CHAPTER 4

THE IMPLEMENTATION OF COVER-SELECTION-BASED AUDIO STEGANOGRAPHY ALGORITHM (CAS)

4.1 Introduction

This chapter discusses the implementation of a cover-selection based audio steganography algorithm based on the design presented in the previous chapter. The parameters of the proposed algorithm which affect the characteristics discussed in Section 4.2. The MATLAB programming language is used to develop the code of the algorithm as it allows the manipulation of matrix, plotting of functions and data and implementation of algorithms which is very needed in this stage. In addition, the empirical study on the proposed algorithm is presented in Section 4.3 to give the reader a clear view on calculation and formulation related to this algorithm Lastly, the differentiation between previous algorithms and proposed algorithm are shown in Section 4.4.

4.2 Proposed Algorithm CAS and its Components

In this section, the proposed CAS design was moved to the level implementation. The algorithm for the embedding part - BCM-LSB, cover selection part – MCAS, and integration part between embedding part and cover selection part CAS are discussed. Section 4.2.1 discusses the algorithm of BCM-LSB while Section 4.2.2 discusses the algorithm of MCAS. Lastly Section 4.2.3 discusses the integrator part of MCAS and BCM-LSB, CAS algorithm.

4.2.1 BCM-LSB: Algorithm

The steganographic BCM-LSB main function algorithm is illustrated in Figure 4.1.

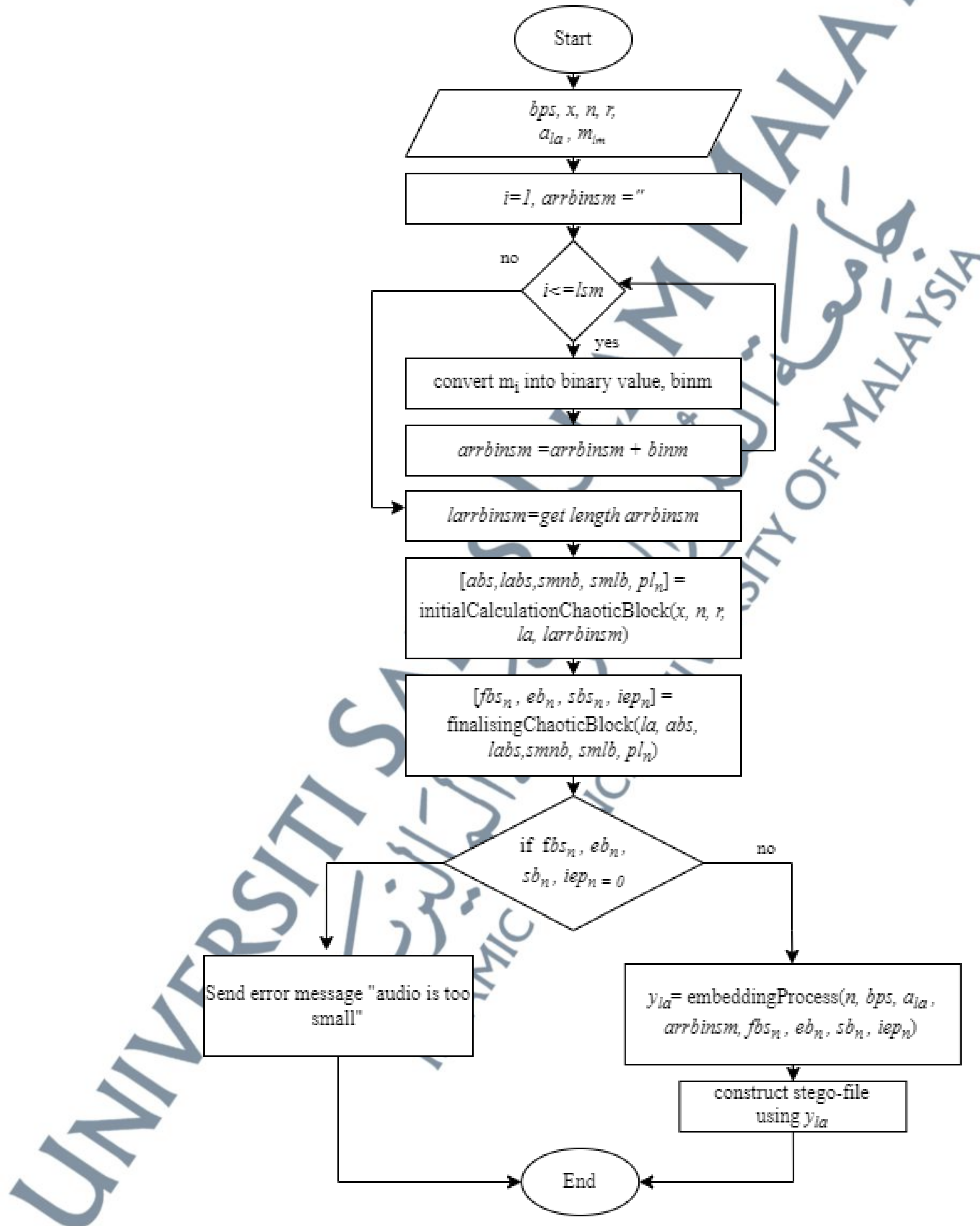


Figure 4.1: BCM-LSB Main Algorithm

UNIVERSITI SAINS ISLAM MALAYSIA

Based on Figure 4.1, BCM-LSB main function algorithm was separated into three stages. The steganographic process was initiated by obtaining four types of user input: 1) secret information, *smlm*, 2) secret key combinations: *x*, *n*, and *r*, 3) cover audio, *ala*, and 4) bit per sample, *bps*. The secret information was subsequently converted into one long secret information binary string, *arrbinsm*. Next, a function named "initialCalculationChaoticBlock()" is used to create an initial block skeleton for the chaotic block. "finalisingChaoticBlock()" function followed after that which is used to finalise the size of audio sample, initial embedding point, starting and ending of audio sample per block. Lastly, the "embeddingProcess()" function is used to embed the secret message accordingly into its designated blocks which is called if the fbs_n, eb_n, sb_n, iep_n, is not zero which produced stego-file, as an output or else it display the error message "audio is too small" which indicates the embedding process is failed due to the limited audio sample for the embedding process to occurs.

A fundamental dynamic security issue is the even distribution of secret information as much as possible throughout the cover audio file during the embedding process. Therefore, a skeleton of chaotic blocks derived from chaotic maps was developed. Figure 4.2 illustrates the flowchart of creating a chaotic block skeleton through function initialCalculationChaoticBlock().

UNIVERSITI SAINS ISLAM MALAYSIA
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

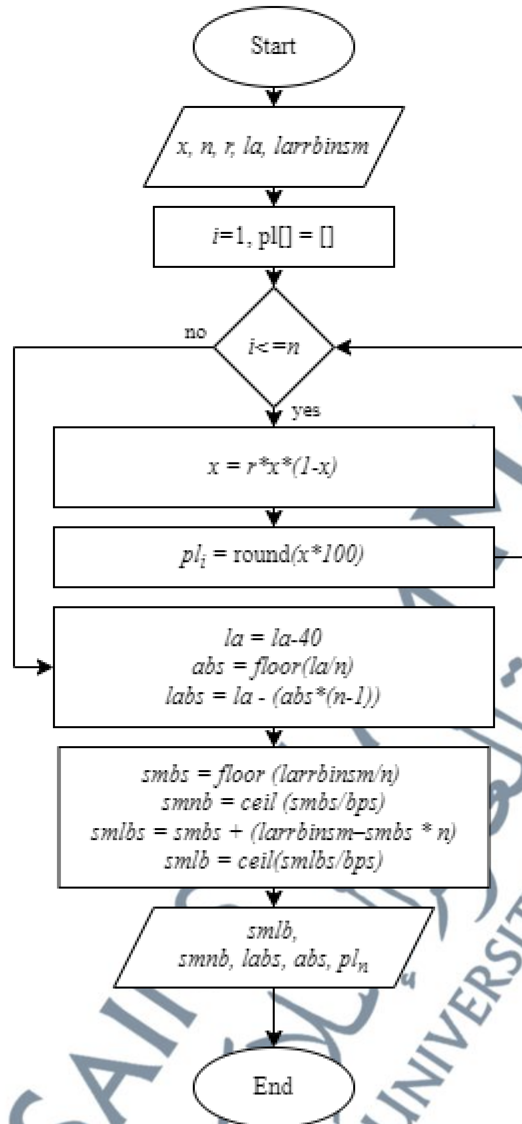


Figure 4.2: initialCalculationChaoticBlock Function Algorithm

Based on Figure 4.2, this subfunction is initiated by obtaining *x, n, r, length of audio, la* and length of *arrbinsm, lbinsm* from the main function. Next, it calculates the initial point for embedding in each block based on the percentage. To finalise the percentage sequence, pl_n: $\{pl_1, pl_2, pl_3, \ldots, pl_i\}$, the value is rounded to two decimal places. The next step is estimating the normal block size for the audio sample, *abs* and the last block size for the audio sample, *labs*. The rough idea is by dividing the *la* with *n*. However, the is a high probability that it will have some remainder. Hence, the remainder also needs to be included to ensure that the distribution of the secret message

continues until the end of the audio sample. The remainder is combined with the last block, hence the length of the last block, *labs*, is calculated. After the size of the normal audio block and the size of the last audio block are estimated, the next step is to estimate the block size for *arrbinsm.* Both size of secret message normal block*, smbs and* size of secret message last block, *smlbs* are calculated. After the *smbs and smlbs* are calculated, these values are used to calculate the size of sample audio used on the normal block, *smnb* and the size of sample audio used on the last block, *smlb.* Instead of using a fixed *bps* to be implemented in the method, this method gives freedom to the user to determine their own *bps* as it improves the randomness on top of chaotic block in addition increasing the capacity if needed. *smbs and smlbs* are divided with the *bps* to calculate *the smlb* and *smnb,* Finally, this subfunction will return *smlb, smnb, labs, abs,* pl_n to the main function.

finalisingChaoticBlock function is implemented to cater to the variety of situations for embedding. This subfunction is initiated by obtaining *abs, labs, smnb, smlb,* pl_n and *la.* Figure 4.3 illustrates the algorithm through the flowchart to finalise the initial embedding point, starting and ending points for each block.

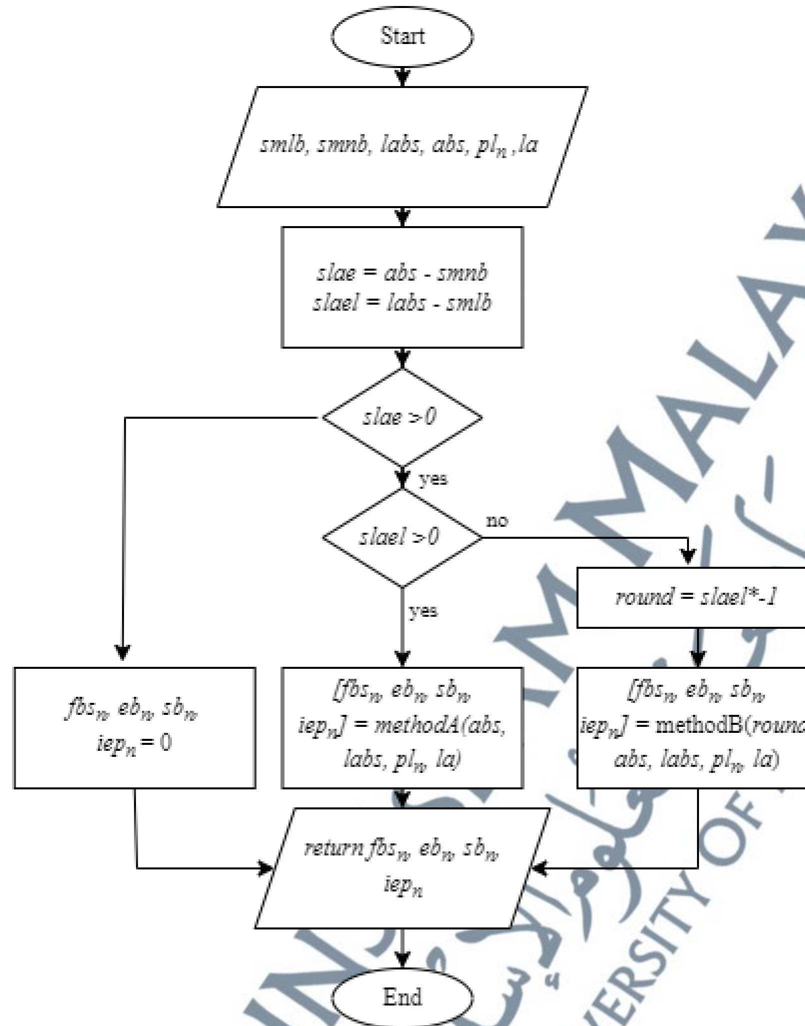


Figure 4.3: finalisingChaoticBlock Function Algorithm

Based on Figure 4.3, the first step of this subfunction is, the samples left after embedding at the normal block, *slae* and sample left after embedding at the last block, *slael* are calculated. Next, *slae* is compared with 0. If the *slae* is greater than 0, it indicates that the audio is big enough to contain the secret message, hence, the next step can be proceed or else, the subfunction returns 0 to indicate the audio cannot be used as it is smaller compared to the size of secret message hence, the user can use bigger audio and repeat the process again or change to higher *bps* to increase the capacity. Next, *slael* is compared to the 0, if it is greater than 0, subfunction methodA() is called in which no adjustment takes place. On the other hand, if *slael is* lower *than 0*, subfunction methodB()

is called. The idea is that as the normal block has extra samples to be spared from the previous condition, hence, the last sample can be adjusted further by taking some portion from the normal block to increase the size of the last block. Both subfunctions methodA and methodB are explained further through the flowcharts in Figure 4.4 and Figure 4.5.

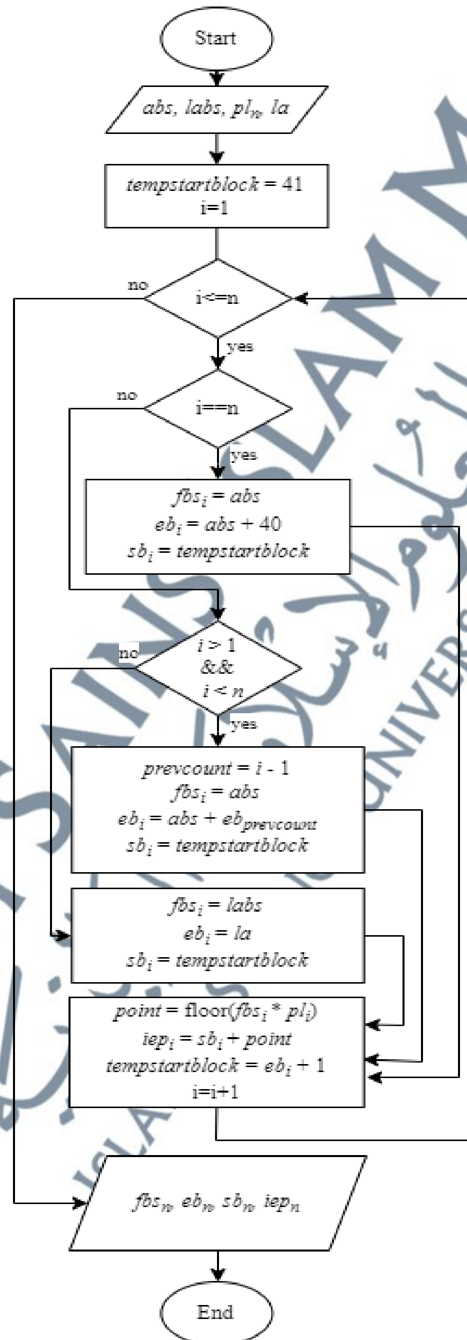
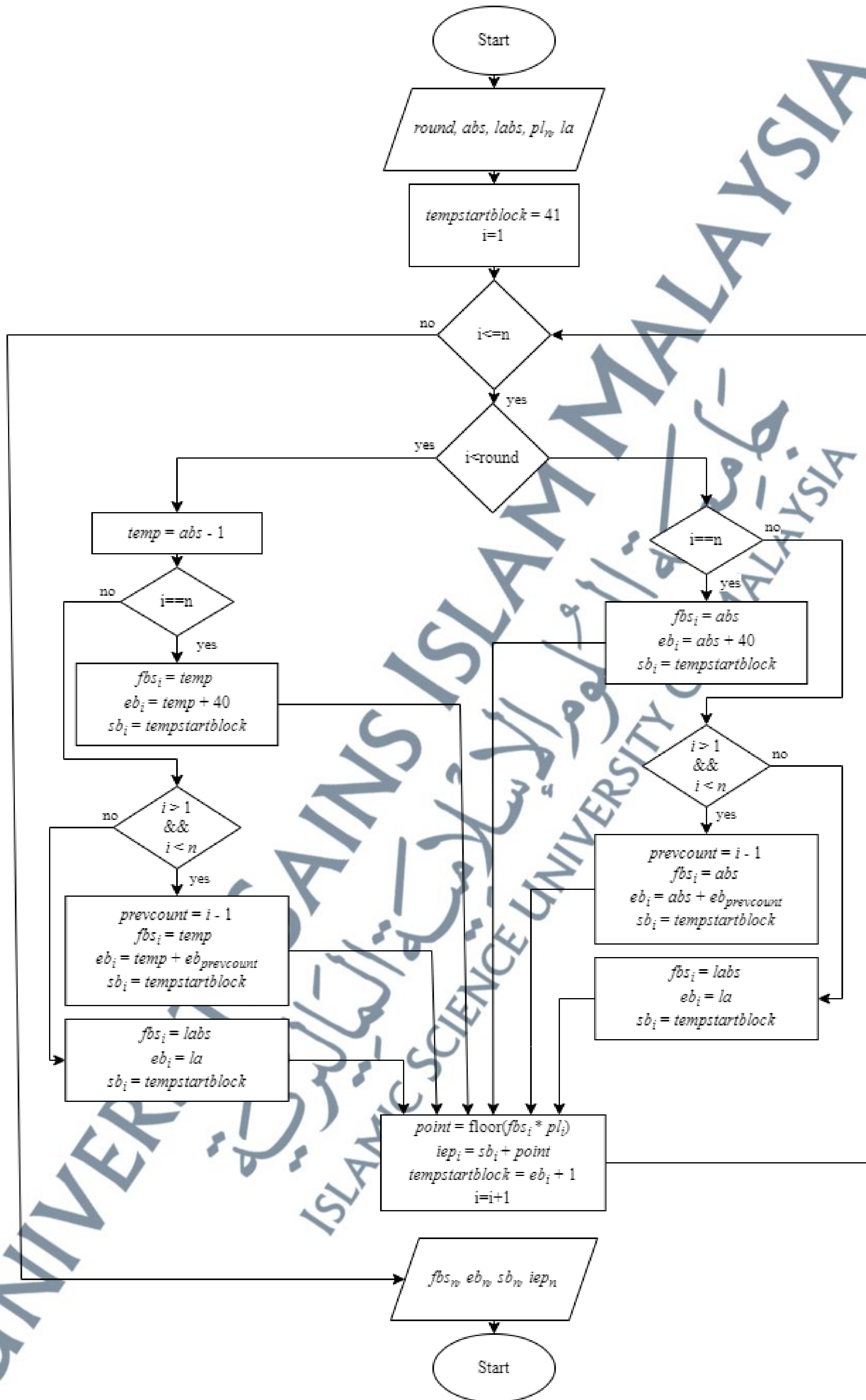


Figure 4.4: methodA subfunction Algorithm

Based on Figure 4.4, methodA() is used to finalise the initial embedding point, iep_n, size of block, fbs_n, start block, sb_n, and end block, eb_n, during the normal condition. It starts by an initialised temporary start block, *tempstartblock at index 41*. Then the next step is determining the value which depends on 3 conditions that loop n times. For the first condition where $i == 1$ which the first block, a slight adjustment needs to be made as the first 40 audio samples are used for storing length of secret message in Algorithm 2. From there, fbs_i, eb_i, sb_i values are assigned. For the second condition where $i > 1$ && $i < n$, the values of fbs_i, eb_i, sb_i are assigned normally by amending the values of eb_{i-1}, sb_{i-1} from the previous block and the *abs*. For the last condition, which is $i == n$, fbs_n, value is assigned using *labs*, eb_n, value is assigned *using la* while sb_n value is assigned using *tempstartblock*. The next step is determining the initial embedding point by calculating the product of fbs_i, *and* pl_i. *tempstartblock* is updated by using eb_i+1. After the end of this for loop, this subfunction returns the values fbs_n, eb_n, sb_n, iep_n.

UNIVERSITI SAINS ISLAM MALAYSIA
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA



UNIVERSITI SAINS ISLAM MALAYSIA
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

Figure 4.5: methodB subfunction Algorithm

Based on Figure 4.5, methodB() subfunction is created to address the issue of unsuitable-sized *labs compared to smlb.* In addition to the basic input from Algorithm 4, it also inquiries about another input *round,* which, used to modify the *round* numbers of blocks to readjust the *labs.* First, it starts by declaring a temporary start block, *tempstartblock at index 41.* Then the next step is determining the values of fbs_i, eb_i, sb_i depends on 2 main conditions with 3 sub conditions that loops *n* times. For the first selection, $i < round$, this selection is used to modify only *round* number of fbs_i instead of changing all the fbs_i, this decision is taken to ensure that only needed samples are added to the last block to prevent too many extra samples at the end unused hence reducing the dynamic security level. The temporary audio block size, *temp* is calculated before going to the sub selection. In selection 1-a where $i == 1$, a slight adjustment needs to be made as the first 40 blocks are used for storing the length of the secret message in Algorithm 2. From there, fbs_i, eb_i, sb_i values are assigned. In the sub selection 1-b where $i > 1$ && $i < n$, fbs_i value is assigned based on the *temp* value, eb_i value is assigned based on previous $eb_{prevcount}$ *and* sb_i value is assigned based on *tempstartblock.* Last sub condition is used to assign value for the last index, *n* of fbs_n, eb_n, sb_n. They are assigned to the values of *labs*, *la* and *tempstartblock* respectively. For the selection 2 and its sub selections, they are similar with the ways of determining fbs_n, eb_n, sb_n in Algorithm 4. Next, initial embedding point, fbs_i is determined by calculating the product from fbs_i, *and* pl_i. At the end of this subfunction, it returns the values fbs_n, eb_n, sb_n, iep_n.

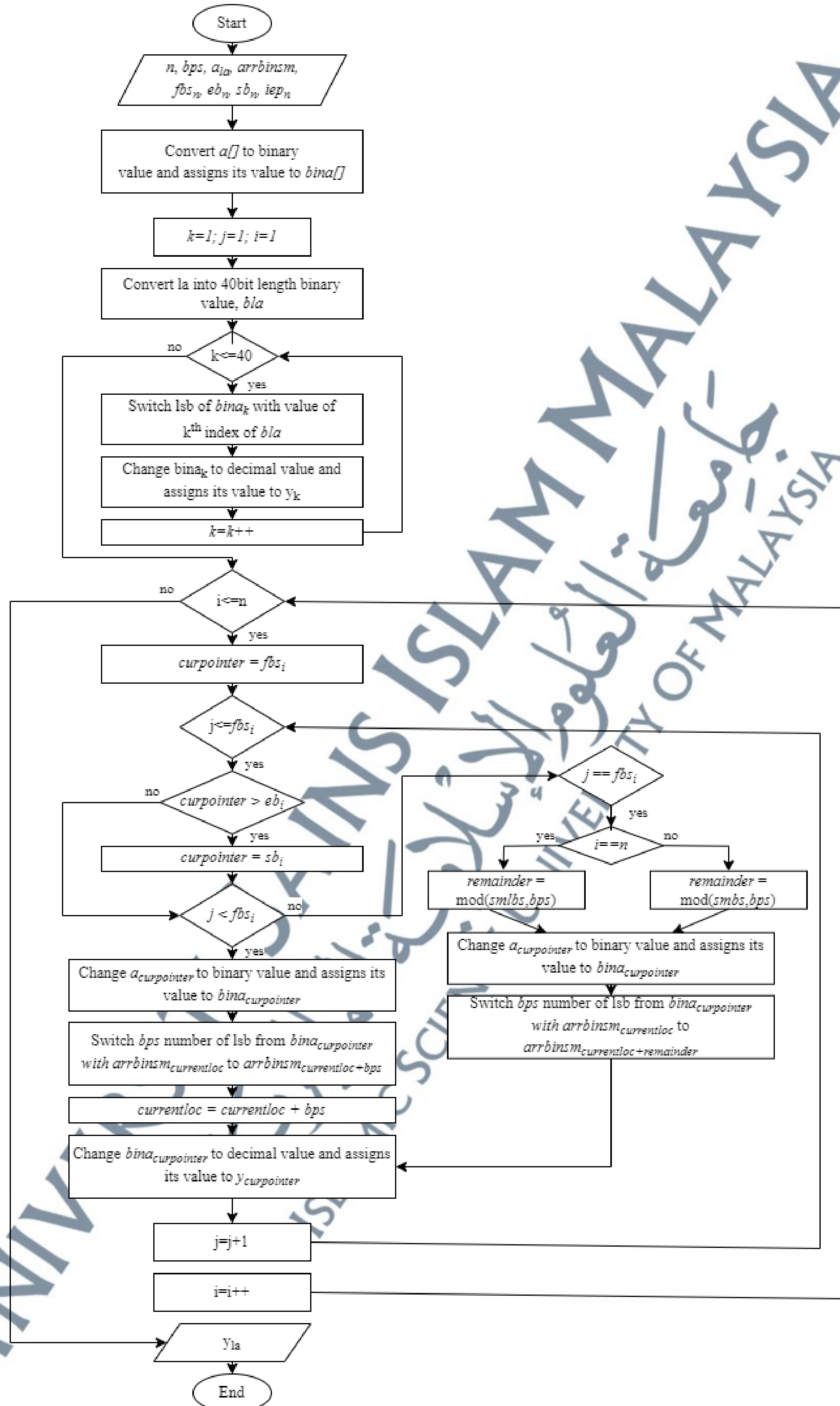


Figure 4.6: embeddingProcess Algorithm

Based on Figure 4.6, the embeddingProcess() utilised a smaller scale sequential low bit embedding approach, which embedded based on the previously calculated chaotic block instead of the whole cover audio. The subfunction of the embedding process algorithm is illustrated through a flowchart in Figure 4.6. The first step is embedded the length of the secret message in the first 40 audio samples. Next, inside the loop $i = 1$ to n which represents smaller sequential embedding per block, the current pointer for monitoring the embedding process to avoid embedding into another block, *curpointer* is initialised with fbs_i. This mechanism, named as the rounding mechanism also implemented to ensure that there is no loss in BCM-LSB capacity if the embedding end before fully completed when reach the eb_i without going back to the sb_i which still unused as the remaining audio samples could be used for embedding. For the next step, there are two situations occur that require consideration on the embedding *arrbinsm* in the *a*. For the condition $j < fbs_i$, it embeds the *arrbinsm* normally according to the *bps* allocated earlier by the user. However, in $j == fbs_i$ it embeds the remainder of *arrbinsm* from the expected *smlb* or *smbs.* Sometimes the remainder could be equal to the *bps* but sometimes it is less than *bps* used, which is a concern. After finishing all the embedding processes up until the last block, the stego-file, y_{la} is returned as an output of this function.

The main parameters that manipulate the performance of BCM-LSB are *bps* used and key *x*, *n* and *r*. *bps* manipulates the performance of capacity, imperceptibility and robustness characteristics while key *x*, *n* and *r* manipulates the dynamic security characteristic performance. further evaluations and analysis of these parameters are presented in Chapter 5.

4.2.2 MCAS: Algorithm

The cover selection part, MCAS main function algorithm was illustrated in Figure 4.7.

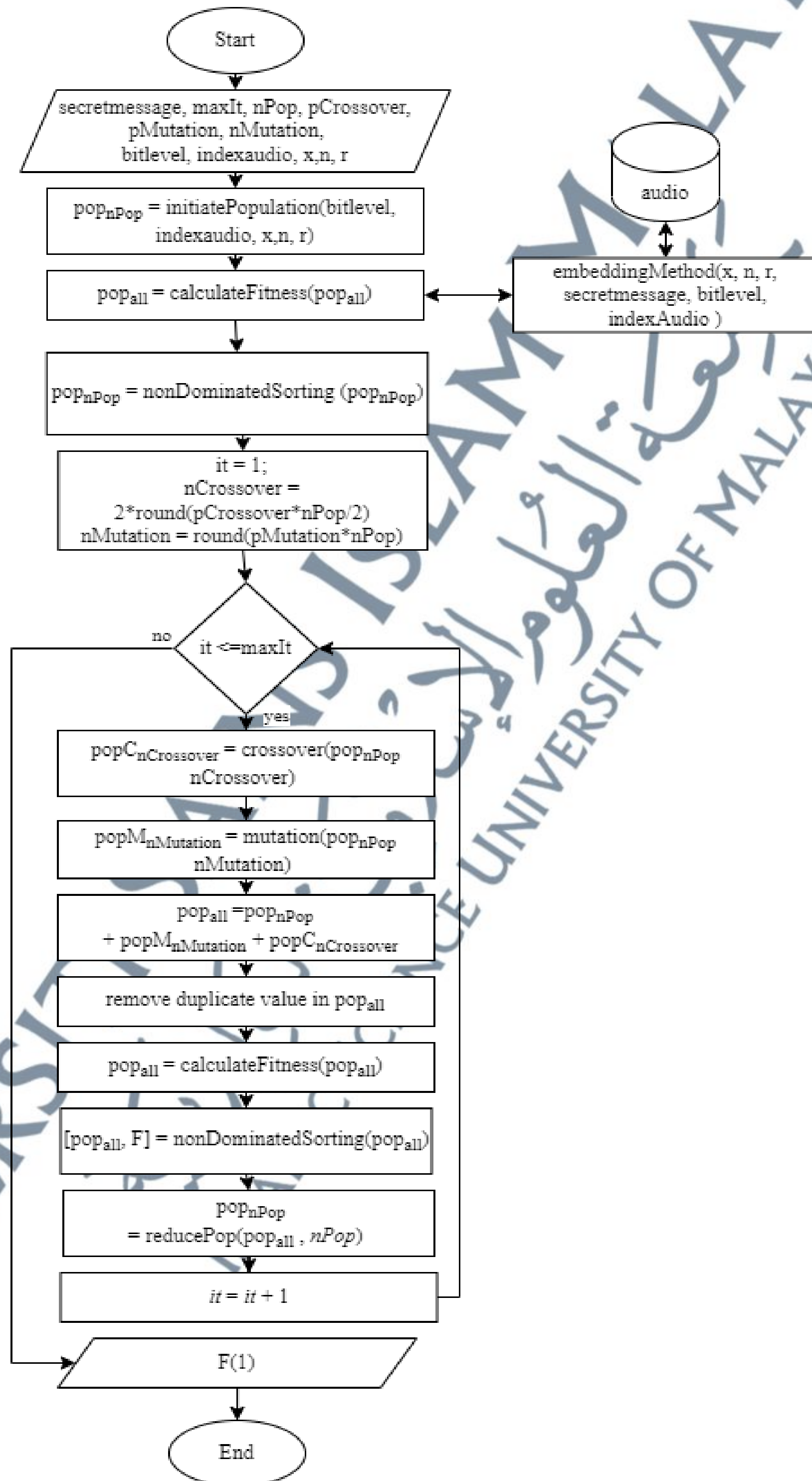


Figure 4.7: MCAS Algorithm flowchart

Based on Figure 4.7, it is separated into several stages which are;

1) initialising the population, function initiatePopulation(),
2) calculate fitness function, calculateFitness(),
3) non-dominate sorting, nonDominatedSorting(),
4) crossover between the chromosome in the population, crossover(),
5) mutation of the chromosome in the population, mutation(), and,
6) sort the population and remove weak chromosome to control number of population, reducePop().

The main algorithm starts by gathering the input from the user which are secret message, *secretMessage*, the range of bit per sample that can be embedded, *bpsRange*, the range of index audio in the database, *indexAudioRange*, number of generation, *maxIt*, size of population, *nPop*, the probability of crossover, *pCrossover*, and probability of mutation, *pMutation*. Then, the function initiatePopulation() is used to initiate the population. The initialisation takes place by creating a new chromosome with random value of 5 alleles which represents the bit embedded per sample, *bitlevel*, index of audio in the database, *indexaudio*, and key *x*, *n* and *r* (*x, n, r)*. Then it will calculate fitness values on the imperceptibility using the Equation 2.2, robustness using the Equation 2.5, dynamic security using Equation 3.20 to Equation 3.22 in the function calculateFitness(). Although it does not consider the capacity, it rejects the audio which does not have enough capacity indirectly by assigning other characteristics value the lowest value possible which will make that chromosome ranked last in the later step of this algorithm. It will return the chromosome with its fitness value to the array of pop_{nPop}.

Next, the algorithm will do the non-dominated sorting on the chromosome in the pop_{nPop} using nonDominatedSorting() function. For each chromosome in pop_{nPop} is

compared to the other chromosome until all the chromosomes are compared and all the non-dominated chromosomes are assigned to the first front. The non-dominated chromosome is determined by the 2 rules discussed earlier in the design part. Then, these non-dominated chromosomes are removed from the comparison and another round of comparison is conducted among leftover chromosomes to determine which chromosome is becoming the next non-dominated chromosome. This process is repeated until all the chromosomes are assigned to their very own front. After all this sorting is finalised, this function will return back the pop_{nPop} with addition of chromosomes assigned front.

Next step this algorithm initialised the *it, nCrossover* and *nMutation* to be used for the main loop. *it* is used to keep the count of the generation produced from the loop and be used as an exit condition for the main loop as the *it* value is same with the maximum generation, *maxIt,* the algorithm will exit the main loop. *nCrossover* is used as a parameter in function crossover while *nMutation* is used as parameter in mutation function.

Next, this algoritm will set the value of $pop_{nCrossover}$ by using the function crossover(). As *nCrossover* refers to the number of expected chromosome newly produced, the crossover function will divide the *nCrossover* with 2 and round up the result to get the number of expected iteration to run the crossover process. For each number of iteration, two random chromosome are picked. Next, a randomised crossover point is determined. Next, the two chromosome are exchanged with the allele among them at the crossover point. The two children (chromosomes) were produced and kept in an array. This process is repeated until achieved the maximum iteration. The array will be returned and set the value of $pop_{nCrossover}$.

Following the crossover() function, the mutation() function is used to set the value of $pop_{nMutation}$. As *nMutation* refers to the number of expected chromosomes newly produced by the mutation function, it becomes the value for iteration to run the mutation process. The type of mutation process that is implemented is one point flip mutation. For each number of iterations, a random chromosome from the pop_{nPop} is selected. Then a random allele is chosen from five (5) existing allele and its maximum and minimum values are determined. Next, a random value is picked within the range of these two values. This value then replaces the current value of that allele and a new chromosome is produced and then is kept in an array. This process is repeated until the maximum iteration is achieved. The array will be returned and set the value of $pop_{nMutation}$.

Next, the all the $pop_{nMutation}$, pop_{nPop} and $pop_{nCrossover}$ are combined into one big array pop_{all}. Then the duplicate value inside pop_{all} is removed. This step is added to fasten the process of calculateFitness() and nonDominatedSorting() function after that. After completing both functions, the reducePop() function is used. This function takes pop_{all} and reduce the chromosome inside the population until it reached the similar number of *nPop*. Higher fronts are given priority to stay inside the population which indicates groups of higher fitness chromosomes. All these processes are repeated by increasing parameter *it* by 1, until *it* is higher than *maxIt*. Lastly it will return the all the chromosomes which are assigned inside the first front, *F(1).*

The main parameters that manipulate the performance of MCAS are number of generation*, maxIt,* population size*, nPop and the usage of reducedPop()* function which removes the duplicate function. All these parameters manipulate the performance of accuracy of MCAS method in finding the best cover audio. Further evaluations and analysis of these parameters are presented in Chapter 5.

UNIVERSITI SAINS ISLAM MALAYSIA ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

4.2.3 CAS: Algorithm

CAS algorithm is the combination of MCAS and BCM-LSB algorithm which are connected as an algorithm that helps user to choose the audio and embed it through steganographic process. The CAS algorithm is illustrated in Figure 4.8.

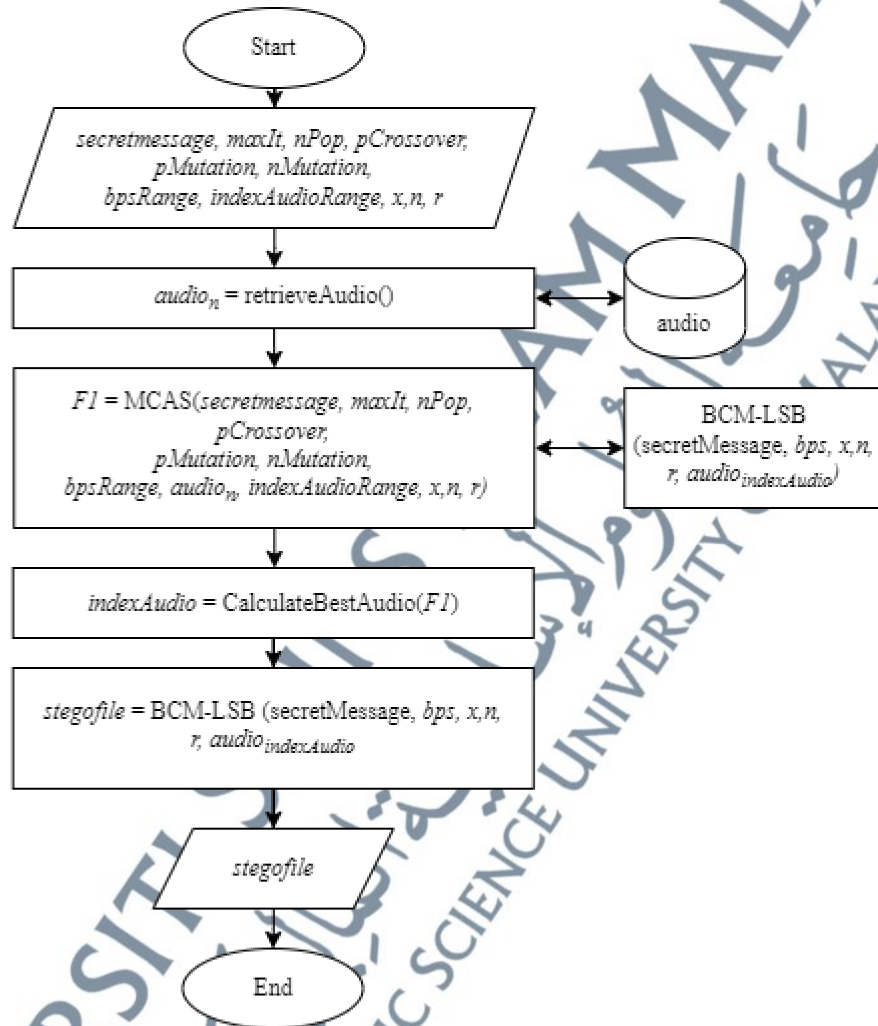


Figure 4.8: CAS Algorithm Flowchart

Based on Figure 4.8, the CAS algorithm starts by taking the input needed for BCM-LSB and MCAS algorithm similar to the previous discussion. After running the MCAS function which produces list of non-dominated solution, F1, the CAS algorithm use the CalculateBestAudio() function to find the best audio based on the trade-off. In

order to find the best audio, this function calculates the normalised combined value of normalised imperceptibility, normalised robustness and normalised dynamic security value for each audio. Next, the normalized combined value will be sorted, and the highest value is chosen. This function will return the index audio for the related to the highest normalised combined value. After that, BCM-LSB will embed the secret message using the variables needed and produce a stego-file as an output.

4.3 Empirical Study for CAS and its Components

In this section, the proposed CAS and its components are discussed empirically. The empirical study of these algorithm is introduced to give the reader more insight into the algorithm itself. Section 4.3.1 discusses the empirical study of the BCM-LSB algorithm Section 4.3.2 discusses the empirical study of the MCAS algorithm. Lastly Section 4.3.3 discusses the integrator part, CAS algorithm empirically.

4.3.1 BCM-LSB: Empirical Study

This section discusses the empirical study of the chaotic block from BCM-LSB. Figure 4.9 illustrated the flowchart of the BCM-LSB with actual value to give the reader a clear view on calculation and formulation related to this algorithm.

UNIVERSITI SAINS ISLAM MALAYSIA
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

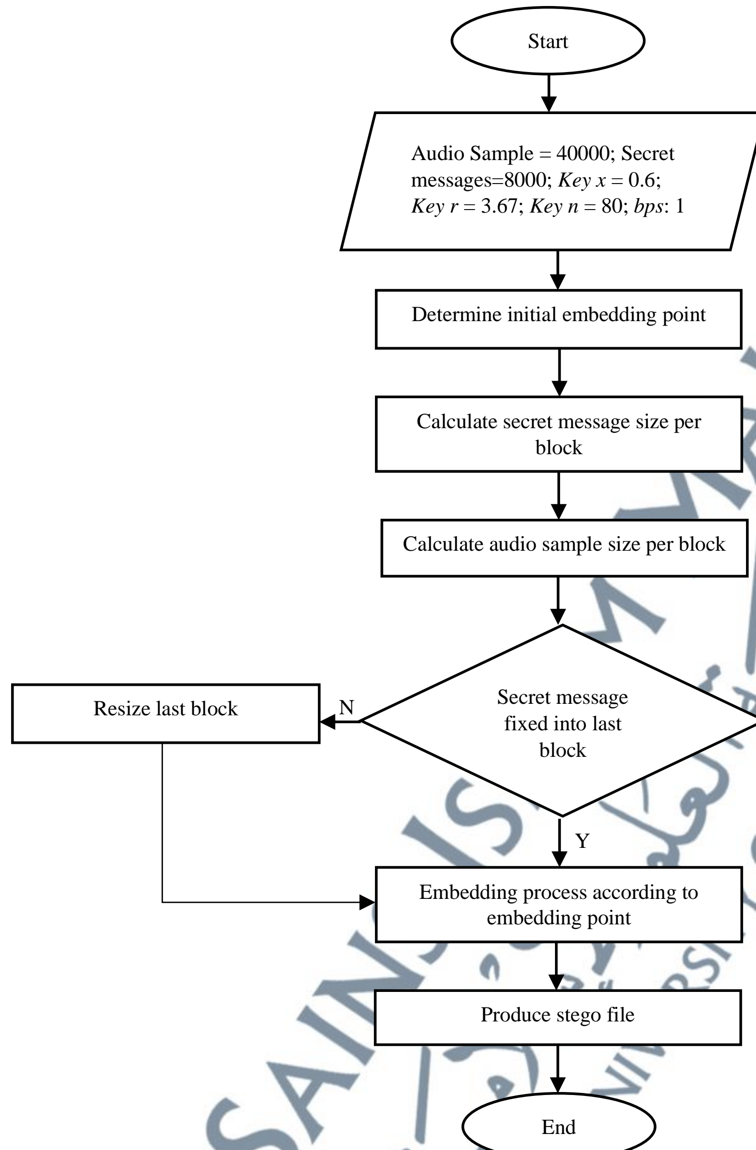


Figure 4.9: Flowchart of BCM-LSB with input values

Based on Figure 4.9, the inputs are set as follows: 1) audio sample: 40000, 2) secret message length: 8000 characters, 3) key *x*: 0.6, 4) key *r*: 3.67, 5) key *n:* 80 and 6) *bps*: 1. Then the BCM-LSB are continued by determining the initial embedding point, calculate the secret message size per block and calculate audio sample per block. Depending on whether the secret message for last block can fix into the last block, the cover audio block may need resizing. After that, the embedding process is conducted to produce the stego-file. The process of determining initial embedding point, calculate

the secret message size per block, calculate audio sample per block and embedding process are discussed as follows:

- **Determining Initial Embedding Point**

The initial embedding point can be determined using Equation 3.2 and Equation 3.3. As an example, the first three initial embedding point can be calculated by using Equation 4.1 and Equation 4.2 as follows:

$$x_{0+1} = 3.67 * 0.600(1 - 0.600) = 0.8808$$
$$x_{1+1} = 3.67 * 0.8808(1 - 0.8808) = 0.3853$$
$$x_{2+1} = 3.67 * 0.3853(1 - 0.3853) = 0.8692$$

(4.1)

Next, $x_{1,} x_2 \ and \ x_3$ values were rounded, which can be denoted as follows:

$$P_{1,} = round(p = \ 0.8808, 2) = 0.88$$
$$P_{2,} = round(p = \ 0.3853, 2) = 0.39$$
$$P_{3,} = round(p = \ 0.8692, 2) = 0.87$$

(4.2)

Therefore, the first three initial embedding points were 0.88, 0.39 and 0.87 from the size of the audio sample block.

- **Calculating Secret Message Size per Block**

There were two sizes of secret message per block which were the normal size of the secret message per block and the size of the secret message at the last block. The normal size of a secret message per block can be computed by substituting Equation 3.4 with the key *n* value and size of binary secret message. The calculation was shown as in Equation 4.3.

$$s_{msg_block} = floor\left(\frac{8000}{80}\right) = 500$$

(4.3)

Therefore, size of secret message per block were 100 per block. Since there were no remainder from Equation 4.3, the size of secret message at the last block remained the same.

- **Calculating Audio Sample Size per Block**

There are two sizes of audio sample per block, which were the normal size of the audio sample per block and the size of the audio sample at the last block. The normal size of the audio sample per block can be computed by substituting Equation 3.8 with the key n value and size of the audio sample. The calculation was shown in Equation 4.4.

$$s_{sample_block} = floor\left(\frac{40000}{80}\right) = 500$$
(4.4)

Therefore, the size of audio sample per block were 500 per block. Since there was no remainder from Equation 4.4, the size of audio sample at the last block remained the same.

- **Embedding Process According to the Initial Embedding Point**

Before conducting the embedding process, the exact initial embedding point needed to be determined. For this round of empirical study, the first block was selected. As we calculated earlier, the size of the audio sample per block is 500. Therefore, initial embedding point can be determined using Equation 4.5 as follows:

$$InitialPoint_{1,} = round(0.88 * 500) = 440$$
(4.5)

Therefore, the initial embedding point for the first block was at the 440 index of the audio sample. Although there are other situations which demand to use other working equations described in the BCM-LSB in Chapter 3, this empirical study is

UNIVERSITI SAINS ISLAM MALAYSIA ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

enough to give the reader a basic view on what happened during the chaotic block creation.

4.3.2 MCAS: Empirical Study

This section discusses the empirical study of the MCAS. Figure 4.10 presents a flowchart for the empirical study of MCAS. Based on Figure 4.10, the inputs are set as follows: 1) size population: 300, 2) number of generations: 20, 3) crossover probability: 0.8, 4) mutation probability: 0.05, and 5) mutation rate:0.02. Then the MCAS is continued by initiating the population, calculating the fitness function for each chromosome, conducting fast non-dominated sorting, crossover, and mutation. Depending on whether the secret message for last block can fix into the last block, the cover audio block may need resizing. After that, the embedding process is conducted to produce the stego-file.

UNIVERSITI SAINS ISLAM MALAYSIA
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

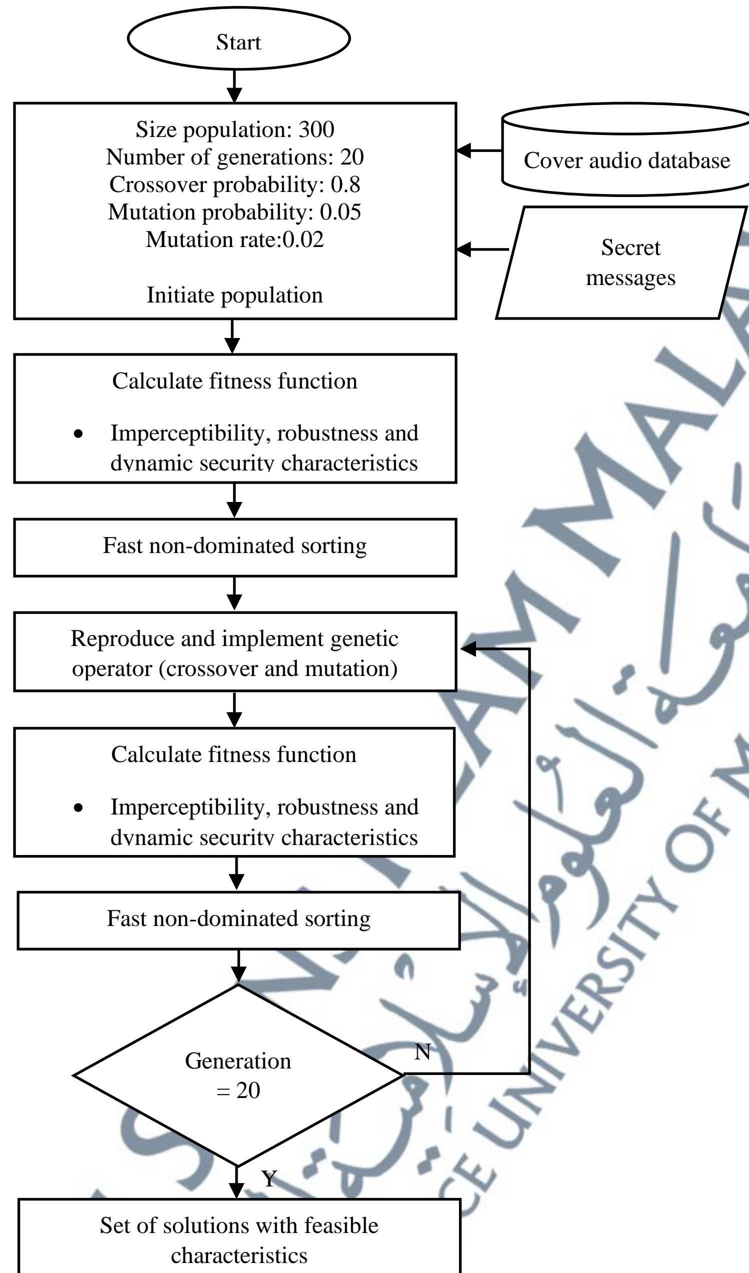


Figure 4.10: MCAS Empirical Study Flowchart

The process of determining initial embedding point, calculate the secret message size per block, calculate audio sample per block and embedding process are discussed as follows:

- **The Initiation population**

Figure 4.11 shows the sample of the initial population.

1	*2*	*3.75*	*0.9*	*1024*
2	*1*	*3.67*	*0.7*	*512*
3	*4*	*3.90*	*0.63*	*8*
.				
.				
.				
3	*4*	*3.90*	*0.63*	*8*

Figure 4.11: Example of the population in the initial generation

Based on Figure 4.11, individual chromosomes were created with 5 blank alleles. Then, each of the alleles was filled with random values of keys *x, r* and *n* alongside the *bps*. However, for the allele, which contains an index audio file, it was filled in sequences to ensure that at least all the cover audio files are evaluated once.

- **Calculate Fitness Function**

As mentioned earlier, there were three fitness functions which represent imperceptibility, robustness and dynamic security. However, since SNR which represents the fitness function for imperceptibility is well-known, the SNR calculation was skipped. Next, the fitness function of dynamic security can be calculated using Equation 3.20 until Equation 3.23. Assuming there were 5 segments which have Seg-SNR values of 50dB, 40dB, 30dB, Infinity and Infinity. Equation 3.23 can be substituted in Equation 4.6 as follows:

$$a = \sum_{1}^{3}\left(\frac{50+40+30}{3}\right) \tag{4.6}$$

where a was equal to 40dB. Therefore Equation 3.22 can be substituted with 40dB. Equation 4.7 which shown below:

$$sd = (|50-40|) + (|40-40|) + (|30-40|) \tag{4.7}$$

UNIVERSITI SAINS ISLAM MALAYSIA ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

which *sd* was equal to 20. After calculating the *sd*, the *nis* can be computed next by substituting Equation 3.21. The calculation of *nis* is shown in Equation 4.8.

$$nis = (5 - 2) * 10000 \tag{4.8}$$

Therefore, *nis* was equal to 30000. After calculating *nis* and *sd*, *DynSec* can be computed by substituting Equation 3.20 with the values of *nis* and *sd* as Equation 4.9.

$$DynSec = 30000 - 20 = 29800 \tag{4.9}$$

Therefore, the current stego-file produced has a dynamic security value of 29080. Lastly, to calculate the robustness, the AWGN attack was conducted and BER was computed after conducting the AWGN attack.

- **Fast Non-dominated Sorting**

After calculating the imperceptibility, robustness and dynamic security fitness function for each solution inside the population, each of the solution was compared with other solution. To conduct an empirical study on the non-dominated sorting, assuming there were four (4) solutions which have their own fitness level. Table 4.1 shows the solutions with their own values.

Table 4.1: Solution Sample with Fitness Value

Solution	Imperceptibility (dB)	Robustness	Dynamic Security
w	92.8177,89994	0.64	89994.54190
x	92.5301	0.62	0
y	67.6880	0.85	0
z	52.2949	0.85	0

As mentioned previously, one solution is considered dominant if: 1) This solution is not worse than other solution for all the objectives, and 2) This solution is strictly

better than other solution for at least one objective. Based on these two rules, the result are as follows:

Solution w was dominating solution x as it did not have any characteristics worse than solution x, and it had better imperceptibility and dynamic security compared to solution x. Solution w did not dominate y because it had a worse value in robustness but it was not dominated by solution y as it had higher imperceptibility and dynamic security compared to solution y. Therefore, solutions w and y were at the first rank. Since solutions x and z cannot dominate each other, both were at the second rank.

Table 4.2 presents the rank of each solution after the empirical study on non-dominated sorting was conducted.

Table 4.2: Solution Sample with Ranking

Solution	Ranking
w	1
x	2
y	1
z	2

- **The Crossover and Mutation**

In this subsection, the crossover mechanism was discussed empirically. Before conducting the crossover operator, the number of solutions created from the crossover needed to be determined first. To calculate the number of solutions created from the crossover, the size of the population needed to be multiplied by the number of crossover probabilities. Therefore, the number of solutions created from crossover is as Equation 4.10:

$$number\ of\ crossover\ children\ =\ 300 * 0.8 = 240 \quad (4.10)$$

Next, the mutation mechanism was discussed empirically. Similar to crossover operator, the number of solutions created from the mutation needed to be determined first. To calculate the number of solutions created from the crossover, the size of the population needed to be multiplied by the number of mutation probabilities. Therefore, the number of solutions created from crossover is as Equation 4.11:

$$number\ of\ crossover\ children\ =\ 300 * 0.05 = 15 \tag{4.11}$$

4.3.3 CAS: Empirical Study

The empirical study of the CAS is presented in this section. It focuses on CalculateBestAudio() Function.

Since there are numerous solutions listed from MCAS under *F1* variable, this empirical study limited the solutions to only four solutions to simplify this empirical study as many solutions only produced repetitive calculations. Table 4.3 shows the solutions *w, x, y* and *z* with their respective parameters and calculated fitness functions values

.

Table 4.3: Solutions and Their Fitness Function Values

.

Solution	Parameters	Imperceptibility (dB)	Robustness	Dynamic Security
w	[3,7,67,31,6]	59.8116	0.85	99999.1615
x	[29,1,69,85,8]	90.5202	0.58	99941.2009
y	[74,4,80,65,7]	78.6990	0.77	99999.0453
z	[261,7,63,55,8]	64.8591	0.84	89994.7674
Min		**59.8116**	**0.58**	**89994.7674**
Max		**90.5202**	**0.85**	**99999.1615**

$Best_Sol_{imperceptibility}$, for each solution were calculated using Equation 4.12 until Equation 4.15 as follows:

UNIVERSITI SAINS ISLAM MALAYSIA
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

$$Best_Sol\ (w)_{impercecptibility} = \frac{59.8116 - 59.8116}{90.5202 - 59.8116} = 0$$

(4.12)

$$Best_Sol\ (x)_{impercecptibility} = \frac{90.5202 - 59.8116}{90.5202 - 59.8116} = 1$$

(4.13)

$$Best_Sol\ (y)_{impercecptibility} = \frac{78.6990 - 59.8116}{90.5202 - 59.8116} = 0.6151$$

(4.14)

$$Best_Sol\ (z)_{impercecptibility} = \frac{64.8591 - 59.8116}{90.5202 - 59.8116} = 0.1644$$

(4.15)

Next, $Best_Sol_{robustness}$ for each solution were calculated using Equation 4.16 until Equation 4.19 as follows:

$$Best_Sol\ (w)_{robustness} = \frac{0.85 - 0.58}{0.85 - 0.58} = 1$$

(4.16)

$$Best_Sol\ (x)_{robustness} = \frac{0.58 - 0.58}{0.85 - 0.58} = 0$$

(4.17)

$$Best_Sol\ (y)_{robustness} = \frac{0.77 - 0.58}{0.85 - 0.58} = 0.7037$$

(4.18)

$$Best_Sol\ (z)_{robustness} = \frac{0.84 - 0.58}{0.85 - 0.58} = 0.9630$$

(4.19)

Next, $Best_Sol_{dynamic_security}$ for solution *w*, *x*, *y* and *z* were calculated using Equation 4.20 until Equation 4.23 as follows:

$$Best_Sol\ (w)_{dynamic_security} = \frac{99999.1615 - 89994.7674}{99999.1615 - 89994.7674} = 1$$

(4.20)

$$Best_{Sol}(x)_{dynamic_{security}} = \frac{99941.2009 - 89994.7674}{99999.1615 - 89994.7674} = 0.9942$$

(4.21)

UNIVERSITI SAINS ISLAM MALAYSIA
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

$$Best_{Sol}(x)_{dynamic_{security}} = \frac{99999.0453 - 89994.7674}{99999.1615 - 89994.7674} = 0.9999$$
(4.22)

$$Best_Sol\ (z)_{dynamic_security} = \frac{99999.0453 - 89994.7674}{99999.1615 - 89994.7674} = 0$$
(4.23)

$Best_Sol$ for each solution was calculated using Equation 4.24 until Equation 4.27 as follows:

$$Best_{Sol}(w) = (0 * 033) + (1 * 0.34) + (1 * 0.33) = 0.67$$
(4.24)

$$Best_Sol\ (x) = (1 * 033) + (0 * 0.34) + (0.9942 * 0.33) = 0.66$$
(4.25)

$$Best_{Sol}(y) = (0.6151 * 033) + (0.7037 * 0.34) + (0.9942 * 0.33) = 0.77$$
(4.26)

$$Best_{Sol}(z) = (0.1644 * 033) + (0.9630 * 0.34) + (0.9999 * 0.33) = 0.38$$
(4.27)

Table 4.4: Solutions and Their Normalized Fitness Function Values

Solution	Imperceptibility	Robustness	Dynamic Security	*Best_Sol*
w	0	1	1	0.67
x	1	0	0.9942	0.66
y	0.6151	0.7037	0.9999	0.77
z	0.1644	0.9630	0	0.38
Min	0	0	0	0.38
Max	1	1	1	0.77

The results from this calculation are shown in Table 4.4. Based on the result shown in this empirical study, solution *y* was the best solution compared to the solution *w, x* and *z*. Therefore, solution y was given to BCM-LSB to continue the process.

4.4 Difference between Proposed Algorithm and Existing

In this section, the difference between proposed algorithm with the existing algorithm. This section is introduced to help the reader to understand the difference by

comparing the parameter used between existing algorithm and the proposed one. The comparison is made in three level, which are: 1) the LSB embedding level, 2) cover selection level and 3) audio steganography model level. Figure 4.12 illustrates and highlights each level in general audio steganography setup.

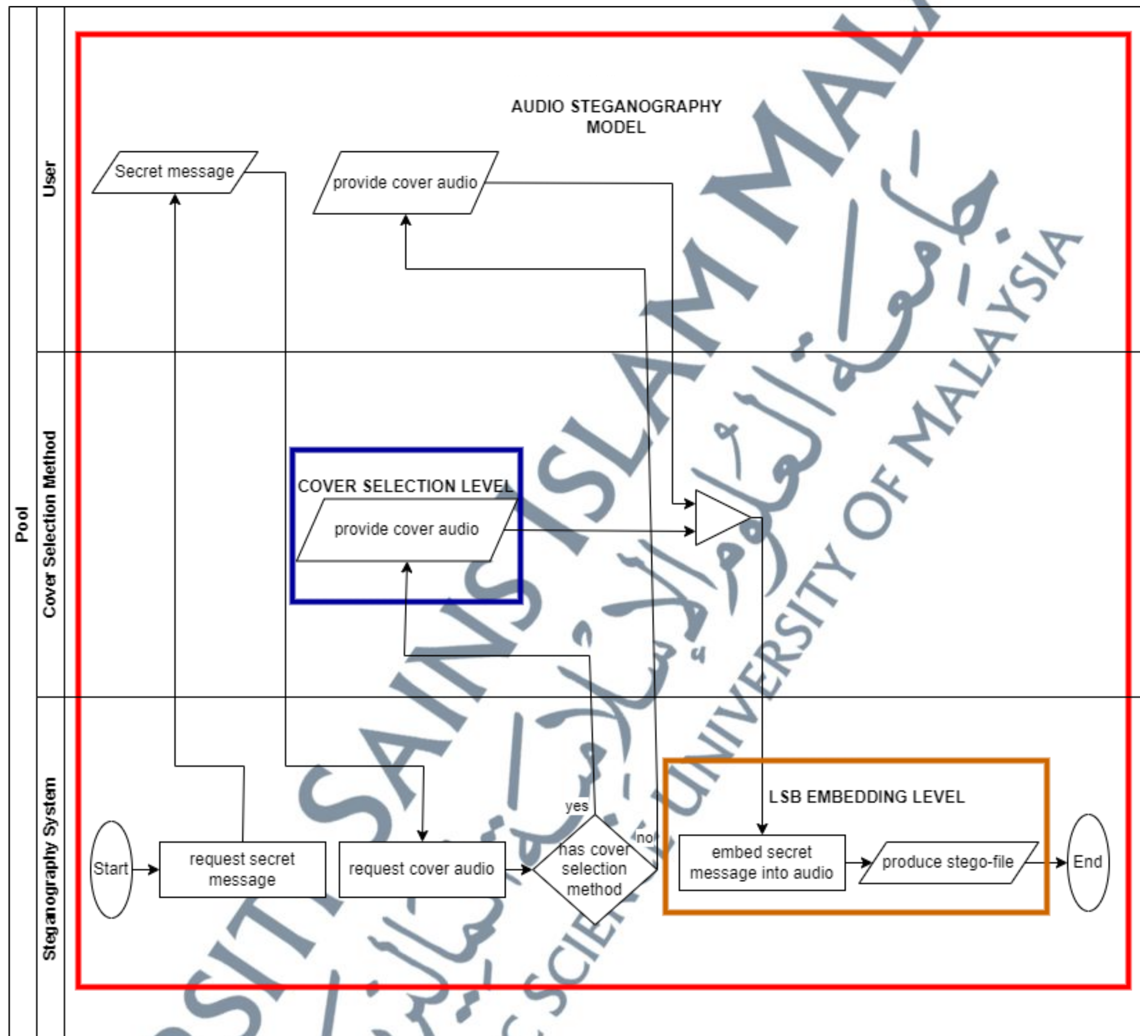


Figure 4.12: CAS Algorithm Flowchart

Based on Figure 4.12, the LSB embedding level is represented by the orange box while cover selection level is represented by the blue box. Lastly, audio steganography model is represented by the red box which covers all the processes including cover selection level and LSB embedding level.

UNIVERSITI SAINS ISLAM MALAYSIA
UNIVERSITY OF MALAYSIA
ISLAMIC SCIENCE

4.4.1 LSB Embedding Level

This subsection discusses the difference between BCM-LSB and existing LSB. This section is needed to highlight the difference between BCM-LSB and existing LSB as there are so many LSB were proposed in the past. This level focuses on the behaviours during embedding process of the secret message into the series audio samples. The existing LSB that are chosen for comparison are seven LSB which reflect each of seven main LSB approaches, and they are listed as follows:

- Indrayani (2020) reflects low bit embedding approach.
- Alsabhany et al. (2020) reflects variable low-bit embedding approach.
- Hashim et al. (2018) reflects bit selection embedding approach.
- Hosny et al. (2019) reflects sample selection embedding approach.
- Xin et al. (2018) reflects averaging amplitude threshold-based embedding.
- Usanto (2022) parity coding embedding approach.
- Bharti et al., (2019) XOR-based embedding approach.

Indrayani (2020), Hashim et al. (2018), Xin et al. (2018), Usanto (2022) and Bharti (2019) are chosen because they are the latest LSB based on LSB approaches while Alsabhany et al. (2020) and Hosny et al. (2019) are chosen because they produce the best result among LSB with similar approaches. The comparison was made by ignoring the preprocessing such as encryption and secret message compression and solely focuses on how these LSB algorithm manipulate the audio and do the embedding process. Figure 4.13 illustrates the general algorithm BCM-LSB and each LSB describes above.

a)

```
for i = 1; i<=totalsampleNeededEmbedding
    embed bps of the secret message into audio_sample_i
    i++;
end
```

b)

```
for j = 1; j < = max_bit_level
    for i = 1; i<=total_audio_samples
        if meet the embedding condition
            embed bps of the secret message into audio_sample_i
        end
    i++;
    end
j++;
end
```

c)

```
for i = 1; i<= totalsampleNeededEmbedding
    if meet the embedding condition
        embed bps of the secret message into audio_samplei_i
    end
i++;
end
```

d)

```
for i = 1; i<= totalsampleNeededEmbedding
    do comparison bps of the secret message with audio_sample_i
    if meet condition
            flip lsb audio_sample_n
    end
i++;
end
```

e)

```
for j = 1; j < = total_block_number
    for i = 1; i<= total_audio_samples_per_block
        embed bps of the secret message into audio_sample_i
        i++;
    end
    j++;
end
```

Figure 4.13: Embedding Algorithm Skeleton a) Hosny et al. (2019), Indrayani (2020), b) Alsabhany et al. (2020), c) Hashim et al. (2018), and Xin et al. (2018), d) Usanto (2022) and Bharti et al. (2019), e) BCM-LSB

Based on Figure 4.13, the main parameters difference between BCM-LSB and other LSB is the audio manipulating style and bit embedded. Table 4.5 provides the extensive comparison between BCM-LSB and existing LSB methods on embedding approach, bit Embedded and supported Elements based on Figure 4.13.

Table 4.5: Comparison Between BCM-LSB and Existing LSB Methods on Related Parameters

LSB	Audio Manipulating Style	Bit embedded	Embedding Condition
BCM-LSB	Manipulates audio by splitting audio into several smaller blocks	Up to 8 bits	No condition
Alsabhany et. al. (2020)	Manipulates whole audio directly	Up to 6 bits	Depends on amplitude value
Bharti et al. (2019)	Manipulates whole audio directly	1 bit	Depends on XOR results
Hashim et al. (2018)	whole audio	1 bit	Depends on MSB
Hosny et al. (2019)	whole audio	Up to 2 bits	No conditions
Indrayani (2020)	whole audio	Up to 8 bits	No condition
Usanto, (2022)	parity coding / whole audio	1 bit	Depends on parity value
Xin et. al. (2018)	averaging amplitude threshold-based whole audio	1 bit	Depends on amplitude value

The comparison results during evaluation stage on audio steganography characteristics are mainly impacted by the parameters presented above and further discussions are presented in Chapter 5.

4.4.2 Cover Selection Level

This subsection discusses the difference between MCAS and existing cover selection method. This section is needed to highlight the difference between MCAS and existing cover selection as there was no cover selection for audio in the past. The existing cover selection methods that are chosen for comparison are cover selection that used the metrics applicable and able to measure the audio steganography characteristics accurately. The existing cover selection are listed as follows:

- Rashid (2020)
- Xin & Jiaojiao (2018)
- Wang et al. (2020)

The main difference between the MCAS with the existing cover selection method is the MCAS considers the trade-off between the measured characteristics while the others only not consider the characteristic. Figure 4.14 illustrates cover selection by Rashid (2020), Figure 4.15 illustrates the cover selection by Xin & Jiaojioa (2018) and Figure 4.16 illustrates cover selection by Wang et al. (2020).

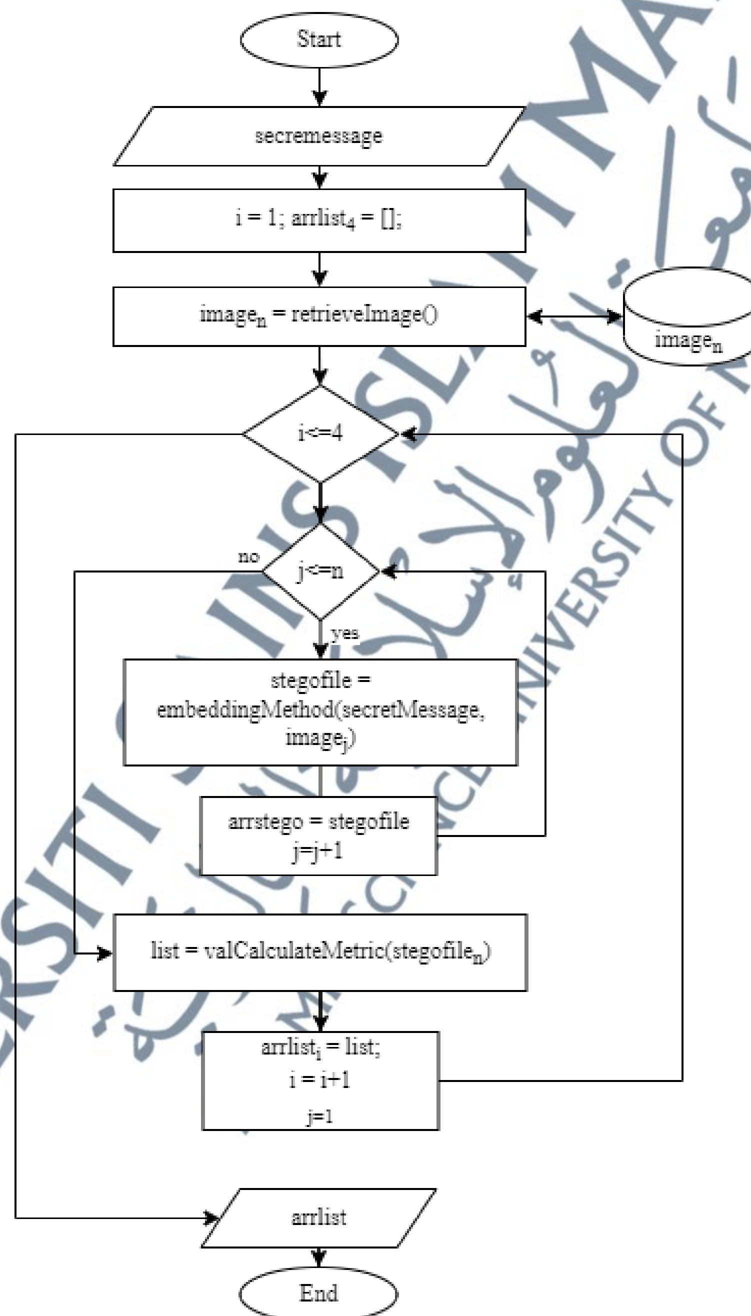


Figure 4.14: Rashid (2020) Algorithm Flowchart

UNIVERSITI ISLAM MALAYSIA

The cover selection algorithm proposed by Rashid (2020) illustrates in Figure 4.14 starts by taking user's secret message as an input. Then, it assigns the value of *i* = 1 and declare empty array for ranking lists, *arrlist* with size of 4. This size is indicated by the number of evaluation metrics which are PSNR, Different Image Histogram, (DIH), Revisited Weighted Stego (RWS) and match bits. For each evaluation metric, valCalculateMetric() is used to calculate the ranking of the image by checking the evaluation metric after the embedding process occurs using mebddingMethod() function which embed the secret message to all the image in the database. After that, this algorithm inserts the list to the array of ranking list, *arrlist*, This process repeated using other listed evaluation metrics. Before ending this algorithm, it returns the array of ranking list based on each evaluation metrics used as an output. The idea of this algorithm is producing the several list which can be used depends on which characteristic the user want to improve. However, as it does not consider other trade-off as it only calculate one metric per list which evaluate one characteristic which will reduce the overall performance of stego-file characteristic.

UNIVERSITI SAINS ISLAM MALAYSIA
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

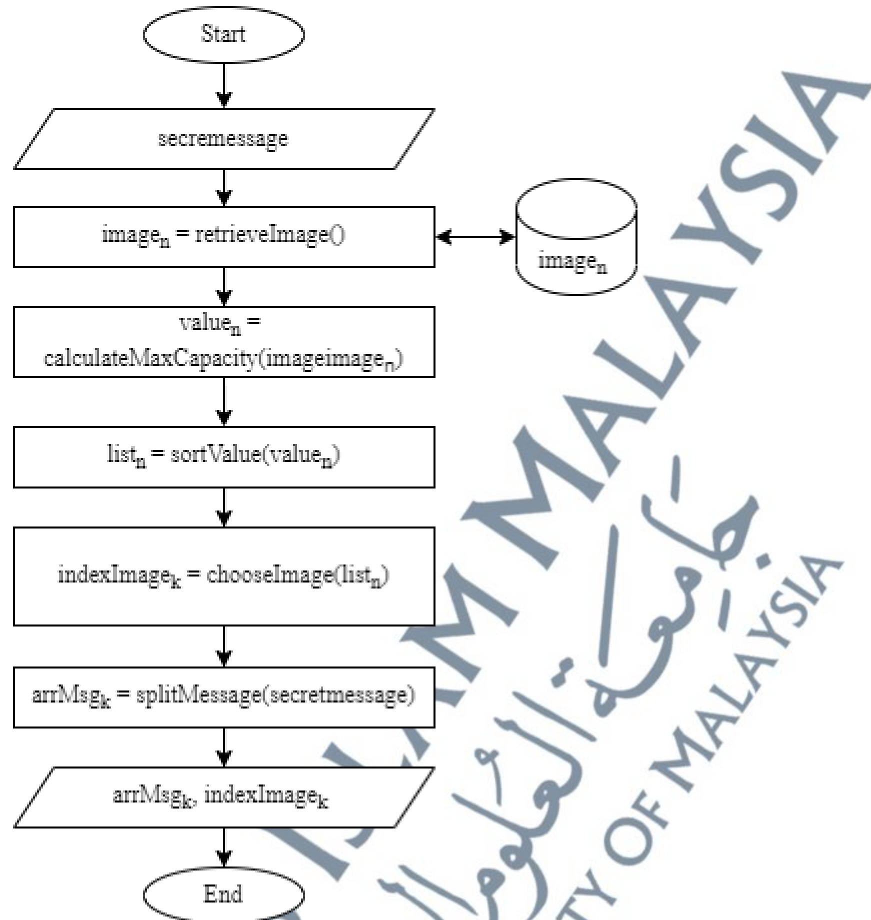


Figure 4.15: Xin and Jiaojiao (2018) Algorithm Flowchart

Based on Figure 4.15, the cover selection algorithm proposed by Xin and Jiaojiao (2018) starts by taking the secret message, *secretmessage* as an input. Next, it calculate maximum capacity for each image from the database by using calculateMaxCapacity() function and assigned it to array $value_n$ which used to store array of maximum capacity for each image. Next, the values inside this array is sorted in descending order in term of maximum capacity of secret message that can be embedded by using sortValue() function which then is assigned to variable $list_n$. Next, the several images is selected based on the $list_n$ and the index of selected image are stored in the form of array in $indexImage_k$ variable where k is the number of image selected. Next, the secret message is split into k amount and stored into $arrMsg_k$ using function splitMessage(). Lastly, this algorithm ends by producing $arrMsg_k$ and $indexImage_k$ as the output. The idea of this

algorithm is maximising the cover image to minimally the number of images used for the batch embedding process. Indirectly, by using this algorithm, the stego-file produced hypothetically has high dynamic security. However, as it does not consider other trade-off which will reduce the overall performance of stego-file characteristic.

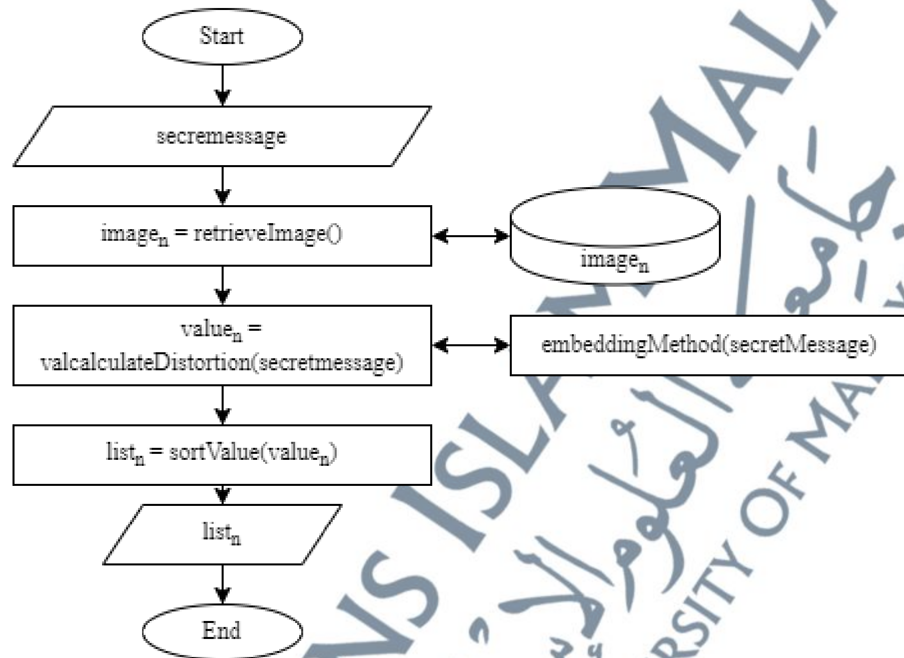


Figure 4.16: Wang et al. (2020) Algorithm Flowchart

Based on Figure 4.16, the cover selection algorithm proposed by Wang et al. (2020) starts by taking the secret message, *secretmessage* as an input. Next, it calculates the distortion value resulted for each embedding process by embeddingMethod() function for n number of images inside the database. The distortion is calculated by counting how many binary numbers are flipped throughout the embedding process. This distortion value is stored inside variable $value_n$. Then the value is sorted by the sortValue() function and stored in an array of variable $list_n$. the value is sorted in ascending order where the lowest distortion value is given priority for the embedding process which occur after this algorithm. Similar to two others cover selection

UNIVERSITI SAINS ISLAM MALAYSIA جامعة العلوم الإسلامية الماليزية ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

algorithm, this algorithm also does not consider the trade-off as it only focusses on the imperceptibility characteristic.

The differences between the proposed cover selection, MCAS and existing research are summarised in Table 4.6.

Table 4.6: Comparison Between MCAS and Existing Cover Selection Methods Related Parameters

Cover Selection Method	Type of Searching Method	Characteristic Evaluated	Metric Used	Cover Type	Number of ranking List provided
MCAS	Heuristic Search	Imperceptibility, robustness, dynamic security	SNR, BER after AWGN attack, newly proposed dynamic security evaluation metric	Audio	1
Rashid (2020)	Brute force	Imperceptibility, robustness	PSNR, Different Image Histogram, (DIH), Revisited Weighted Stego (RWS), Match bits	Image	4
L. Xin & Jiaojiao (2018)	Brute force	Dynamic security	Nearest maximum capacity	Image	1
Wang et al. (2020)	Brute force	Imperceptibility	Total combined distortion	Image	1

The comparison results during evaluation stage on cover selection are mainly impacted by the parameters presented above and further discussions are presented in Chapter 5.

4.4.3 Audio Steganography Model

This subsection discusses the difference between CAS and existing audio steganography models. Currently there is only one model for audio steganography which is a state-of-art model. Figure 4.17 illustrates the current state-of-art audio steganography model while Figure 4.18 illustrates the CAS audio steganography model.

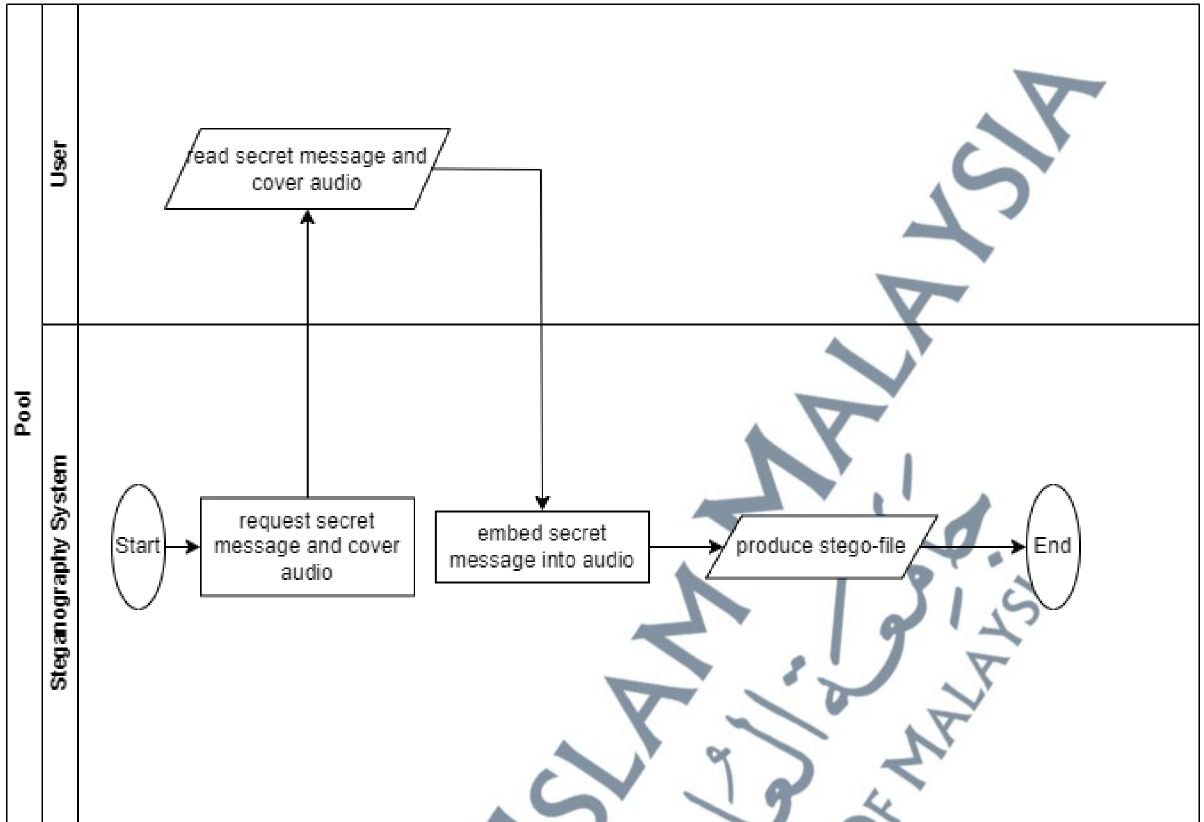


Figure 4.17: State-of-art Audio Steganography Model

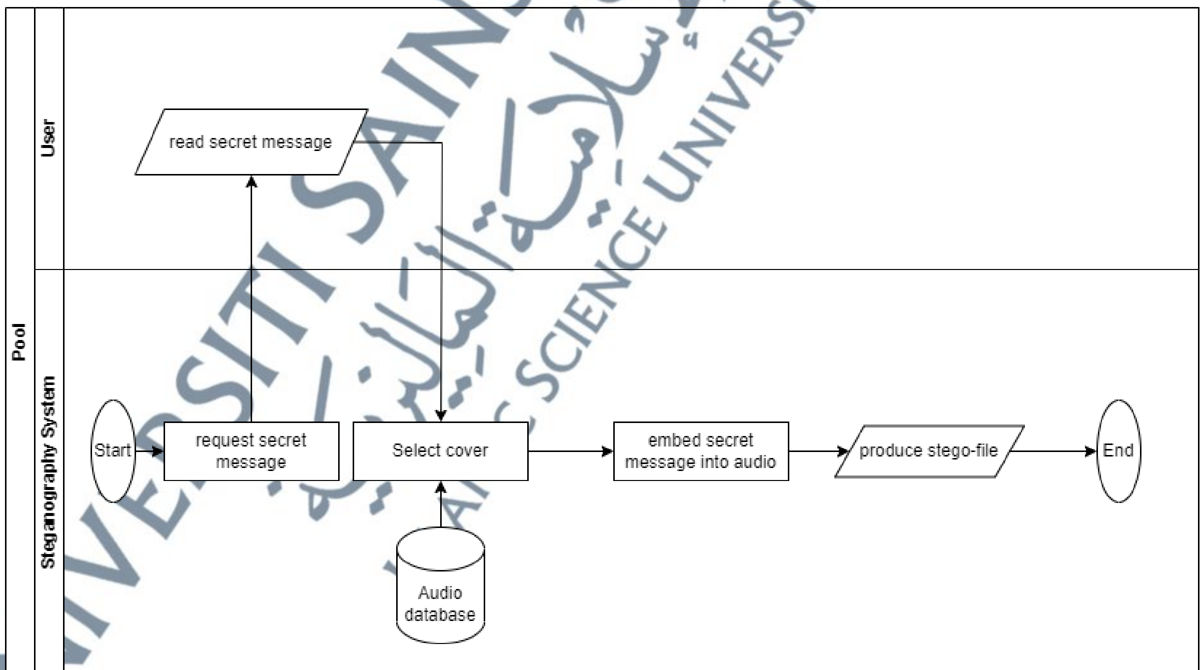


Figure 4.18: State-of-art Audio Steganography Model

The main difference between the state-of-art audio steganographic model in Figure 4.17 and the CAS model in Figure 4.18 is who selects the cover audio for the embedding process. The CAS algorithm suggests the cover audio for the user while in state-of-art audio steganography model depends on the user of steganography himself. CAS algorithm can find the best cover audio depends on the metric proposed and produced consistent good stego-file while current audio steganography model really depends on the user's knowledge which ranged from 'know nothing about steganography' to 'the expert in the field' hence produced inconsistent quality of stego-file. Therefor the CAS algorithm can aid the user to produce consistent quality stego-file.

The comparison results during evaluation stage between these two models based on parameter of who/what selects the cover audio is further discussed in Chapter 5.

4.5 Chapter Summary

In this chapter, the CAS algorithm is presented. The method consists of two algorithms named MCAS and BCM-LSB. BCM-LSB attempts to achieve one main goal which is improving dynamic security while MCAS attempts to achieve three main goals which are proposing a cover selection for the audio, introduction trade-off consideration during the selection process and lastly, introducing accurate dynamic security evaluation metric. LSB has been selected as a core technique for the BCM-LSB due to its high capacity and imperceptibility while the NSGA-II has been selected as a core technique for the MCAS because of low computational time and effectivity on smaller multi-objective problems. The CAS employs chaotic block, introducing metrics which are suitable to evaluate the audio, introducing the new dynamic security formulation and employing non-dominated sorting. Lastly, the difference between proposed

UNIVERSITI SAINS ISLAM MALAYSIA
جامعة العلوم الإسلامية الماليزية
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA

algorithm and existing one are discussed and presented to help user differentiate their used parameters.

UNIVERSITI SAINS ISLAM MALAYSIA
جامعة العلوم الإسلامية الماليزية
ISLAMIC SCIENCE UNIVERSITY OF MALAYSIA