IMT4124 Cryptology

Siegenthaler's correlation attack

Individual project task 3:

This report will discuss an implementation of Siegenthaler's attack as discussed in the report Individual project task 2. The implementation was written in MATLAB. To be able to run the implementation, having MATLAB installed is therefore a prerequisite.

The content of this report will first describe the implementation itself by describing the different functions and how they work. Secondly, user instructions describing how to run the implementation will be provided. Finally, the results of the implementation will be discussed, which includes how successful (or unsuccessful) the implementation was in performing Siegenthaler's correlation attack with varying parameters.

The implementation contains seven files; Geffes_generator.m (+Option1.m, Option2.m and Option3.m), Generate_noise.m, Test_initial_state.m and Siegenthalers_attack.m. The following paragraphs will explain what each of them do:

Geffes_generator.m (+Option1.m, Option2.m and Option3.m):

This function is, as the name implies, an implementation of Geffe's generator (task 1 in the project). The function uses feedback polynomials and initial states for each register and generates a key PN-sequence of a given length by running the output from each register through Geffe's non-linear combiner. The input to the function contains the feedback polynomials, the initial states, and the length. The output of the function is the PN-sequence. The tree functions Option1.m, Option2.m and Option3.m are implementations of other generators that might be better than Geffe's. They are further discussed in the report Individual project part 4.

Generate_noise.m:

This function generates the noise source, which together with the PN-sequence from the chosen generator will be used to generate the intercepted ciphertext. The function uses p_0 and length as parameters and creates a binary stream of the given length, where p_0 determines the probability that a particular symbol is 0.

Siegenthalers_attack.m:

This function contains the majority of the logic of the attack. In broad terms, the function generates the intercepted ciphertext, determines the parameters that will be used in the attack, creates the probability distributions for our two hypotheses, calculates the threshold T, finds the candidate states, tries to determine which of the candidates is the correct one and reports the results. More about this function will be mentioned in the section providing user instructions.

Test_initial_state.m:

This function is used by Siegenthalers_attack.m to determine the correlation measure α. The function takes the intercepted ciphertext, the initial state you want to test, and the feedback polynomial used by the register you are trying to attack as input. The correlation measure is then calculated using the formula described in (Siegenthaler, 1985).

User instructions:

This section will describe how to run the implementation of the attack and explain which configurations one can do to alter the parameters of the attack. An example run will also be given, using the same parameters used in the example given in the previous report (Individual project task 2).

Running the attack with default parameters is simple. You just need to open the file Siegenthalers_attack.m in MATLAB and press the “Run”-button. As the attack goes on, relevant information will be displayed in MATLAB’s command window.

If you want to modify the parameters, you can alter the following variables in the same file (Siegenthalers_attack.m):

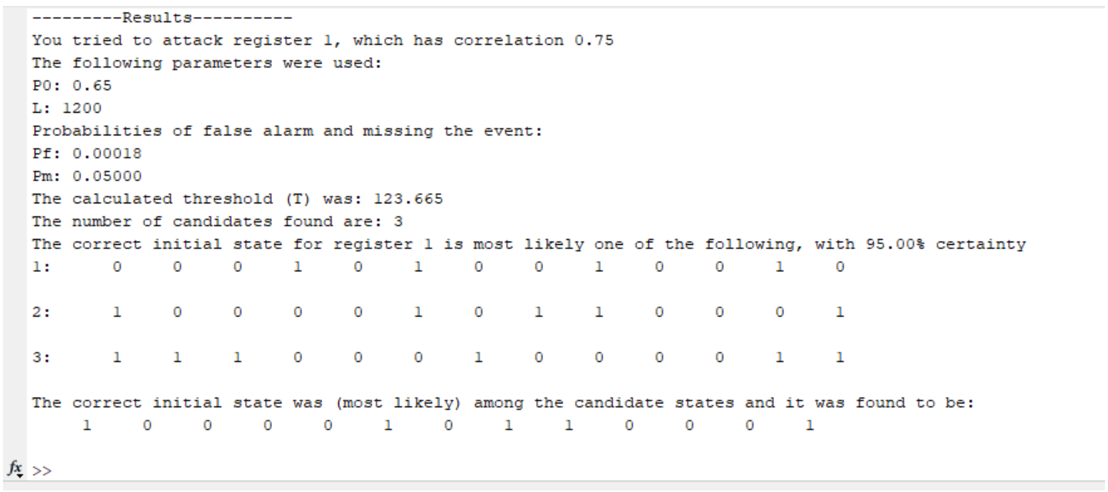
> feedback_polynomials; determines the feedback polynomials that will be used in the chosen generator.
>
> initial_states; determines the initial states that will be used in the registers in the chosen generator.
>
> Combiner_to_use; determines which non-linear combiner the PN-generator should use. Viable options are “Geffes_generator”, “Option1”, ”Option2” and “Option3”.
>
> register_to_attack; determines which register to attack.
>
> q; the correlation of the register you want to attack
>
> pm; determines the target value for probability of missing the event.
>
> pf; determines the target value for probability of a false alarm.
>
> p0; determines the probability that a particular symbol in the noise sequence will be 0.
>
> L; determines the length of the intercepted ciphertext.
>
> ADDITIONAL_CIPHERTEXT_FOR_TESTING; determines the amount of additional intercepted ciphertext bits that will be used to test the candidate states. It can be set to 0, but the certainty of the decision to determine the correct initial state increases with higher values. I sat the default as 1000.

Alterations of these variables can be done at will, but there are some limitations:

1. If the feedback polynomials are changed so that the length of the registers change, the initial states must be changed accordingly to the proper length.
2. The register to attack must be a vulnerable one. In this case, this means register 1 or 3. Register 2 cannot be attacked due to no correlation (q=0.5). If register_to_attack is modified, the variable q must also be changed accordingly (not necessary in this case as it should be 0.75 for both register 1 and 3).
3. Only one of the variables pm and pf can be used as a target value. If not, it can become impossible to determine the threshold T. Set one of them to a wanted value and leave the other at 0 (default).
4. p0 cannot be 0.5 due to reasons mentioned in the previous report (Individual project task 2).
5. If L is too low, it might become impossible to determine a proper threshold. How low it can be is based on the value of the other parameters.

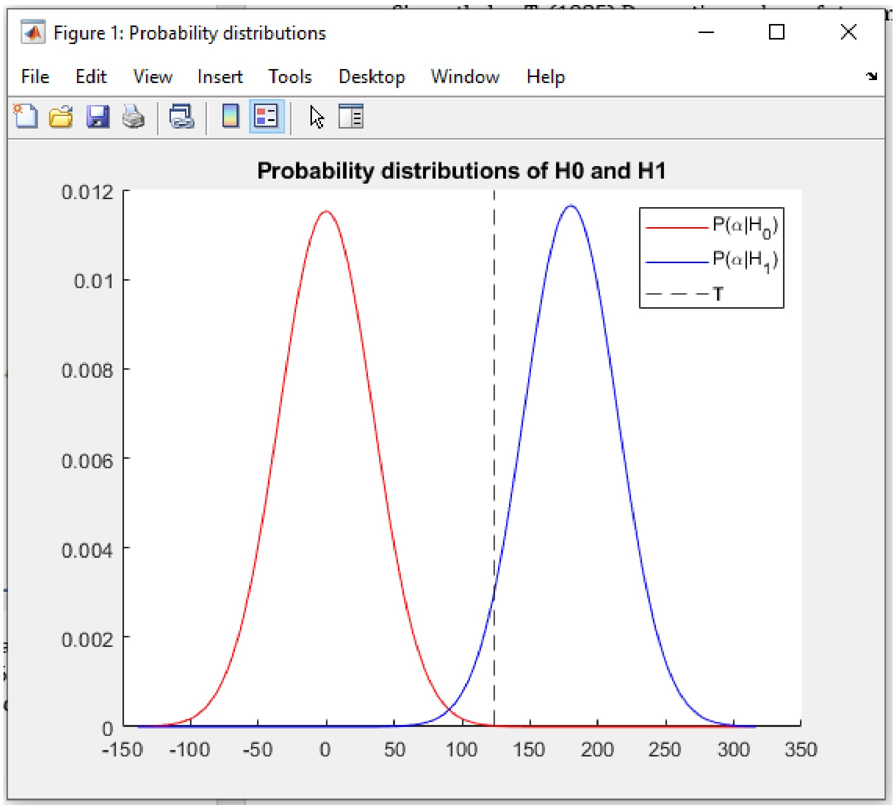
An example run of the attack will now be presented to see how it works in practice. The example will use the same parameters used in the example given in the previous report (Individual project task 2):

The parameters were: register_to_attack=1, q=0.75, p_0=0.65, L=1200 and p_m=0.05. This resulted in the following results:



```
---------Results----------
You tried to attack register 1, which has correlation 0.75
The following parameters were used:
P0: 0.65
L: 1200
Probabilities of false alarm and missing the event:
Pf: 0.00018
Pm: 0.05000
The calculated threshold (T) was: 123.665
The number of candidates found are: 3
The correct initial state for register 1 is most likely one of the following, with 95.00% certainty
1:      0     0     0     1     0     1     0     0     1     0     0     1     0

2:      1     0     0     0     0     1     0     1     1     0     0     0     1

3:      1     1     1     0     0     0     1     0     0     0     0     1     1

The correct initial state was (most likely) among the candidate states and it was found to be:
     1     0     0     0     0     1     0     1     1     0     0     0     1

fx >>
```

As seen in the image, this attack was a successful one. The correct initial state was found and there was a low number of false positives (two). The threshold T was determined to be 123.665, which is the same as was found in the example in the previous report. The results also produced a visualization of our two probability distributions, as seen below:



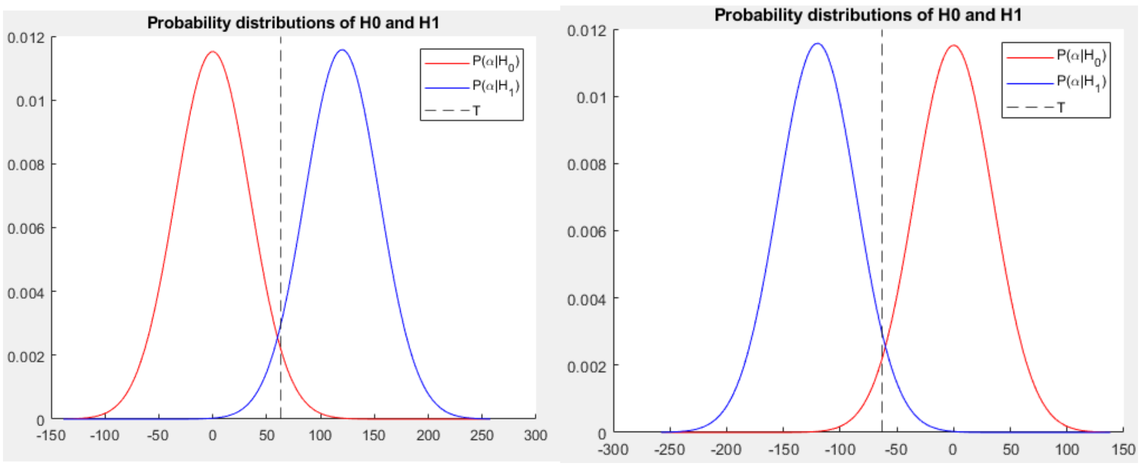
As seen, the two distributions are clearly separated, which makes the attack relatively easy. This is confirmed by the fact that we only got one false positive.

This concludes the example, and the final part of this report will display and discuss the results of several attacks using varying parameters. All the attacks are performed against register 1, with correlation 0.75. The table below displays a summary of these results:

p_m	p_f	p_0	L	T	Number of candidates:	False positives:	Correct initial state was among the candidate states:
0.00048	0.001	0.3	1000	-97.722	11	10	Yes
0.00000	0.001	0.3	5000	-218.512	8	7	Yes
0.00000	0.001	0.3	10000	-309.023	9	8	Yes
0.47114	0.001	0.4	1000	-97.722	8	7	Yes
0.00003	0.001	0.4	5000	-218.512	8	7	Yes
0.00000	0.001	0.4	10000	-309.023	15	14	Yes
0.48947	0.06*	0.45	1000	-49.166	518	517	Yes
0.32785	0.001	0.45	5000	-218.512	8	8	No
0.02793	0.001	0.45	10000	-309.023	6	5	Yes
0.48947	0.06*	0.55	1000	49.166	499	499	No
0.32785	0.001	0.55	5000	218.512	5	5	No
0.02793	0.001	0.55	10000	309.023	12	11	Yes
0.47114	0.001	0.6	1000	97.722	7	7	No
0.00003	0.001	0.6	5000	218.512	7	6	Yes
0.00000	0.001	0.6	10000	309.023	9	8	Yes
0.00048	0.001	0.7	1000	97.722	14	13	Yes
0.00000	0.001	0.7	5000	218.512	6	5	Yes
0.00000	0.001	0.7	10000	309.023	4	3	Yes
0.001	0.00049	0.3	1000	-104.253	1	0	Yes
0.001	0.00000	0.3	5000	-785.902	1	0	Yes
0.001	0.00000	0.3	10000	-1697.220	1	0	Yes
0.001	0.46512	0.4	1000	-2.678	3788	3787	Yes
0.001	0.00003	0.4	5000	-282.583	1	0	Yes
0.001	0.00000	0.4	10000	-692.526	1	0	Yes
0.06*	0.48871	0.45	1000	-0.895	3981	3980	Yes
0.001	0.32666	0.45	5000	-31.761	2727	2726	Yes
0.001	0.02783	0.45	10000	-191.363	217	216	Yes
0.06*	0.48871	0.55	1000	0.895	3985	3984	Yes
0.001	0.32666	0.55	5000	31.761	2719	2718	Yes
0.001	0.02783	0.55	10000	191.363	221	220	Yes
0.001	0.46512	0.6	1000	2.768	3753	3752	Yes
0.001	0.00003	0.6	5000	282.583	1	0	Yes
0.001	0.00000	0.6	10000	692.526	1	0	Yes
0.001	0.00049	0.7	1000	104.253	4	3	Yes
0.001	0.00000	0.7	5000	785.902	1	0	Yes
0.001	0.00000	0.7	10000	1697.220	1	0	Yes

*p_m or p_f of 0.001 was too low to determine a proper threshold when L was 1000 and p_0 was close to 0.5. The value was therefore increased to 0.06.

As seen in the table, several observations can be made. The first thing one might notice is that the threshold T increases/decreases symmetrically as p_0 moves up/down from 0.5. This is not surprising as p_0=0.4 and p_0=0.6 imply the same statistical properties of the noise sequence, and the only difference is the ratio between 0s and 1s. This change in p_0 also alters the relative positions of the probability distributions as the mean of $P(\alpha|H_1)$ becomes negative if p_0<0.5 and positive if $p_0>0.5$. The distance between the distributions are however still the same, so there are no practical implications on the attack. This is also displayed in the graphs below:



Distributions for the hypotheses when $p_0=0.6$ and $p_0=0.4$ respectively.

Another observation is that the attack is trivial if p_0 is far from 0.5, even if L is low. When $p_0=0.3$ or $p_0=0.7$, it was possible to obtain very low values for p_m and p_f, even if L was only 1000.

The attack became trickier as p_0 approached 0.5. If L was low, it was often necessary to compromise either p_m or p_f. As an example, we can look at the situation when L=1000 and $p_0=0.55$. If we aimed for a low value for p_f (0.06), we were able to obtain relatievly few false positives (499), but as a result, p_m became high (0.48947) and due to this, the correct initial state was not among the candidates and the attack should be considered a failure. Alternatively, if we aimed for low p_m (0.006), we would with high confidence be able to not miss the correct initial state, but as a result, p_f grew to 0.48871, and there was thus found 3894 false positives. This is also not ideal. When L is low, it is therefore necessary to determine wheter you want to prioritize p_f or p_m, as it were impossible to obtain good values for both.

However, if L was longer (10000), the attack would be effective even if p_0 was close to 0.5. It was then possible to obtain p_f and p_m <0.03.

Based on the overall test results, it seems like the value of L is the most crucial parameter. By increasing L, p_m and p_f would consequently decrease. To increase the probability of a successful attack against Geffe's generator, the most important step should therefore be to intercept enough ciphertext.

References:

Siegenthaler, T. (1985) Decrypting a class of stream ciphers using ciphertext only, *IEEE Transactions on computers,* (1), pp. 81-85