How to Run the HHC Simulator

Following instructions describes how to execute the HHC simulator with key predistribution. The simulator includes two separate programs:

- *simulator* – the HHC simulator
- *gen_key* – generate key index files based on combinatorial design or random block merging in combinatorial design

Note that these programs are not developed to be input driven since the same simulation parameters are used to collect multiple samples using shell scripts. If specific changes are required relevant header files needs to be updates and program needs be recompiled as explained in section 3. The submitted source code is preconfigured with parameters to simulate a *30×30* network with random node placement, only half of the sensor filed will be covered ($q = 0.5$). Some of the simulation results are shown on the terminal and some of the data is dumped to several files for further analysis. Table 1 summarize list of parameter values.

Table 1 – List of preconfigured parameters and corresponding values.

Parameter	Value
Grid size	30×30
Distance between 2 grid points (both X and Y direction)	6 units
Location of root node	(15, 15)
Node placement probability (q) on the network	0.5
Node placement on network	Random
Node-to-node communication range (R)	30 units
CH-to-CH communication range	3 x 30 units
Max_hops (single hop clusters)	1
TTL	3
Use KPS Consider compromised nodes	Enabled Disabled
Number of key blocks merged (z)	4

Note: *The source code was developed and tested on both Ubuntu 32-bit and 64-bit platforms. It should work on another Linux variant without a problem.*

1. Key File Generator

First of all the key generator needs to be complied from the source code. The source code for key generator is stored in the **gen_key** directory inside the **HHC_KPS** directory. Use the following code segment to compile the program.

```
dilumb@C20303:~/$ cd HHC_KPS
dilumb@C20303:~/HHC_KPS$ cd gen_key
dilumb@C20303:~/HHC_KPS/gen_key$
```

Then compile the source using the **make** command.

```
dilumb@C20303:~/HHC_KPS/gen_key$ make
```

If the compilation is successful you should see something similar to the following:

```
gcc -c kps.c
gcc -c gen_key.c
gcc kps.o gen_key.o -o gen_key -lm
```

If the compilation is successful a file named **gen_key** should be generated.

```
dilumb@C20303:~/HHC_pkd/HHC_KPS/gen_key$ ls
gen_key  gen_key.c  gen_key.o  key_id1.txt  kps.c  kps.h  kps.o
Makefile
```

Then we can use the **gen_key** program to generate key index file based on the Lee-Stinson approach as follows:

```
dilumb@C20303:~/HHC_pkd/HHC_KPS/gen_key$ ./gen_key
Enter p (primer or prime power):5
Enter k (no of keys per block):3
Enter z (no of blocks per node):1
```

In the above example we have selected $p = 5$, $k = 3$ and $z = 1$. If $z > 1$, Chakrabarthi-Maitra-Roy approach is used to generate the key index file. All the key indexes will be saved in the **key_id.txt** file. Sample **key_id.txt** file for above input is shown below.

```
15      25      5       3       1
0       0       0       0       1       0       2       0
0       1       0       1       1       1       2       1
0       2       0       2       1       2       2       2
0       3       0       3       1       3       2       3
0       4       0       4       1       4       2       4
1       0       0       0       1       1       2       2
1       1       0       1       1       2       2       3
1       2       0       2       1       3       2       4
1       3       0       3       1       4       2       0
1       4       0       4       1       0       2       1
2       0       0       0       1       2       2       4
2       1       0       1       1       3       2       0
2       2       0       2       1       4       2       1
2       3       0       3       1       0       2       2
2       4       0       4       1       1       2       3
3       0       0       0       1       3       2       1
3       1       0       1       1       4       2       2
3       2       0       2       1       0       2       3
3       3       0       3       1       1       2       4
3       4       0       4       1       2       2       0
4       0       0       0       1       4       2       3
4       1       0       1       1       0       2       4
4       2       0       2       1       1       2       0
4       3       0       3       1       2       2       1
4       4       0       4       1       3       2       2
```

This file needs to be copied to the simulator directory if to be used within simulator.

2. HHC Simulator

Like the key index generator, the simulator needs to be complied from the source code. Simulator source code is in the **HHC_KPS** directory.

Use following commons to compile the simulator:

```
dilumb@C20303:~/HHC_pkd/HHC_KPS/gen_key$ cd ..
dilumb@C20303:~/HHC_pkd/HHC_KPS/$ make
```

If the compilation is successful you should see something similar to the following

```
gcc -c print_data.c -Wunused
gcc -c kps.c -Wunused
gcc -c energy.c -Wunused
gcc -c simulator.c      -Wunused
gcc print_data.o kps.o energy.o simulator.o -o simulator -lm -Wunused
```

This will generate an executable file named **simulator**.

Before running the simulator makes sure that the key index file (**key_id.txt**) exists, if you are using KPS with HHC algorithm. If such a file does not exist generate a file using **gen_key**. Use $p = 43$, $k = 9$ and $z = 4$ to generate a key file for the network with 450 nodes ($q = 0.5$). Then copy that file to the **HHC_KPS** directory.

Use following command to execute the simulator:

```
dilumb@C20303:~/HHC_pkd/HHC_KPS$ ./simulator
Can't continue. Initial node doesn't exist
```

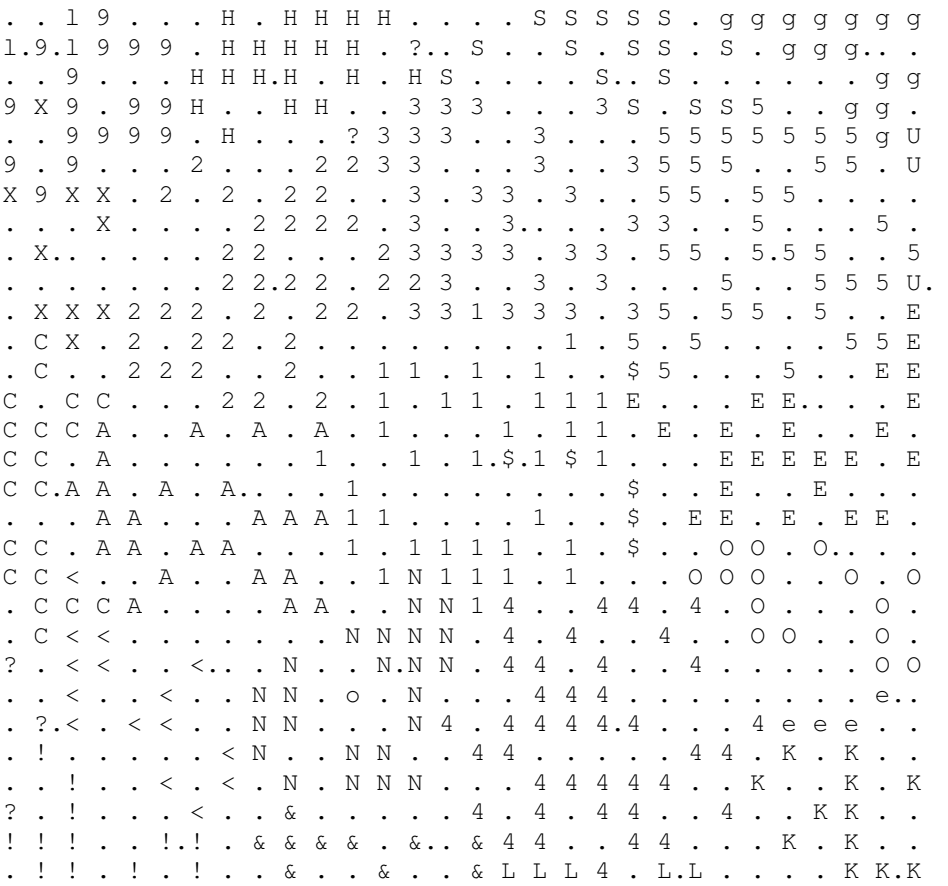
If random node placement did not place the *root node* in the middle of the sensor field the simulator cannot continue. In such a case rerun the program, you may need to rerun several times. This is not a problem in pregenerated networks because we can pick only networks with the root node.

Following output indicates successful execution of the simulator.

```
dilumb@C20303:~//HHC_pkd/HHC_KPS$ ./simulator
No cluster: 1    No common key: 1
No of messages delivered: 20752
```

It indicates the number of nodes without a cluster, number of nodes without a common key with a nearby Cluster Head (CH) and number of messages delivered by the network before the first node die.

The simulator also shows the placement of nodes in the network and their clusters. Node belongs to the same cluster is shown using the same symbol. Symbol followed by a period (.) sign indicates a CH. Grid points without a node is also indicated by a period (.) sign. Following is a sample output.



Simulator also stores some of the simulation data in following files for further processing:

- *circular.txt* - Dump the maximum achievable circularity of each cluster in the network
- *common_keys.txt* - Dump the average number of common keys a node share with its neighbors.
- nodes.txt – Dump all the node related data such as their node ID, cluster ID, node ID of the CH, depth in the cluster tree, number of messages forwarded by a node, etc.

Rerun the simulator to see the effect of random node placement and cluster formation. Section 3 describes meaning of the parameters and values that can be varied. Each time a parameter is change simulator needs to be recompiled.

3. Simulator Parameters

The **types.h** files include all the use configurable parameters. Each parameter value should be appropriately set and program should be recompiled before executing the simulator. Table 2 indicates parameters, their meaning and suitable values. Only parameters that can be varied by the user given.

Table 2 – List of parameters in the types.h file.

Parameter	Description	Value
GRIDX	Distance between two nodes in the grid (X direction)	Any integer ≥ 1
GRIDY	Distance between two nodes in the grid (Y direction)	Any integer ≥ 1
NODESX	Number of nodes in X direction	Any integer ≥ 10
NODESY	Number of nodes in Y direction	Any integer ≥ 10
STARTX	X coordinate of root node	NODESX/2
STARTY	Y coordinate of root node	NODESY/2
NODEPROB	Node placement probability (*q*)	1/q
R	Node-to-node communication range (R)	≥ GRIDX
MAX_HOPS	Max_hops (single hop clusters)	1
MAX_TTL	TTL	2×MAX_HOPS + 1
DATA_PACKET_SIZE	Size of a data packet	Any integer ≥ 1
ACK_PACKET_SIZE	Size of a acknowledgement packet	Any integer ≥ 1
CLUSTER_BCAST_SIZE	Size of a cluster formation broadcast packet	Any integer ≥ 1
KEY_INDEX_SIZE	Size of key index	16 bits × MAX_Z
USE_NODE_FILE	Whether use a pregenerated node file	0 – random node placement 1 – use pregenerated node file
SHOW_NODE_DATA	Show node data on terminal	0 – No data on terminal 1 - Dump data to terminal
CIRCLEFILE	File to save circularity data	circular.txt
NODEFILE	File to save node data	nodes.txt
NODELIST	Pregenerated node file	input_nodes.txt
ENERGYFILE	File to dump node energy related data (disabled)	energy.txt
KEYFILE	Pregenerated key index file used for KPS	key_id.txt
COM_KEYFILE	File to dump average number of common keys of a node with its neighbors	common_keys.txt
USE_KPS	Use key predistribution	0 – Do not use KPS 1 – Use KPS
NODE_COMP	Nodes are compromised	0 – No compromised nodes 1 – Consider compromised nodes
COMP_NODES	Number of compromised nodes	Any integer ≥ 1
MAX_Z	Number of key blocks merged (z)	$2 \leq z \leq 4$