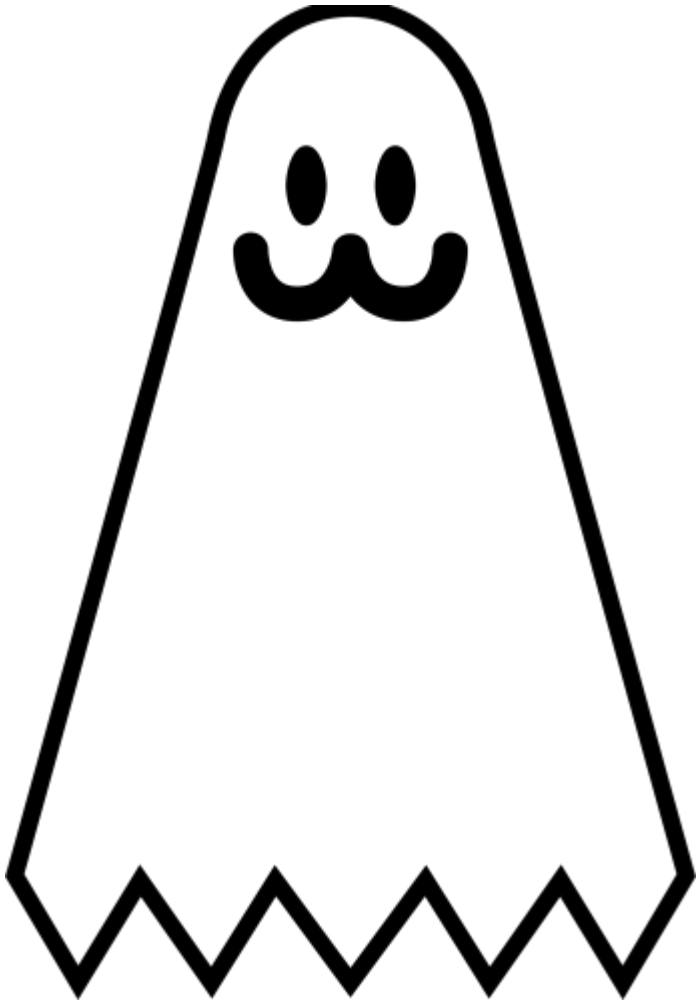
# ANGELITA: A 128-bit SPN Block Cipher

Created and tested by stringzzz
Ghostwarez Co. :3
August 8th, 2022



## Background

I have no doubt about it, I'm an amateur cryptologist. I create these systems for the educational experience. ANGELITA happens to be my best system so far, but I have many limitations. Currently, I am highly lacking in the cryptanalysis department, so I can only run basic tests on this system. That being said, I am currently going to a community college. I am getting set up to transfer to a University, for a degree in Computer Science. My goal is to eventually become a full-fledged cryptologist, learning both Computer Science and Mathematics. I recognize that I still have so much to learn, but ANGELITA is the result of what I have learned so far. Just to note, this encryption algorithm should only be used for testing purposes, not for any real secure application needs. The code is written with many simple tests built in, and these could easily be expanded on to test further. ANGELITA is actually an acronym, it stands for “Algorithm of Number Generation and Encryption Lightweight Intersperse Transform Automator”, a mouthful, I know. One last thing to note before moving on, is that the GLORIA prng built into this algorithm is definitely not secure. It relies on seeding the built-in prng of whatever programming language it is implemented in, which is not a cryptographically good idea. The real reasoning behind it is just to make testing easier, to generate thousands of keys and plaintexts.

## The Design

The idea behind ANGELITA's design was to make the S-Box and P-Box key dependent, so they would be unknown to any attacker. The goal was also to make the encryption/decryption fast, though it seems to fall short in that regard. The Key Schedule expands on the key, and parts of that same Key Schedule are used to generate the S-Box and P-Box by repeated shuffling that is dependent on the bits of the Key Schedule. The rest of the Key Schedule is used for XORing in the encryption/decryption loop. Due to some test results showing bad properties, this system went through multiple changes. Each test will have its own section, explaining the test and the results. There were many mistakes made in the testing process, so the only valid tests are the ones done with ANGELITA3 and 4B.

# The Algorithm Overall

1. Call ANGELITA_KISS (Key Initialize Scheduling Subroutine) to expand the initial key 128 times into the Key Schedule. (This routine will be explained in more detail later on page 3)
2. Split the Key Schedule into 4 parts, 1216 bytes for the 8x8 S-Box generation, 320 bytes for the nybbles P-Box generation, and 2 sets of 256 bytes for the XORs in the encryption/decryption routine.
3. Call TeaParty2 with the Key Schedule S-Box bytes to generate the S-Box outputs through repeated key-dependent shuffling. (TeaParty2 is explained on page 4)
4. Call TeaParty2, this time with the Key Schedule bytes for the P-Box, that works on a set of nybbles. (32 nybbles input/output)
5. For each plaintext block, do the following (Encryption):
   Repeat the following 16 times (n):
   (a.) S-Box each plaintext byte.
   (b.) XOR each plaintext byte with a unique byte from Key Schedule set 1
   (c.) S-Box each plaintext byte.
   (d.) XOR each plaintext byte with a unique byte from Key Schedule set 2
   (e.) If cycle count is a multiple of 4 and not 0, P-Box the nybbles of the plaintext.
6. Output the ciphertext.

Note that decryption is simply the reverse of the step 5 encryption. This was version 1, version 3 had these differences:

5. For each plaintext block, do the following (Encryption):
   Repeat the following 16 times (n):
   (a.) XOR each plaintext byte with a unique byte from Key Schedule set 1
   (b.) S-Box each plaintext byte.
   (c.) XOR each plaintext byte with a unique byte from Key Schedule set 2
   (d.) If cycle count is a multiple of 2 and not 0, P-Box the nybbles of the plaintext.

Version 4B was exactly the same, except that the P-Box was now twice the size and worked on pairs of bits.

# Key Schedule

1. For each byte n in the initial key,
   (a.) make a copy of the initial key.
   (b.) XOR n with all bytes in the copy, except for the one that n came from. (Result is 256 bytes)
2. Place the previous 256 bytes in the temporary Key Schedule.
3. Repeat 7 times:
   (a.) Take the previous 256 bytes and rotate each byte's leftmost bit to the right
   (b.) Add these 256 bytes to Key Schedule
4. Repeat the following twice:
   (a.) Use the first 320 bytes of the temp Key Schedule with TeaParty2 to make a P-Box.
   (b.) Run each block of 16 bits through the P-Box (As nybbles).
   (c.) Use the first 1216 bytes from the temp Key Schedule with TeaParty2 to make an S-Box.
   (d.) Run all bytes of the temp Key Schedule through the S-Box
5. The results are the final Key Schedule.

   Version 3 added an extra set of steps in KISS2:

6. XOR all 16-bit blocks of the temporary Key Schedule together to make one 16-bit block.
7. Store the current temp Key Schedule aside. (KS1)
8. Set the resulting block of step 6 as the initial key.
9. Use KISS to generate another temp Key Schedule (KS2) from the key, store into the Key Schedule.
10. Split up the KS2 Key Schedule into parts for the S-Box, P-Box, and XORs
11. Generate the S-Box and P-Box from those KS2 parts.
12. Using the last KS1 block as the IV, encrypt the entire KS1 in CBC-Mode, using the normal encryption routine.
13. Discard the IV, the rest of the encrypted KS1 is the final Key Schedule.

This additional set of steps was added to attempt to reduce the number of weak keys that would generate a Key Schedule with several repeated patterns. There are now only 16 known weak keys, which consist of all their nybbles being the same values, such as (In Hexadecimal): FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF

## TeaParty2 Shuffling Routine

1. Repeat the following 38 times for S-Box, 80 times for P-Box:
   (a.) For each byte in the box, check each Key Schedule bit:
   - If the bit is a 1, place the current box byte in TeaCup1
   - If the bit is a 0, place the current box byte in TeaCup2

   (b.) Concatenate TeaCup2 with TeaCup1 (In that order)
2. The final set of bytes is the output values for the box

In Version 4B, due to the doubling of the P-Box size, TeaParty2 now repeats 40 times for the P-Box instead of 80.

## The Analysis of ANGELITA3 & 4B

### General Difference

This test took 7500 plaintext blocks generated by the GLORIA prng, encrypted them with a GLORIA key, and then the differences between the plaintext and ciphertext were gathered. Here are the resulting averages for that test:

ANGELITA3: 64.0018/128 bits of difference
ANGELITA4B: 64.0084/128 bits of difference

So, about the same, right in the middle. Here's the highs and lows of both of those tests:

| | High | Low |
|---|---|---|
| ANGELITA3: | 90/128 | 38/128 |
| ANGELITA4B: | 91/128 | 39/128 |

The small difference between the 2 versions doesn't necessarily mean much, but it should be pointed out that it would likely be better if the high and low were closer to the average. This is a re-occurring theme in these tests, so it won't be pointed out again.

### Plaintext Avalanche Test

This is the test that called for a new version, due to a very bad property found in version 3. 50,000 plaintext blocks were generated with GLORIA. A different GLORIA key was generated for each block. Each block was encrypted, storing the ciphertext aside. Then, for each of the 128 bits in the block, that same initial plaintext block was copied, and one of its bits were flipped (For 128 different resulting blocks). Each of

these new blocks were encrypted with the same key as the initial block, and then the difference between each one and the initial ciphertext were calculated. As an added test, if any of these differences were just 1 bit, a counter was incremented and the key, S-Box, P-Box, Key Schedule, initial plaintext/ciphertext, and new plaintext/ciphertext were recorded in a file. These are the results of the tests for Version 3 and 4B:

| | Average Difference | High | Low | 1-Bits |
|---|---|---|---|---|
| ANGELITA3: | 63.124/128 bits | 90/128 | 1/128 | 365/50,000 |
| ANGELITA4B: | 64.0073/128 bits | 89/128 | 39/128 | 0/50,000 |

This test had been expanded from another in Version 3 after seeing that low-case scenario of only 1 bit of change. As shown, this was a re-occurring problem. It happened so many times, that it called for a change. ANGELITA4B came from this change, where the size of the P-Box was doubled, and it's input/output were now 2-bits instead of nybbles. This change made a major impact, as shown in the low-case scenario of this test for ANGELITA4B. It would be a lot better if the high and low was closer to the average, but it is now fairly balanced.

**Pop Count Tests**

There are three of these tests, to count the number of ones in the ciphertexts. The first test generated 500 plaintext blocks, and encrypted each one with 50 different keys. A count of the 1 bits in the ciphertext was then calculated, and an average of all these counts were gathered, along with the low and high counts. Here are the results:

| | Average Count | High | Low |
|---|---|---|---|
| ANGELITA3: | 64.0601/128 | 86/128 | 40/128 |
| ANGELITA4B: | 64.0347/128 | 86/128 | 42/128 |

Middle average, high and low could be better. The next test was similar, but the plaintext block started with all zeroes, and it was encrypted with 20,000 different keys, then the 1 bits in the ciphertexts were counted:

| | Average Count | High | Low |
|---|---|---|---|
| ANGELITA3: | 63.959/128 | 84/128 | 44/128 |
| ANGELITA4B: | 64.0623/128 | 85/128 | 42/128 |

The differences are so minute, and it is likely that repeating the tests would yield slightly different highs and lows. The next test is the same as the last test, except that the plaintext block always started with all 1s, encrypted with 20,000 keys, then the 1 bits in the ciphertexts were counted:

| | Average Count | High | Low |
|---|---|---|---|
| ANGELITA3: | 63.935/128 | 86/128 | 43/128 |
| ANGELITA4B: | 63.9104/128 | 86/128 | 42/128 |

Almost exactly the same as the previous test. It is a theme in these tests that the average is almost right in the middle. As mentioned, the highs and lows could be better, but it all seems to be pretty balanced.

Another test related to these was done, but it wasn't very thorough. A file was encrypted in ECB mode, and then the resulting encrypted file was compressed. The hope was that there would be no redundancy to remove, which actually was the case in CBC mode (Unsurprisingly), but in both ANGELITA3 and 4B with ECB mode, there was about 90,000 bytes of redundancy to remove. This is not ideal at all, a sound encryption algorithm should remove all or at least almost all redundancy during encryption. At this time, I am uncertain how to solve that problem.

## S-Box Difference Test

For this test, 40,000 different keys were generated, their resulting Key Schedules were used to generate the 8x8 S-Boxes. Then, a difference between the inputs (0-255) and the outputs were calculated, with an average, lows, and highs, as usual. Here are the results:

| | Average Difference | High | Low |
|---|---|---|---|
| ANGELITA3: | 1023.98/2048 | 1124/2048 | 940/2048 |
| ANGELITA4B: | 1023.81/2048 | 1114/2048 | 938/2048 |

This follows the trend of the other properties of the algorithm. It seems kind of surprising that the lows and highs aren't scattered around more, considering that the S-Box is only generated by shuffling of the initial 0-255 valued bytes. And even though the shuffling is dependent on the Key Schedule bits, I would expect the results to be more random-like, with lower lows and higher highs. Not sure how to explain the details of these results at this time.

## Key Avalanche Test

Last but not least, this test was done to see what difference 1 bit of change in the key causes in the ciphertext. 1000 keys were generated with GLORIA, along with a plaintext block for each of them. For each key, a block was encrypted with the key, and then the ciphertext was stored away. Then, like the plaintext avalanche test, for each 128

bits in the key, a copy of the key was made with 1 bit flipped. The same plaintext was encrypted with each of these keys, and their resulting ciphertexts were compared with the initial ciphertext for differences. Same as before, an average, high, and low were output as a result.

| | Average Difference | High | Low |
|---|---|---|---|
| ANGELITA3: | 63.9909/128 | 88/128 | 40/128 |
| ANGELITA4B: | 64.0184/128 | 89/128 | 37/128 |

Fairly balanced, in parallel to the other tests.

## Individual Avalanche (ANGELITA4B)

The first part of this test compared the XOR difference between the plaintexts before and after each separate encryption operation, such as the P-Box, XORs, and S-Box. 25,000 pseudo-randomly generated blocks were used in this test. Here are the results, as the average, low case and high case measured in bits of difference:

| | | | | | |
|---|---|---|---|---|---|
| Individual P-Box Differences: | 62.983/128 | High: | 88/128 | Low: | 40/128 |
| Individual Xor 1 Differences: | 64.0025/128 | High: | 90/128 | Low: | 42/128 |
| Individual S-Box Differences: | 64.0194/128 | High: | 87/128 | Low: | 36/128 |
| Individual Xor 2 Differences: | 64.0107/128 | High: | 87/128 | Low: | 36/128 |

The other part of this same test measured the XOR difference of each block after each operation compared to the original plaintext block. This is shown for each cycle:

Cycle: 1/16
P-Box: Skipped cycle Xor 1: 64.1538/128 S-Box: 63.9494/128 Xor 2: 63.842/128

Cycle: 2/16
P-Box: 63.9236/128 Xor 1: 63.9862/128 S-Box: 63.916/128 Xor 2: 63.9438/128

Cycle: 3/16
P-Box: Skipped cycle Xor 1: 64.0846/128 S-Box: 63.9236/128 Xor 2: 64.0173/128

Cycle: 4/16
P-Box: 63.9694/128 Xor 1: 64.011/128 S-Box: 64.009/128 Xor 2: 64.0905/128

Cycle: 5/16
P-Box: Skipped cycle Xor 1: 64.0154/128 S-Box: 64.0729/128 Xor 2: 63.9858/128

Cycle: 6/16
P-Box: 63.9632/128 Xor 1: 63.8895/128 S-Box: 64.0944/128 Xor 2: 63.9395/128

Cycle: 7/16
P-Box: Skipped cycle Xor 1: 64.0292/128 S-Box: 64.038/128 Xor 2: 64.1014/128

Cycle: 8/16
P-Box: 63.8272/128 Xor 1: 64.0768/128 S-Box: 63.8794/128 Xor 2: 64.0234/128

Cycle: 9/16
P-Box: Skipped cycle Xor 1: 64.0568/128 S-Box: 64.0526/128 Xor 2: 64.1759/128

Cycle: 10/16
P-Box: 63.8666/128 Xor 1: 64.0912/128 S-Box: 63.9425/128 Xor 2: 64.1084/128

Cycle: 11/16
P-Box: Skipped cycle Xor 1: 64.0876/128 S-Box: 64.1041/128 Xor 2: 64.085/128

Cycle: 12/16
P-Box: 63.9766/128 Xor 1: 63.9228/128 S-Box: 63.9903/128 Xor 2: 63.8622/128

Cycle: 13/16
P-Box: Skipped cycle Xor 1: 64.082/128 S-Box: 64.093/128 Xor 2: 63.7755/128

Cycle: 14/16
P-Box: 63.8927/128 Xor 1: 63.956/128 S-Box: 63.9948/128 Xor 2: 64.017/128

Cycle: 15/16
P-Box: Skipped cycle Xor 1: 64.2045/128 S-Box: 63.87/128 Xor 2: 64.0554/128

Cycle: 16/16
P-Box: 63.9457/128 Xor 1: 64.0686/128 S-Box: 64.1038/128 Xor 2: 63.9196/128

As shown, the changed bits tend toward the half of the block. These averages seem to show that the algorithm is pretty balanced throughout each cycle. The next test shows a more interesting process.

## Individual Plaintext Avalanche (ANGELITA4B)

10,000 pseudo-random plaintext blocks were generated. For each one, it was encrypted with a unique key, and copies of the block after each operation through each cycle were logged. Then, also for each block, 128 copies were generated, one for each bit in the block. Each copy had one different bit flipped. Finally, each copy was encrypted with the same key that the original block they were copied from was encrypted with. As it went through the encryption algorithm, the blocks after each operation were compared with the matching blocks of the original encryption. In this way, it was possible to track the changes as they went through the algorithm. The results

are shown in terms of differences in bits:

Cycle: 1/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 1/128 High: 1 Low: 1
S-Box Differences: 4.00799/128 High: 8 Low: 1
Xor 2 Differences: 4.00799/128 High: 8 Low: 1

Cycle: 2/16
P-Box Differences: 4.00799/128 High: 8 Low: 1
Xor 1 Differences: 4.00799/128 High: 8 Low: 1
S-Box Differences: 11.4451/128 High: 28 Low: 1
Xor 2 Differences: 11.4451/128 High: 28 Low: 1

Cycle: 3/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 11.4451/128 High: 28 Low: 1
S-Box Differences: 11.4511/128 High: 28 Low: 1
Xor 2 Differences: 11.4511/128 High: 28 Low: 1

Cycle: 4/16
P-Box Differences: 11.4511/128 High: 28 Low: 1
Xor 1 Differences: 11.4511/128 High: 28 Low: 1
S-Box Differences: 28.089/128 High: 69 Low: 1
Xor 2 Differences: 28.089/128 High: 69 Low: 1

Cycle: 5/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 28.089/128 High: 69 Low: 1
S-Box Differences: 28.0739/128 High: 68 Low: 1
Xor 2 Differences: 28.0739/128 High: 68 Low: 1

Cycle: 6/16
P-Box Differences: 28.0739/128 High: 68 Low: 1
Xor 1 Differences: 28.0739/128 High: 68 Low: 1
S-Box Differences: 49.7694/128 High: 88 Low: 1
Xor 2 Differences: 49.7694/128 High: 88 Low: 1

Cycle: 7/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 49.7694/128 High: 88 Low: 1
S-Box Differences: 49.7383/128 High: 89 Low: 1
Xor 2 Differences: 49.7383/128 High: 89 Low: 1

(Continued on next page)

Cycle: 8/16
P-Box Differences: 49.7383/128 High: 89 Low: 1
Xor 1 Differences: 49.7383/128 High: 89 Low: 1
S-Box Differences: 61.1922/128 High: 90 Low: 2
Xor 2 Differences: 61.1922/128 High: 90 Low: 2

Cycle: 9/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 61.1922/128 High: 90 Low: 2
S-Box Differences: 61.1904/128 High: 90 Low: 2
Xor 2 Differences: 61.1904/128 High: 90 Low: 2

Cycle: 10/16
P-Box Differences: 61.1904/128 High: 90 Low: 2
Xor 1 Differences: 61.1904/128 High: 90 Low: 2
S-Box Differences: 63.6527/128 High: 91 Low: 6
Xor 2 Differences: 63.6527/128 High: 91 Low: 6

Cycle: 11/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 63.6527/128 High: 91 Low: 6
S-Box Differences: 63.6575/128 High: 91 Low: 3
Xor 2 Differences: 63.6575/128 High: 91 Low: 3

Cycle: 12/16
P-Box Differences: 63.6575/128 High: 91 Low: 3
Xor 1 Differences: 63.6575/128 High: 91 Low: 3
S-Box Differences: 63.9733/128 High: 89 Low: 7
Xor 2 Differences: 63.9733/128 High: 89 Low: 7

Cycle: 13/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 63.9733/128 High: 89 Low: 7
S-Box Differences: 63.9654/128 High: 93 Low: 6
Xor 2 Differences: 63.9654/128 High: 93 Low: 6

Cycle: 14/16
P-Box Differences: 63.9654/128 High: 93 Low: 6
Xor 1 Differences: 63.9654/128 High: 93 Low: 6
S-Box Differences: 64.0025/128 High: 89 Low: 14
Xor 2 Differences: 64.0025/128 High: 89 Low: 14

Cycle: 15/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 64.0025/128 High: 89 Low: 14
S-Box Differences: 63.993/128 High: 90 Low: 19
Xor 2 Differences: 63.993/128 High: 90 Low: 19
(Continued on next page)

Cycle: 16/16
P-Box Differences: 63.993/128 High: 90 Low: 19
Xor 1 Differences: 63.993/128 High: 90 Low: 19
S-Box Differences: 64.0062/128 High: 89 Low: 39
Xor 2 Differences: 64.0062/128 High: 89 Low: 39

As far as the tested blocks, this shows that even in the worst case scenario where only 1 bit of change keeps traveling throughout the ciphertext, it eventually produces a minimum of 39 bits of differences. As for the high, if the S-Box is ideal, 1 bit of change can produce a maximum of 8 bits of change, all in the same block. Then, the next cycle uses the P-box, distributing those changed bits into a maximum of 4 bytes. After the next S-Box, it is possible that it can produce a change of 32 bits, but in the case of this test it produced a maximum change of 28 bits at this point. By the end, the high was 89 bits of difference.

## Individual Key Avalanche (ANGELITA4B)

This next test was directly based off of the first, but instead of testing modified plaintexts, it was the key that was modified. 2,000 keys were generated, along with 2,000 plaintext blocks, all pseudo-random. Each block was encrypted with a unique key, and all the blocks after each operation were recorded. Then, for each key and block pair, 128 copies of the corresponding key were generated. Each key copy had a bit flipped in a unique position, and then it was used to encrypt the original plaintext block paired with the original key. Finally, the plaintext after each operation was compared with the original plaintext, logging the differences in bits. An average of differences was calculated for each at the end, as well as showing the high and low cases. Here are the results:

Cycle: 1/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 63.9959/128 High: 90 Low: 39
S-Box Differences: 64.0052/128 High: 88 Low: 38
Xor 2 Differences: 63.9846/128 High: 91 Low: 39

Cycle: 2/16
P-Box Differences: 64.0084/128 High: 91 Low: 40
Xor 1 Differences: 64.0153/128 High: 89 Low: 38
S-Box Differences: 63.9866/128 High: 90 Low: 38
Xor 2 Differences: 64.0113/128 High: 89 Low: 40

(Continued on next page)

Cycle: 3/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 64.007/128 High: 91 Low: 38
S-Box Differences: 64.0005/128 High: 91 Low: 40
Xor 2 Differences: 64.0157/128 High: 89 Low: 39

Cycle: 4/16
P-Box Differences: 64.0133/128 High: 88 Low: 38
Xor 1 Differences: 63.9868/128 High: 89 Low: 40
S-Box Differences: 64.0056/128 High: 91 Low: 39
Xor 2 Differences: 63.9932/128 High: 94 Low: 38

Cycle: 5/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 63.9961/128 High: 90 Low: 37
S-Box Differences: 64.0124/128 High: 88 Low: 38
Xor 2 Differences: 63.9903/128 High: 89 Low: 40

Cycle: 6/16
P-Box Differences: 64.0092/128 High: 90 Low: 39
Xor 1 Differences: 63.9892/128 High: 90 Low: 39
S-Box Differences: 64.0003/128 High: 87 Low: 41
Xor 2 Differences: 63.9879/128 High: 91 Low: 37

Cycle: 7/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 64.0017/128 High: 89 Low: 38
S-Box Differences: 64.0057/128 High: 89 Low: 35
Xor 2 Differences: 64.009/128 High: 89 Low: 34

Cycle: 8/16
P-Box Differences: 63.9904/128 High: 90 Low: 39
Xor 1 Differences: 64.0102/128 High: 93 Low: 39
S-Box Differences: 63.9947/128 High: 91 Low: 39
Xor 2 Differences: 63.9677/128 High: 88 Low: 40

Cycle: 9/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 64.0061/128 High: 89 Low: 38
S-Box Differences: 64.0034/128 High: 91 Low: 39
Xor 2 Differences: 63.9888/128 High: 92 Low: 40

Cycle: 10/16
P-Box Differences: 64.0121/128 High: 88 Low: 39
Xor 1 Differences: 63.9935/128 High: 87 Low: 40
S-Box Differences: 64.0113/128 High: 90 Low: 38
Xor 2 Differences: 63.989/128 High: 90 Low: 41
(Continued on next page)

Cycle: 11/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 63.9848/128 High: 90 Low: 33
S-Box Differences: 63.9959/128 High: 90 Low: 38
Xor 2 Differences: 64.0097/128 High: 91 Low: 38

Cycle: 12/16
P-Box Differences: 64.0063/128 High: 90 Low: 38
Xor 1 Differences: 64.0138/128 High: 89 Low: 38
S-Box Differences: 63.9804/128 High: 89 Low: 38
Xor 2 Differences: 63.9893/128 High: 88 Low: 39

Cycle: 13/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 63.9893/128 High: 89 Low: 40
S-Box Differences: 64.012/128 High: 89 Low: 39
Xor 2 Differences: 64.0026/128 High: 89 Low: 37

Cycle: 14/16
P-Box Differences: 64.0009/128 High: 88 Low: 40
Xor 1 Differences: 64.0181/128 High: 87 Low: 41
S-Box Differences: 63.9858/128 High: 94 Low: 38
Xor 2 Differences: 63.9923/128 High: 90 Low: 41

Cycle: 15/16
P-Box Differences: Skipped cycle
Xor 1 Differences: 63.9982/128 High: 88 Low: 36
S-Box Differences: 64.0148/128 High: 88 Low: 40
Xor 2 Differences: 64.0168/128 High: 90 Low: 38

Cycle: 16/16
P-Box Differences: 63.9873/128 High: 93 Low: 40
Xor 1 Differences: 64.0104/128 High: 88 Low: 39
S-Box Differences: 63.9946/128 High: 88 Low: 37
Xor 2 Differences: 63.9888/128 High: 89 Low: 35

This test shows that changing one bit in the key produces a fairly consistent difference of about half, though the highs and lows show that it can go in either direction.

## Further Details of the Algorithm

The following table shows the amounts of each part of the key schedule in comparison to the key and total key schedule.

| Key Schedule | KS Bytes | % Of Key | Key Bytes | Key Bits |
|---|---|---|---|---|
| S-Box | 1216 | 59.375% | 9.5 | 76 |
| P-Box | 320 | 15.625% | 2.5 | 20 |
| XOR 1 | 256 | 12.5% | 2 | 16 |
| XOR 2 | 256 | 12.5% | 2 | 16 |
| Total | 2048 | 100% | 16 | 128 |

What this says is that the number of bits from the original key that ultimately becomes the bytes of key schedule for the S-Box generation is 76. This means there are $2^{76}$ possible S-Boxes that can be generated in this way, out of the total possible 256! 8x8 S-Boxes. Likewise, the number of bits of the key being used for the P-Box generation are 20. So, there are $2^{20}$ (Or 1,048,576) possible P-Boxes that can be generated from that. Due to the P-Box being used to exchange 2-bit values in the plaintext by their index, the output values are in the range 0-63 (Inclusive), so there are 64! possible P-Boxes of this size. Given the massive range of possible boxes, the ones that can be generated in this algorithm are a very small portion of those possibilities.

Given that, one purposeful part of the design was to separate the parts of the key schedule so no part is ever used more than once. The 76 bits of the key are used to generate the S-Box, and then never used again, same with the 20 bits for the P-Box. Then, there are 2 sets of 256 bytes for the XORs in the encryption/decryption routine. There are 16 bytes in the plaintext, and 16 cycles. This ensures that each byte is XORd with a unique key schedule byte twice per cycle, with no recycled bytes. I still cannot vouch for the security of this algorithm due to not being able to analyze it any further, but I think it is important to note that if some key schedule bytes were recovered in any attack, they shouldn't be able to be used to uncover information on the bits in any other section of the key schedule. On top of this, since the S-Box and P-Box are created from a shuffling of their initial values that is dependent on the key schedule bits, the two boxes are not directly related to the key bits. There may be some deeper math that would say otherwise about this, but for now it seems like a nice feature.

## The Not So Conclusive Conclusion

As far as these tests go, the averages are all very consistent in being about half the bits. ANGELITA4B has better plaintext avalanche than ANGELITA3, which is the exact purpose of the modified P-Box. It is important to note that while the average seems consistent, the high and low show it can be biased in either direction at times. It would be more ideal if they were closer to the middle average, but it is not bad for a pure test algorithm.

The reason why I say the conclusion is not conclusive is that, as mentioned at the beginning, I still have a lot to learn especially when it comes to cryptanalysis. For now, this project is something to put aside until my skills develop enough to properly attack the algorithm. Regardless of this, I really enjoyed the process, it only drives me further to get into an actual career in this field. I just hope that when someone sees this project, they don't view it as another amateur attempt to create an “unbreakable” cipher, as this is not the case. I went into this project assuming I was going to make an insecure algorithm, and coming out of it at the end, I still feel it must be insecure simply because I made it based on a lot of assumptions. If I don't eventually uncover the weaknesses in my own algorithm, someone else will. With that, that is all for now, but this won't be the last you hear of me in this field.

stringzzz
Ghostwarez Co. :3
August 8th, 2022