A Symmetric Version of the McEliece Public-Key Cryptosystem

This article exploits the fact that linear codes can correct twice the number of erasures as that of errors, allowing reduction in code size and providing the same level of security. © 1997 by John Wiley & Sons, Ltd.

Int. J. Network Mgmt., **7**, 316–323 (1997)
No of Figures: 2   No of Tables: 5   No of References: 17

*By A. Kh. Al Jabri**

Introduction

The development of many high-speed data networks and the sensitivity of some of the data exchanged over these networks have created an increasing interest in encryption algorithms that are both secure and have high encryption and decryption rates. The McEliece public-key cryptosystem seems to satisfy such requirements. This is attributed to the simple modulo-2 arithmetic involved in both the encryption and decryption processes. This is not available in systems based on number theory where heavy computations are typically involved leading to a slow rate of encryption or decryption or both. The large code size of McEliece system, however, and the extra bandwidth required to transmit the encrypted data are, in fact, major obstacles in any series implementation of such a system.

Many public- and private-key variants of the McEliece system with short code lengths have been proposed in the literature[2,7] most of which have shown some form of weakness. Although some of the recent systems have a reasonable code size, they still have some security problems. An example of the most recent system that has been broken is the Gabidulin cryptosystem.[9] In general, the security of public-key algebraic-code cryptosystems is based on the fact that the decoding problem of a general linear code is an NP-complete. There is no known public-key cryptosystem for which decryption is an NP-complete problem. The RSA, for example, is unlikely to be so. If one does exist, it is possible that an algebraic-code public-key cryptosystem might provide an example. Although such a system might not be secure, evidence suggests that its security can be amplified to the required degree of security.[9] Many algebraic-code cryptosystems have been proposed in the literature whose security is based on the

Biography: A. Kh. Al Jabri. The author received his BSc (Honours) in 1979 from King Saud University in Riyadh, Saudi Arabia, and MSc (Electrical Engineering), MA (Mathematics) and PhD (Electrical Engineering) all from the University of Michigan (Ann Arbor), USA, in 1983, 1985 and 1987, respectively. His research interests include source and channel coding, information theory and cryptography.

**Correspondence to: Dr A. Kh. Al Jabri, EE Department, College of Engineering, King Saud University, PO Box 800, Riyadh 11421, Saudi Arabia*

McEliece concept. Their security, however, depends on the particular structure involved in their design. In fact some of these systems have been broken. The McEliece system is still, however, secure.[9]

Our objective in this article is to exploit some facts from algebraic-coding theory to develop a symmetric version of the McEliece system with enhanced security. The system exploits the fact that linear codes can correct twice the number of erasures as that of errors. In the proposed system erasures are used instead of errors. This allows reduction in the code size while providing the same level of security.

In the next section, preliminaries of algebraic-code encryption are reviewed. In the third section the proposed system is introduced. In the fourth section the security of the proposed system is analyzed. In the fifth section discussion of some related issues to the system is presented. Conclusions are drawn in the final section.

Preliminaries of Algebraic-code Encryption

The McEliece system is based on the existence of a large number of t-errors correcting Goppa codes in which one of these is selected. Let the $k \times n$ generator matrix of this selected code be G'. From this a public matrix G is constructed according to

$$G = SG'P$$

where S and P are $k \times k$ and $n \times n$ scrambling and permutation matrices, respectively. In reference 13 n is chosen to be 2^m for some integer m. In this case

$$k \geq 2^m - mt$$

The code rate, R, is k/n and the minimum distance, $d_{\min}$, satisfies $d_{\min} \geq (2t + 1)$. In what follows $d_{\min}$ is taken to be equal to $2t + 1$. The code construction and the decoding of Goppa codes are described in references 4, 5, 13 and 14. In the McEliece system m and t are 10 and 50, respectively.

Let

U denote the k-dimensional information vector

X the n-dimensional codeword vector, $\mathbf{X} = \mathbf{U}G$

Y the n-dimensional ciphertext vector

Z an n-dimensional random error vector with weight less than or equal to t. Here the weight is used to mean the Hamming weight or the number of ones in the vector.

Y is given by

$$\mathbf{Y} = \mathbf{X} \oplus \mathbf{Z}$$

where the operation $\oplus$ stands for position by position modulo 2 addition of **X** and **Z**. To make the recovery of the information possible using minimum distance decoding, the weight of **Z** must be less than or equal to t where

$$t = \frac{d_{min} - 1}{2}$$

The decryption is performed as follows.

Calculate $\mathbf{Y}P^{-1}$ where P^{-1} is the inverse of the matrix P. The resulting vector is a Goppa codeword with a permuted error vector $\mathbf{Z}P^{-1}$. Decode this using G and multiply the result by S^{-1}, the inverse of S, to obtain the information vector **U**.

The Proposed System

The proposed system is based on the concept of creating erasures in the encoded vector instead of adding random errors to it. Since erasures are errors with known locations, the location of these errors must be known to the two communicating parties. For this purpose we suggest that **Z** be replaced by the output of an error vector generator (EVG) in which the Hamming weight of the generated block must be less than or equal to $d_{\min} - 1$ selected in a uniform fashion among all possible blocks of length n and weight less than or equal to $d_{\min} - 1$. A schematic diagram of the proposed system is shown in Figure 1. The input to the EVG is a secret key, K, that must be made available to the communicating parties before the start of the communication session. The key is assumed to be changed from one session to another. In the sys-

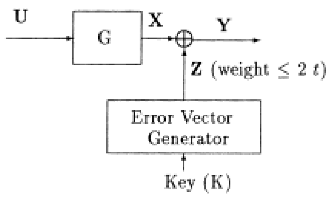

Figure 1. A schematic diagram of the proposed algorithm.

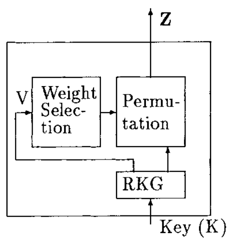

Figure 2. Error Vector Generator.

tem perfect synchronization between the two communicating parties is assumed*.

The proposed system is based on the concept of creating erasures in the encoded vector instead of adding random errors to it.

The encryption process is as in the McEliece system. However, the method of generating the error vector, **Z**, is different. During a communication session it is assumed that the same secret key K is used to generate all the error vectors required during this session through the EVG. This is achieved by letting the secret key drives a running-key generator (RKG) whose output is then used to generate all the subkeys needed in the EVG during the session. Using a single secret key for generating all the required subkeys will simplify the key management. The process of generating a particular **Z** is performed in two stages: the weight generation stage where part of the bits from the RKG are used to select from a list of initial vectors a vector $\mathbf{Z}_w^0$ of length n and weight w and a second stage where another part of the bits from the RKG is used to generate a pseudorandom permutation, Q, that is then used to permute $\mathbf{Z}_w^0$. Note here that Q is different from one block encryption to another in a manner determined by the RKG. That is, the error vector **Z** is given by

$$\mathbf{Z} = \mathbf{Z}_w^0 Q$$

A schematic diagram of the processes involved in these stages is given in Figure 2.

To have good system security, the generated **Z** must be uniformly distributed over the set of all possible vectors of length n and weight less than or equal to $2t$. A proposed method for generating **Z** is discussed next. Let W be a discrete random variable denoting the weight of the generated **Z** with a probability mass function, $P_W(w)$, given by

$$P_W(w) = \frac{\binom{n}{w}}{\sum_{i=0}^{2t}\binom{n}{i}} \qquad 0 \le w \le 2t$$

Now suppose that the input to the weight selection stage is a real random variable, V, uniformly distributed over the interval (0,1) and that this interval is partitioned into subintervals in pro-

w	v range	Initial error vector
0	$0 \le v < P_w(0)$	$\mathbf{Z}_0^0$
1	$P_w(0) \le v < P_w(1) + P_w(0)$	$\mathbf{Z}_1^0$
.	.	.
.	.	.
$2t$	$\sum_{i=0}^{2t-1} P_W(i) \le v \le 1$	$\mathbf{Z}_{2t}^0$

Table 1. Data used in the weight selection stage

*In many real channels a flip in the received bit, from one to zero or zero to one, is not the only source of errors in the received data. On many occasions, some extra symbols are inserted into the received data or some symbols are deleted from these data.[3] This leads to a loss in synchronization between the source and destination on the data level. Therefore, some measures need to be taken to account for their effects. One method is to insert some synchronization symbols within the stream of transmitted symbols. Another possibility is to operate the block cipher in some self-synchronous modes.[16]

portion to these probabilties. That is, the size of the first subinterval is $P_W(0)$, the second $P_W(1)$, and so on. For the *wth*, $w = 1,2,\ldots,2t$ subinterval an arbitrary initial error vectors, $\mathbf{Z}_w^0$ of weight w is assigned. This is summarized in Table 1.

If Q is selected in a uniform way among the set of all possible permutations, then the generated $\mathbf{Z}$ will also be uniform among the set of all possible vectors of length n and weigh less than or equal to $2t$.

Different methods are known in the literature for generating permutations.[16] Since data in algebraic-code cryptosystems are encrypted and decrypted at high rates, the permutation-generating method must be fast. In principle, if the vector to be permuted has L elements, then there are $L!$ possible permutation methods of this vector. For good security, the generated permutation should be selected uniformly among the elemtns of this set. Although the method can be easily implemented, such implementation seems to be slow. One technique that can provide high speed, although it has limited security, is to store a large number of permutations and then select for each block encryption one of them in a pseudorandom way.

Another method for generating a large number of permutations is suggested in reference 6 and is discussed next. To permute a vector of n bits with $n = 2^m$ for some integer m, we first generate a key of size $(n-1)$ pseudorandom bits. The following algorithm can then be used to perform the required permutation:[6]

(1) First divide the block to be permuted into two equal blocks, say b_1^0 and b_2^0 each with $n/2$ bits. If the first bit of the key is 1, then swap the two blocks. That is, the block (b_1^1, b_2^1) will change to (b_2^0, b_1^0). If, on the other hand, the bit is zero then no change.
(2) Treat the resulting two blocks as two separate blocks but with length $n/2$ and perform step 1 on each block using the next two bits from the key.
(3) Continue the same process until the block size to be swapped is of 2 bits in size.

This will result in the required permutation. The number, K_P, of pseudorandom bits required in this case is given by

$$K_P = \sum_{i=1}^{\log_2(n)-1} 2^i = n - 1$$

Note here that the number, N_p, of possible permutations when using this method is

$$N_p = 2^{K_p}$$

Since the code length is 2^m, the permutation can be performed in blocks of bytes (byte = 8 bits) or their multiples. For example if $n = 1024$ bits, then the block of n bits can be considered as a block of 128 bytes (8 bits per byte) or as 64 words (16 bits per word). Such structures will allow implementing such permutations at high speed using modern processors while allowing, at the same time, a large number of permutations. For example, in the 128 blocks of bytes, there will be $2^{128} - 1$ possible permutations. In addition, widely available block ciphers such as DES or IDEA can be used as a RKG if operated, for example, in the Output Feedback (OFB) mode.[16] Such ciphers can be operated at high speed since data are processed at the block level. Our software implementation using optimized C shows an average rate of 50×10^3 permutations per second is achievable on the byte level for $n = 1024$. This corresponds to a data rate of $k \times 50 \times 10^3$ or approximately 25 Mbits per second. If the permutations are to be performed on the bit level, then simulation results show that a rate of 2500 permutations per second is achievable.

System Security

An obvious way to evaluate the security of the system is to calculate the work factor of attacks typically used in evaluating the security of the McEliece system. Since no fast decoding is possible without knowledge of the system's private keys, one can select k bits from the ciphertext vector and use the corresponding columns from G to solve, when possible, for the information vector. After solving, one can check the validity of this solution by encoding the result and subtracting it from the ciphertext vector. If the weight of the resulting vector is less than $2t$ then the decoded vector is accepted as a valid information vector. That is, if $\mathbf{U}'$ is the decrypted information vector, then the interceptor needs to check the weight of $(\mathbf{U}'G +$

	Errors				Erasures		
t	k	R	$\log_2(W_t)$	x	k	R	$\log_2(W_x)$
35	674	0.658	84.09	70	674	0.658	143.81
36	664	0.648	84.13	72	664	0.648	144.02
37	654	0.639	*84.14*	74	654	0.639	144.15
38	644	0.629	84.10	*76*	644	0.629	*144.18*
39	634	0.619	84.02	78	634	0.619	144.14

Table 2. The work factors for $n = 1024$

C). Since the number of added errors is generally random, it is possible that the number of added errors is greater than t. In such cases there is a possibility that the added errors to the codeword make the resulting vector falls within the decoding range of another codeword. In this case, the eavesdropper is always left with some ambiguity about the decoded information.

The work factor, W_F, required to break the McEliece system is given in references 7, 10 and 13:

$$W_F = k^3 \frac{\binom{n}{k}}{\binom{n-\alpha t}{k}}$$

where α is a constant taken to be 1 in the case of errors and 2 in the case of erasures. Note here that the actual work factor must include a factor for the ambiguity in the obtained solution. In this case, the results obtained here are, in a sense, a lower bound on the actual work factor for the attacks involved. Let W_t and W_x denote such work factors in the case of errors only or erasures only, respectively. Tables 2–4 show values for these work factors of $n = 1024$, 512, and 256, respectively. The optimum values for t and x that maximize the work factors and the corresponding work factors in both cases are in italic.

As can be seen from these results, using erasures instead of errors significantly increases the work factor. In particular, for $n = 256$ a reasonable degree of security can be achieved. For a higher level of security $n = 512$ can be used instead. This is half the code length originally proposed by McEliece. In references 10 and 17, an attack was proposed that significantly reduces W_F almost by a factor of 2^{11} for the parameters proposed by McEliece and later optimized in reference 1. The attack is based on a simple procedure for checking the validity of the decrypted information where the interceptor checks whether the selected block of k bits from **Y** is error free and can be decrypted into a valid message. If this is not the case, the attacker then examines the possibility that this

	Errors				Erasures		
t	k	R	$\log_2(W_t)$	x	k	R	$\log_2(W_x)$
18	350	0.684	56.21	36	350	0.684	89.37
19	341	0.666	56.31	38	341	0.666	89.75
20	332	0.648	*56.33*	40	332	0.648	89.96
21	323	0.631	56.26	*42*	323	0.631	*90.01*
22	314	0.613	56.13	44	314	0.613	89.91
23	305	0.596	55.92	46	305	0.596	89.66

Table 3. The work factors for $n = 512$

	Errors				Erasures		
t	k	R	$\log_2(W_t)$	x	k	R	$\log_2(W_x)$
8	192	0.750	39.25	16	192	0.750	57.02
9	184	0.719	39.59	18	184	0.719	57.98
10	176	0.688	*39.75*	20	176	0.688	58.59
11	168	0.656	39.75	*22*	168	0.656	*58.87*
12	160	0.625	39.60	24	160	0.625	58.86
13	152	0.594	39.32	26	152	0.594	58.57

Table 4. The work factors for $n = 256$

selected block contains small error patterns before selecting another block. This procedure, however, cannot be applied here since it is possible that the decrypted message is not unique and the interceptor is always left with some ambiguity about the decoded information.

There is, however, an extra security consideration that has to be taken into account in the proposed system. This is the possibility, in a chosen plaintext attack, that the enemy is able to drive the system by a sequence of zero information vectors. This will yield a sequence of blocks corresponding to the error vector **Z** and hence some information about the output of the RKG. If the generator output can be predicted then the system can be completely broken. Therefore, one should be careful in designing the EVG. If the RKG, however, has good randomness properties, then the generated permuted error vector will be uniformly distributed over the set of all possible error vectors. In this case, no information is leaked about the key. If, for example, a good RKG such as DES in the OFB mode is used, then such attack is less likely to succeed.

This attack can be further reduced by randomizing the input. In practice there are different methods to perform randomization.[12] Examples include text padding where random bits are padded to U then the resulting block is encrypted. This requires reducing the size of **U** and hence reducing the data rate. It seems, however, that the proposed system, in its simple structure, can provide a reasonable degree of security.

To achieve a good security, the weight of **Z** should be uniformly selected among the set of all possible error vectors of weight less than or equal to $2t$. This is to maximize the ciphertext uncertainty or entropy where the entropy, $H(V)$, of a random variable V with a probability mass function $\Pr(V = v) = p_v$ over a finite set ν is given by[8]

$$H(V) = -\sum_{v \epsilon \nu} p_v \log_2 p_v$$

This achieves its maximum value when V has uniform distribution.

Discussion

In general, increasing the amount of added errors to the encoded vector increases significantly the work factor required to break the cipher and hence allowing for the possibility of reducing the code length while keeping the work factor reasonably high. In the proposed system, as in the case of the McEliece system, there is a possibility for further improvement in the code rate by modulating the error vector by information.[11] One needs to note here that it is not allowed to select the position of the erased symbols since these positions are predetermined by the output of the EVG. However, the bits within these positions can be modulated by information. For example, if the weight of **Z** is chosen to be $2t$, then $2t$ bits of information can be modulated. The improved code rate, R_I, is then upper bounded by

$$R_I \leq \frac{k+2t}{n}$$

The improvement in the rate is shown in Table 5 for different code lengths.

This is different from the error case where both the positions of the errors and the value within these positions are random and can be modulated

n	t	R	R_I
1024	38	0.629	0.703
512	21	0.631	0.713
256	11	0.656	0.742

Table 5. Improved rate using error modulation for different codes

by information. The amount of extra bits, I_e, that can be modulated is bounded by

$$I_e \le \log_2\left(\sum_{i=0}^{2t}\binom{n}{i}\right) \text{ bits}$$

Another security factor that has to be taken into account when reducing the code length is the leakage of the input data statistics to the output of the cipher. Note here that the length of the code proposed by McEliece is so long that the chance of the same input being repeated is very slim. However, this is not the case if the size of the code is reduced. In typical block ciphers such as DES, other modes of operations such as block chaining or cipher feedback are usually used. In McEliece type cryptosystems some of these modes cannot be used since the output is not uniquely determined by the input. In the proposed system, however, since the positions of the generated errors are known, some of these modes can be used with some modification to allow for synchronization. Using these modes will also allow for hiding the EVG information, and, therefore, increasing the security of the system.

Another possibility for implementing the private-key version of McEliece system is to add e errors and erase x bits at the same time such that

$$2e + x = d_{\min} - 1$$

The erased positions have again to be determined by an EVG but the error positions can otherwise be selected randomly. By doing so, greater confusion for the data from the RKG is achieved.

Conclusions

In this article a private-key or a symmetric version of the McEliece system has been proposed. The system is based on the same concept suggested by McEliece except that erasures are used instead of errors. Such a modification allows for almost doubling the amount of added errors to the encoded vector and, therefore, increasing the amount of work required by an eavesdropper to recover the information. As a result the code length can be reduced to values much less than those proposed by McEliece while keeping a reasonable level of security.

Acknowledgements

The author would like to thank Professor M. Bossert (Ulm University, Germany) for pointing out the method of permutation considered in the article and the useful discussion related to this point. This was presented in part at the 6th International Conference on Computer Theory and Application, Alexandria, Egypt, 3–5 September 1996.

References

1. C. Adams and H. Meijer, Security related comments regarding McEliece's public-key cryptosystem, *IEEE Trans. Inform.*, **35**, No. 2, 454–5, March 1989.
2. A. Kh. Al Jabri, F. Al Thukair and A. Mirza, Private-key encryption based on concatenation of codes, *IEE Proc.-Commun.*, **141**, No. 3, 105–10, June 1994.
3. A. Kh. Al Jabri and Kh. Al-Mashouq, Channels with insertion and/or deletion and their capacity, *Proc. 1994 IEEE Int. Symp. on Inf. Theory.*, Trondheim, Norway, 27 June to 1 July 1994, pp. 327.
4. E. R. Berlkamp, Goppa codes, *IEEE Trans. Inf. Theory*, **IT-19**, No. 5, 590–93, September 1993.
5. R. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, Reading, MA, 1983.
6. M. Bossert, Private communication.
7. E. F. Brickell and A. Odlyzko, Cryptanalysis: A survey of recent results, *Proc. of the IEEE*, Vol. 76, No. 5, May 1988, pp. 153–165.
8. T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley, New York, 1991.
9. K. Gibson, The security of Gabidulin public-key cryptosystem, in U. Maurer (ed.), Lecture Notes in Computer Science, Science 1070, *Advances in Cryptology: Proc. Eurocrypt '96*, Saragossa, Spain, Springer-Verlag, Berlin 1996, pp. 212–23.
10. P. J. Lee and E. F. Brickell, An Observation on the Security of McEliece's Public-Key Cryptosystem, in G. G. Gunther (ed.) Lecture Notes in Computer Science 330, *Advances in Cryptology: Proc. Eurocrypt '88*, Davos, Switzerland, May 25–27 1988, Springer-Verlag, Berlin; 1988, pp. 275–80.
11. M. C. Lin and H. L. Fu, Information rate of McEliece public-key cryptosystem, *Electronics Letters*, **26**, No. 1, 16–18, January 1990.

12. J. Massey, Contemporary cryptology: an introduction, in G. Simmons (ed.), *Contemporary Cryptology: The Science of Information Integrity*, IEEE Press, New York, 1992, pp. 1–39.
13. R. J. McEliece, *A Public-Key Cryptosystem Based on Algebraic Coding Theory*, DSN Progress Report 42–44, Jet Propulsion Laboratory, CA, January–February 1978, pp. 114–16.
14. F. J. McWilliams and N. J. Sloane, *The Theory of Error Correcting Codes*, 3rd edition, North Mathematical Library, Vol. 16, North Publishing Co., Netherlands 1983.
15. T. R. N. Rao and K. H. Nam, A private-key algebraic-code encryption, *IEEE Trans. on Information Theory*, **33**, No. 4, 829–33, July 1989.
16. B. Schneier, *Applied Cryptography: Protocols, Algorithm, and Source Code in C*, 2nd edition, John Wiley, New York, 1996.
17. J. van Tilburg, On the McEliece public-key cryptosystem, in S. Goldwasser (ed.), Lecture Notes in Computer Science (403), *Advances in Cryptology: Proc. Crypto '88*, Springer-Verlag, Berlin, 1990, pp. 119–131. ■