Cryptologia, 30:55–67, 2006
Copyright © Taylor & Francis Group, LLC
ISSN: 0161-1194 print
DOI: 10.1080/01611190500380090

The Whirlpool Secure Hash Function

WILLIAM STALLINGS

Abstract In this paper, we describe Whirlpool, which is a block-cipher-based secure hash function. Whirlpool produces a hash code of 512 bits for an input message of maximum length less than 2^{256} bits. The underlying block cipher, based on the Advanced Encryption Standard (AES), takes a 512-bit key and operates on 512-bit blocks of plaintext. Whirlpool has been endorsed by NESSIE (New European Schemes for Signatures, Integrity, and Encryption), which is a European Union-sponsored effort to put forward a portfolio of strong cryptographic primitives of various types.

Keywords advanced encryption standard, block cipher, hash function, symmetric cipher, Whirlpool

Introduction

In this paper, we examine the hash function Whirlpool [1]. Whirlpool was developed by Vincent Rijmen, a Belgian who is co-inventor of Rijndael, adopted as the Advanced Encryption Standard (AES); and by Paulo Barreto, a Brazilian cryptographer. Whirlpool is one of only two hash functions endorsed by NESSIE (New European Schemes for Signatures, Integrity, and Encryption) [13].[1] The NESSIE project is a European Union-sponsored effort to put forward a portfolio of strong cryptographic primitives of various types, including block ciphers, symmetric ciphers, hash functions, and message authentication codes.

Background

An essential element of most digital signature and message authentication schemes is a hash function. A hash function accepts a variable-size message M as input and produces a fixed-size hash code $H(M)$, sometimes called a message digest, as output. For a digital signature, a hash code is generated for a message, encrypted with the sender's private key, and sent with the message. The receiver computes a new hash code for the incoming message, decrypts the hash code with the sender's public key and compares. If the message has been altered in transit, there will be a mismatch.

To be useful for message authentication and digital signature, a hash function H must have the following properties:

Address correspondence to William Stallings, c/o Cryptologia, Department of Mathematical Sciences, United States Military Academy, West Point NY 10996, USA. E-mail: ws@shore.net

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

1. H can be applied to a block of data of variable size.
2. H produces a fixed-length output.
3. $H(x)$ is relatively easy to compute for any given x, making both hardware and software implementations practical.
4. For any given value h, it is computationally infeasible to find x such that $H(x) = h$. This is sometimes referred to in the literature as the **one-way property**.
5. For any given block x, it is computationally infeasible to find $y \neq x$ such that $H(y) = H(x)$. This is sometimes referred to as **weak collision resistance**.
6. It is computationally infeasible to find any pair (x, y) such that $H(x) = H(y)$. This is sometimes referred to as **strong collision resistance**.

The general iterated hash structure proposed by Merkle [8] and Damgard [3] is used in virtually all secure hash functions. The hash algorithm involves repeated use of a **compression function**, f, which takes two inputs (an n-bit input from the previous step, called the *chaining variable*, and a b-bit block) and produces an n-bit output. At the start of hashing, the chaining variable has an initial value that is specified as part of the algorithm. The final value of the chaining variable is the hash value. Often, $b > n$; hence the term *compression*. The hash function can be summarized as follows:

$$\begin{aligned} CV_0 &= IV = \text{initial } n\text{-bit value} \\ CV_i &= f(CV_{i-1}, Y_{i-1}), \quad 1 \leq i \leq L \\ H(M) &= CV_L \end{aligned}$$

where the input to the hash function is a message M consisting of the blocks $Y_0, Y_1, \ldots, Y_{L-1}$.

The motivation for this iterative structure stems from the observation by Merkle [8] and Damgard [3] that if the compression function is collision resistant, then so is the resultant iterated hash function. Therefore, the structure can be used to produce a secure hash function to operate on a message of any length. The problem of designing a secure hash function reduces to that of designing a collision-resistant compression function that operates on inputs of some fixed size.

Most of the published proposals for hash functions fall into one of two categories: those based on the use of a block cipher for the compression function, and those based on the use of a compression function specifically designed for the hash function. In either case, it is difficult to design a hash function that is secure [5, 9, 10].

To appreciate this difficulty, it is worthwhile to summarize some recent history of hash functions. MD4, developed by Ron Rivest, one of the designers of the RSA public-key algorithm, was introduced in 1990 and became quite popular. However, some weaknesses were discovered in MD4 and Rivest soon replaced it with a more complex version, called MD5. Both MD4 and MD5 produce a 128-bit hash code. Rivest conjectured that MD5 is as strong as possible for a 128-bit hash code; namely, the difficulty of coming up with two messages having the same message digest is on the order of 2^{64} operations, whereas the difficulty of finding a message with a given digest is on the order of 2^{128} operations. A series of increasingly sophisticated attacks on MD4 and MD5 (e.g., [4], [21]), however, led RSA Laboratories, where Rivest developed MD4 and MD5, to recommend phasing out these algorithms in favor of the Secure Hash Algorithm (SHA) family [15].

SHA was developed by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in

1993. SHA has a hash length of 160 bits. Thus, from the point of view of brute-force attack, it is stronger than MD5: The difficulty of finding a message with a given digest using a brute-force attack is 2^{160}. When weaknesses were discovered in SHA, a revised version was issued as FIPS 180–1 in 1995 and is generally referred to as SHA-1. SHA is based on the hash function MD4 and its design closely models MD4. SHA-1 produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180–2, that defined three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512. These new versions have the same underlying structure and use the same types of modular arithmetic and logical binary operations as SHA-1. In 2005, NIST announced the intention to phase out approval of SHA-1 and move to a reliance on the other SHA versions by 2010. Shortly thereafter, a research team described an attack in which two separate messages could be found that deliver the same SHA-1 hash using 2^{69} operations, far fewer than the 2^{80} operations previously thought needed to find a collision with an SHA-1 hash [22]. This result should hasten the transition to the other versions of SHA [16].

Given the difficulties encountered with this whole line of hash functions based on essentially the same compression function model, it makes sense to investigate the use of a block-cipher based hash function with a strong cipher as its base.

Preneel [14, 11] performed a systematic analysis of block-cipher-based hash functions. In this study, 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 what input, if any, was combined, using XOR, with the ciphertext to produce the intermediate hash code. Based on his analysis, he concluded that only schemes in which the plaintext was combined with the ciphertext were secure. Such an arrangement makes the compression function difficult to invert. Confirmation of these results are reported in [2], but the authors 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.

The Whirlpool Approach

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 typically exhibit certain regularities or weaknesses. For example, [10] demonstrates how to compromise many hash schemes based on properties of the underlying block cipher.
2. Typically, block-cipher-based hash functions are significantly slower than hash functions based on a compression function specifically designed for the hash function.
3. 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.

Since the adoption of AES, however, 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 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 [2, 14].
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 security goals for Whirlpool, as stated on their Web site (**http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html**), can be expressed as follows. Assume we take as hash result the value of any n-bit substring of the full Whirlpool output. The design of Whirlpool sets the following security goals:

- 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) [12].

The designers assert their confidence that these claims have been met with a considerable safety margin. However, a formal proof of these claims has not been achieved.

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.

Whirlpool Hash Structure

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:

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

The encryption key input for each iteration i is the intermediate hash H_{i-1} value from the previous iteration, and the plaintext is the current message block m_i. The output for this iteration (H_i) consists of the bitwise XOR of the current message block, the intermediate hash value from the previous iteration, and the output from W.

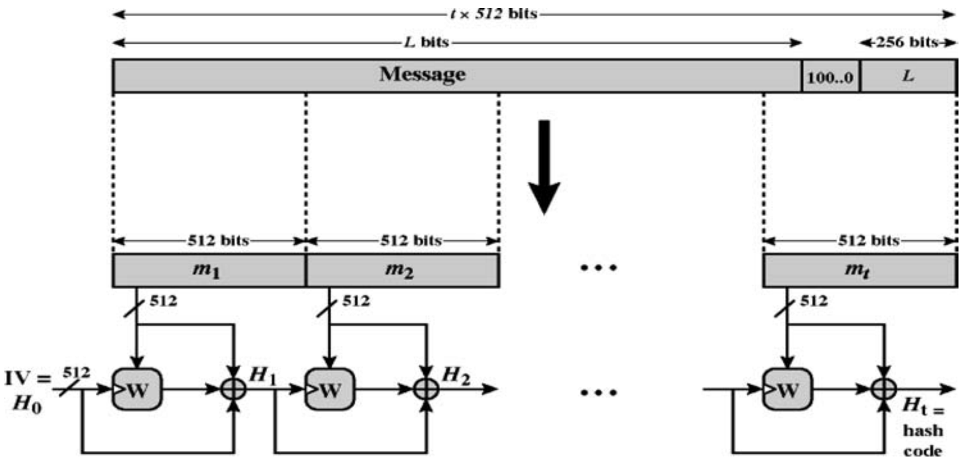

Figure 1. Message digest generation using Whirlpool. Note: triangular hatch marks key input.

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 1 depicts the overall processing of a message to produce a digest. 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 $256 \times 5 = 1{,}280$ 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 1, 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 8×8 matrix of bytes. The transformation between the two is explained subsequently.
- **Step 3: Initialize hash matrix.** An 8×8 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.

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

Table 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

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 [20], but which uses a block size and a key size of 512 bits. Table 1 compares AES and W.

Although W is similar to AES, it is not simply an extension. In fact, AES is one version of the cipher Rijndael, which was submitted as a candidate for the AES. 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 [18].

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

Overall Structure

Figure 2 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,

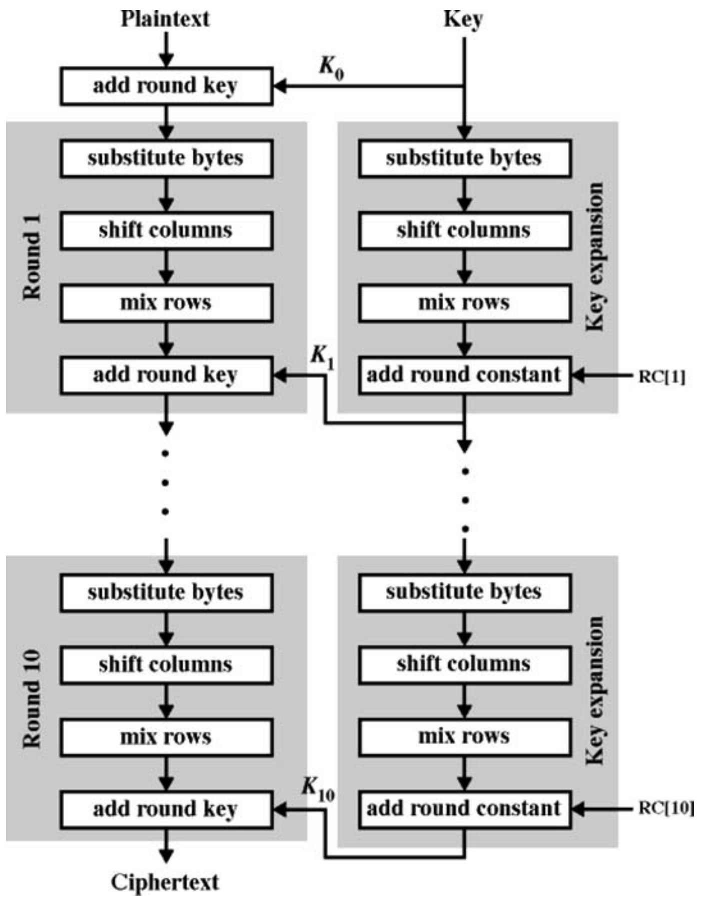

Figure 2. Whirlpool cipher W.

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:

$$RF(\mathbf{K_r}) = \mathbf{AK}[\mathbf{K_r}] \circ \mathbf{MR} \circ \mathbf{SC} \circ \mathbf{SB} \tag{1}$$

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

$$W(\mathbf{K}) = \left(\underset{r=1}{\overset{10}{\mathrm{O}}} RF(\mathbf{K}_r) \right) \circ AK(\mathbf{K}_0)$$

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 3 illustrates that the ordering of bytes within a matrix is by row. So, for example, the first eight bytes of a 512-bit

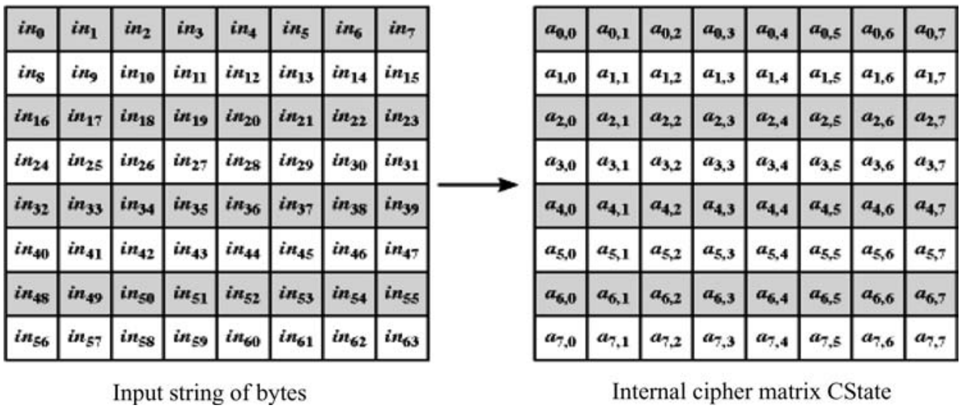
in_0 in_1 in_2 in_3 in_4 in_5 in_6 in_7
in_8 in_9 in_{10} in_{11} in_{12} in_{13} in_{14} in_{15}
in_{16} in_{17} in_{18} in_{19} in_{20} in_{21} in_{22} in_{23}
in_{24} in_{25} in_{26} in_{27} in_{28} in_{29} in_{30} in_{31}
in_{32} in_{33} in_{34} in_{35} in_{36} in_{37} in_{38} in_{39}
in_{40} in_{41} in_{42} in_{43} in_{44} in_{45} in_{46} in_{47}
in_{48} in_{49} in_{50} in_{51} in_{52} in_{53} in_{54} in_{55}
in_{56} in_{57} in_{58} in_{59} in_{60} in_{61} in_{62} in_{63}
Input string of bytes
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{0,4}$ $a_{0,5}$ $a_{0,6}$ $a_{0,7}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,4}$ $a_{1,5}$ $a_{1,6}$ $a_{1,7}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{2,4}$ $a_{2,5}$ $a_{2,6}$ $a_{2,7}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$ $a_{3,4}$ $a_{3,5}$ $a_{3,6}$ $a_{3,7}$
$a_{4,0}$ $a_{4,1}$ $a_{4,2}$ $a_{4,3}$ $a_{4,4}$ $a_{4,5}$ $a_{4,6}$ $a_{4,7}$
$a_{5,0}$ $a_{5,1}$ $a_{5,2}$ $a_{5,3}$ $a_{5,4}$ $a_{5,5}$ $a_{5,6}$ $a_{5,7}$
$a_{6,0}$ $a_{6,1}$ $a_{6,2}$ $a_{6,3}$ $a_{6,4}$ $a_{6,5}$ $a_{6,6}$ $a_{6,7}$
$a_{7,0}$ $a_{7,1}$ $a_{7,2}$ $a_{7,3}$ $a_{7,4}$ $a_{7,5}$ $a_{7,6}$ $a_{7,7}$
Internal cipher matrix CState

Figure 3. Whirlpool matrix structure.

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 μ. For a linear byte array X with elements x_k $(0 \le k \le 63)$, the corresponding matrix $\mathbf{A}$ with elements $a_{i,j}$ $(0 \le i, j \le 7)$, we have the following correspondence:

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

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 11 round keys, as explained subsequently.

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

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 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 $\mathbf{A}$ and an output matrix $\mathbf{B}$:

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

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

[2]A hexadecimal number is indicated by enclosing it in curly brackets when this is needed for clarity.

Table 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	B	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 S-box can be generated by the structure of Figure 4. It consists of two non-linear 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 (Table 2b) is defined as: $E(u) = B^u$ if $u \neq F$ and $E(F) = 0$, where arithmetic is performed over the finite field $GF(2^4)$ with the irreducible polynomial $f(x) = x^4 + x + 1$. The inverse function E^{-1} is also used (Table 2c). The R box is a pseudorandomly generated permutation (Table 2d).

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.

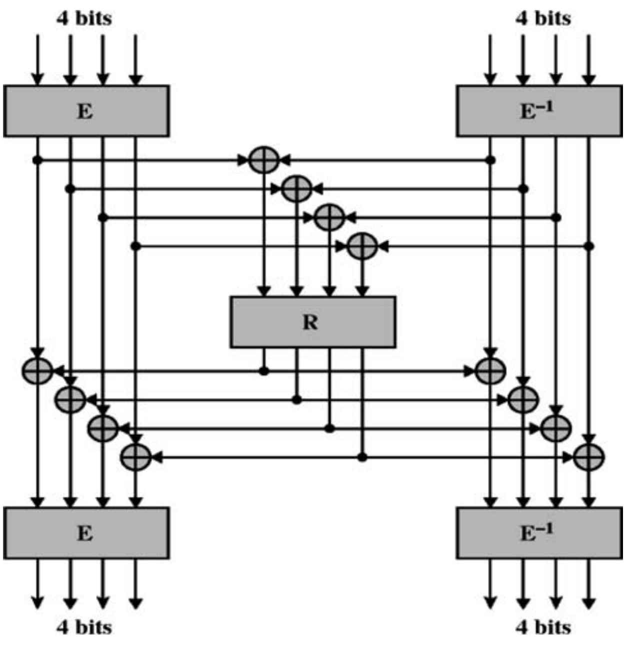

Figure 4. Implementation of Whirlpool CS-Box.

The Permutation Layer SC

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**:

$$\mathbf{B} = SC(\mathbf{A}) \Leftrightarrow b_{i,j} = a_{(i-j)\bmod 8,j} \quad 0 \leq i, j \leq 7.$$

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

Diffusion is a cryptographic property introduced by Claude Shannon [19]. By diffusion, Shannon meant spreading out the influence of a single plaintext digit over many ciphertext digits so as to hide the statistical structure of the plaintext. Generally, this also results in each output digit being affected by many input digits. 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: $\mathbf{B} = \mathbf{AC}$, where $\mathbf{A}$ is the input matrix, $\mathbf{B}$ is the output matrix, and $\mathbf{C}$ is the transformation matrix:

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

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 (Hamming distance = number of bit positions where bit values differ). 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 [6]. The use of MDS codes to provide high diffusion was first proposed in [17].

The matrix **C** is an MDS matrix that has as many 1-elements as possible (3 per row), which leads to an efficient 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:

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

Key Expansion for the Block Cipher W

As shown in Figure 2, 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

[3]We use the symbol $\bullet$ to indicate multiplication over the finite field $GF(2^8)$ and $\oplus$ to indicate bitwise XOR, which corresponds to addition in $GF(2^8)$.

round r $(1 \leq r \leq 10)$ is a matrix $\mathbf{RC}[r]$ in which only the first row is nonzero, and is defined as follows:

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

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

$$\boxed{S[0]}\boxed{S[1]}\boxed{S[2]}\boxed{S[3]}\boxed{S[4]}\boxed{S[5]}\boxed{S[6]}\boxed{S[7]} = \boxed{18}\boxed{23}\boxed{C6}\boxed{E8}\boxed{87}\boxed{B8}\boxed{01}\boxed{4F}$$

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

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

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

Performance of Whirlpool

As yet, there has been little implementation experience with Whirlpool. The NIST evaluation of Rijndael determined that it exhibited good performance (execution speed) in both hardware and software, and it is well suited to restricted-space (low memory) requirements. These criteria were important in the selection of Rijndael for AES. Because Whirlpool uses the same functional building blocks as AES and has the same structure, we can expect similar performance and space characteristics.

One study that has been completed was reported in [7]. The authors compared Whirlpool with a number of other secure hash functions, including all of the versions of SHA. The authors developed multiple hardware implementations of each hash function and concluded that, compared to SHA-512, Whirlpool requires more hardware resources but performs much better in terms of throughput.

About the Author

William Stallings holds a Ph.D. from M.I.T. in Computer Science and a B.S. from Notre Dame in electrical engineering. He has authored numerous books on security, computer networking, and computer architecture. He has five times received the award for the best Computer Science and Engineering textbook of the year from the Textbook and Academic Authors Association. His most recent book is *Cryptography and Network Security, Fourth Edition* (Prentice Hall, 2005). He created and maintains the Computer Science Student Resource Site at WilliamStallings.com/StudentSupport.html. This site provides documents and links on a variety of subjects of general interest to computer science students (and professionals).

References

1. Barreto, P. and V. Rijmen. 2003. The Whirlpool Hashing Function. Submitted to *NESSIE*, May.
2. Black, J., P. Rogaway, and T. Shrimpton. 2002. Black-Box Analysis of the Block-Cipher-Based Hash Function Constructions from PGV, *Proceedings, Advances in Cryptology—CRYPTO '02*, New York: Springer-Verlag, pp. 320–335.
3. Damgard, I. 1989. A Design Principle for Hash Functions, *Proceedings, CRYPTO '89*, New York: Springer-Verlag, pp. 416–427.
4. Dobertin, H. 1996. The Status of MD5 After a Recent Attack, *CryptoBytes*, 2(2): 1–6.
5. Jueneman, R. 1987. Electronic Document Authentication, *IEEE Network Magazine*, 1(2): 17–23.
6. Junod, P. and S. Vaudenay. 2004. Perfect Diffusion Primitives for Block Ciphers: Building Efficient MDS Matrices, *Proceedings, Selected Areas in Cryptography '04*. New York: Springer-Verlag, pp. 84–89.
7. Kitsos, P. and O. Koufopaviou. 2004. Architecture and Hardware Implementation of the Whirlpool Hash Function, *IEEE Transactions on Consumer Electronics*, 50 (1): 208–213.
8. Merkle, R. 1989. One-Way Hash Functions and DES, *Proceedings, CRYPTO '89*. New York: Springer-Verlag, pp. 428–446.
9. Mitchell, C., F. Piper, and P. Wild. 1992. Digital Signatures. Simmons, G. ed, *Contemporary Cryptology: The Science of Information Integrity*. Piscataway, NJ: IEEE Press.
10. Miyaguchi, S., K. Ohta, and M. Iwate. 1990. Confirmation that Some Hash Functions are Not Collision Free, *Proceedings, Advances in Cryptology—EUROCRYPT '90*. New York: Springer-Verlag, pp. 326–343.
11. Preneel, B. 1993. Cryptographic Hash Functions, *Proceedings of the 3rd Symposium on State and Progress of Research in Cryptography*. New York: Springer-Verlag, pp. 161–171.
12. Preneel, B. 1993. Differential Cryptanalysis of Hash Functions Based on Block Ciphers, *ACM Conference on Computer and Communications Security*, pp. 183–188.
13. Preneel, B. 2002. New European Schemes for Signature, Integrity and Encryption (NESSIE): A Status Report, *Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems: Public Key Cryptography. Lecture Notes In Computer Science*, New York: Springer-Verlag, 2274, pp. 297–309.
14. Preneel, B., R. Govaerta, and J. Vandewalle. 1993. Hash Functions Based on Block Ciphers: A Synthetic Approach, *Proceedings, Advances in Cryptology—CRYPTO '93*. New York: Springer-Verlag, pp. 368–378.
15. Randall, J. and M. Szydio. 2004. Collisions for SHA0, MD5, HAVAL, MD4, and RIPEMD, but SHA1 Still Secure, *RSA Laboratories Tech Notes, August 31, 2004* Bedford, MA: RSA Security Inc.
16. Randall, J. 2005. Hash Function Update Due to Potential Weakness Found in SHA-1, *RSA Laboratories Tech Notes, March 11, 2005* Bedford. MA: RSA Security Inc.
17. Rijmen, V. et al. 1996. The Cipher SHARK, *Proceedings, Fast Software Encryption, FSE '96*. New York: Springer-Verlag, pp. 99–111.
18. Rijmen, V. with Willia Stallings. Private communication September 9th, 2005.
19. Shannon, C. 1949. Communication Theory of Secrecy Systems, *Bell Systems Technical Journal*, 28 (4): 656–715.
20. Stallings, W. 2002. The Advanced Encryption Standard, *Cryptologia*, 26 (3): 165–188.
21. Wang, X., D. Feng, and H. Yu. 2004. Collisions for Hash Functions: MD4, MD5, HAVAL-128, and RIPEMD, *Proceedings, Advances in Cryptology—CRYPTO '04*. New York: Springer-Verlag.
22. Wang, X., Y., Yin, and H. Yu. 2005. Finding Collisions in the Full SHA-1, *Proceedings, Advances in Cryptology—CRYPTO '05*. New York: Springer-Verlag.