Machine Learning for Cryptography

Report 2: Hashing, BIC, SAC, and Collisions

Name: Mateusz Tabaszewski

November 12, 2025

1 Introduction

The presented project aims to assess the strength of SHA256, and MD5 hashing algorithms against a weak, toy hashing algorithms, and to visualize the results. The project utilized the following criteria:

- the Strict Avalanche Criterion (SAC) – test that if you change just one single bit in the input message, the output hash should change completely and randomly
- Bit Independence Criterion (BIC) – ensures that the change in one output bit is independent of the change in any other output bit by calculating the correlation measure

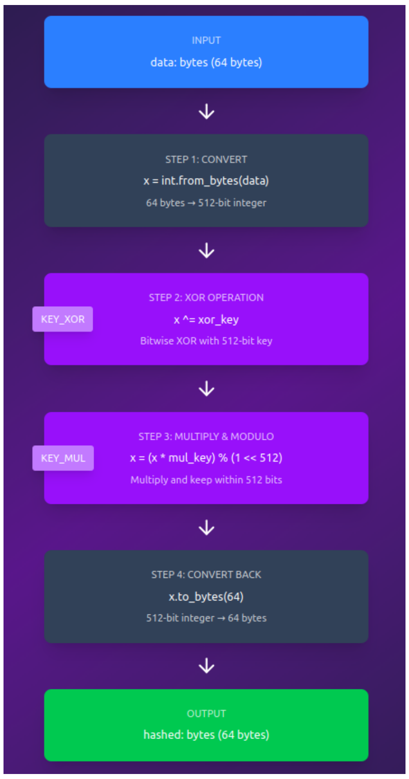
Furthermore, the final task checked the algorithms' defense against collisions, occurring when two input messages produce the exact same output hash. A strong cryptographic hash function is required to exhibit high statistical randomness, and a resilience against collision detection mechanisms. Additionally, a toy (weak) hash is expected to show visible patterns, dependencies, and non-random behaviors, demonstrating poor diffusion and higher predictability that could be exploited by attackers, further showcasing the importance of strong hashing algorithms.

2 Experiments

All experiments were conducted using a fixed message length to avoid padding artifacts, with $N = 10,000$ trials used for the statistical criteria measurements. Additionally, the collision detection experiments were repeated 10 times to produce reliable results. Moreover, the length of each message was 512 bits (64 bytes) for SHA256, custom (toy) hashing algorithm, and random algorithm, and the MD5 algorithm, which corresponds to the block size. The random hashing algorithm would always return a random bit string and served as a sanity check, validation test to compare other hashing algorithms with. Furthermore, when detecting the collisions, only 32 or 16 first bits were used as a test of "near-collisions" due to computational complexity of detecting full collisions.

2.1 Toy Hash

The toy hash is a simple, custom algorithm that treats the 64 bytes of input as a 512-bit number. Afterward, it applies an XOR operation against a key, then a multiplication against another 32-byte key, applies the modulo operation and outputs the 512-bit result as 64 bytes. The diagram showcasing the inner-workings of the algorithm is present in Figure 1.


INPUT
data: bytes (64 bytes)
STEP 1: CONVERT
64 bytes → 512-bit integer
STEP 2: XOR OPERATION
KEY_XOR
Bitwise XOR with 512-bit key
STEP 3: MULTIPLY & MODULO
KEY_MUL
Multiply and keep within 512 bits
STEP 4: CONVERT BACK
512-bit integer → 64 bytes
OUTPUT
hashed: bytes (64 bytes)

```
x = int.from_bytes(data)
x ^= xor_key
x = (x * mul_key) % (1 << 512)
x.to_bytes(64)
```

Figure 1: Custom hashing algorithm visualization.

The algorithm was used as a benchmark to compare the performance of a weak vs a strong hashing algorithm and their properties.

2.2 SAC

The SAC measures the probability that each output bit j flips when a single input bit i is flipped. The experiment was conducted with the following **procedure:**

1. Initialize N trials for at least 32 selected input bits i.
2. For each input bit i and each trial:
   (a) Take message x
   (b) Compute the original hash $y = H(x)$ and the flipped hash value
   $$y' = H(x \oplus e_i),$$
   where e_i is a vector with only the i-th bit set.

(c) Record the output difference vector

$$\Delta = y \oplus y'.$$

3. Aggregate results over N trials to estimate the flip probability

$$p_{i \to j} = \Pr[\Delta_j = 1]$$

The final result of the experiments measured the conditional expectation of flipping the output bit, along with mean values for all bits and standard deviations of the bits.

2.3 BIC

The BIC property is meant to ensure no statistical dependence between pairs of output bits after a single input bit flip. This was calculated by utilizing the same experimental set-up as before, and calculating the Pearson's correlation matrix. The experiment was conducted by utilizing the following **procedure:**

1. Use the same N trials and collected Δ vectors from the SAC experiment, focusing on a few key input bits i.
2. For each pair of output bits (j, k):
   (a) Compute the correlation (e.g., Pearson's correlation coefficient ϕ) between their respective flip indicators (Δ_j, Δ_k).
3. Summarize these correlations into a BIC matrix.

The results showcase the correlation values in the form of a heatmap.

2.4 Collisions

The collisions were investigated in one of two modes: "one" or "any". The former specified that a collision would only occur when the same hash as an initial generated hash was discovered. Whereas, the latter dictated that any collision being the same as any of the previously discovered ones was acceptable. Overall, the following algorithms were tested in the collision detection task:

- **Brute force** – incrementally changing the input message by a bit until collision is found
- **Random** – returning random messages until collision is found
- **Steepest** – local search algorithm using the hamming distance criterion, taking greatest improvement (restarting if stuck until success or iteration limit)
- **Greedy** – local search algorithm using the hamming distance criterion, taking any improvement (restarting if stuck until success or iteration limit)
- **Genetic** – genetic algorithm for discovering the collisions

Only the brute force, and random algorithms were used in “one” and “any” modes, while all the remaining ones were tested only in “one” mode for the sake of efficiency. Furthermore, only “any” mode was tested for 32-bit collisions, while “one” was tested for “16-bit” collisions. All algorithms had a set limit of time after which the task would be deemed a failure, however, the limit was never reached in practice.

3 Results & Observations

The section explores the results and observations for each criterion. Each algorithm was run multiple times for each hashing function, as was specified previously. The analysis of results clearly shows that the SHA256 and MD5 algorithms showcase properties of SAC and BIC that are not represented by the toy algorithm.

3.1 SAC

The Figures 2, and 3 showcase the normalized, and unnormalized (to the common color pallet) results of the analysis of the SAC criterion. The plots showcase the distribution of conditional probabilities for the SHA256, MD5, random, and toy hashing algorithms. Along with vectors showing the mean and standard deviation values before and after hashing the input.

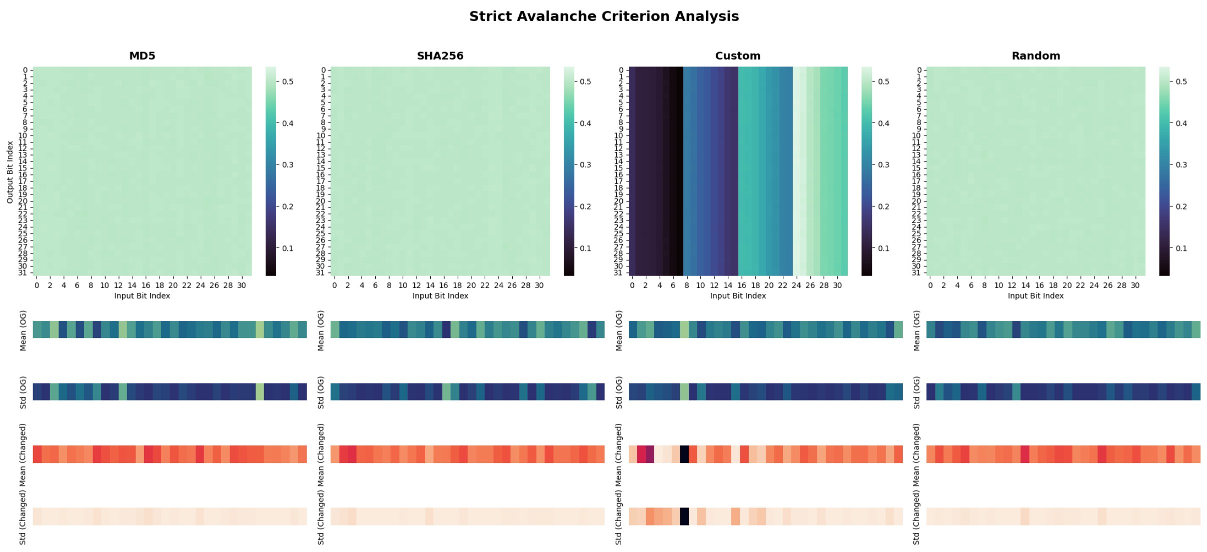


Figure 2: SAC visualization for all hashing algorithms, with normalized values.

It can be seen, that the toy algorithm produces characteristic “gradient-like” plot, while the remaining hashing algorithms remain undistinguishable from random output, showing that they exhibit the SAC property, while the toy algorithms clearly breaks this convention, showing clear, potentially exploitable patterns.

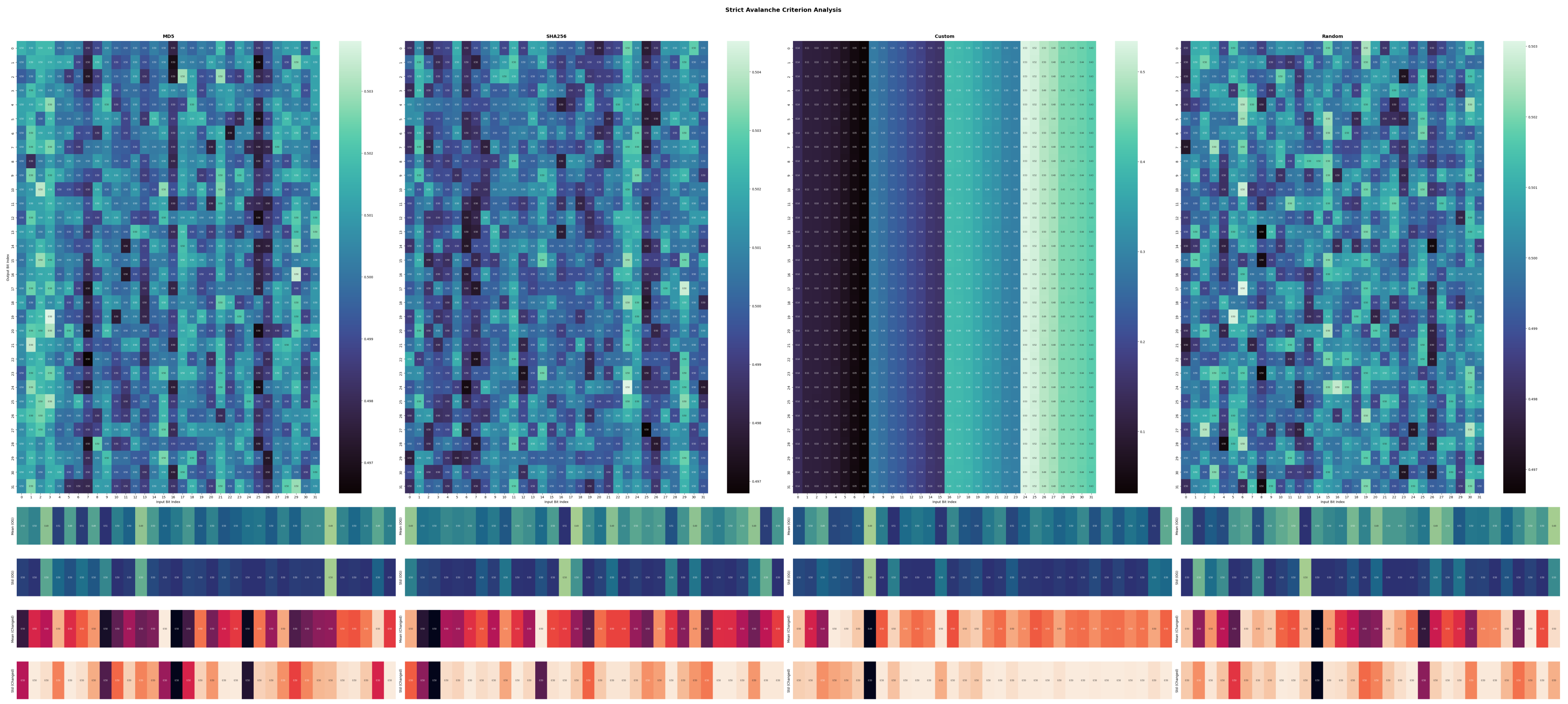


Figure 3: Detailed SAC visualization for all hashing algorithms, with normalized values.

3.2 BIC

The Figures 4, and 5 showcase the normalized, and unnormalized (to the common color pallet) results of the analysis of the BIC criterion. The plots showcase the correlation values for the SHA256, MD5, random, and toy hashing algorithms. The colors showcase the measure of the correlation value.

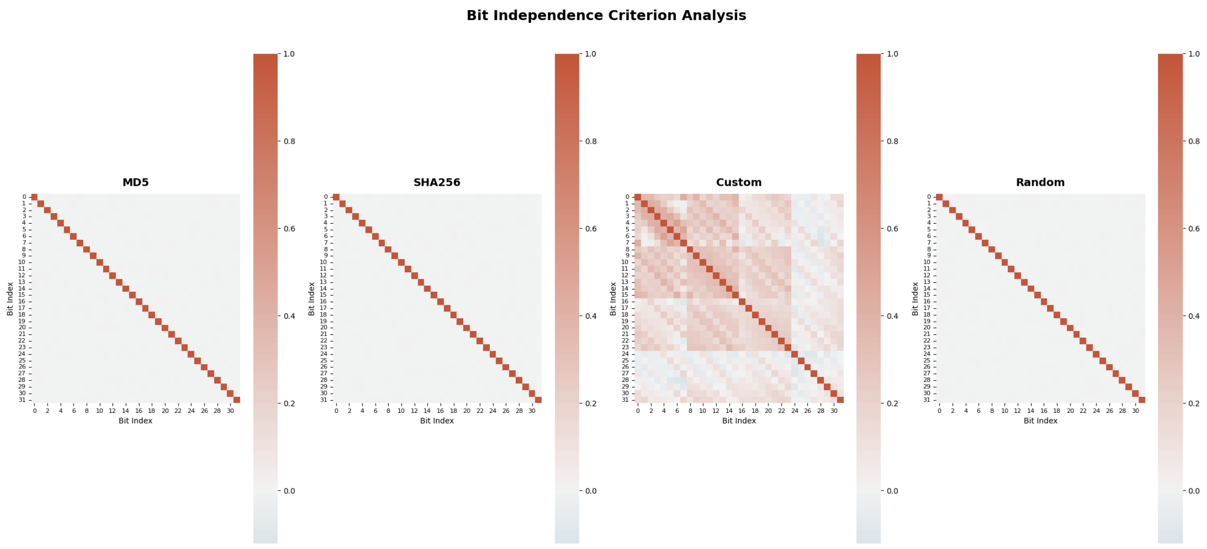


Figure 4: BIC visualization for all hashing algorithms, with normalized values.

It can be seen, that the toy algorithm results in a plot with clear clusters and correlations, something not exhibited by either hashing algorithm, or a random hashing algorithm for that matter. This yet again, proves that the toy example may exhibit easily exploitable patterns, while the standard hashing implementations remain safe from attacks that might target a clear BIC principal violation.

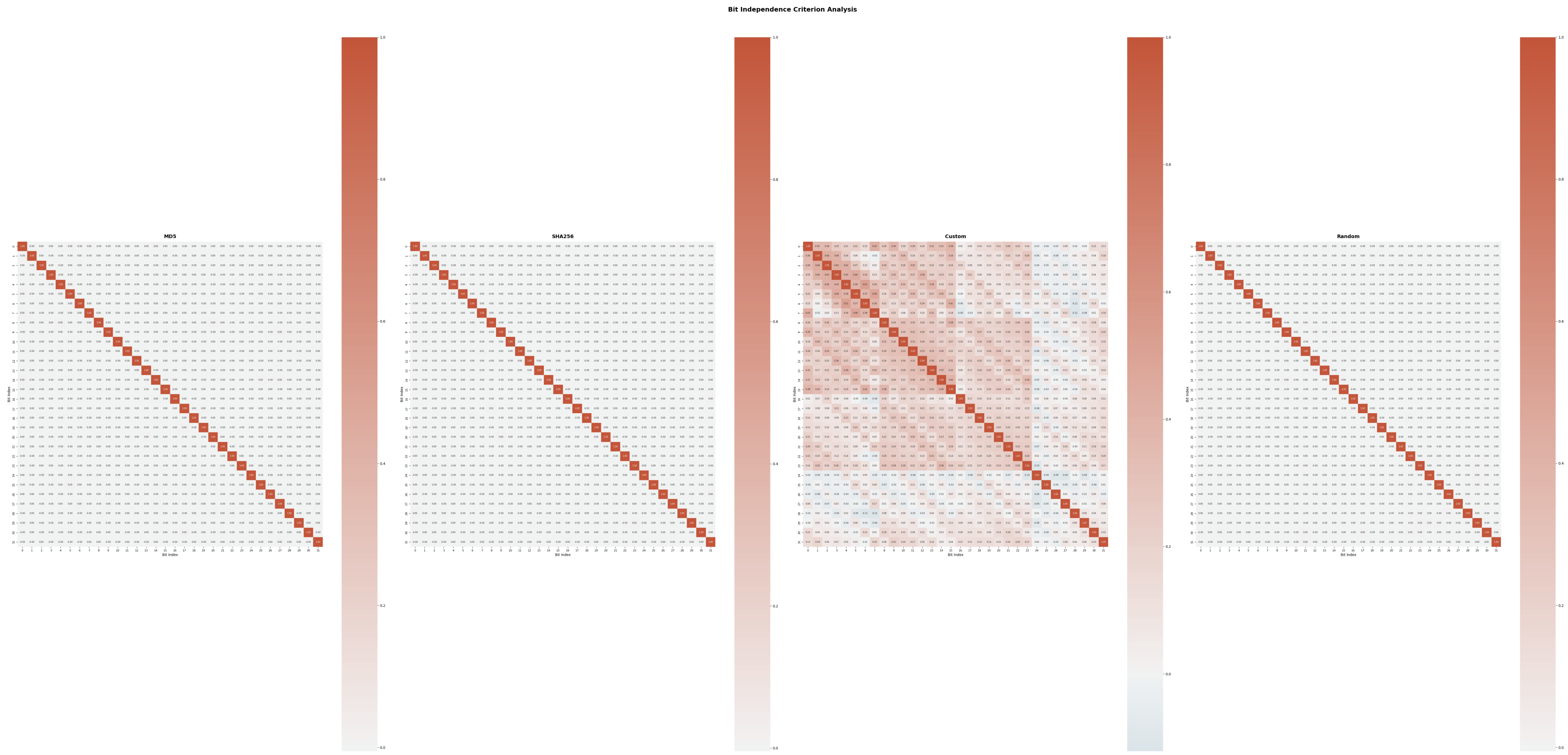


Figure 5: Detailed BIC visualization for all hashing algorithms, with normalized values.

3.3 Collisions

Lastly, the collision tests were conducted in one of two modes, “one”, and “any”, as was detailed previously. As such, Figure 6 shows the comparison of collisions for the brute force and random algorithms in the logarithmic scale. As can be seen, random search outperforms the brute force algorithm. This is likely thanks to a more even sampling of the search space which might allow for an easier way to discover a collision than iteratively adding one bit which might make it difficult to discover the original message if the hashing algorithm is good enough to produce many significantly different for neighboring messages. Another reason might be related to the implementation details, as the random search draws from the same distribution (same function) as the original message generation, meaning that if the distribution of possible values in the *os.urandom* function in Python is not uniform, it might skew the results to producing the same message more often than expected.

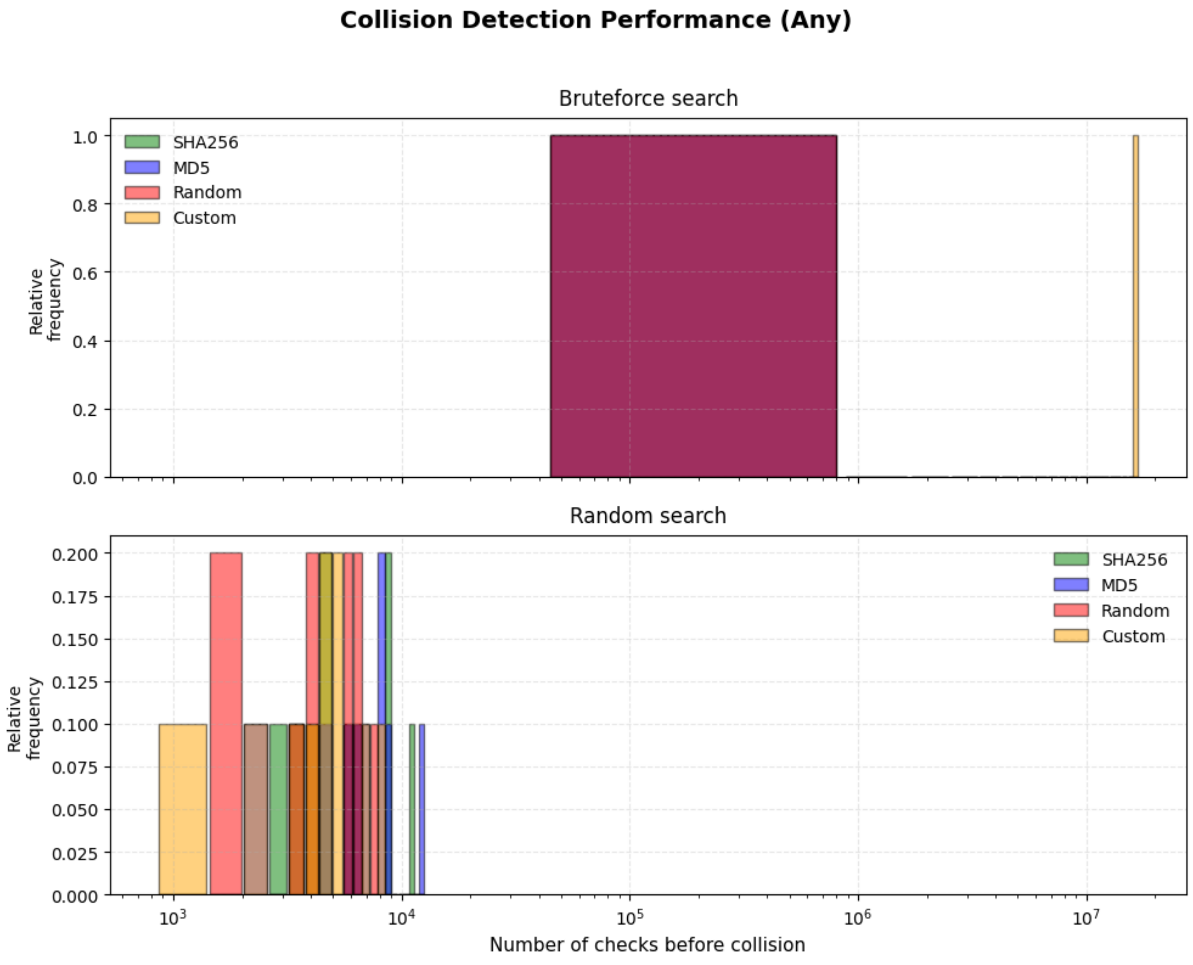


Figure 6: Collision visualization in the “any” mode for all algorithms.

Figure 7 shows the comparison for the "one" mode for all algorithms. It should be noted that when judging the similarity of the hashes, the hamming distance is used to guide the search of the methods. The plot showcases the distribution of the number of checks for each algorithm and every hashing function on a logarithmic scale. It can be seen that the random and brute force approaches still produce surprisingly good results, while local search algorithms perform poorly, showing the error of trying to iteratively find the collisions by incremental changes, as the resulting hashes might be very different leading to a very rugged solution landscape and low fitness-distance correlation. Surprisingly, genetic algorithm did not outperform the random baseline overall Furthermore, poor performance of the genetic algorithm might be attributed to the outlier case which did not even managed to find any collisions. All the remaining algorithms managed to discover a collision within the allotted time limit, as is shown in Figure 8.

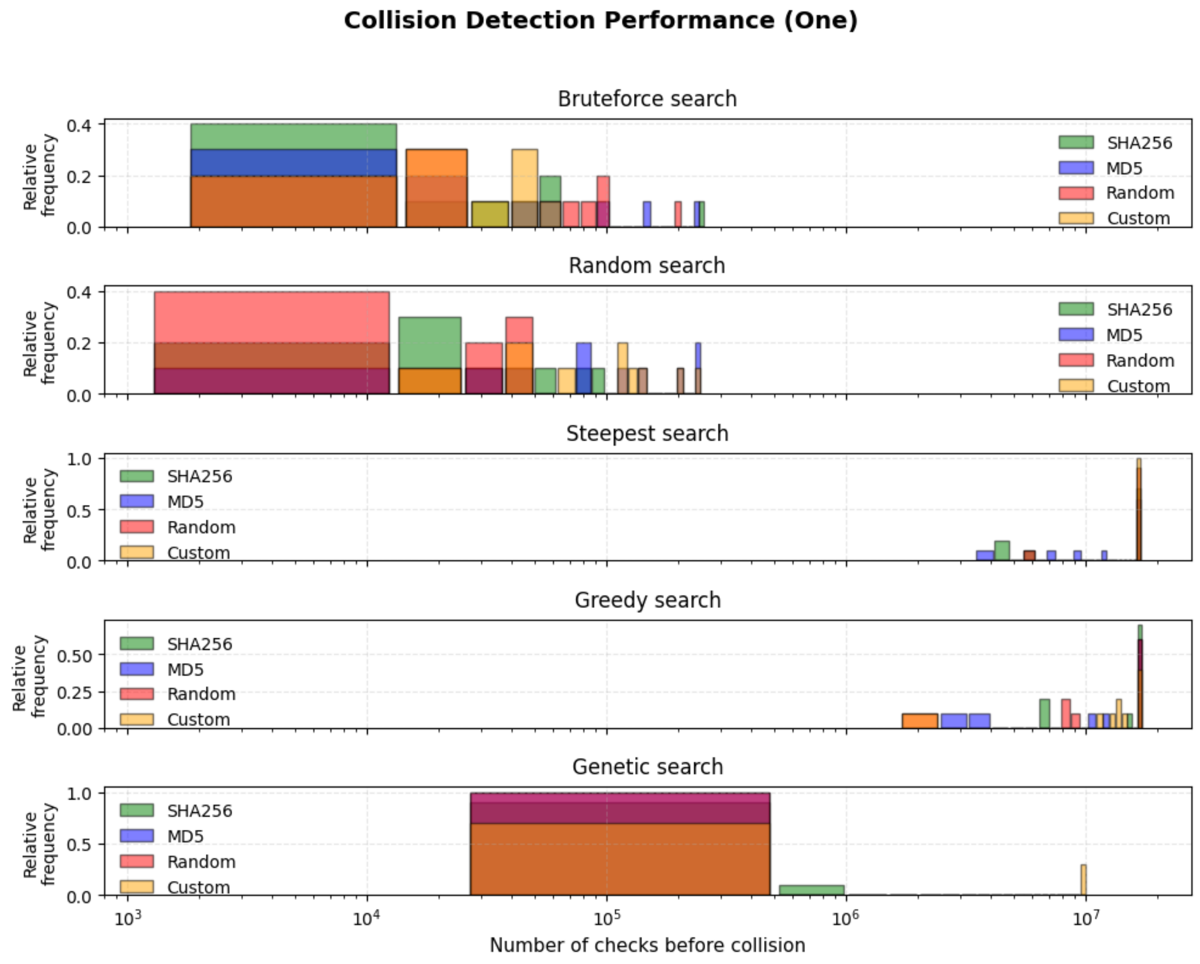


Figure 7: Collision visualization in the “one” mode for all algorithms.

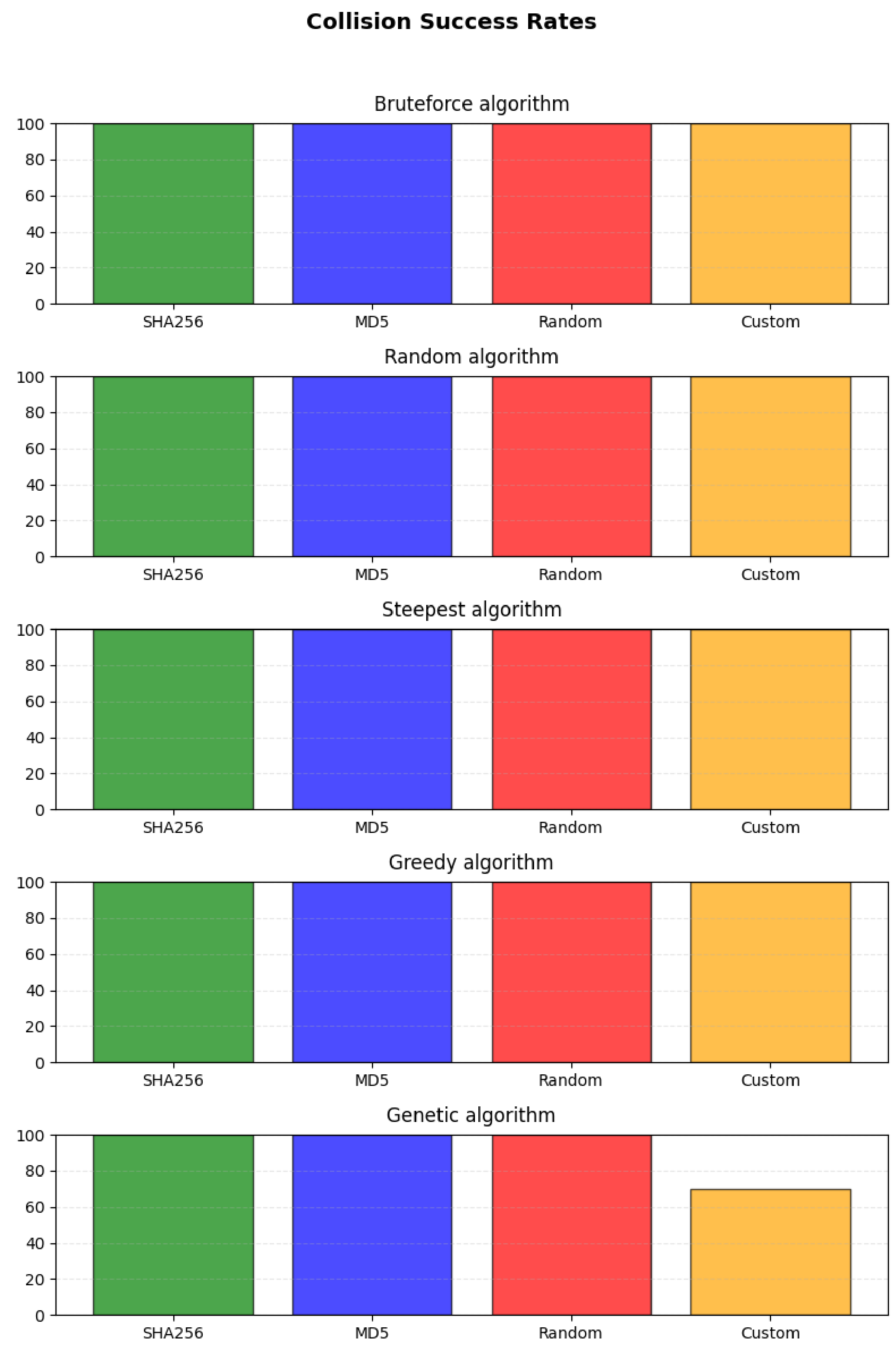


Figure 8: Success-rate visualization in the "one" mode for all algorithms.

The convergence of the evolutionary algorithm was presented in Figure 9. Please note, the lines are of different length for each run, showing when the collision was discovered. The negative value for distance is due to the way the objective was formulated as a maximization task for the DAEP genetic algorithm framework.

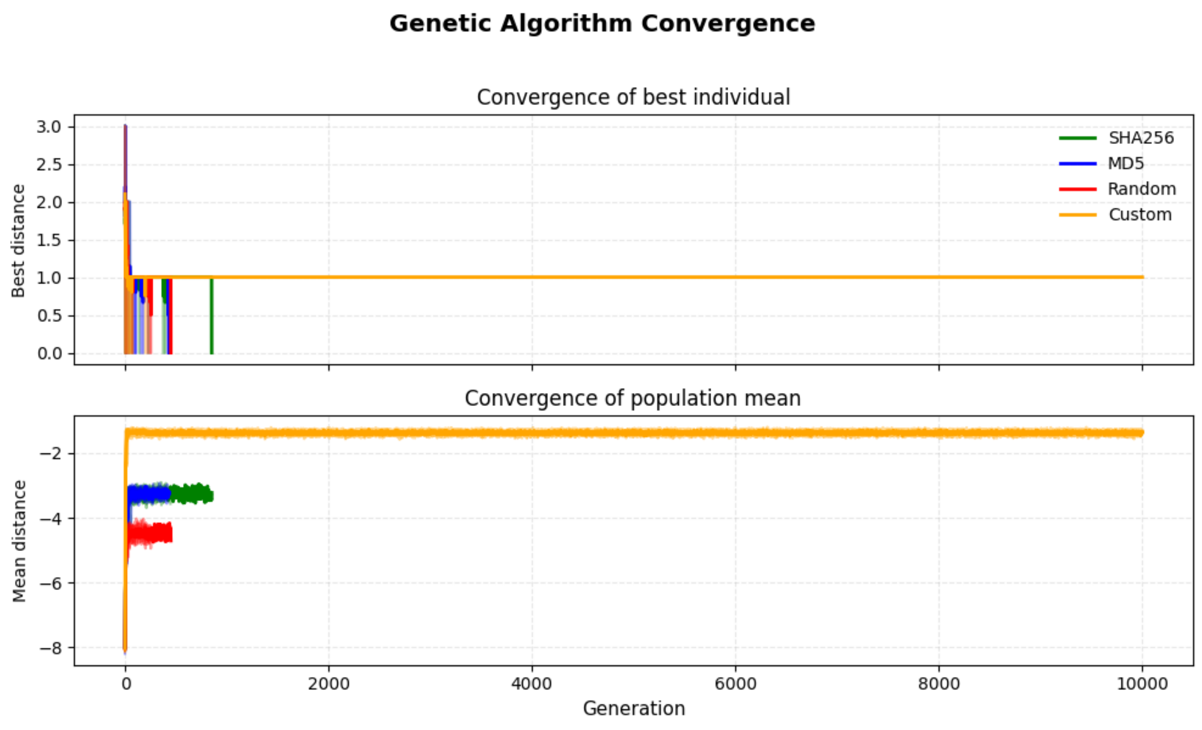


Figure 9: Evolutionary algorithm convergence visualization.

4 Conclusions

The project successfully demonstrated the fundamental differences between cryptographically secure hash functions (SHA256 and MD5) and a weak, toy hashing algorithm through rigorous testing of the Strict Avalanche Criterion (SAC), Bit Independence Criterion (BIC), and collision resistance properties.

The experimental results seem to concur that the common hashing algorithms exhibit the useful and secure properties of SAC and BIC, similarly to a random distribution, making them hard to break by crypto-analysis techniques. on the other hand, these abilities are not possessed by a simple toy algorithm. Furthermore, the study showcased the difficulty of finding collisions even for simple, naive hashing methods. Further emphasizing the importance of human intuition in crypto-analysis techniques.

Lastly, it should be noted that several limitations remain, as collisions were only tested for a small part of the whole possible output of the hashing functions. Furthermore, the relatively small number of trials (N = 10,000 for SAC/BIC and 10 repetitions for collision detection) may not capture rare statistical anomalies or edge cases in the hashing algorithms' behavior. As such, focusing on a more thorough analysis may be of great importance in further research.

In conclusion, the work further emphasizes the importance of established, secure hashing algorithms, while showing risks associated with using simple, self-made solutions which (if not tested rigorously) may exhibit obvious security weaknesses, and as such, using them may carry substantial risk.