Cyclone: A Hash Algorithm with Tweak-Based Diffusion and Non-Linear State Dependencies

Samuel J. Poyner, Independent Researcher & Developer

April 17th, 2025

1: Introduction

Cyclone was built purely out of the bored mind of its creator who, after looking at algorithms like SHA-256 and MD5, decided he wanted to learn how they worked. Cyclone was originally built without any goal other than to experiment with cryptographic and mathematical principles, but has evolved over my few weeks' worth of time working on it, which involved a loop of recognizing problems, thinking about what I could do to fix them instead of doing my schoolwork, and then making adjustments accordingly in my free time. This paper outlines and discusses the inner workings of Cyclone as well as test scores for collision detection, byte diffusion, and avalanche effect.

2: The Algorithm

2.1: Algorithm Design Overview

Cyclone's preprocessing steps are meant to be simple enough, yet able to prep the data for the mixer function. The steps are as follows:

1. If a tweak value has not been entered, generate a random one and ensure that the tweak value is exactly sixteen bytes long.
2. Ensure that the data has been padded to a multiple of 16 bytes
3. Divide the data into chunks of 16 bytes
4. Rotate each chunk leftward in accordance with its index. For example, the chunk at index 1 will be rotated left one time, the chunk at index 2 will be rotated left twice, and this continues through the chunk

Cyclone uses a tweak-based, state-interdependent mixing approach in contrast to common salting and transformation techniques. The core operations of the mixer are as follows:

1. Addition modulo 256
2. XOR with a tweak value
3. XOR with a rotation of a neighboring byte

The main components of the algorithm are designed to ensure diffusion between the bytes by creating dependencies between the bytes, which allows for the current state to be dependent fully on itself, which causes the diffusion to improve due to the small changes being dispersed through the whole state, which will affect the rest of the hash. I have also chosen to use a tweak value over a salt value to allow for the tweak to affect the hash more than a salt value would.

The Cyclone algorithm is also completely modular, meaning that instead of padding the message to a multiple of 16 bytes, making 16-byte chunks, and using a 16-byte tweak, you can instead pad the message to a multiple of 32 bytes, make 32-byte chunks, and use a 32-byte tweak

to generate a 32-byte hash. These numbers must all be the same, because the message is padded to a multiple of 32 bytes to make it divisible into 32-byte chunks. A 32-byte tweak would also be required so that the tweak works completely with the 32 byte chunk. The only limitation to size is the computational power of the device that generates the hash.

2.2: Tweak Values

The tweak works similarly to a salt in the sense that it's selected at the beginning of the hashing process and used to modify the output of the hash function. However, in contrast to a salt, the tweak is not appended to the end of the message input, and is instead used in the mixer function. The tweak value is XORed with the current state value after the addition step in the mixer, allowing the tweak to have more influence over the final state. The tweak step is placed after the addition step and before neighbor rotation.

2.3: Neighbors

The neighbor effect is part of what makes Cyclone what it is. The use of neighbors inside the current state allows for hashes to have state-interdependence, which increases the avalanche effect of the entire function by allowing the neighbors of a byte to pick up on any changes, which allows the change to diffuse through the state. The neighbor is the value to the right of the current byte, and a calculation is performed to obtain a rotation of said neighbor, which is XORed with the current state value, which introduces non-linear state-interdependence due to the non-linear nature of XOR.

2.4: Steps

Here are the algorithm steps to Cyclone:

1. If a tweak value is not provided, generate one. Ensure the tweak value is exactly 16 bytes long.
2. Pad the message until the message is a multiple of 16 bytes long.
3. Divide the message into 16-byte chunks.
4. Rotate each chunk left by an amount equal to its placement in the chunk list. Chunk 1 will be rotated left one time, chunk 2 twice, and so on.
5. Initialize a base state to a set of 16 0x00 bytes.
6. Iterate steps 7-12 over every chunk.
7. For every byte in the chunk at index i, create a variable “s” that will be defined like so: s = (chunk[i] + state[i]) mod 256, where chunk[i] is our current chunk byte and state[i] is our current state byte.
8. XOR “s” and the tweak byte at the same index i as the current byte.
9. Select the byte at the current index plus one modulo 16, or (index+1) mod 16, as the neighbor.
10. Create a variable to store a computed rotation of the neighbor byte by the index number modulo 8 (index mod 8).
11. XOR the rotated neighbor variable with “s”.
12. Set the state values to the corresponding “s” values in our modified chunk.
13. Do steps 6-12 40 times

3: Security Properties

3.1: Avalanche

To test the avalanche effect in Cyclone, I generated hashes using sets of slightly different messages and the same tweak value. One of those sets was “Hello, World!”, “Hello, Wprld!”, “Hfllo, World!”, and “Hfllo, Wprld!” These slight changes were made because the encoding for “p” is one more than the coding for “o”, and the same is true for “f” and “e”. These outputs produced hashes that had average Hamming distances of ~49%.

3.2: Byte Diffusion

In order to ensure that all bytes were being affected evenly, I ran a series of 9,999 tests for Cyclone, which ranged from “msg0000” to “msg9999”, and counted the changes like so: if a byte at hash[i] differed from the same byte in hash[i-1], I increased the counter for that byte by one. The exact scores for each byte are given below:
[9962, 9964, 9956, 9957, 9953, 9965, 9958, 9971, 9965, 9963, 9949, 9961, 9964, 9952, 9965, 9960]

The proximity of these numbers to each other indicate that each byte is changing approximately equally, and the high number of each number further reinforces the impact of the avalanche effect in Cyclone.

3.3: Collision Detection (Birthday Paradox)

It would be computationally infeasible to calculate every possible hash to check for any collisions. Instead, I used the birthday paradox to develop a script to test for collisions. Multiple tests of the hash with the script indicated that no collisions were detected in the algorithm.

Final Thoughts

Originally, this project started out as a simple introductory experiment in hashing, where the message was simply padded to a multiple of 16 bytes, split into 16 byte chunks, and XORed together. Through my own perseverance and perfectionism, I managed to turn my tiny little hashing experiment into a now well-functioning hashing algorithm. While it may not be a cryptographic standard as of April 17th, 2025 and currently remains unproven in formal cryptanalysis, its behavior in testing shows promise. For now, Cyclone stands as a wonderful personal achievement for myself.