ANALYSIS OF KEY PREDISTRIBUTION SCHEMES IN WIRELESS SENSOR NETWORKS

By

Naren K.Gundimeda

A project submitted to School of Graduate Studies

In partial fulfillment of the requirements for the degree of

Master of Science (Scientific Computing)

Faculty of Science

Memorial University of Newfoundland

December 2014

St. John’s, Newfoundland

Table of Contents

List of Tables

List of Figures

List of Acronyms

1 Key Predistribution Scheme (KPS)

2 Balanced Incomplete Block Design (BIBD)

3 Orthogonal Latin Squares (OLS)

4 Mutually Orthogonal Latin Squares (MOLS)

5 Group Divisible Designs (GDD)

6 Transversal Designs (TD)

7 Common Intersection Design (CID)

ABSTRACT

Combinatorial designs are used for designing key predistribution schemes that are applied to wireless sensor networks in communications. This helps in building a secure channel. Private-key cryptography helps to determine a common key between a pair of nodes in sensor networks. Wireless sensor networks using key predistribution schemes have many useful applications in military and civil operations. When designs are efficiently implemented on sensor networks, blocks with unique keys will be the result. One such implementation is a transversal design which follows the principle of simple key establishment. Analysis of designs and modeling the key schemes are the subjects of this project.

Acknowledgement

I would like to thank Dr. Nabil Shalaby giving me step by step guidance from start to end of the project. He mainly understands the student’s ability and makes the student do hard work has inspired me to fully focus on the project.

I would also thank Dr. Martin Plumer for his overall support and encouragement made the project complete and to the writing centre too, which helped in writing this report.

1 Introduction

Distributed Sensor networks have many applications, such as military and civil operations and they are used for target tracking and the monitoring of nuclear power plants. Protecting data needs a special technique called cryptography (secret writing). Secret writing converts the message into a format understandable only to the sender and receiver. Cryptographic systems are divided into two categories: public-key cryptography (a different key for sender and receiver to encrypt and decrypt the data) and private-key cryptography also called symmetric key cryptography (which shares a common key for the sender and receiver). Because cryptography has mathematical operations such as substitution and transposition, the security goals, such as confidentiality, authentication and data integrity are achieved. Confidentiality means protecting the data from unauthorized users, while authentication is to verify the data from the sender and data integrity ensures the data received is not modified.

Key establishment is an important aspect in achieving these goals. Managing the keys uses a set of elements for key generation from a trusted authority. Key management involves key generation, key establishment, key update and key revocation. Key generation is the process of generating the keys from the trusted authority while key establishment generates the key rings by the trusted authority and sends them to the nodes accordingly. In the key-update phase keys are updated after certain time period and revocation deletes compromised keys. Cryptographic terms are described via key, plaintext and ciphertext.

Plaintext is the original data while the key is secret and the sender uses mathematical operations for maintaining the secrecy. Ciphertext is the converted to original data using a key. The process of converting the original data to ciphertext is called encryption and the reverse process is called decryption. Private-key cryptography is employed in sensor networks due to its computational and communication cost efficiency [1].

1.1 Sensor Network

A sensor network consists of tiny sensor nodes present in a fixed network topology with limited resources such as network capability. Fig.1 by Camtepe and Yener [1] explains how sensor network functions in secure channels. Here, we can obeserve the direct link between a and b, and a and c, but we can establish a path through a-c-b so that a and b do not share a key. The length of this path is called the key-path length.

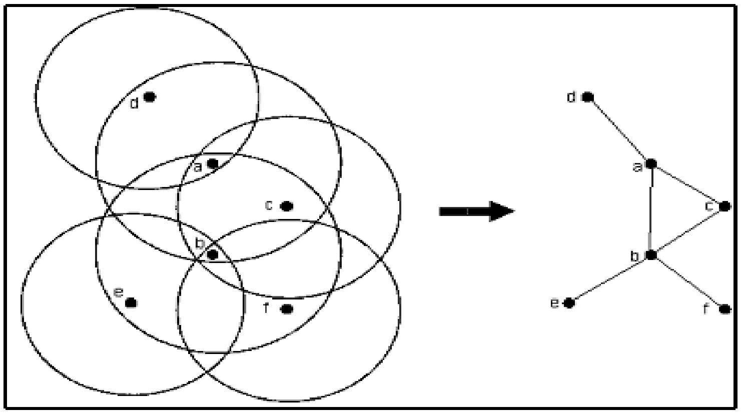


Fig.1: Wireless Sensor Network [1].

1.2 Features of Sensor Networks for Key Management

Cryptographic key management is a challenging task in a sensor network due to its vulnerability to external attacks. Sensor networks have limited resources which make it difficult to implement public key cryptographic algorithms; because of its wireless nature makes it is easy to eavesdrop, and a definite topology is present in a sensor network.

Therefore the better choice would be a key predistribution scheme to establish the keys. In key predistribution, we generally choose keys from a large pool. The main goals of the key establishment are:

1. Key connectivity: If the sensor nodes in a neighbourhood share a common key, then the probability of key connectivity can be described as $P_c = L/N(N-1)/2$, where L is number of links and N is number of nodes

2. Resiliency: When nodes are captured or compromised, the rest of the network is disconnected by affecting s' (number) of nodes. This can be measured by $E(s) = L'/L$, where L is number of links that are affected before s nodes are compromised and L’ is the number of links that are affected after s nodes are compromised.
3. Storage and computational requirements must be kept to a minimum value as the sensor network consists of many (in some uses thousands) tiny sensors.
4. Key revocation: An efficient mechanism should be present for revoking the keys.

1.3 Key Establishment Phases

In [2], the key establishment is described in three phases:

1. Key Predistribution: Before deployment the keys should be preloaded onto the key rings.
2. Shared key discovery: When two nodes are communicating it is better to find a common key.
3. Key Establishment: If a common key does not exist, it is better to choose a key path for secure communication.

Key predistribution schemes are mainly classified into three types: Probabilistic schemes draw keys randomly from the key rings while deterministic schemes follow a definite pattern of the key pool and hybrid approaches use both schemes.

The algorithm and key predistribution are deterministic for this report.

1.4 Main applications of Private Key Cryptography

The main applications of private key cryptography are telecommunications, optics, military service and coding in civil operations, error correcting codes etc.

2 Combinatorial Designs for Private Key Cryptography: Definitions

2.1 Basic Designs

A Combinatorial design is a pair of parameters (X, A) where X represents points or elements, and the A represents blocks or subsets of the elements, for example:

X= {1, 2, 3, 4}.

A= {123, 134, 234, 124}.

This design satisfies certain properties (for example, see BIBD below). This terminology is mainly used in predisitribution schemes as they help to communicate between the nodes in a sensor network (see appendix D for relation between BIBD and hadamard design).

2.1.1 Balanced Incomplete Block Designs (BIBD)

Balanced incomplete block design (BIBD) is a 3 parameter set:

S= (v, k, λ)

Where v is a set of points or elements.

k- Number of elements in the blocks.

λ- Number of times elements occur in the blocks.

An example is:

S = (7, 3, 1).

B1=(1, 2, 4) ,B2=(2, 3, 5) ,B3=(3, 4, 6), B4=(4, 5, 7) ,B5=(5, 6,1) ,B6=(6, 7, 2) ,B7=(7, 1, 3).

Keys = (k1, k2, k4), (k2, k3, k5), (k3, k4, k6), (k4, k5, k7), (k5, k6, k1), (k6, k7, k2),(k7, k1, k3).

There are 7 elements where v= {1, 2, 3, 4, 5, 6, 7}, k=3, λ=1.

Each block should contain a set of elements from set S, so all blocks will have only some elements, if elements repeat only once in blocks then it is called a simple design.

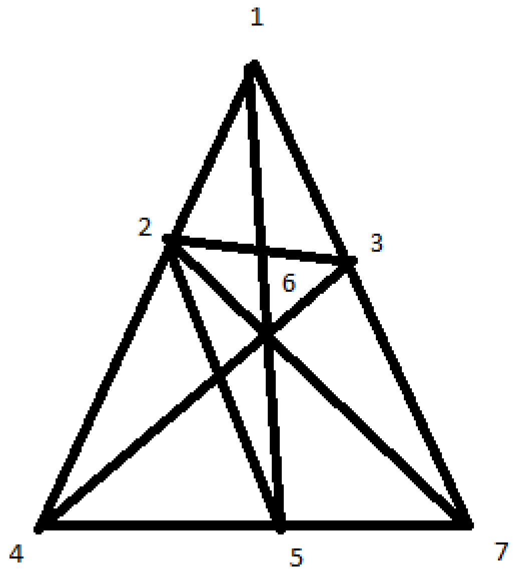


Fig.2: Finite projection of order 2 [3].

In this (v, k, λ) design we have a given example of (7, 3, 1), also known as a Steiner triple system because it has 3 blocks of elements and an affine plane of order n in Fig.2 [3]. This is also a projective plane of order 2 with parameters as (n^2+n+1, n+1, 1)-BIBD.

2.1.1.1 Parameter Conditions on BIBD

The parametric conditions on the BIBD can be described as follows. Every design satisfies the following constraints [4]:

Theorem 1:

λ (v-1)=r(k-1).

Proof:

Let (A, B) be the design where A represents the elements and B represents the blocks, S represents the BIBD (v, k, λ).

Consider the element y is present in r blocks then the other elements form pairs with x and makes r(k-1) blocks. On the other hand, the element present in λ blocks make pairs with v-1 blocks. Combining the two statements derives the result.

Theorem 2:

bk = vr

Let (A, B) be a (v, k, λ)-BIBD, and let b=|B| and size of the blocks B have k elements gives bk elements. Consider the element x is made of r choices and it is present in v blocks. This makes it appear in vr elements which give the final result [4].

Usually BIBD is converted to 5 parameter set (v, b, k, r, λ) design where r is repetition number which is 3 (total number of repetition of elements in all blocks with b determined by using the formula):

$b= \lambda (v(v-1))/k(k-1)$.

An example of the parameter design is the same as the above block design with (7, 7, 3, 3, 1) where block size is 3 and repetition number is also 3.

The incidence matrix of such a design can be described as follows:

A is a b*v matrix and A=a_{ij}.

a_{ij}=1if ith block contains the jth element or a_{ij}=0 otherwise .

For example the Blocks {1,2,3} {2,3,4} {3,4,1}{4,1,2}.

In Table 1 the incidence matrix of the BIBD can be described in binary values as 1's and 0's.

Index i, j	Matrix elements			
	1	2	3	4
1	1	0	1	1
2	1	1	0	1
3	1	1	1	0
4	0	1	1	1

Table 1: Example of Incidence Matrix.

2.1.1.2 Complementary BIBD

In (v, b, r, k, λ) [4] block designs, if there exists S of v elements then there is a complementary BIBD such that the elements are of the S-B blocks. For (v, b, r, k, λ) such that $k \leq v-2$ the complementary design would be (v, b, b-r, v-k, b-2k+λ).

2.1.1.3 Symmetric BIBD

In [4] a BIBD, if one of the parameters is equal to other such that v = b or k = r then it is called a symmetric BIBD. A complementary BIBD can be symmetric.

In BIBD we have parameters (v, k, λ) where in the blocks if there are common elements present then such designs form the symmetric designs.

$A1 \cap A2= \lambda$

Take the BIBD, for example,

S = (v, b, r, k, λ),

(7, 7, 3, 3, 1)

Blocks B= {123, 145, 167, 245, 267, 345, 367}

Here, we can observe that one of the parameters is equal to others, v=7=b and r=k=3. The common intersection element is λ=1 in the blocks. In detail the blocks b1 = {123} and b2 = {145} where the common element is 1. The complementary design can be symmetric (see complementary design definition).

2.2 Latin Squares

A matrix of order n*n such that each of n symbols occurs once in each row and each column. The number n is called order of the square. Example of latin square of order 3 is defined in table 2:

1	2	3
3	1	2
2	3	1

Table 2: Latin Squares.

2.2.1 Orthogonal Latin squares

Suppose that L_1 is a Latin square of order n with entries from X and L_2 is a Latin square of order n with entries from Y. “We say that L1and L2 are orthogonal Latin squares provided that, for every x ∈ X and for every y ∈ Y, there is a unique cell (i, j) such that L_1 (i, j) = x and L*2* (i, j) = y” [4].

2.2.2 Mutually Orthogonal Latin squares (MOLS)

“A set of m Latin squares of order n, say $L_1,...,L_m$, are said to be mutually orthogonal Latin squares (MOLS) if L_i and L_j are orthogonal for all $1\leq i < j \leq m$”. The Latin squares are called mutually orthogonal if they form orthogonal pairs [4].

2.3 Group Divisible Designs

A group divisible design consists of three parameters such as (X, G, A) with the following the properties, where X represents the set of elements, G represents the group comprising the non-empty sets of X, and A represents the blocks of X. The group and block have one common point and every pair of distinct points contained in one block [4].

An example of GDD is S = (9, 3, 1)-BIBD

“X = {1, 2, 3, 4, 5, 6, 7, 8, 9}, and

A = {123, 456, 789, 147, 258, 369, 159, 267, 348, 168, 249, 357}.

If we delete the point 1, for example, then we obtain the following GDD (Y, G, B):

Y = {2, 3, 4, 5, 6, 7, 8, 9},

G = {23, 47, 59, 68}, and

B = {456, 789, 258, 369, 267, 348, 249, 357}” [4].

2.4 Transversal Designs

Let k ≥ 2 and n ≥ 1, then a transversal design satisfies the following properties with (X, G, B) [4].

i) X is the number of elements.

ii) G is the partition in groups.

iii) B is blocks of X.

iv) Any group or any block has a common point (element).

v) Every pair of points is contained in exactly one block.

Example:

From group divisible design we have the following groups:

1	2	3	4
5	6	7	8
9	10	11	12

Table 3: Elements Represented in Groups.

Blocks are represented as the following:

B={1,5,9}{1,6,10}{1,7,11}{1,8,12}{2,5,11}{2,6,12}{2,8,10}{2,7,9}{3,6,9}{3,5,10}{3,7,12}{3,8,11}
{4,5,12}{4,8,9}{4,7,10}{4,6,11}

In Table 3 we have elements arranged as groups and blocks are arranged having a common element in each block, starting at the initial block.

2.5 Projective Planes

A BIBD that forms the plane of order n having the parameters (n^2+n+1, n+1, 1) will be called a projective plane of order n such that $n \geq 2$. For example if n=2, then the projective plane would be (7, 3, 1). If the BIBD in [4] satisfies the following property in the projective plane then it is a symmetric BIBD:

$n^2+n= (n+1)\, n$

So every point in BIBD will intersect at blocks.

2.6 Affine Plane

Let $n \geq 2$ then there exists BIBD which satisfies the parameters such that (n^2, n^2+n, n+1, n, 1) is affine plane of order n.

2.7 Common Intersection Design

Assume there are nodes N_i and N_j such that they can communicate with each other but don't have a common key while the intermediate node will have common key N_h.

Assume there is BIBD with parameters (v, b, r, k) with A elements and B Blocks.

Then the condition of common intersection is $B_h \in B$ and $B_i \cap B_j = \emptyset$ and

$B_i \cap B_h = \emptyset \geq \mu$ (order of CID) [5].

3 Literature Review

Sensor networks use combinatorial design for predistributing the keys before deployment. Various schemes were proposed that follow the probabilistic, deterministic or hybrid patterns which can be applied to derive transversal designs. Various methodologies have been proposed and are also discussed in this chapter.

3.1 Previous Schemes

3.1.1 Blom's Scheme

Blom proposed a k-order matrix scheme over a finite field GF(q) (Galois field [3]). It has a private symmetric matrix S and public Vandermonde matrix [6] as P. By using the private matrix we obtain secrecy, and by using the public matrix we have the data to establish the keys between the nodes. The columnar matrix will be P which is multiplied with S and results in $A = (S*P)^T$. This gives rows of matrix A and columns of matrix P forming the nodes. The private matrix column would be col_j and the row for matrix A is row_i. For generating the keys we have to exchange the columns in the matrix P and multiply by A. Blom's scheme follow the pattern of sharing the key and path length as 1 which tells us this scheme is deterministic. When the resultant matrix and the private matrix are multiplied together we have a symmetric matrix K=A*P generating the keys. If Blom's scheme is applied to large numbers, the matrix multiplication will be a costly computation. Blom's scheme will affect the resiliency when the number of nodes is increased.

Du et al. [7] used Blom's scheme with *ῳ* spaces to increase the resiliency such that each random node uses r spaces out of *ῳ* spaces. When resiliency is increased there will be no key space. If the probability ratio of *r/ῳ* decreases then the probability of sharing the key also decreases.

3.1.2 Song's Scheme

Chan, Perrig and Song [8] proposed a random pairwise key scheme where the probability of any two nodes connecting is p, in the network of n nodes where each node needs to store (np) pairwise keys instead of n-1 keys. As each node will have distinct keys, perfect resiliency is established.

Slijpevic et al. [9] proposed that sensor nodes share a list of master keys by using a random function and a seed, where every sensor node will share the random function and a seed to select the network-wise key.

3.1.3 Gilgor's Scheme

Eschenauer and Gilgor [10] proposed a random key pairwise distribution scheme. In this scheme we can supply many keys before deployment. We have to generate the pool P along with their identities for each sensor. As the identities are present we can generate the keys k from pool P.

Thus, the k keys and their identities are loaded into the memory of the sensor node. In this process, only identities are exchanged without any privacy mechanism. Gilgor's scheme uses many mathematical operations and requires much storage for each sensor node. If two sensor nodes share a common key the set of keys will be generated.

Chan et al. [8] proposed a scheme to develop q-composite predistribution keys, which uses the key pool but increases the overlapping of keys. This will need additional security for the nodes. Since there are q-composite keys, each node will have unique keys but the problem is that the number of nodes are compromised, which will affect the pairwise keys. If we limit the network size, each sensor node will have pairwise key, which will reduce the number of nodes being compromised.

3.2 Symmetric Designs and Key Distributions in Sensor Networks

To describe the BIBD in a finite projection plane, there are a finite set of points and set of subsets or lines, where number of points are $n \geq 2$ such that each point contains the parameters of a symmetric design as $(n^2+n+1, n+1, 1)$. The symmetric design follows the properties of having a point with n+1 lines where every n+1 point has n^2+n+ 1 lines. If we define the symmetric design in terms of a complementary design, then block design, S = (v, k, λ)-BIBD with the blocks D= (D_1,D_2,…,D_n). Then the complementary design of BIBD with five parameters is S'= (v, b, b-r,v-k,v-2k+ λ) and with three parameters S'= (v,v-k,v-2k+ λ) [1]. We have an original design and a complementary design that are symmetrical to each other and contains the intersection element λ [1].

For example:

S= (v, k, λ) = (7, 3, 1)

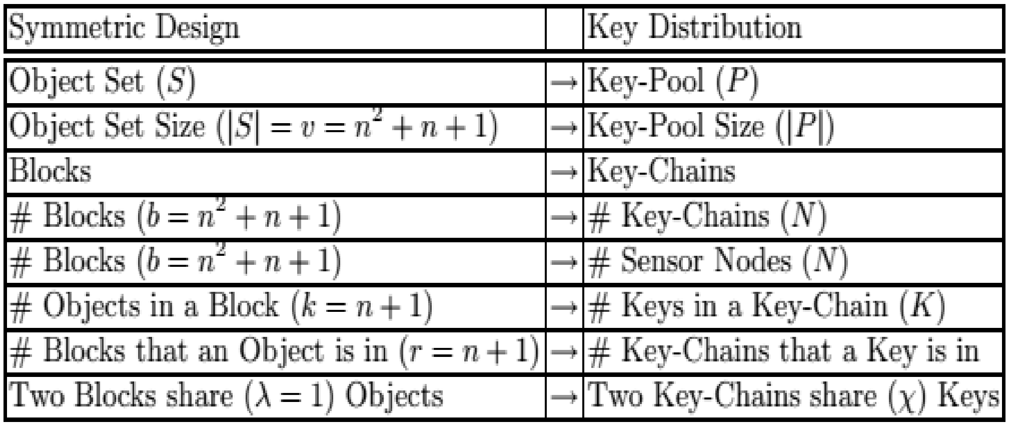
S'= (7, 4, 2)

D = {123,145,167,245,267,345,367}

D'= {1467, 1256, 2367, 2345, 4567, 4567, 1256}.

3.2.1 Combining the Symmetric Design and Key distribution

Table 4 below [1] will help explain the relationship between the key distribution and the symmetric design where N is number of nodes before deployment. Keys will be selected from the key pool, and then the parameters will be changed as follows by having key chains as blocks and key pool size as the network size in the sensor network: if two nodes share a common key, then the intersection would be λ=1 or X keys, such that they follow the conditions v = b and r = k.



Symmetric Design		Key Distribution
Object Set (S)	→	Key-Pool (P)
Object Set Size ($\|S\| = v = n^2 + n + 1$)	→	Key-Pool Size ($\|P\|$)
Blocks	→	Key-Chains
# Blocks ($b = n^2 + n + 1$)	→	# Key-Chains (N)
# Blocks ($b = n^2 + n + 1$)	→	# Sensor Nodes (N)
# Objects in a Block ($k = n + 1$)	→	# Keys in a Key-Chain (K)
# Blocks that an Object is in ($r = n + 1$)	→	# Key-Chains that a Key is in
Two Blocks share ($\lambda = 1$) Objects	→	Two Key-Chains share (χ) Keys

Table 4: Representation of relationship between a Symmetric design and Key distribution [1].

3.2.2 Algorithm Construction

In order to construct the design, a complete set of orthogonal latin squares is defined such that an *n x n* array of n symbols is defined, which is called a mutually orthogonal latin square. Suppose we define the latin squares as A and B, such that in each matrix a_{ij} and b_{ij} there are *n x n* arrays, then we have distinct pairs of elements in each matrix.

Steps in the algorithm are with a given network size N and we have to determine the prime power n such that it satisfies the condition (n^2+n+1 ≥ N) and derive the n-1 orthogonal latin squares mentioned in [3]. Then we will construct the affine plane of the form (n^2, n, 1) which will finally have a projective plane as (n^2+n+1, n+1, 1). This follows the symmetric property.

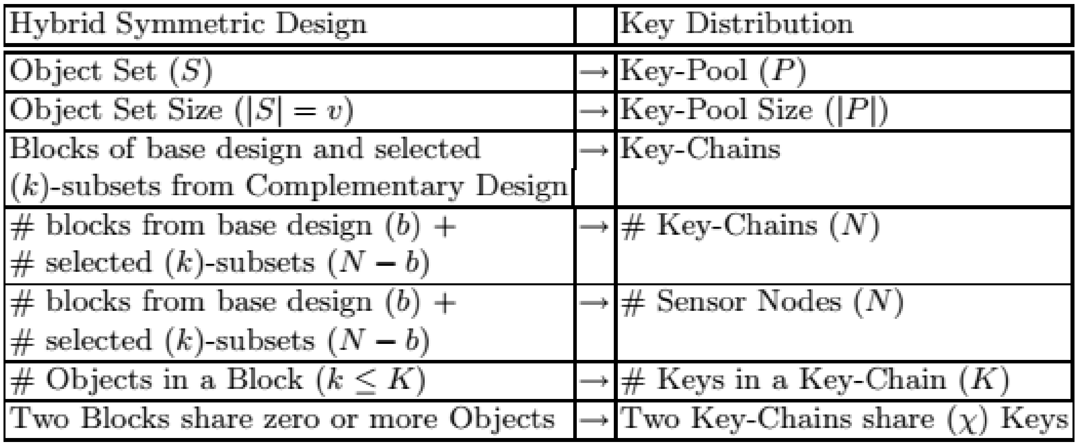
3.2.3 Analysis

In a symmetric design, the average key path length would be 1, and the probability of key share is also 1. Here the resiliency is measured as an important metric for security, as the attacker has defined the nodes selectively or randomly. The key chain size is defined as n+1. The attacker has to test the keys with n+1 key chain. If the symmetric design property is applied to the key chain, then there will be n^2+n keys which must be paired for the n+ 1 keys. In this way the attacker can capture only n^2+1 keys.

Symmetric designs do not support scalability when the sharing probability is 1 for the given fixed key chain size such that N> n^2+n+1, where n is a prime number. For larger networks, the probability of key predistribution schemes will increase and have a possible number of distinct key chains. Symmetric designs do not satisfy the interconnectivity property (of physically being connected when nodes are compromised). So to improve the interconnectivity property, hybrid schemes should be used. These are discussed in the next section.

3.3 Hybrid Designs for Scalable Distributions

Here we use N sensor nodes divided into blocks of size k of object size v [1]. The blocks are symmetric such that N-b blocks are selected among k subsets of the complementary design. Blocks with at most K keys come from the key ring or key pool. In table 5 we have the relation between hybrid design and key predistribution with parameters defined.



Hybrid Symmetric Design		Key Distribution
Object Set (S)	→	Key-Pool (P)
Object Set Size ($\|S\| = v$)	→	Key-Pool Size ($\|P\|$)
Blocks of base design and selected (k)-subsets from Complementary Design	→	Key-Chains
# blocks from base design (b) + # selected (k)-subsets ($N - b$)	→	# Key-Chains (N)
# blocks from base design (b) + # selected (k)-subsets ($N - b$)	→	# Sensor Nodes (N)
# Objects in a Block ($k \leq K$)	→	# Keys in a Key-Chain (K)
Two Blocks share zero or more Objects	→	Two Key-Chains share (χ) Keys

Table 5: Representation of relationship between Hybrid scheme and Key distribution [1].

3.3.1 Description

In a given key chain of size K and network size N, the hybrid design will be a prime power *n* such that in terms of k, where the blocks are of v-k, represent the complementary design and N-b uniformly scales up to form complementary design blocks.

In summary we have to use a key chain size as K and number of nodes as N with power n. We also take the value $k \leq K$. With these parameters, we derive the symmetric design of pool size P and blocks B. Based on this description we would have a complementary design which will be processed to give a final hybrid design.

An example of hybrid design follows:

Generate the key chains with network size as 10 and assume that BIBD with parameters as: $S = (v, k, \lambda) = (7, 3, 1)$.

Form the symmetric blocks form:

B= {123,145,167,246,257,347,346} with the complementary symmetric design as: B'= {4567, 2367, 2345,1357,1346,1256, 1247}.

Assume the subsets as H= {456, 236, 157}.

The blocks of hybrid symmetric design [1] are thus:

B ∪ H= {123, 145, 167, 246, 257, 347, 346, 456, 236, 157}.

3.4 Methods of Key Predistribution

3.4.1 Polynomial Pool Based Key Predistribution

To summarize the polynomial-based predistribution, we have to establish pairwise keys and randomly generate the bivariate t-degree polynomial [11].

$\sum_{i,j=0}^{t} a_{i,j}x^i y^j = f(x,y).$

This polynomial is applied over a finite field F_q, such that q is a prime number and satisfies the property *f(x,y)=f(y,x).* Here, each sensor will have a unique ID. For example, the setup server computes a polynomial share *f(x,y)=f(i,y),* which is used to compute the common key at point i, and the same theory is used to exchange at point j as *f(j,i)=f(i,j).* The t-degree polynomial occupies (t+1) log q storage space to store a polynomial *f(i,x),* and to establish the key, it has to assess the ID of q neighbouring sensor nodes.

In [11] the general framework for pairwise key establishment is described. This framework generates the random bivariate polynomial for sensor networks. Pairwise key establishment is divided into 3 phases:

I) Set-up Phase

II) Direct Key Establishment

III) Key Establishment

3.4.1.1 Set-up Phase

In this phase, the server will randomly generate the bivariate polynomial of t-degree which is represented as F over a finite field F_q, and assigns a polynomial with a unique ID. The setup server picks up a subset of polynomials that are assigned at sensor node i *($F_i \subseteq F$).* Here the problem is to find the subset of polynomials for all sensors.

3.4.1.2 Direct Key Establishment Phase

In this phase we have to generate a pairwise key for the sensor node, if it is not present already, so that the polynomial will have a shared pairwise key. The problem is finding the common bivariate polynomial which is solved by either using predistribution or real-time discovery techniques.

3.4.1.2.1 Predistribution

In [11], the predistribution technique is described, where the setup server will predistribute the information or data to the sensors so that if one node shares the key with another node, it can establish a pairwise key. Even though this method is simple, it cannot connect to the networks on demand because the server will have to be informed about additional sensors attached to the network. In predistribution, the main disadvantage is that the attacker can identify the polynomial easily and target the sensor nodes that share the polynomial which is solved by real time discovery technique.

3.4.1.2.2 Real time discovery

Real time discovery technique [11] is used to build the polynomial on demand if two sensors have a common bivariate polynomial. The first solution for building the polynomial on the fly is to swap the

ID's of polynomials, which have a common polynomial. For example, sensor node i may broadcast an encryption list, α, $E_{K_v}(\alpha), v = 1, ..., |Fi |,$ where K_v is a potential pairwise key the other node may have, but overhead is problematic [11].

3.4.1.3 Key Establishment Phase

In this phase to establish a pairwise key, we have a path between the nodes a and b such that it forms a sequence and establishes the key. The problem is that two nodes cannot communicate directly. The solution is to use predistribution and real time techniques.

3.4.1.3.1 Predistribution

The technique is same as in the direct key establishment phase (see previous section), but the server will store the ID of the sensor node to the other node and will find the key path.

3.4.1.3.2 Real time discovery

This method builds the nodes on demand if there is a key path and establishes the pairwise key from the intermediate node to the destination node which increases the communication overhead.

3.4.2 Random Subset Assignment Method

In the random subset method, the server will select the random subset of polynomials as F and assign it to the server. There is a unique key between each pair of sensors, which shares the polynomial. The problem is that if there are key shares, no pairwise key is constructed. It has three steps:

3.4.2.1 Subset assignment

In this step, you have a bivariate t-degree polynomial which is generated randomly over a finite field and a random set of polynomials are picked and assigned to the sensor node.

3.4.2.2 Polynomial Share Discovery

In this step, sensors will find the common polynomial by using the real discovery technique and broadcast a polynomial ID as $E_{K_v}(\alpha), v = 1, ..., |Fi |,$ where K_v is a potential pairwise key the other node may have.

3.4.2.3 Path Discovery

In the path discovery step, a source node will try to find another node's ID that can set-up a common key. The process is to broadcast the list of polynomials which includes ID's of source node and destination node. If one of the nodes establishes a common key with source and destination, the message will be encrypted and a random key is generated for both of nodes (source node and destination node), which can receive a pairwise key from the message.

The probability is explained as follows [11].

$P=1-\Pi_{i=0}^{s'-1}s-s'-i/s-i.$

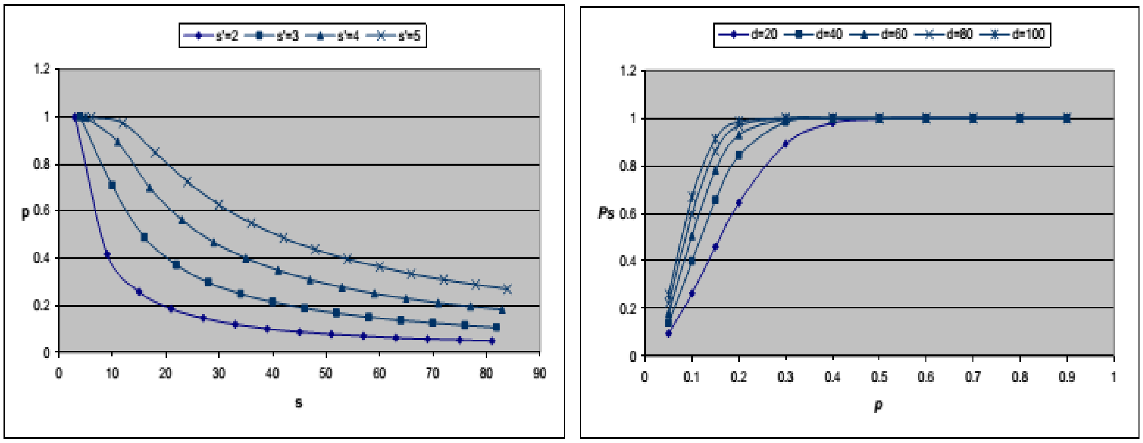
Figure 3a explains the relationship between p and s, and s' where the sensor node establishes the pairwise key, and if there is d neighbouring nodes the probability can be given by [11].

$P_s=1-(1-p)(1-p^2)^d.$

Figure 3(b) explains the relation between p, d and P_s. The probability of comprised nodes of the polynomial can be given by:

$P(i)=NC!/(NC-i)!(s'/s)^i(1-s'/s)^{NC-i}.$

Here's'/s gives the polynomial of choosing F.



(a) The probability p that two sensors share a polynomial v.s. the size s of the polynomial pool

(b) The probability P_s of establishing a pairwise key v.s. the probability p that two sensors share a polynomial

Fig 3: Probabilities of pairwise key establishment [11].

3.4.3 Grid Based Key Predistribution

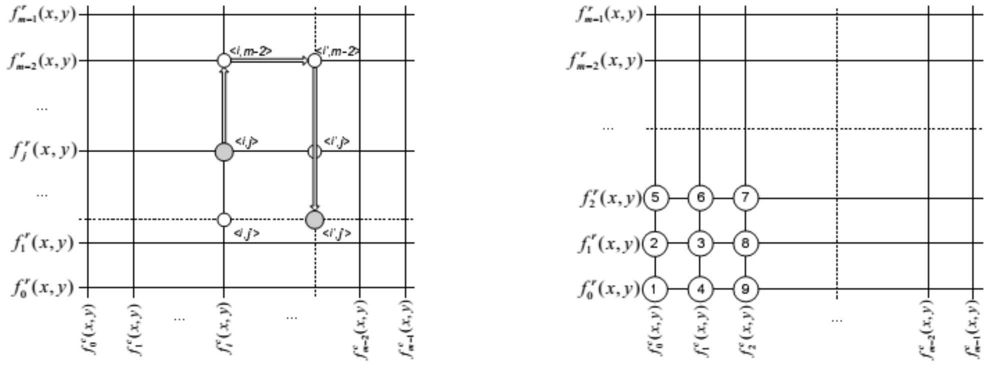
A grid based key predistribution scheme has several advantages [11]:

i) Sensors can establish a pairwise key if no compromised nodes are present and two sensors should always communicate.

ii) The resiliency is very high even when the sensors are compromised.

iii) A sensor will establish a pairwise key if the other node has a same polynomial; otherwise this will not occur.

In a grid based system, we construct a nxn grid with the set of 2n polynomials such that it is of the form $(F_i^c(x,y),F_i^r(x,y))$ [11] and i=0,1,…,m-1 with n=sqrt(M) where M is number of sensor nodes. The sensor will represent the unique point in the x-y coordinate, and distribute the polynomial $(F_i^c(x,y),F_i^r(x,y))$ to the sensor.



(a) The grid

(b) An example order of node assignment

Fig 4: Grid based Predistribution [11].

In a subset assignment; the setup server will generate a bivariate polynomial of t-degree over a finite field. When there is no intersection in the grid, the polynomial will be assigned to the node. Typically the polynomial can be represented as $F = (f_c^i(x,y),f_r^i(x,y)$) [11] where i=0,….m-1.

In polynomial share discovery; we check whether two column or rows will be equal then a pairwise key will be established such they have the polynomial shares.

For path discovery; if there exists a path between i and j such that there are one or more intermediate nodes, the node algorithm will be assigned. Fig 4(a) [11] explains in detail if there exists a path between the nodes, and then a pairwise key will be established.

In some situations, if the intermediate nodes are compromised or out of range, they will have chosen alternative key paths. For example in Fig 4(b) [11] we can establish the random pairwise key by using the nodes (i, m-2) and (i', m-2) for node (i, j) with (i’, j’).

4 Key Predistribution Scheme using Transversal Design

In order to predistribute the keys, Pattanayak and Majhi [12] proposed a scheme based on the transversal design. The algorithm for deriving the transversal design is described here. A transversal design [12] can be defined in terms of group divisible designs, with a group g^u and block size k where X is the finite cardinality g^u, H is partition of X in parts of G, and A is a set of subsets of X following the properties:

i) $H_i \cap A_i \leq 1$ for every $H_i \in H$ and $Ai \in A$.

ii) Each pair of elements of X from different groups occurs in exactly one block in A.

iii) A transversal design is a group divisible design of type n^k and block size k.

4.1 Algorithm

The following steps derive the transversal design [12]:

```
(1) Define X = {0, 1,..., n-1}x$Z_p$ where $Zp$ is the prime number used for modulo operation
        For 0≤x≤n-1, define $H_x=\{x\}\ x\ Z_{p.}$
            Define $H= \{H_x|0\leq x\leq n-1\}$.
             For every ordered pair (i, j) ∈ $Z_p\ x\ Z_p$.
            Define a block $A_{i,j}=\{(x, ix+j\ mod\ q)|\ 0\leq x\leq n-1\}$.
         Obtain A= $\{A_{i,j}|(i,j) \in Z_p\, x\ Z_p\}$.
```

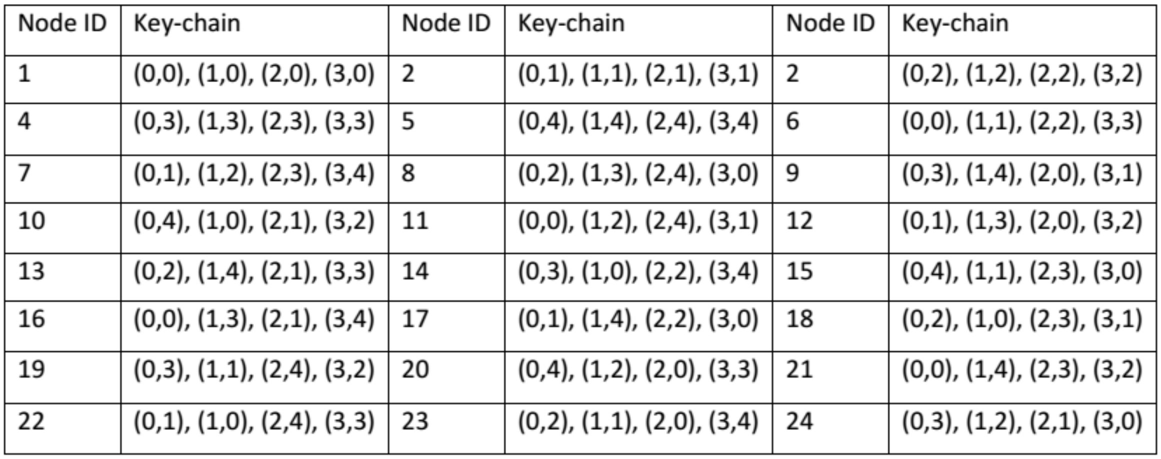
(X, H, A) is the transversal design TD (k,p) where k=G (groups) and p=B (blocks).

An example for transversal design is described here [12]:

Number of blocks=24.

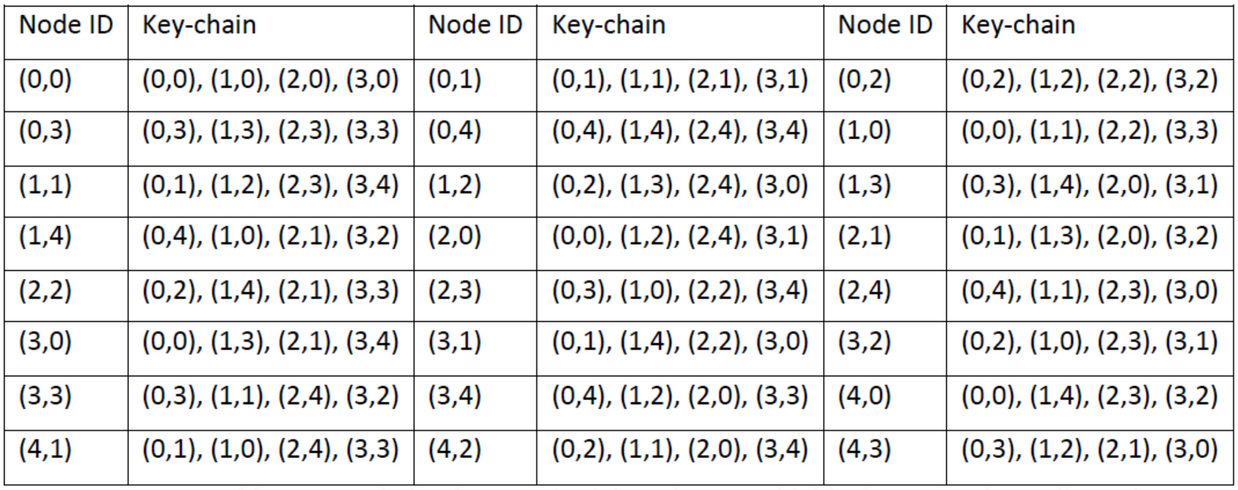
Block size=4.

Prime power =Z_p=5.



Node ID	Key-chain	Node ID	Key-chain	Node ID	Key-chain
1	(0,0), (1,0), (2,0), (3,0)	2	(0,1), (1,1), (2,1), (3,1)	2	(0,2), (1,2), (2,2), (3,2)
4	(0,3), (1,3), (2,3), (3,3)	5	(0,4), (1,4), (2,4), (3,4)	6	(0,0), (1,1), (2,2), (3,3)
7	(0,1), (1,2), (2,3), (3,4)	8	(0,2), (1,3), (2,4), (3,0)	9	(0,3), (1,4), (2,0), (3,1)
10	(0,4), (1,0), (2,1), (3,2)	11	(0,0), (1,2), (2,4), (3,1)	12	(0,1), (1,3), (2,0), (3,2)
13	(0,2), (1,4), (2,1), (3,3)	14	(0,3), (1,0), (2,2), (3,4)	15	(0,4), (1,1), (2,3), (3,0)
16	(0,0), (1,3), (2,1), (3,4)	17	(0,1), (1,4), (2,2), (3,0)	18	(0,2), (1,0), (2,3), (3,1)
19	(0,3), (1,1), (2,4), (3,2)	20	(0,4), (1,2), (2,0), (3,3)	21	(0,0), (1,4), (2,3), (3,2)
22	(0,1), (1,0), (2,4), (3,3)	23	(0,2), (1,1), (2,0), (3,4)	24	(0,3), (1,2), (2,1), (3,0)

Table 6: Key-chains for a TD (4, 5)-based KPS, with block ID are of one-dimension [12].



Node ID	Key-chain	Node ID	Key-chain	Node ID	Key-chain
(0,0)	(0,0), (1,0), (2,0), (3,0)	(0,1)	(0,1), (1,1), (2,1), (3,1)	(0,2)	(0,2), (1,2), (2,2), (3,2)
(0,3)	(0,3), (1,3), (2,3), (3,3)	(0,4)	(0,4), (1,4), (2,4), (3,4)	(1,0)	(0,0), (1,1), (2,2), (3,3)
(1,1)	(0,1), (1,2), (2,3), (3,4)	(1,2)	(0,2), (1,3), (2,4), (3,0)	(1,3)	(0,3), (1,4), (2,0), (3,1)
(1,4)	(0,4), (1,0), (2,1), (3,2)	(2,0)	(0,0), (1,2), (2,4), (3,1)	(2,1)	(0,1), (1,3), (2,0), (3,2)
(2,2)	(0,2), (1,4), (2,1), (3,3)	(2,3)	(0,3), (1,0), (2,2), (3,4)	(2,4)	(0,4), (1,1), (2,3), (3,0)
(3,0)	(0,0), (1,3), (2,1), (3,4)	(3,1)	(0,1), (1,4), (2,2), (3,0)	(3,2)	(0,2), (1,0), (2,3), (3,1)
(3,3)	(0,3), (1,1), (2,4), (3,2)	(3,4)	(0,4), (1,2), (2,0), (3,3)	(4,0)	(0,0), (1,4), (2,3), (3,2)
(4,1)	(0,1), (1,0), (2,4), (3,3)	(4,2)	(0,2), (1,1), (2,0), (3,4)	(4,3)	(0,3), (1,2), (2,1), (3,0)

Table 7: Key-chains for a TD (4, 5)-based KPS, with block ID of two-dimensions [12].

4.2 Previous Work Proposed on Merging the Blocks

In the scheme proposed by Chakraborthy [13], it was observed that the connectivity property (if nodes are compromised, the remaining nodes maintains all links in network) was increasing and was a hybrid scheme having factor z and formed the key-chains in factors of z. This is a heuristic method. To summarize, take the flag value and counter array so that it can store the blocks with factor z. Iterate the block by using the loop, then the first block is passed, which was not used and which has no common key (such that there is no sharing between the nodes). When the blocks are found and they have been marked as used, and check the flag value as false, if there is no such block. After this, we increment the block by 1 and end the loop. We check the properties that are to be satisfied, check the node value for whether it has any interconnectivity property, and merge the blocks. After this step, we calculate the adjacency matrix and increment it 1000 times such that it passes the iteration every time. This ends the algorithm. The move routine will choose a random pair of nodes that share a maximum common key, and another pair that does not. Then we see the intersection element after the first the pair is passed before repeating this process.

4.3 Algorithm analysis

The main feature to discuss is the interconnectivity property (if nodes are compromised, the remaining nodes maintain all links in network). If the interconnectivity property is satisfied then it is a deterministic approach, otherwise it can be a heuristic approach. In the deterministic approach we take the factor z for merging the blocks that forms the key chains to improve the connectivity.

In tables 6 and 7 (constructed from and algorithm listed in [12]), we can observe the consecutive blocks that start from 0, and do not have common keys. The consecutive blocks will be divided from 0 to 25 such that first group forms the blocks from 0 to 4 the next group from 5 to 9 and so on, with the condition $0<x, y<p-1$. Then the blocks are merged consecutively and we take z/p value by having the following conditions:

i) If $p\%z=0$, then $z=x$ and $p=x^y$ and then the nodes are merged and assigned to sensor nodes.

ii) If $p\%z=1$, check if whether there is any common key. If it does not exist, merge the first and second block which is equal to p. In this step we also try the p/z value of remaining blocks, which are chosen from each p block and have no common key.

iii) If p %z =2, then we form the consecutive blocks and check if there is no inter-group common key. Next step would be merging the p/z blocks which will result in p blocks and among every two groups we combine z blocks to form p/z merged blocks.

iv) For p % z >2, we see the logic of combining the blocks with no common key, the remaining blocks are merged which will increase the number of nodes.

For further analysis of the algorithm see [12]. Now let us illustrate the above case with an example with values p=5 and z=3and N=25, K=4 in table 6 and give the blocks in two dimensions and merge each block with other block such that they have common key until we have no block left.

4.4 Key Establishment using the Transversal Design

In the paper mentioned by Pattanayak and Majhi [12], the groups and blocks are two dimensional objects formed by taking the keys as the first component which is in the form (i, x), and repeating the cycle from 0 to 4 of (i,x). In brief, the difference between the first column first group and second column second group is 1; in the third step it increases by 2, and the first column third group is also 2. Here the same pattern is repeated, and the algorithm is presented as mentioned in [12]:

The first step is to take the key-id in one dimension. Assign the value x to key-id, then take the modulus of x over p and assign to y (x%p). Divide the value x by p and name it as z (x/p). After this step we will assign the array a[x] to y variable. Take a variable b and iterate a loop which gives the first group of keys. We follow the same iteration for second column second group and third column third group by iterating the loop until z value for each column, and if we have the two dimensional nodes simply we form the key establishment between the nodes as (i,x), and stop the key establishment. In this way we can establish the common key between x and y such that we calculate the keys for x and y.

4.5 Probability of Key predistribution using CID design

The probability Ni and Nj nodes such that they have a common key and are within range is [5]:

$P=k*(r-1)/b-1$.

The resiliency is measured as follows; when an attacker is comprimising s nodes $V(s) =1-(1-r-2/b-2)^s$.

5 Algorithm for Relationships Between Various Combinatorial Designs

The numerical results generated from the program in appendix A are based on the relationship between orthogonal latin squares and projective planes. We will construct the affine plane based on the orthogonal latin squares and then we will generate the projective plane. With the program results I am able to understand the concept of block designs and projective planes relationships easily. I have also improved my mathematical knowledge by covering the in-depth concept of latin squares. I have constructed the programs in fortran95 language, given in appendix A. We first take the latin square as input and based on indices we generate the affine plane. The second program generates the structure of mutually orthogonal latin squares of order 5 by considering the primitive root of a number. Taking this primitive root we will finid the galois field values and performing the operation we get the MOLS.

6 Conclusion

To summarize, the transversal design is useful for understanding the key predistribution schemes. Implementation on the sensor involves checking if the nodes within the communication range will have a common key. It also specifies the metrics for understanding them. This project also dealt with other algorithms, so that previous schemes are explained, and how merits and demerits are specified. Finally, the key establishment phases are discussed to achieve this goal. This project also involved the computational results in affine planes and mutually orthogonal latin squares in (appendix A) . Future work would be to implement the key predistribution schemes using other combinatorial designs, and help to improve efficiency, along with other metrics.

Appendix A: Computational Program and Results

Theorem: Suppose there exists an affine plane of order n, then there exists n-1 mutually orthogonal latin squares of order n.

Proof:

We can construct n-1 MOLS by using the latin squares L_1, L_2, L_3,…..L_{N-1} which forms the intersection in affine planes as in [4].

We can define the example as in [4].

X = {1,2,3,4,5,6,7,8,9}

We can name the blocks as follows:

B={123,147,159,168,456,258,267,249,789,369,348,357}

Then the latin squares are as follows:

L1	1	3	2
	2	1	3
	3	2	1

L2	1	3	2
	3	2	1
	2	1	3

Suppose we combine latin squares with affine planes can be defined as follows [4]:

A(1,1)={(1,1)(1,2)(1,3)}

A(1,2)={(2,1)(2,2)(2,3)}

A(1,3)={(3,1)(3,2)(3,3)}

A(3,1)={(1,1)(2,3)(3,2)}

A(2,1)={(1,1)(2,1)(3,1)}

A(2,2)={(1,2)(2,2)(3,2)}

A(2,3)={(1,3)(2,3)(3,3)}

A(4,1)={(3,1)(1,2)(2,3)}

A(3,2)={(2,1)(1,2),(3,3)}

A(3,3)={(3,1)(2,2)(1,3)}

A(4,2)={(1,1)(2,2)(3,3)}

A(4,3)={(2,1)(3,2)(1,3)}

Problem : 3 mutually orthogonal latin squares of order 4

1 2 3 4	1 2 3 4	1 2 3 4
2 1 4 3	3 4 1 2	4 3 2 1
3 4 1 2	4 3 2 1	2 1 4 3
4 3 2 1	2 1 4 3	3 4 1 2

Row by column value is represented as:

(1,1),(1,2),(1,3),(1,4)

(2,1),(2,2),(2,3),(2,4)

(3,1),(3,2),(3,3),(3,4)

(4,1),(4,2),(4,3),(4,4)

2nd MOLS (1,1),(2,1),(3,1),(4,1) (1,2),(2,2),(3,2),(4,2)

(1,3),(2,3),(3,3),(4,3) (1,4),(2,4),(3,4),(4,4)

3rd MOLS (1,1),(2,2),(3,3),(4,4) (2,1),(1,2),(3,4),(4,3)

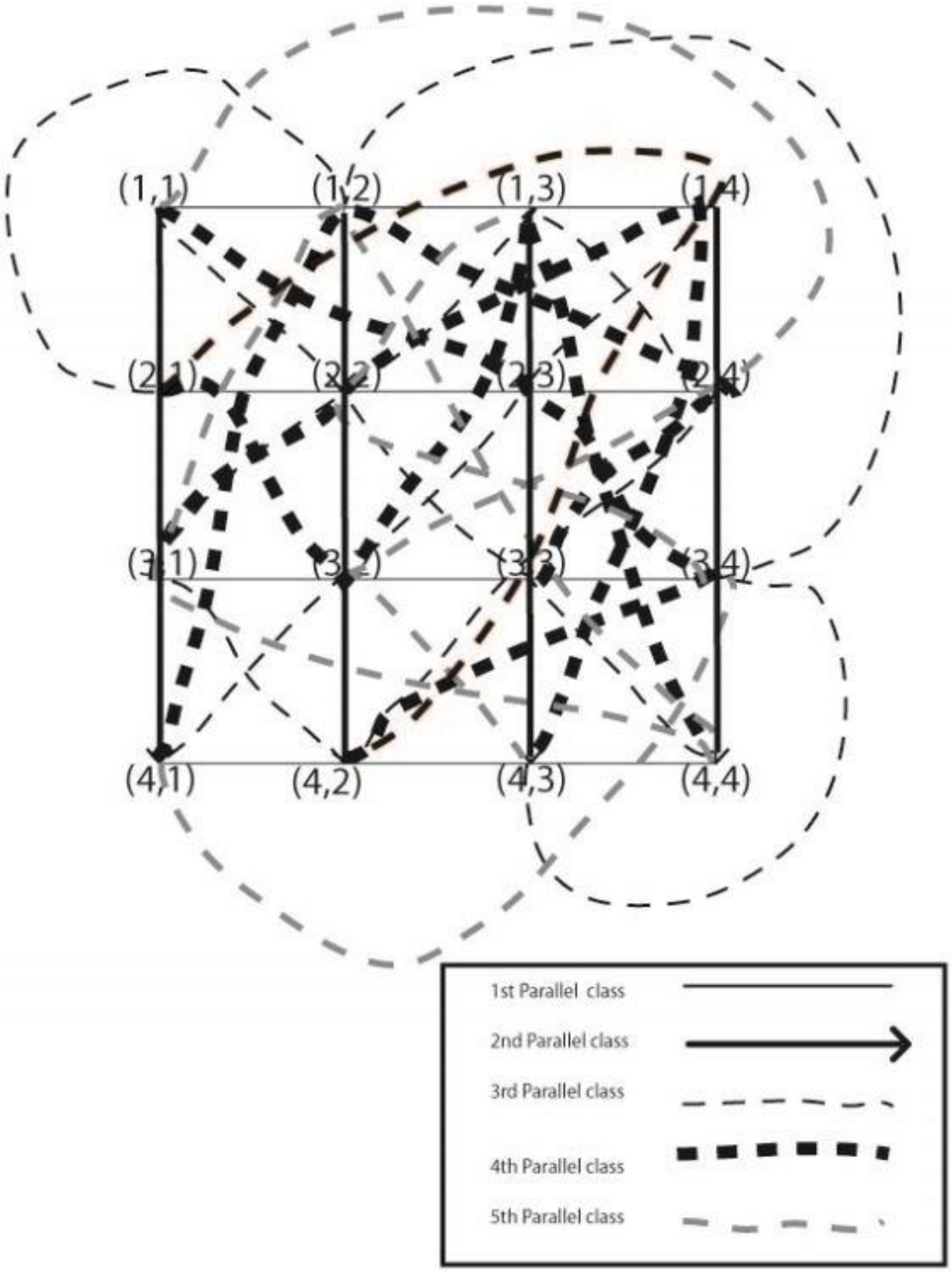
(3,1),(1,3),(2,4),(4,2) (4,1),(1,4),(2,3),(3,2)

4th MOLS (4,1),(1,2),(2,4),(3,3) (1,1),(2,3),(3,4),(4,2)

(2,1),(3,2),(1,3),(4,4) (3,1),(2,2),(1,4),(4,3)

5th MOLS (3,1),(4,4),(1,2),(2,3) (4,1),(3,4),(2,2),(1,3)

(1,1),(2,4),(3,2),(4,3) (2,1),(1,4),(3,3),(4,2)


(1,1)
(1,2)
(1,3)
(1,4)
(2,1)
(2,2)
(2,3)
(2,4)
(3,1)
(3,2)
(3,3)
(3,4)
(4,1)
(4,2)
(4,3)
(4,4)
1st Parallel class
2nd Parallel class
3rd Parallel class
4th Parallel class
5th Parallel class

Code for Generating Affine plane from Orthogonal Latin Squares

Here the construction is based on the latin squares as input and generating vertical and horizontal planes and then constructing the other two latin squares using the indexes in each matrix.

```
program Squares implicit none
integer i, j, k, L1(3,3), L2(3,3), x
L1(1,1)=1;
L1(1,2)=3;
L1(1,3)=2;
L1(2,1)=2;
L1(2,2)=1;
L1(2,3)=3;
L1(3,1)=3;
L1(3,2)=2;
L1(3,3)=1;
L2(1,1)=1;
L2(1,2)=3;
L2(1,3)=2;
L2(2,1)=3;
L2(2,2)=2;
L2(2,3)=1;
L2(3,1)=2;
L2(3,2)=1;
L2(3,3)=3;
```

```
write(*,2000) ((L1(i,j),j=1,3),i=1,3) write(*,2001) ((L2(i,j),j=1,3),i=1,3)

2000 format ( /, 'L1(i,j) = ',/ (3i3))

2001 format ( /, 'L2(i,j) = ',/ (3i3))

do x=1,2

do k=1,3 print*, 'A', x, k do i=1,3

do j=1,3

          if (x==1) then

                    if (L1(i,j)==k) then

                    print*, i, j

                    endif

          endif

           if (x==2) then

                    if (L2(i,j)==k) then

                    print*, i, j

                    endif

          endif

                    enddo

          enddo

          enddo

           enddo

do k=1,3

          print*, 'A', 3, k do j=1,3
```

```
        print*, k, j
        enddo
        enddo
        do k=1,3
        print*, 'A', 4, k do i=1,3
        print*, i, k enddo
        enddo
        stop
        end program Squares
```

Output

```
Compiling the source code....
$gfortran -std=f95 main.f95 -o demo 2>&1
Executing the program....
$demo L1(i,j) =
1 3     2
2 1     3
3 2     1
 L2(i,j) =
1 3     2
3 2     1
2 1     3
A       1       1
```

	1	1
	2	2
	3	3
A	1	2
	1	3
	2	1
	3	2
A	1	3
	1	2
	2	3
	3	1
A	2	1
	1	1
	2	3
	3	2
A	2	2
	1	3
	2	2
	3	1
A	2	3
	1	2
	2	1
	3	3

A	3	1
	1	1
	1	2
	1	3
A	3	2
	2	1
	2	2
	2	3
A	3	3
	3	1
	3	2
	3	3
A	4	1
	1	1

Appendix B: Construction of Mutually Orthogonal Latin Squares (MOLS)

In order to construct the MOLS we have find the values in Galois fields as GF(q)={ λ_1, λ_2, λ_3,…..}

Then the major operation to be performed is $\lambda_i \lambda_k + \lambda_j$.

For example the MOLS of order 5 can be constructed as follows:

5 have the primitive root as 2.

Construction:

2^1=2 mod5 =2.

2^2=4 mod 5=4.

2^3=8 mod 5=3.

2^4=16 mod 5=1.

λ_1=2, λ_2=4, λ_3=3, λ_4=1.

(1,1)=2x2+2=6 mod 5=1:(2,1)=4x2+2=10 mod 5=0:(3,1)=3x2+2=8 mod 5=3:(4,1)=4:(5,1)=2.

(1,2)=2x2+4=8 mod 5=3: (2,2)=4x2+4=12 mod 5=2:(3,2)=3x2+4=10 mod 5=0:(4,2)=1:(5,1)=4.

(1,3)=2x2+3=7 mod 5=2: (2,3)=4x2+3=11 mod 5=1:(3,3)=3x2+3=9 mod 5=4:(4,3)=0:(5,1)=3.

(1,4)=2x2+1=5 mod 5=0: (2,4)=4x2+1=9 mod 5=4:(3,4)=3x2+1=7 mod 5=2:(4,4)=3:(5,1)=1.

(1,5)=2x2+0=4 mod 5=4: (2,5)=4x2+0=8 mod 5=3:(3,5)=3x2+0=6 mod 5=1:(4,1)=2:(5,1)=0.

1	3	2	0	4
0	2	1	4	3
3	0	4	2	1
4	1	0	3	2
2	4	3	1	0

0	2	1	4	3
3	0	4	2	1
4	1	0	3	2
2	4	3	1	0
1	3	2	0	4

3	0	4	2	1
4	1	0	3	2
2	4	3	1	0
1	3	2	0	4
0	2	1	4	3

4	1	0	3	2
2	4	3	1	0
1	3	2	0	4
0	2	1	4	3
3	0	4	2	1

Code for Generating MOLS:

This program mainly explains how to derive the MOLS of order 5. It checks the number is prime or not after that it takes the matrix from the primitive root given and generates the λ values and then processing the matrices will give the final result.

```
program MOLS
   integer Ak(5,5), l(5), i, j, k, ord, lk
   lk=2
 print *, 'Input a number: '
  read *, ord
  code = is_prime(ord) ---------------------Determining the prime number
  if code == 0 then
       do i=1,ord-1
        l(i) = mod(lk**i, ord)-----------------------Generate the primitive root
    enddo
    l(ord)=0
         do i=1,ord
         do j=1,ord
            Ak(i,j)= mod(l(i)*lk +l(j),5)---------------take the matrix
         enddo
    enddo
    write(*,2001) l
    write(*,2000) ((Ak(i,j),j=1,ord),i=1,ord)
        2000 format ( /, 'Ak(i,j) = ',/ (5i5))
    2001 format (/, 'lambda = ',/,(15i5))take the lambda values
        !do k=1,ord
 do i=1,ord
 if ((ord-i) /= lk) then
                    do j=1,ord
                        Ak(i,j)= mod(l(1)*(ord-i) +l(j),5)
                    enddo
                endif
            enddo
        !enddo-----------------------------------------do the processing for matrices
        write(*,2000) ((Ak(i,j),j=1,ord),i=1,ord)
      stop
 else
 stop
 end if write the down matrices
end program MOLS
```

output:

Compiling the source code....

```
$gfortran -std=f95 main.f95 -o demo 2>&1
```

Executing the program....

```
$demo

 Input a number:

lambda =
    2    4    3    1    0
```

```
Ak(i,j) =
   1   3   2   0   4
   0   2   1   4   3
   3   0   4   2   1
   4   1   0   3   2
   2   4   3   1   0

Ak(i,j) =
   0   2   1   4   3
   3   0   4   2   1
   4   1   0   3   2
   1   3   2   0   4
   2   4   3   1   0

Ak(i,j) =
   3   0   4   2   1
   4   1   0   3   2
   1   3   2   0   4
   0   2   1   4   3
   2   4   3   1   0

Ak(i,j) =
   4   1   0   3   2
   1   3   2   0   4
   0   2   1   4   3
   3   0   4   2   1
   2   4   3   1   0
```

Appendix C: Program for Generating the Key Predistribution using Latin Square

This program takes the q as primitive root value. The primitive root will process and give q-1 orthogonal latin squares converted to affine planes. When affine planes are marked into projective plane will give the predistribution schemes based on an equation ax+by+cz = 0. A key (x, y, z) is distributed to node (a, b, c).

```
#include <stdio.h>
#include <stdlib.h>
#define q 4
/**Distribution of keys following KPS using PG(2, q)**/
int main(){
        int i, x, y, b, c;
        int* phi_q = (int*) malloc(sizeof(int) * 10);
        for (i=0; i<q; i++){
                phi_q[i] = i;
        }
        printf( "Node Index || Keys\n", b,c );
        for (b=phi_q[0]; b<=phi_q[q-1]; b++ ) {
                for (c=phi_q[0]; c<=phi_q[q-1]; c++ ) {
//node
                printf( "(1,%d,%d) ||", b,c );
//key
        for (y=phi_q[0]; y<=phi_q[q-1]; y++ ) {
        x=((-(c+b*y)) % q + q)%q; //positive modulo
        printf( " (%d,%d,1) ", x,y);
}
        printf( " (%d,1,0)\n", (-b % q + q) %q);
}
}
for (c=phi_q[0]; c<=phi_q[q-1]; c++ ) {//node
        printf( "(0,1,%d) ||", c % q );
//key
        for (x=phi_q[0]; x<=phi_q[q-1]; x++ ) {
        printf( " (%d,%d,1) ", x, (-c % q + q)%q );
}
        printf( " (1,0,0)\n");
}
//node
        printf( "(0,0,1) ||");
//key
for (x=phi_q[0]; x<=phi_q[q-1]; x++ ) {
        printf( " (%d,1,0) ", x);
}
        printf( " (1,0,0)\n", (-b % q + q) %q);
return 0;
}
```

Output:

```
Executing the program....
$demo
Node Index || Keys
(1,0,0) || (0,0,1) (0,1,1) (0,2,1) (0,3,1) (0,1,0)
(1,0,1) || (3,0,1) (3,1,1) (3,2,1) (3,3,1) (0,1,0)
(1,0,2) || (2,0,1) (2,1,1) (2,2,1) (2,3,1) (0,1,0)
(1,0,3) || (1,0,1) (1,1,1) (1,2,1) (1,3,1) (0,1,0)
(1,1,0) || (0,0,1) (3,1,1) (2,2,1) (1,3,1) (3,1,0)
(1,1,1) || (3,0,1) (2,1,1) (1,2,1) (0,3,1) (3,1,0)
(1,1,2) || (2,0,1) (1,1,1) (0,2,1) (3,3,1) (3,1,0)
(1,1,3) || (1,0,1) (0,1,1) (3,2,1) (2,3,1) (3,1,0)
(1,2,0) || (0,0,1) (2,1,1) (0,2,1) (2,3,1) (2,1,0)
(1,2,1) || (3,0,1) (1,1,1) (3,2,1) (1,3,1) (2,1,0)
(1,2,2) || (2,0,1) (0,1,1) (2,2,1) (0,3,1) (2,1,0)
(1,2,3) || (1,0,1) (3,1,1) (1,2,1) (3,3,1) (2,1,0)
(1,3,0) || (0,0,1) (1,1,1) (2,2,1) (3,3,1) (1,1,0)
(1,3,1) || (3,0,1) (0,1,1) (1,2,1) (2,3,1) (1,1,0)
(1,3,2) || (2,0,1) (3,1,1) (0,2,1) (1,3,1) (1,1,0)
(1,3,3) || (1,0,1) (2,1,1) (3,2,1) (0,3,1) (1,1,0)
(0,1,0) || (0,0,1) (1,0,1) (2,0,1) (3,0,1) (1,0,0)
(0,1,1) || (0,3,1) (1,3,1) (2,3,1) (3,3,1) (1,0,0)
(0,1,2) || (0,2,1) (1,2,1) (2,2,1) (3,2,1) (1,0,0)
(0,1,3) || (0,1,1) (1,1,1) (2,1,1) (3,1,1) (1,0,0)(0,0,1) || (0,1,0) (1,1,0) (2,1,0) (3,1,0) (1,0,0)
```

Appendix D: Hadamard matrix and BIBD

A matrix of order *nxn* such that every entry in the matrix should be equal to 1 or -1 in rows and columns. It is represented as H. The condition for hadamard matrix is $HH^T= nI_n$, where I is the identity of order n:

1	1
1	-1

Table for hadamard matrix.

Suppose we multiply with 1 and -1, we also derive the hadamard matrix again. This is called standardization.

Equivalence of a Hadamard Matrix and BIBD

Suppose there is a hadamard matrix of order 4m, then there exists a BIBD such that it forms (v, k, λ) which is also a symmetric BIBD.

For example:

Suppose with BIBD (4, 3, 1) and the blocks (123, 234, 341, 412)

Then the matrix can be formed as we substitute 0 with -1, and add 1 in a row and a column that gives the hadamard matrix is described in following table:

1	-1	1	1	1
1	1	-1	1	1
1	1	1	-1	1
-1	1	1	1	1
1	1	1	1	1

Relationship between hadmard matrix and BIBD.

Bibliography

[1] S. A. Camtepe & B. Yener, “Combinatorial Design of Key Distribution Mechanisms for Wireless Sensor Networks,” IEEE/ACM Transactions on Networking, Vol. 15, No. 2, April 2007.

[2] Harjot Bawa, Singh and Kumar, “An Efficient Novel Key management scheme using N choose K algorithm for Wireless Sensor Networks,”India, (IJCNC) Vol.4, No.6, November 2012.

[3] I. Anderson, “Combinatorial Designs: Construction Methods”, Ellis Horwood Limited, 1990.

[4] D. R. Stinson, Combinatorial Designs: “Constructions and Analysis. Springer-Verlag,”New York (2004).

[5] Anupam Pattanayak, B. Majhi, “Key Predistribution Schemes in Distributed Wireless Sensor Network using Combinatorial Designs Revisited,” Cryptology eprint Archive, Report 2009.

[6] R. Blom, “An optimal class of symmetric key generation systems,” EUROCRYPT 84, 1985.

[7] W. Du, J. Deng, Y. S. Han, S. Chen, P. K. Varshney, “A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge,” INFOCOM, 2004.

[8] H. Chan, A. Perrig and D. Song, “Random Key Predistribution Schemes for Sensor Networks,” In 2003 IEEE Symposium on Research in Security and Privacy, 2003.

[9] S. Slijepcevic, M. Potkonjak, V. Tsiatsis, S. Zimbeck, M. B. Srivastava, “On communication Security in Wireless Ad-Hoc Sensor Network,” Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE’02), 2002.

[10] L. Eschenauer, V. D. Gligor, “A key-management scheme for distributed sensor networks,” Proceedings of the 9th ACM conference on Computer and communications security, 2002.

[11] D. Liu, P. Ning, “Establishing pairwise keys in distributed sensor networks,” Proceedings of the 10th ACM conference on Computer and communication security, 2003.

[12] Anupam Pattanayak, B. Majhi, “A Deterministic Approach of Merging of Blocks in Transversal Design based Key Predistribution,” India, Cryptology eprint Archive, Report 2009.

[13] D Chakrabarti, S Maitra, B Roy, “A key pre-distribution scheme for wireless sensor networks: merging blocks in combinatorial design,”International Journal of Information Security, Vol. 5, No. 2, pp 105-114, 2006.