



Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Space efficient secret sharing for implicit data security

Abhishek Parakh*, Subhash Kak

Computer Science Department, Oklahoma State University, Stillwater, OK 74078, USA

ARTICLE INFO

Article history:

Received 22 September 2009

Received in revised form 13 September 2010

Accepted 14 September 2010

Available online xxxx

Keywords:

Implicit security

Data partitioning

Recursive secret sharing

Computational security

Information dispersal

ABSTRACT

This paper presents a k -threshold computational secret sharing technique that distributes a secret S into shares of size $\frac{|S|}{k-1}$, where $|S|$ denotes the secret size. This bound is close to the space optimal bound of $\frac{|S|}{k}$ if the secret is to be recovered from k shares. In other words, our technique can be looked upon as a new information dispersal scheme that provides near optimal space efficiency. The proposed scheme makes use of repeated polynomial interpolation and has potential applications in secure information dispersal on the Web and in sensor networks.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

The idea of implicit data security was introduced in [18], where the authors proposed that personal data being stored on the Web be divided into implicitly secure pieces (or partitions) and be distributed over numerous servers for storage. Data pieces are said to be implicitly secure if no explicit encryption key is used and the pieces in themselves do not reveal any information until at least a certain number of them are brought together. In [18] the authors also presented a new data partitioning scheme to achieve this. The idea of implicit security has been further investigated by other researchers and applied to key distribution in sensor networks [4].

The data partitioning scheme presented in [18] resulted in an n -fold increase in storage requirement. In resource constrained environments such as that of sensor networks this n -fold increase also necessitates n -fold increase in bandwidth and transmission costs. Further, the bandwidth requirements also apply to the Web where if a user is to securely store a large amount of data on different servers, he would require an n -fold communication bandwidth.

In this paper we relax the security conditions on implicit security to include computational security, i.e. the data partitions that are created, to be stored on the servers, may only be computationally secure, which will provide adequate security in most practical situations. Computationally secure space efficient secret sharing schemes may be used to create the data partitions, thus reducing the storage requirements and saving bandwidth in data transmission. In order to achieve this we propose a new space efficient secret sharing scheme.

Information theoretically secure secret sharing schemes are space inefficient because a k -out-of- n secret sharing technique generates n shares each of size at least that of the secret itself, leading to a n -fold increase in required storage. In order to improve space efficiency, computational secret sharing techniques have been developed [21,20,23,1] in which a symmetric key is used to encrypt the original secret and the encrypted secret is divided into k pieces to which redundancy is added by the use of block error correction techniques [19,6,13], resulting in n shares. The encryption key is split into shares using

* Corresponding author. Tel.: +1 405 385 9787.

E-mail address: parakh@cs.okstate.edu (A. Parakh).URLs: <http://cs.okstate.edu/~parakh> (A. Parakh), <http://cs.okstate.edu/~subhashk> (S. Kak).

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 a burdensome overhead. Further, the block error-correction technique used for introducing redundancy requires the storage of an $n \times k$ matrix. One of the seminal constructions of computational secret sharing is given in [13] where IDA [19] has been used for introducing redundancy. In order to recover the secret in presence of invalid shares numerous techniques add robustness using hashing-functions [12] and other methods [10], while more general implementations have appeared in [23,1,2,16,15].

Most multisecret sharing schemes require the disclosure of large amounts of public information [25,9,3,5]. For example, a method based on discrete logarithm problem for sharing multiple secrets under the assumption that shareholders do not collude is presented in [9]. Besides the burdensome public information that is to be disclosed, the security of the scheme presented in [9] is much lower than that suggested if the shareholders were to collude. Chien et al. [3] propose a multisecret scheme based on systematic block codes, while Yang et al. [25] improve on the amount of public information required by Chien et al. in [3]. Further, Yang's scheme is based on Shamir's secret sharing scheme such that fewer than the threshold number of pieces do not leak any information. Other schemes, as in [5], focus on improving efficiency of computations involved in share creation and secret reconstruction rather than space efficiency.

1.1. Recursive hiding of secrets

We present an example from earlier work [7,17] that proposed a 2-out-of-2 (and 2-out-of- n) secret sharing scheme for binary secrets, based on recursion. Although the implementation in [7] is quite different from the one proposed in this paper, it is useful in understanding the notion of recursion in secret sharing. Suppose we wish to share a random secret of size 7 bits, $S = 1011011$, into 2 shares such that both of them are needed to reconstruct the secret. A simple implementation would use exclusive-OR transformation to divide the secret into shares. Hence, using conventional scheme, we would create two shares, say $D_{s_1} = 1001101$ and $D_{s_2} = 0010110$ of size 7 bits each. However, using recursion we can create shares smaller than 7 bits each as is shown in Table 1. To execute a recursive algorithm, we first divide the 7 bit secret S into 3 pieces, s_1 , s_2 , and s_3 of size 1, 2 and 4 bits respectively. A concatenation of the pieces $s_1s_2s_3$ is equal to S . Table 1 illustrates the recursive process. The first row in the table shows the creation of two shares of s_1 using exclusive-OR. In the second row, two shares of s_2 are created but these shares partially use the shares of s_1 (the bits in bold are bits from the shares of s_1). Similarly, the third row of the table shows the creation of two shares of s_3 using the shares of s_2 , where again the bold bits depict the shares that are carried over from the previous step. Only the final shares 0010 and 1001 are shared between parties and exclusive-OR has been used at each intermediate step.

The final shares 0010 and 1001 contain the complete secret S . To recover the secret, the parties reconstruct the smaller pieces by properly aligning the shares by traversing up the table and using exclusive-OR; the concatenation of these pieces is the secret. For example, to recreate s_3 the two final shares are exclusive-ORed, while to recreate s_2 the first two bits from $D_{s_{31}}$ and the last two bits from $D_{s_{32}}$ are extracted and exclusive-ORed. Similarly, to recreate s_1 the first and the last bits of shares of s_2 are extracted and exclusive-ORed.

In the above example we have achieved a significant improvement in share sizes from 7 bits each in a conventional scheme to 4 bits each in a recursive scheme. However, the resulting shares have a lower security compared to conventional exclusive-OR based implementations because each party now only needs to determine 4 bits of the other share to recreate the complete secret on its own.

Nevertheless, for a file, say of size $2^{11} - 1$ bits (about 2 Kb), that must be divided into shares, each share would be of size 2^{10} (1 Kb) which may provide adequate security for many applications because each party would need to try on the order of $2^{2^{10}}$ possibilities for the other share.

Although the example is a good representative to explain recursion in secret sharing, it nonetheless has the following limitations. Firstly, it does not provide a threshold scheme. Secondly, secrets that are encoded using the above method may only be of size $2^t - 1$ for some positive integer t ; this is due to the binary tree structure of the scheme (Table 1). Thirdly, it does not provide a mechanism to control the security of the resulting shares, i.e. if $2^t - 1$ bits are encoded, the resulting shares are always of size 2^{t-1} . Note that the security of the scheme depends on share sizes.

This paper overcomes the limitations of [7] and proposes a secret sharing scheme that encodes a secret S into shares of size $\frac{|S|}{k-1}$, where $|S|$ is the size of the secret, without the use of any encryption key. This is close to the optimal bound of $\frac{|S|}{k}$, if the

Table 1

Illustration of recursive secret sharing. A large secret $S = 1011011$ is divided into three consecutive pieces and then the shares are recursively encoded. Only the final shares 0010 and 1001 need be shared between parties.

Secret pieces	Shares				
s_1 : 1	0				$D_{s_{11}}$
	1				$D_{s_{12}}$
s_2 : 01	0	0			$D_{s_{21}}$
	0	1			$D_{s_{22}}$
s_3 : 1011	0	0	1	0	$D_{s_{31}}$
	1	0	0	1	$D_{s_{32}}$

secret is to be recovered from k shares (please see [13]). It is well known that in order to maintain ideal security (information theoretic security) each share must be at least the size of the secret itself. Since we produce shares of size smaller than that of the secret, each share leaks some information about the secret. However, in certain cases, such as in the case of large secrets, direct application of our scheme may provide suitable level of security (which will become clear below). Our proposal is based on recursion, in which a secret is first divided into $k - 1$ pieces and then the pieces are encoded one by one in such a manner that the shares of the already encoded pieces are reused to create new shares for the next piece.

In order to further improve the security, specially in the case of small secrets, we present an extension to our scheme, similar to that proposed by Krawczyk [12], where one may use an encryption key to first encrypt the secret and then divide the encrypted secret into pieces of size $\frac{|S|}{k-1}$ using the new recursive algorithm.

Further, we provide a mechanism to control the size of the resulting shares ranging from $\frac{|S|}{k-1}$ to $|S|$ thus controlling the final security level achieved by the shares (where shares of size $|S|$ provide maximum security).

The proposed recursive secret sharing scheme has applications in distributed online storage of information discussed in [18,19,6]. Systems implementing such distributed data storage have appeared at CMU and IBM [20,8,11,14,24].

The advantages of our new data partitioning scheme may be summarized as follows:

1. The share sizes are on the order of $\frac{|S|}{k-1}$.
2. No side information needs to be stored with the shares.
3. No public information needs to be disclosed (other than the field used).
4. The scheme may be used as a computational secret sharing scheme, information dispersal scheme or as a multisecret sharing scheme.

In Section 2 we present the proposed secret sharing scheme and discuss its security together with an illustrative example. Section 3 talks about how to control the security of proposed scheme by varying the share sizes and the number of random elements chosen. Section 4 is the conclusion.

2. Space efficient secret sharing

The proposed scheme recursively builds upon polynomial interpolation and sampling similar to that presented in [22]. However, Shamir's scheme generates shares of size $|S|$ for a secret S . In contrast, we generate shares of size $\frac{|S|}{k-1}$.

We first briefly review the scheme presented by Shamir [22] (Algorithm 1).

Algorithm 1. Shamir's secret sharing scheme

1. Choose a prime p , $p > \max(S, n)$, where $S \in \mathbb{Z}_p$ is the secret.
2. Choose $k - 1$ random numbers a_1, a_2, \dots, a_{k-1} , uniformly and independently, from the field \mathbb{Z}_p .
3. Using a_i , $1 \leq i \leq (k - 1)$ and secret S , generate polynomial $f(x)$ of degree $k - 1$,

$$f(x) = S + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1} \pmod{p}.$$

4. Sample $f(x)$ at n points $D_i = f(i)$, $1 \leq i \leq n$ such that the shares are given by (i, D_i) .

Reconstruction of the secret is performed by interpolating any k points (shares) and evaluating $S = f(0)$.

In our implementation, we first divide secret S into $k - 1$ pieces. The division of a secret file F was discussed by Rabin in [19] with the aim of information dispersal. For example, to share a secret of size $|S| = 30$ bits, he divides it into say 3 pieces of 10 bits each. In general, we divide the secret into pieces of size $\frac{|S|}{k-1}$. Let the pieces be denoted by s_i , $1 \leq i \leq (k - 1)$, such that their concatenation $s_1s_2 \dots s_{k-1} = S$.

For the proposed scheme assume a finite field \mathbb{Z}_p , where p is a prime and $p > \max(s_{\max}, n)$, where $s_{\max} = \max(s_i)$, $1 \leq i \leq (k - 1)$, and s_1, s_2, \dots, s_{k-1} are the pieces of S . The shares will be denoted as $D_{s_1,1}, D_{s_1,2}, \dots, D_{s_m,m}$ at the intermediate stages, where $1 \leq m \leq i + 1$ and D_1, D_2, \dots, D_n at the final stage. (Note that as in Shamir's scheme, $D_{s,m}$ are the y -coordinates only while the respective x -coordinates m are tacitly known to all parties.)

It is to be noted that since for all practical cases $n < \frac{|S|}{k}$, the stated share size $\frac{|S|}{k}$ is the optimal size of a share for a k -out-of- n secret sharing scheme. But strictly speaking if $n > \frac{|S|}{k}$, then the shares would be of size $|n|$. In this paper we will ignore the latter case and assume $n < \frac{|S|}{k}$ (this is also true for Shamir's secret sharing scheme).

Formally, a *space optimal* (k, n) secret sharing scheme is a secret sharing scheme that, for a secret S produces shares of size $\frac{|S|}{k}$. And a *space efficient secret sharing scheme* is a secret sharing scheme that approaches the optimal factor of $\frac{|S|}{k}$ in terms of share size. In general, we say that a secret scheme provides security on the order of share sizes it produces.

The proposed scheme works as follows: We randomly and uniformly choose a number $a_1 \in \mathbb{Z}_p$ and generate 1st degree polynomial $f_1(x) = a_1x + s_1$. Then we sample $f_1(x)$ at two points $D_{s_1,1} = f_1(1)$ and $D_{s_1,2} = f_1(2)$, to generate two shares for s_1 . This first step can be viewed as a direct execution of Shamir's (2,2) secret sharing scheme. Next we use these two shares of s_1 to generate polynomial $f_2(x) = D_{s_1,2}x^2 + D_{s_1,1}x + s_2$, where the coefficients are the previous two shares and the free term is the new piece. Sampling $f_2(x)$ at three points $D_{s_2,1} = f_2(1)$, $D_{s_2,2} = f_2(2)$, and $D_{s_2,3} = f_2(3)$, generates three shares of s_2 . We can now delete $D_{s_1,1}$ and $D_{s_1,2}$ because the new shares $D_{s_2,1}$, $D_{s_2,2}$, and $D_{s_2,3}$ encode the shares of s_1 within themselves. We then

use the shares of s_2 to create a 3^{rd} degree polynomial $f_3(x)$ with s_3 as its free term, and generate shares for s_3 by sampling the newly created polynomial at 4 points. These four points, D_{s_31} , D_{s_32} , D_{s_33} , and D_{s_34} have the shares of s_1, s_2 as well as s_3 and therefore D_{s_21} , D_{s_22} and D_{s_23} can now be deleted. The process is repeated for pieces s_4, s_5, \dots, s_{k-2} by creating $f_4(x), f_5(x), \dots, f_{k-2}(x)$ and repetitive sampling and reusing of shares and deleting the older shares. At the last step, we generate a polynomial $f_{k-1}(x) = D_{s_{k-2}(k-1)}x^{k-1} + D_{s_{k-2}(k-2)}x^{k-2} + \dots + D_{s_{k-2}1}x + s_{k-1}$ and sample it at n points $D_1 = f_{k-1}(1), D_2 = f_{k-1}(2), \dots, D_n = f_{k-1}(n)$, such that the final shares are given by $(i, D_i), 1 \leq i \leq n$. These final n shares have recursively hidden $k - 1$ secrets within themselves. Fig. 1 illustrates the process.

Algorithm 2. Dealing phase

1. Choose a prime $p, p > \max(s_{max}, n)$, where $s_{max} = \max(s_i), 1 \leq i \leq (k - 1)$, and s_1, s_2, \dots, s_{k-1} are the pieces of secret S .
2. Randomly and uniformly choose a number $a_1 \in \mathbb{Z}_p$ and generate polynomial $f_1(x) = a_1x + s_1$.
3. Sample $f_1(x)$ at two points $D_{s_11} = f_1(1)$ and $D_{s_12} = f_1(2)$, which represent two shares of s_1 .
4. Do for $2 \leq i \leq (k - 1)$

(a) Generate polynomial,

$$f_i(x) = D_{s_{i-1}i}x^i + D_{s_{i-1}(i-1)}x^{i-1} + \dots + D_{s_{i-1}1}x + s_i.$$

(b) Sample $f_i(x)$ to create new shares,

- i. If $i < k - 1$, sample at $i + 1$ points:

$$D_{s_i1} = f_i(1)$$

$$D_{s_i2} = f_i(2)$$

⋮

$$D_{s_i(i+1)} = f_i(i + 1).$$

- ii. If $i = k - 1$, sample at n points:

$$D_1 = f_i(1)$$

$$D_2 = f_i(2)$$

⋮

$$D_n = f_i(n).$$

(c) Delete old shares: $D_{s_{i-1}1}, D_{s_{i-1}2}, \dots, D_{s_{i-1}i}$.

5. The final n shares are given by $(i, D_i), 1 \leq i \leq n$.

Algorithm 2 (Reconstruction phase).

1. Interpolate any k shares (i, D_i) to generate the polynomial of degree $k - 1$,

$$f_{k-1}(x) = D_{s_{k-2}(k-1)}x^{k-1} + D_{s_{k-2}(k-2)}x^{k-2} + \dots + D_{s_{k-2}1}x + s_{k-1}$$

and evaluate $s_{k-1} = f_{k-1}(0)$.

2. Do for all $i = k - 2$ down to 1

(a) Interpolate $i + 1$ shares given by $(m + 1, D_{s_i(m+1)}), 0 \leq m \leq i$ obtained from coefficients of $f_{i+1}(x)$ to generate polynomial of degree i ,

$$f_i(x) = D_{s_{i-1}i}x^i + D_{s_{i-1}(i-1)}x^{i-1} + \dots + D_{s_{i-1}1}x + s_i.$$

(b) Evaluate $s_i = f_i(0)$.

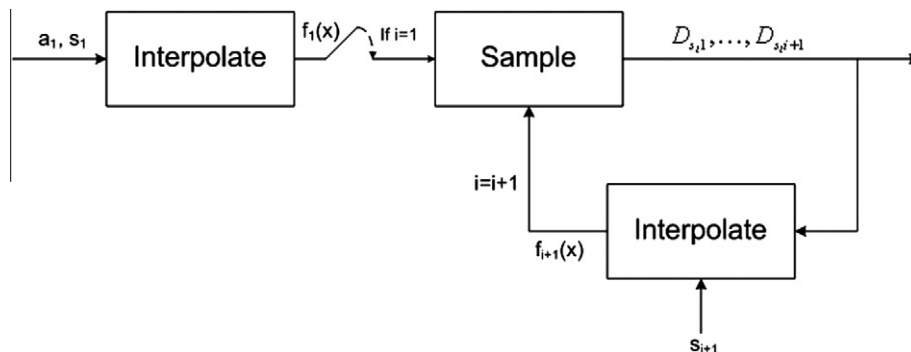


Fig. 1. Illustration of recursive secret sharing.

As seen above the reconstruction of pieces is straightforward and it proceeds in a last-in, first-out manner. Any k of the parties can interpolate the polynomial of degree $k - 1$ such that the free term represents $s_{k-1} = f_{k-1}(0)$. Then using the $k - 1$ coefficients of this polynomial as points (leaving out the free term which is s_{k-1}), interpolate the polynomial of degree $k - 2$ to obtain s_{k-2} . This process is repeated until we obtain s_1 .

Algorithm 2 clearly generates shares of size $|s_i|$, where $|s_i| = \frac{|S|}{k-1}$, for all $i = 1$ to $k - 1$.

For a secret message or file of size 3 KB, with $k = 4$ and $n = 7$, we would create $k - 1 = 3$ pieces of size approximately 1 KB ($= 8 \times 10^3$ bits) each and choose a prime $p > 8 \times 10^3$ bits. Therefore, when $k - 1$ parties collude, the resulting security is on the order of $\frac{1}{p}$ or $\frac{1}{2^{10^3}}$. Consequently, after collusion of $k - 1$ parties there remains 2^{10^3} equally probable possibilities for the k th share.

Algorithmic complexity: **Algorithm 2** uses repeated polynomial interpolation and sampling which has a complexity of $O(b \times \log^2 b)$, where b is the number of points being used. The proposed algorithm interpolates polynomials with increasing number of points at every step so that the complexity of the last step is $O(n \times \log^2 n)$. Hence, the algorithmic complexity is given by $\sum_{b=1}^{k-1} b \times \log^2 b + n \times \log^2 n$.

Example. Assume that we have a secret $S = 17280512$ that is broken into 4 pieces $s_1 = 17$, $s_2 = 28$, $s_3 = 05$, and $s_4 = 12$. Here the pieces are created with a decimal base, i.e. number of decimal digits in the pieces (2 digits each). S is to be shared between 7 parties such that any 5 of them can reconstruct all the 4 secrets. We can now use a prime $p = 31$. (If one were to use a conventional method for secret sharing, one would have to choose a large prime $p > 17280512$, which clearly will yield pieces of that order; however here $p = 31$ suffices, yielding smaller pieces.)

2.1. Dealing phase

1. Randomly and uniformly choose a number $a_1 \in \mathbb{Z}_p$. Let $a_1 = 22$. Generate polynomial, $f_1(x) = a_1x + s_1 = 22x + 17 \pmod{31}$.
2. Sample $f_1(x)$ at two points to generate two shares of piece s_1 , i.e. $D_{s_{11}} = f_1(1) = 8$ and $D_{s_{12}} = f_1(2) = 30$.
3. Generate polynomial $f_2(x) = D_{s_{12}}x^2 + D_{s_{11}}x + s_2 = 30x^2 + 8x + 28$.
4. Sample $f_2(x)$ at 3 points to generate three shares of s_2 , i.e. $D_{s_{21}} = f_2(1) = 4$, $D_{s_{22}} = f_2(2) = 9$, and $D_{s_{23}} = f_2(3) = 12$.
5. Delete $D_{s_{11}}$ and $D_{s_{12}}$.
6. Generate polynomial $f_3(x) = D_{s_{23}}x^3 + D_{s_{22}}x^2 + D_{s_{21}}x + s_3 = 12x^3 + 9x^2 + 4x + 5$.
7. Sample $f_3(x)$ at 4 points to generate four shares of s_3 , i.e. $D_{s_{31}} = f_3(1) = 30$, $D_{s_{32}} = f_3(2) = 21$, $D_{s_{33}} = f_3(3) = 19$, and $D_{s_{34}} = f_3(4) = 3$.
8. Delete $D_{s_{21}}$, $D_{s_{22}}$, and $D_{s_{23}}$.
9. Generate polynomial

$$f_4(x) = D_{s_{34}}x^4 + D_{s_{33}}x^3 + D_{s_{32}}x^2 + D_{s_{31}}x + s_4 = 3x^4 + 19x^3 + 21x^2 + 30x + 12.$$

10. Sample $f_4(x)$ at 7 points, which represents the final 7 shares. Hence, $D_1 = f_4(1) = 23$, $D_2 = f_4(2) = 15$, $D_3 = f_4(3) = 24$, $D_4 = f_4(4) = 3$, $D_5 = f_4(5) = 8$, $D_6 = f_4(6) = 12$, and $D_7 = f_4(7) = 29$.
11. Delete $D_{s_{31}}$, $D_{s_{32}}$, $D_{s_{33}}$, and $D_{s_{34}}$.

The final seven shares are given by $(1, D_1) = (1, 23)$; $(2, D_2) = (2, 15)$; $(3, D_3) = (3, 24)$; $(4, D_4) = (4, 3)$; $(5, D_5) = (5, 8)$; $(6, D_6) = (6, 12)$; and $(7, D_7) = (7, 29)$.

2.2. Reconstruction phase

All the four pieces can be reconstructed using any 5 out of 7 final shares.

Using 5 shares, say $(1, 23)$, $(3, 24)$, $(4, 3)$, $(5, 8)$, and $(7, 29)$, we can interpolate the 4th degree polynomial $f_4(x) = 3x^4 + 19x^3 + 21x^2 + 30x + 12 \pmod{31}$, thus retrieving piece s_4 (the free term of the polynomial) by evaluating $s_4 = f_4(0)$.

Then extracting the coefficients of $f_4(x)$ and using them as y -coordinates of points $x = 1, 2, 3$, and 4 , i.e. $(1, 30)$, $(2, 21)$, $(3, 19)$, and $(4, 3)$ we can regenerate the 3rd degree polynomial $f_3(x) = 12x^3 + 9x^2 + 4x + 5$ by interpolation and retrieve the s_3 as the free term, $s_3 = f_3(0)$.

The coefficients of $f_3(x)$ are then used as points $(1, 4)$, $(2, 9)$, $(3, 13)$ to interpolate $f_2(x) = 30x^2 + 8x + 28$ and reconstruct $s_2 = f_2(0)$.

The coefficients of $f_2(x)$ are used as $(1, 8)$ and $(2, 30)$ to interpolate $f_1(x) = 22x + 17$ and reconstruct $s_1 = f_1(0)$.

The algorithm simulates a Last In First Out (LIFO) queue.

Discussion 1: Although the example is helpful in understanding the algorithm, it also brings out one of its weaknesses, i.e. for small secrets, the security provided by the algorithm may not be sufficient from a secret sharing point of view. However, **Algorithm 2** provides adequate security for sharing of large secrets and is a good information dispersal scheme in other cases.

In order to apply the algorithm for sharing of small secrets, one may do the following:

1. Choose a secret encryption key K and encrypt the secret S using the encryption key. Let us denote the encrypted secret as $E(S)$.
2. Use **Algorithm 2** to create n partitions of the encrypted secret $E(S)$. Denote these partitions as D_1, D_2, \dots, D_n .

3. Use Shamir's secret sharing scheme to create n shares K_1, K_2, \dots, K_n of the encryption key K .
4. Distribute shares $S_i = (D_i, K_i)$ to i parties.

The above construction is similar that of Krawczyk's [13] construction of computational secret sharing scheme, with a difference that Krawczyk uses IDA to create the partitions of the encrypted secret, which requires the storage of an additional $n \times k$ matrix. Even if this matrix is published on a public notice board, it is an enormous overhead. This is because each of the element of the $n \times k$ matrix is on the order of $\frac{|S|}{k}$ in size.

In contrast, we do not require any such matrix to be stored.

Discussion 2: It is clear that the scheme can be used for multi-secret sharing where instead of the pieces of the secret, s_i are different secrets themselves. Thus, $k - 1$ different secrets can be encoded within the n shares such that any k of them can recreate all the $k - 1$ secrets.

Security of the protocol: Assume that a secret S is broken into $k - 1$ pieces such that $|s_1| = |s_2| = \dots = |s_{k-1}|$ and each $s_i \in \mathbb{Z}_p$ from some prime p , or assume s_i are $k - 1$ secrets (as in the case of multisecret sharing). Further, let each s_i be a random number from the field, then we can say the following: *If $k - 1$ parties collude then there remain p equally likely possibilities for the k th share.*

To prove the above statement, we use an inductive procedure:

Step 1: Since s_1 is a random element from the field \mathbb{Z}_p . Apply Shamir's (2,2) secret sharing scheme for s_1 :

Choose randomly and uniformly a number $a_1 \in \mathbb{Z}_p$ and interpolate a first degree polynomial $f_1(x) = a_1x + s_1$. Sample, $f_1(x)$ at two points $D_{s_1,1} = f_1(1)$ and $D_{s_1,2} = f_1(2)$.

Since, Shamir's scheme is information theoretically secure, given $D_{s_1,1}, D_{s_1,2}$ remains equally likely to be any element from \mathbb{Z}_p , and vice versa. Consequently, we can treat $D_{s_1,1}$ and $D_{s_1,2}$ as random numbers from the field and can then be used to interpolate $f_2(x)$.

Step 2: Assume that we have generated $k - 1$ shares, $D_{s_{k-2},1}, D_{s_{k-2},2}, \dots, D_{s_{k-2},k-1}$ and encoded $k - 2$ secrets and that $D_{s_{k-2},i}$ can be treated as random elements from the field \mathbb{Z}_p .

Using $D_{s_{k-2},i}$ as random numbers, interpolate a polynomial of $(k - 1)$ th degree: $f_{k-1}(x) = D_{s_{k-2},k-1}x^{k-1} + D_{s_{k-2},k-2}x^{k-2} + \dots + D_{s_{k-2},1}x + s_{k-1}$. Sample f_{k-1} at n points, D_1, D_2, \dots, D_n , which are the final shares.

Since s_{k-1} was equally likely to be any element from the field, by the properties of interpolation, the knowledge of any $k - 1$ shares D_i leave p equally likely possibilities for the k th share.

3. Controlling the security

It is clear from the construction provided in the previous section that we are trading security for share size. Information theoretic security is achieved when the secret is not divided into pieces and Shamir's scheme is directly applied, creating shares of size as large as the complete secret itself. Consequently, least security is achieved when the shares are $\frac{1}{k}$ th the size of the secret, which however may be an adequate level of security for large secrets and certain applications. The proposed algorithm achieves a factor of $\frac{1}{k-1}$ for the size of the share.

Instead of dividing the secret into $k - 1$ pieces we may divide the secret into $k - 2$ pieces, or $k - 3$ pieces, \dots , or no pieces at all depending on desired level of security. Hence, if in general, we denote by m the number of pieces into which the secret is divided then replacing all occurrences of $k - 1$ with m , the algorithm proceeds as follows:

1. Choose a prime p , $p > \max(s_{max}, n)$, where $s_{max} = \max(s_i)$, $1 \leq i \leq m$, and s_1, s_2, \dots, s_m are the pieces of secret S .
2. Randomly and uniformly choose $k - m$ numbers $a_i \in \mathbb{Z}_p$ and generate polynomial $f_1(x) = a_{k-m}x^{k-m} + a_{k-m-1}x^{k-m-1} + \dots + a_1x + s_1$.
3. Sample $f_1(x)$ at $k - m + 1$ points $D_{s_1,1} = f_1(1), \dots, D_{s_1,(k-m+1)} = f_1(k - m + 1)$, which represent $k - m + 1$ shares of s_1 .
4. If $m \geq (k - m + 1)$
 - (a) $j = k - m + 1$.
 - (b) Do for $2 \leq i \leq m$
 - i. Generate polynomial,

$$f_i(x) = D_{s_{i-1},j}x^j + D_{s_{i-1},(j-1)}x^{j-1} + \dots + D_{s_{i-1},1}x + s_i.$$

- ii. Sample $f_i(x)$ to create new shares,

A. If $i < m$, sample at $j + 1$ points:

$$D_{s_i,1} = f_i(1)$$

$$D_{s_i,2} = f_i(2)$$

\vdots

$$D_{s_i,(j+1)} = f_i(j + 1).$$

B. If $i = m$, sample at n points:

$$D_1 = f_i(1)$$

$$D_2 = f_i(2)$$

⋮

$$D_n = f_i(n).$$

iii. Delete old shares: $D_{s_{i-1}1}, D_{s_{i-1}2}, \dots, D_{s_{i-1}j}$.

iv. $j = j + 1$

5. The final n shares are explicitly given by (i, D_i) , $1 \leq i \leq n$.

The reconstruction phase works in a manner similar to that presented in Algorithm 2 and the details are omitted here.

The resulting security, when $k - 1$ parties collude, in the general case, can be written as $\frac{1}{p}$ (which is the probability of correctly guessing the k th share with only the knowledge of $k - 1$ shares) where the size of p is on the order of $|\frac{S}{m}|$. Here, m may therefore be called a *security factor*.

4. Conclusions

We have presented a recursive scheme that generates shares of size $\frac{|S|}{k-1}$ for a secret S . The scheme is general and it places no restriction on the secret size. The results are close to the optimal factor of $\frac{|S|}{k}$ and represent significant improvement over conventional secret sharing schemes that generate shares of size $|S|$.

A general case when the shares are of size $\frac{|S|}{m}$, m being the security factor varying from 1 to $k - 1$, is presented. The security upon collusion of $k - 1$ parties is $\frac{1}{p}$ where size of p depends on m .

References

- [1] P. Beguin, A. Cresti, General short computational secret sharing schemes, in: *Advances in Cryptology Eurocrypt 95*, LNCS, vol. 921, Springer, 1995, pp. 194–208.
- [2] C. Cachin, On-line secret sharing, in: *IMA Conference on Cryptography and Coding*, LNCS, vol. 1025, Springer, 1995, pp. 190–198.
- [3] Hung-Yu Chien, Jinn-Ke Jan, Yuh-Min Tseng, A practical (t, n) multi-secret sharing scheme, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 83 (12) (2000) 2762–2765.
- [4] Chien-Wen Chiang, Chih-Chung Lin, R.-I. Chang, A new scheme of key distribution using implicit security in wireless sensor networks, in: *Proceedings of the 12th International Conference on Advanced Communication Technology (ICACT)*, vol. 1, February 2010, pp. 151–155.
- [5] Massoud Hadian Dehkordi, Samaneh Mashhadi, New efficient and practical verifiable multi-secret sharing schemes, *Information Sciences* 178 (9) (2008) 2262–2274.
- [6] J. Garay, R. Gennaro, C. Jutla, T. Rabin, Secure distributed storage and retrieval, *Theoretical Computer Science* (1997) 275–289.
- [7] M. Gnanaguruparan, S. Kak, Recursive hiding of secrets in visual cryptography, *Cryptologia* 26 (2002) 68–76.
- [8] Gregory R. Ganger, Pradeep K. Khosla, Mehmet Bakkaloglu, Michael W. Bigrigg, Garth R. Goodson, Garth R. Vijay Pandurangan, Semih Oguz, Vijay P. Craig A.N. Soules, John D. Strunk, Jay J. Wylie, Survivable storage systems, in: *DARPA Information Survivability Conference and Exposition*, IEEE, IEEE Computer Society, 2001, pp. 184–195.
- [9] L. Harn, Efficient sharing (broadcasting) of multiple secrets, *IEE Proceedings – Computers and Digital Techniques* 142 (3) (1995) 237–240.
- [10] Lein Harn, Changlu Lin, Strong (n, t, n) verifiable secret sharing scheme, *Information Sciences* 180 (16) (2010) 3059–3064.
- [11] Arun Iyengar, Robert Cahn, Juan A. Garay, Charanjit Jutla, Design and implementation of a secure distributed data repository, in: *Proceedings of the 14th IFIP International Information Security Conference*, 1998, pp. 123–135.
- [12] H. Krawczyk, Distributed fingerprints and secure information dispersal, in: *Twelfth Annual ACM Symposium on Principles of Distributed Computing (PODC 1993)*, ACM Press, 1993, pp. 207–218.
- [13] H. Krawczyk, Secret sharing made short, in: *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, 1994, pp. 136–146.
- [14] Subramanian Lakshmanan, Mustaque Ahamad, H. Venkateswaran, Responsive security for stored data, in: *International Conference on Distributed Computing Systems*, 2003, p. 146.
- [15] Zhihui Li, Ting Xue, Hong Lai, Secret sharing schemes from binary linear codes, *Information Sciences* 180 (22) (2010) 4412–4419.
- [16] A. Mayer, M. Yung, Generalized secret sharing and group-key distribution using short keys, in: *Compression and Complexity of Sequences 1997*, IEEE Press, 1997, pp. 30–44.
- [17] A. Parakh, S. Kak, A recursive threshold visual cryptography scheme, *Cryptology ePrint Archive*, Report 535, 2008.
- [18] Abhishek Parakh, Subhash Kak, Online data storage using implicit security, *Information Sciences* 179 (19) (2009) 3323–3331.
- [19] M.O. Rabin, Efficient dispersal of information for security, load balancing and fault tolerance, *Journal of the ACM* 36 (2) (1989) 335–348.
- [20] Phillip Rogaway, Mihir 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*, ACM, 2007, pp. 172–184.
- [21] Bruce Schneier, *Schneier's Cryptography Classics Library: Applied Cryptography, Secrets and Lies, and Practical Cryptography*, Wiley, 2007.
- [22] A. Shamir, How to share a secret, *Communications of ACM* 22 (11) (1979) 612–613.
- [23] V. Vinod, A. Narayanan, K. Srinathan, C.P. Rangan, K. Kim, On the power of computational secret sharing, in: *Indocrypt 2003*, vol. 2904, 2003, pp. 265–293.
- [24] Marc Waldman, Aviel D. Rubin, Lorrie Faith Cranor, The architecture of robust publishing systems, *ACM Transactions Internet Technology* 1 (2) (2001) 199–230.
- [25] Chou-Chen Yang, Ting-Yi Chang, Min-Shiang Hwang, A (t, n) multi-secret sharing scheme, *Applied Mathematics and Computation* 151 (2) (2004) 483–490.