Control and Cybernetics

vol. **43** (2014) No. 2

Cryptanalysis of the FSR-255 hash function

by

Marcin Kontak[1] * **and Janusz Szmidt**[2]

[1]Institute of Computer Science, Polish Academy of Sciences
ul. Jana Kazimierza 5, 01-248 Warszawa, Poland
m.kontak@ipipan.waw.pl
[2]Military Communication Institute
ul. Warszawska 22A, 05-130 Zegrze, Poland
j.szmidt@wil.waw.pl

Abstract: In this paper we analyse the security of the FSR-255 cryptographic hash function. As a result of our security analysis we present preimage and second-preimage attacks. The attacks base on practical reversibility of the compression function. The complexity of preimage attack is about 2^{11} evaluations of the compression function. The second-preimage attack has the complexity equivalent to one time evaluation of the compression function. Both of the attacks have been practically realised.

Keywords: cryptography, cryptanalysis, FSR-255 hash function, preimage attack, second-preimage attack, collision

1. Introduction

A hash function is a function that maps strings of arbitrary length to strings of fixed length. A cryptographic hash function is a hash function that can be computed efficiently and has three additional security properties: it should be preimage resistant, second-preimage resistant, and collision-resistant.

A hash function h is preimage resistant if for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage m such that $h(m) = y$ when given any y for which a corresponding input is not known.

A hash function h is second-preimage resistant if for essentially all pre-specified outputs, it is computationally infeasible to find any second input which has the same output as any specified input, i.e., given m, to find a second-preimage $m' \neq m$ such that $h(m') = h(m)$.

*This author is supported by the European Union from the resources of the European Social Fund, project PO KL "Information technologies: Research and their interdisciplinary applications".

A hash function h is called collision resistant if it is computationally infeasible to find any two distinct inputs m, m' which hash to the same output, i.e., such that $h(m) = h(m')$.

Collision resistance implies second-preimage resistance (Menezes, van Oorschot and Vanstone, 1996).

2. FSR-255 algorithm

FSR-255 is a dedicated cryptographic hash function with variable length of hash result. The function was proposed by Janicka-Lipska and Stokłosa (2001) and by Gajewski, Janicka-Lipska and Stokłosa (2003). A general idea was based on the previous work of Stokłosa (1995, 1996). The FSR-255 hash function is oriented at hardware implementations.

The FSR-255 hash function takes as input a message m (bitstring) of arbitrary finite length $n \geq 0$ and gives as output a hash result (bitstring) of a desired length $1 \leq r \leq 255$ (in Janicka-Lipska and Stokłosa, 2001, $r \geq 128$ is recommended).

The following steps are performed to compute the hash value $h(m)$.

1. Extending the message. Append padding bits to the message m so that the extended message x is of the form $x = m \parallel 1 \parallel 0^k \parallel b$ where 0^k is the concatenation of k zero bits, b is a 32-bit representation of $n \bmod 2^{32}$ and $k \geq 0$ is the least integer such that the length (in bits) of x is a multiple of 2040.

2. Splitting the extended message. Divide the extended message x into 255-bit words $x_1, x_2, \ldots, x_q$ such that $x = x_1 \parallel x_2 \parallel \ldots \parallel x_q$.

3. Iterative processing. For each $i = 1, 2, \ldots, q$ compute $H_i = F(H_{i-1}, x_i)$, where $F : \{0,1\}^{255} \times \{0,1\}^{255} \to \{0,1\}^{255}$ is a predefined transformation called the *compression function* of the hash function, H_{i-1} serves as the 255-bit *chaining variable* between stage $i-1$ and stage i, and H_0 is a predefined starting value called the *initialization vector*.

4. Final transformation. Compute $G(H_q)$ where $G : \{0,1\}^{255} \to \{0,1\}^r$ is a predefined final transformation.

We can see in Fig. 1 that in order to produce the output hash value, the FSR-255 hash function uses the general model for iterated hash functions as presented in Menezes, van Oorschot and Vanstone (1996).

The compression function F for FSR-255 is defined as follows:

$$H_i = F(H_{i-1}, x_i) = NPB(x_i \oplus y_{i-1} \oplus H_{i-1}) \oplus H_{i-1}, \tag{1}$$

where $\oplus$ denotes the exclusive-or operation, $NPB : \{0,1\}^{255} \to \{0,1\}^{255}$ is a predefined transformation called the *nonlinear processing block*, and $y_0, y_1, \ldots, y_{q-1}$

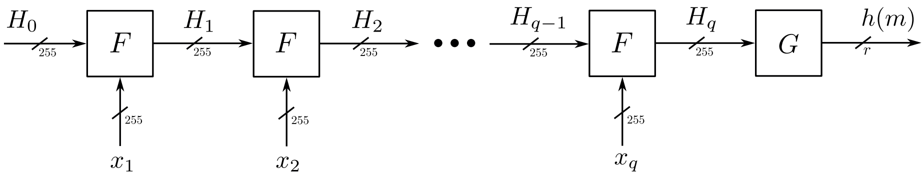

Figure 1. Block diagram of the FSR-255 hash function

are predefined constants such that $y_0 = 0^{255}$, $y_1 = RL_4(H_0)$, and

$$y_i = \begin{cases} RL_4(y_{i-1}) & \text{if } i \not\equiv 0 \pmod 8 \\ H_0 & \text{otherwise} \end{cases} \quad \text{for } i = 2, 3, \ldots, q-1. \tag{2}$$

$RL_k(w)$ denotes left rotation by k bits of the word w, and the initialization vector H_0 = `0x511c` ‖ `0x1b59` ‖ `0x0b4d` ‖ `0x0333` ‖ `0x0979` ‖ `0x04f4` ‖ `0x09ac` ‖ `0x0e0f` ‖ `0x04fa` ‖ `0x0fc3` ‖ `0x01eb` ‖ `0x0353` ‖ `0x01fa` ‖ `0x0674` ‖ `0x0c50` ‖ `0x0e98` ‖ `0x0a75` is a 255-bit word given as the concatenation of seventeen 15-bit words written hexadecimally.

The block diagram of FSR-255 computations is shown in Fig. 2.

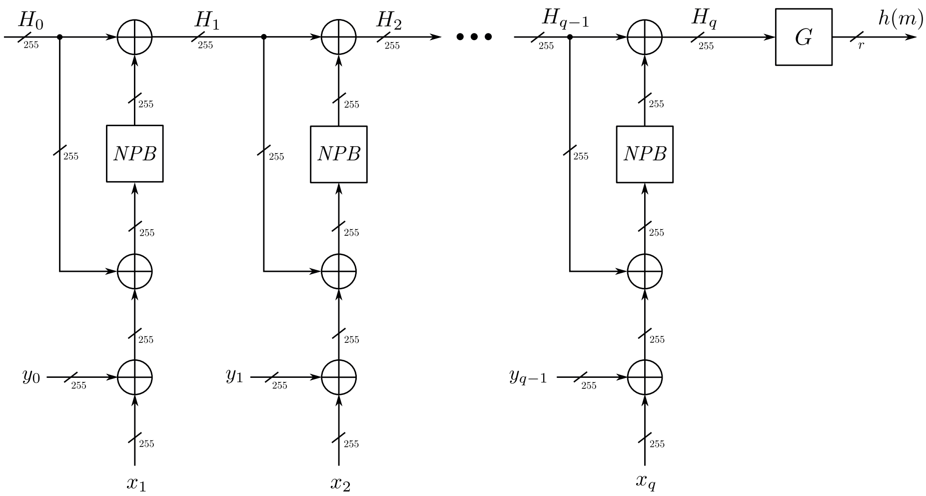

Figure 2. FSR-255 computations

The NPB transformation is a processing structure, shown in Fig. 3, consisting of seventeen 15-bit nonlinear feedback shift registers $NFSR_1$, $NFSR_2$, $\ldots$, $NFSR_{17}$ and the bit permutation function BP. The 255-bit input word v for the NPB transformation is split into seventeen 15-bit words v_i such that $v = v_1 \parallel v_2 \parallel \ldots \parallel v_{17}$, where $v_i \in \{0,1\}^{15}$. Every word v_i is used as initial value for $NFSR_i$. Then, each of the nonlinear feedback shift registers is clocked nineteen times. Afterwards, $BP(z)$ is computed, which gives the output of the

NPB transformation, where $z = z_1 \parallel z_2 \parallel \ldots \parallel z_{17}$ and $z_i \in \{0,1\}^{15}$ denotes the state of $NFSR_i$ after execution of nineteen clock cycles.

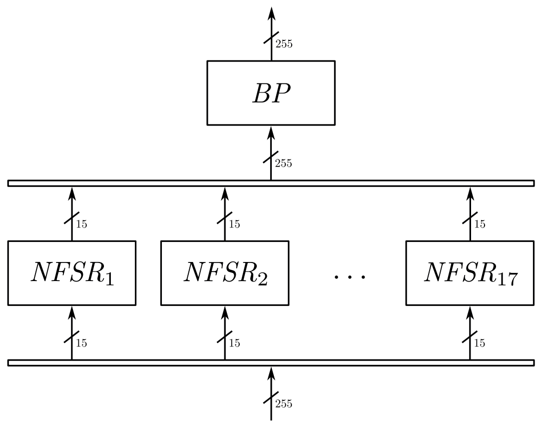

Figure 3. Block diagram of the *NPB* transformation

The general structure of a k-bit feedback shift register is shown in Fig. 4. It consists of k binary storage elements, called *stages*, and a feedback function $f(s_1, s_2, \ldots, s_k)$. For each $i = 1, 2, \ldots, k$, the state variable s_i represents the value of the i-th bit. At each clock cycle, the content of the register is shifted one bit left. The value of the k-th bit is updated according to the feedback function f. When the feedback function is nonlinear, this structure is called a *nonlinear feedback shift register*. The truth tables of seventeen nonlinear feedback functions used in the FSR-255 hash function can be found in Janicka-Lipska (no date).

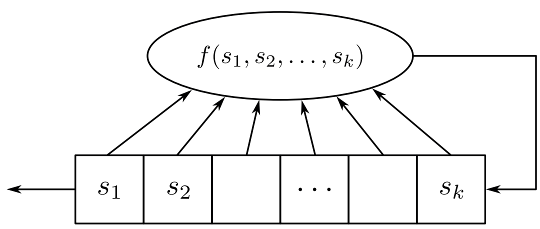

Figure 4. Block diagram of a feedback shift register

The bit permutation function $BP : \{0,1\}^{255} \to \{0,1\}^{255}$ is defined as follows:

$$BP(w) = BP(w_1, w_2, \ldots, w_{255}) = w_{\pi(1)} \parallel w_{\pi(2)} \parallel \ldots \parallel w_{\pi(255)},$$

where $w = w_1 \parallel w_2 \parallel \ldots \parallel w_{255}$, $w_i \in \{0,1\}$, and π = {175, 10, 174, 209, 155, 182, 244, 193, 219, 26, 22, 107, 214, 236, 173, 14, 215, 44, 97, 153, 28, 8, 185, 254, 204, 25, 164, 37, 195, 255, 231, 154, 158, 159, 29, 19, 243, 151, 6, 90, 200, 4, 252, 206, 94, 118, 95, 42, 191, 218, 116, 180, 110, 65, 93, 17, 190, 117, 136, 144, 87, 140, 88, 162, 40, 123, 115, 71, 52, 226, 132, 147, 53, 250, 248, 32, 80, 16, 64, 225, 15, 166, 232, 86, 251, 160, 83, 187, 181, 101, 129, 130, 201, 33, 249, 176, 125, 109, 146, 30, 41, 138, 76, 34, 27, 127, 85, 78, 3, 47, 106, 228, 213, 48, 61, 75, 178, 230, 57, 72, 23, 111, 1, 100, 20, 247, 55, 212, 13, 227, 12, 81, 145, 62, 188, 39, 246, 133, 245, 38, 135, 235, 2, 221, 241, 56, 5, 237, 170, 224, 233, 184, 202, 114, 203, 70, 124, 113, 239, 103, 156, 196, 177, 148, 59, 122, 210, 220, 168, 242, 92, 36, 194, 51, 84, 186, 63, 99, 238, 46, 102, 171, 18, 142, 149, 89, 137, 161, 112, 31, 119, 120, 50, 98, 253, 108, 134, 192, 217, 79, 167, 43, 234, 143, 199, 163, 179, 69, 45, 198, 152, 9, 157, 121, 58, 77, 11, 189, 105, 49, 150, 183, 223, 21, 91, 66, 165, 74, 216, 96, 141, 169, 229, 104, 54, 131, 7, 240, 126, 82, 67, 68, 207, 222, 24, 35, 73, 197, 139, 128, 208, 60, 172, 205, 211} is a predefined 255-element permutation.

The final transformation $G : \{0,1\}^{255} \rightarrow \{0,1\}^r$ returns as the hash result the r leftmost bits of $LPB(H_q)$, where LPB, called the *linear permutation block*, is in fact the bit permutation function for which the permutation π' is defined by succeeding states of an 8-bit shift register with linear feedback function $f(s_1, s_2, \ldots, s_8) = s_1 \oplus s_3 \oplus s_4 \oplus s_5$, and initialized by the 00000001 binary value, i.e., $\pi' = \{$1, 2, 4, 8, 17, 35, 71, 142, 28, 56, 113, 226, 196, 137, 18, 37, 75, 151, 46, 92, 184, 112, 224, 192, 129, 3, 6, 12, 25, 50, 100, 201, 146, 36, 73, 147, 38, 77, 155, 55, 110, 220, 185, 114, 228, 200, 144, 32, 65, 130, 5, 10, 21, 43, 86, 173, 91, 182, 109, 218, 181, 107, 214, 172, 89, 178, 101, 203, 150, 44, 88, 176, 97, 195, 135, 15, 31, 62, 125, 251, 246, 237, 219, 183, 111, 222, 189, 122, 245, 235, 215, 174, 93, 186, 116, 232, 209, 162, 68, 136, 16, 33, 67, 134, 13, 27, 54, 108, 216, 177, 99, 199, 143, 30, 60, 121, 243, 231, 206, 156, 57, 115, 230, 204, 152, 49, 98, 197, 139, 22, 45, 90, 180, 105, 210, 164, 72, 145, 34, 69, 138, 20, 41, 82, 165, 74, 149, 42, 84, 169, 83, 167, 78, 157, 59, 119, 238, 221, 187, 118, 236, 217, 179, 103, 207, 158, 61, 123, 247, 239, 223, 191, 126, 253, 250, 244, 233, 211, 166, 76, 153, 51, 102, 205, 154, 53, 106, 212, 168, 81, 163, 70, 140, 24, 48, 96, 193, 131, 7, 14, 29, 58, 117, 234, 213, 170, 85, 171, 87, 175, 95, 190, 124, 249, 242, 229, 202, 148, 40, 80, 161, 66, 132, 9, 19, 39, 79, 159, 63, 127, 255, 254, 252, 248, 240, 225, 194, 133, 11, 23, 47, 94, 188, 120, 241, 227, 198, 141, 26, 52, 104, 208, 160, 64, 128$\}$.

3. Properties of the nonlinear processing block

The nonlinear processing block *NPB* introduces nonlinearity into the FSR-255 hash function. *NPB* is the composition of two transformations: the bit permutation function *BP* and the layer of seventeen nonlinear feedback shift registers.

LEMMA 1 *The nonlinear processing block NPB is a bijective transformation.*

Proof It is easy to see that the bit permutation function *BP* is bijective as π is a 255-element permutation.

Every nonlinear feedback shift register in the FSR-255 hash function uses a feedback function f_i such that the register generates all the 2^{15} possible states when it is clocked 2^{15} times (Janicka-Lipska, no date). In the *NPB* transformation every register is clocked nineteen times. Thus, for a given 15-bit initial value it stops for sure with a 15-bit state, which is different from the initial value and is unique, i.e., there are no two different initial values for which the register state value after nineteen clock cycles would be the same. Hence, the register working in that way is a 15-bit bijective mapping. Concatenating seventeen such mappings yields a 255-bit bijective mapping.

Composition of two bijective mappings of the same size is a bijective mapping. □

Corollary 1 *Since NPB is bijective, we conclude that NPB^{-1} exists.*

The NPB^{-1} transformation can be determined by composing BP^{-1} and inversion of the layer of seventeen nonlinear feedback shift registers. BP^{-1} can be easily calculated by calculating π^{-1}.

Inversion of the nonlinear feedback shift registers layer is easy if we notice that in one clock cycle of a single *NFSR* only one bit of the state is updated by the feedback function and only one bit is lost by the shift of the stage. So, if we want to obtain the previous state of a single *NFSR*, we have to predict only the previous value of s_1, which can be either 0 or 1, because of uniqueness of the state values. The remaining stages $s_2, s_3, \ldots, s_{15}$ are known. The prediction can be done just by taking one of two possible values for s_1 and checking if we get the current state from the predicted one after the clock cycle of *NFSR*. Repeating this procedure nineteen times for all seventeen shift registers gives us the inversion of the nonlinear feedback shift registers layer.

Thus, the value of NPB^{-1} for a given input can be computed with negligible complexity.

4. The second-preimage attack

To show that a hash function h is not second-preimage resistant we have to find for a given m a distinct m' such that $h(m) = h(m')$. It does not matter how much m' differs from m.

Let m be a given message of length $n \geq 255$ bits and $x = x_1 \parallel x_2 \parallel \ldots \parallel x_q$ be the corresponding extended message. If we were able to find $x'_1 \neq x_1$ such that $F(x'_1, H_0) = F(x_1, H_0)$, i.e., $H'_1 = H_1$, then, by taking $x'_2 = x_2, \ldots, x'_q = x_q$ we would get $H'_q = H_q$ and finally $h(m') = h(m)$. Unfortunately, *NPB* is a bijective transformation, which implies that for a given H_0 the compression function F is bijective, too, and there is no $x'_1 \neq x_1$ for which $F(x'_1, H_0) = F(x_1, H_0)$, i.e., $H'_1 = H_1$. So, the compression function of the FSR-255 hash function is collision-free.

Assume that the message m has the length of $n \geq 2 \cdot 255$ bits. Then, for given x_1, x_2 we can try to find a pair x'_1, x'_2, where $x'_1 \neq x_1$ and $x'_2 \neq x_2$, such that $H'_2 = H_2$. From the FSR-255 specification we have

$$F(x'_2, H'_1) = F(x_2, H_1), \tag{3}$$

where $H'_1 = F(x'_1, H_0)$ and $H_1 = F(x_1, H_0)$.

Because x_1 and x_2 are given, the right hand side of equation (3), equal to H_2, is fixed. Additionally, we can fix x'_1 by choosing its value arbitrarily, it must only be different from x_1. This implies that H'_1 is fixed, too. Then, using equation (1) we can calculate the value of x'_2 as follows:

$$x'_2 = NPB^{-1}(H_2 \oplus H'_1) \oplus H'_1 \oplus y_1. \tag{4}$$

The inversion NPB^{-1} in equation (4) exists and can be efficiently calculated (see Section 3). If $n \geq 2 \cdot 255$, we can take $x'_3 = x_3, \ldots, x'_q = x_q$. Then, we get $H'_q = H_q$ and finally $h(m') = h(m)$. The message m' can be retrieved from x' by removing padding bits.

Example 1 Let us assume that the given message is $m = x_1 \parallel x_2 = 0^{255} \parallel 0^{255}$. Then, we have padding $x_3 = 1 \parallel 0^{254}$, $x_4 = x_5 = x_6 = x_7 = 0^{255}$, $x_8 = 0^{246} \parallel 111111110$ and the FSR-255 hash value $h(m) =$ `0x7443` $\parallel$ `0x5477` $\parallel$ `0x2a0e` $\parallel$ `0x6e19` $\parallel$ `0x4bb4` $\parallel$ `0x6f96` $\parallel$ `0x5e2a` $\parallel$ `0x61b8` $\parallel$ `0x0297` $\parallel$ `0x51cc` $\parallel$ `0x2c86` $\parallel$ `0x24f5` $\parallel$ `0x54d4` $\parallel$ `0x2004` $\parallel$ `0x73f1` $\parallel$ `0x48d6` $\parallel$ `0x00d1` (255-bit value written in the same convention as H_0 was given in Section 2).

Let $x'_1 = 0^{127} \parallel 1 \parallel 0^{127}$, then from equations (4) and (3) we get $x'_2 =$ `0x0000` $\parallel$ `0x0000` $\parallel$ `0x0000` $\parallel$ `0x0000` $\parallel$ `0x0000` $\parallel$ `0x0000` $\parallel$ `0x0042` $\parallel$ `0x0000` $\parallel$ `0x6bd4` $\parallel$ `0x0024` $\parallel$ `0x0000` $\parallel$ `0x0000` $\parallel$ `0x0000` $\parallel$ `0x0000` $\parallel$ `0x0008` $\parallel$ `0x0000` $\parallel$ `0x0000`, and for $m' = x'_1 \parallel x'_2$ we have that $h(m') = h(m)$, which means that we have found the second-preimage m' for the given message m (and we have found the collision as well).

5. The preimage attack

To show that a hash function h is not preimage resistant we have to provide a method of finding a message m for a given digest y such that $h(m) = y$. It does not matter what m looks like (it may have no sense) and how long it is.

Let a digest y, having a length of 255 bits be given. Then, as the G transformation is invertible, we have

$$H_q = G^{-1}(y). \tag{5}$$

If we choose arbitrary values for $x_1, x_2, \ldots, x_{q-1}$, then we can determine the value of H_{q-1}. From equation (1) we have

$$H_q = NPB(x_q \oplus y_{q-1} \oplus H_{q-1}) \oplus H_{q-1} \tag{6}$$

and so

$$x_q = NPB^{-1}(H_q \oplus H_{q-1}) \oplus H_{q-1} \oplus y_{q-1}. \tag{7}$$

The extended message $x = x_1 \parallel x_2 \parallel \ldots \parallel x_q$ gives the value of H_q, for which $G(H_q) = h(m)$. Unfortunately, the x obtained in that way has inappropriate format and m cannot be retrieved from it by removing the padding bits.

Let us assume that in the last block x_q we have the last 222 bits of the message m. Then, padding has to consist of 1 bit equal to one and 32 bits equal to $b = n \bmod 2^{32}$. By changing randomly the values of $x_1, x_2, \ldots, x_{q-1}$ we will get from equation (7) with probability of about 2^{-33} a proper value for x_q, which is of the form $m_q \parallel 1 \parallel b$, where $m_q \in \{0,1\}^{222}$ is the value of the last 222 bits of the sought message m for the given digest y. The whole message m, such that $h(m) = y$, is given by $m = x_1 \parallel x_2 \parallel \ldots \parallel x_{q-1} \parallel m_q$ and has the length of $n = 255 \cdot (q-1) + 222$ bits.

As the length of the extended message x is a multiple of 2040, the proper value for b is of the form $k \cdot 2040 - 33$, where $1 \leq k \leq 2105376$ and padding is of the form given above. So, by changing randomly the values of $x_1, x_2, \ldots, x_{q-1}$ we will get from equation (7) with probability equal about 2^{-11} a value for x_q, which has the proper value of b and thus with probability equal about 2^{-12} has the proper format of the entire padding given above.

Of course, other paddings, like $1 \parallel 0 \parallel b$ or $1 \parallel 0^2 \parallel b$ or even $1 \parallel 0^{222} \parallel b$ can also be taken into account. The probability that we get a proper padding in x_q when we change randomly the values of $x_1, x_2, \ldots, x_{q-1}$ is equal to

$$2^{-11} \sum_{i=1}^{223} \frac{1}{2^i} \approx 2^{-11}. \tag{8}$$

Hence, for x_q we should care only about the proper value of b, because the other values will be proper with probability close to 1.

The length of the message can be matched using fixed points, i.e., x_i such that $F(H_{i-1}, x_i) = H_{i-1}$. It is easy to see from equation (1) that if x_i is a fixed point then $NPB(x_i \oplus y_{i-1} \oplus H_{i-1}) = 0^{255}$. Thus, the fixed point is given by

$$x_i = NPB^{-1}(0^{255}) \oplus H_{i-1} \oplus y_{i-1}. \tag{9}$$

From equation (9) we can calculate the values of $x_i, x_{i+1}, \ldots, x_{i+7}$ and then insert the block $x_i \parallel x_{i+1} \parallel \ldots \parallel x_{i+7}$ to the extended message x without any change of the digest. In that way we can increase the length of the message by 2040 bits and the given digest y still remains the same. This procedure can be repeated several times until we get a message of the desired length, given by the proper value of b.

EXAMPLE 2 Let us assume that the FSR-255 hash value $h(m) = 0^{255}$. Then, for this given hash value, the extended message $x = x_1 \parallel x_2 \parallel \ldots \parallel x_8$, found by the algorithm described above, is equal to
x_1 = `0x1248` ‖ `0x76ec` ‖ `0x7904` ‖ `0x2896` ‖ `0x2ce3` ‖ `0x6420` ‖ `0x000d` ‖ `0x654f` ‖ `0x4f83` ‖ `0x10ab` ‖ `0x506f` ‖ `0x4241` ‖ `0x7432` ‖ `0x6d80` ‖ `0x3de0` ‖ `0x0e0e` ‖ `0x3794`,
x_2 = `0x7db8` ‖ `0x47bc` ‖ `0x23e9` ‖ `0x5dc3` ‖ `0x41a7` ‖ `0x3b72` ‖ `0x15a5` ‖ `0x3042` ‖ `0x6269` ‖ `0x459d` ‖ `0x2443` ‖ `0x0460` ‖ `0x6c17` ‖ `0x413d` ‖ `0x4a88` ‖ `0x4ab8` ‖ `0x775d`,
x_3 = `0x1c78` ‖ `0x0892` ‖ `0x22d6` ‖ `0x2973` ‖ `0x1b39` ‖ `0x0d39` ‖ `0x6ab6` ‖

0x1dac ‖ 0x1ba2 ‖ 0x68ee ‖ 0x15dc ‖ 0x2956 ‖ 0x7609 ‖ 0x695c ‖ 0x4641 ‖ 0x58df ‖ 0x19f0,
x_4 = 0x6a64 ‖ 0x7460 ‖ 0x22c5 ‖ 0x5e38 ‖ 0x4690 ‖ 0x165d ‖ 0x1d47 ‖ 0x4772 ‖ 0x043e ‖ 0x1239 ‖ 0x26c5 ‖ 0x5b85 ‖ 0x7fa8 ‖ 0x23da ‖ 0x7e8d ‖ 0x31f9 ‖ 0x1f1a,
x_5 = 0x7d05 ‖ 0x6baf ‖ 0x7ac4 ‖ 0x114c ‖ 0x634a ‖ 0x1feb ‖ 0x6a38 ‖ 0x22df ‖ 0x3dc7 ‖ 0x5d94 ‖ 0x3df4 ‖ 0x54a8 ‖ 0x4f32 ‖ 0x4772 ‖ 0x4d01 ‖ 0x676b ‖ 0x7f74,
x_6 = 0x0473 ‖ 0x6253 ‖ 0x5f44 ‖ 0x45bb ‖ 0x5117 ‖ 0x6770 ‖ 0x054f ‖ 0x6c85 ‖ 0x5658 ‖ 0x516e ‖ 0x5447 ‖ 0x045a ‖ 0x0c3b ‖ 0x2138 ‖ 0x5e39 ‖ 0x1202 ‖ 0x7172,
x_7 = 0x63e4 ‖ 0x62cc ‖ 0x651c ‖ 0x28f4 ‖ 0x0a32 ‖ 0x58f9 ‖ 0x53b8 ‖ 0x1060 ‖ 0x00e1 ‖ 0x49a1 ‖ 0x42dc ‖ 0x4955 ‖ 0x580b ‖ 0x675c ‖ 0x170a ‖ 0x2554 ‖ 0x4192,
x_8 = 0x7927 ‖ 0x43ab ‖ 0x6a72 ‖ 0x249a ‖ 0x79d8 ‖ 0x0509 ‖ 0x290c ‖ 0x1caf ‖ 0x1422 ‖ 0x4eac ‖ 0x6587 ‖ 0x0aef ‖ 0x7182 ‖ 0x7bf8 ‖ 0x234c ‖ 0x0000 ‖ 0x07d7.

The last 33 bits of x_8 are padding bits. Thus, preimage $m = x_1 \parallel x_2 \parallel \ldots \parallel x_7 \parallel m_8$, where $x_8 = m_8 \parallel 1 \parallel b$ and $b = 000\ldots0011111010111$, is a 32-bit representation of the length of the message m.

6. Summary

In this paper we have shown that FSR-255 hash function is not second-preimage resistant (which implies that it is not collision resistant either) and finding second-preimages for FSR-255 is an easy task (of negligible complexity). The presented second-preimage attack shows that a collision-free compression function does not guarantee the collision resistance of the whole iterated hash function.

We have also shown a preimage attack on the FSR-255 hash function with complexity of about 2^{11}.

As a consequence, when designing iterated hash algorithms, the compression function should not be easily invertible for a given output and intermediate chaining variable.

Due to the security flaws, shown in this paper, the FSR-255 hash function should not be used in cryptographic applications.

Both attacks presented in this paper have been implemented and work in practice.

References

Gajewski, T., Janicka-Lipska, I., Stokłosa, J. (2003) The FSR-255 family of hash functions with a variable length of hash result. In: J. Sołdek, L. Drobiazgiewicz, eds., *Artificial Intelligence and Security in Computing Systems.* Kluwer Academic Publishers, Boston, 239-248.

Janicka-Lipska, I. (no date) Truth tables of some nonlinear feedback functions for 15-bit feedback shift registers with the maximum length of state cycles. Website http://www.sk-kari.put.poznan.pl/Janicka/functions/

Janicka-Lipska, I., Stokłosa, J. (2001) FSR-255 cryptographic hash function. In: W. Burakowski, A. Wieczorek, eds. *NATO Regional Conference on Military Communications and Information Systems*, Zegrze 2001, vol. I, 321-324.

Kontak, M., Szmidt, J. (2013) The FSR-255 Hash Function Is Not Second-Preimage Resistant. In: M. Amanowicz, ed., *Military Communications and Information Technology: Recent Advances in Selected Areas.* Military University of Technology, Warsaw, 229-235.

Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.(1996) *Handbook of Applied Cryptography.* CRC Press, Boca Raton, FL.

Stokłosa, J. (1995) Integrity of data: FSR-hash. In: Z. Bubnicki, ed., *Proceedings of the 12th International Conference on Systems Science.* Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, **III**, 120–125.

Stokłosa, J. (1996) O pewnej funkcji skrótu. (in Polish) In: A. Wieczorek, L. Sufa, eds., *V Krajowa Konferencja Naukowa KNSŁ–96 Systemy łączności i informatyki na potrzeby obrony i bezpieczeństwa RP. Wyższa Szkoła Oficerska Wojsk Łączności, Zegrze,* **III**, *51–56.*