The Curupira-2 Block Cipher for Constrained Platforms: Specification and Benchmarking

Marcos Simplício Jr[1], Paulo S. L. M. Barreto[1] *, Tereza C. M. B. Carvalho[1], Cintia B Margi[1], and Mats Näslund[2]

[1]Laboratory of Computer Architecture and Networks, PCS-EP, University of São Paulo, Brazil
{mjunior,pbarreto,carvalho,cbmargi}@larc.usp.br
[2]Ericsson Research - Stockholm, Sweden
mats.naslund@ericsson.com

Abstract. Privacy is a key concern in Location Based Applications (LBAs), especially due to their intensive use resource constrained devices in which general purpose ciphers are difficult to deploy. In this paper, we address this issue by specifying a new, faster key-schedule algorithm for the Curupira block cipher. This special-purpose cipher follows the Wide Trail Strategy (such as AES) and is tailored for resource-constrained platforms, such as sensors and mobile devices. Furthermore, we present our benchmark results for both the Curupira-1 (which adopts the original key-schedule specification) and the Curupira-2 (which adopts the new one) in appropriate testbeds.
Keywords: location based services, symmetric cryptography, involutional block ciphers, sensor networks, constrained platforms, ciphers benchmark.

1 Introduction

The continuous and widespread development of context-aware applications based on Wireless Sensor Networks (WSNs) shows their potential to allow a high level of integration between computers and the physical environment. Location-Based Applications (LBAs) play an important role in this scenario, automatically adapting their behaviors to the available geo-spatial location information and the nearby sensors and devices. This way, these systems are able to provide both novel and more effective services. However, due to the sensitive nature of the data involved (both location information and other data collected by the network nodes) the security of the communication in these applications is an important concern.

Battery-powered sensor nodes and other limited platforms normally employed in LBAs impose several constraints over the cryptographic algorithms

* Supported by the Brazilian National Council for Scientific and Technological Development (CNPq) under grant 312005/2006-7. †This work was supported by the Research and Development Centre, Ericsson. Telecomunicações S.A., Brazil

that can be effectively deployed. For example, commercial motes usually have a memory size of 8-12 KB for code and 512-4096 bytes of RAM, as well as 4-16 MHz processors [21, 17]. Moreover, messages exchanged in these applications are frequently small, a typical packet being 24 bytes in length [22, 30]. In this context, complex all-purpose algorithms not only take longer to run but also consume more energy, which motivates the research for more efficient alternatives.

To date, many architectures have been proposed to provide security in WSNs. One of the most popular is TinySec [19], which offers link layer security to TinyOS [18], the *de facto* standard operating system for sensor networks. As default block cipher, TinySec has chosen Skipjack [33] due to its superior performance. Meanwhile, RC5 [34] was not considered as an alternative since it is considered to be encumbered with patents and, even if it can run faster than Skipjack when the round keys are pre-computed, this incurs extra RAM requirements [15]. However, as Skipjack uses relatively small (80-bit) keys and 31 out of its 32 rounds can be successfully cryptanalyzed [9], it presents a very low margin of security. These observations lead to security concerns regarding TinySec, as well as other architectures based on the same cipher, such as TinyKeyMan [24], MiniSec [25] and Sensec [23].

The literature includes numerous analyses of modern cryptographic algorithms, aiming to identify those well-suited to WSNs both in terms of security and performance. One of the most extensive is presented by [21], which concludes that Skipjack is the most energy-efficient of all surveyed ciphers, while the Advanced Encryption Standard (AES) [32] and MISTY1 [27] are considered reasonable alternatives in scenarios with higher security requirements. However, MISTY1 is considered to be encumbered with patents, while AES has larger code, memory, and energy requirements, as well as a block size which is too big for sensor networks. It could also be possible to rely on cryptographic hardware [16], but this is not a solution available in many modern devices.

An alternative to address these issues is the adoption of Curupira [6], a special-purpose block cipher specially developed with constrained platforms in mind. The cipher follows the Wide Trail strategy [11], such as the AES itself, which assures a good security against cryptanalysis. Also, it presents an involutional structure (meaning that the encryption and decryption processes are identical except by the key-schedule) and is very flexible in terms of implementation. This way, the cipher is well adapted to resource constrained scenarios such as those faced by LBAs.

In this paper, we propose a new, faster key-schedule algorithm for the Curupira algorithm and analyze its security. We also present a benchmark comparing Skipjack, AES and Curupira in relevant platforms. In order to discern between the two versions of the Curupira cipher, we write 'Curupira-1" for the one adopting the original key-schedule and "Curupira-2" for the new specification. We write simply "Curupira" when the discussion applies to both.

This document is organized as follows. We introduce basic mathematical tools and notation in section 2. An overview of the Curupira-1, including its key-schedule algorithm, is given in section 3. The second version of the key-schedule

is presented in section 4, which also discuss security, performance and implementation issues for the resultant CURUPIRA-2 block cipher. Our benchmark results are presented in section 5. We conclude in section 6.

2 Mathematical preliminaries and notation

The finite field $\mathrm{GF}(2^n)$ will be represented as $\mathrm{GF}(2)[x]/p_n(x)$, where $p_n(x)$ is a primitive pentanomial of degree n over $\mathrm{GF}(2)$, i.e. $\deg(p_n(x)) \equiv n$. This way, all multiplications over $\mathrm{GF}(2^8)$ are made modulo $p_8(x) = x^8 + x^6 + x^3 + x^2 + 1$. This choice of $p_8(x)$ incurs in a simple form for the primitive cube root of unity, $c(x) = x^{85} \bmod p_8(x) = x^4 + x^3 + x^2$.

An element $u = u_7x^7 + \ldots + u_ix^i + \ldots + u_0$ of $\mathrm{GF}(2^8)$ where $u_i \in \mathrm{GF}(2)$ for all $i = 0, \ldots, 7$, will be denoted by the numerical value $u_7 \cdot 2^7 + \ldots + u_i \cdot 2^i + \ldots + u_0$, written in hexadecimal notation. Thus, the polynomial $c(x)$ is written `1C`, while $p_8(x)$ is written `14D`. The multiplication by the polynomials x and $c(x)$ are denoted `xtimes` and `ctimes`, respectively.

The set of all $3 \times n$ matrices over $\mathrm{GF}(2^m)$ is denoted by $\mathbb{M}_n$. Let D and E denote the MDS matrices (see [26] for a definition) as follows:

$$D = \begin{bmatrix} 3 & 2 & 2 \\ 4 & 5 & 4 \\ 6 & 6 & 7 \end{bmatrix}, \quad E = \begin{bmatrix} \texttt{1+c(x)} & \texttt{c(x)} & \texttt{c(x)} \\ \texttt{c(x)} & \texttt{1+c(x)} & \texttt{c(x)} \\ \texttt{c(x)} & \texttt{c(x)} & \texttt{1+c(x)} \end{bmatrix}.$$

3 Overview of the CURUPIRA-1 structure

This section gives a brief description of the CURUPIRA-1 original specification. For further details, we refer to [6].

The CURUPIRA is a block cipher specially tailored for constrained platforms. It operates on 96-bit data blocks (organized as $\mathbb{M}_4$ matrices, mapped by columns instead of by rows) and accepts 96-, 144- or 192-bit keys, with a variable number of rounds. The cipher round structure is similar to the one in BKSQ [12], with the advantage of being involutional, resulting in a more compact cipher. Its round function structure is used for both CURUPIRA-1 and CURUPIRA-2 and is composed by the following self-inverse transforms (see Figure 1:

- *Nonlinear Layer* (γ): all bytes in the block pass through a highly nonlinear S-Box, identical to that used in ANUBIS [7] and KHAZAD [8] block ciphers;
- *Linear Diffusion Layer*(θ): the block is left-multiplied by the MDS matrix D (the one defined in section 2), which results in intra-columnar diffusion;
- *Permutation Layer* (π): all the bytes in the second and third rows of the block are permuted according to the rule $\pi(a) = b \Leftrightarrow b_{i,j} = a_{i,i\oplus j},\ 0 \leqslant i < 3,\ 0 \leqslant j < n$;
- *Key addition Layer* (σ): the round key is XORed with the block.

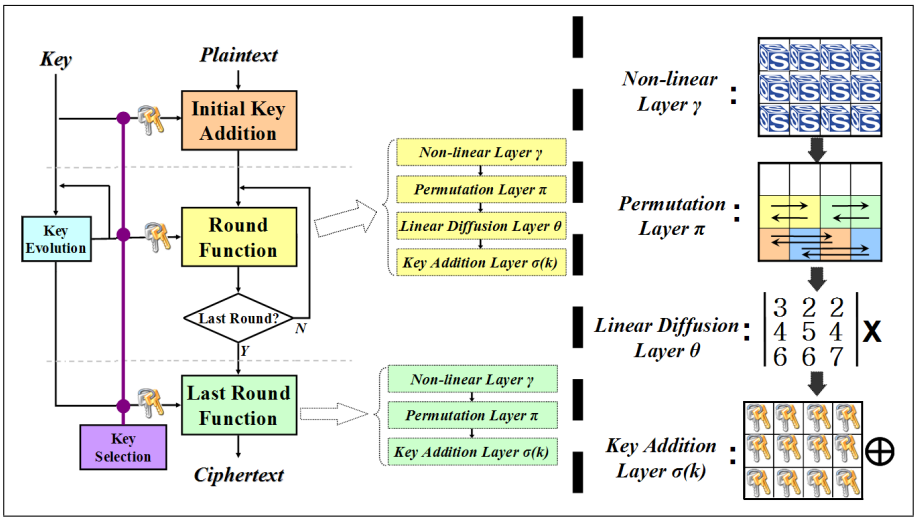

Fig. 1. Curupira Round Structure

These transforms only involve basic operations such as table lookups, XORs and byte shifts. Thus, they can be implemented in most platforms in a very efficient way. Nonetheless, when space is available, they can be further accelerated using pre-computed tables, operating over entire columns of the block instead of byte-to-byte.

3.1 The Curupira-1 key-schedule

The Curupira-1 key-schedule algorithm is easily invertible and follows a structure closely related to the one dictated by the Wide Trail Strategy, which assures a high diffusion speed. Also, it has the advantage of being cyclical, which means that the original key is recovered after a certain number of rounds, avoiding the need of storing any intermediary sub-key during both encryption and decryption.

In this first construction, a $48t$-bit user key $\mathcal{K}$, $2 \leqslant t \leqslant 4$ is internally represented as a matrix $K \in \mathbb{M}_{2t}$. To generate a sub-key K^{n+1} from its predecessor K^n, the sub-keys pass through three different transformations (illustrated in Figure 2):

- Constant Addition ($\sigma(q)$): a set of nonlinear constants, incrementally taken from the S-Box, are XORed with the bytes in the *first row* of the round key; this way, for a $48t$-bit key, the r^{th} round and the column j, the $q_j^{(r)}$ constant is given by: $q_j^{(0)} = 0$ and $q_j^{(r)} = S[2t(r-1)+j]$, $0 \leqslant j \leqslant 2t$;
- Cyclic Shift (ξ): rotates the second row one position to the left, and rotates the third row one position to the right, keeping the first row unchanged;
- Linear Diffusion (μ): the round-key is left-multiplied by the matrix E (defined in section 2).

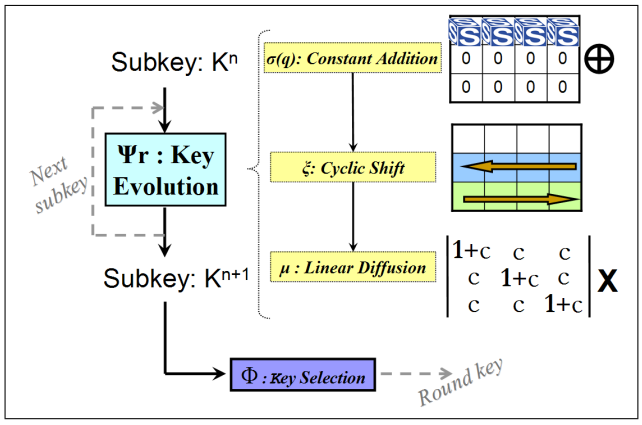

Fig. 2. The original key-schedule specification (adopted by CURUPIRA-1)

Furthermore, the round keys $\kappa^{(r)}$ effectively combined with the data blocks are chosen by the Key Selection algorithm ϕ_r, which applies the S-Box to the first row of the sub-key K^r and truncates it to 96 bits (i.e., the size of the block). Thus, even if the Key Selection is not part of the key evolution, it adds nonlinearity to the key-schedule.

4 The CURUPIRA-2 key-schedule

The CURUPIRA-1 key-schedule specification is quite conservative, aiming for a high security level against related-key attacks. Thus, it is reasonable to adopt a simpler yet secure key-schedule algorithm in order to improve the overall cipher performance. That is the approach of the CURUPIRA-2 key-schedule, which will be described in the following sections.

4.1 Key representation

A $48t$-bits long user key $\mathcal{K}$ ($2 \leqslant t \leqslant 4$) is internally represented as an element K belonging to the finite field $\mathrm{GF}(2^{48t}) = \mathrm{GF}(2)/p_{48t}(x)$, where $p_{48t}(x)$ is a pentanomial in $\mathrm{GF}(2)$ chosen in such a way that x^8 is a primitive root of $p(x)$. An element $u(x)$ in this field can be seen as a byte vector, i.e. $u = (U_{6t-1}, \ldots, U_0)$, where U_0 indicates the lesser significant byte. This way, the cryptographic key $\mathcal{K}$ is directly mapped to K, starting at its most significant byte, K_{6t-1}.

A pentanomial representation was chosen because primitive pentanomials are available for all values of t used by the CURUPIRA. In fact, the use of trinomials would result in a better performance, but unfortunately they do not exist for fields of type $\mathrm{GF}(2^n)$ when n is multiple of 8 [28] and, thus, they cannot be used for any of the cipher key sizes.

For reasons that will become clearer in section 4.3, the pentanomials chosen for CURUPIRA-2 are:

- $p_{96}(x) = x^{96} + x^{16} + x^{13} + x^{11} + 1$;
- $p_{144}(x) = x^{144} + x^{56} + x^{53} + x^{51} + 1$;
- $p_{192}(x) = x^{192} + x^{43} + x^{41} + x^{40} + 1$.

4.2 Schedule constants

The *schedule constants* are denoted $q^{(s)}$, where the index $^{(s)}$ indicates the round in which they are applied. As in the CURUPIRA-1, the constants are directly taken from the S-box and, thus, no extra storage is needed. This time, however, they are interpreted as elements of $\mathrm{GF}(2^{48t})$ in such a way that $q^{(0)} = 0$ and, for $s > 0$, $q^{(s)} = (S[s-1], 0 \ldots, 0)$ i.e. a single S-box output is mapped to the most significant byte of $q^{(s)}$. As shown in the next section, this is a strategic position that makes each constant affect exactly 3 bytes of the round key right after its addition.

4.3 The key evolution Υ_s

The sub-keys are updated during the cipher operation by means of two operations: a reversible transform, $\aleph : \mathrm{GF}(2^{48t}) \to \mathrm{GF}(2^{48t})$; and an auto-inverse transform $\eta : \mathrm{GF}(2^{48t}) \to \mathrm{GF}(2^{48t})$. They are defined in such a way that $\aleph(u) \equiv u \cdot x^8$ and, for the polynomials $u = (U_{6t-1}, \ldots, U_0)$ and $v = (V_{6t-1}, \ldots, V_0)$ in $\mathrm{GF}(2^{48t})$:

$$v = \eta(u) \Leftrightarrow \begin{cases} V_i = U_{11-i} \oplus U_{12+i} & \text{if } 0 \leqslant i < 6t - 12; \\ V_i = U_i & \text{otherwise.} \end{cases}$$

Together with the schedule constants, these transforms compose the *key evolution function* $\Upsilon_r : \mathrm{GF}(2^{48t}) \to \mathrm{GF}(2^{48t})$, defined as $\Upsilon_r(u) \equiv \eta \circ \aleph(u \oplus q^{(r)})$, in such a way that $K^{(0)} \equiv K$ and $K^{(r+1)} = \Upsilon_r(K^{(r)})$.

The η transform is used to ensure a greater diffusion to the schedule when keys greater than 96 bits are adopted, since it combines some of the least significant bytes of the key with the most significant ones. Without this operation, two 144-bit keys differing only at the byte K_{11} would generate sub-keys whose 12 least significant bytes would be identical for the first 6 rounds; also, for 192-bit keys, this result would hold true for the first 12 sub-keys. As these least significant bytes are the ones actually selected in each round, the effect of the η transform is essential to assure a higher diffusion speed for 144- and 192-bit keys.

Also, the $\aleph$ transform is particularly interesting due to its performance, specially on resource-constrained platforms, as stated in the following theorem:

Theorem 1. *Let $p(x) = x^n + x^{k_3} + x^{k_2} + x^{k_1} + 1$ be a primitive pentanomial of degree $n = bw$ over $\mathrm{GF}(2)$ such that $k_3 > k_2 > k_1$, $k_3 - k_1 \leqslant w$, and either $k_3 \bmod w = 0$ or $k_1 \bmod w = 0$. Then multiplication by x^w in $\mathrm{GF}(2^n) = \mathrm{GF}(2)[x]/p(x)$ can be implemented with no more than 5 XORs and 4 shifts on w-bit words. Moreover, if 2×2^w bytes of storage are available, the cost drops to no more than 2 XORs on w-bit words and 2 table lookups.*

Proof. For $u = \bigoplus_{d=0}^{n-1} (u_d x^d) \in \mathrm{GF}(2^n)$, let $U_i = u_{wi+w-1} x^{w-1} + \ldots + u_{wi}$ where $i = 0, \ldots, b-1$, so that $u = U_{b-1} x^{w(b-1)} + U_{b-2} x^{w(b-2)} + \ldots + U_0$, which for brevity we write $u = (U_{b-1}, \ldots, U_0)$. Then one can compute $u \cdot x^w$ as:

$$\begin{aligned}
&(U_{b-1} x^{w(b-1)} + U_{b-2} x^{w(b-2)} + \ldots + U_0) \cdot x^w = \\
&U_{b-1} x^n + U_{b-2} x^{w(b-1)} + \ldots + U_0 x^w = \\
&U_{b-2} x^{w(b-1)} + \ldots + U_0 x^w + U_{b-1}(x^{k_3} + x^{k_2} + x^{k_1} + 1) = \\
&(U_{b-2}, \ldots, U_0, U_{b-1}) \oplus U_{b-1}(x^{k_3} + x^{k_2} + x^{k_1}).
\end{aligned}$$

Assume that $k_3 = w(k+1)$ for some k; the case $k_1 \bmod w = 0$ is handled analogously. Thus:

$$u \cdot x^w = (U_{b-2}, \ldots, U_0, U_{b-1}) \oplus U_{b-1}(x^w + x^{w-k_3+k_2} + x^{w-k_3+k_1}) x^{wk}$$

Since $\deg(U_{b-1}) \leqslant w-1$ and $\deg(x^w + x^{w-k_3+k_2} + x^{w-k_3+k_1}) = w$, their product is a polynomial of degree not exceeding $2w-1$, and hence it fits two w-bit words for any value of U_{b-1}. Besides, multiplication of this value by x^{wk} corresponds to simply displacing it k words to the left. We can define:

$$\begin{aligned}
T_1[U] &\equiv U \oplus (U \gg (w - k_3 + k_2))) \oplus (U \gg (w - k_3 + k_1))), \\
T_0[U] &\equiv (U \ll (k_3 - k_2)) \oplus (U \ll (k_3 - k_1));
\end{aligned}$$

Thus, we can write $u \cdot x^w = (U_{b-2}, \ldots, U_k \oplus T_1[U_{b-1}], U_{k-1} \oplus T_0[U_{b-1}], \ldots, U_0, U_{b-1})$. The values T_1 and T_0 can be either computed on demand or else pre-computed and stored in two 2^w-entry tables. One easily sees by direct inspection that the computational cost is that stated by the theorem.

Applying this theorem for the polynomials adopted by the Curupira-2, we can evaluate the cost of the transforms $\aleph$ and its inverse:

$p_{96}(x) = x^{96} + x^{16} + x^{13} + x^{11} + 1$:
$\aleph : (U_{11}, \ldots, U_0) \cdot x^8 = (U_{10}, \ldots, U_1 \oplus T_1[U_{11}], U_0 \oplus T_0[U_{11}], U_{11})$;
$\aleph^{-1} : (U_{11}, \ldots, U_0) \cdot x^{-8} = (U_0, U_{11}, \ldots, U_2 \oplus T_1[U_0], U_1 \oplus T_0[U_0])$;

$p_{144}(x) = x^{144} + x^{56} + x^{53} + x^{51} + 1$:
$\aleph : (U_{17}, \ldots, U_0) \cdot x^8 = (U_{16}, \ldots, U_6 \oplus T_1[U_{17}], U_5 \oplus T_0[U_{17}], \ldots, U_0, U_{17})$;
$\aleph^{-1} : (U_{17}, \ldots, U_0) \cdot x^{-8} = (U_0, U_{17}, \ldots, U_7 \oplus T_1[U_0], U_6 \oplus T_0[U_0], \ldots, U_1)$;

$p_{192}(x) = x^{192} + x^{48} + x^{45} + x^{43} + 1$:
$\aleph : (U_{23}, \ldots, U_0) \cdot x^8 = (U_{22}, \ldots, U_5 \oplus T_1[U_{23}], U4 \oplus T_0[U_{23}], \ldots, U_0, U_{23})$.
$\aleph^{-1} : (U_{23}, \ldots, U_0) \cdot x^{-8} = (U_0, U_{23}, \ldots, U_6 \oplus T_1[U_0], U_5 \oplus T_0[U_0], \ldots, U_1)$.

For all key sizes, we have $T_0 = U \oplus (U \gg 5) \oplus (U \gg 3)$ and $T_1 = (U \ll 3) \oplus (U \ll 5)$.

These equations show that both $\aleph$ and $\aleph^{-1}$ transforms have the same cost and, thus, it is also valid for Υ_r and its inverse, $\Upsilon_r^{-1}(u) \equiv (\aleph^{-1} \circ \eta(u)) \oplus q^{(r)}$. In contrast, when compared to the key-schedule of the Curupira-1, this second schedule algorithm has one disadvantage: there is no simple way to reinitialize the key after a reduced number of rounds. However, in many applications, its higher speed both during encryption and decryption can be a much more interesting feature, compensating its lack of cyclicity.

4.4 The key selection ϕ_r^*

The round keys $\kappa^{(r)} \in \mathbb{M}_n$ effectively used in each round are calculated by means of the *key selection function* $\phi_r^* : \mathrm{GF}(2^{48t}) \to \mathbb{M}_n$, defined in such a way that:

$$\kappa^{(r)} = \phi_r^*(K) \Leftrightarrow \begin{cases} \kappa_{i,j}^{(r)} = S[K_{i+3j}^{(r)}] \text{ if } i = 0; \\ \kappa_{i,j}^{(r)} = K_{i+3j}^{(r)} \text{ otherwise.} \end{cases}$$

This way, only the 12 least significant bytes are taken by ϕ_r^*. Also, the S-box is applied to the bytes that will be combined with the first row of the block, adding nonlinearity to the key-schedule, while the bytes for the other rows are taken directly. The whole process involved in this second version of the key-schedule algorithm is depicted in Figure 3.

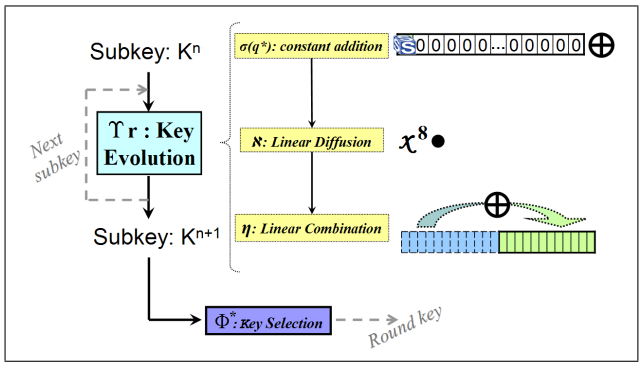

Fig. 3. The new key-schedule specification (adopted by CURUPIRA-2)

4.5 Security analysis revisited

Since, for both versions of our cipher, the round structure remains the same, most of the CURUPIRA-1 security analysis [6, section 4] also applies to the CURUPIRA-2: the adoption of the Wide Trail Strategy in combination with a highly nonlinear S-Box thwarts the most well known modalities of attacks, such as linear, differential and integral cryptanalysis. As a consequence, no attack faster than exhaustive search was found for more than 7 rounds of the cipher. These results were also confirmed by third party analysis [31]. The main difference between the two analysis concerns the existence of weak keys and the viability of related-key attacks.

Weak keys are keys that result in a block cipher mapping with detectable weaknesses, which normally occurs when the nonlinear operations depend on the actual key value. This is not the case for the CURUPIRA, where keys are applied using XOR and all nonlinearity is fixed in the S-box. Also, the nonlinear round

constants considerably reduce the probability of fixed points in the key-schedule process, making the existence of weak keys very unlikely.

Related-key attacks exploit a known relationship between different unknown keys, leading to a predictable behavior for the sub-keys generated by the key schedule. Some of the most widespread techniques involve key differentials and key rotations (cf. [10]) in order. Due to its slower diffusion, it is clear that it is easier to find relationships between subsequent sub-keys in Curupira-2 than in Curupira-1, making the former less resistant to related-key attacks. In fact, between any two rounds, the difference in a single internal byte (i.e. in a position that will only be shifted as a result of the multiplication by x^8) results in a difference on a single byte of the next sub-key, which could be somehow exploited. In spite of this, some fundamental elements are introduced to the key-schedule proposed in this paper in order to prevent attacks. First, the nonlinearity introduced by the key selection thwarts related-key attacks involving differentials. Second, the generation of sub-keys does not involve simple rotations, but rather a multiplication over $\mathrm{GF}(2^{48t})$ after the addition of nonlinear constants. Third, the truncation of the sub-keys make some advanced related-key attack variants such as that described in [13, section 4] improbable. Finally, the slow diffusion in the key schedule is counterbalanced by the round function fast diffusion, assuring that each byte of the key affects many block bytes after a few rounds. Together, these features make this kind of attacks unlikely to work against the full cipher.

Furthermore, for key lengths that are larger than the length of one round key, the existence of sets of keys that produce identical values for at least one round key is inevitable. Thus, even if the η transform adds diffusion power to the key-schedule and prevents the existence of trivial sets with this property, they should be more easy to find than in the Curupira-1. Even so, it remains unclear how such keys could possibly be successfully used in a related-key attack.

As a last remark, the Curupira structure involves only simple operations such as XORs, shifts and table lookups. As long as the running times for these transforms are not data-dependent on the target platform, the cipher implementation can avoid many side-channel attacks (such as timing-attacks [20]) in a straightforward way.

4.6 Implementation and Performance Issues

The Curupira-2 algorithm is very flexible in terms of implementation, offering many memory/performance trade-offs. It cannot only use the same techniques developed for the Curupira-1 round functions [6, section 5]. These include the usage of a few tables with pre-computed results, but its key-schedule also allow some useful optimizations depending on resources available.

The round keys can be either computed on-demand or fully pre-calculated and stored in a table for ready access. In the first case, the cipher requires a reduced amount of RAM memory, since only one sub-key is stored at any given time. However, as the key-schedule of Curupira-2 is not cyclic, there is no easy way to compute the first round key from the last one. An easy way to overcome

this problem is to use two arrays k_a and k_b to store the first and the last sub-keys, respectively: when one wants to encrypt, it suffices to copy k_a into k_b and reuses k_a memory space to create the encryption sub-keys; in the end, k_a will have the last key while k_b will store the first one, assuring that both sub-keys are always available. The decryption is handled analogously. In this case, the last sub-key could be computed during the cipher initialization. We note that this strategy is only possible because the key schedule is easily invertible.

For 6t-byte keys ($2 \leqslant t \leqslant 4$), the round sub-keys can be calculated in any direction at the cost of one circular permutation, 2+6(t-2) XORs and one computation of T_0 and T_1. Also, T_0 and T_1 can be either implemented using two 256-bit tables or calculated on-the-fly, taking 1 XOR + 2 shifts and 2 XORs + 2 shifts, respectively. In fact, the circular permutation does not need to be effectively implemented: the same effect can be achieved if the index corresponding to the most significant byte of the key is stored and used as the first byte of the key for every calculation; this way, it suffices to update this index after each invocation of the $\aleph$ and $\aleph^{-1}$ operations.

Reviewing the CURUPIRA preliminary calculations [6, section 5.1], the cost of its round function is $3R - 1$ XORs, $2(R-1)/3$ `xtimes` operations and R S-box lookups per byte. When the key-schedule and key selection are taken into account, we add at most 1/3 S-Box lookups, 1/3 `ctimes` operations and 2 XORs per key byte and per round in the CURUPIRA-1, while this cost drops to at most 5/12 S-Box lookups, 5/8 XORs and 1/12 computations of T_0 and T_1 per key byte and per round in the CURUPIRA-2. In comparison, Skipjack takes basically 48 XORs and 16 F-table lookups per encrypted byte. Thus, supposing that the cost of any of these basic operations are approximately the same and not counting auxiliary operations not directly related to the ciphers structures (such as counter increments and key index updates), CURUPIRA-1 with 96-bit keys and 10 rounds is about $(45+27)/64 \approx 112.5\%$ as Skipjack when the round keys computed on demand. On the other hand, CURUPIRA-2 with the same key-size corresponds to $(45 + 7.5)/64 \approx 82\%$ of Skipjack computation in the same conditions. This result should be expected for similar implementations of both ciphers on byte-oriented platforms.

Furthermore, more powerful processors (32-bit servers, for example) could implement the Υ_s transform in a more efficient way, operating over columns instead of bytes. Also, the multiplication by x^8 can be easily implemented using a single table that calculates T_0 and T_1 at the same time, an approach similar to those adopted in some very optimized versions of AES [14].

5 Benchmark

In this section, we present the results of our comparison between CURUPIRA, Skipjack and AES in terms of processing time and memory usage. As discussed in section 1, the motivation behind the choice of Skipjack resides in the results presented in [35] and in [21] which shows that, in spite of its low security level, the cipher is a very interesting choice to achieve a high performance in constrained

platforms, surpassing many other hardware-oriented ciphers like MISTY1 and Kasumi [1]. AES, on the other hand, provides higher security but is recommended for less constrained platforms, since it is a less memory-efficient cipher. Considering these remarks, we decided to develop a deep analysis on the comparison of Skipjack and CURUPIRA in both constrained and powerful platforms, while AES is taken into account only on powerful ones. Three different platforms were chosen as testbeds:

- Microcontroller (8 bits): a RISC microcontroller PIC18F8490 [29] equipped with a 8MHz processor, 768 bytes of RAM and a memory size of 16KB for code. The reason behind this choice resides in its capacity, slightly superior to the one presented by the ATmega8535 [2]. This last device, with a 4 MHz processor, 8 KB of flash memory and 512 bytes of RAM, is the one used in the Smart Dust Project [17] for sensor networks.
- Microcontroller Simulator (8 bits): Avrora version 1.6.0 - Beta [36], simulating a microcontroller from the ATmega128 [3] series. The goal of using this simulator is mainly to validate the results obtained with the PIC processor in a more powerful, yet tiny platform.
- Pentium 4 (32 bits): a notebook equipped with Pentium 4 (3.2GHz) and 1GB of RAM. This platform was chosen to evaluate the proposed optimizations of the cipher when the resources in the target platform are abundant.

5.1 Implementation Characteristics

For the 8-bit versions of CURUPIRA and Skipjack, the same C-written implementations were analyzed in both the PIC18F84908 and the Avrora simulator. Furthermore, they adopt similar interfaces in each of these platforms, in order to assure a fair comparison. They also are more speed-oriented than memory-oriented, since the consumption of energy with processing is proportional to the number of operations performed by the algorithm, and this is normally considered the most critical resource in constrained platforms, particularly in WSNs.

For the implementation running on Avrora, as recommended by its documentation, we adopted *avr-objdump* and *avr-gcc* (both GNU utilities) as compilation tools, while *MPLAB IDE v7.60* and *MPLAB C18* compiler are used together with the PIC microcontroller. The speed-optimized versions of each cipher, resulting from the available compiler optimizations, are the ones considered in this document. It is important to notice that, even if both platforms include indirect addressing in their instruction sets, our tests showed that the compilers were not able to fully take advantage of these instructions, resulting in less than optimal machine codes when pointers and/or matrices were used. That is the reason why we decided to evaluate two different programming techniques: one that uses pointers and matrices and another that uses basic-type variables more intensively, avoiding indirect addressing. While the first approach normally results in more flexible code (where the size of the keys can be more easily changed, for example), the second allows more optimized implementations with fixed-size keys (enabling loop unrolling with little loss of compactness)

The implementations running on the 8-bit platforms are detailed below:

Curupira (8 bits) using the proposed optimizations for constrained platforms, we elected two versions of the Curupira for each scheduling algorithm:

1. Curupira$_c$-1: complete version (meaning that it accepts all key sizes) of the Curupira-1. It requires two 256-byte tables, one for the S-Box and another for the `ctimes` operation and uses many pointers and matrices
2. Curupira$_c$-2: complete version of the Curupira-2, using two 256-byte tables for the S-Box and `xtimes` operations. Such as Curupira$_c$-1, it is also based on indirect addressing instructions.
3. Curupira$_{k^{96}}$-1: Curupira-1 restricted to 96-bit keys. It uses the same tables as the Curupira$_c$-1, but relies on basic types instead of indirect addressing instructions.
4. Curupira$_{k^{96}}$-2: Curupira-2 restricted to 96-bit keys, using the same tables as the Curupira$_c$-2 but relying on basic-type variables.

Skipjack (8 bits) two versions were developed according to the specification:

1. Skipjack$_c$: relies on indirect instructions just like Curupira$_c$-1 and Curupira$_c$-2, providing a useful source of comparison with these ciphers. It uses a single 256-byte F-table and calculates the round keys on demand.
2. Skipjack$_k$: adopts programming strategies similar to those present in the Curupira$_{k^{96}}$, strongly relying on operations over basic-type variables instead of matrices and pointers. Such as the Skipjack$_c$, it also uses a 256-byte F-table and calculates the round keys on-the-fly.

For the 32-bits platform, we decided to take advantage of some highly optimized cipher implementations publicly available. The chosen algorithms, written in Java, were compiled and run on Netbeans IDE 5.5, using the JDK 1.6. The details of the implementations are given below:

AES (32 bits) we adopted the implementation of Barreto [5], which pre-computes the round keys and employs ten 256-word tables to greatly accelerate the cipher operation.

Curupira (32 bits) the optimizations in the algorithm are similar to those present in AES, specially regarding the pre-computation of the round keys and the intensive use of tables, in the same number as AES.

Skipjack (32 bits) the cipher tested is a Java adaptation of Barreto's algorithm [4], originally developed in C language. It operates over 16-bit words and stores some important key-dependent pre-computed values in a 10x256-word matrix; this last operation can be seen as a kind of "key-schedule", since it must be performed each time the cipher key is changed.

5.2 Results: 8-bits platforms

The ciphers memory usage, for both 8-bit platforms, is presented in Table 1. This table shows that all tested versions of the Curupira take more space in memory than Skipjack, an expected result considering the higher complexity of its round function and key-schedule algorithms. Despite this difference, the tested ciphers

Table 1. Memory Occupation (in bytes) of the tested ciphers on the 8 bits platforms

Algorithm	*ROM*	*Code-PIC18F8490*	*Code-Avrora Simulator*
CURUPIRA$_c$-1	822	1444	1648
CURUPIRA$_c$-2	512	1238	1718
CURUPIRA$_{k^{96}}$-1	768	1372	1936
CURUPIRA$_{k^{96}}$-2	512	1532	1846
Skipjack$_c$	256	1012	940
Skipjack$_k$	256	972	1352

are both compact enough to be easily deployed in most constrained platforms, taking less than 3KB as a whole.

Both CURUPIRA and Skipjack do not need to pre-compute the round keys and, thus, they require a reduced amount of RAM. We were not able, however, to directly measure the RAM usage with the tools available for the tested platforms. Nevertheless, due to its greater block and key size, we speculate that CURUPIRA takes a higher amount of RAM than Skipjack. For example, when using two arrays to store the first and the last keys, CURUPIRA-2 would take about twice $((96+2*96)/(64+80))$ the amount of RAM needed by Skipjack. These numbers remain very tiny when compared to pre-computed keys storage, though.

For the PIC microcontroller, Figure 4 shows the number of CPU cycles, per byte encrypted, of both tested ciphers. The time measured corresponds to a single encryption of random blocks using different keys. Even if the CURUPIRA$_c$ implementations allow three different key sizes, only the 96-bit keys are depicted in this graph. Also, we explicitly distinguish the processing time required for the key scheduling and the encryption itself (as Skipjack reuses the original key cyclically, we considered it part of the encryption instead of a "key-schedule").

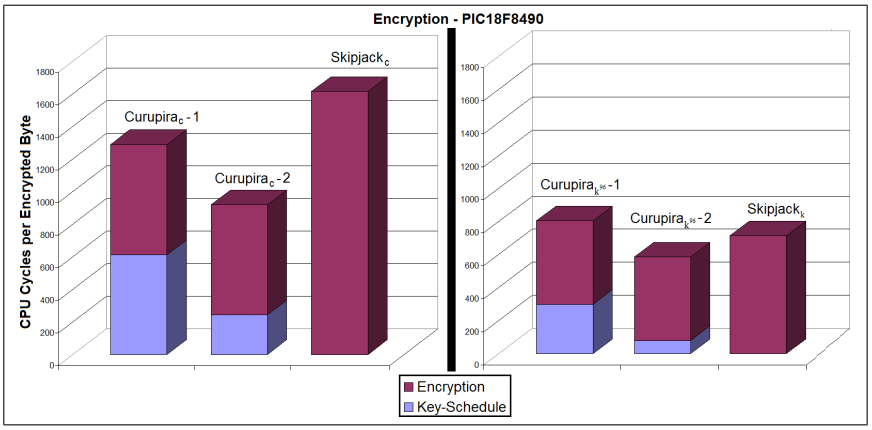

Fig. 4. Comparison between the cipher encryption speeds on the PIC18F8490

The figure shows that CURUPIRA$_c$-1 and CURUPIRA$_c$-2 are respectively $\approx$ 20% and $\approx$ 45% faster than Skipjack$_c$, with the round keys computed on demand. Despite this very positive result, it should be carefully considered since the measured number of cycles includes not only the operations directly involved in the encryption process but also a non-negligible number of auxiliary ones. The influence of these secondary operations is less expressive on both Skipjack$_k$ and CURUPIRA$_{k^{96}}$ and, as depicted in the right side of Figure 4, CURUPIRA$_{k^{96}}$-2 is still $\approx$ 18% faster than Skipjack, while CURUPIRA$_{k^{96}}$-1 is $\approx$ 12% slower. One can see that the cost of both CURUPIRA$_{k^{96}}$ versions in this figure are approximately the ones theoretically calculated in Section 4.6.

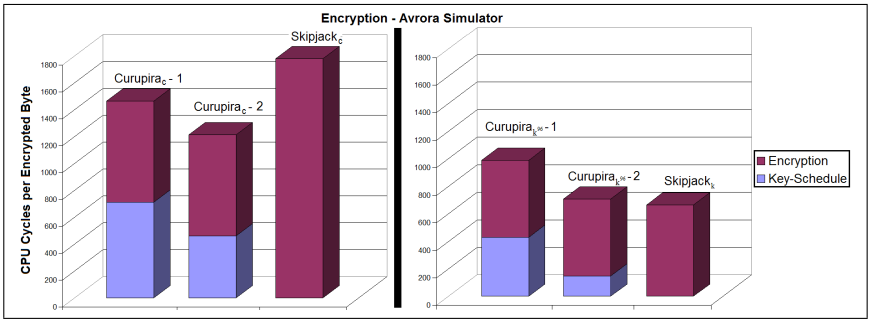

Fig. 5. Comparison between the cipher encryption speeds on the Avrora Simulator

The results on the Avrora Simulator were slightly different from those in the PIC18F8490, as depicted in Figure 5. Skipjack$_k$ speed was considerably improved by this platform change, running about 30% and 4% faster than CURUPIRA$_{k^{96}}$-1 and CURUPIRA$_{k^{96}}$-2, respectively. A further analysis of the assembly codes show that these results were caused by the influence of the compiler, which were able to apply different optimizations to each algorithm. In contrast, this unexpected behavior was not observed with CURUPIRA$_c$ and Skipjack$_c$ implementations, which sustained the relative performances presented on the PIC18F8490.

5.3 Results: 32-bits platforms

The encryption speed of each cipher on the 32-bits platform, with different key sizes (and, thus, number of rounds), is depicted in Figure 6. It is important to point out that, as all round keys are computed at cipher initialization, there is no difference between the encryption speeds of CURUPIRA-1 and CURUPIRA-2.

We obtained similar results for AES and CURUPIRA with the same number of rounds (note the additional graph entry where, for the sake of comparison, both ciphers were tested with the same number of rounds for each key size). This is an expected result since both ciphers adopt well-known optimizations for the Wide Trail Strategy family. On the other hand, the modest result of Skipjack

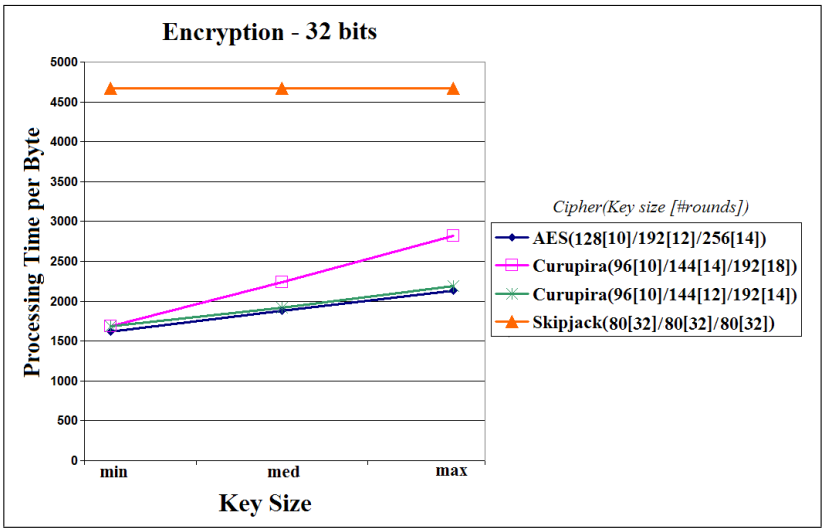

Fig. 6. Encryption Performance x Number of Rounds - 32 bits

(about 3 times slower than the other ciphers) may seem surprising at first sight, since its performance usually is the main factor for its adoption on constrained platforms. However, this can be explained by its 16-bit oriented operations, very attractive on constrained processors but less adapted to fully take advantage of the higher number of bits available on powerful platforms. Both AES and CURUPIRA, though, can easily be implemented to operate over 32- and 24-bit words (columns), respectively.

The processing time involved on the ciphers key-schedules was also measured. As depicted in Figure 7, while CURUPIRA and AES presented similar speeds, Skipjack was about 10 times slower. As this operation has to be performed a single time (at initialization), the impact on the cipher overall operation is reduced on scenarios where the keys are not frequently changed.

6 Conclusions

We have described a new and faster key-schedule proposal for the CURUPIRA block cipher. As a drawback, when compared to the original specification, it has a lower level of security against related-key attacks. However, according to our security analysis, both versions of the full cipher are secure against cryptanalysis.

We also presented a benchmark comparing CURUPIRA, Skipjack and AES in terms of performance and memory occupation, both on constrained and powerful platforms. According to the results obtained, the proposed block cipher is fast and compact, especially when using the new key-schedule presented in this paper. While Skipjack is considered a good candidate for constrained scenarios, such as LBAs dependent on sensors and low-power mobile devices, CURUPIRA is

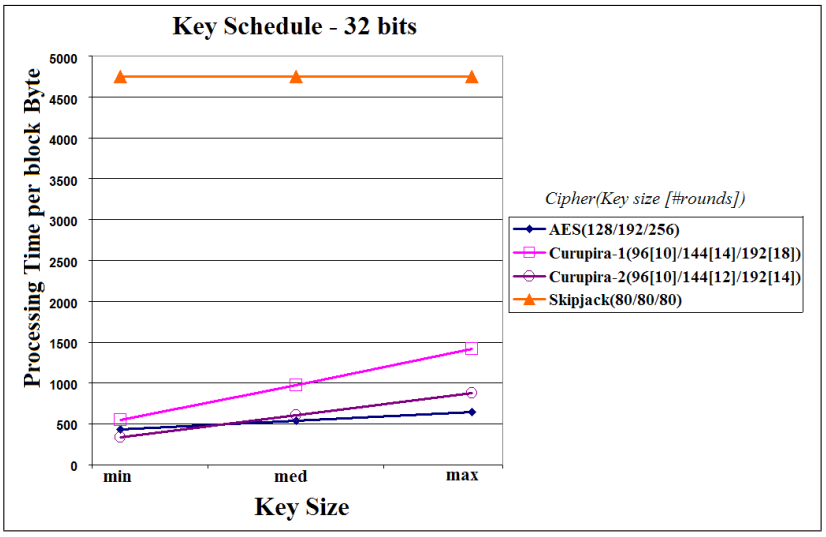

Fig. 7. Key Schedule Performance x Number of Rounds - 32 bits

a suitable alternative to increase the security level and potentially improve performance, introducing a reduced impact in terms of memory usage. Also, when more powerful platforms are also available, the several optimizations allowed by the CURUPIRA can be deployed to obtain an even higher performance in the whole network.

All together, these results show that the CURUPIRA block cipher is a useful solution for providing data encryption at low cost, being recommended for constrained-resource devices and for applications based on such platforms, such as WSNs and LBAs.

6.1 Future and Ongoing work

We are currently working on the deployment of CURUPIRA on a real WSN in order to evaluate its impact (especially in terms of energy consumption) in some significant scenarios. Also, we are developing a new MAC algorithm named MARVIN, designed to provide a low-cost authenticated-encryption scheme on WSNs, particularly when used in conjunction with CURUPIRA.

6.2 Acknowledgments

We would like to thank Richard Gold for his useful comments and the review of this paper.

References

1. 3GPP. Specification of the 3gpp confidentiality and integrity algorithms document 2: Kasumi specification. Technical report, 3GPP, 1999.

2. Atmel. *AVR 8-Bit RISC processor - ATmega8535 (90LS8535)*, 2006.
3. Atmel. *AVR 8-Bit RISC processor - ATmega128 e ATmega128L*, 2007.
4. P. Barreto. The Skipjack block cipher – 32 bit implementation. http://planeta.terra.com.br/informatica/paulobarreto/skipjack32.zip, 1998.
5. P. Barreto. The AES block cipher (rijndael) – 32 bit implementation. http://planeta.terra.com.br/informatica/paulobarreto/JEAX.zip, 2003.
6. P. Barreto and M. Simplicio. CURUPIRA, a block cipher for constrained platforms. In *Anais do 25º Simpsio Brasileiro de Redes de Computadores e Sistemas Distribudos - SBRC 2007*, volume 1, pages 61–74. SBC, 2007.
7. P. S. L. M. Barreto and V. Rijmen. The ANUBIS block cipher. In *First open NESSIE Workshop*, Leuven, Belgium, November 2000. NESSIE Consortium.
8. P. S. L. M. Barreto and V. Rijmen. The KHAZAD legacy-level block cipher. In *First open NESSIE Workshop*, Leuven, Belgium, November 2000. NESSIE Consortium.
9. E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials. In *Advances in Cryptology – Eurocrypt'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 55–64. Springer, 1999.
10. M. Ciet, G. Piret, and J. Quisquater. Related-key and slide attacks: Analysis, connections, and improvements - extended abstract. In *23rd Symposium on Information Theory in the Benelux, Louvain-la-Neuve, Belgium*, pages 315–325, 2002.
11. J. Daemen. *Cipher and hash function design strategies based on linear and differential cryptanalysis.* Doctoral dissertation, Katholiek Universiteit Leuven, March 1995.
12. J. Daemen and V. Rijmen. The block cipher BKSQ. In *Smart Card Research and Applications – CARDIS'98*, volume 1820 of *Lecture Notes in Computer Science*, pages 236–245. Springer, 1998.
13. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved cryptanalysis of RIJNDAEL. In *Fast Software Encryption – FSE'2000*, volume 1978 of *Lecture Notes in Computer Science*, pages 213–230. Springer, 2000.
14. B. R. Gladman. AES second round implementation experience. http://fp.gladman.plus.com/cryptography technology/aesr2/index.htm, 2000.
15. G. Guimaraes, E. Souto, D. Sadok, and J. Kelner. Evaluation of security mechanisms in wireless sensor networks. In *ICW '05: Proceedings of the 2005 Systems Communications*, pages 428–433. IEEE Computer Society, 2005.
16. M. Healy, T. Newe, and E. Lewis. Efficiently securing data on a wireless sensor network. *Journal of Physics*, 76, 2007.
17. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.
18. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.
19. C. Karlof, N. Sastry, and D. Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *2nd International Conference on Embedded Networked Sensor Systems – SenSys'2004*, pages 162–175, Baltimore, USA, 2004. ACM.
20. P. Kocher, J. Jaffe, and B. Jun. Introduction to differential power analysis and related attacks. Technical report, Cryptography Research Inc., 1998.
21. Y. W. Law, J. Doumen, and P. Hartel. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 2(1):65–93, 2006.

22. P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 85–95. ACM, 2002.
23. T. Li, H. Wu, X. Wang, and F. Bao. SenSec design. Technical report, InfoComm Security Department, February 2005.
24. D. Liu, P. Ning, and R. Li. Establishing pairwise keys in distributed sensor networks. In *CCS'03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 52–61. ACM, 2003.
25. M. Luk, G. Mezzour, A. Perrig, and V. Gligor. Minisec: A secure sensor network communication architecture. In *IPSN'07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 479–488, New York, NY, USA, 2007. ACM.
26. F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*, volume 16. North-Holland Mathematical Library, 1977.
27. M. Matsui. New block encryption algorithm MISTY. In *Fast Software Encryption – FSE'97*, volume 1267 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 1997.
28. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, USA, 1999.
29. Microchip. *PIC18F8490 Datasheet*, 2006.
30. R. Müller, G. Alonso, and D. Kossmann. SwissQM: Next generation data processing in sensor networks. In *CIDR*, pages 1–9, 2007.
31. J. Nakahara. Analysis of Curupira block cipher. In *Anais do 8º Simpsio Brasileiro em Segurana da Informao e Sistemas Computacionais*, 2008.
32. NIST. *Federal Information Processing Standard (FIPS 197) – Advanced Encryption Standard (AES)*. National Institute of Standards and Technology, November 2001. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.
33. NSA. *Skipjack and KEA Algorithm Specifications, version 2.0*. National Security Agency, May 1998.
34. R. L. Rivest. The RC5 encryption algorithm. In *Fast Software Encryption – FSE'94*, volume 1008 of *Lecture Notes in Computer Science*, pages 86–96. Springer, 1995.
35. R. Roman, C. Alcaraz, and J. Lopez. A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. *Mobile Networks and Applications*, 12(4):231–244, 2007.
36. B. Titzer, D. Lee, and J. Palsberg. Avrora scalable simulation of sensor networks with precise timing. Center for Embedded Network Sensing Posters - Paper 93, 2004.

The name

According to a Brazilian legend, the Curupira is a spirit of nature and protector of the forests. It assumes the form of a boy with red hair, whose feet are turned backwards. This way, when hunters think they are on the right trail to get it, they in fact are going to the wrong direction, getting confused and lost. This should also work against cryptanalysts :-).