ARCHIVES OF ELECTRICAL ENGINEERING VOL. 68(3), pp. 595–609 (2019)

DOI 10.24425/aee.2019.129344

Spinal code based on lightweight hash function

LINA WANG, XINRAN LI

School of Computer and Communication Engineering
University of Science and Technology Beijing (USTB)
100083, Beijing, P.R. China
e-mail: wln_ustb@126.com

(Received: 15.12.2018, revised: 09.04.2019)

Abstract: A spinal code is the type of rateless code, which has been proved to be capacity-achieving over both a binary symmetric channel (BSC) and an additive white Gaussian noise (AWGN) channel. Rateless spinal codes employ a hash function as a coding kernel to generate infinite pseudo-random symbols. A good hash function can improve the performance of spinal codes. In this paper, a lightweight hash function based on sponge structure is designed. A permutation function of registers is a nonlinear function. Feedback shift registers are used to improve randomness and reduce bit error rate (BER). At the same time, a pseudo-random number generator adopts a layered and piecewise combination mode, which further encrypts signals via the layered structure, reduces the correlation between input and output values, and generates the piecewise random numbers to compensate the shortcoming of the mixed linear congruence output with fixed length. Simulation results show that the designed spinal code with the lightweight hash function outperforms the original spinal code in aspects of the BER, encoding time and randomness.

Key words: spinal codes, lightweight hash function, variable length output, layered pseudo-random number

1. Introduction

A spinal code is a rateless encoding scheme proposed in 2012 that can achieve near-capacity limited transmission over both binary symmetric channel (BSC) and additive Gaussian white noise (AWGN) channel [1]. The spinal code introduces a hash function into the coding structure. Compared with the convolutional encoders, the hash function has a nonlinear pseudo-random structure, and the probability of a hash collision is extremely small. At the same time, using the hash function and the random symbol generator can generate different coding symbols, achieving rateless transmission. In contrast to the traditional graphs or algebraic coding methods, spinal

codes can adapt to time-varying dynamic channels without feedback, automatically adjust the encoding rate [2, 3].

Lots of works related to the applications of the spinal codes have been conducted in different communication environments such as a rapidly time-varying channel, underwater environments and so on [4, 5]. On this basis, some scholars have studied the factors affecting coding performance. Balakrishnan studied the influence of coded symbol sequences on coding performance. It was proved that a non-sequential coded symbol sequence can improve the performance of the spinal codes [6]. Shun studied the spinal code in a diversity receiving system, and proposed a two-way code stream coding method based on a double hash function [7]. In addition, some other scholars have studied the decoding algorithms of the spinal code. Qu Xiaoxu considered the bundle search and cyclic redundancy check decoding as the whole joint decoding, and proposed the CRC-assisted multivariate backtracking decoding algorithm [8]. Ying Li improved the symbolic performance of the decoding structure using a sliding window [9].

Although the spinal code can provide an excellent rate, there are still some problems with spinal code applications. First, as the number of coding blocks increases, using maximum likelihood (ML) decoding leads to an exponential increase in decoding cost calculations [10]. Secondly, it is difficult to achieve error-free transmission because most of the blocks appearing in the last part of the spinal code have a high bit error rate (BER) [11]. Third, the hash function always achieves a low collision rate at the expense of the encoded information length [12].

In this paper, a lightweight hash function is designed and used to construct the spinal code. Compared with the existing hash functions, the designed lightweight hash function uses a variable output information length to accommodate different channels. In the first pass, the output length of the hash function is the shortest during all transmissions. If the received codes cannot be correctly decoded, the output value of the hash function is increased until it can be correctly decoded. Furthermore, a layered pseudo-random number algorithm is proposed to solve the security of the generated random number in encryption and decryption. The pseudo-random number generator is based on the mixed linear congruence. It is easily implemented by combining a layered encryption principle with the piecewise generation of random numbers. Therefore, the reliability can be improved and the shortcoming of mixed linear congruence output with fixed length can be compensated.

The remainder of this paper is organized as follows: Section 2 presents the encoding structure and decoding algorithm of the rateless spinal code. The designed lightweight hash function and the layered pseudo-random number algorithm are described in Section 3 and Section 4, respectively. The encoding parameters are discussed in Section 5. Section 6 presents the experimental results and analyses. Finally, the conclusions and the future work are drawn in Section 7.

2. Rateless spinal codes

2.1. Encoding structure

The encoding mechanism of spinal codes uses a hash function to achieve random encoding of information. The encoding process of the spinal code is depicted in Fig. 1.

In the first step, an n-bit coded block information sequence M is equally divided into n/k information blocks, each of which consists of k bits length, that is $M = \overline{m}_1\, \overline{m}_2 \ldots\, \overline{m}_{nk}$.

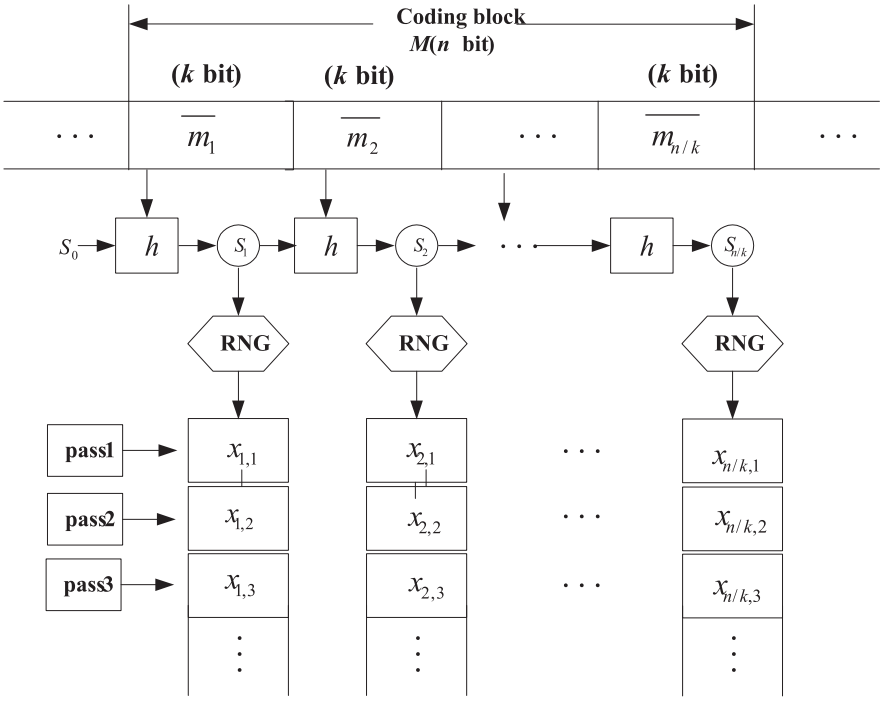

Fig. 1. The encoder of spinal codes

In the second step, the hash function has two input values, each information block m_i and v-bit state information generated by the hash function. They are commonly referred to as spinal values, denoted as s_i. s_i and the i-th input information block m_i (k-bit information) are put into the hash function to generate the next spinal value s_{i+1}, and s_{i+1} as well as the i+1-th input information block m_{i+1}, are put into the hash function to generate the next s_{i+2}. Then, a string of s_i can be obtained via the hash function. The mathematical formula for s_i is given in (1) [13].

$$S_i = H(\overline{m}_i,\ S_{i-1}), \qquad S_0 = 0^v. \tag{1}$$

In the third step, n/k state values can be obtained after encoding. Using s_i as the seed of a random number generator (RNG), a bit operation is performed by using a pseudo random number generator (PRNG), as shown in (2). In the formula, the two inputs of the RNG are the state value of the node and the parameter N. Here N is used to distinguish the pass of the encoding symbol of the RNG [14].

$$\text{RNG} : \{0,1\}^v \times N \ \rightarrow\ \{0,1\}^c. \tag{2}$$

The RNG can generate multiple batches of pseudo random sequences, each batch length is c-bits [15]. The same batch of encoded symbols generated by all status values can make up a transmission channel.

2.2. Decoding algorithm

Maximum likelihood (ML) decoding of the spinal codes is equivalent to searching all paths in the code tree. The path closest to the received information is selected to transmit symbols [16]. The number of all possible decoding paths can be represented by the number of nodes in the last layer of the decoding tree, and the number of nodes in the last layer can be expressed as 2^n (n is encoded information length).

Near-ML spinal decoding starts at a certain level of the decoding tree (set to level d) [17], retaining only B nodes of the lowest cost path in that level node. The algorithm is extended at each subsequent level, and only the decoding paths of B^*2^k (k is information block length) child nodes are calculated during the expansion process. In this way, the overhead of the node B is maintained. In the last level, only one path with the lowest cost can be used as the decoding result [18]. As shown in Fig. 2, starting from the depth 1 (d is set to 2), only B nodes are saved per layer. Shadow-filled nodes have the lowest path, so they are saved and other nodes are discarded. In the depth n/k (n/k is the number of an information block), the lowest cost path can be used as the decoding result. Therefore, the near-ML decoding algorithm is used in this paper.

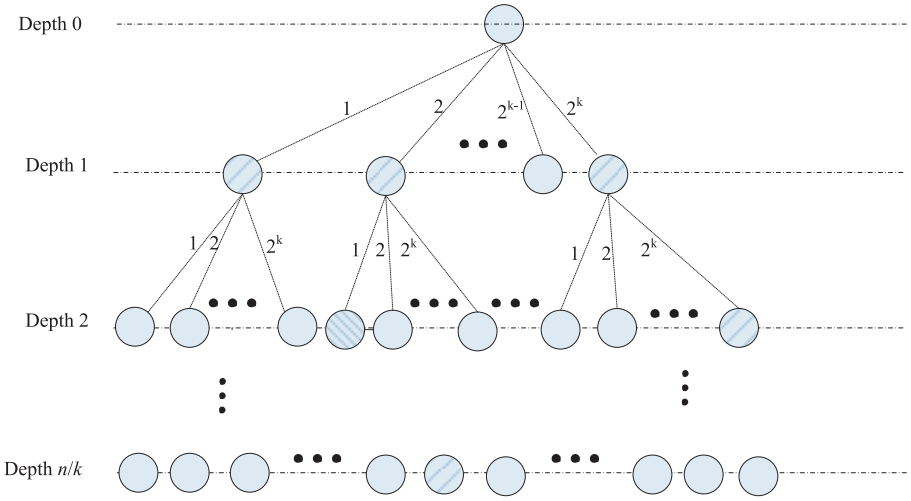

Fig. 2. The near-ML spinal decoding algorithm

3. Designed lightweight hash function

Since a hash function has high security, strong randomness and the highest symbol recovery, it also has increased redundancy and coding complexity. To solve this problem, a lightweight hash function based on a sponge construction is designed. The sponge structure consists of three distinct phases as follows:

Initialization phase. Padding bits are added to the information so that the length of the entire information is a multiple of the bitrate r (the rate or size of information block), this makes the information bits suitable to be divided into r-bit blocks (here, the rate r is set to 4 bits).

Absorbing phase. From the r-bit information block and the first r bits or last r bits of initial state XOR get the state bits, the state bits are calculated by the permutation function P and are repeated until all the information blocks $m_1, m_2, \ldots, m_k$ are absorbed.

Squeezing phase. The b-bit sponge state is applied to the permutation function P, and each state bit is input to obtain the r-bit output value. The calculation is repeated until the required n-bit output lengths are squeezed out, so the output values $h_1, h_2, \ldots, h_n$ can be obtained. The hash function is the concatenation of $h_1, h_2, \ldots, h_n$, i.e. $H = h_1 \| h_2 \| \ldots \| h_n$ [20]. The sponge structure is shown in Fig. 3 [21].

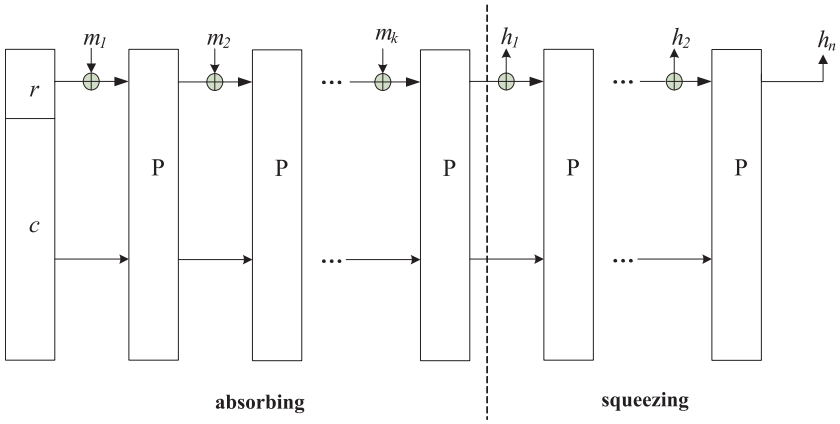

Fig. 3. Sponge structure

The lightweight hash function updates the sponge state using shift registers and a displacement function. The sponge state S can be expressed as $S \leftarrow (S_0, S_1, \ldots, S_{129})$, and the lightweight hash function uses four registers (P_r, Q_r, M_r, N_r) of size 32 or 33. The four register correspond to different bits of the sponge state S. And then, $P_0, P_1, \ldots, P_{31}$ are the bits on the P_r register. $Q_0, Q_1, \ldots, Q_3$ are the bits on the Q_r register. $M_0, M_1, \ldots, M_{31}$ are the bits on the M_r register. $N_0, N_1, \ldots, N_{31}$ are the bits on the N_r register. They are given from (3) to (6).

$$P_r : (P_0, P_1, \ldots, P_{31}) \leftarrow (S_0, S_1, \ldots, S_{31}), \tag{3}$$

$$Q_r : (Q_0, Q_1, \ldots, Q_{32}) \leftarrow (S_{32}, S_{33}, \ldots, S_{64}), \tag{4}$$

$$M_r : (M_0, M_1, \ldots, M_{31}) \leftarrow (S_{65}, S_{66}, \ldots, S_{96}), \tag{5}$$

$$N_r : (N_0, N_1, \ldots, N_{32}) \leftarrow (S_{96}, S_{97}, \ldots, S_{129}). \tag{6}$$

In each round of calculation, the displacement of the sponge state S update is as follows: the process of using the permutation function for the sponge state is $P(S) = P\,(P_0, P_1, \ldots, P_{31}, Q_0, Q_1, \ldots, Q_{32}, M_0, M_1, \ldots, M_{31}, N_0, N_1, \ldots, N_{32})$. P_t, Q_t, M_t, N_t are temporary storage spaces that store the intermediate values of the corresponding registers. L_o is the intermediate output value. The permutation function of the four registers is given as follows:

$$P_t = P_0 P_5 \oplus P_0 N_{23} \oplus P_5 Q_{10} \oplus Q_{10} N_{23} \oplus Q_{10} P_{23} \oplus 1, \tag{7}$$

$$Q_t = Q_{12}P_{21} \oplus Q_{12}Q_{17} \oplus Q_0P_{21} \oplus Q_0Q_{17} \oplus Q_{12}Q_{32}, \tag{8}$$

$$M_t = M_0N_6 \oplus M_0M_{26} \oplus N_6Q_{12} \oplus Q_{12}M_{26} \oplus 1, \tag{9}$$

$$N_t = N_{23}M_{21} \oplus N_{23}Q_{17} \oplus N_0M_{21} \oplus N_0Q_{17} \oplus N_{23}N_{32}, \tag{10}$$

$$L_o = P_0Q_{10} \oplus M_{17}N_{25}, \tag{11}$$

$$P_{31} = P_t \oplus L_o, \tag{12}$$

$$Q_{32} = Q_t \oplus L_o, \tag{13}$$

$$M_{31} = M_t \oplus L_o, \tag{14}$$

$$N_{32} = N_t \oplus L_o. \tag{15}$$

The permutation function is shown in Fig. 4.

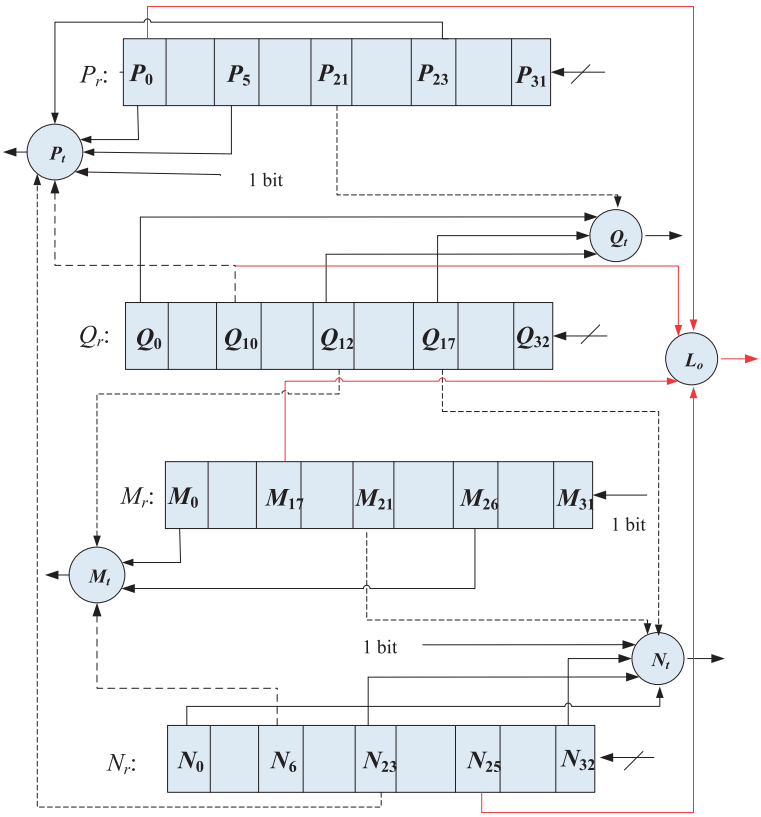

Fig. 4. The permutation function P

In the expression, $\oplus$ represents the XOR operator, and all above is a round of arithmetic operations. In the absorbing phase, the last bit of each register $(P_{31}, Q_{32}, M_{31}, N_{32})$ is XORed with each 4 bits of the incoming information. If it is the first or last bit of the information,

the information uses the permutation function for 61 rounds of status updates. Otherwise, the information uses the permutation function for 32 rounds of status updates. The operation is repeated until all information is absorbed. In the squeezing phase, the permutation function is used to update the registers. The last bit of four registers is output as the bit of the hash function and the operation is repeated until the required hash function length is squeezed, get the output value $H = h_1 \| h_2 \| \ldots \| h_n$.

4. Designed layered pseudo-random number algorithm

Generally, spinal codes use the mixed linear congruence to generate random numbers. This method is simple to operate, but easy to crack and the output length is fixed. On this basis, a layered structure is adopted to improve the security of encryption and decryption, and reduce the correlation between input and output. In other words, the output length of a hash function is different, and the corresponding output length of a random number is different.

The designed algorithm is divided into three layers, namely a shift layer, coding layer and conversion layer. The first layer converts the input symbols into ciphertext. The second layer involves embedding the key into the ciphertext obtained in the first layer, thus enhancing the security. Therefore, the message can only be recovered with the correct key and symbol. The last layer sends the generated pseudo-random number to the user with the key. The advantage of this designed algorithm is that the symbol length varies with the output length of the pass. It can match the lightweight hash function perfectly. Multilayer encryption and decryption can provide security, prevent ciphertext attacks and known pure text attacks. Moreover, it is easy to calculate and efficient. The specific encryption steps are as follows:

Shift Layer. The number of the moving bits of each input symbol is determined by each pass. Pass 1 shifts 6 bytes, and pass 2 shifts 12 bytes, and so on. The generated code m_1 is $6n$ (n is an integer) bits from the original code m_0, thus completing the first level of encryption. This method is simple in calculation and does not need to use a mapping method to remember the new symbols corresponding to each segment of symbols.

Encoding Layer. First the layer differentiates the encrypted m_1 obtained by the shift layer from each bit corresponding to the original symbol m_0 to m_2, and then divides each segment into 8-bit groups. If the length of each group of codes is exactly an integer multiple of 8 bits, the set of 8 bits will be converted into a group of 10 bits. Then, according to the mixed linear congruence formula, each segment is divided by M. If the last set is 4-bit length, it is divided by 4. The number of encryption depends on the message length.

Conversion Layer. This layer is responsible for converting decimal to its binary equivalent and sending the result.

The time complexity analysis is as follows: Unit Time 1 for each shift of symbol m_0 multiplies by N locations to be shifted, that is 1 (Unit Time 1)*N = O(N). For take-back operation, the unit time for each take-back operation multiplies by the length of symbol m_0. That is 1 (Unit Time 2)*L = O(L), where L represents the length of symbol m_0. For XOR operation, the unit time for each XOR operation multiplies by L, that is 1 (Unit Time 3)*L = O(L). For all binary-to-decimal and decimal-to-binary conversions, the unit time for each conversion (Unit Time 4) multiplies by L. There are two conversions, that is 1 (Unit Time 4)*2*L = O(L). Since

L is larger than N, the overall complexity of the algorithm is O(L). Consequently, the runtime complexity is O(L) for encryption and decryption algorithms. It is proved the algorithm has low complexity.

5. Design of encoding parameters

In this section, a series of simulations are conducted and results are discussed. Here, the feedback delay from the decoder to the sender over the AWGN channel is neglected. A random number generator uses layered pseudo-random number algorithms to generate random numbers. The decoder uses the near-ML spinal decoding algorithm. The hash function and random number generator used by the decoder are the same as those of the encoder.

5.1. Information bit block

According to the spinal encoding principle, each information block (k) and the intermediate symbol generated by the previous stage are the input of a hash function, so the value of k has an influence on the code output. According to (2), k affects the ratio of the information length to the code length. This ratio increases as the value of k increases. On the one hand, the value of k is expectedto be large enough in order to maximize the channel utilization as much as possible. On the other hand, the decoding complexity is exponentially related to k. So the value of k should not be too large.

Assume that 1000-bit information is transmitted at the source. The block length and frame length are set to 8 bits and 36 bits respectively. The length of the cyclic check code is 4 bits. Fig. 5 shows that the number of channel symbols for different SNR (signal-to-noise ratio) and k.

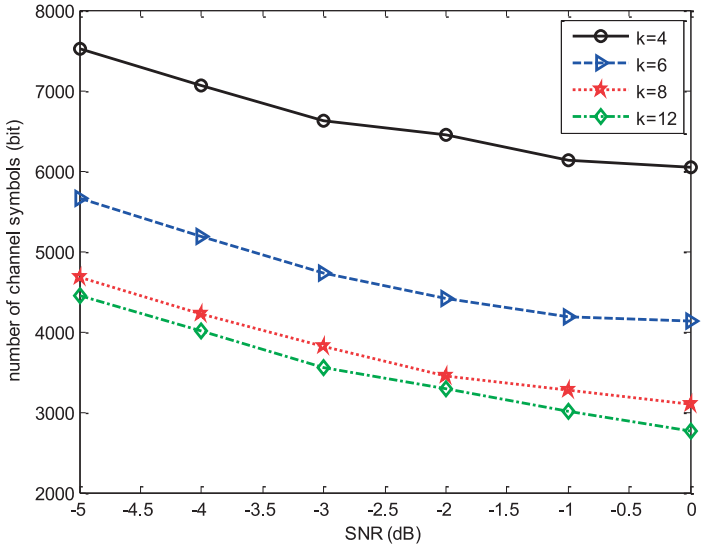

Fig. 5. The number of channel symbols for different SNR and k

The simulation results show that the different k has different effects on the spinal codes. The number of channel symbols is the sum of the transmission channel symbols required for correctly decoding all frames. For the same SNR, the larger k is, the fewer number of channel symbols is. The number of channel symbols required is very close when $k = 8$ and $k = 12$. Therefore, k is set to 8 in order to maximize channel utilization and reduce decoding complexity as much as possible.

5.2. Output length of the hash function

In this paper, the maximum value of the pass is set to 5. If it exceeds 5 times, the receiver can not be correctly decoded. This information is discarded in order to minimize the influence of error decoding information. The lightweight hash function uses a variable length output. In the squeezing phase, 4 bits are squeezed out every time. The output length L of the hash function is adjusted by controlling the length of the squeezed data. To select the shortest decoding length, L is set to 16 bits, 20 bits, ..., 64 bits, respectively. Assume that 1 000 000-bit information are transmitted at the source. And the simulation parameters are consistent with those of Section 5.1.

Fig. 6 shows the relationship between the output length and collision probability. When L is between 16 bits and 36 bits, the collision probability produced by the hash function decreases exponentially. When L is greater than 36 bits, the hash function also does not implement a collision-free state due to the channel noise. To ensure correct output, L is set to 36 bits over pass 1, and then L is set to 40 bits over pass 2. L is set in this way until the pass 5. The information will be discarded if the spinal code is still not correctly decoded when L is 52 bits. Then the next set of information starts to be transmitted.

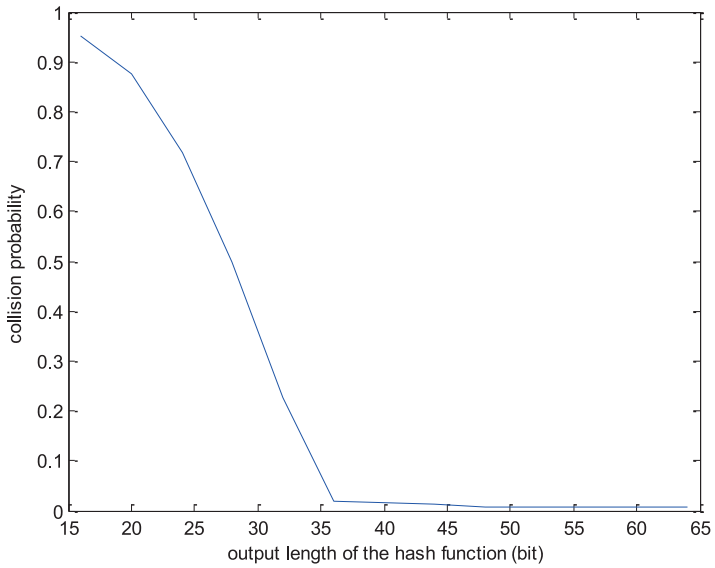

Fig. 6. Collision probability

5.3. Output length of RNG

The RNG can map arbitrary length input information to the fixed-length outputs. The spinal codes require the random number generator output to have independence and uniform distribution characteristics. To ensure independence and uniqueness, the output length of RNG is large enough. However, the output length of RNG should be as small as possible to maximize channel utilization. Assume that the spinal code could be correctly decoded under extremely bad conditions. The SNR is set to –5 dB and the 1 000 000-bit information is transmitted at the source. And the simulation parameters are consistent with those of Section 5.1.

Considering the effects on throughput and BER, the output length of RNG is set to 20 bits (Fig. 7).

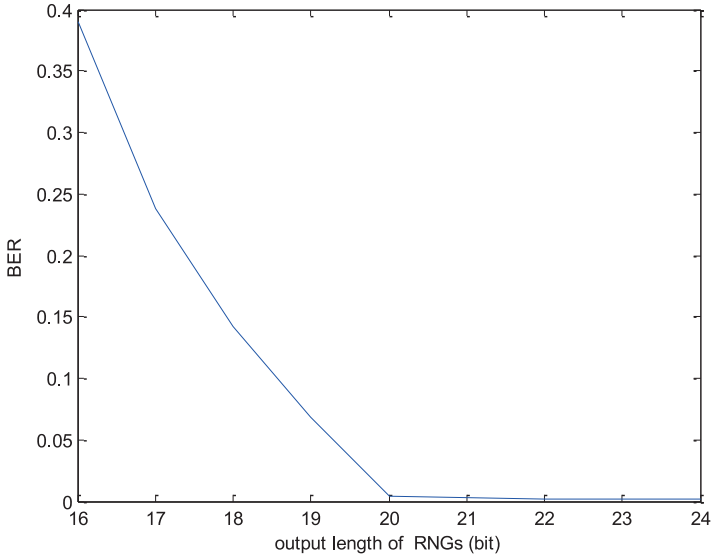

Fig. 7. The BER for different output lengths of RNG

6. Performance evaluation of spinal codes

In this section, the performance of spinal codes over the AGWN channel is evaluated through a serial of simulations.

6.1. Bit error rate performance

The encoder of spinal codes use different hash functions, namely a lightweight hash function, hash one function, one at a time function and MD5 function. The block length and frame length are set to 8 bits and 36 bits respectively. The lightweight hash function has a variable output length, and the output length of the number generator is 20 bits. Assume that 1 000 000-bit information is transmitted at the source.

Fig. 8 shows the BER performance of four different hash functions (namely lightweight hash, hash one, one at a time and MD5). The simulation result shows that the BER of one at a time function is significantly higher than other hash functions. In the case of a low SNR, the error rate of the lightweight hash function is significantly lower than other functions. When the SNR is set to 0, the BER of the lightweight hash function can reach 10^{-5}.

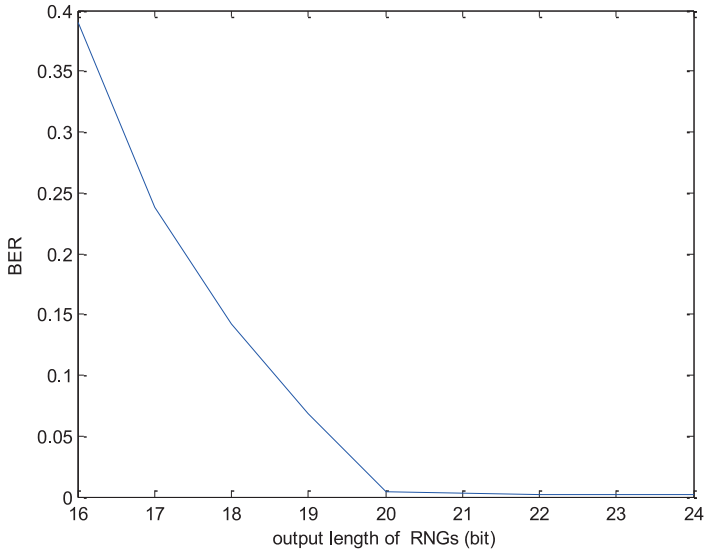

Fig. 8. The BER of different hash functions

6.2. Average encoding time

MD5 needs to go through 4 round operations. There are 16 steps in each round of loop operation. Each step is a nonlinear function operation.Totally about 64 steps are required. In the software implementation of MD5, multiple rounds of loop nesting are used. For computer systems, the nesting of large and multi-level loops is inefficient for the MD5 algorithm. The one at a time function can only get one output value in one operation, which needs six XORs, ten additions and five shift operations [17]. The one at a time function step is simple, requiring a total of three shift operations, four additions, and two XORs. Its disadvantages: a high bit error rate and low overall performance [14].

In the absorbing phase of the lightweight hash function, the sponge status is updated 32 times, and the intermediate information bit sponge status is updated 61 times during the absorbing process. In this process, the permutation function P ensures thorough mixing of the entire information bits. P is a nonlinear function operation and requires 9 steps in total. 4 32-bit shifts and 22 XORs are included. Each update process outputs four hash values. This saves a significant amount of time. Fig. 9 shows the average encoding time using four different hash functions (namely lightweight hash, hash one, one at a time and MD5).

From Fig. 9, it can be seen that the operation time of MD5 is the longest, the lightweight hash and one at a time are most efficient during the encoding process.

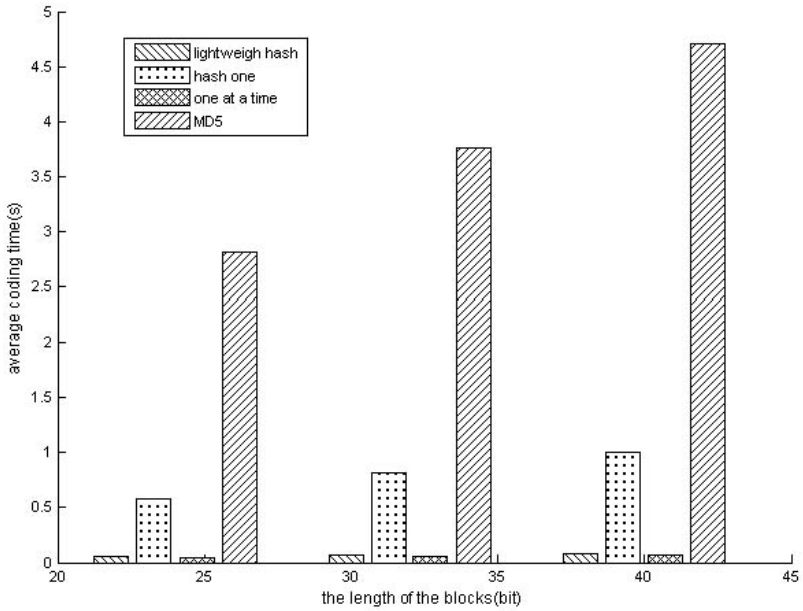

Fig. 9. Average encoding time of different hash functions

6.3. Random test

In this paper, the sequence randomness is tested using the statistical method, to judge whether it is a sequence of uniform pseudorandom numbers, which could satisfy the statistical characteristics of spinal codes. The mean, variance, uniformity distribution and correlation coefficient of the sequence are calculated as the judgment criteria.

For sequence S, the average value is given in (15), and the variance is given in (17) [22].

$$E(S) = \frac{1}{n}\sum_{i=1}^{n} s_i = \frac{M}{2}, \tag{16}$$

$$E(S^2) = \frac{1}{n}\sum_{i=1}^{n} s_i^2 = \frac{M^2}{2}, \tag{17}$$

$$Var(S) = E(S^2) - |E(S)|^2 = \frac{M^2}{12}. \tag{18}$$

For the uniformity test, divide $0 \sim M$ into m intervals. The propability of every interval is equal and should be M/n. Pearson's chi-squared test is used. When H_0 is true, the frequency of sample values in the i-th interval using n tests is close to the probability. The test formula can be expressed as (18) [23], where n_i represents the actual frequency value in the interval and u_i represents the theoretical value.

$$V = \sum_{i=1}^{m} \frac{(n_i - u_i)^2}{n_i}. \tag{19}$$

The independence test is the test of data correlation. If the correlation between the two random variables is smaller, the correlation coefficient ρ is closer to zero. On the contrary, the greater the correlation between the two random variables, the more their correlation coefficient ρ is close to one. The j order correlation coefficient is given in (19) [22].

$$\rho(j) = \frac{\frac{1}{n-j}\sum_{i=1}^{n}(x_i - \overline{x}_i)}{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x}_i)}. \tag{20}$$

The length of the random sequence is 1 600 bits, and the value is normalized to the interval 0 ~ 1. The simulation results are shown in Fig. 10.

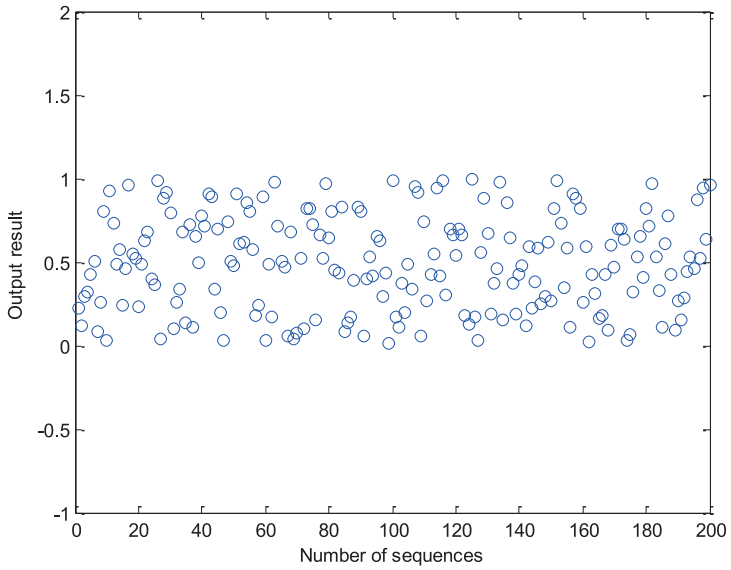

Fig. 10. Pseudo random number distribution

According to the generated random sequence, the values of each performance parameter can be obtained. The mean value is 0.5011 and the variance is 0.073. When the degree of freedom is 200, the significant level is 0.05. If Pearson's chi-squared test value is greater than 233.994, the sequence is non-uniform. However, the actual calculated value is 218.6. This means that the sequence is uniform. The first-order correlation coefficient and the second-order correlation coefficient are 0.0043 and 0.0102, respectively. The correlation is not large. Consequently, the sequence satisfies the randomness distribution.

Through analyses, we can see that the lightweight hash function has good performance in aspects of the BER, average encoding time and randomness.

7. Conclusions

In this paper, a lightweight hash function based spinal code is designed. The lightweight hash function uses a sponge construct with a permutation function. The sponge construction takes a finite state to receive input and output bits of arbitrary length information. Furthermore, a layered and piecewise pseudo-random number generator is designed. The piecewise structure of the pseudo-random number generator can compensate the disadvantage that the length of the output symbol of the mixed linear congruence method is fixed. The layered encryption algorithm is used to improve the security of a spinal code. The performance of the designed spinal code based on the lightweight hash function and layered pseudo-random number generator are evaluated through a series of simulations. The simulation results verify the effectiveness of the designed spinal code.

In this paper, the spinal coding algorithm is improved, which improves the performance of the spinal code to a certain extent. However, there is still a problem that the decoding complexity increases exponentially with the block length when using the truncated tree decoding algorithm. Therefore, in the future work, we need to study and improve its decoding algorithm, which can greatly reduce the complexity while guaranteeing certain performance, which may promote the development of spinal codes.

Acknowledgements

We gratefully acknowledge the anonymous reviewers who read drafts and made many helpful suggestions. This work is supported by the National Natural Science Foundation of China under Grant No. 61701020 and University of Science and Technology Beijing Project under Grant No. 04130017.

References

[1] Chen S., Zhang Z., Zhang L. *et al*., *Belief propagation with gradual edge removal for Raptor codes over AWGN channel*, Proceeding of the 24th International Symposium on the Personal Indoor and Mobile Radio Communications (PIMRC), London, UK, pp. 380–385 (2013).

[2] Erez U., Trott M., Wornell G., *Rateless coding for Gaussian channels*, IEEE Transactions on Information Theory, vol. 58, no. 2, pp. 530–547 (2012).

[3] Yang W., Li Y., Yu X. Sun Y., *Rateless Superposition Spinal coding scheme for half-duplex relay channel*, IEEE Transactions on Wireless Communications, vol. 15, no. 9, pp. 6259–6272 (2016).

[4] Chen P., Li Q., Li Q., Bai B., *Design and performance of spinal codes over fading channels*, High Mobility Wireless Communications (HMWC), International Workshop on High Mobility Wireless Communications, Beijing, China, pp. 140–145 (2014).

[5] Tai Y., Guilloud F., Laot C., Le Bidan R., Wang H., *Joint equalization and decoding scheme using modified spinal codes for underwater communications*, OCEANS 2016 MTS/IEEE Monterey, Monterey, CA, USA, pp. 1–6 (2016).

[6] Balakrishnan H., Iannucci P., Perry J. *et al*., *De-randomizing Shannon: the design and analysis of a capacity-achieving rateless codes*, eprint arXiv:1206.0418 (2012).

[7] Shun O., Koji I., *On dynamic cooperative diversity based on dual-spinal codes*, IEICE Technical Report Wideband System, vol. 113, pp. 181–186 (2014).

[8] QU Xiaoxu, Yang Liming, Miao Quanqiang, *CRC-assisted multivariate backtracking decoding algorithm for Spinal codes*, Computer Engineering, vol. 43, no. 12, pp. 120–123–129 (2017).

[9] Yang W., Li Y., Yu X., *Performance of spinal codes with sliding window decoding*, 2017 IEEE International Symposium on Information Theory (ISIT), Aachen, Germany, pp. 2203–2207 (2017).

[10] Yang W., Li Y., Yu X., Li J., *A low complexity sequential decoding algorithm for rateless spinal codes*, IEEE Communications Letters, vol. 19, no. 7, pp. 1105–1108 (2015).

[11] Dong D., Wu S., Jiang X., Jiao J., Zhang Q., *Towards high performance short polar codes: concatenated with the spinal codes*, 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, Canada, pp. 1–5 (2017).

[12] Tashiro H., Morishima Y., Oka I., Ata S., *PAPR control of OFDM signals using spinal codes*, 2016 International Symposium on Information Theory and Its Applications (ISITA), Monterey, CA, USA, pp. 753–756 (2016).

[13] Yu X., Li Y., Yang W., *Superposition Spinal Codes With Unequal Error Protection Property*, IEEE Access, vol. 5, pp. 6589–6599 (2017).

[14] Allard J.L., Dobell A.R., Hull T.E., *Mixed congruential random number generators for decimal machines*, Journal of the ACM, vol. 10, no. 2, pp. 131–141 (1963).

[15] Shuai C., *Fast linear congruence generator*, 2010 5th International Conference on Computer Science and Education, Hefei, China, pp. 1906–1908 (2010).

[16] Yu X., Li Ying, Yang Weiqiang, *Rateless spinal code for decode-and-forward relay channel*, 2015 International Workshop on High Mobility Wireless Communications (HMWC), Xi'an, China, pp. 71–75 (2015).

[17] Kunhu A., Al-Ahmad H., Taher F., *Medical images protection and authentication using hybrid DWT-DCT and SHA256-MD5 hash functions*, 2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Batumi, Georgia, pp. 397–400 (2017).

[18] Saraiva J.P. *et al.*, *Calculation of sensitivity index using one-at-a-time measures based on graphical analysis*, 2017 18th International Scientific Conference on Electric Power Engineering (EPE), Kouty and Desnou, Czech Republic, pp. 1–6 (2017).

[19] Sowndharya G., Vasuki A., *Reducing bit error rate using CRC verification in turbo codes*, 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, India, pp. 627–631 (2017).

[20] Megha Mukundan P., Manayankath S., Srinivasan C., Sethumadhavan M., *Hash-One: a lightweight cryptographic hash function*, in IET Information Security, vol. 10, no. 5, pp. 225–231 (2016).

[21] Li W., Liao G., Wen Y., Gong Z., *SpongeMPH: A New Multivariate Polynomial Hash Function based on the Sponge Construction*, 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC), Shenzhen, China, pp. 516–520 (2017).