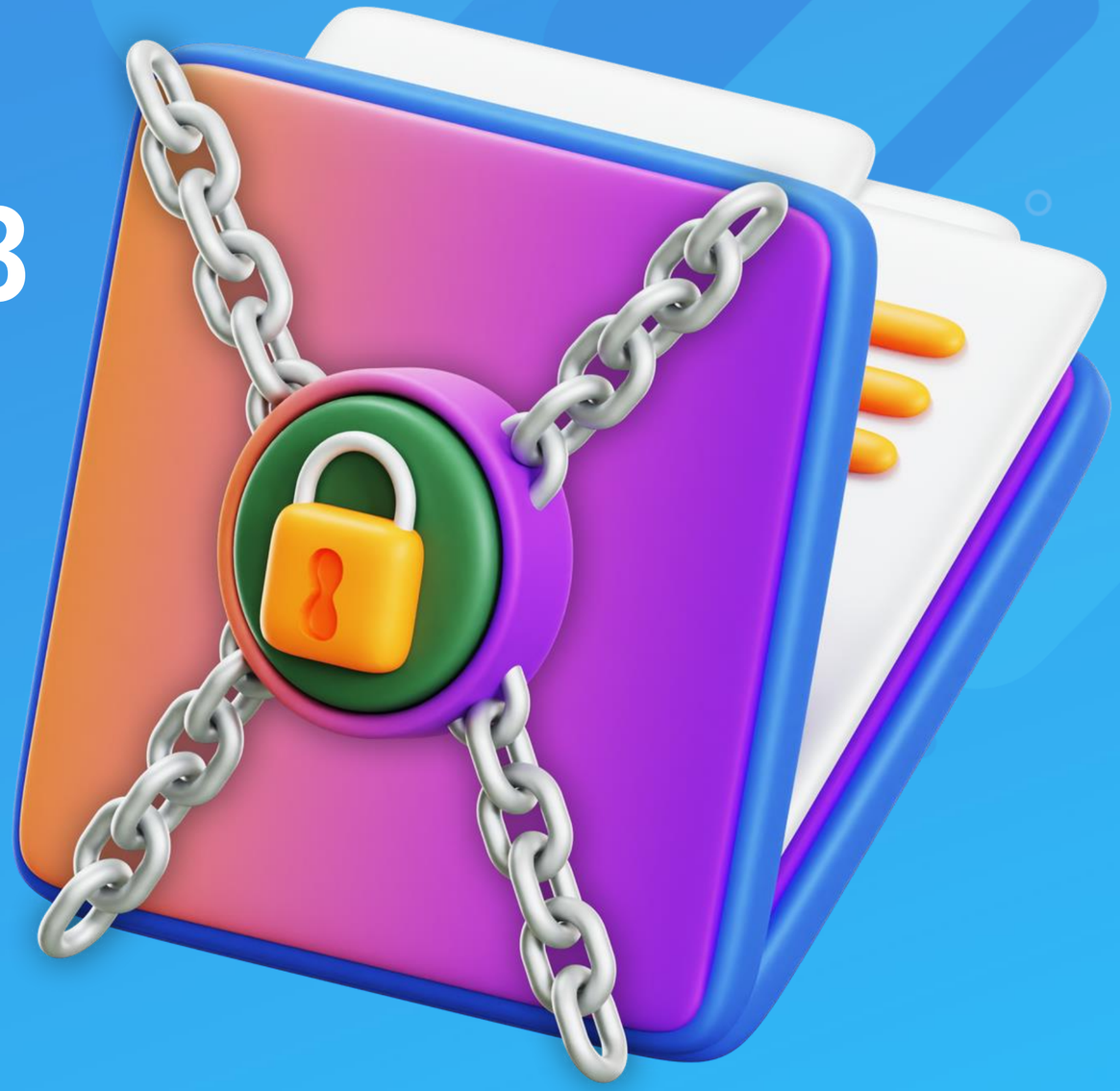
Mã hóa dòng
CRYPTMT VERSION 3



Nhóm 8:

Lê Minh Tuấn - 20203916
Nguyễn Huy Toàn - 20203890
Trần Duy Anh - 20203872
Ngô Thị Thu Thủy - 20203888
Bùi Vũ Bình Giang - 20203897
Đào Thị Hồng Loan - 20203903

4/ 2023 - 01

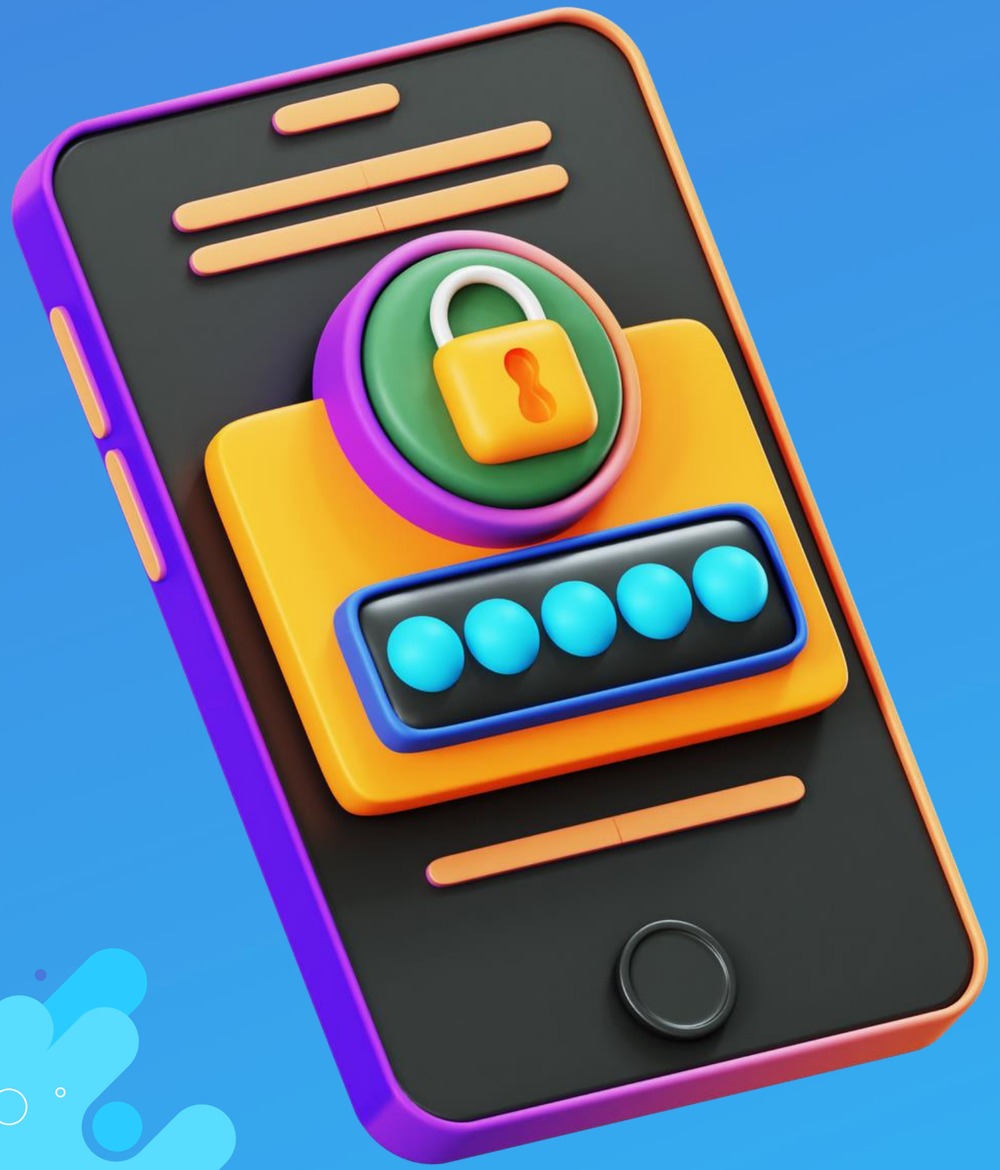
MỤC LỤC:

1.
Giới thiệu

2.
Mã dòng
CRYPTMT
version3

3.
Chống lại
các cuộc
tấn công
thông thường

4.
So sánh
-
Kết luận



I. GIỚI THIỆU

- Cryptographically Secure Mersenne Twister - được tạo bởi Cryptography Research Team thuộc Rambus Inc., một công ty công nghệ có trụ sở tại California, Hoa Kỳ

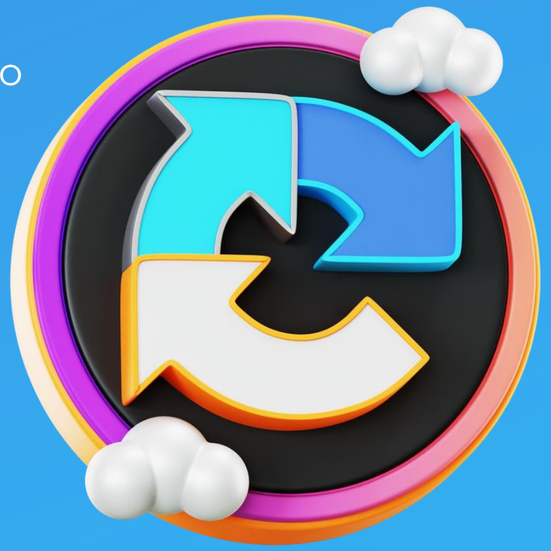
- Nó sử dụng thuật toán sinh số ngẫu nhiên Mersenne Twister để tạo ra các số ngẫu nhiên.
Để đảm bảo tính bảo mật, ver.3 sử dụng một phép trộn ngẫu nhiên khác để biến đổi kết quả của MT

- CRYPTMT phiên bản 3 là một thuật toán sinh số ngẫu nhiên mã hóa được cải tiến để đảm bảo tính bảo mật và tăng cường khả năng sinh số ngẫu nhiên. Nó được sử dụng trong các ứng dụng yêu cầu tính bảo mật cao và đang được xem xét là một trong những giải pháp tốt nhất cho việc tạo số ngẫu nhiên an toàn.

II. Mã dòng CRYPTMT version3

2.1. : Kí hiệu:

W: là bộ nhớ đệm chứa dữ liệu đầu vào
S: là bảng phụ thuộc vào khóa
T: là bảng dùng để trộn dữ liệu
r: là số lượng vòng lặp
N: là kích thước của bộ nhớ đệm
L: là độ dài của khóa
m: là số lượng bộ ba phần tử
t: là số lượng phần tử trong bộ trộn
k: Khóa bí mật



r: Số lượng vòng trong khối vòng
iv: Vectơ khởi tạo, có độ dài 128 bit
r: Số lượng vòng trong khối vòng
w: Độ rộng từng từ
n: Độ dài của keystream được tạo ra
MT: Một khối vòng dựa trên thuật toán Mersenne Twister
B: Một hàm phi tuyến tính trên bit
F: Một hàm phi tuyến tính trên bit

II. Mã dòng CRYPTMT version3

2.1. : Kí hiệu:

Quá trình mã hóa dữ liệu đầu vào m:

1. Sử dụng k và iv để khởi tạo trạng thái của MT.
2. Sử dụng MT để tạo ra keystream với độ dài n.
3. Chia keystream thành các từ có độ dài w.
4. Áp dụng hàm phi tuyến tính B lên từng từ của keystream để tạo ra các khối b1, b2, ..., b{n/w}.
5. Áp dụng hàm phi tuyến tính F lên các khối bi để tạo ra các khối keystream s1, s2, ..., s{n/w}
6. XOR từng byte của dữ liệu đầu vào m với các byte tương ứng của keystream s để tạo ra dữ liệu đã được mã hóa.

Công thức toán học của quá trình mã hóa CRYPTMT Version 3:

$$ci=mi \oplus si$$

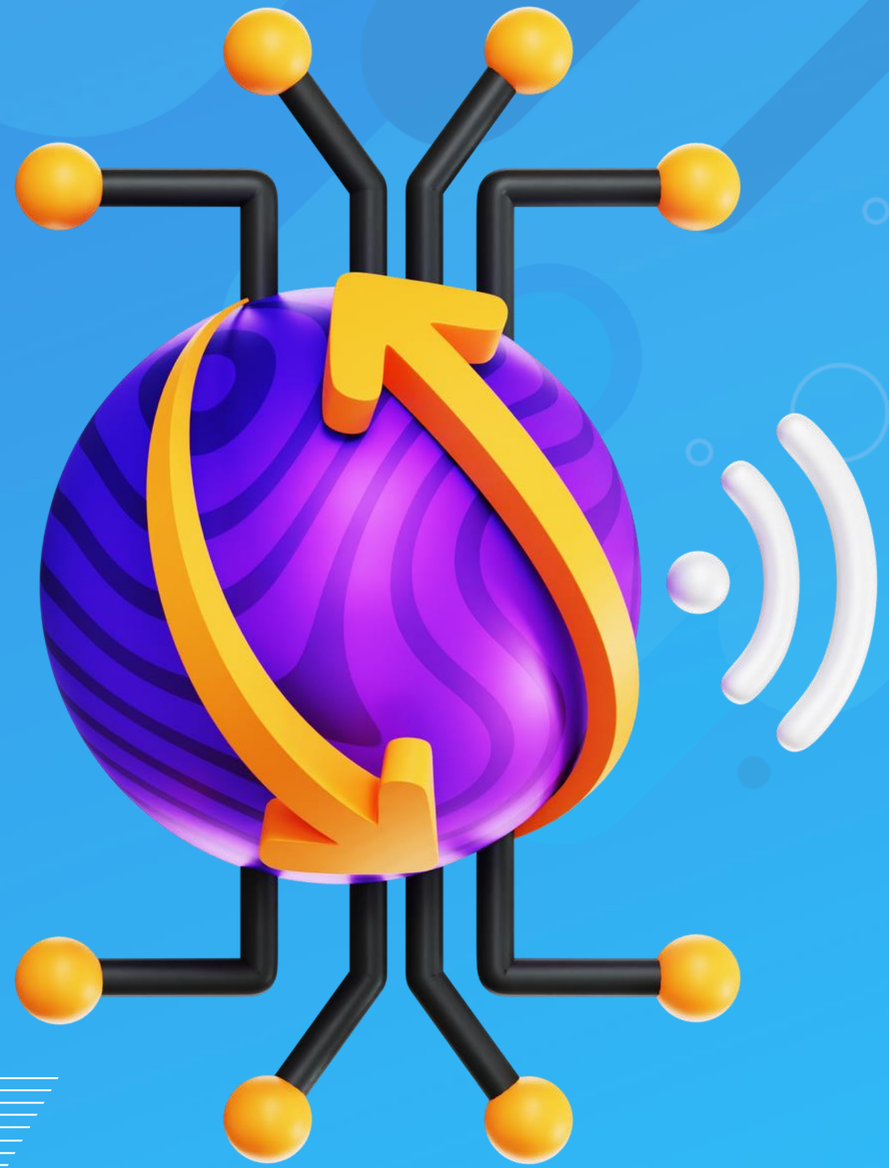
Trong đó:
mi - byte thứ i của dữ liệu cần mã hóa
si - byte thứ i của khóa được tạo
\oplus - phép toán XOR giữa hai byte
Kết quả của phép toán XOR, được lưu trữ trong ci, byte thứ i của dữ liệu đã được mã hóa.

II. Mã dòng CRYPTMT version3

2.2. Simple Fast MT:

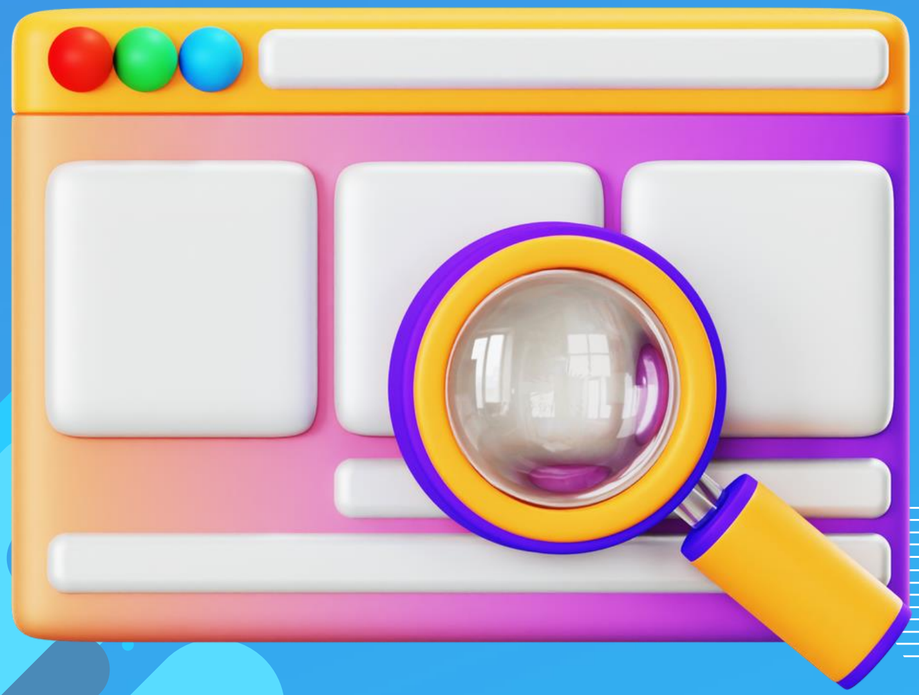
Thuật toán SFMT-19937 được xây dựng dựa trên thuật toán MT19937, với nhiều cải tiến và tối ưu hóa để tăng hiệu suất và độ tin cậy

Thuật toán này có khả năng sinh ra một chuỗi số ngẫu nhiên dài đến hơn 10^6000, với độ tin cậy cao và không bị lặp lại sau một thời gian dài.



II. Mã dòng CRYPTMT version3

2.3. A new filter (Bộ lọc mới):



Ở ver.3, ta dùng phép toán nhị phân $\tilde{\times}$ thay vì × .

Phép $\tilde{\times}$ được định nghĩa : x $\tilde{\times}$ y := $\varphi^{-1}(\varphi(x) \times \varphi(y))$

× là phép nhân trong S

Ở version 3, ta áp dụng bộ lọc được điều chỉnh như sau:

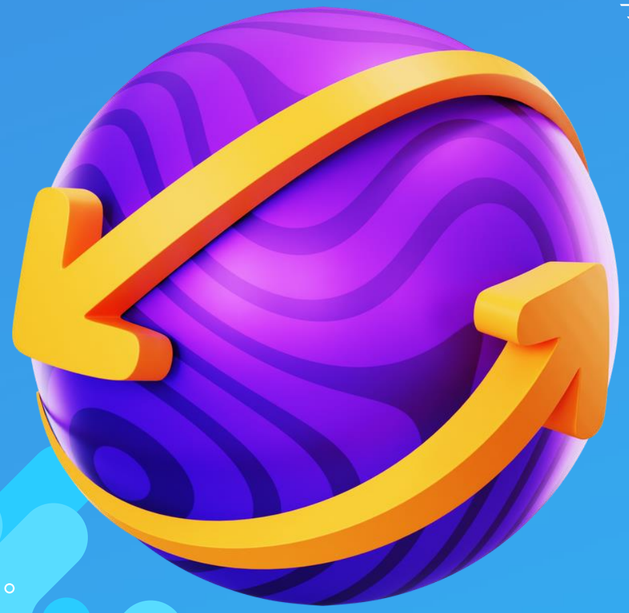
Cho 1 cặp số nguyên 128 bit x, y, ta có :

f(y,x) := ((y ⊕ (y[0][3][2][1] >>32 1)) $\tilde{\times}_{32}$ x

Hàm đầu ra : g(y) := LSB16 32(y ⊕ (y >>32 16)).

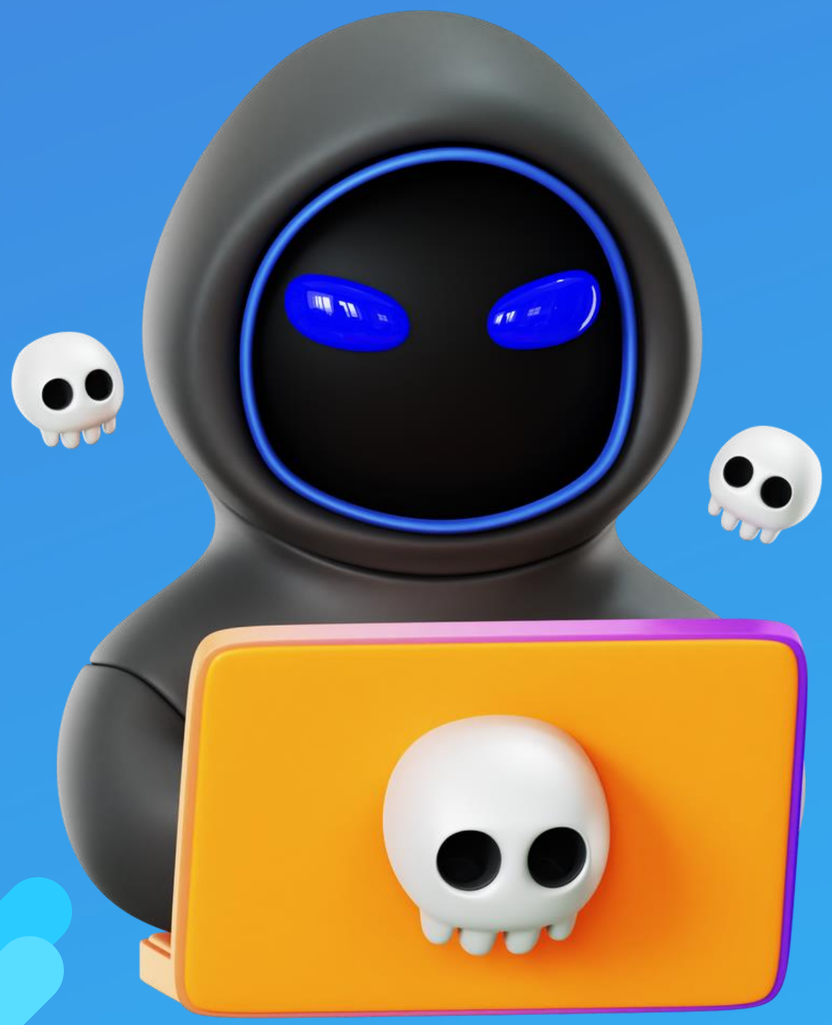
Do đó, bộ lọc mới có bộ nhớ 128-bit, nhận một số nguyên 128-bit và xuất ra một số nguyên (16x4) bit . Tỷ lệ nén của bộ lọc này là (128:64), nhỏ hơn (32:8) trong bộ lọc được đề xuất trước đó. Sự thay đổi tỷ lệ này là dành cho tốc độ, nhưng có thể làm suy yếu security . Để bù lại điều này, hàm đầu ra lấy phép XOR của 16 MSB và 16 LSB của y[i], i = 3, 2, 1, 0

II. Mã dòng CRYPTMT version3



2.4. Conversion to 8-bit integers (Biến đổi thành số nguyên 8 bit):

Vì đầu ra của bộ lọc là số nguyên (16×4)-bit và đặc điểm yêu cầu là đầu ra số nguyên 8 bit, chúng ta cần phân tích chúng thành số nguyên 8 bit.
Ta có:
LOWER16 := (0x0000ffff, 0x0000ffff, 0x0000ffff, 0x0000ffff)
UPPER16 := (0xffff0000, 0xffff0000, 0xffff0000, 0xffff0000)
Là 128 bit masks.
y0, y1, . . . , y2i, y2i+1, . . . là nội dung của bộ nhớ trong bộ lọc ở mỗi bước, tức là được tạo bởi yi+1 := f(yi, xi) trong
y2i và y2i+1 được dùng để tạo ra số nguyên đầu ra thứ i 128 bit zi theo công thức :
zi := [(y2i ⊕ (y2i >>32 16))&LOWER16] | [(y2i+1 ⊕ (y2i+1 <<32 16))&UPPER16]
với | là phép OR
zi được tách thành 16 trong số các số nguyên 8 bit từ các bit thấp hơn đến các bit cao hơn và được sử dụng làm đầu ra số nguyên 8 bit



III. Chống lại các tấn công thông thường

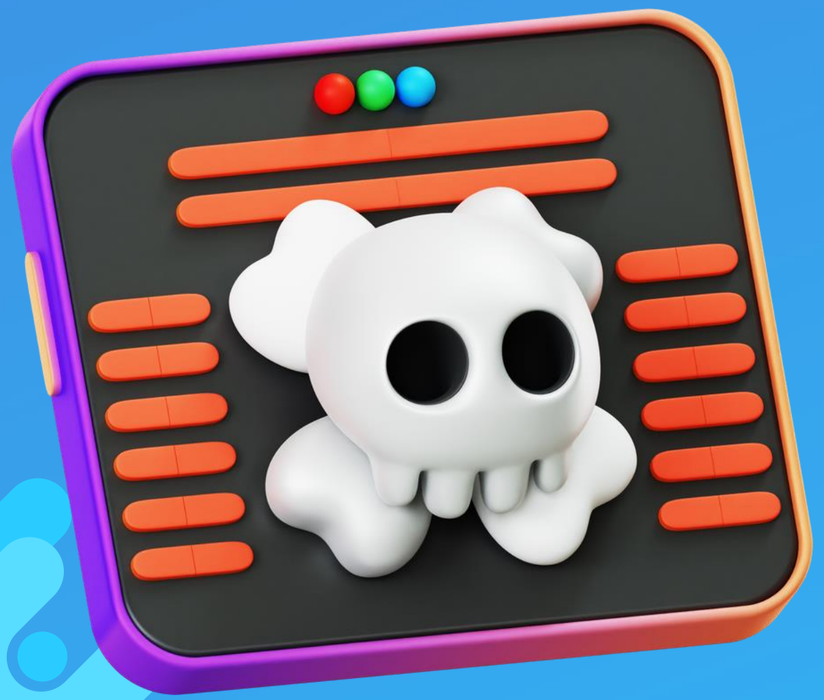
Một số thuộc tính của SFMT được yêu cầu trong phân tích mật mã:

1. SFMT là một máy tự động với không gian trạng thái S là một mảng của các số nguyên 128 bit có độ dài 156

2. Bất kỳ bit nào của luồng số nguyên 8 bit do CryptMT Ver.3 tạo ra đều có chu kỳ là bội số của 219937 − 1.

3. Chuỗi số nguyên 64 bit đầu ra của CryptMT Ver.3, được phân phối đều 156 chiều khuyết q . 2128 < 2159 , và do đó được phân bố đều 1248 chiều dưới dạng số nguyên 8 bit.

III. Chống lại các tấn công thông thường



Time-memory-trade-off attack:

Một cuộc tấn công đánh đổi thời gian-bộ nhớ “đơn thuần” tiêu tốn thời gian tính toán gần bằng căn bậc hai của kích thước không gian trạng thái, đó là O(219968+128) = O(210048)

Algebraic attack - Berlekamp-Massey attack:

không khả thi

Correlation attacks - distinguishing attacks:

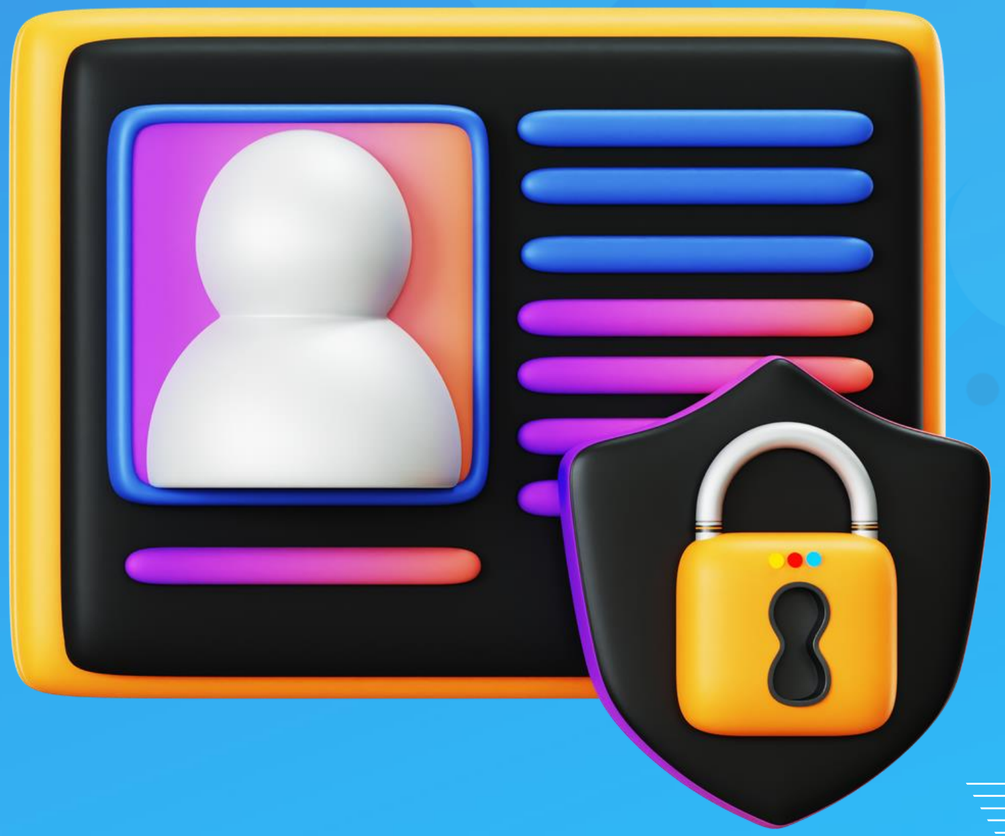
Nếu xem xét một cuộc tấn công phân biệt đơn giản đối với CryptMT Ver.3 có thứ tự ≤ 155, thì mức bảo mật của nó là 219937×2 , vì P/d = 219937 − 1. Do thuộc tính phân phối đều 156 chiều, các cuộc tấn công tương quan hình như không áp dụng được

IV. SO SÁNH - KẾT LUẬN:

Sử dụng công cụ kiểm tra hiệu suất từ eSTREAM để so sánh tốc độ của với CryptMT Ver.3, thu được kết quả:

- Tốc độ tạo số ngẫu nhiên của CryptMT Ver.3 nhanh hơn khoảng 1,8 lần so với phiên bản đầu tiên, và iv-setup nhanh hơn khoảng 48,7 lần. Bảng này cho thấy Version 3 có thể so sánh với các bộ sinh số ngẫu nhiên nhanh khác trong Pentium-M.

- Trình tạo số ngẫu nhiên của CryptMT Ver.3 nhanh thứ tư trong số các thuật toán được so sánh trong nền tảng này



THANK YOU!