# APPENDIX N
# WHIRLPOOL

**William Stallings**




Supplement to
*Cryptography and Network Security, Fifth Edition*
William Stallings
Prentice Hall 2010
ISBN-10: 0136097049
http://williamstallings.com/Crypto/Crypto5e.html

In this appendix, we examine the hash function Whirlpool [BARR03], one of whose designers is also co-inventor of Rijndael, adopted as the Advanced Encryption Standard (AES). Whirlpool is one of only two hash functions endorsed by NESSIE (New European Schemes for Signatures, Integrity, and Encryption).[1] The NESSIE project is a European Union-sponsored effort to put forward a portfolio of strong cryptographic primitives of various types.

Whirlpool is based on the use of a block cipher for the compression function. There has traditionally been little interest in the use of block-cipher-based hash functions because of the demonstrated security vulnerabilities of the structure. The following are potential drawbacks:

1. Block ciphers do not possess the properties of randomizing functions. For example, they are invertible. This lack of randomness may lead to weaknesses that can be exploited.
2. Block ciphers typically exhibit other regularities or weaknesses. For example, [MIYA90] demonstrates how to compromise many hash schemes based on properties of the underlying block cipher.
3. Typically, block-cipher-based hash functions are significantly slower than hash functions based on a compression function specifically designed for the hash function.
4. A principal measure of the strength of a hash function is the length of the hash code in bits. For block-cipher-based hash codes, proposed designs have a hash code length equal to either the cipher block length or twice the cipher block length. Traditionally, cipher block length has been limited to 64 bits (e.g., DES, triple DES), resulting in a hash code of questionable strength.

However, since the adoption of AES, there has been renewed interested in developing a secure hash function based on a strong block cipher and exhibiting good performance. Whirlpool is a block-cipher-based hash function intended to provide security and performance that is comparable, if not better, than that found in non-block-cipher based hash functions, such as SHA. Whirlpool has the following features:

---

[1] The other endorsed scheme consists of three variants of SHA: SHA-256, SHA-384, and SHA-512.

1. The hash code length is 512 bits, equaling the longest hash code available with SHA.
2. The overall structure of the hash function is one that has been shown to be resistant to the usual attacks on block-cipher-based hash codes.
3. The underlying block cipher is based on AES and is designed to provide for implementation in both software and hardware that is both compact and exhibits good performance.

The design of Whirlpool sets the following security goals: Assume we take as hash result the value of any $n$-bit substring of the full Whirlpool output.

- 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.
- It is infeasible to detect systematic correlations between any linear combination of input bits and any linear combination of bits of the hash result, or to predict what bits of the hash result will change value when certain input bits are flipped (this means resistance against linear and differential attacks).

The designers assert their confidence that these goals have been met with a considerable safety margin. However, the goals are not susceptible to a formal proof.

We begin with a discussion of the structure of the overall hash function, and then examine the block cipher used as the basic building block.

## N.1 WHIRLPOOL HASH STRUCTURE

### Background

The general iterated hash structure proposed by Merkle (Figure 11.7) is used in virtually all secure hash functions. However, as was pointed out, there are difficulties in designing a truly

secure iterated hash function when the compression function is a block cipher. Preneel [PREN93a, PREN93b] performed a systematic analysis of block-cipher-based hash functions, using the model depicted in Figure N.1. In this model, the hash code length equals the cipher block length. Additional security problems are introduced and the analysis is more difficult if the hash code length exceeds the cipher block length. Preneel devised 64 possible permutations of the basic model, based on which input served as the encryption key and which served as plaintext and on what input, if any, was combined with the ciphertext to produce the intermediate hash code. Based on his analysis, he concluded that only schemes in which the plaintext was fed forward and combined with the ciphertext were secure. Such an arrangement makes the compression function difficult to invert. [BLAC02] confirmed these results, but pointed out the security problem of using an established block cipher such as AES: The 128-bit hash code value resulting from the use of AES or another scheme with the same block size may be inadequate for security.

## Whirlpool Logic

Given a message consisting of a sequence of blocks $m_1, m_2, \ldots, m_t$, the Whirlpool hash function is expressed as follows:

$$\begin{aligned} H_0 &= \text{initial value} \\ H_i &= \mathrm{E}(H_{i-1}, m_i) \oplus H_{i-1} \oplus m_i = \text{intermediate value} \\ H_t &= \text{hash code value} \end{aligned}$$

In terms of the model of Figure N.1, the encryption key input for each iteration is the intermediate hash value from the previous iteration; the plaintext is the current message block; and the feedforward value is the bitwise XOR of the current message block and the intermediate hash value from the previous iteration.

The algorithm takes as input a message with a maximum length of less than $2^{256}$ bits and produces as output a 512-bit message digest. The input is processed in 512-bit blocks. Figure N.2 depicts the overall processing of a message to produce a digest. This follows the general structure depicted in Figure 11.7. The processing consists of the following steps:

- **Step 1: Append padding bits.** The message is padded so that its length in bits is an odd multiple of 256. Padding is always added, even if the message is already of the desired length. For example, if the message is $256 \times 3 = 768$ bits long, it is padded by 512 bits to a length of $256 \times 5 = 1280$ bits. Thus, the number of padding bits is in the range of 1 to 512.

  The padding consists of a single 1-bit followed by the necessary number of 0-bits.
- **Step 2: Append length.** A block of 256 bits is appended to the message. This block is treated as an unsigned 256-bit integer (most significant byte first) and contains the length in bits of the original message (before the padding).

  The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length. In Figure N.2, the expanded message is represented as the sequence of 512-bit blocks $m_1, m_2, \ldots, m_t$, so that the total length of the expanded message is $t \times 512$ bits. These blocks are viewed externally as arrays of bytes by sequentially grouping the bits in 8-bit chunks. However, internally, the hash state $H_i$ is viewed as an $8 \times 8$ matrix of bytes. The transformation between the two is explained subsequently.
- **Step 3: Initialize hash matrix.** An $8 \times 8$ matrix of bytes is used to hold intermediate and final results of the hash function. The matrix is initialized as consisting of all 0-bits.
- **Step 4: Process message in 512-bit (64-byte) blocks.** The heart of the algorithm is the block cipher W.

## N.2 BLOCK CIPHER W

Unlike virtually all other proposals for a block-cipher-based hash function, Whirlpool uses a block cipher that is specifically designed for use in the hash function and that is unlikely ever to be used as a standalone encryption function. The reason for this is that the designers wanted to make use of a block cipher with the security and efficiency of AES but with a hash length that provided a potential security equal to SHA-512. The result is the block cipher W, which has a similar structure and uses the same elementary functions as AES, but which uses a block size and a key size of 512 bits. Table N.1 compares AES and W.

**Table N.1 Comparison of Whirlpool Block Cipher W and AES**

| | W | AES |
|---|---|---|
| **Block size (bits)** | 512 | 128 |
| **Key size (bits)** | 512 | 128, 192, or 256 |
| **Matrix orientation** | Input is mapped row-wise | Input is mapped column-wise |
| **Number of rounds** | 10 | 10, 12, or 14 |
| **Key expansion** | W round function | dedicated expansion algorithm |
| **GF($2^8$) polynomial** | $x^8 + x^4 + x^3 + x^2 + 1$ (011D) | $x^8 + x^4 + x^3 + x + 1$ (011B) |
| **Origin of S-box** | recursive structure | multiplicative inverse in GF($2^8$) plus affine transformation |
| **Origin of round constants** | Successive entries of the S-box | elements $2^i$ of GF($2^8$) |
| **Diffusion layer** | right multiplication by 8×8 circulant MDS matrix (1, 1, 4, 1, 8, 5, 2, 9) - mix rows | left multiplication by 4×4 circulant MDS matrix (2, 3, 1, 1) - mix columns |
| **Permutation** | shift columns | shift rows |

Although W is similar to AES, it is not simply an extension. Recall that the Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. AES operates on a state of 4×4 bytes. Rijndael with block length 192 bits operates on a state of 4×6 bytes. Rijndael with block length 256 bits operates on a state of 4×8 bytes. W operates on a state of 8×8 bytes. The more the state representation differs from a square, the slower the diffusion goes and the more rounds the cipher needs. For a block length of 512 bits, the Whirlpool developers could have defined a Rijndael

operating on a state of 4×16 bytes, but that cipher would have needed many rounds and it would have been very slow.

As Table N.1 indicates, W uses a row-oriented matrix whereas AES uses a column-oriented matrix. There is no technical reason to prefer one orientation over another, because one can easily construct an equivalent description of the same cipher, exchanging rows with columns.

## Overall Structure

Figure N.3 shows the overall structure of W. The encryption algorithm takes a 512-bit block of plaintext and a 512-bit key as input and produces a 512-bit block of ciphertext as output. The encryption algorithm involves the use of four different functions, or transformations: add key (AK), substitute bytes (SB), shift columns (SC), and mix rows (MR), whose operations are explained subsequently. W consists of a single application of AK followed by 10 rounds that involve all four functions. We can concisely express the operation of a round $r$ as a round function RF that is a composition of functions:

$$\mathrm{RF}(K_r) = \mathrm{AK}[\boldsymbol{K}_r] \circ \mathrm{MR} \circ \mathrm{SC} \circ \mathrm{SB} \qquad \textbf{(N.1)}$$

where $\boldsymbol{K}_r$ is the round key matrix for round $r$. The overall algorithm, with key input $\boldsymbol{K}$, can be defined as follows:

$$\mathrm{W}(\boldsymbol{K}) = \left(\bigcirc_{r=1}^{10} \mathrm{RF}(\boldsymbol{K}_r)\right) \circ \mathrm{AK}(\boldsymbol{K}_0)$$

where the large circle indicates iteration of the composition function with index $r$ running from 1 through 10.

The plaintext input to W is a single 512-bit block. This block is treated as an 8 × 8 square matrix of bytes, labeled **CState**. Figure N.4 illustrates that the ordering of bytes within a matrix is by row. So, for example, the first eight bytes of a 512-bit plaintext input to the encryption cipher occupy the first row of the internal matrix **CState**, the second eight bytes occupy the second row, and so on. The representation of the linear byte stream as a square matrix can be concisely expressed as a mapping function $\mu$. For a linear byte array X with elements $x_k$ ($0 \le k \le$

63), the corresponding matrix **A** with elements $a_{i,j}$ $(0 \le i, j \le 7)$, we have the following correspondence:

$$\mathbf{A} = \mu(\mathrm{X}) \Leftrightarrow a_{i,j} = x_{8i+j}$$

Similarly, the 512-bit key is depicted as a square matrix **KState** of bytes. This key is used as input to the initial AK function. The key is also expanded into a set of 10 round keys, as explained subsequently.

We now look at the individual functions that are part of W.

# Table N.2 Whirlpool S-Box

**(a) S-box**

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 18 | 23 | C6 | E8 | 87 | B8 | 01 | 4F | 36 | A6 | D2 | F5 | 79 | 6F | 91 | 52 |
| 1 | 60 | BC | 9B | 8E | A3 | 0C | 7B | 35 | 1D | E0 | D7 | C2 | 2E | 4B | FE | 57 |
| 2 | 15 | 77 | 37 | E5 | 9F | F0 | 4A | CA | 58 | C9 | 29 | 0A | B1 | A0 | 6B | 85 |
| 3 | BD | 5D | 10 | F4 | CB | 3E | 05 | 67 | E4 | 27 | 41 | 8B | A7 | 7D | 95 | C8 |
| 4 | FB | EE | 7C | 66 | DD | 17 | 47 | 9E | CA | 2D | BF | 07 | AD | 5A | 83 | 33 |
| 5 | 63 | 02 | AA | 71 | C8 | 19 | 49 | C9 | F2 | E3 | 5B | 88 | 9A | 26 | 32 | B0 |
| 6 | E9 | 0F | D5 | 80 | BE | CD | 34 | 48 | FF | 7A | 90 | 5F | 20 | 68 | 1A | AE |
| 7 | B4 | 54 | 93 | 22 | 64 | F1 | 73 | 12 | 40 | 08 | C3 | EC | DB | A1 | 8D | 3D |
| 8 | 97 | 00 | CF | 2B | 76 | 82 | D6 | 1B | B5 | AF | 6A | 50 | 45 | F3 | 30 | EF |
| 9 | 3F | 55 | A2 | EA | 65 | BA | 2F | C0 | DE | 1C | FD | 4D | 92 | 75 | 06 | 8A |
| A | B2 | E6 | 0E | 1F | 62 | D4 | A8 | 96 | F9 | C5 | 25 | 59 | 84 | 72 | 39 | 4C |
| B | 5E | 78 | 38 | 8C | C1 | A5 | E2 | 61 | B3 | 21 | 9C | 1E | 43 | C7 | FC | 04 |
| C | 51 | 99 | 6D | 0D | FA | DF | 7E | 24 | 3B | AB | CE | 11 | 8F | 4E | B7 | EB |
| D | 3C | 81 | 94 | F7 | B9 | 13 | 2C | D3 | E7 | 6E | C4 | 03 | 56 | 44 | 7F | A9 |
| E | 2A | BB | C1 | 53 | DC | 0B | 9D | 6C | 31 | 74 | F6 | 46 | AC | 89 | 14 | E1 |
| F | 16 | 3A | 69 | 09 | 70 | B6 | C0 | ED | CC | 42 | 98 | A4 | 28 | 5C | F8 | 86 |

**(b) E mini-box**

| $u$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E($u$) | 1 | B | 9 | C | D | 6 | F | 3 | E | 8 | 7 | 4 | A | 2 | 5 | 0 |

**(c) $E^{-1}$ mini-box**

| $u$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $E^{-1}(u)$ | F | 0 | D | 7 | B | E | 5 | A | 9 | 2 | C | 1 | 3 | 4 | 8 | 6 |

**(d) R mini-box**

| $u$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R($u$) | 7 | C | B | D | E | 4 | 9 | F | 6 | 3 | 8 | A | 2 | 5 | 1 | 0 |

## The Nonlinear Layer SB

The substitute byte function (SB) is a simple table lookup that provides a nonlinear mapping. W defines a 16 × 16 matrix of byte values, called an S-box (Table N.2), that contains a permutation of all possible 256 8-bit values. Each individual byte of **CState** is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value[2] {95} references row 9, column 5 of the S-box, which contains the value {BA}. Accordingly, the value {95} is mapped into the value {BA}. The SB function can be expressed by the following correspondence, for an input matrix **A** and an output matrix **B**:

$$\mathbf{B} = \mathrm{SB}(\mathbf{A}) \;\Leftrightarrow\; b_{i,j} = \mathrm{S}[a_{i,j}], \quad 0 \le i, j \le 7$$

where S[$x$] refers to the mapping of input byte $x$ into output byte S[$x$] by the S-box.

The S-box can be generated by the structure of Figure N.5. It consists of two nonlinear layers, each containing two 4 × 4 S-boxes separated by a 4 × 4 randomly generated box. Each of the boxes maps a 4-bit input into a 4-bit output. The E box is defined as $\mathrm{E}(u) = \{\mathrm{B}\}^u$ if $u \neq \{\mathrm{F}\}$ and $\mathrm{E}(\{\mathrm{F}\}) = 0$, where arithmetic is performed over the finite field GF($2^4$) with the irreducible polynomial $f(x) = x^4 + x + 1$.

The SB function is designed to introduce nonlinearity into the algorithm. This means that the SB function should exhibit no correlations between linear combinations of input bits and linear combinations of output bits. In addition, differences between sets of input bits should not propagate into similar differences among the corresponding output bits; put another way, small input changes should cause large output changes. These two properties help to make W resistant against linear and differential cryptanalysis.

## The Permutation Layer SC

---

[2] As we did for AES, a hexadecimal number is indicated by enclosing it in curly brackets when this is needed for clarity.

The permutation layer (shift columns) causes a circular downward shift of each column of **CState** except the first column. For the second column, a 1-byte circular downward shift is performed; for the third column, a 2-byte circular downward shift is performed; and so on. The SC function can be expressed by the following correspondence, for an input matrix **A** and an output matrix **B**:

$$\mathbf{B} = \mathrm{SC}(\mathbf{A}) \Leftrightarrow b_{i,j} = a_{(i-j) \bmod 8, j} \qquad 0 \le i, j \le 7$$

The shift column transformation is more substantial than it may first appear. This is because **CState** is treated as an array of eight 8-byte rows. Thus, on encryption, the first 8 bytes of the plaintext are copied to the first row of **CState**, and so on. A column shift moves an individual byte from one row to another, which is a linear distance of a multiple of 8 bytes. Also note that the transformation ensures that the 8 bytes of one row are spread out to eight different rows.

## The Diffusion Layer MR

Recall from Chapter 3 that for a function that exhibits diffusion, the statistical structure of the input is dissipated into long-range statistics of the output. This is achieved by having each input bit affect the value of many output bits; generally, this results in each output bit being affected by many input bits. The diffusion layer (mix rows) achieves diffusion within each row individually. Each byte of a row is mapped into a new value that is a function of all eight bytes in that row. The transformation can be defined by the matrix multiplication: **B** = **AC**, where **A** is the input matrix, **B** is the output matrix, and **C** is the transformation matrix:

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

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications[3] are performed in GF($2^8$) with the irreducible polynomial $f(x) = x^8 + x^4 + x^3 + x^2 + 1$. As an example of the matrix multiplication involved, the first element of the output matrix is

$$b_{0,0} = a_{0,0} \oplus (9 \bullet a_{0,1}) \oplus (2 \bullet a_{0,2}) \oplus (5 \bullet a_{0,3}) \oplus (8 \bullet a_{0,4}) \oplus a_{0,5} \oplus (4 \bullet a_{0,6}) \oplus a_{0,7}$$

Note that each row of **C** is constructed by means of a circular right shift of the preceding row. **C** is designed to be a **maximum distance separable** (MDS) matrix. In the field of error-correcting codes, an MDS code takes as input a fixed-length bit string and produces an expanded output string such that there is the maximum Hamming distance between pairs of output strings. With an MDS code, even multiple bit errors result in a code that is closer to the correct value than to some other value. In the context of block ciphers, a transformation matrix constructed using an MDS code provides a high degree of diffusion [JUNO04]. The use of MDS codes to provide high diffusion was first proposed in [RIJM96].

The matrix **C** is an MDS matrix that has as many 1-elements as possible (3 per row). Overall, the coefficients in **C** provide for efficient hardware implementation.

## The Add Key Layer AK

In the add key layer, the 512 bits of **CState** are bitwise XORed with the 512 bits of the round key. The AK function can be expressed by the following correspondence, for an input matrix **A**, an output matrix **B**, and a round key $K_i$:

$$\mathbf{B} = \text{AK}[K_i](\mathbf{A}) \;\Leftrightarrow\; b_{i,j} = a_{i,j} \oplus k_{i,j}, \quad 0 \le i, j \le 7$$

---

[3] As we did for AES, we use the symbol • to indicate multiplication over the finite field GF($2^8$) and ⊕ to indicate bitwise XOR, which corresponds to addition in GF($2^8$).

## Key Expansion for the Block Cipher W

As shown in Figure N.3, key expansion is achieved by using the block cipher itself, with a round constant serving as the round key for the expansion. The round constant for round $r$ ($1 \le r \le 10$) is a matrix **RC**[$r$] in which only the first row is nonzero, and is defined as follows:

$$\text{rc}[r]_{0,j} = \text{S}[8(r-1)+j], \qquad 0 \le j \le 7, 1 \le r \le 10$$
$$\text{rc}[r]_{i,j} = 0, \qquad 1 \le i \le 7, 0 \le j \le 7, 1 \le r \le 10$$

Each element of the first row is a mapping using the S-box. Thus, the first row of **RC**[1] is

| S[0] | S[1] | S[2] | S[3] | S[4] | S[5] | S[6] | S[7] |
|---|---|---|---|---|---|---|---|

=

| 18 | 23 | C6 | E8 | 87 | B8 | 01 | 4F |
|---|---|---|---|---|---|---|---|

Using the round constants, the key schedule expands the 512-bit cipher key $\boldsymbol{K}$ onto a sequence of round keys $\boldsymbol{K}_0, \boldsymbol{K}_1, \ldots, \boldsymbol{K}_{10}$:

$$K_0 = K$$
$$K_r = \text{RF}[\mathbf{RC}[r]](K_{r-1})$$

where RF is the round function defined in Equation (N.1). Note that for the AK phase of each round, only the first row of **KState** is altered.

## REFERENCES


**BARR03** Barreto, P., and Rijmen, V. "The Whirlpool Hashing Function." *Submitted to NESSIE*, September 2000, revised May 2003.

**BLAC02** Black, J., and Rogaway, P. "Black-Box Analysis of the Block-Cipher-Based Hash Function Constructions from PGV." *Advances in Cryptology – CRYPTO '02*, 2002.

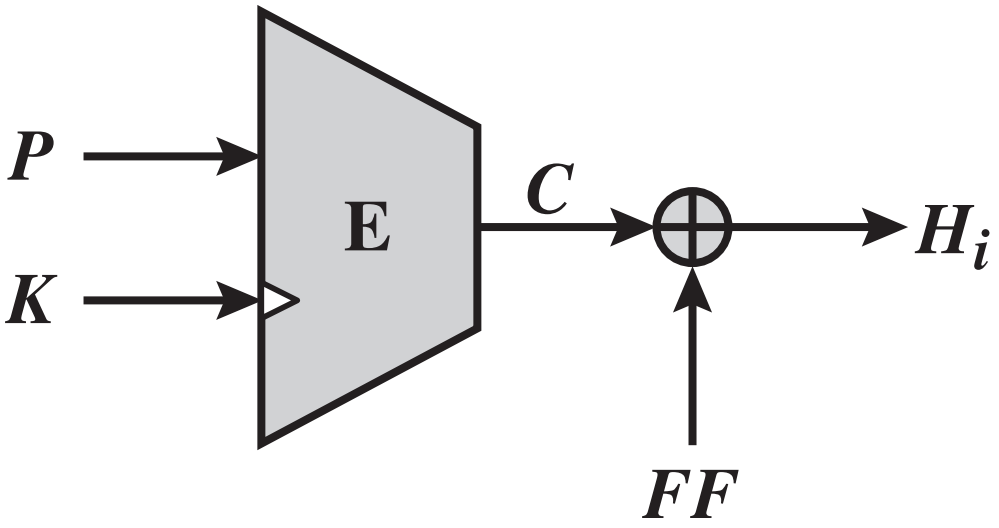
**JUNO04** Junod, P., and Vaudenay, S. "Perfect Diffusion Primitives for Block Ciphers: Building Efficient MDS Matrices." *Selected Areas in Cryptography 2004*, Waterloo, Canada, August 9-10, 2004.

**MIYA90** Miyaguchi, S.; Ohta, K.; and Iwata, M. "Confirmation that Some Hash Functions Are Not Collision Free." *Proceedings, EUROCRYPT '90*, 1990; published by Springer-Verlag.

**PREN93a** Preneel, B.; Govaerta, R.; and Vandewalle, J. "Hash Functions Based on Block Ciphers: a Synthetic Approach." *Proceedings, Advances in Cryptology – CRYPTO '93*, 1993

**PREN93b** Preneel, B. "Cryptographic Hash Functions." *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography*, 1993.

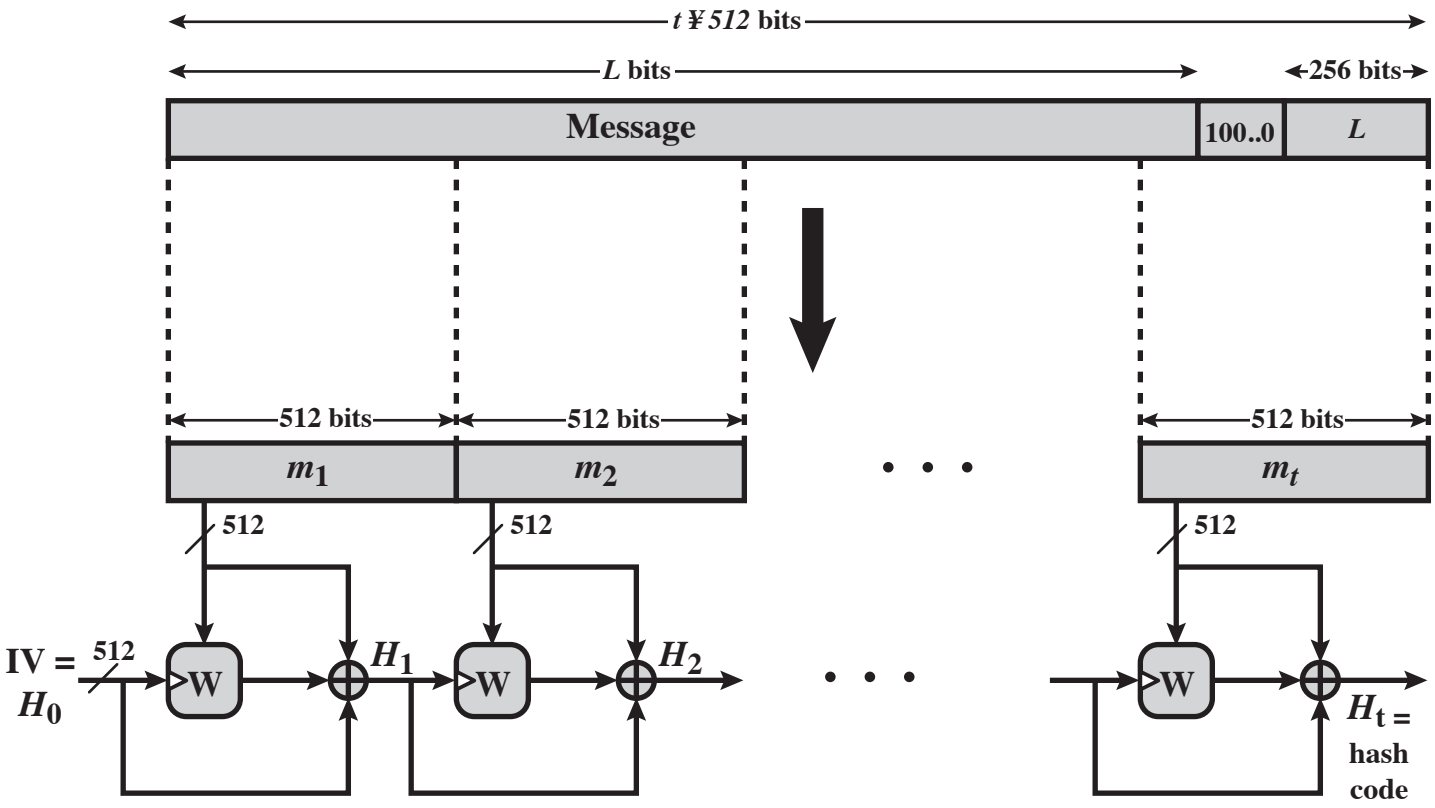
**RIJM96** Rijmen, V.; Daemen, J.; Preneel, B.; Bosselares, A.; and Win, E. "The Cipher SHARK." *Fast Software Encryption, FSE '96*, 1996.




**$m_i$ = $i$th block of message input**
**$H_i$ = $i$th intermediate hash value**
**$P$ = plaintext; $K$ = encryption key; $C$ = ciphertext**
**$FF$ = feed forward value**
**$P$, $K$, and $FF$ can be chosen from the set $(0, m_i, H_{i-1}, m_i \approx H_{i-1})$**

**Note: triangular hatch indicates encryption key input**

## Figure N.1 Model of Single Iteration of Hash Function (hash code equals block length)




Note: triangular hatch marks key input

**Figure N.2 Message Digest Generation Using Whirlpool**

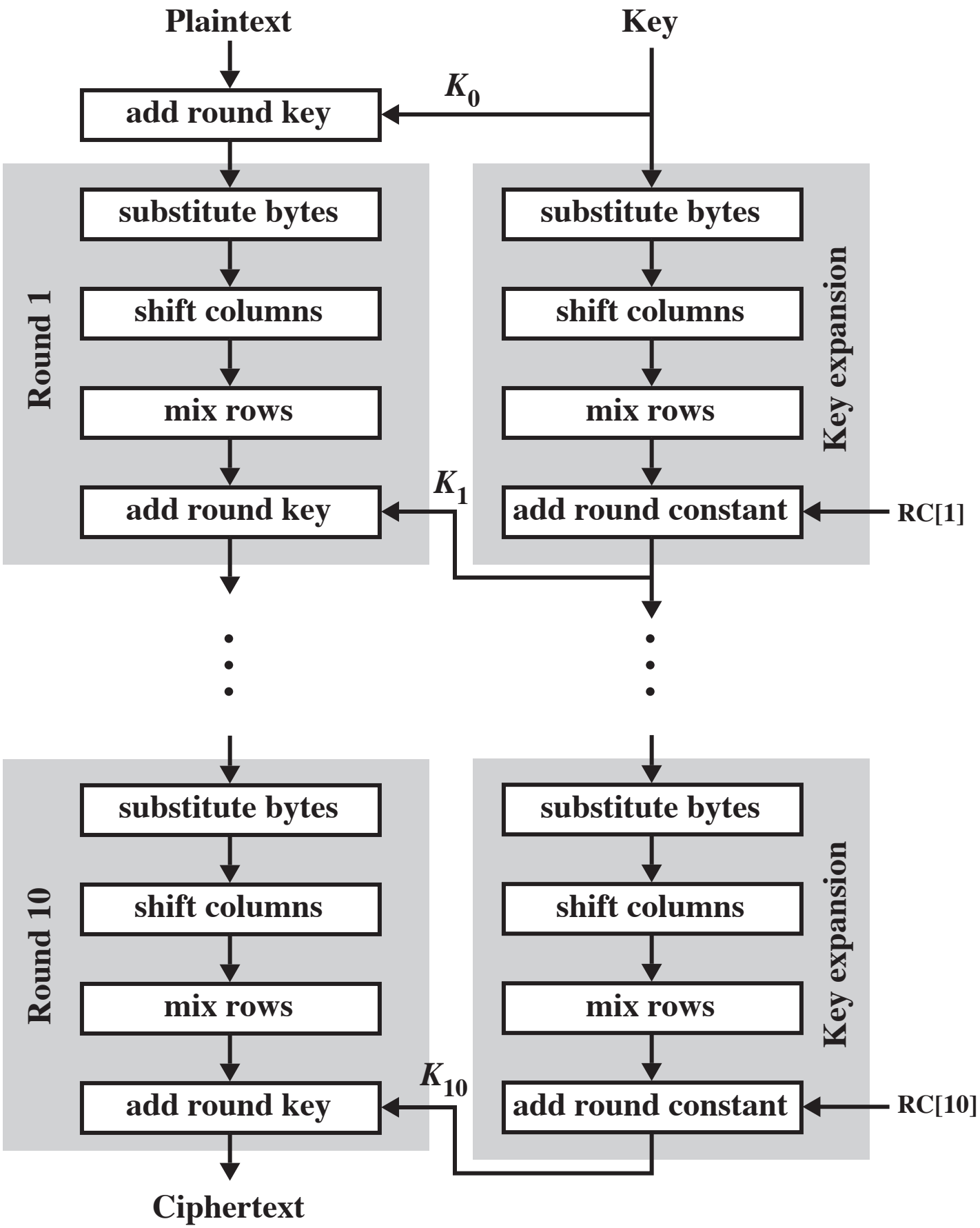



**Figure N.3 Whirlpool Cipher W**

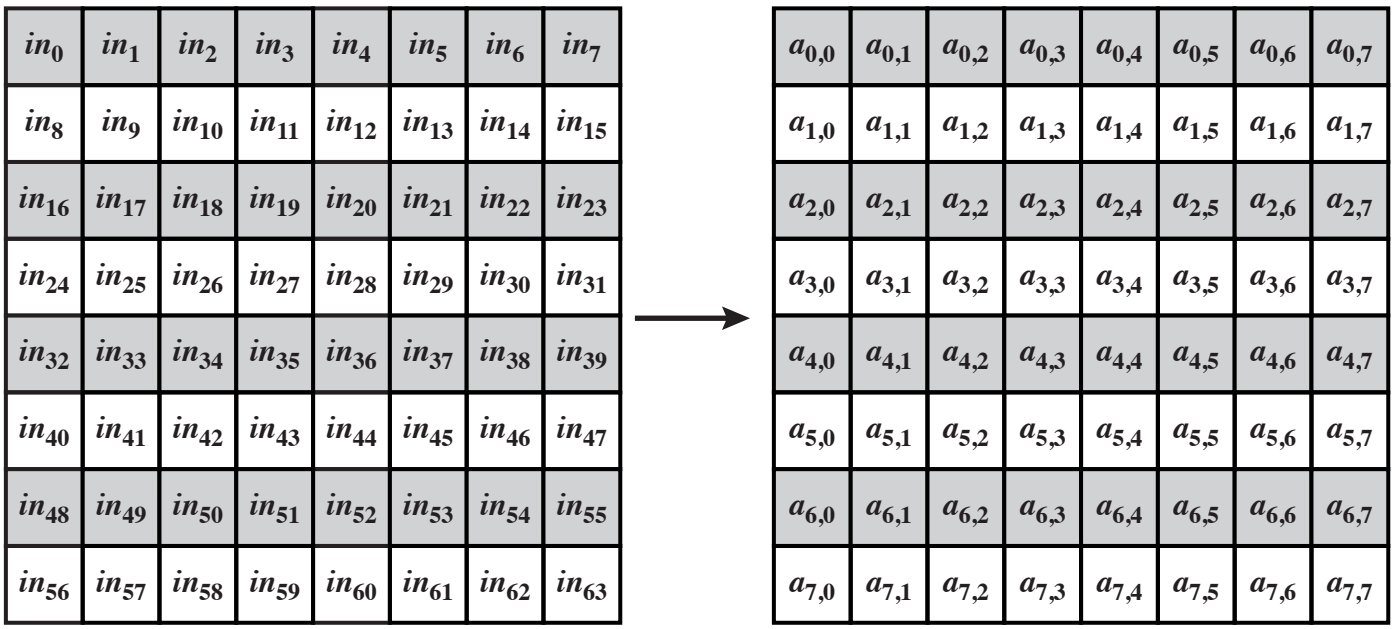



**Figure N.4 Whirlpool Matrix Structure**

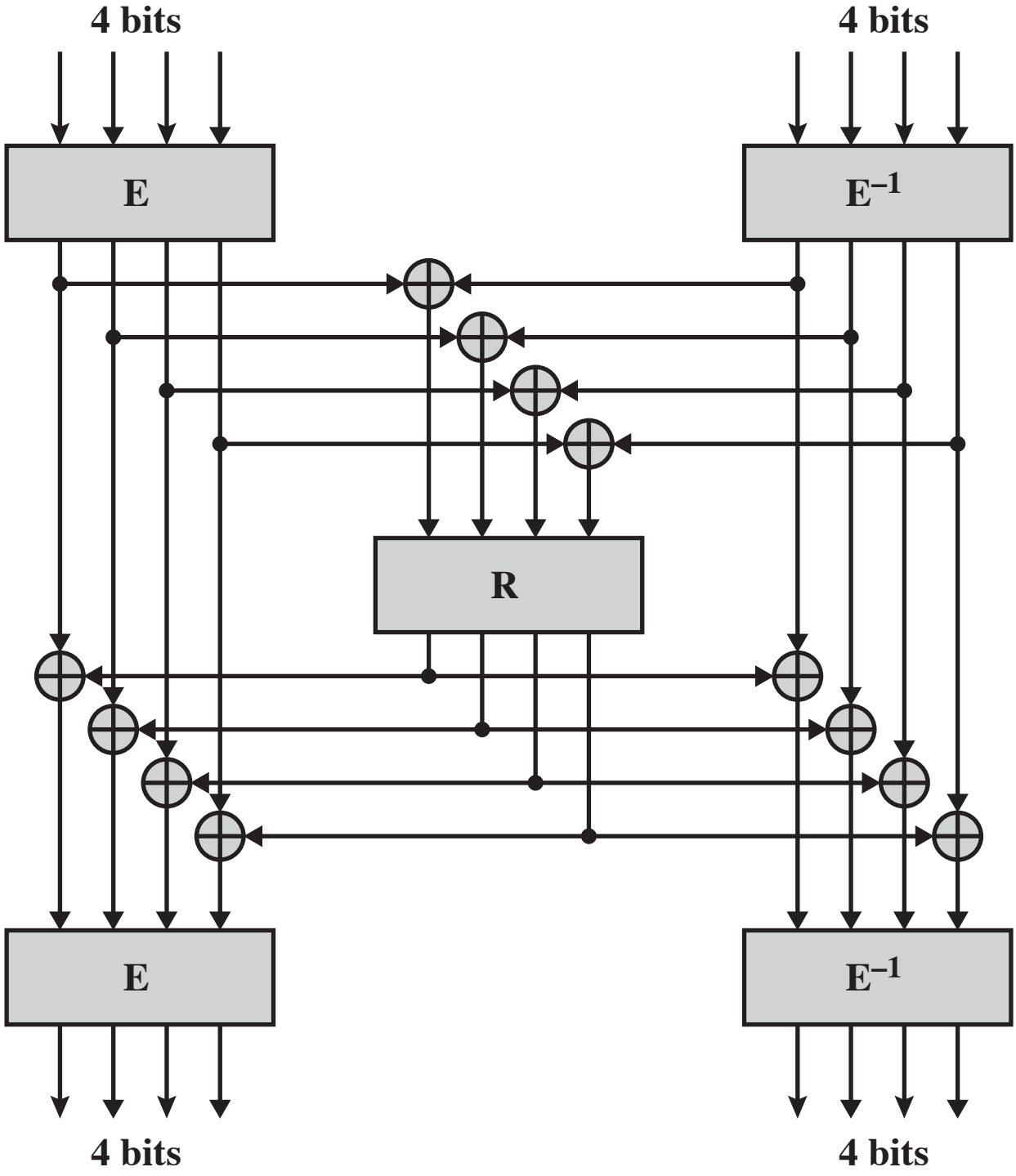



**Figure N.5 Implementation of Whirlpool S-Box**