Chapter 4

Simple and Efficient Secret Sharing Schemes

4.1 Introduction

Secret sharing is a new alternative for outsourcing data in a secure way. It avoids the need for time consuming encryption decryption process and also the complexity involved in key management. The data must also be protected from untrusted cloud service providers. Secret sharing based solution provides secure information dispersal by making shares of the original data and distribute them among different servers. Data from the threshold number of servers can be used to reconstruct the original data. It is often impractical to distribute data among large number of servers. We have to achieve a trade off between security and efficiency. An optimal choice is to use a (2,3) or (2,4) threshold secret sharing scheme, where the data are distributed as shares among three or four servers and shares...
from any two can be used to construct the original data. This provides both security, reliability and efficiency. We propose some efficient and easy to implement secret sharing schemes in this regard based on number theory and bitwise XOR. These schemes are also suitable for secure sharing of images. Secret image sharing based on Shamir’s schemes are lossy and involves complicated Lagrange interpolation. So the proposed scheme can also be effectively utilized for lossless sharing of secret images.

Confidentiality, reliability and efficiency are the major concerns in secure storage of data. The idea of secret sharing for the information dispersal is suggested by Krawczyk et al [124] in 1994. He proposed a computationally secure secret sharing scheme for the distributed storage using Rabin’s [175] information dispersal algorithm and Shamir’s secret sharing scheme. However the data is encrypted using a symmetric key encryption and the shares of the key are distributed along with the data shares. The share size is less than the secret in this case compromising the information theoretic security. Abhishek Parak et al [163] in 2010 proposed a space efficient secret sharing scheme for the implicit data security. They incorporated \( k - 1 \) secrets in \( n \) shares and any \( k \) shares can be used to reconstruct the original secret. A recursive construction using Shamir’s scheme is applied in which computational overhead is more. Recursive methods of secret sharing is also mentioned in [82] [162]. Computational secret sharing schemes are proposed for the space efficiency in [8] [179] [214].

Secret sharing based solution provides information theoretical security on confidentiality with out encryption and hence avoid the complexities associated with encryption and key management. It also provides the guarantee on availability of data. Perfect secret sharing needs large amount of computational overhead. We propose specially designed secret sharing schemes using XOR and number theoretic technique to reduce the computation overhead. Unanimous consent schemes are easy to implement
4.2 Proposed Secret Sharing Schemes

using XOR. But the implementation of a general \((t, n)\) threshold scheme is difficult. Wang et al [215] proposed a scheme based on boolean operation which is used for secret image sharing in 2007. Kurihara et al [125] [126] proposed a \((3, n)\) and a generalized \((t, n)\) secret sharing scheme based on simple XOR operations. Efficient and ideal threshold scheme based on XOR is proposed by Lv et al [140] in 2010. Secret sharing using number theoretic schemes are also developed based on Chinese Reminder Theorem (CRT) [2] [105] [146]. They are not widely used because of the computational complexity. The proposed scheme make use of simple number theoretic concept and the extended Euclid’s algorithm [129].

4.2 Proposed Secret Sharing Schemes

The proposed system suggests a method of storing and retrieving private data in a secure and effective manner. The private data include personal information, sensitive information or unique identification etc. The data storage may be a public information storage such as cloud storage server. We propose number theoretic and XOR based scheme for efficient implementation of secret sharing schemes. It can be used for secure storage and retrieval of secret information. Since it does not involve any encryption, the PKI needed for key management can be avoided. Section 4.2.1 contains the detailed description of the secret sharing algorithm using number theoretic concept. Section 4.2.2 explains the XOR based schemes. The algorithms mentioned below are designed to share a file one byte at a time. The scheme can be used to share both textual data and images.
4.2.1 Schemes Based On Number Theory

In this section, the proposed secret sharing schemes which are based on number theoretic concepts are explained in detail. Two threshold secret sharing schemes of order \((2, 3)\) and \((2, 4)\) are proposed. The Algorithm 4.1 is the \((2, 3)\) secret sharing scheme. The retrieval algorithms depend on which shares are used for the reconstruction and are given in Algorithms 4.2, 4.3, 4.4. A \((2, 4)\) secret sharing scheme is mentioned in Algorithm 4.5. The secret revealing algorithms correspond to different combinations of shares are given in Algorithms 4.6, 4.7, 4.8, 4.10 and 4.11. These algorithms use simple number theory concept. In order to find the inverse of a number, extended Euclid’s algorithm [39] can be used. The share generation and the secret revealing can be done with a complexity of \(O(n)\), where \(n\) is the number of bytes to share. Table lookup can be used for faster performance.
Algorithm 4.1: (2,3) Secret Sharing: Number Theory

**Input:** Input file $S$ to share.

**Output:** Three Shares $S_1, S_2, S_3$ of same size as the original file.

1. Choose a field $Z_p$ where $p = 257$.
2. while not at end of the input file do
   
   /* read a byte or pixel */
   
   $s = \text{read\_byte}(S)$
   
   if $s == 0$ then
      
      $s = 256$
   
   end
   
   /* find cube root of $s$ */
   
   $a = s^{1/3} \pmod{p}$
   
   $r = \text{random}(257)$ /* $r$ is a random number between 1-256 */
   
   /* generate $s_1$, the share1 pixel */
   
   if $s_1 == 256$ then
      
      $s_1 = 0$
   
   end
   
   /* generate $s_2$, the share2 pixel */
   
   $s_2 = r^2 \times a \pmod{p}$
   
   if $s_2 == 256$ then
      
      $s_2 = 0$
   
   end
   
   /* generate $s_3$, the share3 pixel */
   
   $s_3 = r^4 \times a \pmod{p}$
   
   if $s_3 == 256$ then
      
      $s_3 = 0$
   
   end

end
**Algorithm 4.2: (2,3) Secret Revealing: Number Theory** $S_1 S_2$

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Choose a field $\mathbb{Z}_p$ where $p = 257$.</td>
</tr>
<tr>
<td>2</td>
<td>while not at end of the input files do</td>
</tr>
<tr>
<td>3</td>
<td>/* read a byte or pixel from $S_1$ */</td>
</tr>
<tr>
<td>4</td>
<td>/* read a byte or pixel from $S_2$ */</td>
</tr>
<tr>
<td>5</td>
<td>if $s_1 == 0$ then</td>
</tr>
<tr>
<td>6</td>
<td>$s_1 = 256$</td>
</tr>
<tr>
<td>7</td>
<td>end</td>
</tr>
<tr>
<td>8</td>
<td>if $s_2 == 0$ then</td>
</tr>
<tr>
<td>9</td>
<td>$s_2 = 256$</td>
</tr>
<tr>
<td>10</td>
<td>end</td>
</tr>
<tr>
<td>11</td>
<td>$a = s_1^2 \times s_2^{-1} \pmod{p}$</td>
</tr>
<tr>
<td>12</td>
<td>$s = a^3 \pmod{p}$</td>
</tr>
<tr>
<td>13</td>
<td>/* $s$ is the secret data byte or pixel */</td>
</tr>
<tr>
<td>14</td>
<td>if $s == 256$ then</td>
</tr>
<tr>
<td>15</td>
<td>$s = 0$</td>
</tr>
<tr>
<td>16</td>
<td>end</td>
</tr>
</tbody>
</table>

**Input:** Shares $S_1$ and $S_2$

**Output:** The original secret file $S$ which is shared
Algorithm 4.3: (2, 3) Secret Revealing: Number Theory $S_1S_3$

**Input:** Shares $S_1$ and $S_3$

**Output:** The original secret file $S$ which is shared

1. Choose a field $Z_p$ where $p = 257$.

2. While not at end of the input files do

   /* read a byte or pixel from $S_1$ */
   $s_1 = \text{read}_\text{byte}(S_1)$

   /* read a byte or pixel from $S_3$ */
   $s_3 = \text{read}_\text{byte}(S_3)$

3. if $s_1 == 0$ then
   4. $s_1 = 256$
   5. end

4. if $s_3 == 0$ then
   5. $s_3 = 256$
   6. end

7. $s = s_1^4 \times s_3^{-1} \pmod{p}$

   /* $s$ is the secret data byte or pixel */

8. if $s == 256$ then
   9. $s = 0$
   10. end

11. end
Algorithm 4.4: (2, 3) Secret Revealing: Number Theory $S_2S_3$

**Input:** Shares $S_2$ and $S_3$

**Output:** The original secret file $S$ which is shared

1. Choose a field $\mathbb{Z}_p$ where $p = 257$.

2. while not at end of the input files do

   /* read a byte or pixel from $S_2$ */
   3. $s_2 = \text{read\_byte}(S_2)$

   /* read a byte or pixel from $S_3$ */
   4. $s_3 = \text{read\_byte}(S_3)$

   if $s_2 == 0$ then
   5. $s_2 = 256$
   6. end

   if $s_3 == 0$ then
   7. $s_3 = 256$
   8. end

11. $a = s_2^2 \times s_3^{-1} \pmod{p}$

   /* $s$ is the secret data byte or pixel */
   12. $s = a^3 \pmod{p}$

13. if $s == 256$ then

14. $s = 0$

15. end

16. end
4.2. Proposed Secret Sharing Schemes

Algorithm 4.5: (2, 4) Secret Sharing:Number Theory

Input: Input file $S$ to share.
Output: Four Shares $S_1, S_2, S_3, S_4$ of same size as the original file.

1. Choose a field $\mathbb{Z}_p$ where $p = 257$.
2. while not at end of the input file do
3. 
4. $s = \text{read_byte}(S)$ /* read a byte or pixel */
5. 
6. if $s == 0$ then
7. 
8. $s = 256$
9. 
10. end
11. 
12. $r = \text{random}(257)$ /* $r$ is a random number between 1-256 */
13. /* $s_1$ is the share1 pixel */
14. 
15. $s_1 = r$
16. 
17. if $s_1 == 256$ then
18. 
19. $s_1 = 0$
20. 
21. end
22. /* $s_2$ is the share2 pixel */
23. 
24. $s_2 = r \times s \pmod{p}$
25. 
26. if $s_2 == 256$ then
27. 
28. $s_2 = 0$
29. 
30. end
31. /* $s_3$ is the share3 pixel */
32. 
33. $s_3 = r^2 \times s \pmod{p}$
34. 
35. if $s_3 == 256$ then
36. 
37. $s_3 = 0$
38. 
39. end
40. /* $s_4$ is the share4 pixel */
41. 
42. $s_4 = r^3 \times s \pmod{p}$
43. 
44. if $s_4 == 256$ then
45. 
46. $s_4 = 0$
47. 
48. end
49. end
Chapter 4. Simple and Efficient Secret Sharing Schemes

Algorithm 4.6: (2, 4) Secret Revealing: Number Theory $S_1 S_2$

| Input: Shares $S_1$ and $S_2$ |
| Output: The original secret file $S$ which is shared |

1. Choose a field $Z_p$ where $p = 257$.

2. while not at end of the input files do
   /* read a byte or pixel from $S_1$ */
   $s_1 = \text{read}_\text{byte}(S_1)$

   /* read a byte or pixel from $S_2$ */
   $s_2 = \text{read}_\text{byte}(S_2)$

3. if $s_1 == 0$ then
   $s_1 = 256$

4. if $s_2 == 0$ then
   $s_2 = 256$

5. end

6. $s = s_1^{-1} \times s_2 \pmod{p}$

7. if $s == 256$ then
   $s = 0$

8. end

9. end
4.2. Proposed Secret Sharing Schemes

Algorithm 4.7: $(2, 4)$ Secret Revealing: Number Theory $S_1S_3$

<table>
<thead>
<tr>
<th>Input: Shares $S_1$ and $S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: The original secret file $S$ which is shared</td>
</tr>
</tbody>
</table>

1. Choose a field $Z_p$ where $p = 257$.

2. while not at end of the input files do

   /* read a byte or pixel from $S_1$ */
   $s_1 = \text{read}_\text{byte}(S_1)$

   /* read a byte or pixel from $S_3$ */
   $s_3 = \text{read}_\text{byte}(S_3)$

3. if $s_1 == 0$ then

   $s_1 = 256$

4. if $s_3 == 0$ then

   $s_3 = 256$

5. end

6. end

7. $s = (s_1^2)^{-1} \times s_3 \pmod p$

8. if $s == 256$ then

   $s = 0$

9. end

10. end
Algorithm 4.8: (2, 4) Secret Revealing; Number Theory $S_1 S_4$

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Choose a field $Z_p$ where $p = 257$.</td>
</tr>
<tr>
<td>2</td>
<td>while not at end of the input files do</td>
</tr>
<tr>
<td>3</td>
<td>$s_1 = $read_byte($S_1$) /* read a byte or pixel from $S_1$ */</td>
</tr>
<tr>
<td>4</td>
<td>$s_4 = $read_byte($S_4$) /* read a byte or pixel from $S_4$ */</td>
</tr>
<tr>
<td>5</td>
<td>if $s_1 == 0$ then</td>
</tr>
<tr>
<td>6</td>
<td>$s_1 = 256$</td>
</tr>
<tr>
<td>7</td>
<td>end</td>
</tr>
<tr>
<td>8</td>
<td>if $s_4 == 0$ then</td>
</tr>
<tr>
<td>9</td>
<td>$s_4 = 256$</td>
</tr>
<tr>
<td>10</td>
<td>end</td>
</tr>
<tr>
<td>11</td>
<td>$s = (s_1^3)^{-1} \times s_4$ (mod $p$) /* $s$ is the secret byte or pixel */</td>
</tr>
<tr>
<td>12</td>
<td>if $s == 256$ then</td>
</tr>
<tr>
<td>13</td>
<td>$s = 0$</td>
</tr>
<tr>
<td>14</td>
<td>end</td>
</tr>
<tr>
<td>15</td>
<td>end</td>
</tr>
</tbody>
</table>
4.2. Proposed Secret Sharing Schemes

Algorithm 4.9: (2,3) Secret Revealing: Number Theory $S_2S_3$

**Input:** Shares $S_2$ and $S_3$

**Output:** The original secret file $S$ which is shared

1. Choose a field $\mathbb{Z}_p$ where $p = 257$.
2. while not at end of the input files do
3.   $s_2$ = read_byte($S_2$) /* read a byte or pixel from $S_2$ */
4.   $s_3$ = read_byte($S_3$) /* read a byte or pixel from $S_3$ */
5.   if $s_2$ == 0 then
6.     $s_2$ = 256
7.   end
8.   if $s_3$ == 0 then
9.     $s_3$ = 256
10.  end
11.  $s = s_2^2 \times s_3^{-1}$ (mod $p$)
12.  if $s$ == 256 then
13.     $s$ = 0
14.  end
15. end
Algorithm 4.10: (2,4) Secret Revealing: Number Theory $S_2 S_4$

<table>
<thead>
<tr>
<th>Input: Shares $S_2$ and $S_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: The original secret file $S$ which is shared</td>
</tr>
</tbody>
</table>

1. Choose a field $\mathbb{Z}_p$ where $p = 257$.

2. While not at end of the input files do:
   
   /* read a byte or pixel from $S_2$ */
   
   3. $s_2 = \text{read}_0\text{byte}(S_2)$

   /* read a byte or pixel from $S_4$ */
   
   4. $s_4 = \text{read}_0\text{byte}(S_4)$

   5. if $s_2 == 0$ then
   6.     $s_2 = 256$
   7. end

   8. if $s_4 == 0$ then
   9.     $s_4 = 256$
  10. end

11. $s = \sqrt{s_2^3 \times s_4^{\frac{1}{4}} \pmod{p}}$

   /* $s$ is the secret byte or pixel */

12. if $s == 256$ then
13.     $s = 0$
14. end
15. end
4.2. Proposed Secret Sharing Schemes

**Algorithm 4.11: (2,4) Secret Revealing: Number Theory $S_3S_4$**

**Input:** Shares $S_3$ and $S_4$

**Output:** The original secret file $S$ which is shared

1. Choose a field $Z_p$ where $p = 257$.

2. while not at end of the input files do
   3. $s_3$ = read_byte($S_3$) /* read a byte or pixel from $S_3$ */
   4. $s_4$ = read_byte($S_4$) /* read a byte or pixel from $S_4$ */
   5. if $s_3$ == 0 then
      6. $s_3$ = 256
   7. end
   8. if $s_4$ == 0 then
      9. $s_4$ = 256
   10. end
   11. $s = s_3^3 \times (s_4^2)^{-1} \pmod{p}$
   12. if $s$ == 256 then
      13. $s$ = 0
   14. end
   15. end

4.2.2 Schemes based on XOR

An $(n,n)$ scheme using XOR can easily be setup by creating $n - 1$ random shares of same size as the secret and the $n^{th}$ share as the XOR of these $n - 1$ shares and the secret $k$. The secret can be revealed by simply XOR ing of all these shares. In this, we propose two schemes based on XOR. An ideal (2,3) scheme where the size of the share is same as that of the secret is mentioned in Algorithm 4.16 and a non ideal scheme which is also not perfect is mentioned in Algorithm 4.12. In this the size of the share is reduced to half. The scheme can be used when the storage become a constraint. The secret sharing and revealing can be done in time $O(n)$, where $n$ is the number of bytes to share. The secret reconstruction corresponds to different combination of shares in the non ideal scheme are
Chapter 4. Simple and Efficient Secret Sharing Schemes

mentioned in Algorithms 4.13, 4.14 and 4.15. The ideal schemes are mentioned in Algorithms 4.17, 4.18 and 4.19.

Algorithm 4.12: (2, 3) XOR secret sharing-non ideal

| Input: Secret file S to share. |
| Output: Three shares S1, S2 and S3 of half the size of S. |
| 1 while not at end of the input files do |
| 2 s = read_byte(S) /* read a byte or pixel from S */ |
| 3 bs = binary(s) /* bs is the binary representation of s */ |
| /* odd bits of bs taken as share1 data nibble s1 */ |
| 4 s1 = odd_bits(bs) |
| /* even bits of bs taken as share2 data nibble s2 */ |
| 5 s2 = even_bits(bs) |
| /* share3 nibble is formed by xoring s1 and s2 */ |
| 6 s3 = s1 ⊕ s2 |
| 7 end |

Algorithm 4.13: (2, 3) XOR secret revealing S1,S2-non ideal

| Input: Share S1 and S2 |
| Output: The original secret file S which is shared. |
| 1 while not at end of the input files do |
| 2 s1 = read_byte(S1) /* read a byte or pixel from S1 */ |
| 3 s2 = read_byte(S2) /* read a byte or pixel from S2 */ |
| 4 s = intermix(s1, s2) /* intermix the bits of s1 and s2 to construct the secret byte */ |
| 5 end |
4.2. Proposed Secret Sharing Schemes

Algorithm 4.14: (2, 3) XOR secret revealing $S_1S_3$-non ideal

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>Share $S_1$ and $S_3$</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>The original secret file $S$ which is shared.</td>
</tr>
<tr>
<td><strong>1</strong> while <strong>not at end of the input files</strong> do</td>
<td></td>
</tr>
<tr>
<td><strong>2</strong> $s_1$ = read_byte($S_1$) /* read a byte or pixel from $S_1$ */</td>
<td></td>
</tr>
<tr>
<td><strong>3</strong> $s_3$ = read_byte($S_3$) /* read a byte or pixel from $S_3$ */</td>
<td></td>
</tr>
<tr>
<td><strong>4</strong> $s_2$ = $s_1$ $\oplus$ $s_3$ /* intermix the bits of $s_1$ and $s_2$ to construct the secret byte */</td>
<td></td>
</tr>
<tr>
<td><strong>5</strong> $s$ = intermix($s_1, s_2$)</td>
<td></td>
</tr>
<tr>
<td><strong>6</strong> end</td>
<td></td>
</tr>
</tbody>
</table>

Algorithm 4.15: (2, 3) XOR secret revealing $S_2S_3$-non ideal

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>Share $S_2$ and $S_3$</td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>The original secret file $S$ which is shared.</td>
</tr>
<tr>
<td><strong>1</strong> while <strong>not at end of the input files</strong> do</td>
<td></td>
</tr>
<tr>
<td><strong>2</strong> $s_2$ = read_byte($S_2$) /* read a byte or pixel from $S_2$ */</td>
<td></td>
</tr>
<tr>
<td><strong>3</strong> $s_3$ = read_byte($S_3$) /* read a byte or pixel from $S_3$ */</td>
<td></td>
</tr>
<tr>
<td><strong>4</strong> $s_1$ = $s_2$ $\oplus$ $s_3$</td>
<td></td>
</tr>
<tr>
<td><strong>5</strong> $s$ = intermix($s_1, s_2$) /* intermix the bits of $s_1$ and $s_2$ to construct the secret byte */</td>
<td></td>
</tr>
<tr>
<td><strong>6</strong> end</td>
<td></td>
</tr>
</tbody>
</table>
Algorithm 4.16: (2,3) XOR Ideal Secret Sharing

**Input:** Input file $S$ to share.

**Output:** Three Shares $SH_1, SH_2, SH_3$ of same size as the original file.

1. while not at end of the input file do
2.    $s$=read_byte($S$) /* read a byte or pixel */
3.    $r$=random(257) /* $r$ is a random number between 0-256 */
4.    $s_1, s_2$=split_two($s$) /* split $s$ into 2 nibbles */
5.    $r_1, r_2$=split_two($r$) /* split $r$ into 2 nibbles */
6.    $s_0$ = 0000 /* a dummy variable initialized to zero */
7.    $sh_1 = s_0 \oplus r_1|\|s_2 \oplus r_2$
     /* $sh_1$ is the share1 pixel and ’|’ is concatenation operation */
8.    $sh_2 = s_1 \oplus r_1|\|s_0 \oplus r_2$
     /* $sh_2$ is the share2 pixel and ’|’ is concatenation operation */
9.    $sh_3 = s_2 \oplus r_1|\|s_1 \oplus r_2$
     /* $sh_3$ is the share3 pixel and ’|’ is concatenation operation */
10. end
4.2. Proposed Secret Sharing Schemes

Algorithm 4.17: (2, 3) XOR Ideal Secret Recovery $SH_1, SH_2$

<table>
<thead>
<tr>
<th>Input: Shares $SH_1$ and $SH_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Original secret $S$ that is shared</td>
</tr>
</tbody>
</table>

1 while not at end of the input files do
2    $sh_1$ = read_byte($SH_1$)  /* read a byte or pixel */
3    $sh_2$ = read_byte($SH_2$)
4    $x_1, y_1$ = split_two($sh_1$)  /* split a byte into 2 nibbles */
5    $x_2, y_2$ = split_two($sh_2$)
6    $s_1 = x_1 \oplus x_2$
7    $s_2 = y_1 \oplus y_2$
8    $s = s_1 | s_2$
9 end

Algorithm 4.18: XOR Ideal Secret Recovery $SH_1, SH_3$

<table>
<thead>
<tr>
<th>Input: Shares $SH_1$ and $SH_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Original secret $S$ that is shared</td>
</tr>
</tbody>
</table>

1 while not at end of the input files do
2    $sh_1$ = read_byte($SH_1$)  /* read a byte or pixel */
3    $sh_3$ = read_byte($SH_3$)
4    $x_1, y_1$ = split_two($sh_1$)  /* split a byte into 2 nibbles */
5    $x_3, y_3$ = split_two($sh_3$)
6    $s_2 = x_1 \oplus x_3$
7    $s_1 = y_1 \oplus y_3 \oplus s_2$
8    $s = s_1 | s_2$
9 end
Algorithm 4.19: (2, 3) XOR Ideal Secret Recovery $SH_2, SH_3$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong></td>
<td>Shares $SH_2$ and $SH_3$</td>
<td></td>
</tr>
<tr>
<td><strong>Output:</strong></td>
<td>Original secret $S$ that is shared</td>
<td></td>
</tr>
<tr>
<td><strong>1</strong></td>
<td><strong>while not at end of the input files do</strong></td>
<td></td>
</tr>
<tr>
<td><strong>2</strong></td>
<td>$sh_2$ = read_byte($SH_2$) /* read a byte or pixel */</td>
<td></td>
</tr>
<tr>
<td><strong>3</strong></td>
<td>$sh_3$ = read_byte($SH_3$)</td>
<td></td>
</tr>
<tr>
<td><strong>4</strong></td>
<td>$x_2, y_2$ = split_two($sh_2$) /* split a byte into 2 nibbles */</td>
<td></td>
</tr>
<tr>
<td><strong>5</strong></td>
<td>$x_3, y_3$ = split_two($sh_3$)</td>
<td></td>
</tr>
<tr>
<td><strong>6</strong></td>
<td>$s_1 = y_2 \oplus y_3$</td>
<td></td>
</tr>
<tr>
<td><strong>7</strong></td>
<td>$s_2 = x_2 \oplus x_3 \oplus s_1$</td>
<td></td>
</tr>
<tr>
<td><strong>8</strong></td>
<td>$s = s_1</td>
<td></td>
</tr>
<tr>
<td><strong>9</strong></td>
<td><strong>end</strong></td>
<td></td>
</tr>
</tbody>
</table>

4.3 Conclusion

The confidentiality, availability and performance requirement of storage system is addressed in this chapter. Secret sharing based solutions provides information theoretic security and also provides trust and reliability. We developed simple XOR and number theory based schemes which are easy to implement. This will greatly improve the performance of the system. The storage requirement can also be reduced if we use scheme where the share size is only half the size of the original secret. The schemes mentioned in this chapter are simple and easy to implement when sharing data with third party servers. A (2, 3) or (2, 4) threshold secret sharing schemes are the best choices. The cost factor can also be reduced by using the non ideal XOR based scheme, where the share size is reduced to half but the information theoretic security is compromised. A secret vector which indicates the share number that each server stores can be kept secret. A simple substitution or transposition cipher can also be used as a preprocessing step before
sharing the file for additional security. The use of these schemes can be further explored in other areas where the threshold required is as specified in the algorithm. We have used this schemes for efficient sharing of secret images also.