

Simplest (3,2) Threshold Secret Sharing Schemes Based on EXCLUSIVE-OR Function

Sergei Kulikov



Abstract: This paper presents a new XOR-based secret sharing scheme designed for simplicity, efficiency, and strong security guarantees. Unlike traditional threshold schemes that rely on matrix operations and work on large data blocks, the proposed method operates on a per-byte basis, applying lightweight bitwise XOR calculations. The goal of this research is to create a minimalistic yet robust (3,2) threshold secret sharing algorithm that is practical for real-world systems with constrained computational resources. The proposed scheme divides a secret file into three independent shadow shares using a simple transformation: each share is generated as the XOR of the secret with a random byte string and a shifted version of that string. Specifically, the algorithm uses one random string and constructs the second and third shares by XOR-ing the secret with the random string and with the string shifted by one byte in opposite directions. This process is applied independently to each byte of the secret file. The resulting scheme is both perfect and ideal: each share is the same size as the original file, and no information about the secret can be obtained from a single share. Any two out of three shares are sufficient to reconstruct the original data, while fewer reveal nothing. The encoding and decoding procedures are transparent, computationally efficient, and well-suited for implementation in hardware or constrained environments. The algorithm was implemented in Fortran and can be easily integrated into secure cloud storage systems, distributed communication protocols, or embedded devices. This work contributes a lightweight alternative to existing XOR-based sharing methods, providing a novel balance between cryptographic strength and algorithmic simplicity.

Keywords: Secret Sharing. Cryptographic Protocols, Real-Time Systems

Abbreviations:

SSS: Secret Sharing Schemes

I. INTRODUCTION

Secret sharing schemes (SSS) based on the Exclusive-OR (XOR) operation have been explored in several studies [1]. XOR-based (k,n)-threshold schemes allow secure

distribution of a secret among n participants such that any k or more can reconstruct the secret, while fewer than k reveal no information. These schemes are ideal, meaning the size of each share equals the size of the secret, and are computationally efficient due to the lightweight XOR operation [2]. However, most existing XOR-based schemes operate on the whole secret or large blocks, often involving matrix operations [3]. We propose a more memory-efficient alternative that works byte-by-byte.

II. PROCEDURE

We present a (3,2) threshold XOR-based scheme with a streaming, byte-level approach. Each byte of the secret is processed independently, resulting in low memory overhead and fast execution.

Let

$S = (s_1, s_2, \dots, s_t)$ is a secret file of t bytes.

$W_j = (w_1, w_2, \dots, w_t)$ are 3 shadows, $j=1,2,3$

$\oplus$ denotes a bit-wise exclusive-OR operation.

The procedures for Share and Recover are as follows.

A. Sharing Procedure ($S \rightarrow W_1, W_2, W_3$)

1. Generate an initial random byte: $x_0 = \text{random}(0-255)$, $w_{10} = x_0$,
2. For each byte $i=1$ to t $x_i = \text{random}(0-255)$, $w_{1i} = x_i$, $w_{2i} = s_i \oplus x_i$, $w_{3i} = s_i \oplus x_{i-1}$

Thus

- the first shadow W_1 is just the random string of length $t+1$,
- the second shadow W_2 is the XOR of the secret and the first shadow of length t ,
- The third W_3 shadow is the XOR of the secret and the first shadow shifted back by one byte, of length t .

The sharing procedure is illustrated in Figure 1.

Figure 1. Sharing the procedure scheme.

Secret		S_1	S_2	S_3	S_4	...	S_t	
Shadow 1	w_0	w_1	w_2	w_3	w_4	...	w_t	<- random numbers
Shadow 2		$S_1 \oplus w_1$	$S_2 \oplus w_2$	$S_3 \oplus w_3$	$S_4 \oplus w_4$	...	$S_t \oplus w_t$	
Shadow 3		$S_1 \oplus w_0$	$S_2 \oplus w_1$	$S_3 \oplus w_2$	$S_4 \oplus w_3$	...	$S_t \oplus w_{t-1}$	

Manuscript received on 25 July 2025 | First Revised Manuscript received on 02 August 2025 | Second Revised Manuscript received on 17 October 2025 | Manuscript Accepted on 15 November 2025 | Manuscript published on 30 November 2025.

*Correspondence Author(s)

Sergei Kulikov*, Head, Department of Biostatistics, National Research Center for Hematology, Novozykovsky pr. 4A, MOSCOW, Russian Federation (RUS), Russia. Email ID: Smkulikov@mail.ru, ORCID ID: [0000-0002-6288-7570](https://orcid.org/0000-0002-6288-7570)

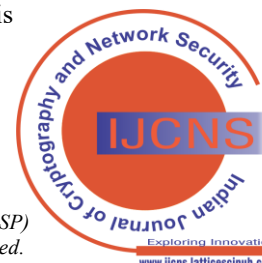
© The Authors. Published by Lattice Science Publication (LSP). This is an open-access article under the CC-BY-NC-ND license (<http://creativecommons.org/licenses/by-nc-nd/4.0/>)

B. Recovery Procedures

1. $W_1, W_2 \rightarrow S$. If we recode the secret by using the first and second shadows, we just combine them by XOR:

For $i=1$ to t $s_i = w_{1i} \oplus w_{2i}$

2. $W_1, W_3 \rightarrow S$. If we recode the secret by using the first and third shadows, we just combine them by XOR. First shadow is shifted by one bite back:



Simplest (3,2) Threshold Secret Sharing Schemes Based on EXCLUSIVE-OR Function

- For $i=1$ to t $s_i = w_{1i-1} \oplus w_{3i}$
3. $W_2, W_3 \rightarrow S$. If we recode the secret by using the second and third shadows, the calculations are a little more complicated.

For $i=1$ to t

- $y_i = w_{2i} \oplus w_{3i}$
- $x_i = y_i \oplus x_{i-1}$
- $s_i = x_i \oplus w_{2i}$

III. PROPERTIES OF THE SCHEME

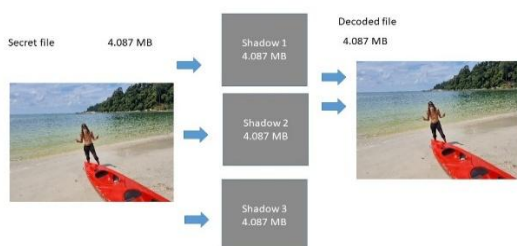
The proposed SSS is

- Ideal: The size of each share equals the size of the secret.
- Perfect secrecy: No information is leaked from a single share.
- Efficient: Uses only XOR and operates byte-by-byte without matrices.
- Redundant: Any two out of three shares suffice to reconstruct the original secret.

IV. EXPERIMENTAL RESULTS

The proposed algorithm was implemented in the FORTRAN programming language. To verify correctness, an image file was encoded and decoded. Since all bytes, including format metadata, were encrypted, shadows cannot be visualized. (Figure 2).

Figure 2: Example of an image file encoded and decoded by the proposed SSS.



A performance test was conducted on a directory containing 1,977 files, totalling 8.6 GB.

Results:

- Sharing time: 27.9 minutes (including I/O); 4.6 minutes (pure computation)
 - Recovery time: 33.8 minutes (including I/O); 0.8 minutes (pure computation)
 - System: Intel® Core™ i7-10510U laptop
- For comparison, a basic Shamir's scheme took:
- Sharing: 35.3 minutes (6.4 pure)
 - Recovery: 34.6 minutes (1.8 pure)

V. CONCLUSION

We presented a simple and efficient (3,2) XOR-based secret sharing scheme operating at the byte level. The algorithm offers high speed and minimal memory usage, making it suitable for secure storage, transfer, and real-time applications.

DECLARATION STATEMENT

I must verify the accuracy of the following information as the article's author.

- **Conflicts of Interest/ Competing Interests:** Based on my understanding, this article has no conflicts of interest.
- **Funding Support:** This article has not been funded by any organizations or agencies. This independence ensures that the research is conducted with objectivity and without any external influence.
- **Ethical Approval and Consent to Participate:** The content of this article does not necessitate ethical approval or consent to participate with supporting documentation.
- **Data Access Statement and Material Availability:** The adequate resources of this article are publicly accessible.
- **Author's Contributions:** The authorship of this article is contributed solely by the author.

REFERENCES

1. Chen, L., Laing, T.M., Martin, K.M. (2016). Efficient, XOR-Based, Ideal (t,n)-threshold Schemes. In: Foresti, S., Persiano, G. (eds) Cryptology and Network Security. CANS 2016. Lecture Notes in Computer Science(), vol 10052. Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-48965-0_28
2. Wang H, Jiang Z, Qian Q, Wang H. An efficient XOR-based visual cryptography scheme with lossless reconfigurable algorithms. International Journal of Distributed Sensor Networks. 2022;18(4). DOI: <http://doi.org/10.1177/15501329221084223>
3. Shima K., Doi H. Hierarchical Secret-Sharing Scheme Based on XOR Operations, Journal of Information Processing, September 2024, 32:719-730, DOI: <http://doi.org/10.2197/ipsjip.32.719>

AUTHOR'S PROFILE



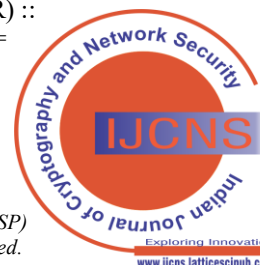
Sergei Kulikov, PhD, Head, Department of Biostatistics, National Medical Research Center of Hematology, Ministry of Health of the Russian Federation, Moscow. Education: Moscow Physical Technical Institute of Science, Russia, Diploma in Applied Mathematical Statistics in 1975. Institute of Control Science, USSR Academy of Sciences, Moscow, PhD in Machine Learning, 1986. Research specialties and interests: Biostatistics & Statistical Epidemiology, Population Haematology, Data Science Analytics. Special Skills: SAS programming and analytics. Committee: Member of the Boarding Committee of the Russian Haematology Society, Member of the Biostatistics working party of the European Leukaemia Network.

Current h-index = 20

Appendix A. Fortran Implementation of the XOR-Based Secret Sharing Scheme

The following FORTRAN program demonstrates an implementation of a simple (3,2) XOR-based secret sharing scheme. It reads binary files from a specified source directory, splits each file into three shadow shares using a bitwise XOR operation, and saves each share into a separate output directory. The implementation ensures that any two of the three shares can reconstruct the original data, while a single share reveals no information. This version includes inline comments for clarity and was used to verify the algorithm's performance and correctness during testing.

```
program main
  implicit none
  ! Constants and array declarations
  integer, parameter :: NR = 1000
  character, dimension(NR) :: IX = "?"
  character, dimension(NR) ::
  IXa = "!", IXb = "!", IXc =
  "!"
  integer inbite, outbite,
  sha1, sha2, sha3
```



```
integer, dimension(NR) :: CX = 999
integer i, j, k, ios, lest, icr, ipos, endc, ifile
integer, parameter :: Nf = 3
integer, parameter :: NN = 16
integer:: re_i, r, kk, ind, reason, NstationFiles, iStatio
integer :: lendir, randJ, r1, r2, r3, n, r0, cot
integer: a(NN)
real: f(NN)
real: ra, randn, t_start, t_stop, t_sum
integer: rr(130)

! Directory and file name buffers
CHARACTER (40): fromdir
CHARACTER (40): todir1, todir2, todir3
CHARACTER (100): filename0, filename1, filename2,
filename3
character (LEN=100), dimension (:), allocatable: station
File Names

! Read input/output directories from file
OPEN (unit=15, file="config.txt", status='old')
read (15, *) fromdir
read (15, *) todir1
read (15, *) todir2
read (15, *) todir3
close (15)

! Copy input files to each of the three output directories
call system ('CHCP 1251 >nul')
call system ('xcopy ' //TRIM (fromdir)//' //TRIM
(todir1)//' /H /Y /C /R /S /I /E /T ')
call system('xcopy ' //TRIM(fromdir)//'
//TRIM(todir2)//' /H /Y /C /R /S /I /E /T ')
call system('xcopy ' //TRIM(fromdir)//'
//TRIM(todir3)//' /H /Y /C /R /S /I /E /T ')

! Get list of files from source directory
call system ('dir //TRIM (fromdir)//' /a: -d/s/b/l >
fileContents.txt')
lendir = len(trim(fromdir)) + 1

open (31, file='fileContents.txt', action="read")
i = 0
do
read (31, '(a)', iostat=reason) ra
if (reason /= 0) exit
i = i + 1
end do
NstationFiles = i
allocate(stationFileNames(NstationFiles))
rewind (31)

! Process each file individually
t_sum = 0
do ifile = 1, NstationFiles
read (31, '(a)') stationFileNames(ifile)
filename0 = stationFileNames(ifile)
filename1 = trim(todir1) // stationFileNames(ifile)
(lendir:)
filename2 = trim(todir2) // stationFileNames(ifile)
(lendir:)
filename3 = trim(todir3) //
stationFileNames(ifile)(lendir:)

OPEN (unit=1, file=filename0, status='old',
ACCESS='stream', form='unformatted')
```

```
OPEN (unit=2, file=filename1, status='new',
ACCESS='stream')
OPEN (unit=3, file=filename2, status='new',
ACCESS='stream')
OPEN (unit=4, file=filename3, status='new',
ACCESS='stream')

do i = 1, 90000
IX = "-"
CX = 999
icr = 0
ipos = 0
endc = NR

! Read raw bytes into IX []
do j = 1, NR
read (1, iostat=ios) IX(j)
if (ios /= 0) exit
end do
endc = j - 1

call cpu_time(t_start)
r0 = 111! Initial random seed

do j = 1, endc
! Encode each byte of the secret using two random
keys
inbite = ichar(IX(j)) + 1
call random_number(randn)
r = FLOOR (255 * randn)
r1 = r
r2 = xor(inbite, r0)
r3 = xor(inbite, r)
r0 = r

! Store shadows
IXa(j) = char(r1)
IXb(j) = char(r2)
IXc(j) = char(r3)
end do

call cpu_time(t_stop)
t_sum = t_sum + (t_stop - t_start)

! Write the three shadows to files
write (2) (IXa(j), j=1, endc)
write (3) (IXb(j), j=1, endc)
write (4) (IXc(j), j=1, endc)

if (ios < 0) exit
end do

close (1)
close (2)
close (3)
close (4)
end do

close (31)
write (*, *) "Total processing time (seconds):", t_sum
9999 write (*, *) "<<<<<<< End of program"
end program main
```

Simplest (3,2) Threshold Secret Sharing Schemes Based on EXCLUSIVE-OR Function

Appendix B. Fortran Implementation of the Secret Recovery Procedure

This appendix provides the FORTRAN implementation for recovering the original data from two of the three shadow shares produced by the XOR-based secret sharing scheme. Depending on the selected rule, the program reads different combinations of shadow files. It applies XOR operations, optionally using sequential shifts and an initial random seed, to reconstruct the original binary file. The method ensures correct recovery when any two shares are available.

```
! A fortran95 program for G95
! By WQY
program main
  implicit none
  integer re_i, i, j, k, icr, ios, endc, xi, yi, icode, ai, zi
  integer r1, r2, r3, rr, fi1, fi2, inbite, outbite, ifile
  integer nstationfiles, reason, rule
  integer lendir0, lendir1, lendir2, lendir3
  integer, parameter :: NR = 1000
  integer outcode(256,256), vvv(1000)
  character, dimension(NR) :: IXa="!", IXb="!", IXout
  character(40) :: fromdir, todir1, todir2, todir3
  character(100) :: filename1, filename2, filename3,
  filename0, ra
  character(LEN=100), dimension(:), allocatable ::
  stationFileNames
  real :: t_start, t_stop, t_sum

  ! Read input/output directories and rule from config
  open (15, file=" config.txt", status='old')
  read (15, *) fromdir
  read (15, *) todir1
  read (15, *) todir2
  read (15, *) todir3
  read (15, *) rule
  close (15)

  ! Copy directory structure
  call system ('CHCP 1251 >nul')
  call system ('xcopy " // trim(todir1) // " " //
  trim(fromdir) // " /H /Y /C /R /S /I /E /T')

  ! Get list of files to process
  call system ('dir " // trim(todir1) // " /a: -d/s/b/l >
  fileContents.txt')
  lendir0 = len(trim(fromdir)) + 1
  lendir1 = len(trim(todir1)) + 1
  lendir2 = len(trim(todir2)) + 1
  lendir3 = len(trim(todir3)) + 1

  open (31, file='fileContents.txt', action="read")
  i = 0
  do
    read(31, '(a)', iostat=reason) ra
    if (reason /= 0) exit
    i = i + 1
  end do
  nstationfiles = i
  allocate(stationFileNames(nstationfiles))
  rewind(31)

  t_sum = 0
  do ifile = 1, nstationfiles
    read(31, '(a)') stationFileNames(ifile)
```

```
    filename0 = trim(fromdir) //
    stationFileNames(ifile)(lendir1:)
    filename1 = trim(todir1) //
    stationFileNames(ifile)(lendir1:)
    filename2 = trim(todir2) //
    stationFileNames(ifile)(lendir2:)
    filename3 = trim(todir3) //
    stationFileNames(ifile)(lendir3:)
    open(1, file=filename0, status='new', access='stream')
    open(2, file=filename1, status='old', access='stream',
    form='unformatted')
    open(3, file=filename2, status='old', access='stream',
    form='unformatted')
    open(4, file=filename3, status='old', access='stream',
    form='unformatted')

    ! Determine which shadows to use based on the rule
    if (rule == 1) then
      fi1 = 2
      fi2 = 3
    else if (rule == 2) then
      fi1 = 2
      fi2 = 4
    else if (rule == 3) then
      fi1 = 3
      fi2 = 4
    end if

    do i = 1, 10000
      endc = NR
      do j = 1, NR
        read(fi1, iostat=ios) IXa(j)
        if (ios < 0) exit
        read (fi2, iostat=ios) IXb(j)
      end do
      endc = j - 1
      call cpu_time(t_start)

      ! Apply selected recovery logic
      if (rule == 1) then
        do j = 1, endc
          if (j == 1) then
            xi = 111
          else
            xi = ichar(IXa(j-1))
          end if
          yi = ichar(IXb(j))
          icode = xor(xi, yi)
          IXout(j) = char(icode)
        end do

      else if (rule == 2) then
        do j = 1, endc
          xi = ichar(IXa(j))
          yi = ichar(IXb(j))
          icode = xor(xi, yi)
          IXout(j) = char(icode)
        end do

      else if (rule == 3) then
        do j = 1, endc
          xi =
          ichar(IXa(j))
```



```

        yi = ichar(IXb(j))
        r1 = xor(xi, yi)
        if (j == 1) then
            r2 = xor(r1, 111)
        else
            r2 = xor(r1, rr)
        end if
        rr = r2
        icode = xor(yi, rr)
        IXout(j) = char(icode)
    end do
end if

call cpu_time(t_stop)
t_sum = t_sum + (t_stop - t_start)
write(1) (IXout(j), j=1, endc)
end do

close(1)
close(2)
close(3)
close(4)
end do

write(*,*) "Sum time = ", t_sum
write(*,*) "End program."
end program main

```