Distributed Storage scheme Based on Secret Sharing Schemes

Shuang Wang
School of Electrical and Computer Engineering, University of Oklahoma, Tulsa, OK, USA
shuangwang@ou.edu

Abstract

Distributed storage systems provide reliable access to data through redundancy spread over individually unreliable nodes. Generally, in such a system, a file of size M will divided into k pieces, each of size M/k, then these k pieces are encoded into n coded pieces and stored at n nodes. In cryptography, secret sharing offers a similar scheme, which means a technique for sharing a secret to a group of members, each of which holds a portion of the secret. The secret can only be retrieved when a certain number t of members combine their shares together, while any combination with fewer than t shares has no extra information about the secret than 0 shares. In this paper, the development of secret sharing schemes is reviewed and the basic algorithm of secret sharing is introduced in details. Moreover, the secret sharing schemes is applied to study the distributed Storage problem. By using the similar method based on Shamir threshold scheme, a file can be split into n small sub-files. With any combination of these t sub-files ($t \le n$), the original file can be recovered without errors.

Keywords: Secret sharing, Shamir Threshold schemes, Distributed Storage.

1. Introduction

The purpose of distributed storage is to store data safely using a distributed collection of storage nodes. Generally, in such a system, a file of size M will divided into k pieces, each of size M/k, then these k pieces are encoded into n coded pieces and stored at n nodes. With any combination of these t sub-files ($t \le n$), the original file can be recovered without errors. There are many existed distributed storage system such as OceanStore [1], Total Recall [2] and DHash[3]. In this paper, I propose a distributed storage scheme that uses the (t,n) threshold scheme in [4].

Secret sharing is a very important embranchment of the modern cryptography. Secret sharing is a technique for sharing a secret to a group of participants, each of which holds a portion of the secret. The secret can only be retrieved when a certain number t of members combine their shares together, while any combination with fewer than t shares has no extra information about the secret than 0 shares.

Since Shamir [4] and Blakley [5] firstly proposed their own secret sharing schemes in 1979 respectively, the issues of secret sharing are studied widely in the last several decades. The secret sharing scheme offers a method for the management of secret key in the fields of economy, military affairs and so on. Suppose in a bank, for the reason of security, a strong room can only be opened when three managers get together. Or the problem of the missiles launch, it need at least two generals who turn their keys together to launch missiles. Thus secret sharing scheme is very useful in practices. In this paper, secret sharing scheme is used to study the Distributed storage problem.

This paper is organized as follows. Section 2 describes the threshold scheme. In Section 3, the distributed storage scheme is proposed. Section 4 and section 5 present the results and the conclusion. Finally, the matlab source codes are presented in the section 7.

2. Threshold Schemes

In this section, I will introduce the threshold schemes in details. The first threshold scheme [4] was invented in 1979 by Shamir and named as the Shamir threshold scheme. At the same year, Blakley [5] presented another secret sharing scheme which was similar with Shamir's. Shamir method could be regarded as a special case of the Blakley method, but it took some advantages of requiring less information to be carried by each participant. The Shamir threshold scheme will be presented in the following subsection.

Before the introduction of threshold schemes, let's define what the threshold scheme is firstly.

Definition. Let t, n be positive integers with $t \le n$. A (t, n) threshold scheme is a method of sharing a message M among a group of n members. The message can only be retrieved when a certain number t of members combine their shares together, while any combination with fewer than t shares has no extra information about the secret than 0 shares.

2.1. Shamir threshold scheme

Shamir threshold scheme [4], also called Lagrange interpolation scheme, was firstly invented by Shamir in 1979. The basic idea of Shamir's scheme is based that two points are needed to determine a line, three points to determine a quadratic and so on.

Suppose we have a prime p, which is larger than all the possible message and also larger than the number n of participants. All the calculations are carried out mod p. Here, we can also use a composite number n, however, it will not guarantee the matrices we obtain might have inverses.

Suppose we want to share a secret message M, which is represented as a number mod p. There are n persons and when any t persons get together, they can retrieve the secret by their sharing parts. To achieve this aim, we need the following steps.

1. Select $t-1$ integers mod p randomly, represented as $s_1, s_2, \ldots s_{t-1}$.
2. Create a polynomial $s(x) = M + s_1 x + \cdots + s_{t-1} x^{t-1} \pmod{p}$
3. Select n different integers $x_1, x_2, \ldots x_n \pmod{p}$ for n participants, and calculate the corresponding values $s(x_1), s(x_2), \ldots, s(x_t)$. Then each participant is assigned a pair (x_i, y_i) with $y_i \equiv s(x_i) \pmod{p}$
4. Keep the polynomial as secret and publish the prime p.

So far, the generation of Shamir's secret sharing is described. In the next, the secret recovery scheme will be presented. New suppose t persons want to recover the secret and their pairs are $(x_1, y_1), \ldots, (x_t, y_t)$.

1. Create t equations according to the pairs as following

$$y_k \equiv M + s_1 x_k^1 + \cdots + s_{t-1} x_k^{t-1} \pmod{p}, 1 \le k \le t$$

2. Rewrite these equations into a matrix equation as

$$\begin{pmatrix} 1 & x_1 & \ldots & x_1^{t-1} \\ 1 & x_2 & \ldots & x_2^{t-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_t & \ldots & x_t^{t-1} \end{pmatrix} \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{t-1} \end{pmatrix} \equiv \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_t \end{pmatrix} \pmod{p}$$

where $s_0 = M$. If the determinant of the left matrix V (also called Vandermonde matrix) is nonzero mod p, this equation always has a unique solution mod p.

3. If we get the solution for this equation, we can obtain the secret $M = s_0$.

For step 2, we can find that the determinant of V is

$$\det V = \prod_{1 \le j < k \le t} (x_k - x_j) \pmod{p}$$

Since the prime p is very large, as long as we have distinct x_k's, the equation has a unique solution.

Now, let's describe an alternative approach to retrieve the secret message. Firstly, let

$$l_k(x) = \prod_{\substack{i=1 \\ i \neq k}}^{t} \frac{x - x_i}{x_k - x_i} \pmod{p}$$

By using the method of fractions mod p, we can get

$$l_k(x_j) = \begin{cases} 1 \ if \ k = j \\ 0 \ if \ k \neq j \end{cases}$$

Finally, we can obtain the **Lagrange interpolation polynomial**

$$p(x) = \sum_{k=1}^{t} y_k l_k(x)$$

which satisfies the requirement $p(x_j) = y_j$ for $1 \le j \le t$. Then if we want to get the secret message, we can simply calculate $p(x)$ and evaluate it at $x = 0$. We can get the universal formula for any secret message M as the following,

$$M \equiv \sum_{k=1}^{t} y_k \prod_{\substack{j=1 \\ j \neq k}}^{t} \frac{-x_j}{x_k - x_j} \pmod{p}$$

3. Distributed Storage Based on Secret Sharing Schemes

In this section, I will present how to use the Shamir threshold scheme to accomplish Distributed Storage. An gray image is used to illustrate this algorithm, since values of byte and pixel are both between 0 and 255. By this scheme, the original image will be split into n different parts and any combination of t different parts can recover the image without errors. This scheme is called (t, n) threshold schemes in this paper. For any other file, we can simply transform each byte from binary number into decimal number, and then we can use the blow scheme to finish distributed storage.

3.1. Review of Shamir threshold scheme

The Shamir threshold scheme will be employed in this secret image sharing scheme. Firstly, let's shortly review the Shamir threshold scheme.

In the (t,n) Shamir threshold scheme, a polynomial $s(x) \equiv M + s_1 x + \cdots + s_{t-1} x^{t-1} \pmod{p}$ is constructed firstly, where M is the message and $s_1, s_2, \ldots, s_{t-1}$ are selected randomly. Then n distinct integers $x_1, x_2, \ldots, x_n$ are given and their corresponding values $s(x_i)$ are calculated. Each participant holds a pairs (x_i, y_i) with $y_i \equiv s(x_i)$. When t people get together and share their pairs, then the message M can be easily obtained by the formula

$$M \equiv \sum_{k=1}^{t} y_k \prod_{\substack{j=1 \\ j \neq k}}^{t} \frac{-x_j}{x_k - x_j} \pmod{p}$$

Based on the above Shamir threshold scheme, the Secret image sharing scheme will be presented in the next subsection.

3.2. Secret sharing scheme for gray images

Suppose that we can split a secret image I into n different shadow images ($S_0, S_1, S_2, \ldots S_{n-1}$). Then the original image can only be recovered by the combination of t different shadow images. To accomplish this goal, the polynomial in Shamir threshold scheme is employed. In Shamir threshold scheme, the coefficients $s_0, s_1, s_2, \ldots s_{n-1}$ are randomly selected from integers less than p. However, in the secret image sharing scheme, values of each pixel are used as these coefficients.

Since the value of a gray image is between 0 and 255, the prime number $p = 257$ is selected as the closest prime number to 255. Then the image with size $H \times W$ is divided into m parts and each part contains t pixel. For example, if the size of an image is 256×256 and $t = 4$, we will get $m = 16384$ parts. For each part k, a polynomial $s_k(x) = s_0 + s_1 x + \cdots + s_{t-1} x^{t-1} \pmod{p}$ is defined, where $s_0, s_1, \ldots s_{t-1}$ correspond to the values of each pixel in part k. According to this polynomial, we can get an array A_x containing m values $s_1(x), s_2(x), \ldots s_m(x)$ for a given x. If we reshape this array A_x into a $a \times b$ matrix, a shadow image will be obtained. Moreover, if we choose n different $x's$, we can get n different shadow images. In this scheme, the number t must be selected appropriately, so that m is an integer and can be represented as $a \times b$.

Here is the summary of this secret sharing scheme for gray images.

1. Select a number t and reshape the image into a new image B with m rows and t columns.
2. For each row k, a result $s_k(x) = s_0 + s_1 x + \cdots + s_{t-1} x^{t-1} \pmod{p}$ is calculated for a given x, where $s_0, s_1, \ldots s_{t-1}$ correspond to the values of each pixel in the row k and $p = 257$. The all results of m rows form an array A_x.

3. Reshape the array A_x into a $a \times b$ matrix, then a shadow image with the size of $a \times b$ will be obtained.
4. Change the value of x, and repeat step 2 and 3, until n different shadow images are generated.

In this scheme, the value of x is simply equal to the index of the shadow image, which means for the 6th shadow image, the value of x is also equal to 6.

3.3. Image Recovery based on shadow images

Suppose n different shadow images are gotten from a (t, n) secret image sharing scheme. Then any combination of t different shadow images will yield a recovered image. We define an array $X = \{x_0, x_1, x_{t-1}\}$ to store the value of x for the selected t shadow images. The recovery scheme will be presented in the followings.

1. Reshape all the t shadow images from $a \times b$ matrices into arrays with m length.
2. Read the i-th element of each array as $s_i(X_l)$, where l indicates the index of different arrays and X_l corresponds to l-th element of the array X. Then we can get t values, such as $s_i(X_0), s_i(X_1), \ldots, s_i(X_{t-1})$
3. By using these t values, we can get the following equations,

$$
\begin{aligned}
s_i(X_0) &\equiv s_0^i + s_1^i X_0 + \cdots + s_{t-1}^i {X_0}^{t-1} \pmod{257} \\
s_i(X_1) &\equiv s_0^i + s_1^i X_1 + \cdots + s_{t-1}^i {X_1}^{t-1} \pmod{257} \\
&\vdots \\
s_i(X_{t-1}) &\equiv s_0^i + s_1^i X_{t-1} + \cdots + s_{t-1}^i {X_{t-1}}^{t-1} \pmod{257}
\end{aligned}
$$

Solve these equations together, we can get values of $s_0^i, s_1^i, \ldots s_{t-1}^i$, which are just pixel values at the i-th rows of the reshaped original image B.

4. Repeat step 2 and 3, until all the elements of each array has been calculated. Then we can get the whole reshaped image B.
5. Reshape the image B from $m \times t$ into $H \times W$. Then we will get the recovered image.

3.4. Analysis of Security

Suppose we want to recover an image from a (t, n) secret image sharing scheme, then we need construct $m = \dfrac{H \times W}{t}$ polynomials, such as $s_i(X_0) = s_0^i + s_1^i X_0 + \cdots + s_{t-1}^i {X_0}^{t-1} \pmod{257}$, for an image with the size $H \times W$. For each polynomial, there are t unknown coefficients such as $s_0^i, s_1^i, \ldots s_{t-1}^i$. Suppose, we only have $t-1$ shadow images, then we construct the following $t-1$ equations.

$$s_i(X_0) \equiv s_0^i + s_1^i X_0 + \cdots + s_{t-1}^i X_0^{t-1} \pmod{257}$$
$$s_i(X_1) \equiv s_0^i + s_1^i X_1 + \cdots + s_{t-1}^i X_1^{t-1} \pmod{257}$$
$$\vdots$$
$$s_i(X_{t-2}) \equiv s_0^i + s_1^i X_{t-2} + \cdots + s_{t-1}^i X_{t-2}^{t-1} \pmod{257}$$

In this situation, there are t unknown coefficients but only $t-1$ equations, which means we cannot know what exactly the t-th root is. The probability, one can guess a correct value, is $1/256$. Therefore, the probability, one can recover the whole image without errors is $(1/256)^m$. For that reason, it is impossible to recover an image with the combination of $t-1$ or less shadow images.

3.5. Problem and Solution

We all know that the value of pixels is between 0 and 255. Since I use 257 as the base number, it is possible that the value of pixels is equal to 256 for some shadow image. For this problem, I provide the following solution. For each pixel, if the value is equal to or larger than 255, this value will be stored by two bytes. The value 255 will be stored as one byte 255 and one byte 0, while the value 256 will be stored as one byte 255 and one byte 1. When we are reading the data of shadow images, if current one byte is 255, we will read the next byte and add the value of this byte to 255 as the final value. By this method, we can store each pixel efficiently. Regarding the display of shadow image, all value of 256 will be displayed as value 255.

Since the value of byte is between 0 and 255, the above sharing scheme is valid for any file stored based on byte.

4. Results

In this section, the image peppers is used to show the result of secret image sharing algorithm in practices. The image peppers, shown in Figure 2, is a 256 pixel width, 256 pixel height and 8-bits gray image. The value of each pixel is an integer between 0 and 255. Matlab codes are designed to compute these results and are attached in section 8.

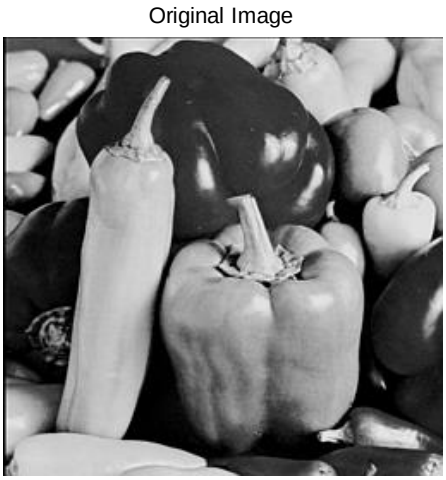

Figure 2. The original image peppers

In this section, we use a $(4,8)$ secret image sharing scheme, where $t=4$ and $n=8$. By using the secret image sharing scheme described in section 3, we generate 8 shadow images which are shown in Figure 3. Since $t=4$ and the size of original image is 512×512, the size of each shadow image is 128×128. Since I use the $(4,8)$ secret image sharing scheme, I can use any four combinations of shadow images in the Figure 3 to get recovered image. The recover image is shown in Figure 4. By comparing with the original image, there is no error between the original image and the recovered image.

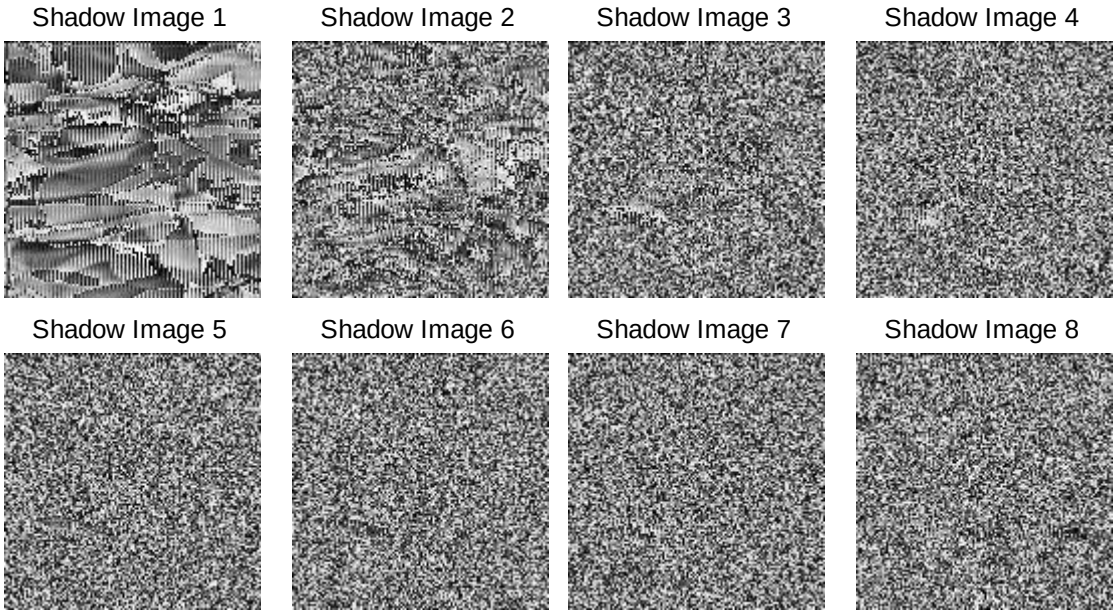

Figure 3. The 8 shadow images.

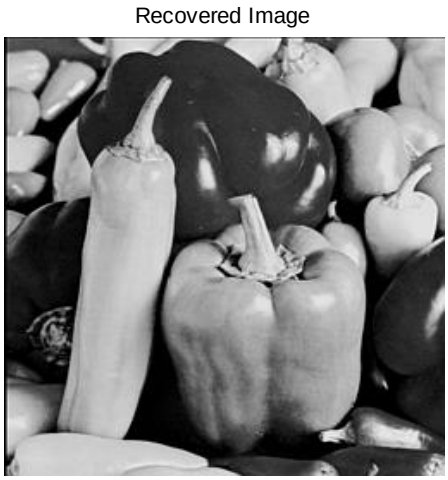

Figure 4. The recovered Image obtained from the shadow images by using the secret image sharing scheme.

5. Conclusion

In this paper, I first review the development of secret sharing schemes and introduce the threshold schemes in details. Furthermore, I apply the secret sharing schemes to study the distributed storage problem. By using the Shamir threshold scheme, an image is split into n small shadow images. With any combination of t shadow images ($t \le n$), the original image will be recovered. Then I prove that this scheme can be used in any file. The results show that secret sharing scheme is valid for solving the distributed storage problem.

6. References

[1] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *IEEE Internet Computing*, pp. 40–49, September 2001.

[2] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total recall: System support for automated availability management," in *NSDI*, 2004.

[3] F. Dabek, J. Li, E. Sit, J. Robertson, M. Kaashoek, and R. Morris, "Designing a dht for low latency and high throughput," 2004.

[4] Shamir. How to Share a Secret Communications of the ACM, 1979, 22(11): 612-613

[5] G. R. Blakley. Safeguarding Cryptographic Keys. Proc. AFPS 1979, National Computer Conference, 1979, 48: 313-3 17

7. Source codes of Matlab for secret image sharing in this paper

```
%% Main Function
function ImageSharing
clear; close all;
Height = 256;
Width =256;
t = 4;
```

```
w = 8;
Users = [1,3,5,8];
base = 257;
Sub_width  = Width/t^(0.5);
Sub_height = Height/t^(0.5);
if(length(Users)<t)
    error('invalid length of Users');
end;
if(mod(Sub_height,1) ~= 0)
    error('invalid length of Sub height');
end;
Im_Name = ' peppers.bin';
[Im_linear, Im_Square] = ReadImage(Im_Name,Height,Width);
P_Im = GetPreparedIm(Im_linear,t);
Shadow_Image = GetShadow_Image(P_Im,w,base);
ShowShadowImage(Shadow_Image,Sub_width,Sub_height);
RequiredShadowImage = GetRequiredShadowImage(Shadow_Image,Users);
RecoveredImage =
RecoverImage(RequiredShadowImage,Users,Height,Width,base);

figure;
imshow(uint8(RecoveredImage));
title('Recovered Image');

%% function RecoveredImage = RecoverImage(RequiredShadowImage)
function RecoveredImage =
RecoverImage(RequiredShadowImage,Users,Height,Width,base)
TT = size(RequiredShadowImage);
xx = ones(TT(2),TT(2));
for i = 1:TT(2)
   xx(:,i) = xx(:,i).*(Users.^(i-1))';
end;
for i = 1:TT(1)
    if(mod(i,100)==0)
        disp(sprintf('%d of %d',i,TT(1)))
    end;
    RecoveredImage(i,:) = solveEq(xx,RequiredShadowImage(i,:),base);
end;
RecoveredImage = reshape(RecoveredImage(:),Height,Width)';

%% function b = solveEq(Users,RequiredShadowImage(i,:),TT(2),base);
function b = solveEq(xx,yy,base)
b = GetIntMod(inv(sym(xx))*yy',base)';
%% function Re_b = GetIntMod(b,base)
function Re_b = GetIntMod(b,base)
[n,d] = numden(b);
n = double(n);
d = double(d);
Re_b = mod(n.*powermod(d,-1,base),base);

%% function RequiredShadowImage = GetRequiredShadowImage();
function RequiredShadowImage =
```

```
GetRequiredShadowImage(Shadow_Image,Users)
TT = length(Users);
for i = 1:TT
    RequiredShadowImage(:,i) = Shadow_Image(:,Users(i));
end;

%% function ShowShadowImage
function ShowShadowImage(Shadow_Image,Sub_width,Sub_height)
TT = size(Shadow_Image);
for i =1:TT(2)
    figure;
    imshow(uint8(reshape(Shadow_Image(:,i),Sub_width,Sub_height)));
    cmd = sprintf('title(''Shadow Image %d\'')',i);
    eval(cmd);
end;

%% function get Sharing_Polynomial
function Shadow_Image = GetShadow_Image(P_Im,w,base)
TT = int32(size(P_Im));
P_Im = int32(P_Im);
w=int32(w);
base = int32(base);
Shadow_Image = int32(zeros(TT(1),w));
for i=1:w
    for j = 1:TT(2)
        Shadow_Image(:,i) = Shadow_Image(:,i) + mod(P_Im(:,j)*i^(j-1),
base);
    end;
end;
 Shadow_Image = mod(Shadow_Image,base);

%% function get PreparedIm
function P_Im = GetPreparedIm(Im_linear,t)
if(mod(length(Im_linear),t) ~= 0)
    error('invalid length')
end;
P_Im = reshape(Im_linear,[],t);

%% function Read Image
function [Im_data, Im]=ReadImage(Im_Name,Height,Width)
Im_fp = fopen(Im_Name,'rb');
Im_data = fread(Im_fp,'uint8');
h = figure;   % create a new figure
Im = uint8(reshape(Im_data,Height,Width)');
imshow(Im);
title('Original Image');

%% function powermod
function y = powermod(a,z,n)
% This function calculates y = a^z mod n
% If a is a matrix, it calculates a(j,k)^z mod for every element in a
[ax,ay]=size(a);
% If a is negative, put it back to between 0 and n-1
```

```
a=mod(a,n);
% Take care of any cases where the exponent is negative
if (z<0),
   z=-z;
   for j=1:ax,
      for k=1:ay,
         a(j,k)=invmodn(a(j,k),n);
      end;
   end;
end;

for j=1:ax,
for k=1:ay,
  x=1;
  a1=a(j,k);
  z1=z;
  while (z1 ~= 0),
     while (mod(z1,2) ==0),
        z1=(z1/2);
        a1=mod((a1*a1), n);
     end;  %end while
     z1=z1-1;
     x=x*a1;
     x=mod(x,n);
  end;
  y(j,k)=x;
end; %end for k
end; %end for j
function y = invmodn( b,n);
% This function calculates the inverse of an element b mod n
% It uses the extended euclidean algorithm
n0=n;
b0=b;
t0=0;
t=1;
q=floor(n0/b0);
r=n0-q*b0;
while r>0,
   temp=t0-q*t;
   if (temp >=0),
      temp=mod(temp,n);
   end;
   if (temp < 0),
      temp= n - ( mod(-temp,n));
   end;
   t0=t;
   t=temp;
   n0=b0;
   b0=r;
   q=floor(n0/b0);
   r=n0-q*b0;
end;
if b0 ~=1,
```

```
   y=[];
   disp('No inverse');
else
   y=mod(t,n);
end;
```