A Deterministic Approach of Merging of Blocks in Transversal Design based Key Predistribution

Anupam Pattanayak, B. Majhi

Computer Science & Engineering Department,
National Institute of Technology, Rourkela, India – 769008
{**anupam.pk@gmail.com, bmajhi@nitrkl.ac.in**}

Abstract – Transversal Design is a well known combinatorial design that has been used in deterministic key predistribution scheme. Merging of blocks in a design sometimes helps to obtain a key predistribution scheme with better performance. A deterministic merging strategy to merge the blocks has been discussed. Also, a simple key establishment method for transversal design based key predistribution scheme has been discussed.

Keywords: Key predistribution, Transversal Design, Deterministic Merging, Key Establishment

1. Introduction

Distributed Wireless Sensor Network consists of several tiny sensors. A sensor has very limited resources such as processor capability, communication range, memory capacity, and battery power. In many cases these sensors are deployed without having any control or sometimes with partial control on the locations of these sensors. It may be sometimes desired that the communication among these sensors be secured. In that case we need a particular cryptographic technique to be used. Due to limited resources of sensor nodes, the use of private key cryptography is advocated mostly. With the amazing advancement of hardware, it would be a reality in near future that a sensor network would use public key cryptography without compromising much on their resource front. As we just mentioned, private key cryptography (or symmetric key cryptography) is mostly used in sensor network. To further reduce load of symmetric key generation, what is done is predistribute key every sensor node before its deployment. To make this key predistribution further effective, generally a set of keys are pre distributed to the sensor nodes, so that they can communicate directly with more number of sensor nodes. A sensor node has a communication range within which it can communicate with other node. Now since sensor nodes are generally deployed without any control on their deployment locations, initially a node is unaware of the identity of its neighbor nodes. So they broadcast their identification and by some deterministic means they derive the common key of each other, if they share any such key.

2. Combinatorial Design

The origin of combinatorial design lies in the statistical field design of experiments. But it has found wide application in various branches of science and engineering. It has been vastly used in coding theory, and various branches of cryptography such as Boolean function, authentication code, Key predistribution etc [1]. In the following we first give a brief introduction to combinatorial design and then describe about *transversal design*.

2.1 Introduction to Combinatorial Design

Combinatorial Design is the study of set of subsets with various constraints. These constraints are of numerous types. Depending upon the type of these constraints on the set of subsets the design is classified. Let us describe some basic terminologies in combinatorial design. Elements of the universal set S are called *treatments* or *varieties*. Subsets considered of this set S are called *blocks*. If the length of each block is same and the each element appears same number of times in the blocks then this design (set of blocks) is *regular*. The number of times a treatment appears in the design is called *degree* of elements. Consider a set S = {1, 2, 3, 4, 5, 6}. Now suppose our family of subsets are {{1,2,3}, {2,4,5}, {3,5,6}, {1,4,6}}. For our convenience we write the subsets as follows {123, 245, 356, 146}. This family of subsets is an example of regular design where v = |S| = 6, number of blocks b = 4, degree r = 2, block size b = 3. A regular design is denoted by (v, b, r, k)-design. For a regular design the following relationship holds:

$$v.r = b.k \longrightarrow (1)$$

The number of times a pair of treatments (x, y) appears together in the blocks is denoted by λ_{xy} and is called covalency of x and y. If λ_{xy} is same for any x, y $\in$ S, then the design is called (v, b, r, k, λ)-design. For a (v, b, r, k, λ)-design the following condition holds:

$$r.(k-1) = \lambda.(v-1) \longrightarrow (2)$$

If all the treatment appears in a block then that block is *complete*. If all the blocks in a regular design are complete then the design is called *complete design.* Complete designs are of little interest unless some further structure is imposed (such as in *Latin Square*). In a design, if at least one block is incomplete then that design is an *incomplete design*. If v = b, the design is called *symmetric*.

2.2 Transversal Design

A *Group-divisible Design* of type g^u and block size k is a triplet (X, H, A), where X is a finite set of cardinality gu, H is a partition of X into u parts (called groups) of size g, and A is a set of subsets of X (called blocks), that satisfy following properties [2]:

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

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

A *Transversal Design* TD(k,n) is a group-divisible design of type n^k and block size k. So $H_i \cap A_i = 1 \; \forall \; H_i \in H, A_i \in A$.

3. Key Predistribution Scheme based on Transversal Design

The first key predistribution scheme that used combinatorial design is by Camtepe and Yener [3]. They have used projective plane and generalized quadrangle to predistribute keys. Key predistribution scheme proposed by Lee and Stinson uses *Transversal design* [4]. Transversal Design is a special kind of group-divisible design. Here, any two distinct blocks intersect in zero or one point. A very good article by Martin on applicability of combinatorial designs in key predistribution can be found in [5]. To get overview of all combinatorial design based key predistribution schemes one can refer the article by Pattanayak and Majhi [6]. In the below we write the algorithm for obtaining a Transversal Design.

3.1 Algorithm To Derive TD(k, p)

We write the algorithm to construct transversal design as given in [4].

Step 1. Define $X = \{ 0, 1, \ldots , k-1\} \times Z_p$.

Step 2. For $0 \leq x \leq k-1$, define $H_x = \{ x \} \times Z_p$.

Step 3. Define $H = \{ H_x \mid 0 \leq x \leq k-1 \}$.

Step 4. For every ordered pair $(i, j) \in Z_p \times Z_p$,

define a block $A_{i,j} = \{ (x, ix+j \bmod q) \mid 0 \leq x \leq k-1 \}$.

Step 5. Obtain $A = \{ A_{i,j} \mid (i, j) \in Z_p \times Z_p \}$.

Step 6. (X, H, A) is a TD(k, p).

3.2 Example of Transversal Design based KPS

To understand the above algorithm, we give the following example showing how do the blocks are formed. In this example number of blocks = 24, block size = 4, a prime power = 5. Now the blocks are constructed as below:

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 1. Key-chains for a TD(4,5)-based KPS, with block ID are of one-dimension.

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 2. Key-chains for a TD(4,5)-based KPS, with block ID are of two-dimension.

4. Previous Work on Merging of Blocks

In KPS based on transversal design, common key between any two nodes is atmost one. Chakraborty et al observed this and thought of increasing this connectivity. For this they devised a hybrid scheme based on transversal design [7]. In this scheme a merging factor z is decided. And z many blocks are merged to form a merged-block and this forms the key-chain for a sensor node. In this way not only they increased the connectivity provided by the KPS but also improved the resiliency offered. Below we present their heuristic algorithm for merging of blocks which works better than the scenario when blocks are merged randomly [7].

1. flag = true; count = 0; all the blocks are marked as unused;

2. an array node[..] is available, where each element of the array can also store z many blocks;

3. while(flag) {

    (a) choose a random block, mark it as used and put it in node[count];

    (b) for(i=1; i<z; i=i+1) {

        i. search all the unused blocks in random fashion and put the first available on one in *node[count]* which has no common key with the existing blocks already in *node[count]*;

        ii. mark this block as used;

        iii. if such a block is not available then break the for loop and assign flag = false;

    (c) } (end for)

    (d) if flag = true then count = count + 1;

4. } (end while)

5. report that count many nodes are formed such that there is no intra node connectivity;

6. for rest of the (r^2 – count.z) many blocks, merge z blocks randomly to form a node (they may have intra-node connectivity) to get ($\left\lfloor \frac{r^2}{z} \right\rfloor - count$) many extra nodes. This constitutes the initial configuration;

7. calculate the adjacency matrix;

8. make 1000 **moves** in succession, choose the one that gives rise to the maximum increase in connectivity and make the corresponding change in the configuration. Call it an *iteration*;

9. perform 1000 such iteration;

10. end;

The subroutine ***move*** has been defined as follows:

1. From the list of pair of nodes sharing maximum number of common keys, select one pair of nodes randomly. Call them a and b;
2. From the list of pair of nodes sharing no common key, select one pair of nodes randomly. Call them c and d;
3. Select one block each from a and b, and remove them such that the removed blocks intersect each other and a and b are still connected upon their removal. Let the removed blocks be α and β respectively.

4. Select one block each from a and b. Let the removed blocks be γ and δ respectively.
5. Put γ in a, δ in b, Put α in c, β in d.
6. Undo the above changes.

5. Our Approach for Merging Blocks

The above merging approach is heuristic, and not deterministic. The way blocks are chosen for merging are initially random with the condition that there should be no common key among the merged blocks. When one cannot proceed with merging with this condition of no inter-node connectivity, but still nodes are remaining for merging, the above said approach merges the remaining node without satisfying the inter-node connectivity. Then it applies mutation to the merged blocks to bring inter-node connectivity close to minimum.

To merge the blocks to form key-chain we first need to decide the merging factor, z. We should choose the merging factor in such a way that the increased load on memory of sensor node to store enlarged key-chain is not too high as well as the connectivity of the overall network improves considerably.

Recall that p is the prime power chosen for the transversal design. We observe that every consecutive p blocks, starting from block zero, don't have any common key among themselves. One can easily verify this observation in Table 1. Whenever we will be referring 'consecutive blocks of p blocks' or 'p consecutive blocks', we must remember that we have started with the first block only. For example if p=5, then we will have a transversal design with 25 blocks. So first group of consecutive blocks are block 0 – block 4, next group of consecutive blocks are block 5 – block 9, and so on. So in other words whenever we are referring the word 'consecutive blocks' or 'group of consecutive blocks', the condition is that block-id % p must be zero. If the block id is of the form (x, y) where $0 \leq x, y \leq p-1$, then the condition is that y must be zero of a block for being the starting block of consecutive p blocks.

we merge consecutive blocks (merging factor z) taken from consecutive p blocks. but if z<p then z%p blocks will be left out of merging consideration, and we get z/p (integer division) number of merged blocks. We have to consider different scenarios.

1. If p%z=0 (this happens when $p=x^y$ where x is a prime, and z=x) then we can simply take every consecutive group of z blocks and then merge them to form merged-blocks and assign to sensor nodes.

2. p%z=1. We know, among p consecutive blocks (if we start from the first block itself) there will be no common key. So we try to form p/z number of merged-blocks in such a way that remaining blocks, one from every group of p blocks will not have any common key. Say there are x such remaining blocks. In reality, this x is always equal to p. Then we can form x/z number of merged blocks with no common key. If we want to eliminate possibility of having common

key or a node with less than z*k number of keys then we will stop here only, and do not try to use the remaining blocks.

3. p%z=2. In this case from every group of p consecutive blocks we will have two remaining blocks. So we first take two groups of p blocks from all the group of consecutive p blocks such that there is no inter-group common key. After this, we take the remaining p/z blocks from each group and merge them to form the p.(p/z) merged blocks. Among each of the two groups of p blocks which were taken out at first, we continue with merging any z number of blocks to form p/z merged-blocks. Although there is a possibility of still remaining more than z number of blocks in total, our observation suggests that they always have common key among them. That is, we can not proceed to have another z number of blocks with no common key. So we can either leave those blocks or to allow few merged-blocks with inter-block common key.

4. for p%z >2 we can proceed with the same logic as above but, our observation suggests that in that cases the number of nodes that will be remaining after taking all merged-blocks will be little bit higher. So either we have to compromise with few merged-blocks with inter-block common key, or just leave those blocks unused which may turn little bit disadvantageous.

Now we formally present the algorithm for merging of blocks:

1. Start.
2. If $p=x^y$ and p%z=0 (i.e., z=x) then
    a. For i=0;i<p;i++
        i. For j=0;j<p/z;j++
            1. Take any z number blocks from the i-th group og consecutive blocks and form a merged-block.

        [end of inner-for loop]

    [end of outer-for loop]

3. If p%z > 0 then
    a. For i=0;i<p%z;i++
        i. Take a counter j and initialize with any value between 0 and p; i.e., $0 \leq j \leq p-1$.
        ii. For k=0;k<p;k++
            1. Choose (j+1)-th block from the k-th consecutive blocks. If this (j+1)-th block is the last block in this group then in the next iteration we will select 0-th block from next group instead of (j+1)-th block.

        [end of inner-for loop]

    [end of outer-for loop]

b. Out of these each group of p blocks, form p/z merged-blocks, with each merged-block is made following the next step.
c. By taking any z blocks together and merging them to get a merged-block with no inter-block common key.
d. for i=0;i<p;i++

    i. From each of the group of consecutive blocks choose any unused z blocks and merge them to form a merged-block with no inter-block common key.

    [end of for loop]

e. To deal with remaining blocks that are still not used for merging, either compromise to have merged-blocks with common inter-block key, or simply scrap these blocks depending upon given options.

4. In case we decided to allow few merge-blocks with common key it may happen that at last there will x number be blocks left where x<z. Depending upon the options, here also we can choose to have either a merged-block consists of x number of blocks, or not to use these x number of blocks.
5. Assign each of these merged-blocks to a sensor node.
6. Stop.

Now let us illustrate the above example with an example. Suppose p=5, and z=3. Other parameters are as follows- N =25, k=4, as it was earlier.

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)

Here, p%z=5%3=2. First, we choose j=0, which satisfies the condition $0 \leq j \leq p-1$. So we choose the block 0, block 6, block 12, block 18, and block 24. This forms a group of 5 blocks (blocks 0, 6, 12, 18, and 24) with no inter-block common key. Next, we choose j=2. So in the process of forming a group of 5 blocks with no common key we first choose block 2, then block 8, next block 14. Now since 14 is the last block

of the group consecutive 5 blocks starting at block 11, we choose the next block as the first block in the next group which is block 15, and next we choose block 21. So we form another group of 5 blocks (blocks 2, 8, 14, 15 and 21) with no inter-block common key. From these two groups of blocks we choose any 3 blocks and make merged-blocks. Say, we take blocks {0, 6, 12} to form a merged-block, and then take blocks {2,8, 14} to form the next merged block. Next we form p merged-blocks as follows: First merged-block consists of blocks {1, 3, 4}, second merged-block consists of blocks {5, 7, 9}, third merged-block consists of {10, 11, 13}, fourth merged-block consists of {16, 17, 19}, and fifth merged-block consists of {20, 22, 23}. So we obtain total 2+5=7 merged blocks. Still we have 3remaining blocks – namely block 15, 18, 21, and 24. We see that we can merge any three of these to get a new merged-block but that would give us a merged-block with inter-block common key. So depending upon our constraints either we consider this merged-block with inter-block common key, or we leave these four blocks as unusable. Note that even if we consider this merged-block, say blocks {15, 18, 21}, we will still have one block left unused namely block 24. We can assign this single block to a sensor node but then it will have small common key-chain.

6. Key Establishment in Transversal Design based KPS

We first illustrate our proposal by an example. In transversal design the each block consists of keys (i.e., treatments or varieties) of components. If we consider the whole design as a two-dimensional object, we can see that first component of any key in column i is of the form (i, x) where $0 \leq x \leq k - 1$. Let us again take the same example of previous section. We see that the second component of first column always repeats after a cycle of four digits - 0 to 4. In other words we can restate it as follows. The distance between two second component elements in the i-th ($0 \leq i \leq 4$) blocks in every group of p consecutive blocks is zero. In the second column, this distance increases by one. In this example, the first block in first group (i.e., block 0) contains the element 0 as the second component of second column, then the corresponding element in first block of second group (i.e., block 5) is 1, and in the first block of third group (i.e., block 10) is 2. In the third column, this distance increases by two. And this pattern holds true for all the columns. Let us present the algorithm. To make the algorithm easy to understand we have presented the algorithm for KPS with block size four, which can be generalized easily with little bit of extra effort.

1. Input: key-id, KEY_ID of a sensor node
2. if (key-id is one-dimensional)
    a. Take an integer variable x, and initialize x=KEY_ID
    b. Take another integer variable y, and initialize y=x%p
    c. Take another integer variable z, and initialize z=x/p
    d. a=y
    e. initialize b=0
    f. for(j=1;j≤z;j++)
        i. b=b+1

    [end of for loop]

    g. b=b%p

h. for(j=1;j≤y;j++)
    i. b=b+1

    [end of for loop]

i. b=b%p
j. initialize c=0
k. for(j=1;j≤z;j++)
    i. c=c+2

    [end of for loop]

l. c=c%p
m. for(j=1;j≤y;j++)
    i. c=c+1

    [end of for loop]

n. c=c%p
o. initialize d=0
p. for(j=1;j≤z;j++)
    i. d=d+3

    [end of for loop]

q. d=d%p;
r. for(j=1;j≤y;j++)
    i. d=d+1

    [end of for loop]

s. d=d%p
t. so the keys of node with id KEY_ID are (0,a), (1,b),(2,c),(3,d)

[end of if]

3. if(KEY_ID two-dimensional of the form (w,x))
    a. y=x
    b. z=w
    c. go to step 2 d.

    [end of if]

4. stop

So to establish the common key with other node y, a node x should know the id of node y. Given that, node x can compute all the keys of x by itself and then compare with its own to determine if x has any common key with y. There exists method that directly computes the common key, if that exists, by using multiplicative inverse operation [1]. Our method is lengthy, but does not use multiplicative inverse operation.

7. Conclusion

We have presented the deterministic strategy for merging of blocks. The existing work on this was not deterministic, but heuristic one. One can further investigate the difference it makes on the resiliency and connectivity of the distributed wireless sensor network.

8. References

[1] C. J. Colbourn and P. C. Van Oorschot, Applications of Combinatorial Design in Computer Science, ACM Computing Surveys, Vol. 21, No. 2, pp 223-250, June 1989.

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

[3] 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.

[4] J. Lee & D. R. Stinson, A Combinatorial Approach to Key Predistribution for Distributed Sensor Networks,. Wireless Communications and Networking Conference, IEEE_vol. 2 pp 1200-1205, March 2005

[5] K. M. Martin. On the Applicability of Combinatorial Designs to key predistribution for wireless sensor networks. arXiv e-print Archive Report. 2009.

[6] Anupam Pattanayak, B. Majhi. Key Predistribution Schemes in Distributed Wireless Sensor Network using Combinatorial Designs Revisited. Cryptology eprint Archive. Report 2009/131. 2009.

[7] 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. Springer.