A Tree Based Recursive Scheme for Space Efficient Secret Sharing

Abhishek Parakh and Subhash Kak

Computer Science Department, Oklahoma State University
Stillwater, OK 74078

Abstract. This paper presents a k-out-of-n recursive secret sharing scheme based on an n-ary tree data structure. In recursive hiding of secrets, the user encodes additional secrets in the shares of the secret intended to be original shared without an expansion in the size of the latter, thereby decreasing the effective share size per secret and increasing the overall space efficiency of secret sharing, with a tradeoff in security. The proposed scheme has applications in secure distributed storage and information dispersal protocols. It may be used as a steganographic channel to transmit hidden information in secret sharing, which may be used for authentication and verification of shares and the reconstructed secret itself.

Keywords. Recursive hiding of secrets, computational secret sharing, space efficiency, visual cryptography.

1 Introduction

Information theoretically secure secret sharing schemes [1–6], are space inefficient. For example, a k-out-of-n, denoted as (k, n), secret sharing scheme expands a secret of b bits into n shares each of at least b bits in size. Furthermore, since only k of these shares are needed to recreate the secret, each bit of any share, in a threshold secret sharing scheme, effectively conveys at most $\lceil \frac{1}{k} \rceil$ bits of secret. If $k = n$, as in the case of a non-threshold scheme, where all the shares must be brought together to recreate the secret, the effective information conveyed by each bit of any share is $\lceil \frac{1}{n} \rceil$ bits of the secret.

One way to improve space efficiency is to distribute shares smaller in size than the secret itself, however at the cost of reduction in security. Computational secret sharing techniques have been developed to achieve this [18–22], in which a symmetric key is used to encrypt the original secret and the encrypted secret is divided into pieces to which redundancy is added by the use of block error correction techniques [22–24]. The encryption key is split into shares using information theoretically secure methods of secret sharing. This leads to an n-fold increase in key size, shares of which have to be stored with every piece of the encrypted secret, hence incurring an overhead [22].

A method for sharing multiple secrets with security based on assumption of hardness of discrete logarithm problem is presented in [25]. In [26] a scheme based on systematic block codes and in [27, 28] schemes using Shamir's secret sharing have been proposed to share multiple secrets, however they require a large amount of side information to be stored as public knowledge and further [26–29] attempt to maintain ideal security.

An extension of secret sharing schemes is visual cryptography [1] that aims at dividing an image into two or more shares such that when a predetermined number of shares are aligned and stacked together, the secret image is revealed [1, 8, 9], without the requirement of any computation. However, information theoretically secure approaches to visual cryptography also suffer from inefficiency in terms of number of bits of secret conveyed per bit of share.

A number of multi-image hiding schemes have been developed [10–13]. However, the implementation in [11] does not hide the information using "real" visual cryptography, in the sense that computation needs to be performed to extract the hidden information. Lou et. al. [10] use a secret key to generate a permutation order and the key needs to be conveyed in addition to the shares, becoming a overhead and also amounts to computation, which was to be avoided by visual cryptography. In [12], two watermarks are hidden, one during half-toning of original image and second while creating shares, however, both these hiding techniques do not conform to visual cryptography because, to extract first hidden image, an exclusive OR reconstruction of the image needs to be performed, which is never done in visual cryptography and the extraction of the second hidden image requires additional XOR operations. Fang and Lin [13] hide only integer from 0 to 6, which has very limited applications and again the integers are not decoded visually but require computation to be extracted.

Wu and Chen [14] propose a scheme to hide two images at a rotation angle of 90 degrees while Wu and Chang [15] discuss hiding multiple images using circular shares at a limited number of rotation angles. Hsu et. al. [16] extended the scheme to allow arbitrary rotation angle by rolling shares into rings. However, circular shares distort the aspect ratio of the original image. Further knowing how and by what degree the shares are to be rotated requires additional side information to be supplied along with the share. Also pixel expansion is an issue, for example in [17], which encodes two or more secrets into circular shares, the pixel expansion is proportional to the number of secrets being hidden.

Another way to improve information efficiency is to hide additional secrets in the shares of the original secret, without increasing the share size of the latter in comparison to what it would be without the additional hidden secrets. This effectively reduces the share sizes per secret, taking both hidden and original secrets into account. Recursive hiding of secrets was proposed in [7] for this purpose and to serve as a steganographic channel, with applications to binary images and binary text but was only limited to a (2,2) secret sharing. The idea involved is recursive hiding of smaller secrets in shares of larger secrets with secret sizes doubling at every step, thereby increasing the information that every bit of share conveys to $\frac{(n-1)}{n}$ bit of secret i.e. nearly 100%. However, the scheme described in [7] is a non-threshold scheme where all the shares are needed to recreate the secret.

The appendix presents a recursive 2-out-of-n visual secret sharing scheme which is an extension of the 2-out-of-2 visual secret sharing scheme presented in [7]. This small but important extension helps in understanding the basic idea behind recursive hiding of secrets.

In this paper, we present a general (non-visual) k-out-of-n secret sharing scheme that hides additional information in the shares of the original secret, without any increase in the share sizes of the latter. This represents an increase in space efficiency of the secret sharing scheme. Further, our algorithm acts as a dual algorithm in the sense that it can be used as a multi-secret sharing algorithm as well as a computational secret sharing algorithm.

When used as a multi-secret sharing algorithm, it recursively encodes different secrets into the shares of other secrets such that the "inner" secrets do not lead to any expansion in share sizes of the "outer" secrets. In other words, the shares of the "outer" secrets now also convey the "inner" secrets, thereby increasing the information efficiency.

When used as a computational secret sharing algorithm, we simulate a multi-secret sharing algorithm, by dividing the original (larger) secret into multiple pieces of smaller sizes. These pieces are then recursively encoded into shares such that some of the pieces act as the "inner secrets" and some as "outer secrets".

Further, since the algorithm produces shares of size on the order of the size of the pieces, it effectively results in smaller secret share sizes than conventional schemes. It is to be noted, however, that the security in both cases, the multisecret sharing and the computational secret sharing schemes is computational.

The proposed recursive secret sharing scheme has applications in distributed online storage of information discussed in [23, 24]. Systems implementing such distributed data storage have appeared at CMU and IBM [31–34, 19].

In section 2, we present a generalization of the (2,2) recursive scheme presented in [7] to a (2,3) threshold scheme for binary strings. The aim of this section is to make clear the idea of recursion, for text, in secrecy context and how it can be applied to improve the efficiency of secret sharing. Section 3, extends the (2,3) recursive scheme for text to a (n, k) recursive scheme, where, the new scheme is based on repeated polynomial interpolation and sampling. Section 4 discusses the use of the proposed secret sharing scheme as a computational secret sharing scheme and section 5 comments on the security of the scheme, while section 6 is conclusions.

2 A Comparison Based $(2, 3)$ Recursive Scheme

For text represented as a binary sequence, a 2 out of 3 secret sharing scheme can be developed using a simple comparison based algorithm as follows: we divide a secret bit into 3 shares p_1, p_2, and p_3 such that $p_1 = p_2 = p_3$ if we wish to encode bit 0, and $p_1 \neq p_2 \neq p_3$ if we wish to encode bit 1.

To satisfy the above conditions we would need at least 3 symbols, say 0, 1 and 2. Therefore to encode bit 0 we could create pieces $p_1p_2p_3$ as 000, 111, or 222. Whereas the candidates to encode bit 1 would be 012 and all possible permutations of it, i.e. 021, 102, 120, 210, and 201. In all, to encode secret bit 0 and secret bit 1, we have 3 and 6 possibilities, respectively.

This asymmetry in the number of choices does not affect the security of the scheme because an adversary can guess a bit correctly, by brute force, with a probability of one-half. This is the maximum security one can achieve for any one bit. Now in our scheme in the encoded form for any bit, given one of its shares, there exist two possible choices for the second share, i.e. either the second share is the same as the first or the second share is different from the first. Therefore, a player can guess the second share only with a probability of one-half.

Example 1. If M is a 27 bit long message that we wish to encode into 3 shares and the threshold is 2, then non-recursive shares S_1, S_2, and S_3 may be created as follows:

M : 011011010110110011100101101
S_1 : 102012012010201201201020102
S_2 : 110020022120111210101221001
S_3 : 121001002200021222001122200

Viewed as a ternary alphabet, the efficiency of this system is 33%. As a comparison, if 0, 1 and 2 are encoded using prefix coding as 0, 10, and 11 respectively, then we are effectively mapping each bit of secret into 5 bits of shares and the efficiency is only $\frac{27}{27\times 5} = \frac{1}{5}$, i.e. 20%.

The above efficiency can be improved by recursively hiding additional secrets in the shares of M. However, since each bit is mapped into 3 shares, in order to take advantage of the recursive technique, the secrets at each step must increase by a factor of 3. We can then hide the following secrets M_1, M_2, and M_3 in shares of M as follows (figure 1):

Note that at each step we have used the shares of the previous smaller messages to create the shares of larger messages; these smaller shares are denoted in bold. Also, we have distributed the shares at each step so that no player has access to all the shares of the smaller messages and hence, every message (seen from a per-message view point) remains secure until at least two players come together. This approach is different from that discussed in [7], where the shares of smaller messages were all accumulated into one of the larger shares instead of distributing them among all the possible players. As a result in [7], any player having that share which encodes the smaller images could in principle recreate these smaller images without the help of the other player, which in some cases might not be acceptable. Therefore, our new approach is more secure for certain applications.

Figure 2 illustrates the development of recursion tree for example 1. It is seen that the tree forms a ternary structure with nodes at each level giving rise to 3 nodes in the following level, and the nodes shown in bold are the nodes carried over from the previous level. The shares are distributed from left to right one at a time, i.e. if we number the tree leaves starting from the left as 1,2,3,1,2,3

Secret	Shares	
$M_1: 1$	0	$S_{M_{11}}$
	2	$S_{M_{12}}$
	1	$S_{M_{13}}$
$M_2: 010$	011	$S_{M_{21}}$
	021	$S_{M_{22}}$
	001	$S_{M_{23}}$
$M_3: 110101101$	**011**122102	$S_{M_{31}}$
	121**021**200	$S_{M_{32}}$
	201220**001**	$S_{M_{33}}$
$M: 011011010110110011100101101$	**011122102**011101221001121202	S_1
	020101122**121021200**101022100	S_2
	002110112201211212**201220001**	S_3

Fig. 1. Recursive hiding of smaller messages in the shares of larger messages

and so on. Then these numbers denote the player's number to whom the shares belong.

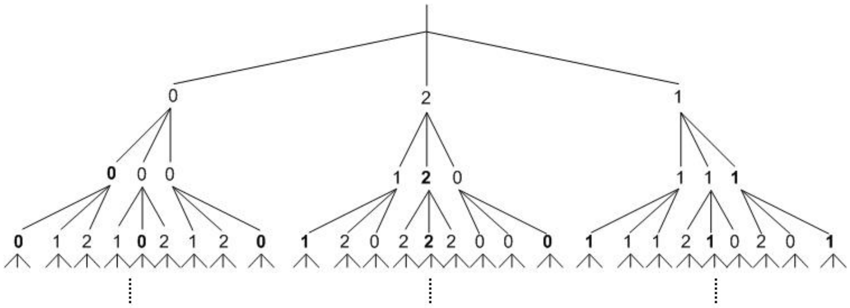


Fig. 2. Illustration of recursion tree for example 1 (partial illustration)

Also seen in figure 1 is that using recursive hiding of secrets, we have been able to encode 13 bits of M_1, M_2, and M_3 and 27 bits of M into shares of M alone. As a result the efficiency, considering a ternary alphabet, is $\frac{13+27}{3\times 27} = \frac{40}{81} \approx \frac{1}{2}$ (i.e. 50% compared to 33% in the non-recursive case). If one considers the binary representations of each character then each share now conveys $\frac{13+27}{5\times 27} = \frac{8}{27} \approx \frac{1}{3}$ bits. Compared to 1/5 bits in the conventional approach, this is an almost 40% increase in efficiency.

3 Proposed k-out-of-n Recursive Secret Sharing Scheme

A secret sharing scheme based on polynomial sampling and interpolation in finite field is discussed in [3], where the secret is mapped as a point at $x = 0$ and $k-1$ additional points are chosen randomly and uniformly from the field. These k points are then interpolated to generate a polynomial of degree $k-1$, which is then sampled at n points (except at $x = 0$). These n samples, the (x, y) points, are distributed as shares among the players.

In order to reconstruct the secret, any k of the shares (points) can be interpolated to regenerate the $k-1^{th}$ degree polynomial, which can then be sampled at $x = 0$ to retrieve the secret.

At each level of recursion, in our proposed scheme, we also use polynomial interpolation and sampling. However, we will be hiding additional pieces of information within the shares of the original secret. We work in a finite field $\mathbb{Z}_p$, where p is a prime and it is public knowledge. Further, we assume that a secret S is represented as string of numbers $S = s_1 s_2 \ldots s_r$, where each $s_i \in \mathbb{Z}_p$ and $|S| = r = n^h$, where $|S|$ denotes the length of secret S for some integer h. For example, if we assume that the secret is a text message composed of ASCII characters, then it can be represented as a string of numbers less than $p = 257$ [23].

Furthermore, assume that we have another string denoted by $M = m_1 m_2 \ldots m_x$, $m_i \in \mathbb{Z}_p$, where $|M| = x = \frac{n^h-1}{n-1}$, to be hidden within the shares of the original secret S. For instance, in example 1, $n = 3$, $h = 3$, and hence in the shares of the original secret $|S| = n^h = 3^3 = 27$ bits long we were able to encode $\frac{n^h-1}{n-1} = 13$ additional bits of information.

The upper limit on the number of additional "pieces" of information that can be encoded within the shares of the original message of size n^h, in the proposed scheme, is $\frac{n^h-1}{n-1}$.

We also observe that the recursive schemes proposed in this paper (as in section 2), forms a n-ary tree structure where the original secret forms the leaves of the tree and the hidden information forms the internal nodes. Consequently, a comparison can be made between the efficiency of conventional secret sharing schemes and tree based recursive secret sharing schemes.

In information theoretic secret sharing schemes, each share of the secret is of the same size as the secret itself, as a result, for a secret of given size n^h, each of the n shares are of size n^h. This results in an efficiency of $\eta_c = \frac{n^h}{n^h \cdot n} = \frac{1}{n}$, where η denotes efficiency and subscript c denotes conventional (information theoretically secure) scheme.

A tree based recursive scheme hides $\frac{n^h-1}{n-1}$ pieces of additional information within the n shares each of size n^h. Consequently, the efficiency of tree based recursive schemes is $\eta_r = \frac{1}{n^h \cdot n} \cdot (n^h + \frac{n^h-1}{n-1})$. A recursive tree based secret sharing improves the efficiency of conventional secret sharing methods by a factor of $e = 1 + \frac{1}{n^h} \cdot (\frac{n^h-1}{n-1})$. Which is one plus the ratio of the number of additional pieces hidden within the pieces of the original secret to the number of pieces of the original secret.

With the above in mind, the proposed scheme works as follows:

Inputs: Original secret - $S = s_1 s_2 \dots s_r$; message to be hidden - $M = m_1 m_2 \dots m_x$; n; k and p where s_a, $m_b \in \mathbb{Z}_p$, $1 \le a \le r$, $1 \le b \le x$, $r = n^h$ and $x = \frac{n^h - 1}{n-1}$.

Algorithm 1a. Creation of shares

1. Choose $k-1$ random numbers $r_l \in \mathbb{Z}_p$, $l = 1$ to $k-1$ uniformly and randomly.
2. Interpolate a polynomial $p_{11}(x)$ using k points $(0, m_1)$ and (l, r_l), $1 \le l \le k-1$. Let $p_{ic}(x)$, denotes the c^{th} polynomial at the i^{th} level in the recursion (tree).
3. Sample $p_{11}(x)$ at n points: D_{11}^1, D_{11}^2, ..., D_{11}^n, where in D_{ic}^k, k refers to the index number of the sample as well as the x-coordinate at which the sample is taken; i and c are the same as noted in point 2 above.
4. Initialize $a = 1$, $b = 2$.
5. For $i = 2$ to h
   (a) $c = 1$
   (b) For $k = 1$ to n^{i-2}
       For $j = 1$ to n
         if $i < h$
           i. Interpolate $p_{ic}(x)$, using points $(0, m_b)$, $(j, D^j_{(i-1)k})$ and $k-2$ randomly and uniformly chosen numbers.
           ii. Sample $p_{ic}(x)$ to generate samples D^q_{ic}, $1 \le q \le n$.
           iii. $b = b + 1$, $c = c + 1$
         else
           i. Interpolate $p_{ic}(x)$, using points $(0, s_a)$, $(j, D^j_{(i-1)k})$ and $k-2$ randomly and uniformly chosen numbers.
           ii. Sample $p_{ic}(x)$ to generate samples D^q_{ic}, $1 \le q \le n$.
           iii. $a = a + 1$ and $c = c + 1$
           iv. Distribute D^q_{ic}, $1 \le q \le n$ to players from 1 to n, respectively.

We do not consider the final shares as a part of the tree. Hence, the tree has only n^h leaves, which are then interpolated and sampled at n points to generate the final shares.

Algorithm 1b. Reconstruction of secret and hidden information

1. For $1 \le c \le r$
   (a) Interpolate k shares D^i_{hc}, $1 \le i \le k$ to generate polynomial $P_{hc}(x)$.
   (b) Sample $P_{hc}(x)$ at $x = 0$ to retrieve s_c.
2. For $i = h - 1$ down to 1
   (a) $j = 1$, $b = 1$, $q = \frac{n^{i-1} - 1}{n-1} + 1$
   (b) For $c = 1$ to n^i
     i. Sample $P_{(i+1)c}(x)$ at point $x = b$, denote as D^x_{ij}.
     ii. $b = b + 1$

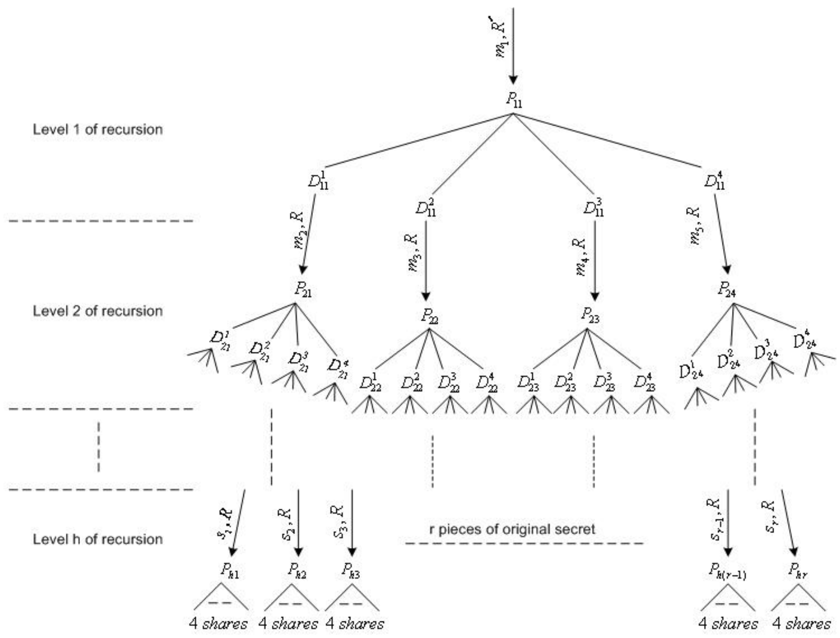


Fig. 3. Illustration of application of algorithm 1 for $n = 4$.

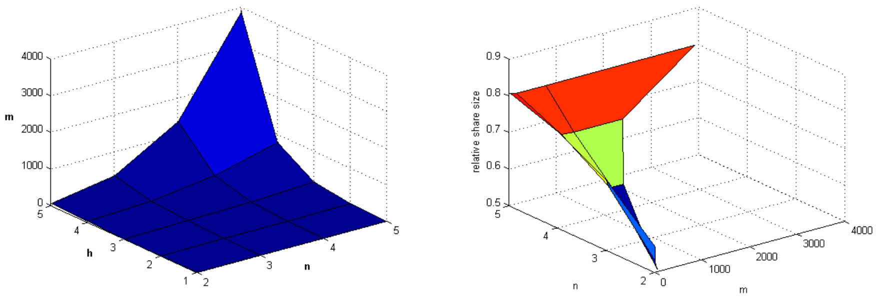
iii. if $c \bmod n = 0$
   A. Interpolate (x, D_{ij}^{x}), $x = 1$ to n to generate $P_{ij}(x)$.
   B. Sample $P_{ij}(x)$ at $x = 0$ to retrieve m_q.
   C. $x = 1$, $j = j + 1$, $b = 1$, $q = q + 1$.

The share reconstruction process traverses the tree from leaves to the root (figure 3), while the reconstruction process retrieves the secret and the hidden messages in a last in first out manner. In figure 3, R denotes a vector of length $k-2$ numbers randomly and uniformly chosen from the field. Note that R' is a $k-1$ element vector in the first step (level 1). Each instance of R is independent.

4 Proposed Computational Secret Sharing Scheme

We can use the proposed scheme of section 3 as a computational secret sharing scheme by dividing the secret into smaller pieces and then recursively encoding the pieces as suggested in algorithm 1a. However the difference would be that instead of inner pieces being that of a different message to be hidden, they would be pieces of the secret itself. If we create m pieces p_i of the secret, then in the ideal case m should be equal to $\frac{n^{h+1}-1}{n-1}$, for a given n and some integer h, where $|p_i| = \frac{|S|}{m}$. Further, the tree has n^h number of leaves which then yield n shares each, resulting in $n \cdot n^h$ number of shares. Therefore, each of the n player receives a share of effective size $n^h \cdot \frac{|S|}{m} = (1 - \frac{m-1}{m \cdot n}) \cdot |S|$. This represents a reduction in share sizes; for example, if the secret is broken into $m = 15$ pieces and $n = 2$, then the effective share size for each player is $(1 - \frac{14}{30}) \cdot |S|$ compared to $|S|$ in the conventional case.

However, the above holds only when $m | \frac{n^{h+1}-1}{n-1}$. A plot for the values of m for $n = 2$ to 5 and $h = 1$ to 5 is shown in figure 4a.



(a) (b)

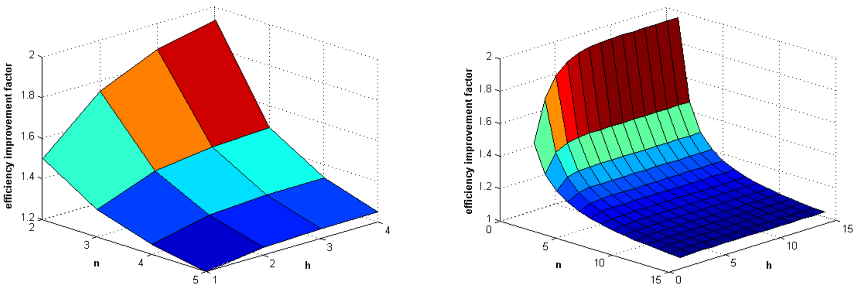
Fig. 4. (a) Plot of possible values m can take as n and h vary. (b) Plot of new effective share sizes relative to the size of the original secret S versus m and n (only a few values are shown).

Figure 4b shows how the size of resulting shares change relative to the size of original secret. The plot in figure 4b is drawn for the values that the number of pieces m can take from that shown in figure 4a against the number of players n. The z-axis is the ratio of the new effective share size to the share size in a conventional scheme, i.e. $1 - \frac{m-1}{m \cdot n}$.

The efficiency improvement factor for the ideal case is given by $1 + \frac{1}{n^h} \cdot (\frac{n^h-1}{n-1})$. A plot of how efficiency improvement factor varies as a factor of n and h is given in figure 5.

Figure 5 shows that given a n, more the height h, better the efficiency improvement factor. An efficiency factor of 2 implies a 50% reduction in share size compared to information theoretically secure schemes where each share is of size $|S|$. Therefore, new effective share size is given by $|S_N| = \frac{|S|}{\text{efficiency improvement factor}}$, where $|S|$ is the original secret size.

Optimal number of pieces: In practice, since the number of pieces may be arbitrary, the n-ary tree may or may not be complete, and one may need to stuff additional (dummy) pieces to complete the tree. The number of pieces required to complete the tree is a factor of the height h of tree. Further in order to maximize the information efficiency, we would like to determine what h one would want to choose.



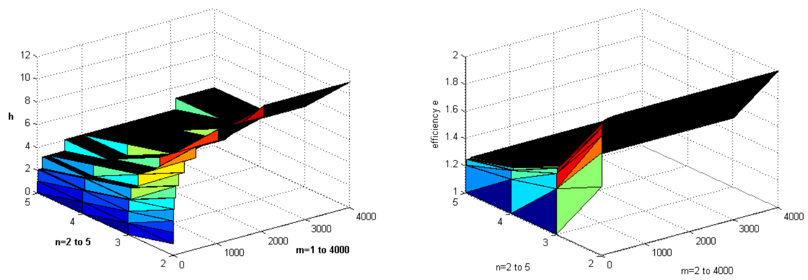
(a) (b)

Fig. 5. (a) Plot of efficiency improvement factor as a function of n and h. (b) Plot of efficiency improvement factor as a function of n and h (larger values).

In general, we assume that we are working in a decimal base, i.e. each secret may be represented as a sequence of integers 0-9. This means that the smallest piece one may create is a single digit number. Also, note that the prime p used in algorithm 1, can only be chosen after the piece sizes are decided upon. For example, in the case of smallest possible pieces (single digits) a prime $p = 11$ would suffice. However, if one was to choose each piece to be of two digits in size, then a prime $p = 101$ would be needed. Let us redefine m to be the number of smallest pieces in the original secret, for example the number of digits in the secret.

In order to understand how stuffing of pieces would work, assume that we are working in a binary field; therefore smallest piece that can be created is of one bit in size. Since the total number of pieces has to form a n-ary tree, if we denote by m the number of pieces of the original secret then m may not always be an integral multiple of $\frac{n^{h+1}-1}{n-1}$ for the given value of n and any value of h. As a result, in order to complete the tree, we may either need to adjust the piece sizes (to more than one bit in size, in turn changing the prime p required in algorithm 1), and/or stuff the secret with dummy bits. Below, we will investigate both the cases and determine which case results in a better efficiency.

Assume single bit pieces and bit stuffing (if required), and that m and n are fixed. Since the last level of the tree has n^h number of leaves, each player will receive n^h bits. Therefore, the value of h chosen will decide the number of bits stuffed to complete the tree. Also, since we would want to have each piece as a single bit, we note that if $n^h > m$, then each player will receive more bits than originally in the secret, which is not desired. Consequently, one of the criterions in choosing h is that $n^h \leq m$, where m is the total number of bits in the original secret. This gives the upper bound on h. Also, to maintain the requirement of one bit per piece, the total number of nodes in the tree must be greater than the total number of bits in the original secret. Hence, $\frac{n^{h+1}-1}{n-1} \geq m$ gives the lower bound for h.



(a) (b)

Fig. 6. (a) Shows how the height h changes as a function of the number of pieces m and the number of players n. (b) Shows how the efficiency improvement factor changes as a function of arbitrary m and n. In both (a) and (b) each piece is taken to be of the minimum size possible (ex. a single digit when using a decimal base).

On the other hand, if we assume that each pieces may be larger in size than the minimum size possible, then starting from the upper bound on h, $h = \lfloor log_n(m) \rfloor$, we could gradually reduce the tree height by one at each step and then readjust the shares so as to complete the tree, with or without stuffing of pieces. Then a comparison could be made between the resulting efficiencies. Note that as we change the piece sizes, we may require changing the prime p for algorithm 1.

For example, again consider working in a binary field and that the original secret consists of $m = 8$ bits and let $n = 2$, so that the algorithm constructs a binary tree. Since, the smallest piece possible is a bit, if one was to construct a recursion tree with one bit pieces, then the tree at the 4^{th} (last) level requires 7 bits stuffed to be complete. When these bits are divided into shares, since the last level now has 8 bits, it would result in each of the two shares being 8 bits.

However, if one was to reduce the tree height by one level by adjusting the pieces to be of two bits each then the 3^{rd} (last) level of the tree would require some pieces to be stuffed and would result in 4×2 bits for each share. This does not result in any efficiency improvement yet. However, if one was to further reduce another level of the tree, so that the tree now has only two levels, then each piece would be of 3 bits each, where only 1 dummy bit is stuffed. Since, now the 2^{nd} (last) level of the tree has 2×3 bits, each share would be of 6 bits each. This is a reduction in share sizes compared to a non-recursive scheme.

Consequently, in general, in order to maximize efficiency, the height h of the recursion tree must be chosen such that it minimizes $\lceil \frac{|S|}{n^{h+1}-1} \cdot (n-1) \rceil \times n^h$ for $1 \leq h \leq \lfloor log_n(m) \rfloor$. Here, m denotes the number of pieces of the smallest size present in the original secret corresponding to the base that we working in.

The appendix shows plots for the maximum efficiency improvement achievable and the corresponding height against the number of (smallest) pieces present in the original secret.

5 On Security of the Proposed Schemes

When applying a secret sharing scheme based on polynomial interpolation and sampling (Shamir's scheme) where $k-1$ random numbers are interpolated along with the secret to generate a k^{th} degree equation, the samples (taken appropriately, excluding at $x = 0$) do not provide any information about the secret which is mapped as point at $x = 0$. The first step in algorithm, hence, directly executes Shamir's secret sharing scheme. The shares so generated, can then be treated as random numbers and are reused as one of the points in further encoding of secrets.

As a result, during the share construction (assuming previous shares are treated as random number), we have used $k-1$ random numbers at each step (a length $k-2$ vector R and a sample D_{ij}^{k} from the previous iteration, see figure 3). This along with the secret (or secret piece) are used to interpolate a k^{th} degree polynomial which is sampled at n points at each step.

Now, if the dummy pieces are pre-agreed pieces (such as special characters or zeros), then each player would know, two points, one the sample given to him and the other the dummy piece itself used during interpolation, and hence would need only $k-2$ additional players to collude to recreate the node corresponding to that polynomial. This may lead to partial disclosure of secret. As a result, the dummy pieces chosen to stuff must be uniformly and randomly chosen from the field. Side information regarding the number of dummy pieces stuffed would convey to the players, how many trailing in the recreated secret pieces are to be discarded.

Assuming that the dummy pieces are randomly and uniformly chosen elements from the field, it is clear that k players need to collude in order to recreate the nodes in the last level of the tree and then proceed from there. However, since we have encoded additional secrets or created smaller effective shares, the overall security of the scheme is inversely proportional to the efficiency improvement factor. This security is in turn relative to the security achieved in information theoretically secure schemes.

6 Conclusions

This paper has presented a recursive scheme for multi-secret sharing, which in turn can be used to create shares of smaller sizes by dividing the secret into smaller pieces and then simulating multi-secret sharing. The scheme forms a recursion tree and does not require any encryption key, unlike previously proposed computational secret sharing schemes.

The efficiency of the scheme have been analyzed and it is seen that a efficiency improvement is achieved with a tradeoff against the security of the scheme and an inverse relation between the two has been established.

The proposed scheme has widespread applications in secure distributed storage and information dispersal protocols. Further, it may be used as a steganographic channel to transmit hidden information in secret sharing, which may be used for authentication and verification of shares and the reconstructed secret itself.

References

1. Naor, M. and Shamir, A.: Visual Cryptography. Advances in Cryptology-Eurocrypt, 950:1-12, 1995.
2. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In Proceedings of the 28th Annual Symposium on Foundations of Computer Science (October 12 - 14, 1987). SFCS. IEEE Computer Society, Washington, DC, 427-438, 1987.
3. Shamir, A.: How to share a secret. Commun. ACM 22, 11 (Nov. 1979), 612-613, 1979.
4. Pedersen, T. P.: Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In Proceedings of the 11th Annual international Cryptology Conference on Advances in Cryptology (August 11 - 15, 1991). J. Feigenbaum, Ed. Lecture Notes In Computer Science, vol. 576. Springer-Verlag, London, 129-140, 1992.
5. Herzberg, A., Jarecki, S., Krawczyk, H., and Yung, M.: Proactive Secret Sharing Or: How to Cope With Perpetual Leakage. In Proceedings of the 15th Annual international Cryptology Conference on Advances in Cryptology (August 27 - 31, 1995). D. Coppersmith, Ed. Lecture Notes In Computer Science, vol. 963. Springer-Verlag, London, 339-352, 1995.
6. He, J. and Dawson, E.: "Multistage secret sharing based on one-way function," Electronics Letters , vol.30, no.19, pp.1591-1592, 15 Sep 1994.
7. Gnanaguruparan, M. and Kak, S.: Recursive Hiding of Secrets in Visual Cryptography. Cryptologia 26: 68-76, 2002.
8. Horng, G., Chen, T. and Tsai D.: Cheating in Visual Cryptography. Design, Codes and Cryptography 38:219-236, 2006.
9. Prisco, R. and Santis, A.: Cheating Immune (2,n)-Threshold Visual Secret Sharing. LNCS 4116:216-228, 2006.
10. Hao Luo; Jeng-Shyang Pan; Zhe-Ming Lu: "Hiding Multiple Watermarks in Transparencies of Visual Cryptography," Intelligent Information Hiding and Multimedia Signal Processing, 2007. IIHMSP 2007. Third International Conference on , vol.1, no., pp.303-306, 26-28 Nov. 2007.
11. Hao Luo; Faxin Yu: "Data Hiding in Image Size Invariant Visual Cryptography," Innovative Computing Information and Control, 2008. ICICIC '08. 3rd International Conference on , vol., no., pp.25-25, 18-20 June 2008.
12. Luo, H., Lu, Z., and Pan, J.: Multiple Watermarking in Visual Cryptography. In Proceedings of the 6th international Workshop on Digital Watermarking (Guangzhou, China, December 03 - 05, 2008). Y. Q. Shi, H. Kim, and S. Katzenbeisser, Eds. Lecture Notes In Computer Science, vol. 5041. Springer-Verlag, Berlin, Heidelberg, 60-70, 2008.
13. Wen-Pinn Fang and Ja-Chen Lin.: Visual cryptography with extra ability of hiding confidential data. J. Electron. Imaging 15, 023020, 2006.
14. C.C. Wu, L.H. Chen.: A study on visual cryptography, Master Thesis, Institute of Computer and Information Science, National Chiao Tung University, Taiwan, R.O.C., 1998.
15. Hsien-Chu Wu, Chin-Chen Chang: Sharing visual multi-secrets using circle shares, Computer Standards & Interfaces, Volume 28, Issue 1, July 2005, Pages 123-135.

16. Hwa-Ching Hsu; Tung-Shou Chen; Yu-Hsuan Lin: "The ringed shadow image technology of visual cryptography by applying diverse rotating angles to hide the secret sharing," Networking, Sensing and Control, 2004 IEEE International Conference on , vol.2, no., pp. 996-1001 Vol.2, 2004.
17. Shyu, S. J., Huang, S., Lee, Y., Wang, R., and Chen, K.: Sharing multiple secrets in visual cryptography. Pattern Recogn. 40, 12 (Dec. 2007), 3633-3651, 2007.
18. B. Schneier: Schneier's Cryptography Classics Library: Applied Cryptography, Secrets and Lies, and Practical Cryptography, Wiley, 2007.
19. P. Rogaway and M. Bellare: "Robust computational secret sharing and a unified account of classical secret-sharing goals," in CCS '07: Proceedings of the 14th ACM Conference on Computer and Communications Security. New York, NY, USA: ACM, 2007, pp. 172-184.
20. V. Vinod, A. Narayanan, K. Srinathan, C. P. Rangan, and K. Kim: "On the power of computational secret sharing," Indocrypt 2003, vol. 2904, pp. 265-293, 2003.
21. A. Cresti: "General short computational secret sharing schemes," in Advances in Cryptology EUROCRYPT 95, volume 921 of Lecture Notes in Computer Science. Springer, 1995, pp. 194-208.
22. H. Krawczyk: "Secret sharing made short," Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology, pp. 136-146, 1994.
23. M. O. Rabin: "Efficient dispersal of information for security, load balancing and fault tolerance," Journal of the ACM, vol. 36, no. 2, pp. 335-348, 1989.
24. J. Garay, R. Gennaro, C. Jutla, and T. Rabin: "Secure distributed storage and retrieval," Theoretical Computer Science, pp. 275-289, 1997.
25. L. Harn: "Efficient sharing (broadcasting) of multiple secrets," IEE Proceedings - Computers and Digital Techniques, vol. 142, no. 3, pp. 237-240, May 1995.
26. H.-Y. Chien, J.-K. Jan, and Y.-M. Tseng: "A practical (t,n) multi-secret sharing scheme," IEICE transactions on fundamentals of electronics, communications and computer sciences, vol. 83, no. 12, pp. 2762-2765, 2000.
27. L.-J. Pang and Y.-M. Wang: "A new (t,n) multi-secret sharing scheme based on shamir's secret sharing," Applied Mathematics and Computation, vol. 167, no. 2, pp. 840 - 848, 2005.
28. C.-C. Yang, T.-Y. Chang, and M.-S. Hwang: "A (t,n) multi-secret sharing scheme," Applied Mathematics and Computation, vol. 151, no. 2, pp. 483 - 490, 2004.
29. C.-W. Chan and C.-C. Chang, "A scheme for threshold multi-secret sharing," Applied Mathematics and Computation, vol. 166, no. 1, pp. 1 - 14, 2005.
30. Parakh, A. and Kak, S.: A Recursive Threshold Visual Cryptography Scheme. Cryptology ePrint Archive, Report 2008/535, 2008.
31. G. R. Ganger, P. K. Khosla, M. Bakkaloglu, M. W. Bigrigg, G. R. Goodson, G. R, V. Pandurangan, S. Oguz, V. P, C. A. N. Soules, J. D. Strunk, and J. J. Wylie.: Survivable storage systems, in In DARPA Information Survivability Conference and Exposition, IEEE. IEEE Computer Society, 2001, pp. 184195.
32. A. Iyengar, R. Cahn, J. A. Garay, and C. Jutla.: Design and implementation of a secure distributed data repository, in In Proc. of the 14th IFIP Internat. Information Security Conf, 1998, pp. 123135.
33. S. Lakshmanan, M. Ahamad, and H. Venkateswaran.: Responsive security for stored data, International Conference on Distributed Computing Systems, vol. 0, p. 146, 2003.
34. M. Waldman, A. D. Rubin, and L. F. Cranor.: The architecture of robust publishing systems, ACM Trans. Internet Technol., vol. 1, no. 2, pp. 199230, 2001.

Appendix

6.1 Efficiency and height plots

Figures 7 and 8 show the maximum efficiency improvement factor that can be achieved and the corresponding height h for it as the number of (smallest) pieces vary in the original secret.

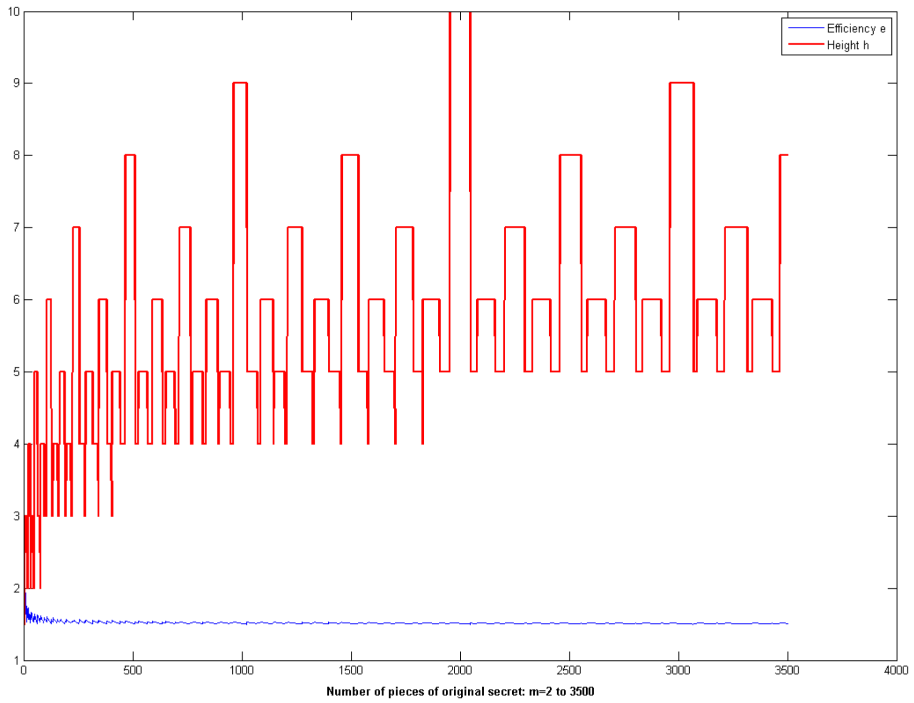
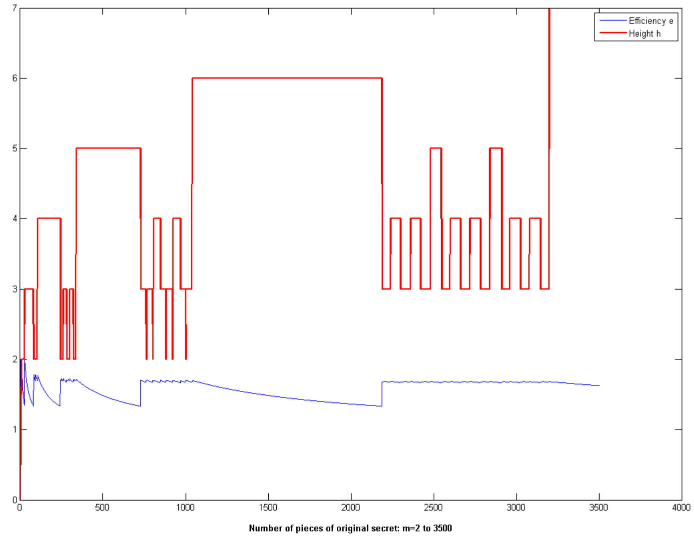


Fig. 7. Plot of maximum efficiency improvement factor with corresponding height h against m. Here n=2.

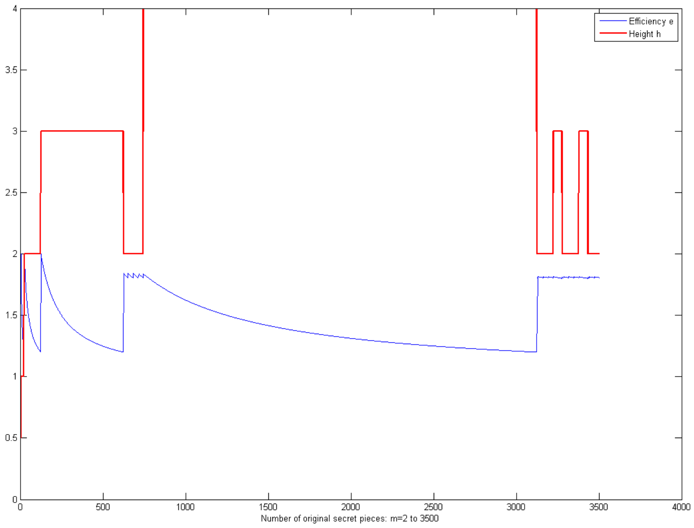
6.2 Recursive hiding in threshold visual cryptography

The idea described in [7] is applied to images to develop a recursive 2 out of 3 visual cryptographic scheme [30]. For this purpose we divide each pixel into 3 subpixels as shown in figure 9.

As seen in figure 9, when the partitions of white pixel are stacked upon each other one third of the pixel is white and hence appears light gray to human eye.



(a)



(b)

Fig. 8. Plot of maximum efficiency improvement factor with corresponding height h against m (a) n=3 (b) n=5.

However, the subpixels of the black pixel are so arranged that when 2 shares are stacked together, the resulting pixel is completely dark.

Yet another way to create subpixels would be to have only one third of the subpixel colored dark. Therefore, when subpixels of a white pixel are stacked upon each other they would appear light gray and the stacking of the subpixels of a black pixel would result in dark gray. However, the human eye can perceives the difference between gray and completely dark pixels better than two different shades of gray itself. Hence our construction of subpixels in figure 9.

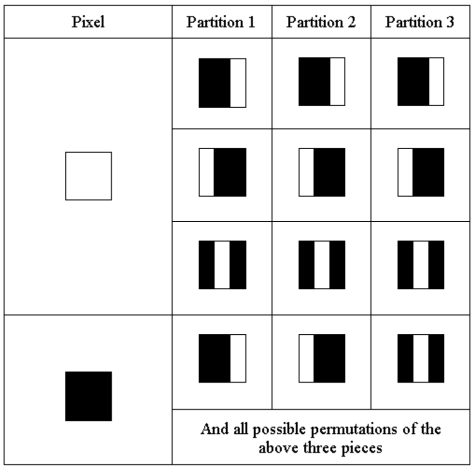


Fig. 9. Possible partitions for black and white pixels

As an example to make the working of the proposed scheme clear, we present in figure 10 the encoding of a 3×3 pixel image such that each share of the 3×3 image contains shares of a 1 pixel secret image and a 3×1 pixel secret image.

The subpixels of an original pixel can be represented as a matrix. For example if the original pixel was black then the 3 shares representing it may be written as $[100]^T$, $[010]^T$, and $[001]^T$. Since, these matrices can be stored as a sequence of bits; it implies that there is an expansion by a factor of 1×9=9 because the original black pixel can be represented as a single bit 1, while each of the 3 shares consists of 3 sub-pixels requiring 3 bits for their representation. If we were not to perform a recursive hiding, we would be creating 9×9=81 bits for each share corresponding to 9 pixels of the original image (figure 10). However, using recursive hiding we have been able to hide additional 1×9+3×9=9+27=36 bits of information in those 81 bits, thereby increasing the information conveyed per share of the original image.

Higher efficiency could be achieved if we were to number the subpixels as 0, 1, and 2 and use prefix coding to represent these numbers and store them instead of storing the matrices or pixels. This would only lead to a per bit expansion

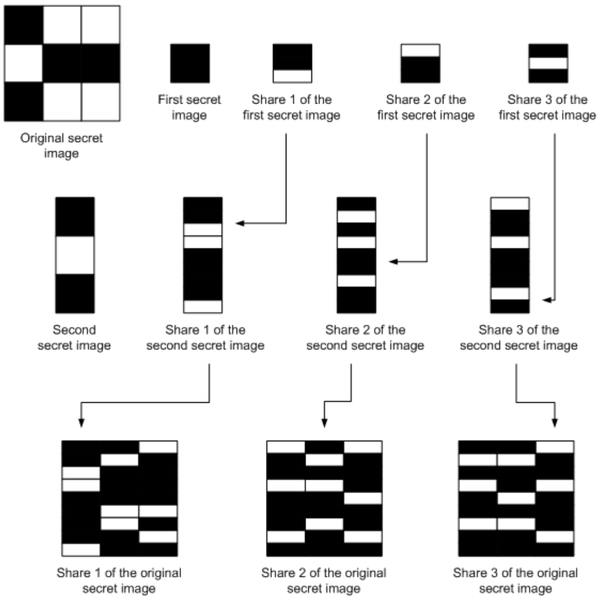


Fig. 10. Illustration of recursive hiding of secret images in shares of larger original image using a 2-out-of-3 threshold scheme

factor of 5, instead of 9 and the efficiency improvement will be similar to that in the case of text, i.e. an improvement of 40%.

Figure 11 shows the application of the proposed scheme to three images, smallest image being a Smiley face, next being a watermark and the third and the largest image being that of Lena.

Figure 12 shows the reconstruction of hidden images after appropriately extracting the smaller shares from the shares of Lena.

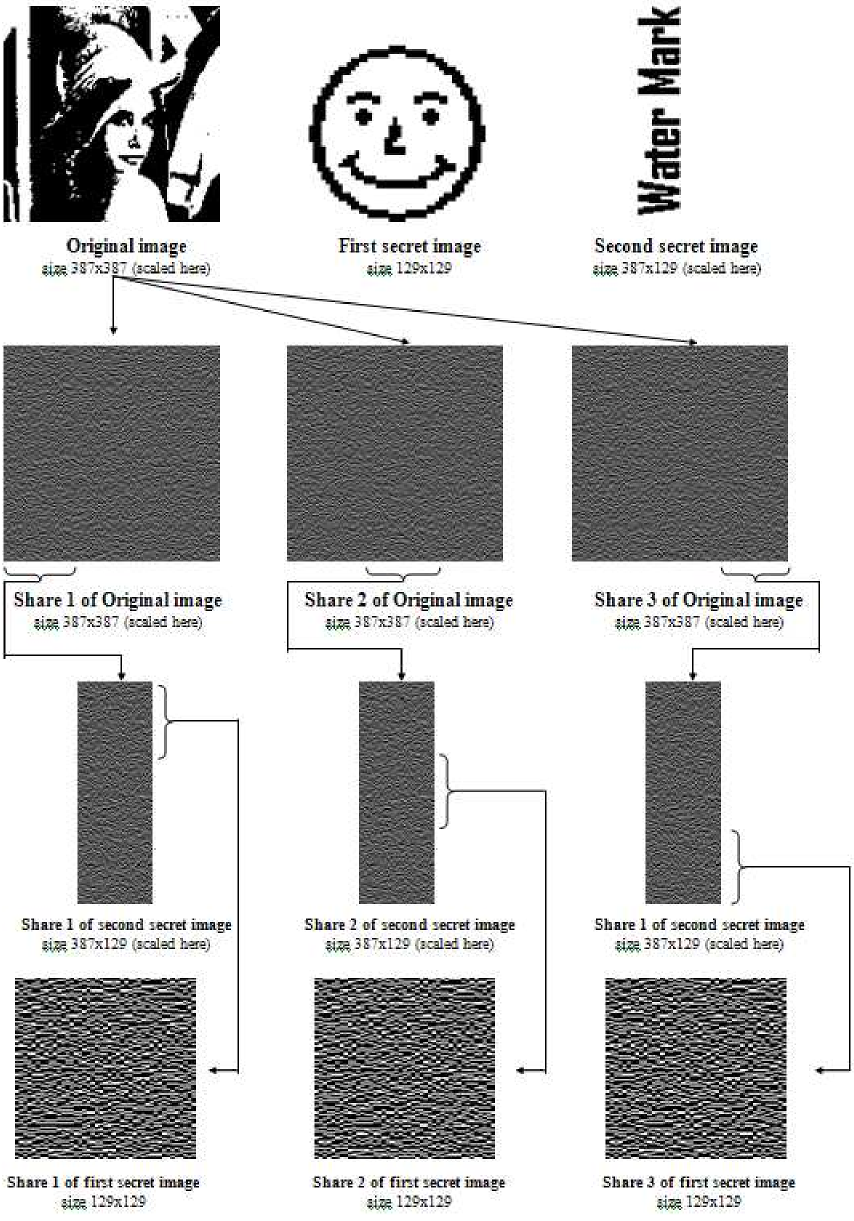


Fig. 11. Illustration of the process of recursive hiding of secrets in shares of larger original image

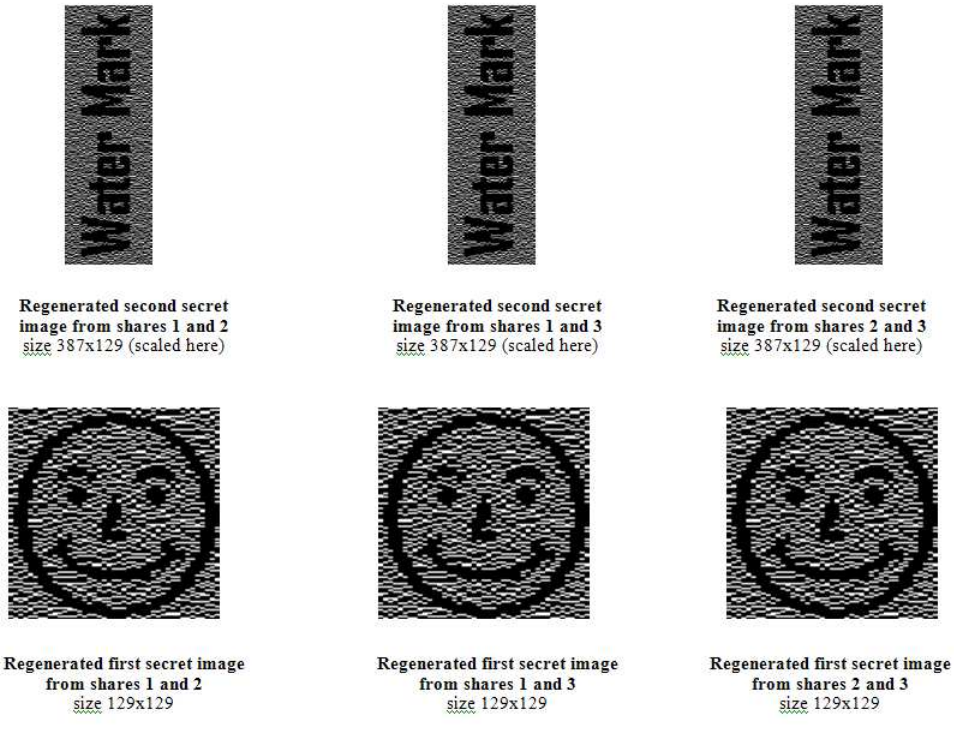


Fig. 12. Illustration of regeneration of smaller images from the shares hidden inside the shares of the original larger image