

The Latin squares and the secret sharing schemes

Chi Sing Chum and Xiaowen Zhang

Abstract. The Latin square is a good candidate in a secret sharing scheme to represent a secret, because of the huge number of the Latin squares for a reasonably large order. This makes outsiders difficult to discover the secret due to tremendous possibilities. We can improve the efficiency by distributing the shares of the critical set, instead of the full Latin square, to the participants. By different critical sets of the same Latin square, different secret sharing schemes can be implemented. However, finding a critical set of a large order Latin square is very difficult. This makes the implementation of Latin square based secret sharing scheme hard. We explore these limitations, then we propose to apply herding hash technique to overcome them.

Keywords. Secret sharing scheme, Latin square, partial Latin square, critical set, hash functions, herding and Nostradamus attack.

2010 Mathematics Subject Classification. 05B15, 94A60, 94A62.

1 Introduction

How to set up an effective procedure to keep a secret is important. However, how to represent the secret is equally important. If we can discover the secret by exhaustive search, then we can bypass the secret sharing scheme, no matter how good it is. Also, it would be efficient to keep the secret short, and difficult to discover at the same time. Latin square is a good candidate in a secret sharing scheme. We can use a Latin square to represent the secret, because of the huge number of different Latin squares for a reasonably large order. For example, there are about 10^{37} different Latin squares of order 10. This makes it difficult for outsiders to discover the secret without any knowledge due to the tremendous possibilities. We can even improve the efficiency by distributing the shares of the critical set, instead of the full Latin square, to the participants. Whenever any group of the participants join together to form any critical set, the original Latin square and hence the secret can be recovered.

There are Latin square based secret sharing schemes in the literature. Cooper, Donovan, Seberry [9] used critical sets of Latin square in the design of secret sharing schemes. Their schemes are not perfect because each share of a participant is a component of a critical set, therefore, each share contains partial information of

the secret. Chaudhry and Seberry [5] had another secret sharing scheme based on critical sets of Room squares. Due to the same reason, this scheme is not perfect, either. Chaudhry, Ghodosi, Seberry [4] proposed a perfect secret sharing scheme from Room squares, but the scheme is not flexible, nor ideal. Each participant needs to have different shares for different authorized subsets he/she belongs to. To summarize, there are practical limitations to implement such secret sharing schemes due to the limited knowledge about Latin squares and their critical sets.

In order to conquer the aforementioned limitations, we propose to apply cryptographic hash functions, herding attack technique to Latin square based secret sharing schemes. We can use hash functions to store a partial Latin square in a hash, such a partial Latin square is easily extended to the full Latin square. Then, we set up a Latin square based ideal perfect $(t + 1, n)$ threshold scheme, which utilizes the herding hash function and Nostradamus attack technique to iterative hash functions. This applies to any general secret sharing scheme. We further show how to set up a verifiable secret sharing scheme by using two hash functions. The flexibility and security of our proposed schemes are dramatically improved.

In the following sections, we will review some basic properties of the cryptographic hash function, secret sharing scheme, and Latin square. Then, we will discuss why Latin square or its critical set is a good choice for secret representation and its relationship to the secret sharing scheme. Further, we will enumerate the limitations of the Latin square in a secret sharing scheme. Finally, we will propose how to apply cryptographic hash functions, herding attack technique, with examples, to Latin square based secret sharing schemes to overcome these limitations. We conclude the paper with the future research. Note, in the rest of the paper, unless specified, scheme refers to secret sharing scheme.

2 Cryptographic hash functions

A cryptographic hash function [29,30] takes an input string of arbitrary length and generates an output string of fixed length, which is called message digest, or hash value, or just hash. Hash functions have many applications in information security area, such as digital signatures, message authentication codes, and authentication protocols.

2.1 Properties of hash function

The following are common properties that a well designed cryptographic hash function should have.

- 1) For all practical purposes, given any message x , the message digest $h(x)$ can be calculated very quickly.

- 2) Given a message digest y , it is computationally infeasible to find x such that $h(x) = y$. This is called preimage resistant. 1) and 2) imply that h is a one way function.
- 3) Given an input and output pair (x, y) for a hash function, it should remain infeasible to find a second preimage x' such that $x \neq x'$, but $h(x) = h(x') = y$. This property is called second preimage resistance.
- 4) It is infeasible to find two different inputs, x and x' , that produce the same output, i.e. $x \neq x'$, but $h(x) = h(x')$. This property is called collision resistance.

A hash function must have the flexibility to process messages of arbitrary length. Most currently used hash functions, such as MD family and SHA family, are built from iterations of a compression function C using Merkle–Damgård construction [10, 22], they are also called *iterative hash functions*. The process is as follows.

- (a) Pad the arbitrary length message M into multiple v -bit blocks: $m_1, \dots, m_b$.
- (b) Iterate the compression function $h_i = C(h_{i-1}, m_i)$, where i is from 1 to b and h_0 is the initial value (or initial vector) IV.
- (c) Output h_b is the hash of the message M , i.e., $H(M) = h_b = C(h_{b-1}, m_b)$.

2.2 Herding and Nostradamus attack

Iterative hash functions are vulnerable to herding and Nostradamus attack. This attack makes use of the fact that it is not difficult to find intermediate hash values that can be substituted for genuine blocks during iterative application of a compression function and generate the same final hash value, h . Kelsey and Kohno [19] gave a detailed analysis on this attack. Stevens, Lenstra and Weger [28] applied the technique to predict the winner of the 2008 US Presidential Elections using a Sony PlayStation 3 in November 2007. The hash of the result matched with that they committed to the public before the election. So, they claimed that they have correctly predicted the next US president.

Let H be an iterative hash function and C be its underlying compression function. The first step is to build a large set of intermediate hashes at the first level: $h_{11}, h_{12}, \dots, h_{1w}$. The second step is to build a set of intermediate hashes at the

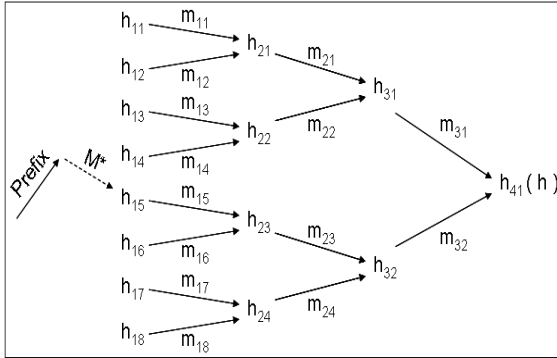


Figure 1. A diamond structure to illustrate Nostradamus attack.

second level: $h_{21}, h_{22}, \dots, h_{2w/2}$ so that the following conditions are satisfied:

finds a message m_{11} such that $C(h_{11}, m_{11}) = h_{21}$
 finds a message m_{12} such that $C(h_{12}, m_{12}) = h_{21}$
 finds a message m_{13} such that $C(h_{13}, m_{13}) = h_{22}$
 finds a message m_{14} such that $C(h_{14}, m_{14}) = h_{22}$
 $\vdots$

By repeating this process, these message blocks are linked so that each intermediate hash at level 1 can reach the final hash, say h . This is called the diamond structure (see Figure 1).

We claim we can predict something happens in the future by announcing the hash h to the public. When the result is available, we construct a message as

$$M = \text{Prefix} \parallel M^* \parallel \text{Suffix},$$

where Prefix contains the real results that we claimed that we knew before it happens. M^* is a block of message which can link the Prefix to one of the intermediate hash at level 1. Suffix is the rest of message blocks which linked the M^* to the final hash. In Figure 1, $M = \text{Prefix} \parallel M^* \parallel m_{15} \parallel m_{23} \parallel m_{32}$ with $\text{Suffix} = m_{15} \parallel m_{23} \parallel m_{32}$, and $H(M) = h_{41} = h$.

3 Secret sharing schemes

A secret sharing scheme [29, 30] is a method to split and distribute a secret among a group of participants, each of whom receives a share of the secret. The secret can only be recovered when the participants join together to combine their shares.

There are many practical applications of the secret sharing schemes. For example, they can be used to protect a private key from being accessed by outsiders. When we examine the problem of maintaining sensitive information, we will consider two issues: *availability* and *secrecy*. If only one person keeps the entire secret, then there is a risk that the person might lose it or the person may not be available when it is needed. We can solve the availability and reliability issues by letting more than one person keep the same secret. But the more people who can access the secret, the higher the chance the secret will be leaked. A secret sharing scheme is designed to address these issues.

3.1 Sharmir's scheme

In 1979 Shamir [24] proposed the $(t + 1, n)$ threshold scheme, in which a secret is split into pieces (shares) and distributed among n participants $p_1, p_2, \dots, p_n$ where any group of $t + 1$ or more participants ($t \leq n - 1$) can recover the secret. Meanwhile, any group of t or fewer participants cannot recover the secret. By sharing a secret in this way the availability and reliability issues can be solved. Share distributing and secret recovering [2, 29, 30] will be discussed as follows.

Share distributing: The dealer generates a polynomial of degree t over $\mathbb{Z}_q$, where q is a prime number and $q > n$. The coefficients $a_0, a_1, \dots, a_t$ are chosen arbitrarily. $P(x) = a_0 + a_1x + \dots + a_{t-1}x^{t-1} + a_tx^t$ where $a_t \neq 0, a_i \in \mathbb{Z}_q, 0 \leq i \leq t$, and a_0 is the secret.

The dealer calculates $y_i = P(x_i), 1 \leq i \leq n, x_i \neq 0$. $x_1, x_2, \dots, x_n$ are public information. Values $y_1, y_2, \dots, y_n$ are distributed to the n participants so that each participant gets one share, i.e., $y_1 \rightarrow p_1, y_2 \rightarrow p_2, \dots$, etc.

Secret recovering (i): When any $(t + 1)$ participants join together, we have the following system of $(t + 1)$ equations. For simplicity, we assume $p_1, p_2, \dots, p_{t+1}$ join together.

$$\begin{aligned} y_1 &= P(x_1) = a_0 + a_1x_1 + \dots + a_tx_1^t \pmod{q}, \\ y_2 &= P(x_2) = a_0 + a_1x_2 + \dots + a_tx_2^t \pmod{q}, \\ &\vdots \\ y_{t+1} &= P(x_{t+1}) = a_0 + a_1x_{t+1} + \dots + a_tx_{t+1}^t \pmod{q}. \end{aligned}$$

In matrix representation, it will be:

$$\begin{bmatrix} 1 & x_1 & \cdots & x_1^t \\ 1 & x_2 & \cdots & x_2^t \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_{t+1} & \cdots & x_{t+1}^t \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{t+1} \end{bmatrix} \pmod{q}.$$

Let M be the above $(t + 1) \times (t + 1)$ matrix, which is a Vandermonde matrix. Its determinant is

$$\det(M) = \prod_{i \leq j < k \leq t+1} (x_k - x_j) \pmod{q}.$$

Since we choose different points for the participants, i.e., different x_i 's, we have $\det(M) \neq 0$, and this guarantees a unique solution. We can solve the system of equations by Gaussian elimination or Cramer's rule. Hence the secret can be recovered.

Secret recovering (ii): Another method is to use Lagrange interpolation. We can construct the polynomial of degree t by any $(t + 1)$ different points: $(x_1, y_1), \dots, (x_{t+1}, y_{t+1})$

$$P(x) = \sum_{i=1}^{t+1} y_i l_i(x), \quad \text{where} \quad l_i(x) = \prod_{j=1, j \neq i}^{t+1} \frac{x - x_j}{x_i - x_j} \pmod{q},$$

$$P(x) = \frac{(x - x_1) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_{t+1})}{(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_{t+1})} \pmod{q}.$$

So, the secret a_0 will be

$$a_0 = P(0) = \sum_{i=1}^{t+1} y_i \prod_{j=1, j \neq i}^{t+1} \frac{-x_j}{x_i - x_j} \pmod{q}.$$

3.2 Entropy

In information theory, developed by Shannon [25, 26], entropy is a measure of information or uncertainty. Also see [3, 29, 30] for more details on entropy. Let X be a random variable with probability distribution $p(x)$, where $p(x) \geq 0$, $\sum_{x \in X} p(x) = 1$. Then the entropy of X is defined as

$$E(X) = - \sum_{x \in X} p(x) \log_2 p(x).$$

We assume $p(x) \log_2 p(x) = 0$, if $p(x) = 0$. This is justified because

$$\lim_{p(x) \rightarrow 0} p(x) \log_2 p(x) = 0.$$

Example. Let X be a random variable of the event of an unbiased fair coin flipping with the possible outcomes of $\{\text{Head}, \text{Tail}\}$, with the equal probability $p(\text{Head}) = p(\text{Tail}) = 1/2$, then:

$$E(X) = -p(\text{Head}) \log_2 p(\text{Head}) - p(\text{Tail}) \log_2 p(\text{Tail}) = \frac{1}{2} + \frac{1}{2} = 1.$$

If the coin is biased with $p(\text{Head}) = 1$ and $p(\text{Tail}) = 0$, then

$$E(X) = -p(\text{Head}) \log_2 p(\text{Head}) - p(\text{Tail}) \log_2 p(\text{Tail}) = 0.$$

In this case there is no uncertainty. We can use that $E(X) = 0$ to infer that $\exists x_i \in X$ such that $p(x_i) = 1$ and $p(x_j) = 0$ for $j \neq i$.

Let X and Y be two random variables, $x \in X$, $y \in Y$, the joint entropy $H(X, Y)$ is defined as

$$H(X, Y) = - \sum_x \sum_y p(x, y) \log_2 p(x, y).$$

The conditional entropy $H(X|Y)$ is defined as

$$\begin{aligned} H(X|Y) &= \sum_{y \in Y} p(y) H(X|Y = y) \\ &= - \sum_{y \in Y} p(y) \left(\sum_{x \in X} p(x|y) \log_2 p(x|y) \right) \\ &= - \sum_{y \in Y} \sum_{x \in X} p(y) p(x|y) \log_2 p(x|y). \end{aligned}$$

However, if X and Y are independent, then

$$\begin{aligned} H(X|Y) &= - \sum_{y \in Y} p(y) \left(\sum_{x \in X} p(x|y) \log_2 p(x|y) \right) \\ &= \sum_{y \in Y} p(y) \left(- \sum_{x \in X} p(x) \log_2 p(x) \right) \\ &= 1 \cdot H(X) = H(X). \end{aligned}$$

3.3 Access structure

It is reasonable to assume that any $t + 1$ or more participants can always recover the secret. We call this property *monotone*. A group of participants, which can

recover the secret when they join together, is called an *authorized subset*. In the above example, any group of $t + 1$ or more participants forms an authorized subset, since we assume it has the monotone property. On the other hand, any group of participants that cannot recover the secret is called an *unauthorized subset*. An *access structure* Γ is a set of all authorized subsets.

Given any access structure Γ , $A \in \Gamma$ is called a minimal authorized subset if $B \subset A$ then $B \notin \Gamma$.

We use Γ_0 to denote the set of the minimal authorized subsets of Γ . In a $(t + 1, n)$ threshold scheme, let P be the set of the participants:

$$\begin{aligned}\Gamma &= \{A | A \subseteq P \text{ \& } |A| \geq (t + 1)\} \\ \Gamma_0 &= \{A | A \subseteq P \text{ \& } |A| = (t + 1)\}.\end{aligned}$$

In secret sharing, we first define the access structure. Then we realize the access structure by a secret sharing scheme. Ito, Saito, and Nishizeki [17] proved that any general access structure can be realized by a secret sharing scheme.

3.4 Perfect and ideal scheme

Shamir's scheme allows no partial information to be given out even up to t participants joined together [29]. In other words, any group of up to t participants cannot get more information about the secret than any outsider. A secret sharing scheme with this property is called a *perfect scheme*. If any partial information about the secret is given out, it would be easier for an unauthorized subset to discover the secret. That is why we prefer to have a perfect scheme.

In terms of entropy in information theory, we have

$$\begin{aligned}H(S|A) &= 0, \quad \text{if } A \in \Gamma \quad (\text{correctness}); \\ H(S|A) &= H(S), \quad \text{if } A \notin \Gamma \quad (\text{privacy}),\end{aligned}$$

where S is the secret, $H(X)$ denotes the entropy of random variable X . The first equation says that for an authorized subset A there is no uncertainty and the secret S is determined/recovered. The second equation says that for an unauthorized subset A the uncertainty remains unchanged and no information about the secret S is leaked out even the participants pool all their shares together.

Based on the information theory, the length of any share must be at least as long as the secret itself in order to have perfect secrecy. The argument is that up to t participants have zero information about the secret under perfect sharing scheme, but when one extra participant joins the group, the secret can be recovered. That means any participant has his share at least as long as the secret. If the shares and the secret come from the same domain, we call it an *ideal scheme*. In this case, the shares and the secret have the same size.

3.5 Proactive scheme

In a secret sharing scheme, we need to consider the possibility that a smart adversary may find out all the shares in an authorized set to discover the secret eventually if he is given a very long time to gather the necessary information. This means that if the adversary can successfully break in $(t + 1)$ servers, in a $(t + 1, n)$ threshold scheme he can steal the secret. In order to prevent this from happening, we may try to reset the shares. We re-fresh and re-distribute all the shares to all the participants periodically. After finishing this phase, the old shares are erased safely and the secret remains unchanged. By doing so, the information gathered by the adversary between two resets would be useless, as the old shares are obsolete and erased completely. In order to break the system an adversary has to get enough information of the shares within any two periodic resets. This would make it more difficult to achieve.

Based on Shamir's scheme, Herzberg, Jarecki, Krawczyk, and Yung [16] derived a proactive scheme, which uses the following method to reset the shares.

Let $P(x)$ is an arbitrary polynomial of degree t over $\mathbb{Z}_q$, same as in the Shamir's scheme,

$$P(x) = a_0 + a_1x + \cdots + a_{t-1}x^{t-1} + a_tx^t,$$

where q is a prime number, $a_t \neq 0$ and $a_0, a_1, \dots, a_t \in \mathbb{Z}_q$. The dealer generates another polynomial $Q(x)$ of degree t over $\mathbb{Z}_q$ without a constant term,

$$Q(x) = b_1x + \cdots + b_{t-1}x^{t-1} + b_tx^t,$$

where $b_t \neq 0$ and $b_1, \dots, b_t \in \mathbb{Z}_q$. Then the dealer adds $P(x)$ and $Q(x)$ together to get $S(x)$ as

$$S(x) = a_0 + c_1x + \cdots + c_{t-1}x^{t-1} + c_tx^t,$$

where $c_t \neq 0$ and $c_i = a_i + b_i \bmod q$ for $i = 1, \dots, t$. The dealer then sends out new shares $S(1), S(2), \dots, S(n)$ to the n participants to replace the old shares $P(0), P(1), \dots, P(n)$. The scheme remains a $(t + 1, n)$ threshold scheme with the same original secret.

The above technique can be extended so that each participant i , by turn, generates a polynomial $P_i(x)$ of degree t without a constant term and sends to all other participants the corresponding values of $P_i(1), \dots, P_i(i-1), P_i(i+1), \dots, P_i(n)$. After the above exchange process, each participant i resets his new shares as follows:

$$\text{newshare} = \text{oldshare} + P_1(i) + \cdots + P_n(i).$$

After the calculation of the new shares, all participants will destroy their old shares safely. In other words, all the participants can engage in the share renewing process. This method can eliminate all the work done by the dealer and be more secure.

3.6 Verifiable scheme

Shamir's original sharing scheme assumes the dealer and all the participants are honest. However, in reality, we need to consider the situation that the dealer or some of the participants might be malicious. In this case, we need to set up a verifiable scheme so that the shares of the participants can be verified to be valid. In order to make this possible, additional information is required for the participants to verify their shares' consistency.

Feldman [14] presented a simple verifiable scheme that is based on Shamir's scheme. It is based on the homomorphic properties of the exponentiation function: $x^{a+b} = x^a \cdot x^b$.

The idea is to find a cyclic group G of order p , where p is a prime. Since it is cyclic a generator of G , say g , exists. As other cryptographic protocols, we assume the parameters of G are carefully chosen so that the discrete logarithm problem is hard to solve in G . The dealer then generates a polynomial over $\mathbb{Z}_q$ of degree t as

$$P(x) = a_0 + a_1x + \cdots + a_{t-1}x^{t-1} + a_tx^t$$

where $a_t \neq 0$ and $a_0, a_1, \dots, a_t \in \mathbb{Z}_q$.

The dealer sends out $P(i)$ to participant i as before. In addition, he puts the commitments, $g^{a_0}, g^{a_1}, \dots, g^{a_t}$, in a public place for the participants to verify.

Each participant i can verify the validity of the dealer if the following equation is true.

$$P(i) = (g^{a_0})(g^{a_1})^i(g^{a_2})^{i^2} \cdots (g^{a_t})^{i^t}, \quad i = 1, \dots, n.$$

Based on the homomorphic properties of the exponentiation, the above condition will hold true if the dealer sends out consistent information. If this is the case, we conclude that the dealer is honest, and the scheme is verifiable. Later, when the participants return their shares for secret recovering, the dealer can verify their shares' validity by the same method.

Feldman's scheme is a verifiable scheme, however it is not a perfect scheme since partial information, g^{a_0} , is leaked out. We assume it is difficult to get the secret a_0 from g^{a_0} if the discrete logarithm problem is hard to solve under G .

Many existing schemes are subject to certain limitations. One particular scheme is only applicable to one specific access structure. If we want to apply one scheme to another access structure, either it doesn't work or it is inefficient. Although Ito,

Saito, and Nishizeki [17] proved that any general access structure can be realized by a secret sharing scheme, but there is no guarantee that the scheme is efficient. Also, a scheme may not have all the desired properties, such as perfect, ideal, verifiable, and proactive.

4 Latin square

A Latin square of order n is an array that consists of n rows and n columns such that for any row and any column only one out of the n symbols occurs exactly once. For simplicity, we usually use $0, \dots, n-1$ to represent the symbols so that each entry in a Latin square can be represented as a triple (i, j, k) , where $0 \leq i, j, k \leq n-1$, and i, j, k are the row, the column and the symbol, respectively. For any order n , there exists a Latin square of this order. The addition table of the additive group $\mathbb{Z}/n\mathbb{Z}$ of integers mod n is an example [23].

4.1 Use a Latin square as a secret

Suppose we use a Latin square to represent the secret and its order, n , is made public. For an empty $n \times n$ array, there are $n!$ ways to fill out the first row. Now consider the second row. There are $n-1$ choices for filling the '0'. There are $n-1$ or $n-2$ choices for filling the '1' depending on whether the '0' was filled under the '1' in the first row or not. So there are at least $n-2$ choices for filling the '1'. We continue with '2', there are at least $n-3$ choices. So, there are at least $(n-1)!$ ways to fill out the second row. By similar argument, we can see there are at least $n!(n-1)!(n-2)! \cdots 2!$ Latin squares of order n . This is just a lower bound. For a reasonably large n , say $n \geq 10$, there are many different Latin squares of this order. This definitely makes an outsider very difficult to figure out the secret itself without having any related knowledge.

If the order n increases by 1, the number of Latin squares will grow exponentially. For instance the number of Latin squares of order 10 and 11 are as follows [21, 23].

$$L_{10} = 10! \times 9! \times 7,580,721,483,160,132,811,489,280;$$

$$L_{11} = 11! \times 10! \times 5,363,937,773,277,371,298,119,673,540,771,840.$$

To our best knowledge, there is no effective method to determine the number of Latin squares of a given larger order, for instance $n \geq 10$. The number of Latin square of a given order is an open problem. By now, the number of Latin squares of order 12 has not been determined.

4.2 Partial Latin square and extension of a partial Latin square

A partial Latin square of order n is an array that consists of n rows and n columns such that for any row and any column no symbol occurs more than once and one or more cells can be empty. That is, there exists one or more pair (i, j) such that there is no symbol in row i and column j .

Some partial Latin squares can be extended to Latin squares of the same order, while others cannot be extended. In the following example (see Table 1), the partial Latin square on the left can be extended into a Latin square in the middle. But the partial Latin square on the right cannot be extended to a full Latin square.

0		3	
	2		
		1	
			3

0	1	3	2
3	2	0	1
2	3	1	0
1	0	2	3

0		3	1
	2		

Table 1. Partial Latin square extendibility.

In 1960 Evans [13] conjectured that any partial Latin square of order n can always be extended to a full Latin square if the size of the partial Latin square (i.e., the number of non-empty elements) is up to $n - 1$. Twenty years later, this was proved to be true by Smetaniuk [27]. $n - 1$ is the optimal number as we can see from the right table in Table 1.

We define a partial Latin square as a *Latin rectangle* [31] if the first m rows are all filled ($m < n$) and the remaining $n - m$ rows are all empty. A Latin rectangle can always be extended to a full Latin square by adding row by row. This can be proved by Hall’s condition in prefect matching [15]. Given a partial Latin square, there may be different ways to extend it to different Latin squares of the same order.

Following [31], a *cutoff* of a Latin square L is defined as the left side of Table 2. Given a Latin square L of order n with the symbols $\{0, 1, \dots, n - 1\}$, we construct a partial Latin square P of order $n + 1$, see the right side of Table 2.

- 1) Entries in the back diagonal are filled with the new symbol, say n .
- 2) The upper entries of the back diagonal are copied from the original Latin square L (i.e., the cutoff of L).

The partial Latin square P can be extended to a full Latin square of order $n + 1$ [31].

×	×	×	×	×
×	×	×	×	
×	×	×		
×	×			
×				

 L

×	×	×	×	×	n
×	×	×	×	n	
×	×	×	n		
×	×	n			
×	n				
n					

 P

Table 2. From cutoff to partial Latin square.

4.3 Critical set and strong critical set

A critical set of a Latin square is a partial Latin square which can be extended to a full Latin square uniquely. Also, after deletion of any entry of a critical set, the unique completion property does not hold any more. For a given Latin square, there may exist critical sets of different sizes.

By definition, we know we can recover the original Latin square from one of its critical sets and the completion is unique. However, whether we can complete a Latin square from a partial Latin square is an NP-complete problem [8]. That means the recovery of the Latin square from one of its critical set may be time-consuming. We really need some criteria to speed up the process.

Donovan, Cooper, Nott and Seberry [12] defined a strong critical set. Let L be a Latin square of order n and C one of its critical set. Let $|C|$ be the size of C , i.e., the number of non empty cells in C . If there is a sequence of partial Latin squares $\{P_0, P_1, \dots, P_m\}$ such that

- 1) $C = P_0 \subset P_1 \subset \dots \subset P_m = L$, where $m = n^2 - |C|$;
- 2) for any $i, 0 \leq i \leq m-1$, $P_i \cup \{(r_i, c_i, k_i)\} = P_{i+1}$ and $P_i \cup \{(r_i, c_i, k)\}$ is not a partial Latin square if $k \neq k_i$.

That means we start from the critical set C and enter an entry to an empty cell one at a time until we finish the extension to a full Latin square L . When we get a new partial Latin square P_{i+1} , $0 \leq i \leq m-1$ each time, there always exists a cell (r_i, c_i) that can be filled with only one symbol k_i . We call this critical set a strong critical set if it has the above properties. In other words, the “forcing out” process makes a strong critical set to be extended to a full Latin square more easily.

5 Application of critical set in secret sharing

Cooper, Donovan, Seberry [9] proposed to form a collection of critical sets of a Latin square, say S . Elements of S are distributed to participants. Any group of participants is an authorized subset if their shares pooled together is one of the critical sets forming S .

(1) For example: A $(2, 3)$ threshold scheme is shown in Table 3.

0						0	1	2
	2						2	0
					1		0	1
C_1			C_2			C_3		
						L		

Table 3. A $(2, 3)$ threshold secret sharing scheme.

We can easily verify that all the partial Latin squares C_1, C_2, C_3 are critical sets. They can be extended uniquely to the full Latin square L . This unique completion property does not hold true any more if any entry of any partial Latin square C_1, C_2, C_3 is deleted.

Let S be the union of the three critical sets C_1, C_2, C_3 . Then $S = \{(0, 0, 0), (1, 1, 2), (2, 2, 1)\}$. We distribute a triple to a participant as a share. Any two participants can recover the full Latin square. So we have a $(2, 3)$ threshold scheme.

(2) The above simple example can be extended to the following general case. Let $C_1, C_2, C_3, \dots, C_n$ be the critical sets of a given Latin square of size $s_1, s_2, \dots, s_n$. Each C_i consists of a set of triples as follows:

$$\begin{aligned}
 C_1 &= \{(x_{11}, y_{11}, k_{11}), \dots, (x_{1s_1}, y_{1s_1}, k_{1s_1})\} \\
 C_2 &= \{(x_{21}, y_{21}, k_{21}), \dots, (x_{2s_2}, y_{2s_2}, k_{2s_2})\} \\
 &\vdots \\
 C_n &= \{(x_{n1}, y_{n1}, k_{n1}), \dots, (x_{ns_n}, y_{ns_n}, k_{ns_n})\}.
 \end{aligned}$$

A triple (x_{ij}, y_{ij}, k_{ij}) is interpreted as follow: x_{ij} is the row of the j th element in C_i , y_{ij} is the column of the j th element in C_i , and k_{ij} is the symbol of the j th element in C_i .

In general, we make S as a union of some critical sets of a given Latin square L which represents a secret. Then, the dealer distributes a share in S , in this case a triple of the Latin square, to each participant. Whenever, a group of participants joins together to form a critical set, the original Latin square, and hence the secret can be recovered.

Chaudhry, Ghodosi, and Seberry [4] proposed a perfect scheme based on Room squares. This can be applied to Latin square. The idea is to generate shares randomly for all the participants in an authorized subset with the exception of the last participant, whose shares will be determined by the shares of the other participants in such a way that all the shares when summing up will be equal to the value of the critical set, which is the secret. Modular arithmetic is applied here. This process is repeated for all the authorized subsets.

Example. Let $C = \{(0, 0, 0), (1, 1, 1)\}$ be the critical set of the Latin square L as in Table 4.

$$L = \{(0, 0, 0), (0, 1, 2), (0, 2, 1); (1, 0, 2), (1, 1, 1), (1, 2, 0); (2, 0, 1), (2, 1, 0), (2, 2, 2)\}.$$

0			0	2	1
	1		2	1	0
			1	0	2
C			L		

Table 4. Calculation of the share for the last participant.

Let $\{P_1, P_2, P_3\}$ be an authorized subset over C . Suppose we generate the following random shares S_1, S_2 for P_1 and P_2 as: $S_1 = \{(0, 1, 2), (2, 0, 0)\}$ and $S_2 = \{(1, 2, 1), (0, 2, 1)\}$. Then share S_3 for P_3 will be calculated as

$$\begin{aligned} S_3 &= \{(0 - (0 + 1), 0 - (1 + 2), 0 - (2 + 1)), \\ &\quad (1 - (2 + 0), 1 - (0 + 2), 1 - (0 + 1))\} \\ &= \{(2, 0, 0), (2, 2, 0)\}. \end{aligned}$$

All arithmetic is done in mod 3. It can be easily verified that P_1, P_2, P_3 can recover the critical set when they pool their shares together. If any participant is missing, it makes the unauthorized subset contain nothing more than any outsider.

We summarize the reasons why we want to apply critical sets to the secret sharing scheme:

- 1) Since a critical set can always be extended to a full Latin square uniquely, it would be more efficient to distribute shares of a critical set rather than a full Latin square.
- 2) A $(t + 1, n)$ threshold scheme or multilevel scheme can be implemented through critical sets, as discussed in Cooper, Dononvan, and Seberry [9].

6 Limitations of Latin square based schemes

More research has been done since the original secret sharing ideas of Shamir [24] and Blakley [1] in 1979. Latin square was suggested as a good candidate for secret sharing schemes. However, there are certain limitations as follows.

- 1) By just distributing shares of a critical set to participants, partial information will be available to any unauthorized subset. That means there is a good chance for any unauthorized subset to figure out the remaining shares by trial and error method. So, the schemes proposed by Cooper, Donovan, Seberry [9] and Chaudhry and Seberry [5] are not perfect.
- 2) The scheme proposed by Chaudhry, Ghodosi, Seberry [4] is not flexible if there is only one authorized set. In this case it is just a secret splitting scheme. If more than one authorized set exists, the secret sharing scheme is not ideal. Each participant needs to have different shares for different authorized subsets he/she belongs to.
- 3) As we know, distributing shares of a critical set instead of a Latin square is definitely more efficient. However, there are two issues need to be considered:
 - a) Even getting all the shares about a critical set, it may not be easy to get back the original Latin square, the shared secret. In order to speed up the recovery process, we should use a strong critical set.
 - b) However, if the participants of an authorized subset join together, it will be much easier for them to figure out the shared secret if the chosen critical set is a strong one.
- 4) Given a Latin square of large order (say ≥ 10), there are many critical sets of different sizes. It is very difficult to verify or find such critical sets [7].
 - a) Control: Let S be a collection of critical sets C_1, C_2, C_3 of Latin square L . We would like to design a secret sharing scheme such that any authorized set of participants can recover C_1 or C_2 or C_3 . But there is a possibility that S contains another critical set C_4 . If individuals of any unauthorized set (in the sense that they cannot recover C_1, C_2 or C_3) can pool their shares to form C_4 , then they can recover L . Hence some careful controls are needed, especially given the condition that critical set of large order Latin square is difficult to find or verify.

Example. As shown in the example in Section 3 (see Table 3), C_1, C_2, C_3 are critical sets of L . Suppose the dealer does not notice that C_3 is the critical set. He assigns $(0, 0, 0)$ to A, $(1, 1, 2)$ to B, and $(2, 2, 1)$ to C. So A and B can come up with C_1 to recover L ; B and C can come up with C_2 to recover L . So two

participants need to recover the secret and B is more important in the sense he or she must be present. However, A and C (an unauthorized subset in the dealer's mind) can come up with C_3 to recover the secret.

- (b) Implementation: The knowledge about the critical sets of Latin squares of a large order is very limited. These hinder the implementation of various schemes based on critical sets.

7 Apply hash function to Latin square based schemes

Zheng, Hardjono, and Seberry [33] discuss how to reuse shares in a secret sharing scheme by using universal hash function. In this section, we'll show how to use general hash function properties including herding, and Nostradamus attacks [19] to design and improve Latin square based schemes.

7.1 Motivation

- 1) If the following attacks are practical, so does this approach based on the reasonable assumption that there are a small number of authorized subsets, say ≤ 8 :
 - (a) multi-collisions [18],
 - (b) expandable message with fixed points [11],
 - (c) expandable message with multi-collisions [20],
 - (d) herding and Nostradamus attacks [19].
- 2) Stevens, Lenstra, and Weger [28] had applied herding technique to predict the winner of the 2008 US presidential election.
- 3) Advancement of technology: HPC (High Performance Computing) and GPU (Graphics Processing Unit).
- 4) Highly suitable for parallel computation.
- 5) Other advantages based on hash function properties.

7.2 Store Latin square in a hash

If we want to use the hash to store a Latin square of order 10, we need to store 81 numbers (since the last row and last column are not necessary). If we use 4 bits to store a number, and 10 bits to store 3 numbers, we can choose SHA-384 or SHA-512 to fulfill the requirements easily. However, it is impractical to use SHA-384 or SHA-512 in the proposed scheme.

We proceed in the following way (see Table 5). This shows an improvement over [6].

All the A 's, B 's, C 's, and D in a row or column are different because of Latin property. A 's, B 's, C 's, and D have different encoding/decoding methods. There is a one-to-one correspondence between the remaining decimals and number of bits in each stage.

A	A	B	B	B	B	C	C	D	
A	A	B	B	B	B	C	C	D	
B	B	.	.	.	.	.	.	B	
B	B	.	.	.	.	.	.	B	
B	B	.	.	.	.	.	.	B	
B	B	.	.	.	.	.	.	B	
C	C	.	.	.	.	.	.	C	
C	C	.	.	.	.	.	.	C	
D	D	.	.	.	.	.	.	D	

Table 5. Store a Latin square of order 10 in fewer bits.

The 0th row:

- (a) Use 7 bits to represent the first two A 's.
- (b) For the next four B 's, use 3 bits each.

For example, if the first two digits (A 's) are 7 and 3, then the remaining decimals will be decoded as

$$\begin{aligned} 9 &\leftrightarrow 111; & 8 &\leftrightarrow 110; & 6 &\leftrightarrow 101; & 5 &\leftrightarrow 100; \\ 4 &\leftrightarrow 011; & 2 &\leftrightarrow 010; & 1 &\leftrightarrow 001; & 0 &\leftrightarrow 000. \end{aligned}$$

- (c) For the next two C 's, use 2 bit each.

For example if four B 's are 4, 6, 1, and 2, then the remaining numbers will be encoded as

$$9 \leftrightarrow 11; \quad 8 \leftrightarrow 10; \quad 5 \leftrightarrow 01; \quad 0 \leftrightarrow 00.$$

(d) For the next D , use 1 bit.

For example, if two C 's are 8 and 5, then the remaining numbers will be encoded as

$$9 \leftrightarrow 1; \quad 0 \leftrightarrow 0.$$

We need $7 + 4 \times 3 + 2 \times 2 + 1 = 24$ bits to encode the 0th row. Decoding can be done easily in the same way.

The 1st row: Same as the 0th row.

We start to encode the rest of the column 0. It takes $4 \times 3 + 2 \times 2 + 1 = 17$ bits. Repeat the same process for columns 1 to 8. In total we need $24 + 24 + 17 \times 9 = 201$ bits for the whole Latin square.

For a Latin square of order 9, we use Table 6 for the encoding as follows:

- (a) 4 bits for A ,
- (b) 3 bits for B ,
- (c) 2 bits for C ,
- (d) 1 bit for D .

A	B	B	B	B	C	C	D	
B	B	.	.	.	.	.	B	
B	B	.	.	.	.	.	B	
B	B	.	.	.	.	.	B	
B	B	.	.	.	.	.	B	
C	C	.	.	.	.	.	C	
C	C	.	.	.	.	.	C	
D	D	.	.	.	.	.	D	

Table 6. Store a Latin square of order 9 in fewer bits.

So 21 bits for the first row, 17 bits for each column from 2nd row to 8th row. The total number of bits to encode a Latin square of order 9 is $21 + 17 \times 8 = 157$. Therefore we can store the Latin square into a SHA-160 (SHA-1 and SHA-0), and still leave 3 bits unused.

Unused bits can be filled in randomly. The above are simple examples to demonstrate how to use hash to represent a fixed secret. From now on, when we talk about a partial Latin square, we mean the one that can be easily extended uniquely back to the original Latin square, or vice versa. In other words, throughout the rest of the paper, these two terms will be used interchangeably.

7.3 A modified diamond structure

In the proposed new scheme, we set up one message M_{priv} for one authorized subset. Since the hashes of the M_{priv} messages are known, we don't need to find a linking block to link the Prefix to one of the starting hashes. In reality we only expect to have a small number of authorized subsets, so we do not need to set up a huge diamond structure. This greatly reduces the effort (see Figure 2). However, h may not be a Latin square. So we need to generate a long list of Latin squares: $L_1, L_2, \dots$, and then find a linking block M^* to link h to one of these Latin squares. This is similar to the traditional diamond structure but it is set up at the end, instead of the beginning. Building the list of $L_1, L_2, \dots$ Latin squares is a one-time work and can be done in parallel.

According to Kelsey and Kohno [19], the work required to build a diamond structure consisting of 2^k intermediate hashes in the first level is $2^{(n/2+k/2+2)}$ where n is the number of bits in the hash of the hash function that is being used to build the structure. The work needs to search a linking block to link the prefix to one of these 2^k intermediate hashes is $2^{(n-k)}$.

So, the work for setting up the diamond structure in our scheme is $2^{(n/2+k/2+2)} + 2^{(n/2)}$, assuming we built $2^{(n/2)}$ Latin squares.

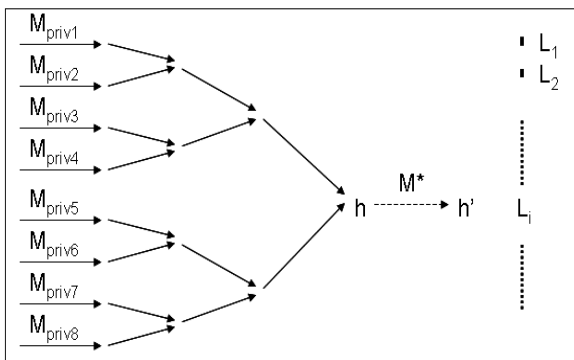


Figure 2. A modified diamond structure. $C(h, M^*) = h' = \text{one of the Latin squares } L_1, L_2, \dots$

Since we don't expect there is a large number of authorized sets, the work required should be $2^{(n/2+c)}$, where c is just a small positive integer.

7.4 Set up an ideal perfect $(t + 1, n)$ threshold scheme

Let's continue with Section 7.2 and suppose the secret is a Latin square stored in a hash. Let's consider how to apply a hash function f to set up a $(t + 1, n)$ threshold secret sharing scheme. The approach we take is based on herding hash technique.

First we randomly generate a share of the hash size for each participant. Then, we set up different authorized subsets so that each subset consists of $(t + 1)$ or more distinct participants.

Let T be the size of the access structure, i.e., the total number of all authorized subsets.

$$T = C(n, t + 1) + C(n, t + 2) + \cdots + C(n, n),$$

where $C(n, t) = (n!)/(t!(n - t)!)$ is the combination function. If any $(t + 1)$ participants are an authorized subset, so does any set consisting of more than $(t + 1)$ participants. This is the so called monotone property. So, we need to consider $C(n, t + 1)$ authorized sets only. Let $N = C(n, t + 1)$. That means we need to have N messages for these N authorized subsets. There is a one-to-one correspondence between messages and authorized subsets.

Each participant holds a share and the combination of the shares of any one of these N authorized subsets will generate one of these N messages. The next step is to herd the hashes of these N messages into the final hash as the Nostradamus attack by setting up the linking messages.

Suppose an authorized subset consists of participants $P_1, P_2, \dots, P_b$ and their shares are sub-messages $m_1, m_2, \dots, m_b$. When they join together, they can form $M_{\text{priv}} = m_1 || \dots || m_b$ and find the corresponding linking message M_{pub} , as shown in Figure 3. Then they can recover the secret h by applying the hash function f to $M_{\text{priv}} || M_{\text{pub}}$, i.e., $f(M_{\text{priv}} || M_{\text{pub}}) = h$.

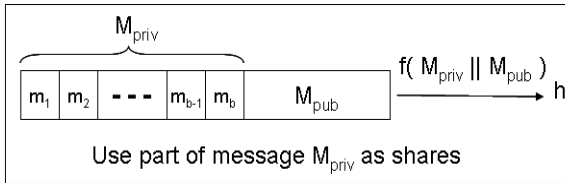


Figure 3. Message M and sub-messages, i.e., shares m_i .

For any message M_{priv} obtained by combining the shares of the participants in an authorized subset, there is a corresponding message M_{pub} in the diamond structure. Linking these two messages can reach the final hash of the diamond structure. So, we have a $(t + 1, n)$ threshold scheme based on herding hash functions technique. The linking messages are stored in a public place which can be accessed by any participant. When any subset of $(t + 1)$ or more participants join together, they can look for the corresponding linking message and plus their shares to recover the secret.

Properties of the proposed scheme include:

- a) Perfect: One of the basic properties of a cryptographic hash function is its randomness. Based on the message, we cannot figure out any information about the hash. This avoids revealing partial information to any participant. When all participants join together, they can recover the secret by applying the hash function f to the message $M = M_{\text{priv}} || M_{\text{pub}}$. In order to maintain the security level, the length of each share should be at least as long as the hash. On the other hand, increasing the length of the share does not increase the security level. So, we would like to have each share generated randomly and of the same length as the hash. Suppose a participant in a minimal authorized subset is missing, the rest of the participants of the subset can guess his/her share and then calculate the hash. If the hash is not a Latin square, they can rule out the possibility. However, they don't gain any additional information comparing to an outsider who just guesses the Latin square directly. So, we still consider that it is perfect in this sense.
- b) Ideal: The scheme is ideal since each participant holds one share which has the same size of the hash.
- c) Fast recovery of secret: The calculation of hash function is fast, this can assure that the partial Latin square and hence the full Latin square can be recovered quickly.
- d) Avoidance of critical sets: Under the new scheme, looking for critical sets of large size can be avoided. This makes it more efficient and better controlled as discussed above.
- e) Application of minimal authorized subset: As we explained earlier, we can speed up the whole process by considering the minimal authorized subset only. Given any access structure Γ , $A \in \Gamma$ is called a minimal authorized subset if $B \subset A$ then $B \notin \Gamma$.
- f) General access structure: As we shall see in the following example, this approach can be extended to general access structure.

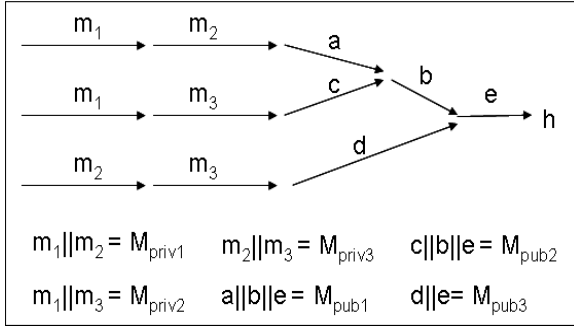


Figure 4. A (2, 3) threshold scheme example.

g) In a traditional secret sharing scheme, we have a secret first and then set up and distribute shares to the participants. In our scheme, we first set up shares for participants and then create a secret later. We can say the shares and secret are set up in the same setting. This should make the scheme more efficient.

Example. A (2, 3) threshold scheme. Let m_1, m_2 , and m_3 be shares of participants P_1, P_2 , and P_3 , respectively. The access structure consists of four authorized subsets, also shown in Figure 4. $M_{\text{pub1}}, M_{\text{pub2}}, M_{\text{pub3}}, M_{\text{pub4}}$ will be the linking messages stored in the public area.

- a) $\{P_1, P_2\}$ $m_1 || m_2 || M_{\text{pub1}}$
- b) $\{P_1, P_3\}$ $m_1 || m_3 || M_{\text{pub2}}$
- c) $\{P_2, P_3\}$ $m_2 || m_3 || M_{\text{pub3}}$
- d) $\{P_1, P_2, P_3\}$ $m_1 || m_2 || m_3 || M_{\text{pub4}}$

As mentioned above we only consider the minimal authorized subset of the access structure. In this case, we can skip $m_1 || m_2 || m_3 || M_{\text{pub4}}$.

Suppose we know P_2, P_3 are family members or good friends, we don't want them to recover the secret. Then, a general (2, 3) threshold scheme doesn't work. In our case, we can just simply skip the setup of $m_2 || m_3 || M_{\text{pub3}}$.

It is easy to show that this method is good for any general access structure.

7.5 Set up a verifiable scheme

A cryptographic hash function has an application as a message authentication code to certify that original message was not altered. We can apply this idea to secret sharing scheme so that any dishonest participant who does not return the original

share will be caught in the secret recovering stage. On the other hand, the participants can verify whether the dealer really sends out consistent shares for them to keep.

Since the different schemes may have different settings, let's assume the following:

- a) the dealer is honest;
- b) the dealer does not exist after the initial setup.

Let f, g be cryptographic hash functions. Let $m_1, m_2, \dots$ be the shares of the participants $P_1, P_2, \dots$. The dealer distributes each share to each participant and then publishes the hashes (by hash function g) of each share as commitments $g_1, g_2, \dots$, as in Feldman's case [14], in a public area.

Participant i verifies his/her share by checking if $g(m_i) = g_i$ holds. If all participants confirm that taking his/her share as input to the hash function g , he/she gets the hash value equal to one of the commitments published by the dealer, we conclude the dealer sends out consistent shares. Likewise, when the participants of any authorized subset combine their shares during the secret recovering stage, they can be verified.

Hash function g is used to make the scheme verifiable. Hash function f is used to recover the shared secret: $f(M_{\text{priv}} || M_{\text{pub}})$. Partial information was given out here, however, if g is preimage resistant, it would be infeasible to find the original share m_i from g_i . Participant i can fool the party if he or she can find m'_i such that $g(m_i) = g(m'_i) = g_i$. However, this is also extremely difficult to achieve if g is second preimage resistant.

8 Limitations of our proposed scheme

a) It is much more difficult to find a preimage or a second preimage than a collision in a hash function. So, it would be infeasible to set up the scheme to recover a particular fixed secret. By the same reason, it is also much difficult to set up a proactive scheme which requires renewal of new shares of the participants but the original secret remains unchanged. So, we suggest the following ways to handle this.

- (1) Safeguard a fixed secret: we use a secret encryption/decryption system for the original secret. Generate a random symmetric key by the proposed scheme. For any secret sharing scheme, it would make sense to safeguard

a short secret, such as a symmetric key, rather than the original large secret, such as a classified document.

- (2) Proactive secret sharing: we renew the shares periodically in order to increase the security, but we want to keep the secret unchanged. The main reason is to avoid spending time to repeat the process. However, as mentioned in 5.3 g), our new scheme will generate the shares and secret at the same time. If we repeat the process, we actually simulate a proactive scheme.

b) The number of public messages may be exponential depending on the access structure. A threshold scheme is an example of this. However, in practical situation, we expect the size of the authorized subsets should be small and it is justified to use a comparatively low cost of public area to store them.

c) Using a hash function can recover the secret fast and this certainly fulfills the requirements of availability. However, if a scheme demands a fast initial setup, then it would be a challenge as to how fast we can build a diamond structure. In this case we need to do the setup by parallel computations. P. van Oorschot and M. Wiener [32] have done research in this area.

9 Conclusion and future research

In this paper, we use cryptographic hash functions to improve the security and performance of secret sharing schemes based on a Latin square or its critical sets. We store a partial Latin square in a hash for a fast retrieval of the shared secret; we set up an ideal perfect $(t + 1, n)$ threshold scheme with different desirable properties. This also applies to any general access structure. The security of the scheme depends on the number of the Latin squares. If the number is too large for the attacker to do the exhaustive search, it will be safe.

One direction is the practical aspect. We believe that application of cryptographic hash functions to secret sharing schemes and effective implementation of such schemes is an interesting research area in the future. On the theoretical side, how to set up a mathematical model for complexity analysis is another direction. How to speed up the construction of the diamond structure and extend the idea to a more general case would be another future research area to explore.

Acknowledgments. The authors would like to thank Professor Michael Anshel for his valuable discussions. We also want to thank Professor Joseph Vaisman for getting us many useful references, and Mr. Vincent Falco for proofreading the draft.

Bibliography

- [1] G. R. Blakley, Safeguarding cryptographic keys, in: *Proc. of the National Computer Conference, American Federation of Information Processing Societies Proceedings* 48, pp. 313–317, 1979.
- [2] D. Bogdanov, *Foundations and properties of Shamir's secret sharing scheme*, University of Tartu, available online www.cs.ut.ee/~peeter_1/teaching/seminar07k/bogdanov.pdf, 2007.
- [3] R. M. Capocelli, A. De Santis, L. Gargano and U. Vaccaro, On the size of shares for secret sharing schemes, *Journal of Cryptology* **6** (1993), 157–167.
- [4] G. Chaudhry, H. Ghodosi and J. Seberry, Perfect Secret Sharing Schemes from Room Squares, *Journal of Combinatorial Mathematics and Combinatorial Computing* **28** (1998), 55–61.
- [5] G. Chaudhry and J. Seberry, Secret sharing schemes based on Room squares, in: *Proc. of DMTCS'96 - Combinatorics, Complexity and Logic*, pp. 158–167, 1996.
- [6] C. Chum and X. Zhang, Applying hash functions in the Latin square based secret sharing schemes, in: *Proc. of The 2010 International Conference on Security and Management (SAM'10)*, pp. 197–203, 2010.
- [7] C. J. Colbourn, M. J. Colbourn and D. R. Stinson, The computational complexity of recognizing critical sets, in: *Proc. of Graph Theory Singapore 1983*, Lecture Notes in Mathematics 1073, pp. 248–253, 1983.
- [8] C. J. Colbourn, The Complexity of Completing Partial Latin Squares, *Discrete Applied mathematics* **8** (1984), 25–30.
- [9] J. A. Cooper, D. Donovan and J. Seberry, Secret sharing schemes arising from Latin squares, *Bulletin of the ICA* **12** (1994), 33–43.
- [10] I. Damgård, A design principle for hash functions, in: *Proc. of CRYPTO 1989*, LNCS 435.
- [11] R. D. Dean, *Formal Aspects of Mobile Code*, Ph.D. thesis, Princeton University, 1999.
- [12] D. Donovan, J. A. Cooper, D. J. Nott and J. Seberry, Latin squares: Critical sets and their lower bounds, *Ars Combinatoria* **39** (1995), 33–48.
- [13] T. Evans, Embedding incomplete Latin squares, *The American Mathematical Monthly* **67** (1960), 958–961.
- [14] P. Feldman, A practical scheme for non-interactive verifiable secret sharing, in: *Proc. of the 28th IEEE Symposium on the Foundations of Computer Science*, pp. 427–437, 1987.
- [15] P. Hall, On representatives of subjects, *Journal of the London Mathematical Society* **10** (1935), 26–30.

- [16] A. Herzberg, S. Jarecki, H. Krawczyk and M. Yung, Proactive secret sharing, in: *Proc. of CRYPTO 1995*, LNCS 963, 1995.
- [17] M. Ito, A. Saito and T. Nishizeki, Secret sharing scheme realizing general access structure, in: *Proc. of IEEE GLOBECOM 1987*, pp. 99–102, 1987.
- [18] A. Joux, Multicollisions in iterated hash functions. Application to cascaded construction, in: *Proc. of CRYPTO 2004*, LNCS 3152, pp. 306–316.
- [19] J. Kelsey and T. Kohno, *Herding hash functions and the Nostradamus attack*, Cryptology ePrint Archive, Report 2005/281, 2005.
- [20] J. Kelsey and B. Schneier, Second preimages on n -bit hash functions for much less 2^n work, in: *Proc. of EUROCRYPT'05*, LNCS 3494, pp. 474–490, 2005.
- [21] B. D. McKay and I. M. Wanless, On the number of Latin squares, *Ann. Combin.* **9** (2005), 335–344.
- [22] R. C. Merkle, One way hash function and DES, in: *Proc. of CRYPTO 1989*, LNCS 435, pp. 428–446, 1989.
- [23] G. Mullen and C. Mummert, *Finite Fields and Applications (Student Mathematical Library)*, American Mathematical Society, 2007.
- [24] A. Shamir, How to share a secret, *Communications of the ACM* **22** (1979), 612–613.
- [25] C. E. Shannon, A mathematical theory of communication, *Bell Systems Technical Journal* **27** (1948), 379–423, 623–656.
- [26] C. E. Shannon, Communication theory of secrecy systems, *Bell Systems Technical Journal* **28** (1949), 656–715.
- [27] B. Smetaniuk, A new construction on Latin square, I: A proof of the Evans' Conjecture, *Ars Combinatoria* **11** (1981), 155–172.
- [28] M. Stevens, A. K. Lenstra and B. Weger, *Predicting the winner of the 2008 US Presidential Elections using a Sony PlayStation 3*, Available online www.win.tue.nl/hashclash/Nostradamus, November 2007.
- [29] D. Stinson, *Cryptography, Theory and Practice*, 3rd ed, Chapman and Hall/CRC, 2005.
- [30] W. Trappe and L. Washington, *Introduction to Cryptography with Coding Theory*, 2nd ed, Prentice Hall, 2006.
- [31] J. H. van Lint and R. M. Wilson, *A Course in Combinatorics*, 2nd ed, Cambridge University Press, 2001.
- [32] P. van Oorschot and M. Wiener, Parallel collision search with cryptanalytic applications, *Journal of Cryptology* **12** (1999), 1–28.
- [33] Y. Zheng, T. Hardjono and J. Seberry, Reusing Shares in Secret Sharing Schemes, *Computer Journal* **37** (1994), 199–205.

Received June 22, 2010.

Author information

Chi Sing Chum, Computer Science Department, Graduate Center, CUNY, 365 Fifth Ave., New York, NY 10016, USA.

E-mail: cchum@gc.cuny.edu

Xiaowen Zhang, Computer Science Department, College of Staten Island, CUNY, 2800 Victory Blvd, Staten Island, NY 10314, USA.

E-mail: xiaowen.zhang@csi.cuny.edu