SW-SSS: Slepian-Wolf Coding-based Secret Sharing Scheme

Kazumasa Omote and Tran Phuong Thao

Japan Advanced Institute of Science and Technology (JAIST),
1-1 Asahidai, Nomi, Ishikawa, Japan, 923-1292
{omote, tpthao}@jaist.ac.jp

Abstract. A secret sharing scheme is a method for protecting distributed file systems against data leakage and for securing key management systems. The secret is distributed among a group of participants where each participant holds a share of the secret. The secret can be only reconstructed when a sufficient number of shares are reconstituted. Although many secret sharing schemes have been proposed, these schemes have not achieved an optimal share size and have not supported the share repair feature. This paper proposes a secret sharing scheme based on the Slepian-Wolf coding, named the SW-SSS, to obtain an optimal share size and to provide the share repair without recovering the secret. Furthermore, the share in the SW-SSS is constructed using the exclusive-OR (XOR) operation for fast computation. Unlimited parameters are also supported in the SW-SSS. To the best of our knowledge, we are the first applying the Slepian-Wolf coding to a secret sharing scheme.

Keywords: Secret Sharing Scheme, Slepian-Wolf Coding, XOR Network Coding

1 Introduction

Distributed data mining has become very useful today with the increase in the amount of data. This in turn increases the need to preserve the privacy of the participants. Two approaches which can be used in privacy preserving data mining are encryption and secret sharing. The secret sharing is recently being considered as a more efficient approach because it does not require expensive operations (i.e., modular exponentiation of large numbers) and does not require key managements like the encryption approach [1–3]. Thus, improving secret sharing is our motivation in this paper.

Secret sharing schemes (SSS) are ideal for storing information that is sensitive. The basic ideas of SSS were introduced by Shamir [4] and Blakley [5]. A secret S is encoded into n shares. Each participant receives one share. This is known as the (m, n)-threshold SSS in which any m or more shares can be used to reconstruct the secret and in which the bit-size of a share is the same as the bit-size of the secret. The efficiency of the scheme is evaluated by the entropy of each share, and it must hold that $H(C_i) \geq H(S)$ where $H(S)$ and $H(C_i)$ are the entropies of a secret S and shares C_i $(i \in \{1, \cdots, n\})$ [6, 7]. To improve the efficiency of the Shamir-SSS, the Ramp-SSS was proposed [8–12] with a trade-off between security and coding efficiency. In a (m, L, n)-Ramp-SSS, the secret S can be reconstructed from any m or more shares; no information about S can be obtained from less than L shares; but a partial information of S can be leaked from any arbitrary set of $(m-t)$ shares with equivocation $\frac{t}{m}H(S)$ for $t \in \{1, \cdots, m-L\}$. It can attain that $H(C_i) = H(S)/(m-L)$, and thus the Ramp-SSS is more efficient than the Shamir-SSS [8,9]. The drawback of these SSSs is the heavy computational cost because the shares are constructed using polynomials. An example of a polynomial is the Reed-Solomon code, which takes $O(n \log n)$ field operations for the secret distribution and $O(m^2)$ field operations for the secret reconstruction [13]. To address this drawback, the SSSs based on the XOR operation (XOR-SSS) were proposed to replace the polynomials. Examples include: the $(2, 3)$-SSS [14], the $(2, n)$-SSS [15], the $(3, n)$-SSS [16], and the (m, n)-SSS where $m = \{2, 3, 4, n-3, n-2, n-1\}$ [13]. However, the threshold (m, n) in these XOR-SSSs is limited. Therefore, the (m, n)-SSSs where m and n are arbitrary were presented in [17–20]. Kurihara et al. then improved [18] to achieve the Ramp-SSS in [21].

Most of these schemes can support the secret reconstruction but cannot support the direct share repair feature. This means that when a share is corrupted, without the direct share repair, $\mathcal{D}$ must reconstruct the secret S before generating the new share to replace the corruption. If the direct share repair is supported, the corrupted share can be repaired directly from the remaining healthy shares without the need to reconstruct S. Breaking with the flow of previous schemes, the SSS based on the network coding (NC-SSS) was introduced [22, 23] to deal with the direct share repair. However, the schemes were constructed using a linear combination instead of the XOR. This paper shows that a special network coding based on the XOR [24–26], can be applied for SSS to address the drawbacks. The resulting scheme

is called the XORNC-SSS. This paper then shows that another coding named the Slepian-Wolf coding (SWC) [27, 28], which is used to compress a data stream in a network, can be also applied for SSS to reduce the share size while still satisfying the benefits of the XORNC-SSS.

Contribution. This paper revisits the XOR network coding and applies it to the SSS (called XORNC-SSS). The XORNC-SSS has the following four advantages: (i) the shares are constructed using the XOR for fast computation; (ii) (m, n) can be chosen arbitrarily; (iii) the direct share repair is supported; (iv) the size of a share is smaller than the size of the secret.

This paper then mainly proposes the SSS based on the Slepian-Wolf coding (called SW-SSS). The share size in the SW-SSS is surprisingly less than the share size in the XORNC-SSS. In other words, the SW-SSS improves the fourth advantage of the XORNC-SSS while still satisfying the first three advantages of the XORNC-SSS.

Roadmap. The backgrounds of SSS, network coding and Slepian-Wolf coding are described in Section 2. The XORNC-SSS and SW-SSS schemes are presented in Section 3 and 4. The properties and the efficiency analysis are given in Section 5 and 6. The conclusion and future work are drawn in Section 7.

2 Background

2.1 Secret Sharing Scheme (SSS)

Shamir-SSS. The Shamir-SSS [4, 5] consists of n participants $\mathcal{P} = \{P_1, \cdots, P_n\}$ and a dealer $\mathcal{D}$. Two algorithms $\mathsf{ShareGen}$ and $\mathsf{Reconst}$ are run by $\mathcal{D}$. The share generation algorithm $\mathsf{ShareGen}$ inputs a secret S and outputs a set $C = \{c_1, \cdots, c_n\}$. c_i $(i \in \{1, \cdots, n\})$ is called a *share* and is given to a participant P_i. The secret reconstruction algorithm $\mathsf{Reconst}$ inputs a set of m shares and outputs the secret S. A (m, n)-Shamir-SSS has the following properties:

- perfect SSS: any $m \leq n$ participants or more can reconstruct the secret S and no $(m-1)$ participants or less can learn any information of the secret S. Formally, let $H(S)$ and $H(A)$ denote the entropy of the secret S and a set of shares $A \subseteq C$, respectively.

$$H(S|A) = \begin{cases} 0, & \text{if } |A| \geq m \\ H(S), & \text{if } |A| < m \end{cases}$$

- ideal SSS: the size of a share is the same as the size of the secret: $|c_i| = |S|$.

Ramp-SSS. The Ramp-SSS [8–12] was proposed to improve the coding efficiency of the Shamir-SSS. The Ramp-SSS has three parameters (m, L, n) and is constructed in a way that any m shares or more can reconstruct the secret S. Any set of $(m - t)$ shares where $t \in \{1, \cdots, m - L\}$ can learn a partial information of the secret S. Any L shares or less cannot obtain any information of the secret S. Formally,

$$H(S|A) = \begin{cases} H(S), & \text{if } |A| < L \\ \frac{m-|A|}{m}, & \text{if } L \leq |A| < m \\ 0, & \text{if } |A| \geq m \end{cases}$$

The Ramp-SSS is more efficient than the Shamir-SSS because in any (m, L, n)-Ramp-SSS, $H(C_i) = H(S)/(m - L)$ [8, 9].

2.2 Network Coding (NC)

The NC [29–32] was proposed to improve the data transmission efficiency. Suppose that a source node wants to send a message M to a receiver node. The source node firstly divides M into m blocks: $M = v_1 || \cdots || v_m$. $v_i \in \mathbb{F}_q^l$ $(i = \{1, \cdots, m\})$ where $\mathbb{F}_q^l$ denotes a l-dimensional finite field $\mathbb{F}$ over a large prime q. Before transmitting, the source node augments v_i with the vector of length m which contains a single '1' in the i-th position and '0' elsewhere. A resulting augmented block has the following form:

$$w_i = (\overbrace{\underbrace{0, \cdots, 0, 1}_{i}, 0, \cdots, 0}^{m}, v_i) \in \mathbb{F}_q^{l+m}.$$

Then, the source node sends the m augmented blocks as packets to the network. Each node in the network linearly combines the packets it receives and transmits the resulting linear combination to its

adjacent nodes. Suppose that an intermediate node receives t packets. The intermediate node randomly chooses t coefficients $\alpha_1, \cdots, \alpha_t \in \mathbb{F}_q$ and linearly combines the received packets by $c = \sum_{i=1}^{t} \alpha_i w_i$. The intermediate node then sends c to the network. Therefore, the receiver node can receive the linear combinations of the augmented blocks. Assume that the receiver node receives m packets $y_1, \cdots, y_m \in \mathbb{F}_p^{l+m}$ which are the linear combinations of $w_1, \cdots, w_m$. The receiver node can find $v_1, \cdots, v_m$ using the accumulated coefficients contained in the first m coordinates of y_i.

2.3 Slepian-Wolf Coding (SWC)

The SWC [27] was proposed to compress data in a network. The SWC has several approaches: syndrome-based, binning idea, LDPC-based and parity-based [28, 33, 34]. For the most efficient computation, this paper uses the *binning idea*. Suppose that the source has two data b_1 and b_2 which have the same size. To compress, b_1 is divided into a number of bins. During encoding, the index of the bin that the input belongs to is transmitted to the receiver instead of the input itself. For example, $|b_1|$ is divided into k bins. Each bin contains $\frac{|b_1|}{k}$ elements. If the SWC is not used, $\log_2 |b_1|$ bits are required to transmit the input to the receiver. If the SWC is used, only $\log_2 k$ bits are required. The receiver cannot decode b_1 if b_2 does not present because the receiver cannot know the corresponding element of the bin index. If b_2 is obtained, the receiver can decode b_1 by picking the element in the bin that is best matched with b_2.

2.4 Notations

Throughout this paper, the following notations are used:
- S denotes the secret.
- $\mathcal{D}$ denotes the dealer.
- m denotes the number of secret blocks (the first threshold).
- b_i denotes a secret block where $i \in \{0, \cdots, m-1\}$.
- n denotes the number of participants (the second threshold).
- L denotes the number of participants whose shares are collectively constructed from m secret blocks.
- P_i denotes a participant where $i \in \{0, \cdots, n-1\}$.
- c_i denotes a share stored in P_i.
- s_i denotes the XOR used to construct c_i ($s_i = b_j \oplus b_t \oplus b_z$).
- d_i denotes the metadata of c_i (the number of '1' bits in s_i).
- j denotes the index of the first operand of s_i.
- t denotes the index of the second operand of s_i.
- z denotes the index of the third operand of s_i.
- $|b|$ denotes the bit-size of a secret block ($|b| = \frac{|S|}{m}$).
- $\oplus$ denotes the XOR operator.
- $||$ denotes the concatenation operator.

3 The XORNC-SSS scheme

The XOR-based NC is revisited in this section. Although the XOR-based NC exits in previous network coding schemes [24–26], the new thing here is that the construction is applied to a (m, n)-SSS instead of network coding. No previous scheme considers this applying.

$\mathcal{D}$ firstly divides S into m blocks: $S = b_0||\cdots||b_{m-1}$ ($|b_i| = \frac{|S|}{m}$) and encodes S into n shares. Each participant P_i holds a share c_i ($i \in \{0, \cdots, n-1\}$). To compute c_i, $\mathcal{D}$ chooses secret blocks randomly and combines them using the XOR. $\mathcal{D}$ then pads that XOR with a vector of length m which contains a '1' bit in the index of each chosen secret block and $(m-1)$ '0' bits elsewhere. The padded vector is called the coefficient of c_i. Suppose that c_i is constructed from t secret blocks $b_{i_0}, \cdots, b_{i_{t-1}}$. Let $s_i = b_{i_0} \oplus \cdots \oplus b_{i_{t-1}}$. c_i has the following form:

$$c_i = (\underbrace{a_0, a_2, \cdots, a_{m-1}}_{m}, \underbrace{s_i}_{\frac{|S|}{m}})$$

where $a_i = 1$ if $i \in \{i_0, \cdots, i_{t-1}\}$ and $a_i = 0$ elsewhere. The share size is $|c_i| = m + \frac{|S|}{m}$. The ideal property of a SSS is $|c_i| = |S|$ (Section 2.1). The XORNC-SSS achieves a better share size if $|c_i| \leq |S|$. From this inequality, $(m - \frac{|S|}{2})^2 \leq \frac{|S|^2}{4} - |S|$. Because $|S|$ is large in a real system, $\frac{|S|^2}{4} - |S| \approx \frac{|S|^2}{4}$. Therefore, $m \leq |S|$. In other words, if the parameters are chosen s.t. $m \leq |S|$, the scheme can reduce the

share size. Furthermore, the coefficients are chosen s.t. the matrix consisting of the coefficients of any m shares has rank m. This condition is to ensure that m secret blocks can be reconstructed from any m shares. To reconstruct S, $\mathcal{D}$ chooses any m shares to find m secret blocks, then concatenates them together. To repair a corrupted share, $\mathcal{D}$ requires m healthy shares to reconstitute using the XOR.

Example. Suppose that $S = b_0||b_1||b_2$ and $n = 4$. $\mathcal{D}$ creates the shares $c_0 = (1, 1, 1, b_0 \oplus b_1 \oplus b_2)$, $c_1 = (1, 1, 0, b_0 \oplus b_1)$, $c_2 = (1, 0, 1, b_0 \oplus b_2)$, $c_3 = (1, 0, 0, b_0)$. $\mathcal{D}$ sends $\{c_0, \cdots, c_3\}$ to the participants $\{P_0, \cdots P_3\}$, respectively. To reconstruct S, $\mathcal{D}$ chooses $m = 3$ shares (suppose c_0, c_2, c_3) and constructs the following equation system:

$$\begin{cases} s_0 = b_0 \oplus b_1 \oplus b_2 \\ s_2 = b_0 \oplus b_2 \\ s_3 = b_0 \end{cases}$$

Then, $\{b_0, b_1, b_2\}$ are computed using the Gaussian elimination. $S = b_0||b_1||b_2$. Suppose that $\mathcal{P}_2$ is corrupted, $\mathcal{D}$ requires $\mathcal{P}_0, \mathcal{P}_1$ and $\mathcal{P}_3$ to provide s_0, s_1 and s_3. $\mathcal{D}$ repairs s_2 by $s_2 = s_0 \oplus s_1 \oplus s_3$.

4 The proposed SW-SSS scheme

In this scheme, a share c_i does not have the same form as in the XORNC-SSS. Instead, c_i is the index of the bin that the XOR belongs to. This scheme focuses on the share generation, secret reconstruction and share repair. Checking a corrupted participant is beyond the scope of this paper. Several existing schemes can be used: homomorphic MAC [32, 35] or homomorphic signature [36, 37].

4.1 Share Generation (ShareGen)

Each XOR is constructed from three different secret blocks. From m secret blocks, there are $\binom{m}{3}$ XORs. Because only n out of $\binom{m}{3}$ XORs are required for n participants, these n XORs are chosen s.t. the index sequence of the three secret blocks is a permutation of the proper set $\{0, \cdots, m-1\}$ in an ascending order. Each XOR itself is also sorted in an ascending order. Namely, $\mathcal{D}$ chooses the shares for n participants from the following XORs respectively, until $\mathcal{C}$ has enough n XORs:

$$\begin{gathered} (b_0 \oplus b_1 \oplus b_2), (b_0 \oplus b_1 \oplus b_3), \cdots, (b_0 \oplus b_1 \oplus b_{m-1}), \\ (b_0 \oplus b_2 \oplus b_3), (b_0 \oplus b_2 \oplus b_4), \cdots, (b_0 \oplus b_2 \oplus b_{m-1}), \\ \cdots \\ (b_1 \oplus b_2 \oplus b_3), (b_1 \oplus b_2 \oplus b_4), \cdots, (b_1 \oplus b_2 \oplus b_{m-1}), \\ \cdots \end{gathered}$$

Algorithm 1: ShareGen

```
   Input  : m, n, |S|
   Output: {c_0, d_0}, ··· , {c_{n-1}, d_{n-1}}
 1 S = b_0|| ··· ||b_{m-1};
 2 count ← 0;
 3 for j ← 0 to m − 3 do
 4 |   for t ← j + 1 to m − 2 do
 5 |   |   for z ← t + 1 to m − 1 do
 6 |   |   |   s_i ← b_j ⊕ b_t ⊕ b_z;
 7 |   |   |   d_i ← s_i.count('1');
 8 |   |   |   M_i ← ListAnagram(|b|, d_i) ;
 9 |   |   |   c_i ← index(M_i, s_i) ;
10 |   |   |   count + +;
11 |   |   |   if (count == n − 1) then
12 |   |   |   |   return
   |   |   |   |   {c_0, d_0}, ··· , {c_{n-1}, d_{n-1}}
13 |   |   |   end
14 |   |   end
15 |   end
16 end
```

Given m, n, $|S|$, the ShareGen (Algorithm 1) outputs n pairs of share c_i and its metadata d_i. $\mathcal{D}$ firstly divides $S = b_0|| \cdots ||b_{m-1}$ (line 1). The block size is $|b| = \frac{|S|}{m}$. $\mathcal{D}$ computes the XOR $s_i = b_j \oplus b_t \oplus b_z$ for each share (line 6). $\mathcal{D}$ finds the number of '1' bits in s_i, denoted by d_i (line 7). $\mathcal{D}$ constructs a set M_i consisting of all permutations of each XOR (line 8) using ListAnagram (the pseudo code of this function is omitted because it is supported in many programming languages). The elements in M_i are sorted in an ascending order. $\mathcal{D}$ finds the corresponding index of s_i in M_i, denoted by c_i (line 9). Observe that the share is not s_i but the index of s_i in the set M_i. The number of elements in M_i is $|M_i| = \binom{|b|}{d_i}$. The number of bits for representing a share is at most $\log_2 |M_i|$. The bandwidth and the storage cost can be reduced because the size of an index is less than the size of a XOR. ShareGen finally returns $\{c_0, d_0\}, \cdots, \{c_{n-1}, d_{n-1}\}$. $\mathcal{D}$ distributes $\{c_i, d_i\}$ to the participant $\mathcal{P}_i$.

Example 4-1. Suppose that $S = 10100111001110110001$. S is divided into $m = 5$ blocks: $b_0 = 1010, b_1 = 0111, b_2 = 0011, b_3 = 1011$ and $b_4 = 0001$ ($|S| = 20$, $|b_i| = 4$). Suppose that $n = 8$, the shares are

Table 1: Example 4-1

$\mathcal{P}_0$	$s_0 = b_0 \oplus b_1 \oplus b_2 = 1110$	$d_0 = 3$	$M_0 = \{0111, 1011, 1101, 1110\}$	$c_0 = 3_{dec} = 11$
$\mathcal{P}_1$	$s_1 = b_0 \oplus b_1 \oplus b_3 = 0110$	$d_1 = 2$	$M_1 = \{0011, 0101, 0110, 1001, 1010, 1100\}$	$c_1 = 2_{dec} = 10$
$\mathcal{P}_2$	$s_2 = b_0 \oplus b_1 \oplus b_4 = 1100$	$d_2 = 2$	$M_2 = \{0011, 0101, 0110, 1001, 1010, 1100\}$	$c_2 = 5_{dec} = 101$
$\mathcal{P}_3$	$s_3 = b_0 \oplus b_2 \oplus b_3 = 0010$	$d_3 = 1$	$M_3 = \{0001, 0010, 0100, 1000\}$	$c_3 = 1_{dec} = 1$
$\mathcal{P}_4$	$s_4 = b_0 \oplus b_2 \oplus b_4 = 1000$	$d_4 = 1$	$M_4 = \{0001, 0010, 0100, 1000\}$	$c_4 = 3_{dec} = 11$
$\mathcal{P}_5$	$s_5 = b_0 \oplus b_3 \oplus b_4 = 0000$	$d_5 = 0$	$M_5 = \{\}$	$c_5 = 0_{dec} = 0$
$\mathcal{P}_6$	$s_6 = b_1 \oplus b_2 \oplus b_3 = 1111$	$d_6 = 4$	$M_6 = \{1111\}$	$c_6 = 0_{dec} = 0$
$\mathcal{P}_7$	$s_7 = b_1 \oplus b_2 \oplus b_4 = 0101$	$d_7 = 2$	$M_7 = \{0011, 0101, 0110, 1001, 1010, 1100\}$	$c_7 = 1_{dec} = 1$

$\{c_0, \cdots, c_7\}$. To construct c_0, $s_0 = b_0 \oplus b_1 \oplus b_2 = 1110$ is used. Because the number of '1' bits in s_0 is 3, $d_0 = 3$. Because $d_0 = 3$ and $|b_i| = 4$, $M_0 = \{0111, 1011, 1101, 1110\}$. The elements in M_0 are sorted in an ascending order and are indexed as $\{0, \cdots, 3\}$. Because $|M_0| = \binom{4}{3} = 4$, at most $\log_2 4 = 2$ bits are required to represent c_0 instead of 4 bits of s_0. Because the index of s_0 in M_0 is 3, $c_0 = 3_{(decimal)} = 11$. $\{c_0, d_0\}$ are sent to the participant $\mathcal{P}_0$. Similarly, $\{c_1, \cdots c_7\}$ are computed as in Table 1.

4.2 Secret Reconstruction (Reconst)

To reconstruct S, $\mathcal{D}$ requires m participants to provide their shares (suppose c_{k_0}, $\cdots$, $c_{k_{m-1}}$). These shares are chosen s.t. the binary matrix consisting of the coefficient vectors of the XORs has full rank. Given m pairs of $\{c_{k_i}, d_{k_i}\}$, the Reconst algorithm (Algorithm 2) outputs m secret blocks $\{b_0, \cdots, b_{m-1}\}$. For each c_{k_i}, $\mathcal{D}$ firstly lists all permutations given $|b|$ and d_{k_i} (line 2). $\mathcal{D}$ finds the XOR s_{k_i} by picking the c_{k_i}-th element in the set M_{k_i} (line 3). $\mathcal{D}$ executes LocateIndices to find the indices of three operands of s_{k_i} (line 4). $\mathcal{D}$ constructs a vector v_{k_i} consisting of $m+1$ elements: m first elements are the binary coefficients of m secret blocks and the finally element is s_{k_i}. Namely, $v_{k_i} = [e_0, e_1, \cdots, e_{m-1}, s_{k_i}]$ where $e_x \in \{0, 1\}$ for $x = 0, \cdots, m-1$. $e_x = 1$ when x is the index of each operand in s_{k_i} (j_{k_i}, t_{k_i} and z_{k_i}). $e_x = 0$ elsewhere. All v_{k_i}'s are constructed (line 5-15) and are combined into a matrix Q (line 16). $\mathcal{D}$ executes the Gaussian elimination on Q to obtain a matrix Q' (line 17) in order to solve m unknowns $b_0, \cdots, b_{m-1}$. The explanation of GaussEliminate is omitted because it is a common function. $\mathcal{D}$ filters $b_0, \cdots, b_{m-1}$ from Q' (line 18). $\mathcal{D}$ reconstructs $S = b_0 || \cdots || b_{m-1}$ (line 19).

Algorithm 2: Reconst

Input : $\{c_{k_0}, d_{k_0}\}, \cdots, \{c_{k_{m-1}}, d_{k_{m-1}}\}$
Output: $\{b_0, \cdots, b_{m-1}\}$

```
1  for i ← 0 to m − 1 do
2      M_{k_i} ← ListAnagram(|b|, d_{k_i});
3      s_{k_i} ← M_{k_i}[c_{k_i}];
4      j_{k_i}, t_{k_i}, z_{k_i} ← LocateIndices(m, k_i);
5      v_{k_i} ← [ ];
6      for x ← 0 to m − 1 do
7          if (x == j_{k_i}) or (x == t_{k_i}) or (x == z_{k_i})
           then
8              v_{k_i}[x] ← 1;
9          end
10         else
11             v_{k_i}[x] ← 0
12         end
13     end
14     v_{k_i}[m] ← s_{k_i};
15 end
16 Q ← [v_{k_0}, v_{k_1}, ··· , v_{k_{m-1}}]^T;
17 Q' ← GaussEliminate(Q);
18 {b_0, ··· , b_{m-1}} ← filter(Q');
19 S ← b_0 || ··· ||b_{m-1};
20 return S
```

Algorithm 3: LocateIndices

Input : m, k_i
Output: $j_{k_i}, t_{k_i}, z_{k_i}$

```
1  count ← −1;
2  for j ← 0 to m − 3 do
3      for t ← j + 1 to m − 2 do
4          for z ← t + 1 to m − 1 do
5              count + +;
6              if (count == k_i) then
7                  j_{k_i} ← j;
8                  t_{k_i} ← t;
9                  z_{k_i} ← z;
10             end
11         end
12     end
13 end
14 return j_{k_i}, t_{k_i}, z_{k_i}
```

Example 4-2. Following the Example 4-1, suppose that $\{c_1, c_3, c_5, c_6, c_7\}$ are chosen to reconstruct S because the matrix consisting of the coefficient vectors of $\{s_1, s_3, s_5, s_6, s_7\}$ has full rank. Because $|b| = 4$ and $d_1 = 2$, $M_1 = \{0011, 0101, 0110, 1001, 1010, 1100\}$. Because the element whose index in M_1 is $c_1 =$

$10_{bin} = 2_{dec}$ is 0110, $s_1 = 0110$. Similarly, $s_3 = 0010$, $s_5 = 0000$, $s_6 = 1111$ and $s_7 = 0101$. A vector v_{k_i} is then constructed for each c_{k_i}. Because $s_1 = b_0 \oplus b_1 \oplus b_3 = 0110$, $(j_1, t_1, z_1) = (0, 1, 3)$. Therefore, $v_1 = [1, 1, 0, 1, 0, 0110]$. Similarly, $v_3 = [1, 0, 1, 1, 0, 0010]$, $v_5 = [1, 0, 0, 1, 1, 0000]$, $v_6 = [0, 1, 1, 1, 0, 1111]$ and $v_7 = [0, 1, 1, 0, 1, 0101]$. The matrix Q and Q' are constructed as follows:

$$Q = \begin{pmatrix} v_1 \\ v_3 \\ v_5 \\ v_6 \\ v_7 \end{pmatrix} = \left(\begin{array}{ccccc|c} 1, & 1, & 0, & 1, & 0, & 0110 \\ 1, & 0, & 1, & 1, & 0, & 0010 \\ 1, & 0, & 0, & 1, & 1, & 0000 \\ 0, & 1, & 1, & 1, & 0, & 1111 \\ 0, & 1, & 1, & 0, & 1, & 0101 \end{array}\right) \overset{Gauss-elimination}{\rightarrow} Q' = \left(\begin{array}{ccccc|c} 1, & 0, & 0, & 0, & 0, & 1010 \\ 0, & 1, & 0, & 0, & 0, & 0111 \\ 0, & 0, & 1, & 0, & 0, & 0011 \\ 0, & 0, & 0, & 1, & 0, & 1011 \\ 0, & 0, & 0, & 0, & 1, & 0001 \end{array}\right)$$

From Q', $b_0 = 1010, b_1 = 0111, b_2 = 0011, b_3 = 1011$ and $b_4 = 0001$. Finally, $S = b_0 || \cdots || b_4$.

4.3 Share Repair (ShareRepair)

When a participant is corrupted, $\mathcal{D}$ performs the ShareRepair algorithm (Algorithm 4) to repair the corrupted share.

Algorithm 4: ShareRepair

Input : $\mathcal{P}_{corr}$
Output: c_{corr}, d_{corr}

1 $j_{corr}, t_{corr}, z_{corr} \leftarrow$ LocateIndices$(m, corr)$;
2 Choose $\alpha, \beta \in \{0, \cdots, m-1\}$ s.t. $\alpha, \beta \neq j_{corr}, t_{corr}, z_{corr}$;
3 $\{j_{r_1}, t_{r_1}, z_{r_1}\} \leftarrow$ AscendingSort$(j_{corr}, t_{corr}, \alpha)$;
4 $\{j_{r_2}, t_{r_2}, z_{r_2}\} \leftarrow$ AscendingSort$(j_{corr}, z_{corr}, \beta)$;
5 $\{j_{r_3}, t_{r_3}, z_{r_3}\} \leftarrow$ AscendingSort$(j_{corr}, \alpha, \beta)$;
6 $\mathcal{P}_{r_1} \leftarrow$ LocateParticipant$(j_{r_1}, t_{r_1}, z_{r_1})$;
7 $\mathcal{P}_{r_2} \leftarrow$ LocateParticipant$(j_{r_2}, t_{r_2}, z_{r_2})$;
8 $\mathcal{P}_{r_3} \leftarrow$ LocateParticipant$(j_{r_3}, t_{r_3}, z_{r_3})$;
9 $\mathcal{P}_{r_1}, \mathcal{P}_{r_2}, \mathcal{P}_{r_3}$ are required to provide $\{c_{r_1}, d_{r_1}\}$, $\{c_{r_2}, d_{r_2}\}$, $\{c_{r_3}, d_{r_3}\}$ to $\mathcal{D}$;
10 $M_{r_1} \leftarrow$ ListAnagram$(|b|, d_{r_1})$;
11 $M_{r_2} \leftarrow$ ListAnagram$(|b|, d_{r_2})$;
12 $M_{r_3} \leftarrow$ ListAnagram$(|b|, d_{r_3})$;
13 $s_{r_1} \leftarrow M_{r_1}[c_{r_1}]$;
14 $s_{r_2} \leftarrow M_{r_2}[c_{r_2}]$;
15 $s_{r_3} \leftarrow M_{r_3}[c_{r_3}]$;
16 $s_{corr} \leftarrow s_{r_1} \oplus s_{r_2} \oplus s_{r_3}$;
17 $d_{corr} \leftarrow s_{corr}.count('1')$;
18 $M_{corr} \leftarrow$ ListAnagram$(|b|, d_{corr})$;
19 $c_{corr} \leftarrow index(M_{corr}, s_{corr})$;
20 **return** $\{c_{corr}, d_{corr}\}$

Algorithm 5: LocateParticipant

Input : $j_{r_i}, t_{r_i}, z_{r_i}$
Output: $\mathcal{P}_{r_i}$

1 $count \leftarrow -1$;
2 **for** $j \leftarrow 0$ **to** $m-3$ **do**
3 **for** $t \leftarrow j+1$ **to** $m-2$ **do**
4 **for** $z \leftarrow t+1$ **to** $m-1$ **do**
5 $count++$;
6 **if** $(j == j_{r_i})$ **and** $(t == t_{r_i})$ **and** $(z == z_{r_i})$ **then**
7 **return** $count$
8 **end**
9 **end**
10 **end**
11 **end**
12 **return** $count$

Suppose that the participant $\mathcal{P}_{corr}$ is corrupted, $\mathcal{D}$ uses three healthy participants to repair $\mathcal{P}_{corr}$. $\mathcal{D}$ firstly finds the indices of $b_{j_{corr}}$, $b_{t_{corr}}$ and $b_{z_{corr}}$ by LocateIndices (line 1). $\mathcal{D}$ chooses $\alpha, \beta \in \{0, \cdots, m-1\}$ s.t. $\alpha, \beta \neq j_{corr}, t_{corr}, z_{corr}$ (line 2). The idea to find three shares to repair $\mathcal{P}_{corr}$ is that:

$$b_{j_{corr}} \oplus b_{t_{corr}} \oplus b_{z_{corr}} = (b_{j_{corr}} \oplus b_{t_{corr}} \oplus b_{\alpha}) \oplus (b_{j_{corr}} \oplus b_{z_{corr}} \oplus b_{\beta}) \oplus (b_{j_{corr}} \oplus b_{\alpha} \oplus b_{\beta})$$

$\{j_{corr}, t_{corr}, \alpha\}$, $\{j_{corr}, z_{corr}, \beta\}$ and $\{j_{corr}, \alpha, \beta\}$ are sorted in an ascending order using AscendingSort (line 3-5). The pseudo code of the AscendingSort is omitted because it is a simple function. Let $\{j_{r_1}, t_{r_1}, z_{r_1}\}$, $\{j_{r_2}, t_{r_2}, z_{r_2}\}$ and $\{t_{r_3}, t_{r_3}, z_{r_3}\}$ denote the results of these sorts, respectively. $\mathcal{D}$ finds the three participants who store the three XORs using LocateParticipant (line 6-8). $\mathcal{D}$ obtains the XORs $\{s_{r_1}, s_{r_2}, s_{r_3}\}$ by picking the c_{r_1}-th, c_{r_2}-th and c_{r_3}-th in the set M_{r_1}, M_{r_2} and M_{r_3}, respectively (line 10-15). s_{corr} is recovered by $s_{corr} = s_{r_1} \oplus s_{r_2} \oplus s_{r_3}$ (line 16). $\mathcal{D}$ finds the metadata d_{corr} by counting the number of '1' bits in s_{corr} (line 17). $\mathcal{D}$ computes the share c_{corr} (line 18-20) as the ShareGen algorithm.

Example 4-3. Following the Example 4-1 and 4-2, suppose that $\mathcal{P}_4$ is corrupted. $\{j_{corr}, t_{corr}, z_{corr}\} = \{0, 2, 4\}$ because $\mathcal{P}_4$ uses $b_0 \oplus b_2 \oplus b_4$. Choose $\alpha = 1$ and $\beta = 3$ because $1, 3 \neq 0, 2, 4$.

$$b_0 \oplus b_2 \oplus b_4 = (b_0 \oplus b_2 \oplus b_1) \oplus (b_0 \oplus b_4 \oplus b_3) \oplus (b_0 \oplus b_1 \oplus b_3).$$

Let $\{j_{r_1}, t_{r_1}, z_{r_1}\} = \{0,1,2\}$, $\{j_{r_2}, t_{r_2}, z_{r_2}\} = \{0,3,4\}$, $\{j_{r_3}, t_{r_3}, z_{r_3}\} = \{0,1,3\}$. Given $\{j_{r_1}, t_{r_1}, z_{r_1}\} = \{0,1,2\}$, $\mathcal{P}_0$ is chosen because $\mathcal{P}_0$ uses $(b_0 \oplus b_1 \oplus b_2)$. Given $\{j_{r_2}, t_{r_2}, z_{r_2}\} = \{0,3,4\}$, $\mathcal{P}_5$ is chosen because $\mathcal{P}_5$ uses $(b_0 \oplus b_3 \oplus b_4)$. Given $\{j_{r_3}, t_{r_3}, z_{r_3}\} = \{0,1,3\}$, $\mathcal{P}_1$ is chosen because $\mathcal{P}_1$ uses $(b_0 \oplus b_1 \oplus b_3)$. $\mathcal{P}_0, \mathcal{P}_5$ and $\mathcal{P}_1$ are then required to provide $\{c_0, d_0\}, \{c_5, d_5\}$ and $\{c_1, d_1\}$ to $\mathcal{D}$. Given $\{c_0, d_0\} = \{11, 3\}$, $\mathcal{D}$ finds $s_0 = 1110$. Given $\{c_5, d_5\} = \{0, 0\}$, $\mathcal{D}$ finds $s_5 = 0000$. Given $\{c_1, d_1\} = \{10, 2\}$, $\mathcal{D}$ finds $s_1 = 0110$. $\mathcal{D}$ computes $s_{corr} = s_0 \oplus s_5 \oplus s_1 = 1000$. Because the number of '1' bits in 1000 is 1, $d_{corr} = 1$. Because $d_{corr} = 1$ and $|b| = 4$, $M_{corr} = \{0001, 0010, 0100, 1000\}$. Because $|M_{corr}| = 4$, at most $\log_2 4 = 2$ bits are required to represent c_{corr}. The share c_{corr} is the index of s_{corr} in M_{corr}, which is $c_{corr} = 3_{dec} = 11_{bin}$.

5 Property Analysis

This section analyses two properties of the main scheme SW-SSS.

5.1 Secrecy

We firstly consider the secret reconstruction condition. Let *epoch* be a time step in which the participants are checked. If a corrupted participant is detected, it will be repaired in the epoch.

Theorem 1. *The secret can be reconstructed if in any epoch, at least m out of n participants are healthy, and the matrix consisting of the coefficient vectors of the XORs has full rank.*

Proof. $S = b_0 || \cdots || b_{m-1}$. To reconstruct S, $b_0 || \cdots || b_{m-1}$ are viewed as the unknowns that need to be solved. To solve these unknowns, at least m shares $c_{i_1}, \cdots, c_{i_{m-1}}$ along with their metadata $d_{i_1}, \cdots, d_{i_{m-1}}$ are required to make the matrix have full rank.

$$\begin{cases} s_{i_0} = b_{j_0} \oplus b_{t_0} \oplus b_{z_0} \\ s_{i_1} = b_{j_1} \oplus b_{t_1} \oplus b_{z_1} \\ \cdots \\ s_{i_{m-1}} = b_{j_{m-1}} \oplus b_{t_{m-1}} \oplus b_{z_{m-1}} \end{cases}$$

In other words, the number of required participants is at least m, in order to ensure that the equation system is solvable. Theorem 1 leads to a constrain that $n > m$. Because each of n shares is constructed from any three out of $\binom{m}{3}$ secret blocks, another constraint is $n \leq \binom{m}{3}$. Therefore, m and n should be chosen s.t. $m < n \leq \binom{m}{3}$. □

The secrecy is now analysed as follows. Let $H(S)$ be the entropy of the random variable which is induced by S. Let L denote the number of participants whose shares are collectively constructed from m secret blocks. The SW-SSS satisfies the property of the Ramp-SSS as the following theorem:

Theorem 2. *The secrecy of t random variables $\{C_{i_1}, \cdots, C_{i_t}\}$ representing any t shares $\{c_{i_0}, \cdots, c_{i_{t-1}}\}$ and t random variables $\{D_{i_0}, \cdots, D_{i_{t-1}}\}$ representing any t metadata $\{d_{i_0}, \cdots d_{i_{t-1}}\}$ is:*

$$H(S|(C_{i_0}, D_{i_0}), \cdots, (C_{i_{t-1}}, D_{i_{t-1}})) = \begin{cases} H(S), & \text{if } t < L \\ \frac{m-t}{m} H(S), & \text{if } L \leq t < m \\ 0, & \text{if } m \leq t \end{cases}$$

Proof. s_i is the original coded sequence that can be uniquely determined by the share c_i and its metadata d_i. From the property of the conditional entropy, $H(S|(c_{i_0}, d_{i_0}), \cdots, (c_{i_{t-1}}, d_{i_{t-1}})) \leq H(S|s_{i_0}, \cdots, s_{i_{t-1}})$. The equality holds if S is uniformly distributed. For each case of t, the secrecy is given as follows:

- Case 1 ($t < L$): $\{s_{i_0}, \cdots, s_{i_{t-1}}\}$ are constructed from inadequate m secret blocks $b_0, \cdots, b_{m-1}$. The matrix consisting of the coefficient vectors of s_{i_j} does not have full rank. Thus, $H(S|s_{i_0}, \cdots, s_{i_{t-1}}) = H(S)$. This yields $H(S|(C_{i_0}, D_{i_0}), \cdots, (C_{i_{t-1}}, D_{i_{t-1}})) = H(S)$.
- Case 2 ($L \leq t$): the matrix consisting of the coefficient vectors of s_{i_j} has rank t. Thus, $H(S|s_{i_0}, \cdots, s_{i_{t-1}}) = \frac{m-t}{m} H(S)$. This yields: $H(S|(C_{i_0}, D_{i_0}), \cdots, (C_{i_{t-1}}, D_{i_{t-1}})) = \frac{m-t}{m} H(S) < H(S)$.
- Case 3 ($m \leq t$): from (2), $\frac{m-t}{m} H(S) = 0$. This yields: $H(S|(C_{i_0}, D_{i_0}), \cdots, (C_{i_{t-1}}, D_{i_{t-1}})) = 0$. □

5.2 Share Size

The comparison is given in Table 2. In the previous schemes, the share size is $|S| = m \cdot |b|$ (ideal SSS). In the XORNC-SSS, the share size is $m + |b|$ (Section 3). In the SW-SSS, the share size is at most $\mathsf{log}_2 \binom{|b|}{d_i}$ (Section 4.1). For $\forall |b|$ and $\forall d_i \in [0, |b|]$, $\mathsf{log}_2 \binom{|b|}{d_i} < |b|$. Thus, $\mathsf{log}_2 \binom{|b|}{d_i} = \frac{|b|}{x}$ for some $x > 1$. It is clear that $m + |b| > \frac{|b|}{x}$.

Table 2: The share size comparison

	Previous schemes (Shamir-SSS, XOR-SSS, NC-SSS)	**XORNC-SSS**	**SW-SSS**
Share size	$m \cdot \|b\|$	$m + \|b\|$	$\frac{\|b\|}{x}$

6 Efficiency Analysis

Let $(\times)$ and $(\oplus)$ denote the complexity of a multiplication operation in a finite field and the complexity of a XOR, respectively. The $\oplus$ operation is much faster than $\times$ operation, $(\times) \gg (\oplus)$. The efficiency comparison is given in Table 3.

Table 3: The efficiency comparison

		Previous schemes			**Proposed schemes**	
		Shamir-SSS [4]	**XOR-SSS** [21]	**NC-SSS** [22]	**XORNC-SSS**	**SW-SSS**
Feature	XOR-based	No	Yes	No	Yes	Yes
	Arbitrary threshold	Yes	Yes	Yes	Yes	Yes
	Direct share repair	No	No	Yes	Yes	Yes
Storage		$O(m\|b\|)$	$O(m\|b\|)$	$O(m\|b\|)$	$O(m + \|b\|)$	$O(\frac{\|b\|}{x} + \log_2 \|b\|)$
Computation	ShareGen	$O(n \log n)(\times)$	$O(n_p n)(\oplus)$	$O(mn)(\times)$	$O(mn)(\oplus)$	$O(n)(\oplus)$
	Reconst	$O(m^2)(\times)$	$O(n_p^2)(\oplus)$	$O(m^2)(\times)$	$O(m^2)(\oplus)$	$O(m^2)(\oplus)$
	ShareRepair	N/A	N/A	$O(m)(\times)$	$O(m)(\oplus)$	$O(1)(\oplus)$
Communication	ShareGen	$O(nm\|b\|)$	$O(nm\|b\|)$	$O(nm\|b\|)$	$O(n(m + \|b\|))$	$O(n(\frac{\|b\|}{x} + \log_2 \|b\|))$
	Reconst	$O(m^2\|b\|)$	$O(m^2\|b\|)$	$O(m^2\|b\|)$	$O(m(m + \|b\|))$	$O(m(\frac{\|b\|}{x} + \log_2 \|b\|))$
	ShareRepair	N/A	N/A	$O(m^2\|b\|)$	$O(m(m + \|b\|))$	$O(\frac{\|b\|}{x} + \log_2 \|b\|)$

6.1 Storage Cost

In the previous schemes and the XORNC-SSS scheme, the storage cost is the same as the share size because each P_i only stores a share. In the SW-SSS scheme, each P_i stores the share c_i ($|c_i| = \mathsf{log}_2\binom{|b|}{d_i}$) and the metadata d_i ($|d_i| = \mathsf{log}_2|b|$ because $d_i \in [1, |b|]$). Therefore, the storage cost is $O(\mathsf{log}_2\binom{|b|}{d_i} + \mathsf{log}_2|b|)$. For $\forall|b|$ and $\forall d_i \in [0, |b|]$, $\mathsf{log}_2\binom{|b|}{d_i} < |b|$. Thus, $\mathsf{log}_2\binom{|b|}{d_i} = \frac{|b|}{x}$ for some $x > 1$, and $(\frac{|b|}{x} + \mathsf{log}_2|b|) < (|b| + m)$ if $\mathsf{log}_2|b| < m$. This inequality holds if the parameters are chosen s.t. $|b| < 2^m$. If S is divided s.t. any three blocks are different in the same number of bits ($d_0, \cdots, d_{n-1}$ are the same), P_i does not need to store d_i because it becomes a shared information. The storage cost is thus the same as the share size.

6.2 Computation Cost

ShareGen. In the Shamir-SSS, the cost is $O(n \log n)$ because each share is computed from a polynomial. In the XOR-SSS scheme, the cost is $O(n_p n)$ where n_p is the smallest prime s.t. $n_p \geq n$. In the NC-SSS, the cost is $O(mn)$ because n shares are computed from a linear combination of m secret blocks. The XORNC-SSS also computes the shares as the NC-SSS. However, it uses the XOR instead of a linear combination over field multiplications. In the SW-SSS, the cost is $O(n)$ because n shares are computed from the XORs of a tuple of three secret blocks.

Reconst. The costs in the previous schemes, XORNC-SSS and SW-SSS schemes are $O(m^2)$ times $(\times)$ or $(\oplus)$ because the schemes use Gaussian elimination to solve m secret blocks (or to solve the secret and $(m-1)$ coefficients in the Shamir-SSS). Only in the XOR-SSS scheme, the dimension of the matrix used for the Gaussian elimination is $(n_p \times n_p)$, not $(m \times m)$ where n_p is the smallest prime s.t. $n_p \geq n$. Hence, the cost of the XOR-SSS scheme is $O(n_p^2)$ times $(\oplus)$.

ShareRepair. The costs in the NC-SSS and XORNC-SSS schemes are $O(m)$ because a corrupted share is repaired using m healthy shares. The difference between the two schemes is that the NC-SSS uses the field linear combinations while the XORNC-SSS uses the XORs. In the SW-SSS, the cost is $O(1)$ because a new share is computed from three healthy shares.

6.3 Communication Cost

ShareGen. $\mathcal{D}$ distributes n shares to n participants. In the previous schemes, the cost is $O(nm|b|)$ because the share size is $m|b|$. In the XORNC-SSS, the cost is $O(n(m+|b|))$ because the share size is $(m+|b|)$. In the SW-SSS scheme, the cost is $O(n(\frac{|b|}{x}+\log_2|b|))$ because the share size is $(\frac{|b|}{x}+\log_2|b|)$.

Reconst. In the previous schemes and the proposed schemes, m participants are required to provide their shares to $\mathcal{D}$, the costs are thus m times the storage cost of each scheme.

ShareRepair. The share repair is not supported in the Shamir-SSS and XOR-SSS schemes. In the NC-SSS and XORNC-SSS schemes, the costs are $O(m^2|b|)$ and $O(m(m+|b|))$ because m healthy participants are required to provide their shares to $\mathcal{D}$ for the share repair. In the SW-SSS scheme, the cost is $O(\frac{|b|}{x}+\log_2|b|)$ because only three healthy shares are required for the share repair.

6.4 Computation Evaluation

This section presents the simulation of the SW-SSS scheme to show that is is applicable to a real system. The program that is written by Python 2.7.3 is executed using a computer with Intel Core i5, 2.40 GHz, 4 GB of RAM, Win 7 64-bit OS. Each secret block, b_i, is set to be 2^{10} bits. Each result is the average of 100 runs. The gmpy2 library is used. The simulation results in Fig. 1 are observed with three sets of the computation performance: ShareGen, Reconst, and ShareRepair by varying the number of secret blocks, m. The number of participants, n, is set to be $n = m+1$. The graphs reveal that the computation time of ShareRepair is almost constant and is independent on m. The computation time of ShareGen and Reconst linearly increases with m. The average slopes of increment in ShareGen and Reconst are 0.004 and 0.012, respectively. From these results, if the secret size, $|S|$, is 2^{26} bits ($m = 65,500$), which is almost an upper bound of the size of a secret (e.g., secret key, signature), the computation time of ShareGen, Reconst and ShareRepair is merely 374.72s (6mins), 905.28s (15.1mins) and 0.031s, respectively.

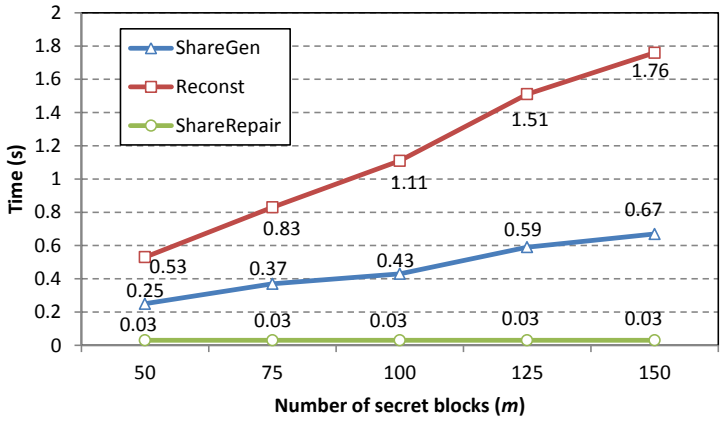

Fig. 1: The computation performance of SW-SSS

7 Conclusion and Future Work

This paper firstly revisits the XOR-based NC to apply it for the SSS (called XORNC-SSS) in order to support the share repair, to obtain the arbitrary threshold, and to reduce the share size. This paper then proposes the main SW-SSS scheme to optimize the share size. The key idea is based on the binning idea of the Slepian-Wolf coding, which is commonly used in data compression in a network. The security analysis is provided based on the entropy theory. The efficiency analysis is discussed based on the complexity theory. The simulation results of the SW-SSS scheme reveal that it is applicable to a real SSS system. Future research is required to investigate the implementation of the previous schemes.

References

1. Pedersen TB, Saygin Y, Savas E (2007) Secret Sharing vs. Encryption-based Techniques For Privacy Preserving Data Mining. Sciences New York (December):17-19.
2. Nirali RN, Devesh CJ (2013) A Game Theory based Repeated Rational Secret Sharing Scheme for Privacy Preserving Distributed Data Mining. SECRYPT 2013:512-517.
3. Selim VK, Thomas BP, Erkay S, Ycel S (2007) Efficient Privacy Preserving Distributed Clustering Based on Secret Sharing. Conf. on Emerging tech. in knowledge discovery and data mining (PAKDD), 4819:280-291.
4. Shamir A (1979) How to share a secret. Proc. of Communication of the ACM, 22(11):612-613.

5. Blakley GR (1979) Safeguarding cryptographic keys. AFIPS National Computer Conf., vol.48, pp.313-317.
6. Karnin E, Greene JW, and Hellman ME (1983) On secret sharing systems. IEEE Trans. Inform. Theory, 29(1):3541.
7. Capocelli RM, Santis AD, Gargano L, and Vaccaro U (1993) On the size of shares for secret sharing schemes. J. of Cryptology, 6:157167.
8. Blakley GR, Meadows C (1984) Security of ramp schemes. Proc. of CRYPTO on Advances in cryptology. LNCS 196, Springer-Verlag, pp.242-269.
9. Yamamoto H (1985) On secret sharing systems using (k, L, n) threshold scheme. IEICE Trans. Fundamentals (Japanese Edition), J68-A(9):945-952.
10. Kurosawa K, Okada K, Sakano K, Ogata W, T. Tsujii (1993) Non perfect secret sharing schemes and matroids. Workshop on the Theory and Application of Cryptographic Techniques (EUROCRYPT). LNCS 765, Springer-Verlag, pp.126-141.
11. Ogata W, Kurosawa K (1998) Some basic properties of general nonperfect secret sharing schemes. J. Universal Computer Science, 4(8):690-704.
12. Okada K, Kurosawa K (1994) Lower bound on the size of shares of nonperfect secret sharing schemes. Conf. on the Theory and Applications of Cryptology (ASIACRYPT). LNCS 917, SpringerVerlag, pp.3441.
13. Wang Y (2012) Efficient LDPC Code Based Secret Sharing Schemes and Private Data Storage in Cloud without Encryption. Technical report, UNC Charlotte.
14. Ishizu H, Ogihara T (2004) A study on long-term storage of electronic data. IEICE General Conf., D-9-10(1):125.
15. Hosaka N, Tochikubo K, Fujii Y, Tada M, Kato T (2007) $(2, n)$-threshold secret sharing systems based on binary matrices. Symposium on SCIS (in Japanese), pp 2D1-4.
16. Kurihara J, Kiyomoto S, Fukushima K, Tanaka T (2008) A fast (3,n)-threshold secret sharing scheme using exclusive-OR operations. IEICE Trans., E91-A(1):127-138.
17. Shiina N, Okamoto T, Okamoto E (2004) How to convert 1-out-of-n proof into k-out-of-n proof, Symposium on SCIS (in Japanese), pp 1435-1440.
18. Kurihara J, Kiyomoto S, Fukushima K, Tanaka T (2008) A new (k, n)-threshold secret sharing scheme and its extension. 11th conf. on Information Security (ISC), pp.455-470.
19. Chunli L, Jia X, Tian L, Jing J, Sun M (2010) Efficient Ideal Threshold Secret Sharing Schemes Based on EXCLUSIVE-OR Operations. 4th Conf. on Network and System Security (NSS), pp.136-143.
20. Wang Y and Desmedt Y (2014) Efficient Secret Sharing Schemes Achieving Optimal Information Rate. Information Theory Workshop (ITW).
21. Kurihara J, Kiyomoto S, Fukushima K, Tanaka T (2009) A fast (k-L-N)-Threshold Ramp secret sharing scheme. IEICE Trans. fundamentals, doi:10.1587/transfun.E92.A.1808.
22. Kurihara M, Kuwakado H (2012) Secret Sharing Schemes Based on Minimum Bandwidth Regenerating Codes. Symposium on Info. Theory and its Applications (ISITA), pp.255-259.
23. Liu J, Wang H, Xian M, Huang K (2013) A Secure and Efficient Scheme for Cloud Storage against Eavesdropper. 15th Conf. on Info. and Comm. Security (ICICS), pp.75-89.
24. Cai N, and Raymond W.Y (2002) Secure network coding. IEEE Int. Symposium on Information Theory.
25. Katti S, Rahul H, Hu W, Katabi D, Medard M, Crowcroft J (2008) XORs in the air: practical wireless network coding. Trans. on Networking, 16(3):497-510.
26. Yu Z, Wei Y, Ramkumar B, Guan Y (2009) An Efficient Scheme for Securing XOR Network Coding against Pollution Attacks. 28th Conf. on Computer Communication (INFOCOM), pp.406-414.
27. Slepian D, Wolf JK (1973) Noiseless coding of correlated information sources. IEE Trans. on Information Theory, 19(4):471-480.
28. Samuel Cheng (2010) Slepian-Wolf Code Designs. Available: http://tulsagrad.ou.edu/samuel_cheng/ information_theory_2010/swcd.pdf.
29. Ahlswede R, Ning Cai, Li SYR, Yeung RW (2000) Network information flow. IEEE Trans. on Information Theory on 46(4): 1204-1216.
30. Ho T, Medard M, Koetter R, Karger DR, Effros M, Shi J, Leong B (2006) A random linear network coding approach to multicast. IEEE Trans. Information Theory 52(10):4413-4430.
31. Li S-YR, Raymond WY, Ning Cai (2003) Linear network coding. IEEE Trans. on Information Theory, 49(2):371-381.
32. Agrawal S, Boneh D (2009) Homomorphic MACs: MAC-Based Integrity for Network Coding. 7th Conf. on Applied Cryptography and Network Security (ACNS), pp.292-305.
33. Stankovic V, Liveris AD, Xiong Z, Georghiades CN (2006) On code design for the Slepian-Wolf problem and lossless multiterminal networks. IEEE Trans. on Information Theory, 52(4):1495-1507.
34. Stankovi V, Liveris AD, Xiong Z, Georghiades CN (2004) Design of Slepian-Wolf Codes by Channel Code Partitioning. Data Compression Conf. (DCC), pp.302-311.
35. Cheng C, Jiang T (2012) An Efficient Homomorphic MAC with Small Key Size for Authentication in Network Coding. IEEE Trans. on Computers, 62(10):2096-2100.
36. Johnson R, Molnar D, Song D and Wagner D (2002) Homomorphic Signature Schemes. CT-RSA, pp.244-262.
37. Freeman D.M (2012) Improved security for linearly homomorphic signatures: a generic framework. 15th conf. on Practice and Theory in Public Key Crypt. (PKC), vol.7293, pp.697-714.