Springer (2009)
4. De Cannière,C., Preneel,B.: Trivium Specifications. eSTREAM technical report (2006) http://www.ecrypt.eu.org/stream/ciphers/trivium/trivium.pdf
5. Daemen,J., Govaerts,R., Vandewalle,J.: A Hardware Design Model for Cryptographic Algorithms. ESORICS'92, LNCS, vol. 648, pp. 419–434. Springer (1992)
6. Daemen,J., Govaerts,R., Vandewalle,J.: A Framework for the Design of One-Way Hash Functions Including Cryptanalysis of Damgård One-way Function based on a Cellular Automaton. ASIACRYPT'91, LNCS, vol. 739, pp. 82–96. Springer (1991)
7. Damgård, I.B.: A Design Principle for Hash Functions. CRYPTO'89, LNCS, vol. 435, pp. 416–427. Springer (1989)

8. Feldhofer,M., Dominikus,S., Wolkerstorfer,J.: Strong Authentication for RFID Systems Using the AES Algorithm. CHES'04, LNCS, vol. 3156, pp. 357–370, (2004)
9. Feldhofer,M., Rechberger,C.: A Case Against Currently Used Hash Functions in RFID Protocols. On the Move to Meaningful Internet Systems OTM'06, LNCS, vol. 4277, pp. 372–381. Springer (2006)
10. Garber,D.: Braid Group Cryptography. CoRR, vol. abs/0711.3941, pp. 1–75 (2007)
11. Good,T., Chelton,W., Benaissa,M.: Hardware Results for Selected Stream Cipher Candidates. Presented at the *State of the Art of Stream Ciphers SASC'07*, Bochum, Germany (2007)
12. Hell,M., Johansson,T., Meier,W.: Grain: a Stream Cipher for Constrained Environments. International Journal of Wireless and Mobile Computing. vol. 2, pp. 86–93 (2007)
13. Hong,D., Sung,J., Hong,S., Lim,J., Lee,S., Koo,B.S., Lee,C., Chang,D., Lee,J., Jeong,K., Kim,H., Kim,J., Chee,S.: HIGHT: a New Block Cipher suitable for Low-Resource Device. CHES'06, LNCS, vol. 4249, pp. 46–59. Springer (2006)
14. Lim,C., Korkishko,T.: mCrypton: A Lightweight Block Cipher for Security of Lowcost RFID Tags and Sensors. Information Security Applications WISA'05, LNCS, vol. 3786, pp. 243–258. Springer (2005)
15. Merkle,R.C.: One way Hash Functions and DES. CRYPTO'89, LNCS, vol. 435, pp. 416–427. Springer (1989)
16. Moldovyan,A.A., Moldovyan,N.A.: A cipher based on data-dependent permutations. Journal of Cryptology, (15):1, pp. 61–72 (2002)
17. Ouafi,K., Vaudenay,S.: Pathchecker: An RFID Application for Tracing Products in Supply-Chains. Presented at the *International Conference on RFID Security 2009*, Leuven, Belgium (2009)
18. Poschmann,A., Leander,G., Schramm,K., Paar,C.: New Lightweight DES Variants Suited for RFID Applications. FSE'07, LNCS, vol. 4593, pp. 196–210. Springer (2007)
19. Rolfes,C., Poschmann,A., Leander,G., Paar,C.: Ultra-Lightweight Implementations for Smart Devices - Security for 1000 Gate Equivalents. CARDIS 2008, LNCS, vol. 5189, pp. 89–103. Springer (2008)
20. Secure Hash Standard. *Federal Information Processing Standard* publication #180-2. U.S. Department of Commerce, National Institute of Standards and Technology (2002)
21. Tillich,J.P., Zémor,G.: Hashing with SL_2. CRYPTO'94, LNCS, vol. 839, pp. 40–49. Springer (1994)
22. Wheeler,D.J., Needham,R.M.: TEA: a Tiny Encryption Algorithm. FSE'94, LNCS, vol. 809, pp. 363–366. Springer (1994)
23. Yu,Y., Yang,Y., Fan,Y., Min,H.: Security Scheme for RFID Tag. Technical report WP-HARDWARE-022, Auto-ID Labs white paper (2006) http://www.autoidlabs.org/single-view/dir/article/6/230/page.html
24. Yoshida,H., Watanabe,D., Okeya,K., Kitahara,J., Wu,J., Küçük,Ö., Preneel,B.: MAME: A Compression Function With Reduced Hardware Requirements. CHES'07, LNCS, vol. 4727, pp. 148–165. Springer (2007)