Cryptography, The Art of Disguising The Written Word

David Franklin, Independent Consultant, New Hampshire, USA

ABSTRACT

"Can we hide or encrypt that variable somehow so that it appears as complete gibberish?" That was the question put to me on a Friday afternoon, which started the journey into what is known as Cryptography, the process of converting information into unintelligible goblygook and then taking that text and converting it back into its original form. This paper takes a look at cryptography, introduces monoalphabetic, polyalphabetic and transposition cyphers, and finally introduces key-based cypher that is considered to be the essential element in generating modern cyphertext. Emphasis will be placed on encrypting a single variable within a dataset, rather than a whole dataset as there are tools from SAS and third party software to do this task.

INTRODUCTION

Encrypting data so that it makes it difficult for people to see it, other than those authorized to do so, has become an important subject when data is stored. SAS® provides a way that whole datasets can be encrypted, for example using the ENCRYPT option inside the DATA statement, but there is little in the way of encrypting a single variable inside a dataset. This paper takes a look at a some cyphers used today, and in the past, and shows how they can be used to encrypt the data for a variable.

MONOALPHABETIC CYPHERS

These are commonly known as substitution cyphers where one character is substituted for another. In many ways this type of substitution is already done to our data, for example a gender format with 1=Male and 2=Female, where without the decode it is almost impossible to state with certainty what the coded values mean.

Among this category of cypher the most famous cypher is the Caesar Cypher, where each character is "shifted" a universal specified number of characters. The following example shows what happens when a phrase is encoded using this method using a shift of 13 characters:

```
    Plaintext:  WELCOME TO NESUG
    Cyphertext: JRYPBZR GB ARFHT
```

The related SAS code is shown below:

```
    %macro caesar(plaintxt= /*Original String*/
                 ,shift=13 /*Shift number*/
                 );
        data _null_;
            retain charstr 'ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ';
            length cyphertxt $200;
            origlen=length("&origstr");
            cyphertxt=repeat(' ',origlen-1);
            do i=1 to origlen;
                if substr("&origstr",i,1)=' ' then substr(cyphertxt,i,1)=' ';
                else do;
                    x=index(charstr,substr("&origstr",i,1));
                    y=sum(x,&shift);
                    z=substr(charstr,y,1);
                    substr(cyphertxt,i,1)=substr(charstr,y,1);
                end;
            end;
            cyphertxt=strip(cyphertxt);
            put "plaintxt=&plaintxt" / cyphertxt=;
        run;
    %mend caesar;
```

Unfortunately these types of cypher can be easily broken by character frequency analysis and it not best to encrypt data using this type of cypher.

POLYALPHABETIC CYPHERS

These type of cypher involve two or more cypher alphabets meaning that a single character can be represented by more than one character in the encrypted message. These types of cypher are harder to crack.

Among this category of cypher the more well known of this type was the Vigenere Cypher which is based on the following table:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
								(continued)																		
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

This cypher uses the table along with a keyword to encode the message, as the following example illustrates:

```
Plaintext:  WELCOME TO NESUG
Keyword:    VERMONT VE RMONT
Cyphertext: RICOCZX OS EQGHZ
```

The related SAS code is shown below:

```
%macro vigenere(plaintxt= /*Original String*/
               ,keyword=  /*Keyword*/
               ,action=E  /*E=Encrypt,D=Decrypt*/
               );
    data _null_;
      retain charstr 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
      length cyphertxt $200 ol $1 kl $1 pl 3 nl $1;
      do i=1 to length("&plaintxt");
          ol=substr("&plaintxt",i);
          kl=substr("&keyword",mod(i-1,length("&keyword"))+1,1);
            if upcase("&action")="E" then
              pl=mod(sum(index(charstr,ol),index(charstr,kl))-2,26)+1;
            else if upcase("&action")="D" then
              pl=mod(index(charstr,ol)-index(charstr,kl)+(26*4),26)+1;
          nl=substr(charstr,pl,1);
          cyphertxt=cats(cyphertxt,nl);
      end;
      put "plaintxt=&plaintxt" / cyphertxt=;
    run;
%mend vigenere;
```

There are a number of cyphers of this sort, including the Gronsfeld cypher which uses a number as the key instead of characters as in the Vigenere Cypher.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
					(continued)																					
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I

TRANSPOSITION CYPHERS

These type of cypher involve the rearranging of characters to encrypt the message.

An example of this type of cypher is known as the Rail Fence Cypher which literally takes a block of text and transposes it - the plaintext is written in a zig-zag direction on successive "rails" of an imaginary fence. The following example demonstrates its use using a four fence layout:

```
    Plaintext:  WELCOMETONESUG
    Grid:       W.....E.....U.
                .E...M.T...S.G
                ..L.O...O.E...
                ...C.....N....
    Cyphertext: WEUEMTSGLOOECN
```

In order for this to work successfully the number of rails must be two or more – the more the better.

A SAS macro that will do this cypher is below:

```
    %macro RailFence(plaintxt= /*Original String*/
                    ,rails=3   /*Number of Rails, should be >=2*/
                    );
        data _null_;
            length cyphertxt $50;
            array z{50} _temporary_;
            retain dir 1;
            do i=1 to length("&plaintxt");
                if i=1 then z{i}=1;
                else z{i}=z{i-1}+(dir);
                if z{i}=&rails then dir=-1;
                else if z{i}=1 then dir=1;
            end;
            do j=1 to &rails;
                do i=1 to length("&plaintxt");
                    if z{i}=j then
                      cyphertxt=cats(cyphertxt,substr("&plaintxt",i,1));
                end;
            end;
            put "plaintxt=&plaintxt"/cyphertxt=;
        run;
    %mend RailFence;
```

Some versions of this cypher have filler characters a the end so that in the case of the example above there would be two extra characters to finish up the sequence going down.

KEY-BASED CYPHERS

Keys are the essential component in generating modern cyphertext. In basic form, a key is a string of bits used allowing for data to be encrypted and decrypted, as will as do other mathematical operations. In many forums people talk about the lengths of keys in terms of bits or 'n-bit blocks' - the longer the key the more difficult it is to break the encrypted message.

One of the best know of this type of cypher is the RSA key and this cypher relies on two "keys" - a public key that is used to encrypt the message and a private key that is needed to decrypt the cyphertext. This method relies on the fact that the two keys are related using the follow formula:

$$C = M^e \bmod n$$

and

$$M = C^d \bmod n$$

where C is the cyphertext, M is the plaintext, e is the public key and d is the private key.

To generate the two keys the following procedure is used:

1. Set two prime numbers, p and q
2. Let n = pq
3. Let m = (p-1)(q-1)
4. Choose a small number e such that it is a coprime to m, that is e and m must have no common factor other than 1
5. Find d, such that de mod m = 1

Finally, e and n can be used as the public key and d and n as the private key. The only real calculations here of any consequence are steps 4 and 5. In step 4 it is possible to check if e and m are coprime by using the Euclidean algorithm, the source code of which is given below:

```
%macro gcd(x,y);
   data _null_;
      gcd = x;
      ref = y;
      do while (ref > 0);
         res = mod(gcd,ref);
         gcd = ref;
         ref = res;
      end;
      put gcd= x= y=;
   run;
%mend gcd;
```

Step 5, is equivalent to solving the equation de = 1 + im, where i is any positive integer. The following is an example, using two small prime numbers for easy calculation purposes:

1. p = 5, q = 29
2. n = pq = 145
3. m = (p-1)(q-1) = 4 * 28 = 112
4. e = 3
5. d = 75

Thus the public key is n=145 and e=3, and the private key is n=145 and d=75.

Now lets do a message. For this example the word NESUG is going to be encoded:

NESUG is set as characters 13 4 18 20 6

Encrypting the text we do the following calculations:

N: 13^3 mod 145 = 22

E: 4^3 mod 145 = 64

S: 18^3 mod 145 = 32

U: 20^3 mod 145 = 25

G: 6^3 mod 145 = 71

Therefore the cyphertext that would be sent is 22 64 32 25 71.

To decode the cyphertext received it is necessary to do the following calculations:

22^75 mod 145 = 13 that translates to N

64^75 mod 145 = 4 that translates to E

32^75 mod 145 = 18 that translates to S

25^75 mod 145 = 20 that translates to U

71^75 mod 145 = 6 that translates to G

The security of RSA depends on the time it takes to factor large numbers - the math above is not trivial and it is laborious. Below is a macro that was derived for doing large calculations like the above:

```
%macro largemod(x=   /*X value*/
               ,pwr= /*Power value*/
               ,md=  /*MOD value*/
               );
    data _null_;
        mlt=.; x=&x; pwr=&pwr; md=&md; i=0;
        do while(pwr>2 and i<10);
            i+1;
            if mod(pwr,2)=1 then do;
                if mlt=. then mlt=x;
                else mlt=mlt*x;
                pwr=pwr-1;
            end;
            x=mod(x**2,md);
            pwr=pwr/2;
        end;
        if mlt=. then y=mod(x**pwr,md);
        else y=mod(mlt*(x**pwr),md);
        put y=;
    run;
%mend largemod;
```

CONCLUSION

This paper just scratches at the surface of cryptography but has presented four methods, along with SAS code, that are in general use. With some basic mathematics and programming it is possible to make you own encryption. Some of the techniques used in practice use a combination of these, among other methods. However, it must be pointed out that there currently is no such thing as a secure transmission.

REFERENCES

Two previously published papers that may be of further interest are:

Effective Data Encryption Algorithms using the SAS System, by Annette I. Ladan (SUGI 20)

Using SAS Bitwise Functions to Scramble Data Fields with Key, by Sheng Luo and Xinsheng Lin SUGI22

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

David Franklin
Independent Consultant
16 Roberts Road
Litchfield NH 03052
Work Phone: 603-275-6809
Email: 100316.3451@compuserve.com
Web: http://ourworld.compuserve.com/homepages/dfranklinuk