Transactions of the Illinois State Academy of Science
(1991), Volume 84, 1 and 2, pp. 54-64

A Cryptosystem Defined Using Elements of Norm Three of an Arithmetic in Cayley's Algebra

Duckkyung Kim and Patrick J. C. Lamont
Department of Computer Science
Western Illinois University
Macomb, IL 61455

ABSTRACT

This paper describes the implementation of a cryptographic system using the arithmetics of Cayley's classical algebra. A prototype, using 100 norm 3 codewords, running on an International Business Machines Personal Computer AT with Turbo Pascal as programming language and a space character as delimiter between variable length frames gave interactive encryption and decryption without delay, an information rate with an upper bound of 10 bits per character, and a maximum frame length of 18 before single precision overflow. In view of the time required to perform exhaustive search factorization, the number of parentheses patterns, the fact that codewords may have diverse components and patterns of sign, the fact that bases can be changed, and the choice of order and orders in the decryption algorithm, it would appear that plaintext attack is time consuming and that systems may be judged secure.

INTRODUCTION

The paper describes the implementation of security in data communications using an original cryptographic method of algebra and number theory [14].

A Cayley algebra C is an eight dimensional algebra defined over the real numbers and having nonassociative multiplication. All octants or elements of C satisfy a rank equation and have a multiplicative norm equal to the sum of the squares of the coefficients. A Cayley arithmetic J has a multiplicative identity, is closed under addition, subtraction and multiplication, and is such that, for all elements of J, the rank equation has rational integral coefficients [6].

The objective of the paper is to describe prototype systems software for the encryption and decryption of data using elements of norm three of an arithmetic J and to illustrate arithmetic and cryptographic properties of any J.

The encryption process translates strings of characters into frames of Cayley integers. The compaction process uses the nonassociative, norm preserving Cayley multiplication. The multiplication admits unique factorization up to congruence modulo 2 in J. An algorithm is known for the process of decompaction and decryption [14].

The existence, for example, of a Chinese remainder theorem and Moebius inversion formulas [9] for the arithmetics J show that the subject of number theory in Cayley arithmetics is a productive field of research. An IBM PC AT computer supported the development of the prototype which is believed to be important from the point of view of mathematics, cryptography, data security, integrity and communication. Techniques of pure mathematics research and software systems design, development and quality control were applied.

LOGISTICS

The stages in the development of the encrypter and decrypter prototype, referred to as the two parts of the project, were as follows.
(i) Lectures were given to interested graduate students at this university on Cayley's algebra, arithmetics, and cryptography. Reading material was assigned from the texts by Lidl [15] and Welsh [21] on cryptography and the text by Klir [5] on the architecture of systems problem solving, and notes of the author on systems programming.
(ii) The two parts of the project were specified and defined. Discussion of data structures was followed by algorithm development and discussion of modularity. A top down plan, with flowcharts, of the required software prototype was designed.
(iii) Modules of the prototype were designed, coded and tested, and finally assembled into the complete package. The lectures, and design, analysis and coding training experience were offered as part of the seminar, guided study and research classes available to graduate students at this university.
(iv) In the development of the main project, quality control and absolute accuracy were fundamental to the evaluation process. It was found that possibilities for designing other cryptographic systems, including those using the difficult twisted composition algebras defined over an algebraic number field, exist. Further research on Cayley arithmetics was carried out to deal with these possibilities.
(v) An IBM PC AT computer, dedicated to the project, provided sufficient computer power for the development of the two parts of the project.

The detailed description of the work undertaken designing and developing the two parts of the project is given in the following sections. Cayley multiplication has already been successfully programmed and plays a fundamental role in both encryption and decryption.

THE PROGRAM CRYPT

The goal of the program CRYPT is to develop a cryptosystem with elements of fixed norm 3 as code words. The program is made up of 5 units: unit DATATYPE, unit READTBL, unit MULTIPLY, unit ENCRYPT, and unit DECRYPT. Unit DATATYPE declares the common data types and variables used in the system. Unit READTBL reads the codeword table and input plaintext strings. Unit MULTIPLY performs Cayley multiplication to obtain a cipher or encoded frame. Unit ENCRYPT encodes input strings, generates a cipher and records control characters. Unit DECRYPT decodes a cipher using the control characters and produces plaintext.

The program, CRYPT, is a prototype. Input to be fed to the encryption process and output to be obtained from the decryption process consist of strings of uppercase letters, spaces, and special characters except '^'. The maximum number of input/output characters which can be encoded by CRYPT at one time is in multiples of 80. Not more than 18 characters can be encrypted and decrypted in one frame because the single precision integer type variable is in the range -32,768 to 32,767 in Turbo Pascal.

The codeword table has fields: uppercase character, space or special character, control character, and codeword or octant. The control character ranges from 0 to 7 and is determined by a heuristic mathematical calculation. Control characters provide a key to the system and are output by the encryption process and input to the decryption process. A codeword or octant consists of any combination of three signed ones and five zeros. The table is arranged in the order of ASCII code (Figure 1).

A cipher is generated by the encryption process and is fed to the decryption process. A cipher consists of 8 integers. The greatest common factor of the integers of a cipher must not be divisible by 3 to ensure unique factorization.

A frame is a set of codewords that can be encrypted together as a single octant without causing overflow on the available machinery.

The encryption process reads input strings supplied by the user. The process first checks their validity. If there is any invalid character, the process is stopped. The process regards a space included in an input string as a delimiter. In other words, input strings are separated by spaces into frames and the process performs encryption only for frames. Each cipher is checked to determine whether the greatest common divisor of its components is divisible by 3. If the greatest common divisor is divisible by 3, an appropriate special character or characters are inserted into the frame. The new frame is encrypted again. The encryption process is repeated until we obtain a cipher with greatest common divisor not divisible by 3.

For each input character, the encryption process looks up the corresponding character in the codeword table. Control characters and ciphers are transmitted

to the decryption process. The decryption process finds all characters and codewords with control character equal to the given control character and multiplies the transmitted cipher by the conjugates of the codewords corresponding to the given control character. If the resulting cipher is divisible by 3, the character is part of the transmitted plaintext and is output.

PROTOTYPE PROGRAM UNITS

Unit DATATYPE defines global variables used during the execution of the program CRYPT.

```
Const
    length_byte=8;
    length_code=56;
    max_length=80;
Type
    octant=array[1..length_byte] of integer;
    { array for codeword }
    stack_char=array[1..max_length] of char;
    { array for input }
    stack_codeword=array[1..max_length] of octant;
    { array for output }
    table=record              { record for the codeword table }
        alpha:char;           { letters and special characters }
        control:char;         { control characters of letters }
                              { and characters }
        codeword:octant;      { codeword of letters and special }
                              { characters }
    table_struc=array [1..length_code] of table;
    { array for the codeword table }
    matrix_2=array[1..1,1..length_byte] of integer;
    { workspace for an octant }
Var
    i_message,s_message:stack_char;
    control_char:stack_char;
    code:stack_codeword;
    code_table:table_struc;
    flag:char;
    no_ch:integer;
```

Unit ENCRYPT encrypts input strings and calls unit MULTIPLY to produce an octant cipher. The unit contains the procedure Calculate_Codeword.

```
Procedure Calculate_Codeword   (x:integer;
                                var code1:stack_codeword);
{ This procedure repeats the calculation until we obtain      }
{ octant.                                                     }
{ The Parameter -    x : the number of the input string       }
```

```
{ Output Parameter - code1 : array of codewords                    }
{                                    At the end of this            }
{                                    procedure, the first          }
{                                    element of code1 is the       }
{                                    desired octant.               }
var
    a,b:integer;
    result:matrix_2     { a work space for an octant }
    matrix2:matrix_2;  { a work space for an octant }
begin
for a := x downto 1 do
begin
{ move the codeword to the work space to calculate an              }
{octant                                                            }
                for b : 1 to length_byte do
                    matrix2[1,b] :=code1[a,b];
                { does the matrix multiplication }
                Multi_Codeword (code1[a-1],matrix2,result);
{ move the value obtained from Multi_CodeWord to code1   }
                for b := 1 to length_byte do
                    code1[a-1,b] := result[1,b];
        end; { for loop }
end; { Calculate_Codeword }
```

Unit MULTIPLY performs Cayley multiplication. The unit is called by ENCRYPT and DECRYPT.

```
Procedure Multi_CodeWord   (a:octant;
                            b:matrix_2;
                            var m:matrix_2);
{ This procedure is to calculate codewords to get string           }
{ encrypted.                                                       }
{ Input Parameters -    a : a work space for an octant             }
{                           the second last element of the         }
{                           table 'code'                           }
{                       b : a work space for an octant             }
{                           the last element of the table          }
{                           'code'                                 }
{ Output Parameter -    m : a work space for the octant            }
{ calculated                                                       }
var
    i,j,k,:integer;
    s:integer;
    c,d:array[1..length_byte,1..length_byte] of integer;
e,z:octant;             {work spaces for an octant }
begin
        for i := 1 to length_byte do
            for j := 1 to length_byte do
```

```
            c[i,j] := a[i] * b[1,j];
        s:=c[1,1];
        for i:=2 to length_byte do
            s := s - c[i,i];
        for i :=2 to length_byte do
        begin
            e[i] := c[1,i]+c[i,1];
            for j := 2 length_byte do
                c[i,j] :+ a[i] * b[1,j];
            s:= c[1,1];
            for i:=2 to length_byte do
                s := s - c[i,i];
            for i :=2 to length_byte do
            begin
                e[i] := c[1,i]+c[i,1};
                for j := 2 to length_byte do
                d[i,j] := c[i,j]-c[j,i];
        end; { for loop }
        z[1] := d[3,4]+d[5,6]+d[8,7]+e[2];
        z[2] := d[4,2]+d[5,7]+d[6,8]+e[3];
        z[3] := d[2,3]+d[7,6]+d[5,8]+e[4];
        z[4] := d[6,2]+d[7,3]+d[8,4]+e[5];
        z[5] := d[2,5]+d[4,7]+d[8,3]+e[6];
        z[6] := d[3,5]+d[6,4]+d[1,8]+e[7];
        z[7] := d[3,6]+d[4,5]+d[7,2]+e[8];
        m[1,1] := s;
        for k := 2 to length_byte do
        m[1,k] := z[k-1];
end; { Multi_CodeWord }
```

EXAMPLES AND CONCLUSIONS

Here is an example of running the program.

```
* Enter the input string ... VIA*VERITAS*VITA
*    Control Char. ==> 0730073723300723
*         Octant ==> -2212 -1573  1251 -2519  -155  -128  3986  -3441
* Char. Decrypted ==> VIA*VERITAS*VITA
```

The system encrypts VIA*VERITAS*VITA and decrypts without any difficulty.

Here is another example.

```
* Enter the input string ... VIA VERITAS VITA
*    Control Char. ==> 073
        Octant ==>   -2   -3   2   1   0   2   -1  2
* Char. Decrypted ==> VIA
* Press Enter to see more.
```

```
*    Control Char. ==> 07372363
         Octant ==>   37   -28  4 -21  38   33   7 37
* Char. Decrypted ==> VERITA!S
* Press Enter to see more.
*    Control Char. ==> 0723
*         Octant ==>   -2   -4 -2   0  -4   2  1  6
* Char. Decrypted ==> VITA
```

Here VIA VERITAS VITA is regarded as three separate input frames. The system performs the encryption and decryption processes for each frame. In the case of VERITAS, a special character, "!", is inserted into the string because the cipher of VERITAS has common factor divisible by 3.

A third example follows.

```
* Enter the input string ...WATSON
*    Control Char. ==> 73234675
         Octant ==>  -15  38  19  -14  -3  -43  19  46
     Char. Decrypted ==> WATSO!"N
```

Two special characters are inserted into the string, 'WATSON'. The location where a special character is inserted is determined by the system.

The demonstration program runs in an interactive mode. An evaluation program was written which does not employ the interactive mode. Here an input frame is 10 characters long. For example, 7,218 characters were read and 43,295 bits were needed to generate the encryption. The information rate was found to be about 8 bits per character.

REFERENCES

[1] F. Van der Blij, History of the octaves, Wis-en Natuurkindig Tijdschrift (=Simon Stevin) 34, III (1961), 106-125.

[2] F. van der Blij and T. A. Springer, The arithmetics of the octaves and of the group G_2, Nederl. Akad. Wetensch. Proc. (=Indag. Math.) 62A (1959), 406-418.

[3] L. E. Dickson, On quaternions and their generalization and the history of the eight square theorem, Annals of Mathematics (2), 20 (1919), 155-171, 297.

[4] A. Hurwitz, Ueber die Composition der quadratischen Formen von beliebig vielen Variabeln, Nachrichten der Gesellschaft der Wissenschaften Goettingen (1898), 309-316.

[5] G. J. Klir, Arhitecture of Systems Problem Solving, New York and London (1985).

[6] P. J. C. Lamont, Arithmetics in Cayley's algebra, Proc. Glasgow Mathematical Assoc. 6 (1963), 99-106.

[7] P. J. C. Lamont, Ideals in Cayley's algebra, Nederl. Akad Wetensch. Proc. (=Indag. Math.) 66A (1963), 394-400.

[8] P. J. C. Lamont, Approximation theorems for the group G_2, Nederl. Akad. Wetensch. Proc. (=Indag. Math.) 67A (1964), 187-192.

[9] P. J. C. Lamont, Factorization and arithmetic functions for orders in composition algebras, Glasgow Mathematical Journal 14 (1973), 86-95.

[10] P. J. C. Lamont, The number of Cayley integers of given norm, Proc. Edinburgh Math. Soc. 25 (1982), 101-103.

[11] P. J. C. Lamont, Computer generated natural inner automorphisms of Cayley's algebra, Glasgow Mathematical Journal 23 (1982), 187-189.

[12] P. J. C. Lamont, A computer approach to investigating inner automorphisms of Cayley's algebra. Transactions of the Illinois State Academy of Science 75 (1982), 65-70.

[13] P. J. C. Lamont, The nonexistence of a factorization formula for Cayley numbers, Glasgow Mathematical Journal 24 (1983), 131-132.

[14] P. J. C. Lamont, Unique factorization in Cayley arithmetics and cryptology. To appear. Glasgow Mathematical Journal (1991).

[15] R. Lidl and H. Niederreiter, Introduction to finite fields and their applications, Cambridge (1986).

[16] C. P. Pfleeger, Security in Computing, Englewood Cliffs (1989).

[17] V. Pless, Introduction to the theory of error-correcting codes, New York (1982).

[18] R. A. Rankin, On representations of a number as a sum of squares and certain related identities, Proceedings of the Cambridge Philosophical Society 41 (1945), 1-11.

[19] R. A. Rankin, A certain class of multipicative functions, Duke Math. J. 13 (1946), 281-306.

[20] O. Taussky, Sums of squares, Am. Math. Monthly 77 (1970), 805-830.

[21] D. Welsh, Codes and Cryptography, Oxford (1988).

Figure 1

CODEWORD TABLE

Character	Control Character	Codeword
	1	10000011
!	6	01011000
"	7	01010100
#	4	01010010
$	5	01010001
%	0	01001100
&	3	01001010
`	2	01001001
(	2	01000110
)	3	01000101
*	0	01000011
+	5	00111000
,	4	00110100
-	7	00110010
.	6	00110001
/	3	00101100
:	0	00101010
;	1	00101001
<	1	00100110
=	0	00100101
>	3	00100011
?	2	00011100
@	1	00011010

Character	Control Character	Codeword
A	3	11100000
B	2	11010000
C	5	11001000
D	4	11000100
E	7	11000010
F	6	11000001
G	1	10110000
H	6	10101000
I	7	10100100
J	4	10100010
K	5	10100001
L	7	10011000
M	6	10010100
N	5	10010010
O	4	10010001
P	1	10001100
Q	2	10001010
R	3	10001001
S	3	10000110
T	2	10000101
U	2	00010011
V	0	01110000
W	7	01101000
X	6	01100100
Y	5	01100010

Character	Control Character	Codeword
Z	4	01100001
[	0	00011001
\	0	00010110
]	1	00010101
_	7	00001110
`	6	00001101
¦	5	00001011
¦	4	00000111