# The WHIRLPOOL Hashing Function

Paulo S.L.M. Barreto[1]* and Vincent Rijmen[2]**

[1] Scopus Tecnologia S. A.
Av. Mutinga, 4105 - Pirituba
BR–05110–000 São Paulo (SP), Brazil
pbarreto@scopus.com.br
[2] Cryptomathic NV,
Lei 8A,
B–3000 Leuven, Belgium
vincent.rijmen@cryptomathic.com

**Abstract.** We present WHIRLPOOL, a 512-bit hash function operating on messages less than $2^{256}$ bits in length. The function structure is designed according to the Wide Trail strategy and permits a wide variety of implementation tradeoffs.

*(Revised on May 24, 2003)*

## 1 Introduction

In this document we describe WHIRLPOOL, a one-way, collision resistant 512-bit hashing function operating on messages less than $2^{256}$ bits in length.

WHIRLPOOL consists of the iterated application of a compression function, based on an underlying dedicated 512-bit block cipher that uses a 512-bit key. The round function and the key schedule are designed according to the Wide Trail strategy [2]. WHIRLPOOL implementations on 8-bit and 64-bit processors benefit especially from the function structure, which nevertheless is not oriented toward any particular platform.

As originally submitted for the NESSIE project [17], WHIRLPOOL employed a randomly generated substitution box (S-box) whose lack of internal structure tended to make efficient hardware implementation a challenging and tricky process. The present document describes an S-box that is much more amenable to hardware implementation, while not adversely affecting any of the software implementation techniques suggested herein. No effective algebraic attack based on the recursive structure of the new S-box has been reported.

* Co-sponsored by the Laboratório de Arquitetura e Redes de Computadores (LARC), Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica da Universidade de São Paulo, Brazil.

** Co-sponsored by the Institute for Applied Information Processing and Communications (IAIK), Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria.

Recently, Shirai and Shibutani [22] discovered a flaw in the WHIRLPOOL diffusion matrix that made its branch number suboptimal. Although this flaw *per se* does not seem to introduce an effective vulnerability, the present document replaces that matrix by one that, besides displaying optimal branch number and thus keeping our security analysis unchanged, also leads to more efficient implementation in 8-bit platforms and hardware.

This document is organised as follows. The mathematical preliminaries and notation employed are described in section 2. A mathematical description of the WHIRLPOOL primitive is given in section 3. A statement of the claimed security properties and expected security level is made in section 4. An analysis of the primitive with respect to standard cryptanalytic attacks is provided in section 5 (a statement that there are no hidden weaknesses inserted by the designers is explicitly made in section 5.5). Section 6 contains the design rationale explaining design choices. Implementation guidelines to avoid implementation weaknesses are given in section 7. Estimates of the computational efficiency in software are provided in section 8. The overall strengths and advantages of the primitive are listed in section 9.

## 2 Mathematical preliminaries and notation

We now summarise the mathematical background and notation that will be used throughout this paper.

### 2.1 Finite fields

We will represent the field $\mathrm{GF}(2^4)$ as $\mathrm{GF}(2)[x]/p_4(x)$ where $p_4(x) = x^4 + x + 1$, and the field $\mathrm{GF}(2^8)$ as $\mathrm{GF}(2)[x]/p_8(x)$ where $p_8(x) = x^8 + x^4 + x^3 + x^2 + 1$. Polynomials $p_4(x)$ and $p_8(x)$ are the first primitive polynomials of degrees 4 and 8 listed in [14], and were chosen so that $g(x) = x$ is a generator of $\mathrm{GF}(2^4) \setminus \{0\}$ and $\mathrm{GF}(2^8) \setminus \{0\}$, respectively.

A polynomial $u = \sum_{i=0}^{m-1} u_i \cdot x^i \in \mathrm{GF}(2)[x]$, where $u_i \in \mathrm{GF}(2)$ for all $i = 0, \ldots, m-1$, will be denoted by the numerical value $\sum_{i=0}^{m-1} u_i \cdot 2^i$, and written in hexadecimal notation. For instance, we write $\mathtt{13}_x$ to denote $p_4(x)$.

### 2.2 Matrix classes

$\mathcal{M}_{m\times n}[\mathrm{GF}(2^8)]$ denotes the set of $m \times n$ matrices over $\mathrm{GF}(2^8)$.

$\mathrm{cir}(a_0, a_1, \ldots, a_{m-1})$ stands for the $m \times m$ circulant matrix whose first row consists of elements $a_0, a_1, \ldots, a_{m-1}$, i.e.

$$\mathrm{cir}(a_0, a_1, \ldots, a_{m-1}) \equiv \begin{bmatrix} a_0 & a_1 \ldots & a_{m-1} \\ a_{m-1} & a_0 \ldots & a_{m-2} \\ \vdots & \vdots \ \ddots & \vdots \\ a_1 & a_2 \ldots & a_0 \end{bmatrix},$$

or simply $\mathrm{cir}(a_0, a_1, \ldots, a_{m-1}) = c \Leftrightarrow c_{ij} = a_{(j-i) \bmod m},\ 0 \leqslant i, j \leqslant m-1$.

## 2.3 MDS codes

The Hamming distance between two vectors $u$ and $v$ from the $n$-dimensional vector space $\mathrm{GF}(2^p)^n$ is the number of coordinates where $u$ and $v$ differ.

The Hamming weight $\mathrm{w_h}(a)$ of an element $a \in \mathrm{GF}(2^p)^n$ is the Hamming distance between $a$ and the null vector of $\mathrm{GF}(2^p)^n$, i.e. the number of non-zero components of $a$.

A *linear* $[n, k, d]$ *code* over $\mathrm{GF}(2^p)$ is a $k$-dimensional subspace of the vector space $(\mathrm{GF}(2^p))^n$, where the Hamming distance between any two distinct subspace vectors is at least $d$ (and $d$ is the largest number with this property).

A *generator matrix* $G$ for a linear $[n, k, d]$ code $\mathcal{C}$ is a $k \times n$ matrix whose rows form a basis for $\mathcal{C}$. A generator matrix is in *echelon* or *standard* form if it has the form $G = [I_{k\times k}\, A_{k\times(n-k)}]$, where $I_{k\times k}$ is the identity matrix of order $k$. We write simply $G = [I\, A]$ omitting the indices wherever the matrix dimensions are irrelevant for the discussion, or clear from the context.

Linear $[n, k, d]$ codes obey the *Singleton bound*: $d \leqslant n - k + 1$. A code that meets the bound, i.e. $d = n-k+1$, is called a *maximal distance separable* (MDS) code. A linear $[n, k, d]$ code $\mathcal{C}$ with generator matrix $G = [I_{k\times k}\, A_{k\times(n-k)}]$ is MDS if, and only if, every square submatrix formed from rows and columns of $A$ is non-singular (cf. [15, chapter 11, § 4, theorem 8]).

## 2.4 Cryptographic properties

A product of $m$ distinct Boolean variables is called an $m$-th order product of the variables. Every Boolean function $f : \mathrm{GF}(2)^n \to \mathrm{GF}(2)$ can be written as a sum over $\mathrm{GF}(2)$ of distinct $m$-order products of its arguments, $0 \leqslant m \leqslant n$; this is called the algebraic normal form of $f$.

The *non-linear order* of $f$, denoted $\nu(f)$, is the maximum order of the terms appearing in its algebraic normal form. A *linear* Boolean function is a Boolean function of non-linear order 1, i.e. its algebraic normal form only involves isolated arguments. Given $\alpha \in \mathrm{GF}(2)^n$, we denote by $l_\alpha : \mathrm{GF}(2)^n \to GF(2)$ the linear Boolean function consisting of the sum of the argument bits selected by the bits of $\alpha$:

$$l_\alpha(x) \equiv \bigoplus_{i=0}^{n-1} \alpha_i \cdot x_i.$$

A mapping $S : \mathrm{GF}(2^n) \to \mathrm{GF}(2^n), x \mapsto S[x]$, is called a *substitution box*, or S-box for short. An S-box can also be viewed as a mapping $S : \mathrm{GF}(2)^n \to \mathrm{GF}(2)^n$ and therefore described in terms of its component Boolean functions $s_i : \mathrm{GF}(2)^n \to \mathrm{GF}(2), 0 \leqslant i \leqslant n-1$, i.e. $S[x] = (s_0(x), \ldots, s_{n-1}(x))$.

The *non-linear order* of an S-box $S$, denoted $\nu_S$, is the minimum non-linear order over all linear combinations of the components of $S$:

$$\nu_S \equiv \min_{\alpha\in\mathrm{GF}(2)^n} \{\nu(l_\alpha \circ S)\}.$$

The *δ-parameter* of an S-box $S$ is defined as

$$\delta_S \equiv 2^{-n} \cdot \max_{a\neq 0,\, b} \#\{c \in \mathrm{GF}(2^n) | S[c \oplus a] \oplus S[c] = b\}.$$

The value $2^n \cdot \delta$ is called the *differential uniformity* of $S$.

The *correlation* $c(f, g)$ between two Boolean functions $f$ and $g$ is defined as:

$$c(f, g) \equiv 2^{1-n} \cdot \#\{x | f(x) = g(x)\} - 1.$$

The extreme value (i.e. either the minimum or the maximum, whichever is larger in absolute value) of the correlation between linear functions of input bits and linear functions of output bits of $S$ is called the *bias* of $S$.

The *λ-parameter* of an S-box $S$ is defined as the absolute value of the bias:

$$\lambda_S \equiv \max_{(i,j)\neq(0,0)} |c(l_i, l_j \circ S)|.$$

The *branch number* $\mathcal{B}$ of a linear mapping $\theta : \mathrm{GF}(2^p)^k \to \mathrm{GF}(2^p)^m$ is defined as

$$\mathcal{B}(\theta) \equiv \min_{a\neq 0}\{\mathrm{w_h}(a) + \mathrm{w_h}(\theta(a))\}.$$

Given a $[k + m, k, d]$ linear code over $\mathrm{GF}(2^p)$ with generator matrix $G = [I_{k\times k}\, M_{k\times m}]$, the linear mapping $\theta : \mathrm{GF}(2^p)^k \to \mathrm{GF}(2^p)^m$ defined by $\theta(a) = a{\cdot}M$ has branch number $\mathcal{B}(\theta) = d$; if the code is MDS, such a mapping is called an *optimal diffusion mapping* [20].

### 2.5 Miscellaneous notation

Given a sequence of functions $f_m, f_{m+1}, \ldots, f_{n-1}, f_n$, $m \leqslant n$, we use the notation $\bigcirc_{r=m}^{n} f_r \equiv f_m \circ f_{m+1} \circ \cdots \circ f_{n-1} \circ f_n$, and $\bigcirc_{m}^{r=n} f_r \equiv f_n \circ f_{n-1} \circ \cdots \circ f_{m+1} \circ f_m$; if $m > n$, both expressions stand for the identity mapping.

## 3 Description of the Whirlpool primitive

The Whirlpool primitive is a Merkle hashing function (cf. [16, algorithm 9.25]) based on a dedicated block cipher, $W$, which operates on a 512-bit *hash state* using a chained *key state*, both derived from the input data. In the following we will individually define the component mappings and constants that build up Whirlpool, then specify the complete hashing function in terms of these components.

### 3.1 Input and output

The hash state is internally viewed as a matrix in $\mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$. Therefore, 512-bit data blocks (externally represented as byte arrays by sequentially grouping the bits in 8-bit chunks) must be mapped to and from this matrix format. This is done by function $\mu : \mathrm{GF}(2^8)^{64} \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ and its inverse:

$$\mu(a) = b \Leftrightarrow b_{ij} = a_{8i+j},\ 0 \leqslant i, j \leqslant 7.$$

## 3.2 The non-linear layer $\gamma$

Function $\gamma : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ consists of the parallel application of a non-linear substitution box $S : \mathrm{GF}(2^8) \to \mathrm{GF}(2^8)$, $x \mapsto S[x]$ to all bytes of the argument individually:

$$\gamma(a) = b \Leftrightarrow b_{ij} = S[a_{ij}],\ 0 \leqslant i, j \leqslant 7.$$

The substitution box is discussed in detail in section 6.2.

## 3.3 The cyclical permutation $\pi$

The permutation $\pi : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ cyclically shifts each column of its argument independently, so that column $j$ is shifted downwards by $j$ positions:

$$\pi(a) = b \Leftrightarrow b_{ij} = a_{(i-j) \bmod 8, j},\ 0 \leqslant i, j \leqslant 7.$$

The purpose of $\pi$ is to disperse the bytes of each row among all rows.

## 3.4 The linear diffusion layer $\theta$

The diffusion layer $\theta : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ is a linear mapping based on the $[16, 8, 9]$ MDS code with generator matrix $G_C = [I\, C]$ where $C = \mathrm{cir}(\mathtt{01}_x, \mathtt{01}_x, \mathtt{04}_x, \mathtt{01}_x, \mathtt{08}_x, \mathtt{05}_x, \mathtt{02}_x, \mathtt{09}_x)$,i.e.

$$C = \begin{bmatrix}
\mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x \\
\mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x \\
\mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x \\
\mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x \\
\mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x \\
\mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x & \mathtt{04}_x \\
\mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x & \mathtt{01}_x \\
\mathtt{01}_x & \mathtt{04}_x & \mathtt{01}_x & \mathtt{08}_x & \mathtt{05}_x & \mathtt{02}_x & \mathtt{09}_x & \mathtt{01}_x
\end{bmatrix},$$

so that $\theta(a) = b \Leftrightarrow b = a \cdot C$. The effect of $\theta$ is to mix the bytes in each state row.

## 3.5 The key addition $\sigma[k]$

The affine key addition $\sigma[k] : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ consists of the bitwise addition (exor) of a key matrix $k \in \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$:

$$\sigma[k](a) = b \Leftrightarrow b_{ij} = a_{ij} \oplus k_{ij},\ 0 \leqslant i, j \leqslant 7.$$

This mapping is also used to introduce round constants in the key schedule.

### 3.6 The round constants $\boldsymbol{c^r}$

The round constant for the $r$-th round, $r > 0$, is a matrix $c^r \in \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$, defined as:

$$\begin{aligned} c_{0j}^r &\equiv S[8(r-1)+j],\ 0 \leqslant j \leqslant 7, \\ c_{ij}^r &\equiv 0, \qquad\qquad\qquad 1 \leqslant i \leqslant 7, 0 \leqslant j \leqslant 7. \end{aligned}$$

### 3.7 The round function $\boldsymbol{\rho[k]}$

The $r$-th round function is the composite mapping $\rho[k] : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$, parametrised by the key matrix $k \in \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ and given by:

$$\rho[k] \equiv \sigma[k] \circ \theta \circ \pi \circ \gamma.$$

### 3.8 The key schedule

The key schedule expands the 512-bit cipher key $K \in \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$ onto a sequence of round keys $K^0, \ldots, K^R$:

$$\begin{aligned} K^0 &= K, \\ K^r &= \rho[c^r](K^{r-1}),\ r > 0, \end{aligned}$$

### 3.9 The internal block cipher $\boldsymbol{W}$

The dedicated 512-bit block cipher $W[K] : \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)] \to \mathcal{M}_{8\times 8}[\mathrm{GF}(2^8)]$, parametrised by the 512-bit cipher key $K$, is defined as

$$W[K] = \left( \bigcirc_{1}^{r=R} \rho[K^r] \right) \circ \sigma[K^0],$$

where the round keys $K^0, \ldots, K^R$ are derived from $K$ by the key schedule. The default number of rounds is $R = 10$.

### 3.10 Padding and MD-strengthening

Before being subjected to the hashing operation, a message $M$ of bit length $L < 2^{256}$ is padded with a 1-bit, then with as few 0-bits as necessary to obtain a bit string whose length is an odd multiple of 256, and finally with the 256-bit right-justified binary representation of $L$, resulting in the padded message $m$, partitioned in $t$ blocks $m_1, \ldots, m_t$. These blocks are viewed as byte arrays by sequentially grouping the bits in 8-bit chunks.

### 3.11 The compression function

Whirlpool iterates the Miyaguchi-Preneel hashing scheme [16, algorithm 9.43] over the $t$ padded message blocks $m_i$, $1 \leqslant i \leqslant t$, using the dedicated 512-bit block cipher $W$:

$$\begin{aligned} \eta_i &= \mu(m_i), \\ H_0 &= \mu(IV), \\ H_i &= W[H_{i-1}](\eta_i) \oplus H_{i-1} \oplus \eta_i, \; 1 \leqslant i \leqslant t, \end{aligned}$$

where $IV$ (the *initialisation vector*) is a string of 512 0-bits.

### 3.12 Message digest computation

The Whirlpool message digest for message message $M$ is defined as the output $H_t$ of the compression function, mapped back to a bit string:

$$\text{Whirlpool}(M) \equiv \mu^{-1}(H_t).$$

## 4 Security goals

In this section, we present the goals we have set for the security of Whirlpool. A cryptanalytic attack will be considered successful by the designers if it demonstrates that a security goal described herein does not hold.

### 4.1 Expected strength

Assume we take as hash result the value of any $n$-bit substring of the full Whirlpool output. Then:

- The expected workload of generating a collision is of the order of $2^{n/2}$ executions of Whirlpool.
- Given an $n$-bit value, the expected workload of finding a message that hashes to that value is of the order of $2^n$ executions of Whirlpool.
- Given a message and its $n$-bit hash result, the expected workload of finding a second message that hashes to the same value is of the order of $2^n$ executions of Whirlpool.

Moreover, it is infeasible to detect systematic correlations between any linear combination of input bits and any linear combination of bits of the hash result. It is also infeasible to predict what bits of the hash result will change value when certain input bits are flipped, i.e. Whirlpool is resistant against differential attacks.

These claims result from the considerable safety margin taken with respect to all known attacks. We do however realise that it is impossible to make non-speculative statements on things unknown.

## 5 Analysis

In contrast to the extension of public research on block cipher cryptanalysis, hashing function constructions based on block ciphers have received surprisingly scarce attention. We summarise here the available research results applicable to Whirlpool components.

### 5.1 Security of the compression function

The Miyaguchi-Preneel scheme is one of the few still unbroken methods to construct a hashing function from an underlying block cipher [18]. Its security properties are discussed in [16, section 9.4.1]; in particular, it is "provably secure" if certain ideal properties hold for the underlying block cipher. Recent research results by Black, Rogaway and Shrimpton [1] further analyses the security properties of the Miyaguchi-Preneel and other schemes from a black-box perspective, quantitatively corroborating the choice made for Whirlpool.

### 5.2 Differential cryptanalysis

The application of differential cryptanalysis techniques to hash functions based on block ciphers has been studied in [19, 21]. Although there are some important differences between differential attacks on block ciphers and differential attacks on hash functions, basically the same techniques and reasonings apply. Both attacks require that a differential characteristic is found, that has a sufficiently large probability.

The branch number of the $\theta$ transform is $\mathcal{B} = 9$. Due to the Square pattern propagation theorem (cf. [20, proposition 7.9]), for any two different input values of $W$, it holds that the number of S-boxes with a different input value in four consecutive rounds is at least $\mathcal{B}^2 = 81$. As a consequence, no differential characteristic over four rounds of $W$ has probability larger than $\delta^{\mathcal{B}^2} = (2^{-5})^{81} = 2^{-405}$. This makes a classical differential attack very unlikely to succeed for the full hash function.

### 5.3 Attacks against the internal block cipher $W$

For completeness, we list the best attacks known against the internal block cipher $W$ with reduced number of rounds. We point out, however, that these attacks are not directly applicable to Whirlpool.

The best key recovery attack known against $W$ reduced to 7 rounds is an extension of the attack by Gilbert and Minier [7]. The attack requirements are $2^{64}$ guesses for one column of the first round key $\times$ $2^{32}$ $c$-sets $\times$ 16 values to be encrypted per entry $\times$ 2 tables $\times$ $2^{144}$ entries/table. This sums up to $2^{245}$ steps.

It is possible to mount an attack against 7 rounds of $W$ using ideas described in [6], but the complexity is extremely high: $2^{512}$ S-box lookups, $2^{128}$ bits of storage and $O(2^{512})$ plaintexts. This is essentially the complexity of finding a preimage or second preimage by brute force (and certainly much larger than the complexity of finding a collision by means of the birthday paradox).

No attack is known against more rounds of $W$ faster than exhaustive search.

### 5.4 Encryption-decryption cascade:

Since WHIRLPOOL does not use the decryption form of the internal cipher $W$, encryption-decryption cascades as described in [16, pp. 39] would imply the existence of semi-weak keys, such that encryption with one key corresponds to decryption with another key. We don't believe that semi-weak keys exist for WHIRLPOOL.

### 5.5 Designers' statement on the absence of hidden weaknesses

In spite of any analysis, doubts might remain regarding the presence of trapdoors deliberately introduced in the algorithm. That is why the NESSIE project asks for the designers' declaration on the contrary.

Therefore we, the designers of WHIRLPOOL, do hereby declare that there are no hidden weaknesses inserted by us in the WHIRLPOOL primitive.

## 6 Design rationale

### 6.1 Hashing mode

Why Miyaguchi-Preneel instead of, say, Matyas-Meyer-Oseas (MMO)? Notice that the key schedule resembles encryption of the cipher key under a pseudo-key defined by the round constants, so that the core of the hashing process could be formally viewed as two interacting encryption lines. Consider the encryption $W[H_{i-1}](\eta_i)$. We could write the last round key as $K^R = W'[c](H_{i-1})$; this quantity is exored onto the cipher state as the last encryption step. Now take a look at the MMO recursion: $H_i = W[H_{i-1}](\eta_i) \oplus \eta_i$. Formally applying this construction to the "key encryption line" we get ${K'}^R = W'[c](H_{i-1}) \oplus H_{i-1}$. Using this value as the effective last round key formally creates two interacting MMO lines (as compared to the interacting encryption lines), and results in the Miyaguchi-Preneel scheme, which therefore shows up as the natural choice for the compression function.

### 6.2 Choice of the substitution box

The originally submitted form of WHIRLPOOL used a pseudo-randomly generated S-box, chosen to satisfy the following conditions:

- The $\delta$-parameter must not exceed $8 \times 2^{-8}$.
- The $\lambda$-parameter must not exceed $16 \times 2^{-6}$.
- The non-linear order $\nu$ must be maximum, namely, 7.

The bounds on $\delta$ and $\lambda$ correspond to twice the minimum achievable values for these quantities. An additional condition, that the S-box has no fixed point, was imposed in an attempt to speed up the search. This condition was inspired by the empirical study reported in [26, section 2.3], where the strong correlation found between the cryptographic properties and the number of fixed points of a substitution box suggests minimising the number of such points. The polynomial and rational representations of $S$ over $\mathrm{GF}(2^8)$ are checked as well, to avoid any obvious algebraic weakness (which could lead e.g. to interpolation attacks [10]).

However, the extreme lack of structure in such an S-box hinders efficient hardware implementation. Moreover, a flaw that went unnoticed in the random search program caused the value of $\lambda$ for the original S-box to be incorrectly reported as $15 \times 2^{-6}$ instead of the actual value $16 \times 2^{-6}$ (corresponding to a negative bias)[1]. Therefore, we now describe an alternative S-box that, besides strictly satisfying the design conditions, is amenable to much more efficient implementation in hardware, while not affecting the software implementation techniques presented here in any reasonable way.

The new S-box is illustrated in figure 1. This structure has its origin in a simple three-layer construction consisting of two non-linear layers (each containing two $4 \times 4$ S-boxes) separated by a symmetric linear transform $M : \mathrm{GF}(2^4)^2 \to \mathrm{GF}(2^4)^2$. The most general form such a transform can assume is given by the matrix

$$M = \begin{bmatrix} a+1 & a \\ a & a+1 \end{bmatrix}, \; a \in \mathrm{GF}(2^4),$$

which reduces to the structure in figure 1 by setting $R(u) \equiv a \cdot u$ (the actual $R$ is pseudo-randomly generated as described below). Thus, writing $S$ as a mapping $S : \mathrm{GF}(2^4)^2 \to \mathrm{GF}(2^4)^2$, $S(u, v) = (u', v')$, we have

$$u' = E(E(u) \oplus r), \; v' = E^{-1}(E^{-1}(v) \oplus r),$$

where $r \equiv R(E(u) \oplus E^{-1}(v))$.

The $E$ table (as well as its inverse $E^{-1}$) is not generated at random; rather, it is derived from a simple exponential mapping with optimal $\delta$, $\lambda$, and $\nu$, namely:

$$E : \mathrm{GF}(2^4) \to \mathrm{GF}(2^4) : E(u) = \begin{cases} \mathtt{B}_x^u \text{ if } u \neq \mathtt{F}_x, \\ \mathtt{0}_x \text{ otherwise,} \end{cases}$$

where the occurrence of $u = u_3x^3 + u_2x^2 + u_1x + u_0$ as an exponent in $\mathtt{B}_x^u$ is taken to be its numerical value $\sum_{i=0}^{3} u_i \cdot 2^i$. The basis $\mathtt{B}_x$ was chosen so that $E$ has neither fixed points (i.e. values $u$ such that $E(u) = u$) nor points $u$ such that $E(E(u)) = u$. Notice that $E^{-1}$ satisfies the same properties.

The $R$ table is a pseudo-randomly generated permutation with optimal $\delta$, $\lambda$, and $\nu$, chosen so that the S-box built from $E$, $E^{-1}$ and $R$ satisfies the design criteria listed at the beginning of this section.

Table 1 lists the $E$ permutation, and table 2 shows the $R$ permutation found by the searching algorithm.

[1] We thank the NESSIE evaluation team for pointing out this discrepancy [5].

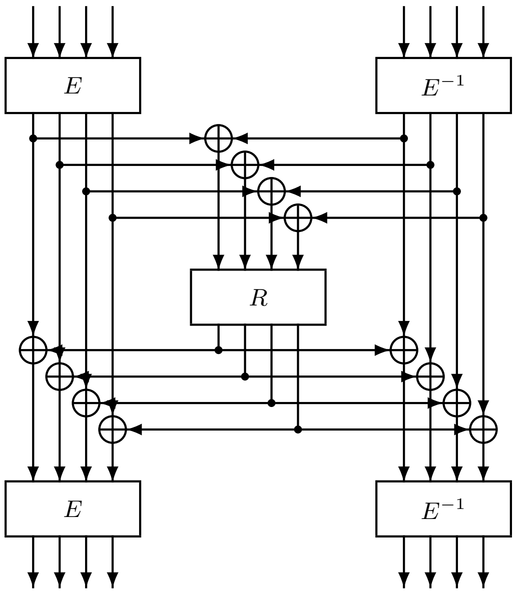

**Fig. 1.** Structure of the WHIRLPOOL S-box. $E$ corresponds to the mapping $E : \mathrm{GF}(2^4) \to \mathrm{GF}(2^4) : E(u) = \mathtt{B}_x^u$ if $u \neq \mathtt{F}_x$, and $E(\mathtt{F}_x) = 0$. $R$ is pseudo-randomly generated in a verifiable way. Both have optimal values of $\delta$, $\lambda$, and $\nu$.

The random search we carried out was able to find an S-box with $\lambda = 14 \times 2^{-6}$, slightly better than the design bound. A description of the searching algorithm and a listing of the resulting S-box are given in the appendix.

**Table 1.** The $E$ mini-box

| $u$ | $0_x$ | $1_x$ | $2_x$ | $3_x$ | $4_x$ | $5_x$ | $6_x$ | $7_x$ | $8_x$ | $9_x$ | $A_x$ | $B_x$ | $C_x$ | $D_x$ | $E_x$ | $F_x$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $E[u]$ | $1_x$ | $B_x$ | $9_x$ | $C_x$ | $D_x$ | $6_x$ | $F_x$ | $3_x$ | $E_x$ | $8_x$ | $7_x$ | $4_x$ | $A_x$ | $2_x$ | $5_x$ | $0_x$ |

**Table 2.** The $R$ mini-box

| $u$ | $0_x$ | $1_x$ | $2_x$ | $3_x$ | $4_x$ | $5_x$ | $6_x$ | $7_x$ | $8_x$ | $9_x$ | $A_x$ | $B_x$ | $C_x$ | $D_x$ | $E_x$ | $F_x$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $R[u]$ | $7_x$ | $C_x$ | $B_x$ | $D_x$ | $E_x$ | $4_x$ | $9_x$ | $F_x$ | $6_x$ | $3_x$ | $8_x$ | $A_x$ | $2_x$ | $5_x$ | $1_x$ | $0_x$ |

### 6.3 Choice of the diffusion layer

The adopted circulant matrix $C$ has as many 1-elements as possible (namely, 3 per row) for an $8 \times 8$ circulant MDS matrix; furthermore, any element has

Hamming weight at most 2, and polynomial degree at most 3. These constraints are especially advantageous for smart cards and dedicated hardware, and from all matrices satisfying these criteria the actual selection leads to a particularly efficient implementation on those platforms (see section 7.3).

## 6.4 The last round

One difference between the WHIRLPOOL structure and the structure of SQUARE [3] and RIJNDAEL [4] is the fact that, in the former, the operation $\theta$ is present in all rounds, while in the latter it is not present in the first or in the last round. Firstly, we will explain why one application of the operation $\theta$ can be left out without changing the security level of the cipher. Subsequently, we list some motivations to leave out one application of the operation $\theta$, followed by the motivation why it was actually kept in for WHIRLPOOL.

**Why it is possible to leave $\theta$ out:** Consider a SQUARE-like cipher with two rounds, and an extra key addition:

$$E = \sigma[K^2] \circ \tau \circ \gamma \circ \theta \circ \sigma[K^1] \circ \tau \circ \gamma \circ \theta \circ \sigma[K^0]. \tag{1}$$

As explained in [3], the operations $\theta$ and $\sigma[K]$ can be exchanged, provided that the key $K$ is replaced by an equivalent key $K' = \theta(K)$. Consequently, we can write (1) as:

$$E = \sigma[K^2] \circ \tau \circ \gamma \circ \theta \circ \sigma[K^1] \circ \tau \circ \gamma \circ \sigma[\theta(K^0)] \circ \theta. \tag{2}$$

In (2), it is obvious that the first application of $\theta$ does not contribute to the security of the cipher, because it can always be undone by an attacker, without knowing the key. Therefore, we can leave it out of the definition of our cipher (1). The new definition becomes:

$$E' = \sigma[K^2] \circ \tau \circ \gamma \circ \theta \circ \sigma[K^1] \circ \tau \circ \gamma \circ \sigma[K^0]. \tag{3}$$

Observe that in this analysis we did not make any assumption about the attack that an attacker is trying to mount. We proved generally that the security of $E$ and $E'$ are equivalent.

**Motivation to leave $\theta$ out:** One motivation to leave out one application of $\theta$, is that it does not contribute to the cipher's security. Furthermore, implementations on small processors that execute all transformations explicitly will probably experience increased performance. Thirdly, (3) has the advantage that encryption and decryption are more similar to one another[2] than for (1).

[2] However, in WHIRLPOOL the internal cipher $W$ operates only in encryption mode, hence the third motivation to keep $\theta$ is not important here.

**Motivation to leave $\theta$ in:** The best motivation to keep all rounds identical to one another, is the performance on processors with a medium-sized fast cache memory. If not all rounds are identical, then the number of tables that have to be stored in memory increases. For fast implementations of SQUARE, it turns out that the tables for the complete rounds can be stored in the cache, but there is no place left for the tables of the incomplete round. The net result is that the round without $\theta$ takes longer to execute.

## 7 Implementation

WHIRLPOOL can be implemented very efficiently. On different platforms, different optimisations and tradeoffs are possible. We make here a few suggestions.

### 7.1 64-bit processors

We suggest a lookup-table approach to implement $\rho$. Let $C_k$ be the $k$-th row of the circulant matrix $C$; using eight tables $T_k[x] \equiv S[x] \cdot C_k,\ 0 \leqslant k \leqslant 7$, i.e.:

$$\begin{aligned}
T_0[x] &= S[x] \cdot [\,\texttt{01}_x\ \texttt{01}_x\ \texttt{04}_x\ \texttt{01}_x\ \texttt{08}_x\ \texttt{05}_x\ \texttt{02}_x\ \texttt{09}_x\,],\\
T_1[x] &= S[x] \cdot [\,\texttt{09}_x\ \texttt{01}_x\ \texttt{01}_x\ \texttt{04}_x\ \texttt{01}_x\ \texttt{08}_x\ \texttt{05}_x\ \texttt{02}_x\,],\\
T_2[x] &= S[x] \cdot [\,\texttt{02}_x\ \texttt{09}_x\ \texttt{01}_x\ \texttt{01}_x\ \texttt{04}_x\ \texttt{01}_x\ \texttt{08}_x\ \texttt{05}_x\,],\\
T_3[x] &= S[x] \cdot [\,\texttt{05}_x\ \texttt{02}_x\ \texttt{09}_x\ \texttt{01}_x\ \texttt{01}_x\ \texttt{04}_x\ \texttt{01}_x\ \texttt{08}_x\,],\\
T_4[x] &= S[x] \cdot [\,\texttt{08}_x\ \texttt{05}_x\ \texttt{02}_x\ \texttt{09}_x\ \texttt{01}_x\ \texttt{01}_x\ \texttt{04}_x\ \texttt{01}_x\,],\\
T_5[x] &= S[x] \cdot [\,\texttt{01}_x\ \texttt{08}_x\ \texttt{05}_x\ \texttt{02}_x\ \texttt{09}_x\ \texttt{01}_x\ \texttt{01}_x\ \texttt{04}_x\,],\\
T_6[x] &= S[x] \cdot [\,\texttt{04}_x\ \texttt{01}_x\ \texttt{08}_x\ \texttt{05}_x\ \texttt{02}_x\ \texttt{09}_x\ \texttt{01}_x\ \texttt{01}_x\,],\\
T_7[x] &= S[x] \cdot [\,\texttt{01}_x\ \texttt{04}_x\ \texttt{01}_x\ \texttt{08}_x\ \texttt{05}_x\ \texttt{02}_x\ \texttt{09}_x\ \texttt{01}_x\,],
\end{aligned}$$

then a row $b_i$ of $b = (\theta \circ \pi \circ \gamma)(a)$ can be calculated with eight table lookups and seven exor operations as:

$$b_i = \bigoplus_{k=0}^{7} T_k[a_{(i-k) \bmod 8, k}];$$

the key addition then completes the evaluation of $\rho$ with a single additional exor. The $T$-tables require $2^8 \times 8$ bytes of storage each. An implementation can use the fact that the corresponding entries of different $T$-tables are cyclical permutations of one another and save some memory at the expense of introducing extra permutations at runtime. Usually this decreases the performance of the implementation.

### 7.2 32-bit processors

Any circulant matrix $C$ of order $2m$ shows the following structure:

$$C = \begin{bmatrix} U & V \\ V & U \end{bmatrix},$$

where $U$ and $V$ are matrices of order $m$. A 32-bit implementation may take advantage of this structure by representing elements $c \in \mathrm{GF}(2^8)^8$ as pairs $c = \begin{bmatrix} \hat{c}_0 & \hat{c}_1 \end{bmatrix}$ of elements $\hat{c}_i \in \mathrm{GF}(2^8)^4$:

$$b = \theta(a) \Leftrightarrow \begin{cases} \hat{b}_0 = \hat{a}_0 U \oplus \hat{a}_1 V, \\ \hat{b}_1 = \hat{a}_0 V \oplus \hat{a}_1 U, \end{cases}$$

with twice the complexity derived for 64-bit processors regarding the number of table lookups and exors, but using smaller tables (each occupying $2^8 \times 4$ bytes).

## 7.3 8-bit processors

On an 8-bit processor with a limited amount of RAM, e.g. a typical smart card processor, the previous approach is not feasible. On these processors the substitution is performed byte by byte, combined with the $\sigma[k]$ transformation. For $\theta$, it is necessary to implement the matrix multiplication. The following piece of pseudo-code calculates one row of $b = \theta(a)$, using a table $X$ that implements multiplication by the polynomial $g(x) = x$ in $\mathrm{GF}(2^8)$ (i.e. $X[u] \equiv x \cdot u$) and five auxiliary variables $t_0$ to $t_4$, at the cost of 46 exors and 24 table lookups:

```
$t_0 \leftarrow a_{i1} \oplus a_{i3} \oplus a_{i5} \oplus a_{i7}$;
$t_1 \leftarrow a_{i3} \oplus a_{i6}$;
$t_2 \leftarrow a_{i5} \oplus a_{i0}$;
$t_3 \leftarrow a_{i7} \oplus a_{i2}$;
$t_4 \leftarrow a_{i1} \oplus a_{i4}$;
$b_{i0} \leftarrow a_{i0} \oplus t_0 \oplus X[a_{i2} \oplus X[t_1 \oplus X[t_4]]]$;
$b_{i2} \leftarrow a_{i2} \oplus t_0 \oplus X[a_{i4} \oplus X[t_2 \oplus X[t_1]]]$;
$b_{i4} \leftarrow a_{i4} \oplus t_0 \oplus X[a_{i6} \oplus X[t_3 \oplus X[t_2]]]$;
$b_{i6} \leftarrow a_{i6} \oplus t_0 \oplus X[a_{i0} \oplus X[t_4 \oplus X[t_3]]]$;
$t_0 \leftarrow a_{i0} \oplus a_{i2} \oplus a_{i4} \oplus a_{i6}$;
$t_1 \leftarrow a_{i4} \oplus a_{i7}$;
$t_2 \leftarrow a_{i6} \oplus a_{i1}$;
$t_3 \leftarrow a_{i0} \oplus a_{i3}$;
$t_4 \leftarrow a_{i2} \oplus a_{i5}$;
$b_{i1} \leftarrow a_{i1} \oplus t_0 \oplus X[a_{i3} \oplus X[t_1 \oplus X[t_4]]]$;
$b_{i3} \leftarrow a_{i3} \oplus t_0 \oplus X[a_{i5} \oplus X[t_2 \oplus X[t_1]]]$;
$b_{i5} \leftarrow a_{i5} \oplus t_0 \oplus X[a_{i7} \oplus X[t_3 \oplus X[t_2]]]$;
$b_{i7} \leftarrow a_{i7} \oplus t_0 \oplus X[a_{i1} \oplus X[t_4 \oplus X[t_3]]]$;
```

## 7.4 Techniques to avoid software implementation weaknesses

Hash functions do not use secret keys. In principle, they are not vulnerable to the key recovery techniques described by Kocher *et al.* [12, 13]. However, hash functions are sometimes used as building blocks for other cryptographic primitives, such as MACs, that use secret keys. In that case, the necessary attention should be given to the implementation of the round transformation as well as the key scheduling of the primitive.

A first example is the *timing attack* [12] that can be applicable if the execution time of the primitive depends on the value of the key and the plaintext. This is typically caused by the presence of conditional execution paths. For instance, multiplication by a constant value over a finite field is sometimes implemented as a multiplication followed by a reduction, the latter being implemented as a conditional exor. This vulnerability is avoided by implementing the multiplication by a constant by means of table lookups, as proposed in sections 7.1, 7.2, and 7.3.

A second class of attacks are the attacks based on the careful observation of the power consumption pattern of an encryption device [13]. Protection against this type of attack can only be achieved by combined measures at the hardware and software level. We leave the final word on this issue to the specialists, but we hope that the simple structure and the limited number of operations in WHIRLPOOL will make it easier to create an implementation that resists this type of attacks.

### 7.5 Hardware implementation

We have currently no precise figures on the available performance and required area or gate count of WHIRLPOOL in ASIC or FPGA, nor do we have a description in VHDL. However, we expect that the results on RIJNDAEL [9, 23] will carry over to some extent[3].

## 8 Efficiency estimates

Using the reference C implementation on a 1 GHz Pentium III platform, we observe that WHIRLPOOL operates at about 73 cycles per hashed byte. The compression function runs at about 56 cycles per hashed byte.

Many factors explain the observed performance. First, a 32-bit processor was used to test a native 64-bit implementation; better results are expected by merely running the speed measurement on an Alpha or Itanium processor. Second, it seems that the pipe parallelism capabilities of the Pentium were not fully used; this may reflect a non-optimising implementation of 64-bit arithmetic support by the C compiler, and might be overcome by an assembler implementation. Third, the tables employed in the reference implementation are quite large, and the built-in processor cache might not be enough to hold them, the data being hashed, and the hashing code at once, thus degrading processing speed.

[3] In particular, the S-box structure can be implemented in about 1/5 the number of gates used by the implementation of the RIJNDAEL S-box reported in [24], which takes about 500–600 gates [25].

## 9 Advantages

WHIRLPOOL is much more scalable than most modern hashing functions. Even though is not specifically oriented toward any platform, it is rather efficient on many of them, its structure favouring extensively parallel execution of the component mappings. At the same time, it does not require excessive storage space (either for code or for tables), and can therefore be efficiently implemented in quite constrained environments like smart cards, although it can benefit from larger cache memory available on modern processors to achieve higher performance. It does not use expensive or unusual instructions that must be built in the processor. The mathematical simplicity of the primitive resulting from the design strategy tends to make analysis easier. And finally, it has a very long hash length; this not only provides increased protection against birthday attacks, but also offers a larger internal state for entropy containment, as is needed for certain classes of pseudo-random number generators [11].

## 10 Acknowledgements

We are grateful to Raïf Naffah, for carefully reading and suggesting improvements for the implementation guidelines provided in this paper, and for implementing several versions of WHIRLPOOL in Java.

We are deeply indebted to Brian Gladman for providing software and hardware facilities to search for efficient mini-box implementations in terms of Boolean functions.

We thank Paris Kitsos for pointing out an error in the diagram displayed in figure 1.

Finally, we would also like to thank the NESSIE project organisers and evaluation team for making this work possible.

## References

1. J. Black, P. Rogaway, and T. Shrimpton, *Black-box analysis of the block-cipher-based hash-function constructions from PGV*, Advances in Cryptology – Crypto'2002, Lecture Notes in Computer Science, vol. 2442, Springer-Verlag, 2002, pp. 320–335.
2. J. Daemen, *Cipher and hash function design strategies based on linear and differential cryptanalysis*, Ph.D. thesis, Katholieke Universiteit Leuven, March 1995.
3. J. Daemen, L.R. Knudsen, and V. Rijmen, *The block cipher* SQUARE, Fast Software Encryption – FSE'97, Lecture Notes in Computer Science, vol. 1267, Springer-Verlag, 1997, pp. 149–165.
4. J. Daemen and V. Rijmen, *The design of* RIJNDAEL, Springer-Verlag, Berlin, 2002, Also described in NIST FIPS 197.
5. NESSIE evaluation team, *private communication*, 2001.
6. N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, *Improved cryptanalysis of* RIJNDAEL, Fast Software Encryption – FSE'2000, Lecture Notes in Computer Science, vol. 1978, Springer-Verlag, 2000, pp. 213–230.

7. H. Gilbert and M. Minier, *A collision attack on 7 rounds of* Rijndael, Third Advanced Encryption Standard Candidate Conference, NIST, 2000, pp. 230–241.
8. M. Hoskin, *The cambridge illustrated history of astronomy*, Cambridge University Press, London, 1997.
9. T. Ichikawa, T. Kasuya, and M. Matsui, *Hardware evaluation of the AES finalists*, Third Advanced Encryption Standard Candidate Conference, NIST, 2000, pp. 279–285.
10. T. Jakobsen and L.R. Knudsen, *The interpolation attack on block ciphers*, Fast Software Encryption – FSE'97, Lecture Notes in Computer Science, vol. 1267, Springer-Verlag, 1997, pp. 28–40.
11. J. Kelsey, B. Schneier, D. Wagner, and C. Hall, *Cryptanalytic attacks on pseudorandom number generators*, Fast Software Encryption – FSE'98, Lecture Notes in Computer Science, vol. 1372, Springer-Verlag, 1998, pp. 168–188.
12. P. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology – Crypto'96, Lecture Notes in Computer Science, vol. 1109, Springer-Verlag, 1996, pp. 104–113.
13. P. Kocher, J. Jaffe, and B. Jun, *Differential power analysis*, Advances in Cryptology – Crypto'99, Lecture Notes in Computer Science, vol. 1666, Springer-Verlag, 1999, pp. 388–397.
14. R. Lidl and H. Niederreiter, *Introduction to finite fields and their applications*, Cambridge University Press, London, 1986.
15. F.J. MacWilliams and N.J.A. Sloane, *The theory of error-correcting codes*, North-Holland Mathematical Library, vol. 16, North-Holland, London, 1977.
16. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997.
17. NESSIE, *New European Schemes for Signatures, Integrity, and Encryption*, http://cryptonessie.org, 2000.
18. B. Preneel, *Analysis and design of cryptographic hash functions*, Ph.D. thesis, Katholieke Universiteit Leuven, January 1993.
19. ———, *Differential cryptanalysis of hash functions based on block ciphers*, Computer and Communications Security, ACM, 1993, pp. 183–188.
20. V. Rijmen, *Cryptanalysis and design of iterated block ciphers*, Ph.D. thesis, Katholieke Universiteit Leuven, October 1997.
21. V. Rijmen and B. Preneel, *Improved characteristics for differential cryptanalysis of hash functions based on block ciphers*, Fast Software Encryption – FSE'95, Lecture Notes in Computer Science, vol. 1008, Springer-Verlag, 1995, pp. 242–248.
22. T. Shirai and K. Shibutani, *On the diffusion matrix employed in the* Whirlpool *hashing function*, NESSIE public report, 2003.
23. B. Weeks, M. Bean, T. Rozylowicz, and C. Ficke, *Hardware performance simulations of round 2 AES algorithms*, Third Advanced Encryption Standard Candidate Conference, NIST, 2000, pp. 286–304.
24. D. Whiting, *AES implementations in* 0.25*μm ASIC*, NIST AES electronic discussion forum posting, 2000.
25. ———, *private communication*, 2001.
26. A.M. Youssef, S.E. Tavares, and H.M. Heys, *A new class of substitution-permutation networks*, Selected Areas in Cryptography – SAC'96, Workshop record, 1996, pp. 132–147.

## A Generation of the WHIRLPOOL S-box

The only part of the S-box structure left unspecified in figure 1 is the $R$ permutation, which is generated pseudo-randomly in a verifiable way.

The searching algorithm starts with a simple permutation without fixed points (namely, the negation mapping $u \mapsto \bar{u} = u \oplus \texttt{F}_x$), and derives from it a sequence of $4 \times 4$ substitution boxes ("mini-boxes") with the optimal values $\delta = 1/4$, $\lambda = 1/2$, and $\nu = 3$. Each such mini-box is combined with $E$ and $E^{-1}$ according to the diagram shown in figure 1; finally, the resulting $8 \times 8$ S-box, if free of fixed points, is tested for the design criteria regarding $\delta$, $\lambda$, and $\nu$.

Given a mini-box at any point during the search, a new one is derived from it by choosing a pair of distinct values that are not the image of one another and swapping them, keeping the result free of fixed points; this is repeated until the running mini-box has optimal values of $\delta$, $\lambda$, and $\nu$.

The pseudo-random number generator is implemented with RIJNDAEL [4] in counter mode, with a fixed key consisting of 256 zero bits and an initial counter value consisting of 128 zero bits.

The following pseudo-code fragment illustrates the computation of the chain of mini-boxes and the resulting S-box:

```
// initialize $R$ to the negation permutation:
**for** ($u \leftarrow 0$; $u < 256$; $u$++) {
    $R[u] \leftarrow \bar{u}$;
}
// look for S-box conforming to the design criteria:
**do** {
    // generate a random permutation free of fixed points:
    **do** {
        **do** {
            // randomly select $x$ and $y$ such that
            // $x \neq y$, $R[x] \neq y$, and $R[y] \neq x$:
            $z \leftarrow$ RandomByte(); $x \leftarrow z \gg 4$; $y \leftarrow z \,\&\, \texttt{0F}_x$;
        } **while** ($x = y \;\vee\; R[x] = y \;\vee\; R[y] = x$);
        // swap entries:
        $u \leftarrow R[x]; R[x] \leftarrow R[y]; R[y] \leftarrow u$;
    } **while** ($\delta(R) > 1/4 \;\vee\; \lambda(R) > 1/2 \;\vee\; \nu(R) < 3$);
    // build S-box from the mini-boxes (see figure 1):
    **for** ($u \leftarrow 0$; $u < 256$; $u$++) {
        $x \leftarrow E[u \gg 4]$; $y \leftarrow E^{-1}[u \,\&\, \texttt{0F}_x]$;
        $r \leftarrow R[x \oplus y]$; $x \leftarrow x \oplus r$; $y \leftarrow y \oplus r$;
        $S[u] \leftarrow (E[x] \ll 4) \mid E^{-1}[y]$;
    }
    // test design criteria:
} **while** (#FixedPoints($S$) $> 0 \;\vee\; \delta(S) > 2^{-5} \;\vee\; \lambda(S) > 2^{-2} \;\vee\; \nu(S) < 7$);
```

## B Hardware implementation

Restricting the allowed logical gates to AND, OR, NOT, and XOR, the $E$ mini-box and its inverse can be implemented with 18 logical gates each, while the $R$ mini-box needs 17 logical gates. Therefore, the complete S-box can be implemented with 101 gates.

The pseudo-code fragments shown in figure 2 illustrate this ($u = u_3x^3 + u_2x^2 + u_1x + u_0 \in \mathrm{GF}(2^4)$ denotes the mini-box input, $z = z_3x^3 + z_2x^2 + z_1x + z_0 \in \mathrm{GF}(2^4)$ denotes its output, and the $t_k$ denote intermediate values). We point out, however, that the search for efficient Boolean expressions for the mini-boxes has not been thorough, and it is likely that better expressions exist.

| $z = E[u]$ | $z = E^{-1}[u]$ | $z = R[u]$ |
|---|---|---|
| $t_0 \leftarrow u_0 \wedge u_2$ | $t_0 \leftarrow \neg u_0$ | $t_0 \leftarrow \neg u_0$ |
| $t_0 \leftarrow t_0 \oplus u_1$ | $t_1 \leftarrow u_0 \vee u_1$ | $t_1 \leftarrow u_2 \wedge u_3$ |
| $t_2 \leftarrow \neg u_0$ | $t_1 \leftarrow t_1 \oplus u_3$ | $t_2 \leftarrow u_0 \oplus t_1$ |
| $t_1 \leftarrow u_3 \oplus t_2$ | $t_2 \leftarrow u_2 \wedge t_1$ | $t_2 \leftarrow t_2 \vee u_1$ |
| $t_2 \leftarrow t_0 \vee t_1$ | $z_3 \leftarrow t_0 \oplus t_2$ | $t_3 \leftarrow u_3 \vee t_0$ |
| $z_0 \leftarrow u_0 \oplus t_2$ | $t_2 \leftarrow u_0 \wedge u_2$ | $z_2 \leftarrow t_2 \oplus t_3$ |
| $t_2 \leftarrow u_2 \wedge t_0$ | $t_3 \leftarrow u_0 \vee u_3$ | $t_2 \leftarrow \neg u_2$ |
| $t_1 \leftarrow t_1 \oplus t_2$ | $t_3 \leftarrow t_3 \oplus t_2$ | $t_2 \leftarrow t_2 \oplus t_3$ |
| $t_2 \leftarrow u_3 \vee z_0$ | $t_3 \leftarrow t_3 \wedge u_1$ | $t_3 \leftarrow u_1 \vee t_2$ |
| $z_1 \leftarrow t_2 \oplus t_1$ | $z_0 \leftarrow t_0 \oplus t_3$ | $z_3 \leftarrow t_1 \oplus t_3$ |
| $t_2 \leftarrow u_2 \oplus t_1$ | $t_2 \leftarrow t_2 \oplus u_1$ | $t_3 \leftarrow t_3 \oplus t_0$ |
| $t_1 \leftarrow t_1 \oplus u_3$ | $t_3 \leftarrow u_2 \oplus t_0$ | $t_0 \leftarrow u_0 \vee z_3$ |
| $t_0 \leftarrow t_0 \oplus t_2$ | $t_4 \leftarrow z_3 \oplus t_2$ | $t_1 \leftarrow \neg u_1$ |
| $t_1 \leftarrow t_1 \vee t_0$ | $t_1 \leftarrow t_1 \wedge t_4$ | $t_1 \leftarrow t_1 \oplus u_3$ |
| $z_2 \leftarrow t_2 \oplus t_1$ | $t_2 \leftarrow t_2 \vee t_1$ | $z_0 \leftarrow t_0 \oplus t_1$ |
| $t_1 \leftarrow z_1 \wedge z_2$ | $z_2 \leftarrow t_3 \oplus t_1$ | $t_3 \leftarrow t_3 \vee z_0$ |
| $t_1 \leftarrow t_1 \vee z_0$ | $t_0 \leftarrow t_0 \vee u_3$ | $z_1 \leftarrow t_2 \oplus t_3$ |
| $z_3 \leftarrow t_0 \oplus t_1$ | $z_1 \leftarrow t_0 \oplus t_2$ | |

**Fig. 2.** Boolean expressions for $E$, $E^{-1}$, and $R$

For completeness, table 3 lists the resulting $8 \times 8$ Whirlpool S-box.

## C The name

Whirlpool is named after the Whirlpool galaxy in Canes Venatici (M51, or NGC 5194), the first one recognised to have spiral structure by William Parsons, third Earl of Rosse, in April 1845 [8].

**Table 3.** The Whirlpool S-box

| | 00$_x$ | 01$_x$ | 02$_x$ | 03$_x$ | 04$_x$ | 05$_x$ | 06$_x$ | 07$_x$ | 08$_x$ | 09$_x$ | 0A$_x$ | 0B$_x$ | 0c$_x$ | 0d$_x$ | 0E$_x$ | 0F$_x$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00$_x$ | 18$_x$ | 23$_x$ | c6$_x$ | E8$_x$ | 87$_x$ | B8$_x$ | 01$_x$ | 4F$_x$ | 36$_x$ | A6$_x$ | d2$_x$ | F5$_x$ | 79$_x$ | 6F$_x$ | 91$_x$ | 52$_x$ |
| 10$_x$ | 60$_x$ | Bc$_x$ | 9B$_x$ | 8E$_x$ | A3$_x$ | 0c$_x$ | 7B$_x$ | 35$_x$ | 1d$_x$ | E0$_x$ | d7$_x$ | c2$_x$ | 2E$_x$ | 4B$_x$ | FE$_x$ | 57$_x$ |
| 20$_x$ | 15$_x$ | 77$_x$ | 37$_x$ | E5$_x$ | 9F$_x$ | F0$_x$ | 4A$_x$ | dA$_x$ | 58$_x$ | c9$_x$ | 29$_x$ | 0A$_x$ | B1$_x$ | A0$_x$ | 6B$_x$ | 85$_x$ |
| 30$_x$ | Bd$_x$ | 5d$_x$ | 10$_x$ | F4$_x$ | cB$_x$ | 3E$_x$ | 05$_x$ | 67$_x$ | E4$_x$ | 27$_x$ | 41$_x$ | 8B$_x$ | A7$_x$ | 7d$_x$ | 95$_x$ | d8$_x$ |
| 40$_x$ | FB$_x$ | EE$_x$ | 7c$_x$ | 66$_x$ | dd$_x$ | 17$_x$ | 47$_x$ | 9E$_x$ | cA$_x$ | 2d$_x$ | BF$_x$ | 07$_x$ | Ad$_x$ | 5A$_x$ | 83$_x$ | 33$_x$ |
| 50$_x$ | 63$_x$ | 02$_x$ | AA$_x$ | 71$_x$ | c8$_x$ | 19$_x$ | 49$_x$ | d9$_x$ | F2$_x$ | E3$_x$ | 5B$_x$ | 88$_x$ | 9A$_x$ | 26$_x$ | 32$_x$ | B0$_x$ |
| 60$_x$ | E9$_x$ | 0F$_x$ | d5$_x$ | 80$_x$ | BE$_x$ | cd$_x$ | 34$_x$ | 48$_x$ | FF$_x$ | 7A$_x$ | 90$_x$ | 5F$_x$ | 20$_x$ | 68$_x$ | 1A$_x$ | AE$_x$ |
| 70$_x$ | B4$_x$ | 54$_x$ | 93$_x$ | 22$_x$ | 64$_x$ | F1$_x$ | 73$_x$ | 12$_x$ | 40$_x$ | 08$_x$ | c3$_x$ | Ec$_x$ | dB$_x$ | A1$_x$ | 8d$_x$ | 3d$_x$ |
| 80$_x$ | 97$_x$ | 00$_x$ | cF$_x$ | 2B$_x$ | 76$_x$ | 82$_x$ | d6$_x$ | 1B$_x$ | B5$_x$ | AF$_x$ | 6A$_x$ | 50$_x$ | 45$_x$ | F3$_x$ | 30$_x$ | EF$_x$ |
| 90$_x$ | 3F$_x$ | 55$_x$ | A2$_x$ | EA$_x$ | 65$_x$ | BA$_x$ | 2F$_x$ | c0$_x$ | dE$_x$ | 1c$_x$ | Fd$_x$ | 4d$_x$ | 92$_x$ | 75$_x$ | 06$_x$ | 8A$_x$ |
| A0$_x$ | B2$_x$ | E6$_x$ | 0E$_x$ | 1F$_x$ | 62$_x$ | d4$_x$ | A8$_x$ | 96$_x$ | F9$_x$ | c5$_x$ | 25$_x$ | 59$_x$ | 84$_x$ | 72$_x$ | 39$_x$ | 4c$_x$ |
| B0$_x$ | 5E$_x$ | 78$_x$ | 38$_x$ | 8c$_x$ | d1$_x$ | A5$_x$ | E2$_x$ | 61$_x$ | B3$_x$ | 21$_x$ | 9c$_x$ | 1E$_x$ | 43$_x$ | c7$_x$ | Fc$_x$ | 04$_x$ |
| c0$_x$ | 51$_x$ | 99$_x$ | 6d$_x$ | 0d$_x$ | FA$_x$ | dF$_x$ | 7E$_x$ | 24$_x$ | 3B$_x$ | AB$_x$ | cE$_x$ | 11$_x$ | 8F$_x$ | 4E$_x$ | B7$_x$ | EB$_x$ |
| d0$_x$ | 3c$_x$ | 81$_x$ | 94$_x$ | F7$_x$ | B9$_x$ | 13$_x$ | 2c$_x$ | d3$_x$ | E7$_x$ | 6E$_x$ | c4$_x$ | 03$_x$ | 56$_x$ | 44$_x$ | 7F$_x$ | A9$_x$ |
| E0$_x$ | 2A$_x$ | BB$_x$ | c1$_x$ | 53$_x$ | dc$_x$ | 0B$_x$ | 9d$_x$ | 6c$_x$ | 31$_x$ | 74$_x$ | F6$_x$ | 46$_x$ | Ac$_x$ | 89$_x$ | 14$_x$ | E1$_x$ |
| F0$_x$ | 16$_x$ | 3A$_x$ | 69$_x$ | 09$_x$ | 70$_x$ | B6$_x$ | d0$_x$ | Ed$_x$ | cc$_x$ | 42$_x$ | 98$_x$ | A4$_x$ | 28$_x$ | 5c$_x$ | F8$_x$ | 86$_x$ |