

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN CÔNG NGHỆ TRI THỨC**

**NGUYỄN XUÂN HUY – TRẦN QUỐC HUY**

# **KHẢO SÁT MÃ DÒNG VÀ ỨNG DỤNG**

**KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT**

**TP. HCM, 2011**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**BỘ MÔN CÔNG NGHỆ TRI THỨC**

**NGUYỄN XUÂN HUY – 0712196**

**TRẦN QUỐC HUY – 0712204**

# **KHẢO SÁT MÃ DÒNG VÀ ỨNG DỤNG**

**KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT**

**GIÁO VIÊN HƯỚNG DẪN**

**PGS.TS. NGUYỄN ĐÌNH THỨC**

**KHÓA 2007 – 2011**

## NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

TpHCM, ngày ..... tháng ..... năm .....

Giáo viên hướng dẫn

## NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Khóa luận đáp ứng yêu cầu của Khóa luận cử nhân CNTT.

TpHCM, ngày ..... tháng ..... năm .....

Giáo viên phản biện

## LỜI CẢM ƠN

Chúng em xin chân thành cảm ơn Bộ môn Công nghệ Tri thức cũng như Khoa Công nghệ Thông tin, trường Đại học Khoa học Tự nhiên đã tạo điều kiện tốt cho chúng em thực hiện đề tài khóa luận tốt nghiệp này.

Chúng em xin chân thành cảm ơn thầy Nguyễn Đình Thúc đã tận tình hướng dẫn, chỉ bảo và đóng góp ý kiến cho chúng em trong suốt quá trình thực hiện đề tài.

Chúng em xin chân thành cảm ơn quý thầy cô trong Khoa đã tận tình giảng dạy và trang bị cho chúng em những kiến thức và kỹ năng quý báu trong những năm học tại trường vừa qua.

Chúng con xin nói lên lòng biết ơn sâu sắc đối với Ông Bà, Cha Mẹ đã chăm sóc, nuôi dưỡng chúng con thành người.

Xin chân thành cảm ơn các anh chị và bạn bè đã ủng hộ, giúp đỡ và động viên chúng em trong suốt thời gian học tập và nghiên cứu.

Mặc dù đã cố gắng hoàn thành khóa luận trong phạm vi và khả năng cho phép nhưng chắc chắn sẽ không tránh khỏi những thiếu sót. Chúng em kính mong nhận được sự cảm thông và tận tình chỉ bảo của quý thầy cô và các bạn.

Sinh viên

Nguyễn Xuân Huy – Trần Quốc Huy

Tháng 07/2011

## MỤC LỤC

Chương 1. MỞ ĐẦU.....	15
1.1. Lý do cần đến mã dòng hiện nay .....	16
1.2. Mục tiêu của đề tài .....	18
1.3. Yêu cầu của đề tài .....	20
1.4. Bố cục luận văn.....	20
Chương 2. LÝ THUYẾT MÃ DÒNG .....	23
2.1. So sánh mã dòng với mã khối .....	25
2.2. Phân loại mã dòng.....	27
2.3. Một số kiến trúc mã dòng.....	29
2.3.1. Mã dòng đồng bộ cộng.....	29
2.3.2. Mã dòng tự đồng bộ cộng .....	30
2.3.3. Mã dòng đồng bộ không cộng.....	31
2.3.4. Phương pháp mã dòng sử dụng mã khối.....	33
2.3.5. Mã phân phối hợp tác.....	36
2.4. Các loại Generator.....	40
2.4.1. Máy trạng thái hữu hạn và bộ sinh dòng khóa.....	41
2.4.2. Bộ sinh dựa trên bộ đếm .....	42
2.4.3. Bộ sinh số học .....	44

2.4.4. Bộ sinh dựa trên thanh ghi dịch chuyển.....	48
2.5. Trường hữu hạn $GF(p)$ và $GF(p^m)$ .....	57
2.5.1. Trường hữu hạn (trường Galois).....	57
2.5.2. Cách biểu diễn phân tử trong trường hữu hạn .....	59
2.5.3. Tính toán trên trường hữu hạn .....	61
2.6. Các khía cạnh mật mã của Sequence .....	64
2.6.1. Độ phức tạp tuyến tính và đa thức cực tiểu .....	64
2.6.2. Phân phối mẫu của dòng khóa .....	73
2.6.3. Hàm tương quan.....	74
2.6.4. Độ phức tạp cầu .....	77
2.7. Tính an toàn của mô hình mã dòng.....	81
2.7.1. Tính an toàn dựa trên kiến trúc mã dòng .....	82
2.7.2. Tính an toàn dựa trên các khía cạnh mật mã của dòng khóa .....	83
2.7.3. Tính an toàn dựa trên kiến trúc của generator .....	85
Chương 3. MÃ DÒNG TRÊN MẠNG DI ĐỘNG .....	98
3.1. Giới thiệu về mạng di động.....	99
3.1.1. Các chuẩn mạng di động.....	99
3.1.2. Bảo mật trên mạng di động .....	100
3.2. Mã dòng ZUC .....	101

3.2.1. Cấu tạo của ZUC.....	101
3.2.2. Cấu tạo và hoạt động của LFSR.....	102
3.2.3. Tái cấu trúc dãy bit.....	103
3.2.4. Hàm phi tuyến $F$ .....	104
3.2.5. Hoạt động của ZUC .....	106
3.3. Ứng dụng của ZUC .....	110
3.3.1. Mã hóa 128-EEA3.....	110
3.3.2. Chứng thực 128-EIA3.....	112
3.4. Tiêu chí thiết kế và tính an toàn của ZUC .....	114
3.4.1. Tiêu chí thiết kế LFSR .....	114
3.4.2. Tiêu chí thiết kế của BR.....	116
3.4.3. Thiết kế và tính an toàn của hàm phi tuyến $F$ .....	118
Chương 4. CHƯƠNG TRÌNH THỰC HIỆN .....	127
4.1. Giới thiệu.....	128
4.2. Mô hình ứng dụng.....	129
4.2.1. Yêu cầu chức năng chương trình .....	129
4.2.2. Phương pháp tạo keystream .....	129
4.2.3. Mô hình hoạt động của chương trình.....	130
4.2.4. Giao diện chương trình và hướng dẫn thực thi .....	131



4.3. Kết quả thực nghiệm .....	134
4.4. Tổng kết chương .....	135
KẾT LUẬN .....	137
HƯỚNG PHÁT TRIỂN .....	139
TÀI LIỆU THAM KHẢO.....	140
Phụ lục A. Một số thuộc tính mật mã khác của hàm Boolean .....	145
A.1. Bậc đại số của hàm Boolean .....	145
A.2. Độ miễn đại số của hàm Boolean.....	148
Phụ lục B. S-box trong AES .....	149
Phụ lục C. Một số khái niệm khác .....	150
C.1. Lượng tin .....	150
C.2. Các tiên đề ngẫu nhiên Golomb .....	151

## DANH SÁCH HÌNH VẼ

<i>Hình 1. Logo của tổ chức 3GPP. ....</i>	<i>16</i>
<i>Hình 2. Sự khác nhau giữa mã khối và mã dòng. ....</i>	<i>26</i>
<i>Hình 3. Mã dòng đồng bộ cộng. ....</i>	<i>28</i>
<i>Hình 4. Mã dòng tự đồng bộ cộng. ....</i>	<i>30</i>
<i>Hình 5. Keystream Generator như máy trạng thái hữu hạn tự điều khiển. ....</i>	<i>41</i>
<i>Hình 6. Bộ đếm với hàm ra phi tuyến. ....</i>	<i>43</i>
<i>Hình 7. Một số generator dựa trên bộ đếm. ....</i>	<i>43</i>
<i>Hình 8. Một mô hình của loại thanh ghi Fibonacci. ....</i>	<i>49</i>
<i>Hình 9. Một mô hình của loại thanh ghi Galois. ....</i>	<i>49</i>
<i>Hình 10. Mô hình generator sử dụng bộ trộn kênh. ....</i>	<i>50</i>
<i>Hình 11. Mô hình generator “dừng và chạy”. ....</i>	<i>51</i>
<i>Hình 12. Hoạt động của generator “bước luân phiên ” trong trường hợp đầu ra của thanh ghi điều khiển là 1. ....</i>	<i>52</i>
<i>Hình 13. Hoạt động của generator “bước luân phiên ” trong trường hợp đầu ra của thanh ghi điều khiển là 0. ....</i>	<i>52</i>
<i>Hình 14. Mô hình hoạt động của thanh ghi trong generator cơ. ....</i>	<i>53</i>
<i>Hình 15. Generator kết hợp phi tuyến. ....</i>	<i>54</i>
<i>Hình 16. Mô hình của Generator phép cộng. ....</i>	<i>55</i>

<i>Hình 17. Mô hình generator lọc.....</i>	<i>56</i>
<i>Hình 18. Mô hình NLFSR Galois.....</i>	<i>56</i>
<i>Hình 19. Mô hình NLFSR Fibonacci. ....</i>	<i>57</i>
<i>Hình 20. LFSR tổng quát thể hiện sự đệ quy. ....</i>	<i>65</i>
<i>Hình 21. Kiến trúc tổng quát của ZUC.....</i>	<i>102</i>
<i>Hình 22. Kiến trúc của S-box <math>S_0</math>.....</i>	<i>121</i>
<i>Hình 23. Mô hình hoạt động của ứng dụng Voice Chat ở chế độ công khai.....</i>	<i>130</i>
<i>Hình 24. Mô hình hoạt động của ứng dụng Voice Chat ở chế độ riêng tư.....</i>	<i>131</i>
<i>Hình 25. Giao diện chương trình SCVoiceChat-server.exe.....</i>	<i>132</i>
<i>Hình 26. Giao diện chương trình SCVoiceChat-Client.exe.....</i>	<i>133</i>
<i>Hình 27. Biểu đồ so sánh tốc độ thực thi giữa 128-EEA3 và AES. ....</i>	<i>135</i>

## DANH SÁCH BẢNG

<i>Bảng 1. Các độ phi tuyến của các hàm cân bằng.</i> .....	90
<i>Bảng 2. Khảo sát sự thay đổi của các hàm nhị phân thành phần <math>f_j</math> khi bit đầu vào thứ <math>i</math> bị thay đổi đối với S-box trong AES.</i> .....	96
<i>Bảng 3. S-box <math>S_0</math>.</i> .....	106
<i>Bảng 4. S-box <math>S_1</math>.</i> .....	106
<i>Bảng 5. Biến đổi <math>P_1</math>.</i> .....	121
<i>Bảng 6. Biến đổi <math>P_2</math>.</i> .....	121
<i>Bảng 7. Biến đổi <math>P_3</math>.</i> .....	121
<i>Bảng 8. Khảo sát sự thay đổi của các hàm nhị phân thành phần <math>f_j</math> khi bit đầu vào thứ <math>i</math> bị thay đổi đối với S-box <math>S_0</math> của hàm phi tuyến <math>F</math>.</i> .....	122
<i>Bảng 9. Khảo sát sự thay đổi của các hàm nhị phân thành phần <math>f_j</math> khi bit đầu vào thứ <math>i</math> bị thay đổi đối với S-box <math>S_1</math> của hàm phi tuyến <math>F</math>.</i> .....	124
<i>Bảng 10. So sánh các tính chất của S-box trong AES và hai S-box <math>S_0</math> và <math>S_1</math> trong hàm phi tuyến <math>F</math>.</i> .....	125
<i>Bảng 11. So sánh tốc độ thực thi giữa giải thuật 128-EEA3 và giải thuật AES.</i> .....	134

## THUẬT NGỮ, VIẾT TẮT VÀ KÝ HIỆU

GSM	Hệ thống thông tin di động toàn cầu
3GPP	Hiệp hội dự án đối tác thế hệ thứ 3
DACAS	Trung tâm nghiên cứu an toàn tuyến thông và bảo mật dữ liệu của Viện hàn lâm khoa học Trung Quốc
UMTS	Hệ thống viễn thông di động toàn cầu
AES	Chuẩn mã hóa Advanced Encryption Standard
DES	Chuẩn mã hóa Data Encryption Standard
CD	Mã phân phối hợp tác
SG	Bộ sinh dãy
FSM	Máy trạng thái hữu hạn
GF	Trường Galois (ví dụ $GF(2^n)$ )
NSG	Bộ sinh dãy tự nhiên
LFSR	Thanh ghi dịch chuyển hồi tiếp tuyến tính
ZUC	Phương pháp mã dòng ZUC
SAC	Strict Avalanche Criterion
ANF	Dạng chuẩn đại số của hàm Boolean
S-box	Bảng thay thế
$\oplus$	Phép XOR luận lý

$\boxplus$	Phép cộng trong module $2^{32}$
$a \parallel b$	Phép nối hai dãy bit $a$ và $b$
$a_H$	Lấy 16 bit bên trái của số nguyên $a$
$a_L$	Lấy 16 bit bên phải của số nguyên $a$
$a \lll_n k$	Quay có nhớ thanh ghi $a$ (dài $n$ bit) về bên trái $k$ bit
$a \gg 1$	Dịch phải số $a$ 1 bit
$(a_1, a_2, \dots, a_n) \rightarrow (b_1, b_2, \dots, b_n)$	Phép gán các giá trị $a_i$ cho giá trị $b_i$ tương ứng

## TÓM TẮT KHÓA LUẬN

### **Vấn đề nghiên cứu:**

Tìm hiểu và nghiên cứu các lý thuyết về mã dòng. Khảo sát mã dòng trên mạng di động. Hiện thực hóa ứng dụng Voice Chat sử dụng mã dòng ZUC để đảm bảo tính bí mật dữ liệu trên đường truyền, bên cạnh đó tiến hành thực nghiệm để chứng minh tốc độ của mã dòng nhanh hơn so với mã khối. Phân tích tính an toàn và thực nghiệm đo đạc các đặc tính mật mã quan trọng của mã dòng ZUC.

### **Hướng tiếp cận:**

Tìm hiểu các khái niệm căn bản về mã dòng.

Xác định và tiến hành xây dựng chương trình thực hiện sử dụng mã dòng ZUC.

Xác định các vấn đề nghiên cứu cụ thể của mã dòng.

Nghiên cứu các nguyên lý thiết kế mô hình mã dòng.

Nghiên cứu các lý thuyết về toán học liên quan đến mã dòng.

Nghiên cứu các đặc tính mật mã quan trọng ảnh hưởng đến tính an toàn của mô hình mã dòng.

Tìm hiểu một số mã dòng trên mạng di động.

Khảo sát chi tiết mô hình mã dòng ZUC.

Phân tích và thực nghiệm đo đạc các đặc tính mật mã quan trọng ở mã dòng ZUC.

Thực nghiệm so sánh tốc độ giữa mã dòng ZUC (thông qua thuật toán mã hóa 128-EEA3) và mã khối AES.

# Chương 1. MỞ ĐẦU

## Tóm tắt chương:

✚ Nội dung chương mở đầu trình bày lý do cần đến mã dòng hiện nay, mục tiêu và yêu cầu của luận văn. Tóm tắt của từng chương sẽ được trình bày trong phần bố cục luận văn.



## 1.1. Lý do cần đến mã dòng hiện nay

Ngày nay với sự phát triển vượt bậc của công nghệ thông tin và truyền thông đã đem lại rất nhiều những ứng dụng tiện dụng đến với người dùng. Xu hướng phát triển của công nghệ hiện đại là trên môi trường mạng, trong đó mạng di động đang và sẽ có nhiều hứa hẹn. Trong tương lai gần như mọi ứng dụng đều có thể đưa lên chiếc điện thoại gọn nhẹ. Vấn đề bảo mật ngày nay không chỉ cấp bách trong mạng internet toàn cầu, mà ngay cả ở mạng di động cũng rất cần được sự quan tâm. Nhu cầu đảm bảo bí mật khi thực hiện các cuộc gọi, hay các dịch vụ thông qua mạng di động là điều mà người dùng rất quan tâm. Điều này càng được quan tâm hơn khi có sự xuất hiện thêm hàng loạt những công nghệ mạng di động mới như GPRS, 3G, EPS (LTE – SAE), .... Các công nghệ này đều do tổ chức 3GPP công bố. Dưới đây là Logo của tổ chức 3GPP, được lấy từ trang web của tổ chức (<http://www.3gpp.org>):



*Hình 1. Logo của tổ chức 3GPP.*

Để đáp ứng các nhu cầu bảo mật trên mạng di động thì các công nghệ di động đều phải áp dụng các kỹ thuật mã hóa phù hợp. Trong tất cả các kỹ thuật mã hóa, mã dòng (stream cipher) là thích hợp để áp dụng trong mạng di động. Đây là một kỹ thuật mã hóa thuộc loại mã đối xứng (symmetric cryptography). Việc bảo mật bằng cách dùng mã dòng trong GSM có những mục đích như: mã hóa đảm bảo bí mật dữ liệu, chứng thực, đảm bảo tính toàn vẹn [2]. Có hai loại mã đối xứng đó là: mã khối (block

cipher) và mã dòng (stream cipher). Trong đó như ta đã biết, mã khối sẽ làm việc bằng cách chia khối dữ liệu cần mã hóa ban đầu thành những khối dữ liệu nhất định, nghĩa là phải biết trước kích thước cũng như bản thân khối dữ liệu đó. Các dữ liệu được lưu thông trên mạng di động điển hình nhất là dữ liệu của một cuộc gọi đường như không được biết trước kích thước, hay còn gọi là dữ liệu được sinh ra và biến thiên theo thời gian (time-varying). Do yêu cầu xử lý tín hiệu biến thiên theo thời gian này của mạng di động nên đòi hỏi kỹ thuật mã hóa áp dụng cũng phải thỏa mãn cơ chế này. Mã dòng hoạt động với biến đổi của nó biến thiên theo thời gian trên những khối bản rõ (plaintext) riêng biệt [1], các phần sau của luận văn sẽ làm sáng tỏ chi tiết về khả năng đáp ứng được các yêu cầu của mã dòng trên mạng di động. Đó là lý do cho thấy ***tầm quan trọng của việc ứng dụng mã dòng trong vấn đề bảo mật ở mạng di động***.

Nhìn về quá khứ, ta thấy kỷ nguyên của mã dòng thực sự là vào những năm 1960. Vào thời gian đó, rất nhiều tổ chức sử dụng đến mã dòng như: những nhu cầu của quân đội và ngoại giao, các tổ chức gián điệp, các tổ chức cung cấp dịch vụ viễn thông, các doanh nghiệp,... Những thiết bị mã hóa điện tử bán dẫn đã bắt đầu xuất hiện. Do các thiết bị này có bộ nhớ với dung lượng rất thấp nên mã dòng trở nên phổ biến hơn mã khối. Tuy nhiên ngày nay với sự phát triển công nghệ trên các thiết bị, các vấn đề đó không còn là trở ngại, nên mã khối lại chiếm ưu thế hơn. Bằng chứng là ngay cả trên nền tảng GSM, ở thế hệ thứ 3 mã khối Kasumi đã thay thế mã dòng A5/x ở thế hệ thứ 2. Trên công nghệ Wi-Fi, ở phiên bản IEEE 802.11a/b còn đang sử dụng mã dòng RC4, nhưng sang phiên bản IEEE 802.11i thì được thay thế bởi mã khối AES [6].

Nhưng không vì vậy mà mã dòng lại không thể phát triển được. Hội thảo ***The State of the Art of Stream Ciphers*** (SASC), một hội thảo chuyên về mã dòng được tổ chức bởi ECRYPT (<http://www.ecrypt.eu.org>), vẫn đang được thu hút. Ông Steve Babbage (công tác tại Vodafone Group R&D) có đề cập, mã dòng rất hữu dụng vì “***tốc***

*độ rất nhanh”, có hiệu lực và nhỏ gọn* đối với những thiết bị bị hạn chế như: những thiết bị có nguồn năng lượng (pin) thấp như trong **RFID**; hay như Smart cards (8-bit processors) [7]. Trong bài báo của mình ([6]), Adi Shamir (một trong những người phát minh ra RSA) có đề cập, ứng dụng mật mã của RFID được nghiên cứu rộng rãi ở Hàn Quốc, ông cho rằng nó sẽ là một công nghệ rất quan trọng và thành công trong thập kỷ tới. Và ông cũng mong đợi rằng các ứng dụng trên RFID này sử dụng mã dòng nhiều hơn là mã khối. Cuối cùng ông còn nhận xét rằng, tình trạng kiến thức và sự tự tin của chúng ta về mã dòng còn yếu. ***Nghĩa là chúng ta hoàn toàn có thể tin tưởng vào một tương lai của việc ứng dụng mã dòng.***

Các thuật toán bảo mật trong mạng GSM xuất phát từ ba thuật toán mã hóa là A3, A5 và A8. GSM sử dụng một số thuật toán đã có như A5/1, A5/2 và A5/3 cho việc bảo mật. Tuy nhiên chúng có thể bị bẻ bởi một vài các tấn công [3]. Ngày càng có thêm các thế hệ mới của mạng di động, như thế hệ mới nhất là công nghệ **EPS**, một công nghệ mới nhất đang được dự định phát triển lên thành thế hệ 4G. Bởi vậy hiện nay có những thảo luận về các thuật toán bảo mật mới để ứng dụng vào các công nghệ mới này, điển hình là các thảo luận những thuật toán của tổ chức **3GPP** như 128-EEA3 và 128-EIA3 cho công tác bảo mật trên công nghệ EPS [2].

Mã dòng thích hợp cho việc hiện thực hóa bằng phần mềm hay phần cứng. Nó rất thích hợp để cài đặt trực tiếp trên các thiết bị phần cứng có cấu hình thấp. Nên nó có thể được hiện thực hóa trên các máy điện thoại di động.

## **1.2. Mục tiêu của đề tài**

Với việc hiểu được nhu cầu cần thiết của mã dòng, chúng tôi tiến hành xây dựng ***chương trình thử nghiệm sử dụng mã dòng*** dựa vào mã nguồn mở đã có, từ đó nhận diện ra các vấn đề nghiên cứu liên quan.

Mã dòng là một chủ đề nghiên cứu rộng, đầy thách thức, và đang được các nhà nghiên cứu mã quan tâm vì khả năng ứng dụng quan trọng của nó trên mạng di động toàn cầu. Cơ sở lý thuyết của mã dòng có liên quan với Lý thuyết số [4] và lý thuyết về Trường (cụ thể là trường Galois hay Galoa), nên chắc chắn nó tận dụng được những phương pháp và lập luận mạnh của các lĩnh vực toán học này. Đây là một điểm đầy thử thách nhưng cũng rất thú vị đối với chúng tôi khi nghiên cứu về đề tài này. Với một mong muốn làm sáng tỏ những chân lý của cơ sở lý thuyết mã dòng, chúng tôi mạnh dạn đầu tư công sức để đi sâu tìm hiểu những cơ sở lý thuyết mã dòng ấy. Phần đầu của luận văn này trình bày những *cơ sở lý thuyết và các nguyên lý thiết kế các mô hình của mã dòng*.

Các thuật toán mã dòng thực chất được chia thành hai thành phần trong kiến trúc của nó. Một thành phần là quá trình làm việc của *bộ sinh dòng khóa* (keystream generator), và phần thứ hai nhận các *keystream* được sinh ra bởi bộ sinh dòng khóa này để tiến hành công việc mã hóa (hay chứng thực, đảm bảo tính toàn vẹn) của mình. Đối với các thuật toán mã dòng, phần thứ hai này có thể chỉ đơn giản là thực hiện nhiệm vụ XOR dòng khóa và bản rõ để tạo thành bản mã. Do đó tầm quan trọng của các thuật toán mã dòng tập trung chủ yếu vào các generator [4]. Luận văn đi sâu phân tích *kiến trúc và cơ chế hoạt động của các generator* khác nhau.

Đối với một thuật toán/mô hình mật mã nói chung hay mã dòng nói riêng, tính an toàn là yếu tố quan trọng hàng đầu. Do đó luận văn sẽ đi sâu phân tích *các khía cạnh mật mã liên quan đến tính an toàn của mô hình mã dòng*.

Luận văn tìm hiểu *một số mô hình mã dòng ứng dụng trong mạng di động*. Trong đó bao gồm cả các thuật toán chưa được công bố chính thức ứng dụng trong mạng di động cho những công nghệ mới, mà mới chỉ là những bản thảo. Điển hình là mã dòng ZUC [31] do **DACAS** (Trung tâm nghiên cứu an toàn tuyến thông và bảo mật dữ liệu của

Viện hàn lâm khoa học Trung Quốc) thiết kế, luận văn sẽ đi sâu phân tích mô hình mã dòng này.

### 1.3. Yêu cầu của đề tài

Nghiên cứu các cơ sở lý thuyết của mã dòng.

Phân tích, nắm rõ kiến trúc và nguyên lý hoạt động của các thuật toán mã dòng và generator tương ứng.

Tìm hiểu các mô hình mã dòng được ứng dụng trong mạng di động. Khảo sát chi tiết mô hình mã dòng ZUC.

Hiện thực chương trình minh họa.

Thực nghiệm, đo đạc các tính chất mật mã quan trọng của mô hình mã dòng ZUC.

### 1.4. Bố cục luận văn

Nội dung của luận văn được trình bày gồm:

**Chương 1. MỞ ĐẦU** trình bày lý do cần đến mã dòng hiện nay trong thực tế, mục tiêu thực hiện đề tài mã dòng, đồng thời xác định được các yêu cầu đặt ra của luận văn.

**Chương 2. LÝ THUYẾT MÃ DÒNG** trình bày và hệ thống hóa các kiến thức căn bản của mã dòng, đồng thời so sánh sự khác nhau giữa mã dòng và mã khối, nêu ra các loại kiến trúc mã dòng, các loại bộ sinh dòng khóa; giới thiệu lại lý thuyết về trường hữu hạn (trường Galois) đóng vai trò cơ sở toán học quan trọng để hiểu rõ các khái niệm liên quan đến mã dòng như: dòng khóa được sinh ra bởi bộ sinh, LFSR, S-box; trình bày các khía cạnh mật mã của dòng khóa: độ phức tạp tuyến tính và đa thức cực tiểu, phân phối mẫu, hàm tương quan, độ phức tạp cầu; sau cùng là trình bày và hệ thống phần rất quan trọng, đó là tính an toàn của mô hình mã dòng với các ý tưởng:

tính an toàn dựa vào kiến trúc mã dòng, tính an toàn dựa vào các khía cạnh mật mã của dòng khóa, đặc biệt là tính an toàn dựa vào kiến trúc của bộ sinh sẽ đi sâu phân tích và khảo sát các đặc tính mật mã quan trọng của hàm Boolean và S-box ảnh hưởng đến tính an toàn của bộ sinh như: tính phi tuyến (nonlinearity) và tiêu chuẩn SAC (Strict Avalanche Criterion) của hàm Boolean, tính đồng nhất sai phân của S-box.

**Chương 3. MÃ DÒNG TRÊN MẠNG DI ĐỘNG** trình bày giới thiệu về mạng di động và các thuật toán bảo mật đã có trên mạng di động; trình bày lại mô hình mã dòng ZUC và các ứng dụng của nó trong hai thuật toán bảo mật là: thuật toán mã hóa 128-EEA3 và thuật toán chứng thực thông điệp 128-EIA3; trình bày các tiêu chí thiết kế các lớp (layer) trong cấu tạo của ZUC, đặc biệt đi sâu phân tích và thực nghiệm đo đạc để kiểm tra các đặc tính mật mã quan trọng của hai S-box  $S_0$  và  $S_1$  trong hàm phi tuyến  $F$  là: tính phi tuyến của S-box, tính đồng nhất sai phân của S-box, tiêu chuẩn SAC và tính cân bằng (balance) của các hàm thành phần của S-box.

**Chương 4. CHƯƠNG TRÌNH THỰC HIỆN** trình bày kết quả về ứng dụng thử nghiệm Voice Chat, được hiện thực thông qua thuật toán mã hóa 128-EEA3 dùng generator ZUC để đảm bảo bí mật dữ liệu trên đường truyền giữa những người thực hiện cuộc hội thoại với nhau; trình bày mô hình của ứng dụng với các yêu cầu chức năng và mô hình hoạt động; thực nghiệm so sánh tốc độ giữa 128-EEA3 và AES; tổng kết các kết quả đạt được và chưa đạt được của chương trình thực hiện.


**Phụ lục A** trình bày một số đặc tính mật mã khác của hàm Boolean và của S-box ảnh hưởng đến tính an toàn của bộ sinh là bậc đại số (algebraic degree) và độ miễn đại số (algebraic immunity).

**Phụ lục B** trình bày lại cấu trúc và sự an toàn của S-box trong thuật toán mã khối AES.

**Phụ lục C** trình bày một số khái niệm khác như: lượng tin, các tiên đề ngẫu nhiên Golomb.

## Chương 2. LÝ THUYẾT MÃ DÒNG

### Tóm tắt chương:

 Chương 2 hệ thống hóa và khảo sát các lý thuyết liên quan đến mã dòng. Nội dung chương này trình bày các vấn đề chính sau:

- Trình bày tóm tắt mã dòng và so sánh sự khác nhau giữa mã dòng và mã khối.
- Trình bày các loại mã dòng: mã dòng đồng bộ và mã dòng tự đồng bộ; trình bày và phân tích tính chất của các kiến trúc mã dòng: mã dòng đồng bộ cộng, mã dòng tự đồng bộ cộng, mã dòng đồng bộ không cộng, phương pháp mã dòng sử dụng mã khối, mã phân phối hợp tác; trình bày các loại bộ sinh có thể được dùng trong mô hình mã dòng.
- Giới thiệu lại các kiến thức cần thiết về trường hữu hạn (trường Galois), đóng vai trò nền tảng để hiểu rõ các khái niệm liên quan đến mã dòng như: dòng khóa được sinh ra bởi bộ sinh, LFSR, S-box.
- Trình bày và hệ thống các khía cạnh mật mã của dòng khóa được sinh ra: độ phức tạp tuyến tính và đa thức cực tiểu, phân phối mẫu, hàm tương quan, độ phức tạp cầu.
- Hệ thống và phân tích các vấn đề liên quan đến tính an toàn của mô hình mã dòng, với 3 ý tưởng là: tính an toàn dựa trên kiến trúc mã dòng, tính an toàn dựa trên các khía cạnh mật mã của dòng khóa, tính an toàn dựa trên kiến trúc của bộ sinh. Ý tưởng về tính an toàn dựa trên kiến trúc của bộ sinh được khảo sát và phân tích kỹ lưỡng về các đặc tính mật mã quan

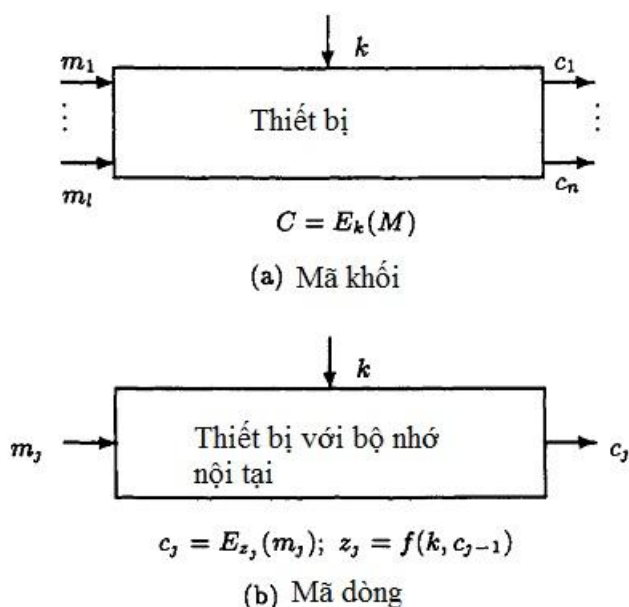


*trọng của hàm Boolean và S-box ảnh hưởng đến tính an toàn của bộ sinh như: tính phi tuyến (nonlinearity) và tiêu chuẩn SAC (Strict Avalanche Criterion) của hàm Boolean, tính đồng nhất sai phân của S-box.*

## 2.1. So sánh mã dòng với mã khối

Mã hóa đối xứng được chia làm hai loại là: *mã khối* (block ciphers) và *mã dòng* (stream ciphers).

**Đối với mã khối**, khi mã hóa, dữ liệu ban đầu được chia thành các khối (block) thường có kích thước bằng nhau, và kích thước này sẽ tùy thuộc vào thuật toán mã hóa được dùng như DES, 3DES, AES, RC2,... Nếu áp dụng DES thì các khối dữ liệu phải có kích thước là 64 bits, còn nếu áp dụng AES thì kích thước này phải là 128 bits. Mã khối cần đến một khóa  $k$  trong suốt quá trình mã hóa, khóa này cũng tùy thuộc vào thuật toán mã hóa áp dụng như trên. Trong thực tế khi áp dụng mã khối thì dữ liệu ban đầu phải biết trước về kích thước. Nghĩa là áp dụng mã khối cho dữ liệu đã biết trước cụ thể. Sau khi dữ liệu ban đầu được chia ra thành các khối có kích thước nhất định, quá trình mã hóa sẽ sử dụng đến một trong các *kiểu hoạt động* (mode of operation) để tạo thành bản mã tương ứng cho dữ liệu ban đầu. Các mode of operations như ECB, CBC, CFB, OFB, CTR.



Hình 2. Sự khác nhau giữa mã khối và mã dòng.

**Đối với mã dòng**, trong thực tế khi được áp dụng thì dữ liệu thường ở dạng biến thiên theo thời gian. Nghĩa là không biết trước được dữ liệu ban đầu. Mỗi phần của dữ liệu hiện tại sẽ được mã hóa cùng với một khóa  $z_j$  tương ứng,  $j \in [0, \infty)$ . Các  $z_j$  tạo thành một *dòng khóa* (keystream), mỗi  $z_j$  được gọi là một *keyword*. Hàm mã hóa đơn giản nhất trong thực tế có thể chỉ đơn giản là một phép XOR giữa các bits bản rõ và keystream tương ứng. Chính xác hơn là mỗi *ký tự* (character) của bản rõ XOR với  $z_j$ . Mô hình mã dòng sử dụng một khóa  $k$  ban đầu để sinh ra các  $z_j$ . Thực thể đảm nhiệm chức năng sinh dòng khóa này được gọi là *bộ sinh dòng khóa* (keystream generator). Ta có thể biểu thị keystream là  $z^\infty = z_0 z_1 z_2 \dots$  [4].

Một mô hình mã dòng có tính **tuần hoàn** (có chu kỳ - periodic) nếu keystream lặp lại sau  $d$  ký tự với  $d$  là giá trị cụ thể [4]. Nghĩa là số giá trị các keyword  $z_j$  là hữu hạn ( $d$  giá trị) mặc dù chuỗi keystream là vô hạn trong trường hợp tổng quát.

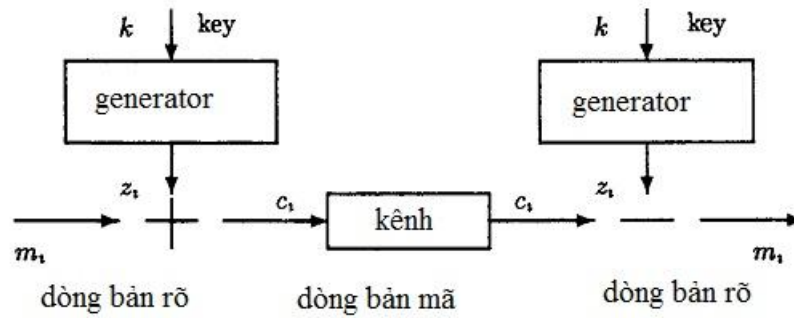
Hay ta có một định nghĩa tổng quát của mã dòng:

**Định nghĩa mã dòng** [16]: Cho  $K$  là một không gian khóa của một hệ mã và cho  $k_1 k_2 \dots \in K$  là một dòng khóa. Hệ mã này được gọi là một mã dòng nếu việc mã hóa trên chuỗi bản rõ  $m_1 m_2 \dots$  thu được bằng cách áp dụng lặp đi lặp lại của phép mã hóa trên những đơn vị thông điệp bản rõ,  $E_{k_j}(m_j) = c_j$ , và nếu  $d_j$  là nghịch đảo của  $k_j$ , việc giải mã xảy ra như  $D_{d_j}(c_j) = m_j$  với  $j \geq 1$ . Nếu tồn tại một giá trị  $l \in \mathbb{N}$  sao cho  $k_{j+l} = k_j$  với mọi  $j \in \mathbb{N}$ , ta gọi mã dòng tuần hoàn với chu kỳ  $l$ .

## 2.2. Phân loại mã dòng

Về căn bản một thuật toán mã dòng thuộc về một trong hai loại: **mã dòng đồng bộ** (synchronous cipher), và **mã dòng tự đồng bộ** (self-synchronous cipher) hay còn có tên gọi khác là **mã dòng bất đồng bộ** (asynchronous cipher). Tuy nhiên, những người từ dự án eSTREAM đã cho một định nghĩa tổng quát hơn về mã dòng, họ xem một mã dòng như một thực thể có một **trạng thái nội tại biến thiên theo thời gian** (time-varying internal state), và xem **mã dòng đồng bộ** và **mã dòng tự đồng bộ** là hai trường hợp đặc biệt [10].

Trong **mã dòng đồng bộ**, **trạng thái tiếp theo** (next state) của hệ thống mã hóa được mô tả độc lập với bản rõ và bản mã. **Trạng thái** (state) là giá trị của một tập hợp các biến mang lại duy nhất một sự mô tả cho trạng thái của thiết bị [1]. Ta hiểu trạng thái như là giá trị của một mảng nhiều phần tử. Thiết bị ở đây được hiểu như là một thành phần trong cấu tạo của bộ sinh dòng khóa (generator). Nó có thể là một thanh ghi (register) bao gồm nhiều phần.



Hình 3. Mã dòng đồng bộ cộng.

Hình trên diễn đạt quy tắc mã hóa và giải mã của mô hình **mã dòng đồng bộ cộng**. Khi mã hóa, lần lượt các ký tự bản rõ được “+” (cộng) với keyword  $z_i$  để sinh ra ký tự bản mã tương ứng. Khi giải mã thì làm ngược lại bằng cách “-” (trừ). “+” và “-” ở đây chỉ mang nghĩa đặc trưng cho quá trình mã hóa và giải mã. Chúng có thể chỉ đơn giản là phép XOR chẳng hạn. Từ hình rõ ràng ta thấy quá trình sinh keystream hoàn toàn độc lập với bản rõ và bản mã.

Ngược lại, đối với **mã dòng tự đồng bộ**, mỗi ký tự của keystream được suy ra từ một số  $n$  cố định của những ký tự bản mã trước đó. Vì vậy, nếu một ký tự bản mã bị mất hoặc bị hư (thay đổi) trong quá trình truyền dữ liệu, lỗi sẽ bị lan truyền cho  $n$  ký tự trong quá trình giải mã. Nhưng nó sẽ tự đồng bộ lại sau  $n$  ký tự bản mã nhận được [4]. Chẳng hạn ta khảo sát trong trường hợp  $n = 1$ :

**Giả sử ta có chuỗi các ký tự bản mã  $C$  bị thay đổi tại  $c_{j-1}$ .**

- **Khi dùng mã dòng tự đồng bộ** theo công thức mã hóa:  
 $c_j = E_{z_j}(m_j); z_j = f(k, c_{j-1})$ . Suy ra công thức giải mã:  
 $m_j = D_{z_j}(c_j); z_j = f(k, c_{j-1})$ . Ta thấy hiển nhiên  $c_{j-1}$  bị thay đổi thì kết quả giải mã  $m_{j-1}$  bị lỗi (không đúng như ban đầu trước khi mã hóa). Do  $c_{j-1}$  bị thay đổi làm cho  $z_j$  bị sai, nên kết quả giải mã  $m_j$  bị lỗi. Trong khi đó, việc

giải mã  $m_{j+1}$  lại phụ thuộc vào  $c_j$  ( $c_j$  không bị thay đổi) nên kết quả giải mã  $m_{j+1}$  không bị lỗi. *Như vậy chỉ cần sau một ký tự bản mã, quá trình giải mã đã tự đồng bộ. Điều này cũng đúng cho trường hợp  $c_{j-1}$  bị mất.*

- Còn **khí dùng mã dòng đồng bộ** theo công thức mã hóa:  $c_j = z_j \oplus m_j$ . Suy ra công thức giải mã  $m_j = z_j \oplus c_j$ . Trong trường hợp  $c_{j-1}$  **bị thay đổi** thì dễ dàng nhìn thấy quá trình giải mã chỉ bị lỗi tại  $m_{j-1}$ . Tuy nhiên, khi  $c_{j-1}$  **bị mất**, lúc đó chuỗi các ký tự bản mã *bị thụt lùi lại một ký tự*. Nghĩa là  $c_j$  đóng vai trò của  $c_{j-1}$ ,  $c_{j+1}$  đóng vai trò của  $c_j, \dots$ . Nói cách khác, kể từ  $c_{j-1}$  trở về sau tất cả các ký tự bản mã đều bị lỗi. Dẫn đến quá trình giải mã tất cả các ký tự sau đó đều bị lỗi.

Như trên ta đã giải thích về một sự khác nhau thú vị giữa hai loại mã dòng. Ngoài ra, *mã dòng tự đồng bộ* không có tính tuần hoàn bởi vì mỗi ký tự khóa  $z_j$  phụ thuộc vào toàn bộ các ký tự bản rõ trước đó [4]. Điều này thì ngược lại đối với *mã dòng đồng bộ* vì thông thường nó có tính tuần hoàn.

## 2.3. Một số kiến trúc mã dòng

Có nhiều phương pháp mã dòng khác nhau, thuộc vào những loại dưới. Đặc biệt với một số phương pháp, ta thấy được bóng dáng của mã khối trong việc ứng dụng vào mã dòng.

### 2.3.1. Mã dòng đồng bộ cộng

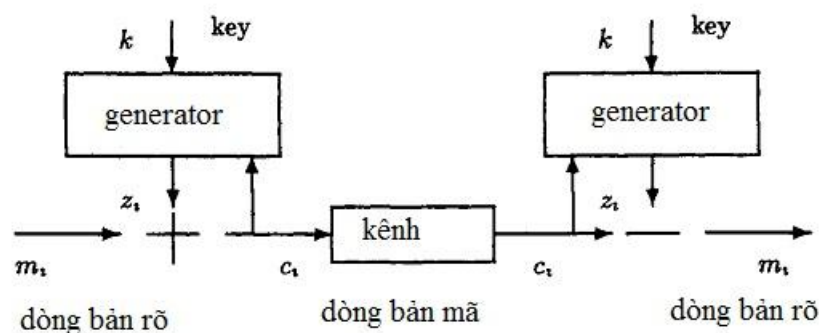
Như đã đề cập ở trên, *mã dòng đồng bộ cộng* (additive synchronous stream ciphers) sinh dòng khóa độc lập với dữ liệu bản rõ. Thuật toán sinh dòng khóa phải được thực hiện sao cho dòng khóa có thể được tái lập cho quá trình giải mã. Mã dòng đồng bộ cộng như theo *Hình 3* là một loại mã dòng đồng bộ quan trọng.

Như ở phần 2.2. *Phân loại mã dòng* đã giải thích về sự **đồng bộ** của loại kiến trúc mã dòng này. Còn tính “cộng” trong kiến trúc này có thể hiểu là do phép cộng/trừ giữa dòng khóa và bản rõ/bản mã, hay chỉ đơn giản là một phép XOR.

### Nhận xét:

*Vấn đề chính trong loại mã dòng này là thiết kế bộ sinh dòng khóa. Bởi vì việc kết hợp những ký tự bản rõ và bản mã là rất đơn giản, đòi hỏi bộ sinh dòng khóa cho mã dòng đồng bộ cộng phải được đủ mạnh [4].*

### 2.3.2. Mã dòng tự đồng bộ cộng



Hình 4. Mã dòng tự đồng bộ cộng.

Trong một mã dòng tự đồng bộ, mỗi ký tự dòng khóa nhận được từ một số  $n$  cố định của những ký tự bản mã trước đó. Phần 2.2. *Phân loại mã dòng* cũng đã giải thích về sự **tự đồng bộ** của kiến trúc mã dòng này. Những mã như *mã khóa tự động* (autokey ciphers) và *hệ thống mã hồi tiếp* (cipher feedback systems) là những ví dụ của **mã dòng tự đồng bộ cộng** (additive self-synchronous stream ciphers) [4].

Một *mã khóa tự động* có khóa nhận được từ dữ liệu bản rõ mà nó mã hóa. Một lớp quan trọng các mã dòng tự đồng bộ cộng khác, trong đó quá trình mã phản hồi tới *bộ sinh dòng khóa* như trong Hình 4.

### **Nhận xét:**

*Những vấn đề chính liên quan đến loại mã dòng này là việc thiết kế bộ sinh dòng khóa và cách mà ký tự bản mã phản hồi được dùng trong bộ sinh dòng khóa. Loại mã dòng này khó thiết kế và phân tích hơn do liên quan đến sự phản hồi [4].*

### **2.3.3. Mã dòng đồng bộ không cộng**

Cả hai loại **mã khối** và **mã dòng cộng** đều có những điểm **thuận lợi** và **bất lợi**. **Mã dòng đồng bộ cộng** có điểm bất lợi ở chỗ, với một cặp ký tự bản mã-bản rõ sẽ tiết lộ ngay ký tự khóa dòng tương ứng khi ký tự bản rõ được mã hóa. Điều này có thể tạo điều kiện cho một số loại **tấn công phục hồi khóa** (key-recovering attacks) như **tấn công tương quan** (correlation attacks) và **tấn công đụng độ** (collision attacks), **tấn công đương lượng-máy** (equivalent-machine attacks) như một tấn công dựa trên thuật toán Berlekamp-Massey, **tấn công xấp xỉ-máy** (approximate-machine attacks) dựa trên xấp xỉ tuyến tính. Một điểm thuận lợi của nó là khóa dòng **biến thiên theo thời gian** (time-varying), đảm bảo rằng cùng một ký tự bản rõ thường cho ra những ký tự bản mã khác nhau tương ứng ở các thời điểm khác nhau. Điều này thường che đậy một số thuộc tính xác suất của bản rõ [4]. Sở dĩ kiến trúc mã dòng này được gọi là **mã dòng đồng bộ không cộng** (nonadditive synchronous stream cipher) là bởi do nó không còn là mã dòng đồng bộ cộng, mà đã được nâng cấp từ mã dòng đồng bộ cộng cùng với mã khối để tạo nên một kiến trúc mã dòng an toàn hơn.

**Mã khối** có điểm bất lợi ở chỗ, khóa của nó không thể được thay đổi thường xuyên do vấn đề quản lý khóa, chỉ quy ước dùng duy nhất một khóa. Hơn nữa, cùng một khối (block) bản rõ luôn luôn cho ra tương ứng các khối bản mã giống nhau nếu một khóa được chọn và cố định. Điều này có thể tạo điều kiện cho nhiều tấn công như **tấn công sai phân** (differential attacks) trên một số khối bản mã thích hợp. Một điểm



thuận lợi của nó là có thể phát hiện sự thay đổi của bản rõ bởi vì bản rõ được mã hóa theo từng khối [4].

**Để giữ được các ưu điểm của cả hai loại mã dòng cộng và mã khối**, nhưng cũng để triệt tiêu các khuyết điểm của cả hai phương pháp, một *phương pháp mã khối động* (dynamic block ciphering approach) sẽ được mô tả như bên dưới. Với phương pháp này, một bộ sinh dòng khóa và một thuật toán mã khối biết trước được kết hợp với nhau. Các ký tự dòng khóa sinh ra bởi bộ sinh dòng khóa được dùng để làm **khóa động** của thuật toán mã khối cho mỗi khối bản rõ [4].

Cho một thuật toán mã khối với chiều dài khối bản rõ là  $n$ , gọi  $E_k(.)$  và  $D_k(.)$  là các ký hiệu tương ứng với hàm mã hóa và giải mã, ở đây  $k$  là khóa. Để dùng thuật toán mã khối cho việc mã hóa và giải mã động, một khóa động  $k_i$  cho thuật toán được sinh ra bởi một *bộ sinh dãy* (sequence generator) SG là  $(z_{ii}, z_{ii+1}, \dots, z_{ii+t-1})$ , ở đây  $t$  là một số nguyên dương, và  $z^\infty$  ký hiệu dãy được sinh ra bởi SG. Tham số  $t$  có thể là 1 hoặc một hằng số cố định khác. Vì vậy công thức mã hóa và giải mã được thể hiện như sau:

$$\begin{aligned}c_i &= E_{k_i}(m_i), \\m_i &= D_{k_i}(c_i),\end{aligned}$$

ở đây,  $m_i$  là khối bản rõ,  $c_i$  là khối bản mã ở lần thứ  $i$ . Bởi vì khóa  $k_i$  biến thiên theo thời gian, nên phương pháp mã này là mã khối động hay còn gọi là phương pháp **mã dòng đồng bộ không cộng**. Khóa của hệ thống bao gồm cả bộ sinh dòng khóa SG [4], **nghĩa là bản thân bộ sinh dòng khóa được sử dụng trong kiến trúc mã dòng này phải được giấu kín**.

**Ví dụ:** Giả sử ta xét trên thuật toán mã hóa AES với độ dài khóa là 128. Ta muốn biến nó trở thành thuật toán mã hóa khối động hay mã dòng đồng bộ không cộng, bằng cách sử dụng một generator sinh khóa động. Generator này sinh dãy bao gồm các keyword

với kích thước 32 bit. Như vậy khóa động  $k_i$  phải bao gồm 4 keyword, do đó  $k_i = (z_{4i}, z_{4i+1}, z_{4i+2}, z_{4i+3})$ . Trong trường hợp này  $t = 4$ .

### **Nhận xét:**

*Trong kiến trúc mã dòng này, thường không nhất thiết phải có một độ phức tạp tuyến tính (linear complexity) (xem phần: 2.6. Các khía cạnh mật mã của Sequences) lớn đối với dãy sinh ra của SG. Nếu hệ thống theo kiến trúc mã dòng đồng bộ không cộng được thiết kế tốt thì phần lớn những tấn công được biết đối với mã dòng cộng và mã khối không áp dụng được cho hệ thống này. Để tấn công nó, ta cần đến những phương thức mới [4].*

*Việc sử dụng những bộ sinh dãy nhanh và những thuật toán mã khối nhanh trong hệ thống, sẽ mang lại tốc độ cho mô hình mã dòng áp dụng kiến trúc này.*

### **2.3.4. Phương pháp mã dòng sử dụng mã khối**

Có một số loại **kiểu hoạt động** (mode of operation) của **mã khối**. Phổ biến là bốn loại: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback Chaining (CFB) và Output Feedback Chaining (OFB).

Trong **kiểu ECB**, quá trình mã (mã hóa, giải mã) được áp dụng theo từng khối độc lập. Cho  $M = M_1 M_2 \dots M_t$  là bản rõ, sau khi mã hóa thu được kết quả theo [4]:

$$C_i = E_k(M_i) \text{ với } i = 1, 2, \dots, t$$

Vì vậy bản mã tương ứng là  $C = C_1 C_2 \dots C_t$ . Sự giải mã được mô tả bởi:

$$M_i = D_k(C_i) \text{ với } i = 1, 2, \dots, t,$$

ở đây  $D_k(x)$  là hàm ngược của  $E_k(x)$ . Kiểu hoạt động này khá đơn điệu và cứng nhắc.

Trong **kiểu CBC**, các khối được kết lại nhau với một giá trị khởi tạo  $IV$ . Trong kiểu này ta giả sử rằng không gian bản rõ và bản mã là đồng nhất, và không gian khối (block space) này là một nhóm Aben (Abelian group) với toán tử  $+$ . Khối bản mã đầu tiên được xác định như [4]:

$$C_1 = E_k(M_1 + IV),$$

ở đây  $IV$  là một giá trị khởi tạo thuộc không gian khối. Các khối bản mã khác sau đó được tính như sau:

$$C_i = E_k(M_i + C_{i-1}) \text{ với } i = 2, 3, \dots, t$$

Để giải mã, khối bản rõ đầu tiên thu được như:

$$M_1 = D_k(C_1) - IV,$$

ở đây “ $-$ ” là toán tử ngược của “ $+$ ”. Những khối bản rõ khác sau đó được tính như:

$$M_i = D_k(C_i) - C_{i-1}, \text{ với } i = 2, 3, \dots, t.$$

Nếu ta so sánh các công thức mã của CBC trên với công thức mã của mã dòng tổng quát ở *Hình 2*, rõ ràng có thể xem kiểu CBC làm cho mã khối trở thành mã dòng với *bộ nhớ nội tại* (internal memory). Bộ nhớ nội tại trong CBC ở đây, có thể hiểu là để mã hóa  $C_i$  phải cần đến  $C_{i-1}$ , vậy phải cần một sự nhớ lại khối bản mã đã mã hóa được trước đó, điều này cần đến một “*bộ nhớ*”. Đối với *mã dòng đồng bộ cộng*, bộ nhớ nội tại này nằm trong bộ sinh dòng khóa của hệ thống, mà một ví dụ điển hình là **LFSR** (xem **Phần 2.4.4.1**). LFSR chính là *thanh ghi* (register) nếu hiện thực bằng phần cứng, nó đóng vai trò quan trọng trong việc tạo ra dòng khóa [4].

**Kiểu CFB** trong mã khối còn được dùng cho quá trình thực hiện mã dòng. Giả sử rằng ta có một mã khối với không gian khối bản rõ và bản mã là  $A^n$ , ở đây  $(A, +)$  là một

nhóm Aben. Cho  $E_k(x)$  là hàm mã hóa,  $rchop_u(x)$  là ký hiệu hàm có chức năng xóa bỏ  $u$  ký tự phải nhất của đối số  $x$ , và  $lchop_u(x)$  là ký hiệu hàm có chức năng xóa bỏ  $u$  ký tự trái nhất của đối số  $x$ . Một biến thể của kiểu CFB được mô tả như sau. Chọn  $m$  là số nguyên nằm giữa 1 và  $n$ . Mã dòng dựa trên mã khối xét trên  $(A^m, +)$ , ở đây toán tử “+” trên  $A^m$  là toán tử mở rộng của “+” trên  $A$ . Ví dụ:

$$(x_1, \dots, x_m) + (y_1, \dots, y_m) = (x_1 + y_1, \dots, x_m + y_m),$$

ở đây  $(x_1, \dots, x_m) \in A^m$  và  $(y_1, \dots, y_m) \in A^m$ . Chọn một giá trị khởi tạo  $X_1$ , việc mã hóa ký tự bản rõ thứ  $i$  ( $M_i \in A^m$ ) như sau [4]:

$$C_i = M_i + rchop_{n-m}(E_k(X_i)), \quad X_{i+1} = lchop_m(X_i) \parallel C_i,$$

ở đây  $\parallel$  là ký hiệu phép ghép (hai chuỗi dữ liệu). Còn giải mã như sau:

$$M_i = C_i - rchop_{n-m}(E_k(X_i)), \quad X_{i+1} = lchop_m(X_i) \parallel C_i.$$

Trong trường hợp này, kiểu CFB được thực hiện như mã dòng, cũng ***cần đến một thanh ghi nội tại***. Thanh ghi nội tại (internal register) này được dùng để cập nhật  $X_i$  như theo công thức  $X_{i+1} = lchop_m(X_i) \parallel C_i$ . Công thức này là một công thức quy nạp, rõ ràng việc tính giá trị  $X_{i+1}$  phải dùng đến giá trị của  $X_i$ . Do vậy giá trị  $X_i$  này phải được lưu trữ ở bước trước đó bởi thanh ghi và được cập nhật sau đó bởi  $X_{i+1}$ .

**Kiểu OFB** trong mã khối cũng được dùng cho quá trình thực hiện mã dòng. Như trong kiểu CFB, ban đầu một mã khối với không gian cả bản rõ và bản mã là  $A^n$ , ở đây  $(A, +)$  là một nhóm Aben. Mã dòng dựa trên mã khối được mô tả như sau. Không gian bản rõ và bản mã của mã dòng là  $A^m$ , ở đây  $m$  có thể được chọn tùy ý giữa 1 và  $n$ . Mã dòng có một thanh ghi nội tại để cập nhật giá trị  $X_i \in A^n$ . Cho  $X_1$  là giá trị khởi tạo của thanh ghi. Việc mã hóa ký tự bản rõ thứ  $i$  ( $M_i \in A^m$ ) như [4]:

$$C_i = M_i + \text{rchop}_{n-m}(E_k(X_i)), \quad X_{i+1} = E_k(X_i),$$

Giải mã được định nghĩa bởi:

$$M_i = C_i - \text{rchop}_{n-m}(E_k(X_i)), \quad X_{i+1} = E_k(X_i).$$

***Để thấy sự khác nhau duy nhất giữa CFB và OFB là sự cập nhật của thanh ghi nội tại.***

Trong bốn kiểu hoạt động của mã khối như ở trên, đã có ba kiểu có thể dùng để thực hiện mã dòng. ***Như vậy có rất nhiều cách sử dụng mã khối cho mã dòng. Ngay cả mã dòng đồng bộ không cộng như đã được đề cập ở phần trước cũng dựa trên mã khối.***

Kiến trúc mã phân phối hợp tác được trình bày ngay ở phần dưới đây cũng sử dụng đến mã khối.

### 2.3.5. Mã phân phối hợp tác

Hệ thống ***mã phân phối hợp tác*** (cooperatively distributed (CD) cipher) hay còn gọi là mã ***CD*** được thiết kế nhằm mục đích giữ được các ưu điểm của cả hai loại mã dòng cộng và mã khối, nhưng đồng thời cũng triệt tiêu các khuyết điểm của cả hai phương pháp trên [4].

Hệ thống mã phân phối hợp tác gồm có  $s$  thành phần:  $s$  thuật toán mã khối cho trước, với kích thước khối của tất cả là như nhau; thiết bị “điều khiển” quá trình mã (mã hóa hay giải mã) là một *bộ sinh dãy* với bộ nhớ nội tại, ký hiệu là SG. SG sinh ra dãy các phần tử trên tập  $Z_s = \{0, 1, \dots, s-1\}$ .

Cho  $k_0, \dots, k_{s-1}$  là các khóa tương ứng với các thuật toán mã khối cho trước;  $E_0(k_0, \bullet), \dots, E_{s-1}(k_{s-1}, \bullet)$  là các hàm mã hóa với các khóa tương ứng;

$D_0(k_0, \bullet), \dots, D_{s-1}(k_{s-1}, \bullet)$  là các hàm giải mã với các khóa tương ứng. Cho  $k_{sg}$  là khóa của bộ sinh dãy,  $z_i$  là ký tự sinh ra của SG tại thời điểm  $i$ . Ở mỗi thời điểm, chỉ duy nhất một trong các thuật toán mã khối đã cho được dùng đến (cho cả mã hóa lẫn giải mã). Chúng ta có công thức mã hóa [4]:

$$c_i = E_{z_i}(k_{z_i}, m_i).$$

ở đây  $m_i$  và  $c_i$  là khối bản rõ và bản mã thứ  $i$ . Tương tự, công thức giải mã được định nghĩa bởi:

$$m_i = D_{z_i}(k_{z_i}, c_i).$$

Trong kiến trúc mã CD này, SG quyết định hoạt động của mỗi thành phần mã khối, ***nó quyết định thành phần mã khối nào sẽ được dùng cho việc mã hóa/giải mã một khối dữ liệu tại một thời điểm.*** Có thể có trường hợp các hàm mã hóa  $E_0, \dots, E_{s-1}$  giống nhau, nhưng khi đó các khóa  $k_0, \dots, k_{s-1}$  sẽ phải khác nhau từng đôi một [4].

***Tính an toàn của kiến trúc mã dòng này có thể được phân tích thông qua ngữ cảnh bị tấn công*** như sau. ***Đầu tiên***, ta xem xét tấn công trên *mã khối*. Tất cả các tấn công trên mã khối được thực hiện dưới sự giả định rằng khóa được cố định và có duy nhất một thuật toán mã hóa (giải mã tương ứng). Những tấn công như *tấn công sai phân* và *tấn công tuyến tính*. Các tấn công này đều không thể áp dụng được tới hệ thống mã CD này với cách đơn giản, nếu chúng ta có ít nhất hai thuật toán mã hóa khác nhau hoặc ít nhất hai khóa khác nhau trên mã khối trong hệ thống này. ***Thứ hai*** phần lớn trong số các tấn công trên mã dòng áp dụng cho các bộ sinh dòng khóa của *mã dòng cộng*. Nếu hệ thống mã CD được thiết kế đúng đắn sao cho bộ sinh dòng khóa an toàn trước các tấn công, thì những tấn công đó sẽ không hiệu nghiệm.

Hệ thống mã CD là một quá trình thực hiện theo mã dòng, mặc dù nó là một sự tổ hợp của mã khối và mã dòng. Một thông điệp bản rõ thường tương ứng với các bản

mã khác nhau tại các thời điểm khác nhau. Mục đích của *sự hợp tác* và *phân phối* là để làm vô hiệu các tấn công được biết trên cả mã khối và mã dòng cộng [4].

Nếu hệ thống được thiết kế đúng đắn, ta có thể tạo một mã CD rất mạnh từ một số mã khối rất yếu và một bộ sinh dãy yếu. ***Điều này lại cho thấy sức mạnh của sự hợp tác và phân phối.***

Những thành phần và thiết bị điều khiển trong hệ thống CD sẽ được chọn một cách chu đáo. Dưới đây chúng ta xem xét hệ thống bao gồm hai thành phần mã khối [4].

**Cho**  $K_0$  và  $K_1$  là các không gian khóa của hai mã khối tương ứng. Giả sử rằng mỗi khóa có thể thuộc  $K_0$  hay  $K_1$ . Cho  $p_0 = \Pr(z = 0)$ ,  $p_1 = \Pr(z = 1)$  và

$$n_i(m, c) = |\{k_i \in K_i \mid E_i(k_i, m) = c\}|, i = 0, 1.$$

Cho  $\Pr(m, c)$  là xác suất sao cho  $c$  là một khối bản mã tương ứng của khối bản rõ  $m$ . Ta thấy rằng [4]:

$$\Pr(m, c) = p_0 \frac{n_0(m, c)}{|K_0|} + p_1 \frac{n_1(m, c)}{|K_1|},$$

$$\Pr(z = i; (m, c)) = p_i \frac{n_i(m, c)}{|K_i|}, i = 0, 1.$$

Áp dụng ***công thức Bayes*** ta có kết quả các xác suất có điều kiện sau:

$$\Pr(z = 0 \mid (m, c)) = \frac{|K_1| p_0 n_0(m, c)}{|K_1| p_0 n_0(m, c) + |K_0| p_1 n_1(m, c)},$$

$$\Pr(z = 1 \mid (m, c)) = \frac{|K_0| p_1 n_1(m, c)}{|K_1| p_0 n_0(m, c) + |K_0| p_1 n_1(m, c)}$$

Do đó, ta có hệ thức sau cho ***lượng tin trung bình*** (average mutual information) [5][40]:

$$I(z; (m, c)) = - \frac{|K_1| p_0 n_0(m, c)}{|K_1| p_0 n_0(m, c) + |K_0| p_1 n_1(m, c)} \times \log \frac{|K_1| p_0 n_0(m, c)}{|K_1| p_0 n_0(m, c) + |K_0| p_1 n_1(m, c)} \\ - \frac{|K_0| p_1 n_1(m, c)}{|K_1| p_0 n_0(m, c) + |K_0| p_1 n_1(m, c)} \times \log \frac{|K_0| p_1 n_1(m, c)}{|K_1| p_0 n_0(m, c) + |K_0| p_1 n_1(m, c)}$$

Để cực tiểu hóa lượng tin trung bình này thì:

$$p_0 \frac{n_0(m, c)}{|K_0|} = p_1 \frac{n_1(m, c)}{|K_1|}$$

Chú ý rằng:

$$\sum_{c \in C} \frac{n_0(m, c)}{|K_0|} = \sum_{c \in C} \frac{n_1(m, c)}{|K_1|} = 1.$$

Kéo theo:

$$p_0 = \sum_{c \in C} p_0 \frac{n_0(m, c)}{|K_0|} = \sum_{c \in C} p_1 \frac{n_1(m, c)}{|K_1|} = p_1.$$

Suy ra  $p_0 = p_1 = 1/2$  và  $\frac{n_0(m, c)}{|K_0|} = \frac{n_1(m, c)}{|K_1|}$ .

Với những phân tích trên, ta thu được nguyên tắc thiết kế sau. Cho hệ thống mã CD với hai thành phần mã khối, *các tham số* cần đạt các giá trị như sau [4]:

1.  $p_0 \approx \frac{1}{2}$ ;
2.  $\frac{n_0(m, c)}{|K_0|} \approx \frac{n_1(m, c)}{|K_1|}$ , và nếu một trong  $n_0(m, c)$  hay  $n_1(m, c)$  bằng 0, thì giá trị còn lại cũng phải bằng 0.

**Nhận xét:**



Rõ ràng một mã được an toàn chống lại các **tấn công dựa vào duy nhất bản mã** nếu nó được an toàn chống lại các **tấn công biết trước bản rõ**. Cho một số cặp khối bản rõ-bản mã, việc đầu tiên của một nhà thám mã là cố gắng lấy một ít thông tin về dòng khóa và sau đó cố gắng phục hồi lại khóa của SG hoặc xây dựng một bộ sinh để sinh ra kết quả giống như vậy, bằng cách phân tích các tham số  $n_0(m,c)$  và  $n_1(m,c)$  của hai mã khối đối với các cặp bản rõ-bản mã đã cho. Nếu hai mã khối không được thiết kế tốt, và nhà thám mã biết được  $n_0(m,c) = 0$ , thì sau đó anh ta biết ngay là giá trị sinh ra của bộ sinh là 1, nghĩa là mã khối được chọn là  $E_1$ . Nếu một tấn công trên SG thành công, thì sau đó nó chỉ còn việc tấn công vào hai mã khối theo một cách thông thường. Như vậy nghĩa là ý nghĩa của sự hợp tác bị mất đi. Nguyên tắc thiết kế trên được dụng ý để làm vô hiệu loại tấn công chia để trị này.

Mặt khác, SG sẽ được thiết kế sao cho dãy sinh ra của nó có các phân phối mẫu (pattern distribution) (xem phần: 2.6. Các khía cạnh mật mã của Sequences) tốt. Nếu dãy điều khiển (dãy kết quả sinh ra bởi SG) là 111...1000...0, thì sự hợp tác hiển nhiên rất yếu.

Một hệ thống CD có thể được an toàn hơn so với các mã khối. Nếu SG được thiết kế tốt, mã CD có thể tận dụng được các mã khối yếu [4].

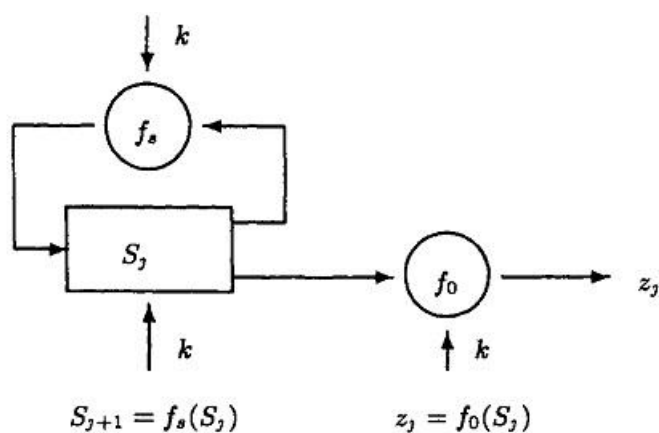
## 2.4. Các loại Generator

Như đã từng đề cập, **bộ sinh dòng khóa** (generator) là một thành phần quan trọng trong một mô hình mã dòng. Nó có nhiệm vụ sinh ra **dòng khóa** đáp ứng nhu cầu mã hóa và giải mã (cũng như đảm bảo tính toàn vẹn, chứng thực,...) trong một mô hình áp dụng mã dòng. Trong trường hợp tổng quát ta có thể nói, kết quả của bộ sinh dòng khóa là một **dãy** (sequence) hay một **dãy giả ngẫu nhiên** (ngẫu nhiên), hoặc là **các số giả ngẫu nhiên** (pseudo-random numbers).

Có nhiều loại generator khác nhau với cấu tạo và nguyên lý hoạt động để cho ra kết quả khác nhau. Có nhiều generator được mô hình bởi các **máy trạng thái hữu hạn** (finite state machine – FSM). Ta bắt đầu với việc đề cập tới các máy trạng thái hữu hạn này.

### 2.4.1. Máy trạng thái hữu hạn và bộ sinh dòng khóa

Các **máy trạng thái hữu hạn** là những hệ thống quan trọng cho việc mô hình hóa các thiết bị mật mã. Có những ví dụ tiêu biểu về các hệ thống mã dòng với một máy trạng thái hữu hạn có thể được mô hình bởi sự kết hợp của các *thanh ghi dịch chuyển* (shift-register) [8]. Các máy trạng thái hữu hạn là những đối tượng toán học quan trọng cho việc mô hình hóa phần cứng điện tử. Trong một *mã dòng đồng bộ*, *generator khóa chạy* (running-key generator) có thể được xem đại khái như một máy trạng thái hữu hạn tự điều khiển (autonomous), như được thể hiện ở Hình 5.



Hình 5. Keystream Generator như máy trạng thái hữu hạn tự điều khiển.

Keystream generator như một máy trạng thái hữu hạn gồm có một **bộ ra** (output alphabet) và một **tập trạng thái**, cùng với hai **hàm** và một **trạng thái khởi tạo**. **Hàm trạng thái tiếp** (next state function)  $f_s$  ánh xạ trạng thái hiện tại  $S_j$  thành một trạng thái mới  $S_{j+1}$  từ tập trạng thái, và **hàm ra** (output function)  $f_o$  ánh xạ trạng thái hiện tại  $S_j$

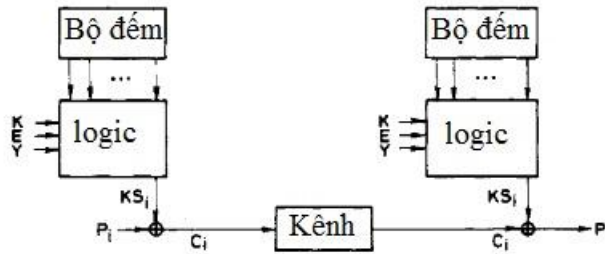
thành một ký tự ra  $z_j$  từ bộ ra. **Khóa  $k$**  có thể được dùng cho *hàm trạng thái tiếp* và *hàm ra* cũng như *trạng thái khởi tạo* [4].

Vấn đề cơ bản của việc thiết kế một keystream generator là để tìm một *hàm trạng thái tiếp*  $f_s$  và một *hàm ra*  $f_0$ , được đảm bảo để sinh ra một khóa chạy  $z^\infty$  thỏa mãn các yêu cầu mật mã nhất định như **độ phức tạp tuyến tính** lớn và **tính ổn định độ phức tạp tuyến tính** tốt, **tự tương quan** tốt, **phân phối mẫu** đều,...(xem **Phần 2.6**) [4].

Để có được những yêu cầu trên, những loại máy trạng thái hữu hạn đặc trưng được dùng như những generator khóa chạy. Đáng tiếc là lý thuyết về máy tự động điều khiển có hàm trạng thái phi tuyến đã không được phát triển tốt. Có nhiều loại keystream generator được đề xuất. Một số dễ thi hành (implement), nhưng tính an toàn của chúng khó điều khiển. Một số an toàn chống lại các loại tấn công nào đó, nhưng lại thi hành tương đối chậm. Các *generator số học* (number-theoretic generator) và *generator đếm* (counter generator) là những generator điển hình được mô hình từ các máy trạng thái hữu hạn [4].

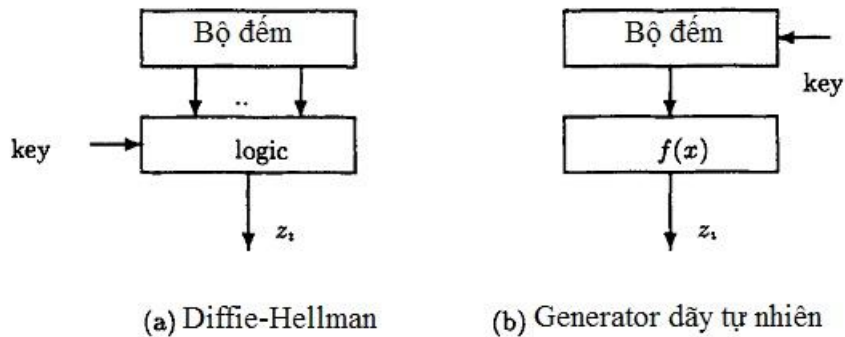
#### 2.4.2. Bộ sinh dựa trên bộ đếm

**Bộ đếm** (counter) là máy tự động đơn giản nhất có một **chu kỳ** (period), mà thường được lấy là  $q^n$ , ở đây  $q$  là một số dương. Một bộ đếm chu kỳ  $N$  đếm các số  $0, 1, \dots, N - 1$  theo chu kỳ. *Dãy kết quả* (output sequence) của bộ đếm có chu kỳ lớn, nhưng thiếu các thuộc tính được yêu cầu khác. Để đếm, dãy kết quả có thể được dẫn ra thông qua một *biến đổi kết quả phi tuyến* (nonlinear output transformation) [9]. Hay chính là việc ứng dụng một **hàm phi tuyến** (nonlinear function) cho một bộ đếm để xây dựng keystream generator như *Hình 6* [4].



Hình 6. Bộ đếm với hàm ra phi tuyến.

Trong loại generator này, **khóa** được dùng để điều khiển hàm (logic). Các **giá trị khởi tạo** của bộ đếm có thể được xem như một phần của khóa hoặc như một giá trị ngẫu nhiên. Một đề xuất đặc biệt được cho bởi **Diffie và Hellman** là dùng một thành phần cố định của một thuật toán mã khối như là **hàm ra** (logic) cho generator trong Hình 6 [4].



Hình 7. Một số generator dựa trên bộ đếm.

Nếu ta xem xét các bộ đếm với chu kỳ  $N$  bất kỳ, và dùng một hàm xác định  $f(x)$  từ  $Z_N$  vào một nhóm Aben  $G$ , ta có một generator như Hình 7(b). Trong generator này, **khóa  $k$**  là một trong các số nguyên  $0, 1, \dots, N - 1$ , và bộ đếm bắt đầu chu kỳ đếm của nó ở giá trị khóa. Các **đối số  $x$  của  $f(x)$**  là những giá trị nguyên liên tiếp được cung cấp bởi bộ đếm. Như vậy **dãy** hay **dòng khóa** sinh ra trong  $G$  được cho bởi [4]:

$$z_i = f((i + k) \bmod N),$$

ở đây **phần dư modulo**  $N$  nhận giá trị nguyên từ 0 tới  $N - 1$ .

Có ít điểm khác nhau giữa hai generator trong Hình 7. Trong **generator ở Hình 7(a)**, khóa hoặc một phần khóa được dùng để điều khiển **hàm ra**, trong khi trong **generator ở Hình 7(b)** hàm  $f(x)$  được xác định và khóa đơn giản là giá trị khởi tạo của **thanh ghi** (register). Generator ở Hình 7(b) được gọi là **bộ sinh dãy tự nhiên** (natural sequence generator – **NSG**), bởi vì mỗi **dãy tuần hoàn** (dãy có chu kỳ) có thể thu được bởi generator này theo *một cách tự nhiên*, và *hiệu quả cạnh an toàn của generator này* có thể được phân tích và điều khiển. **Mã dòng đồng bộ cộng** (additive synchronous stream cipher) dựa trên loại generator này được gọi là **mã dòng tự nhiên cộng** (additive natural stream cipher) [4].

#### **Nhận xét:**

*Một keystream generator được thiết kế không đúng đắn có thể bị bẻ bởi một tấn công sai phân hoặc một số tấn công khác. Nếu generator được thiết kế đúng đắn, NSG có thể kháng lại tất cả các tấn công đó [4]. NSG là đối tượng mà các nhà nghiên cứu mã hay đề cập trong các nghiên cứu mã dòng.*

### **2.4.3. Bộ sinh số học**

Một số hệ thống thực tế sớm nhất được dụng ý để hoạt động như các **generator số giả ngẫu nhiên** (pseudo – random number generator) hơn là các **generator dòng khóa** (keystream generator) [1]. Các số giả ngẫu nhiên được cần đến không chỉ trong mật mã, mà còn trong những mô phỏng số học cho các phương thức Monte Carlo, lấy mẫu, phân tích số học, kiểm tra sai sót của chip máy tính, lập trình máy chơi bạc (slot machine). Tuy nhiên, những ứng dụng khác nhau yêu cầu các thuộc tính ngẫu nhiên của các số khác nhau. Chẳng hạn những số ngẫu nhiên cho các mô phỏng (Monte Carlo) thì khác so với cho các mục đích mật mã [4]. Một số generator thuộc vào loại này như: **generator đồng dư tuyến tính** (congruential generator), **generator  $1/p$** ,

*generator lũy thừa* (power generator), *generator dựa trên phép mũ* (generator based on the exponential operation) [4].

### 2.4.3.1. Generator đồng dư tuyến tính

Một *generator đồng dư tuyến tính* (Linear Congruential Generator – LCG) thường được dùng để sinh ra các số ngẫu nhiên, và số tiếp theo  $X_{i+1}$  trong một dãy các số  $X_i$  được định nghĩa như cách sau:

$$X_{i+1} = (aX_i + b) \bmod M, \quad (2.4.3.1)$$

ở đây  $0 \leq X_i \leq M - 1$ . Trong đó  $(a, b, M)$  là những tham số định nghĩa generator và  $X_0$  là giá trị khởi đầu của dãy (có thể cho bộ bốn số  $a, b, M, X_0$  là **khóa** của generator). Các generator đồng dư tuyến tính được sử dụng rộng rãi trong thực tế ở các phương thức Monte Carlo, nhưng chúng **yếu về mặt mật mã** [4], với những minh chứng sau:

Rõ ràng phương pháp này **không có tính an toàn mật mã nếu môđun  $M$  được biết**. Trong trường hợp này, có thể giải tìm  $x$  nhờ vào đồng dư thức:  $(X_2 - X_1) \equiv x(X_1 - X_0) \bmod M$ ; (Giá trị của  $x$  chính là  $a$ , từ đó có thể tính được  $b$ ). Sau đó phần còn lại của dãy có thể được tính chính xác bằng cách dùng hệ thức:  $X_{i+1} = x(X_i) + (X_1 - x(X_0)) \bmod M$ , [11].

Ngoài ra, một vấn đề đặt ra là khi  $a, b, M$  và  $X_0$  đều không được biết thì phải tấn công như thế nào. Cần phải có **một cách thức suy đoán** để tìm các giá trị của dãy. Ta có thuật toán sau để giải quyết vấn đề này:

**Thuật toán Plumstead** [12]: Giả sử LCG được cho như theo công thức (2.4.3.1) với  $a, b, M$  và  $X_0$  không biết, và không có thuộc tính nào trong chúng được giả định gì, ngoài  $M > \max(a, b, X_0)$ . Thuật toán sẽ tìm một đồng dư thức:  $X_{i+1} = (\hat{a}X_i + \hat{b}) \bmod M$ , có thể

với  $a$  và  $b$  khác nhưng sinh ra dãy tương tự như đồng dư thức ban đầu. Quá trình suy đoán bao gồm hai giai đoạn như sau. Cho  $Y_i = X_{i+1} - X_i$ .

**Giai đoạn 1:** Trong giai đoạn này, ta sẽ tìm  $\hat{a}$  và  $\hat{b}$  như sau:

1. Tìm  $t$  nhỏ nhất sao cho  $d = \gcd(Y_0, Y_1, \dots, Y_t)$  và  $d$  chia hết  $Y_{t+1}$ .
2. Với mỗi  $i$  với  $0 \leq i \leq t$ , tìm  $u_i$  sao cho:

$$\sum_{i=0}^t u_i Y_i = d.$$

3. Gán  $\hat{a} = \frac{1}{d} \sum_{i=0}^t u_i Y_{i+1}$ , và  $\hat{b} = X_1 - \hat{a}X_0$ .

Giai đoạn này sẽ cho  $X_{i+1} = (\hat{a}X_i + \hat{b}) \bmod M$ , với tất cả  $i \geq 0$ .

**Giai đoạn 2:** Trong giai đoạn này, ta bắt đầu dự đoán  $X_{i+1}$  và nếu cần thiết có thể thay đổi môđun  $M$ . Khi một dự đoán  $X_i$  được thực hiện, giá trị đúng thực sự sẽ được sẵn sàng cho thuật toán suy đoán (để đối chiếu đúng – sai). Ban đầu, gán  $i = 0$ ,  $M = \infty$  và giả sử  $X_0$  và  $X_1$  được biết trước (chúng ta có thể tái sử dụng các số ở giai đoạn trước). Lặp các bước sau:

1. Gán  $i = i + 1$  và dự đoán:

$$X_{i+1} = (\hat{a}X_i + \hat{b}) \bmod M.$$

2. Nếu  $X_{i+1}$  không đúng, gán  $M = \gcd(M, \hat{a}Y_{i-1} - Y_i)$ .

**Phân tích thuật toán Plumstead:** rõ ràng mỗi bước trong cả hai giai đoạn được thực hiện trong thời gian đa thức theo độ lớn của  $M$ . Theo Plumstead, đã chứng minh rằng  $t$  trong *Giai đoạn 1* được giới hạn bởi  $t \leq \lceil \log_2 M \rceil$ , số dự đoán sai được thực hiện trong

Giai đoạn 2 được giới hạn bởi  $2 + \log_2 M$ . Vì vậy thuật toán tối ưu với độ phức tạp  $O(\log_2 M)$  trong trường hợp xấu nhất [12].

Một lần nữa với những minh chứng trên, ta có thể **kết luận**: generator đồng dư tuyến tính yếu về mặt mật mã.

### 2.4.3.2. Các generator số học khác

**Generator  $1/p$ :** có khai triển hữu tỉ [4]:

$$\frac{1}{p} = .d_0 d_1 d_2 \dots d_j d_{j+1} \dots$$

Khai triển này được thực hiện trong cơ số  $d$ . Ta nói  $(p, d)$  là các tham số định nghĩa generator, và khởi đầu là một vị trí xác định  $j$  của số ngẫu nhiên đầu tiên. Ví dụ, cho dãy ngẫu nhiên xác định bởi công thức:  $x_n = d_{j+n}$  với  $n \geq 0$ .

**Generator lũy thừa:** được định nghĩa bởi [4]:

$$x_{n+1} = x_n^d \bmod N,$$

trong đó  $(d, N)$  là các tham số định nghĩa generator và  $x_0$  là giá trị khởi đầu. Có hai trường hợp đặc biệt của generator lũy thừa, cả hai đều xảy ra khi  $N = p_1.p_2$  là tích của hai số nguyên tố lẻ phân biệt. Nếu  $d$  được chọn sao cho  $\gcd(d, \phi(N)) = 1$  ( $d$  và  $\phi(N)$  nguyên tố cùng nhau), thì ánh xạ  $x \rightarrow x^d$  là một hoán vị trên  $Z_N^*$ , và generator này còn được gọi là **generator RSA**. Ở đây  $\phi(N)$  là phi hàm Euler của  $N$ .

Nếu ta chọn  $d = 2$  và  $N = p_1 p_2$  với  $p_1 = p_2 \equiv 3 \pmod{4}$ , thì đây gọi là **generator bình phương** (square generator) [4].

**Generator số học dựa trên phép mũ:** được cho bởi [4]:



$$x_{n+1} = g^{x_n} \bmod N,$$

trong đó  $(g, N)$  là các tham số định nghĩa generator và  $x_0$  là giá trị khởi đầu.

**Nhận xét các bộ sinh số học:** các bộ sinh này có thể thực hiện khá chậm khi môđun  $M$  lớn. Bằng cách thay đổi các bộ sinh trên, có thể thu được một số **bộ sinh bit số học**. Chẳng hạn **bộ sinh bit RSA** hay **bộ sinh bình phương (Rabin)** với việc cho ra dãy các **bit bé nhất** (*least significant bit – LSB*). Các bộ sinh bit số học này có tốc độ nhanh hơn hẳn [13].

#### 2.4.4. Bộ sinh dựa trên thanh ghi dịch chuyển

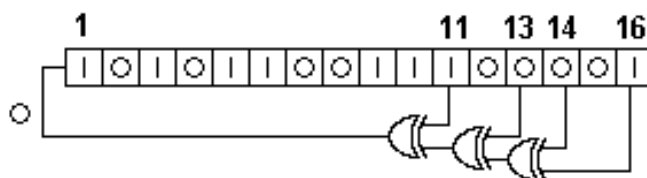
**Thanh ghi dịch chuyển** (shift register) thường được sử dụng để tạo bộ sinh vì hai lý do cơ bản. Đầu tiên các generator này tạo ra các dãy phù hợp với tinh thần của các tiên đề ngẫu nhiên Golomb (xem **Phụ lục**). Đồng thời hoạt động của các bộ sinh dựa trên thanh ghi dịch chuyển phù hợp cho việc phân tích bằng phương pháp đại số hơn.

##### 2.4.4.1. Thanh ghi dịch chuyển hồi tiếp tuyến tính

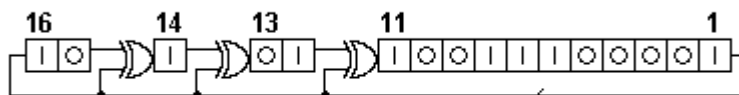
Trong các thanh ghi dịch chuyển thì **thanh ghi dịch chuyển hồi tiếp tuyến tính** (linear feedback shift register – LFSR) thường được áp dụng trong việc tạo các generator hơn vì loại thanh ghi này phù hợp cho cài đặt các xử lý có tốc độ cao và có thể được cài đặt trên phần cứng lẫn phần mềm. Ngoài ra nó còn mang các thuộc tính thống kê tốt.

Thanh ghi này được chia làm  $n$  ô (**stage**), mỗi ô được đánh số từ trái sang phải với các giá trị  $0, 1, 2, \dots, n-1$ . Mỗi ô đều chứa thông tin và thông tin của tất cả  $n$  ô trong thanh ghi được gọi là một **trạng thái** (state). Trong generator, sau một xung tín hiệu thanh ghi sẽ thay đổi trạng thái của nó: thông tin của stage  $i$  sẽ chuyển sang stage  $i-1$ , thông tin của stage 0 sẽ được lấy ra ngoài. Thông tin của stage  $n-1$  được tạo bằng cách sử dụng các phép biến đổi tuyến tính trên thông tin các stage khác, công việc này gọi là hồi tiếp

(feedback). Căn cứ vào cách thay đổi trạng thái người ta phân ra làm hai loại thanh ghi khác nhau: thanh ghi Fibonacci và thanh ghi Galois.



Hình 8. Một mô hình của loại thanh ghi Fibonacci.



Hình 9. Một mô hình của loại thanh ghi Galois.

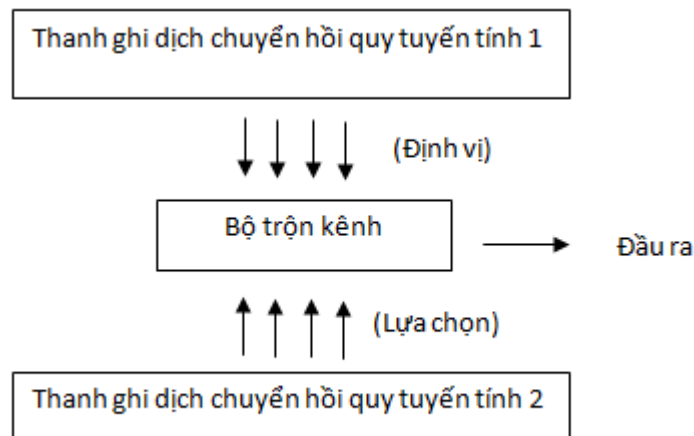
Do việc sử dụng các biến đổi tuyến tính nên các bộ sinh dựa trên loại thanh ghi này có thể tạo ra các chuỗi có độ phức tạp tuyến tính dễ đoán. Điều đó làm tăng nguy cơ tấn công dựa trên độ phức tạp tuyến tính. Người ta phải tìm cách sử dụng các biến đổi phi tuyến lên các bộ sinh. Có hai hướng tiếp cận vấn đề này như sau.

- Dùng **generator kết hợp** (combination generator): dùng nhiều thanh ghi và kết hợp các đầu ra của các thanh ghi này bằng một hàm phi tuyến.
- Dùng **generator lọc** (filter generator): chỉ sử dụng một thanh ghi và dùng một hàm phi tuyến để biến đổi trạng thái của thanh ghi thành keystream.

## 2.4.4.2. Generator kết hợp

### 2.4.4.2.1. Sử dụng bộ trộn kênh (Multiplexer)

Bộ trộn kênh là thiết bị vật lý dùng để thu nhận các tín hiệu từ một nguồn đầu vào. Việc thu nhận tín hiệu này lại chịu sự điều khiển từ một nguồn đầu vào khác. Generator có thể sử dụng bộ trộn kênh theo cách kết hợp hai thanh ghi lại với nhau. Sau một xung thời gian, generator lấy  $k$  bit từ thanh ghi thứ nhất và dùng một hàm biến đổi  $k$  bit này thành một số tự nhiên  $n$ . Sau đó generator sẽ lấy  $n$  bit từ thanh ghi thứ 2 làm keystream.

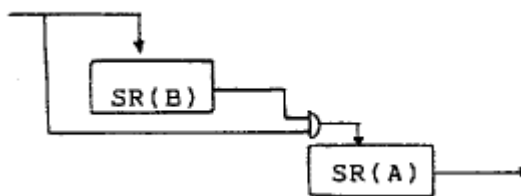


Hình 10. Mô hình generator sử dụng bộ trộn kênh.

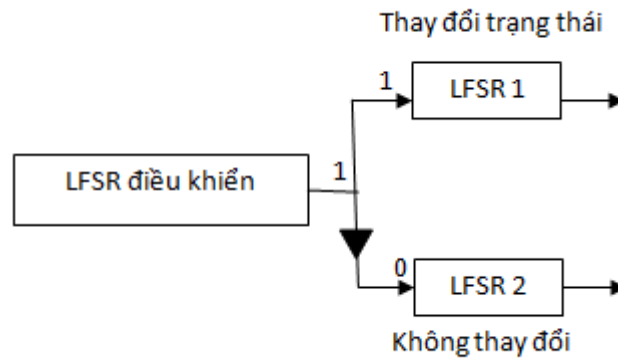
### 2.4.4.2.2. Điều khiển tín hiệu định thời

Sau một khoảng thời gian nhất định thanh ghi sẽ thay đổi trạng thái. Trong generator sử dụng phương pháp điều khiển tín hiệu định thời thì một thanh ghi sẽ quyết định sau khoảng thời gian đó một thanh ghi khác có được quyền thay đổi trạng thái không. Mô hình đầu tiên của generator loại này là generator “dừng và chạy” (stop-and-go generator) sử dụng hai thanh ghi. Khi thanh ghi thứ nhất thay đổi trạng thái nếu đầu ra của thanh ghi thứ nhất là 1 thì thanh ghi thứ hai sẽ thay đổi trạng thái, ngược lại thanh

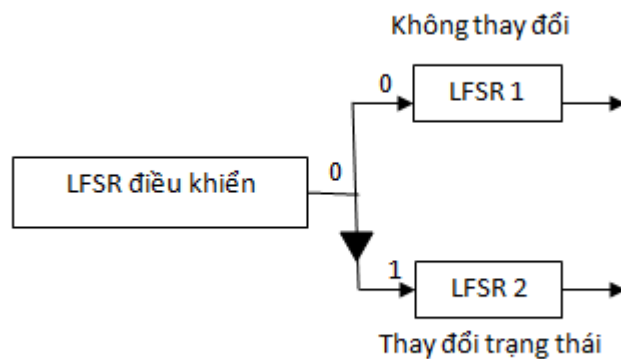
ghi thứ hai sẽ không thay đổi và đầu ra thanh ghi thứ hai sẽ giống với đầu ra của lần trước đó [23]. Gunther đã cải tiến loại generator này thành loại “generator bước luân phiên” (alternating step generator). Mô hình của Gunther sử dụng ba thanh ghi, thanh ghi thứ nhất sẽ quyết định việc thay đổi trạng thái của hai thanh ghi còn lại. Sau một xung thời gian, nếu đầu ra của thanh ghi thứ nhất là 0 thì thanh ghi thứ hai sẽ được thay đổi trạng thái còn thanh ghi thứ ba không thay đổi. Ngược lại nếu đầu ra của thanh ghi thứ nhất là 1 thì chỉ thanh ghi thứ ba được thay đổi trạng thái [24]. Một hướng cải tiến khác của generator “dừng và chạy” sử dụng kết nối dạng thác nước (cascade connection) nhiều thanh ghi sắp theo thứ tự trong đó đầu ra của thanh ghi phía trước sẽ quyết định việc thay đổi trạng thái của thanh ghi sau nó.



Hình 11. Mô hình generator “dừng và chạy”.



Hình 12. Hoạt động của generator “bước luân phiên” trong trường hợp đầu ra của thanh ghi điều khiển là 1.

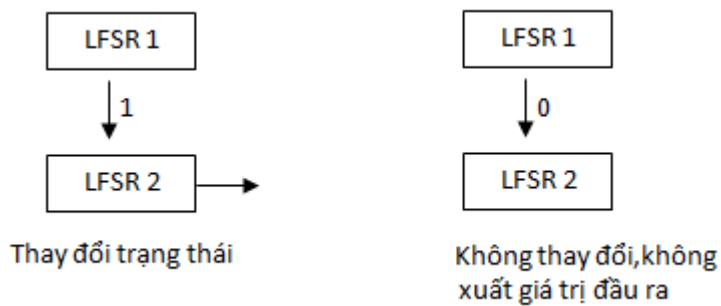


Hình 13. Hoạt động của generator “bước luân phiên” trong trường hợp đầu ra của thanh ghi điều khiển là 0.

#### 2.4.4.2.3. Generator co (shrinking generator)

Một trong những hướng kết hợp các thanh ghi khác của generator là phương pháp làm co đầu ra của các thanh ghi lại. Phương pháp này cũng được xem là dạng mở rộng của phương pháp điều khiển tín hiệu định thời. Người ta chia generator sử dụng phương pháp này làm hai loại: *generator co* và *generator tự co* (self-shrinking generator). Mô hình đơn giản của generator co gồm hai thanh ghi. Giống như generator “dừng và

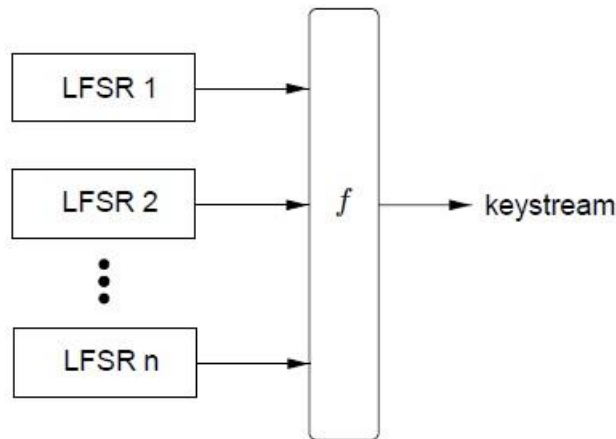
chạy” nếu bit đầu ra của thanh ghi thứ nhất là 1 thì thanh ghi thứ hai cập nhật trạng thái bình thường. Nhưng nếu bit đầu ra là 0 thì thanh ghi thứ hai không thay đổi và cũng không tạo ra bit đầu ra nào. Khi đó, trong cùng một thời gian thanh ghi thứ hai sẽ ra số lượng bit ít hơn một thanh ghi bình thường.



*Hình 14. Mô hình hoạt động của thanh ghi trong generator co.*

Trong mô hình generator tự co thay vì sử dụng một thanh ghi làm tham số điều khiển và một thanh ghi làm đầu ra người ta sẽ gom hai chức năng này vào cùng một thanh ghi [25]. Cách cài đặt này sẽ tiết kiệm không gian phần cứng trong khi số lượng bit đầu ra của generator cũng không thay đổi so với cách sử dụng hai thanh ghi.

#### **2.4.4.2.4. Generator kết hợp phi tuyến**

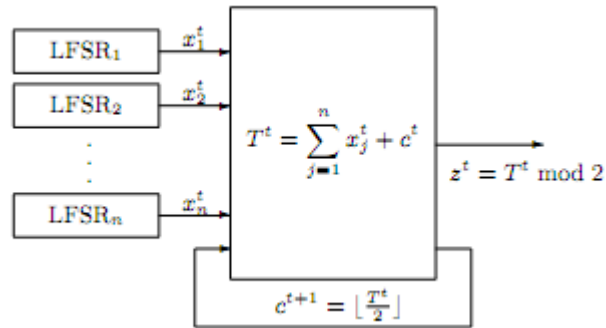


Hình 15. Generator kết hợp phi tuyến.

Keystream được sinh ra thông qua một hàm phi tuyến  $f$  của các kết quả sinh ra của các thành phần LFSR. Generator này được gọi là **generator kết hợp phi tuyến** (nonlinear combination generator), và  $f$  được gọi là **hàm kết hợp** (combining function). **Bậc phi tuyến** (nonlinear order) của  $f$  là giá trị lớn nhất trong các bậc của các số hạng xuất hiện trong biểu diễn đại số thông thường của nó. Ví dụ  $f(x_1, x_2, x_3, x_4, x_5) = 1 \oplus x_2 \oplus x_3 \oplus x_4 x_5 \oplus x_1 x_3 x_4 x_5$  có **bậc phi tuyến** là 4 [18]. **Bậc phi tuyến** của hàm Boolean còn có tên gọi khác là **bậc đại số** của hàm Boolean (xem thêm Phụ lục).

Một generator cụ thể thuộc loại này là **generator phép cộng** (summation generator).

Trong tin học phép cộng hai số nguyên (integer) sẽ được trình biên dịch tính toán trên byte. Đối với các phép toán mà trình biên dịch thực hiện đó, phép chèn số dư là một phép biến đổi phi tuyến trên bit. Tính chất này được áp dụng để tạo ra generator phép cộng. Generator phép cộng dùng nhiều thanh ghi, đầu ra của thanh ghi sẽ được đưa vào như là các số cùng hàng đơn vị của một phép cộng. Một thiết bị sẽ có nhiệm vụ cộng các giá trị đầu ra, tạo và lưu số dư đồng thời xuất ra kết quả cộng [26].



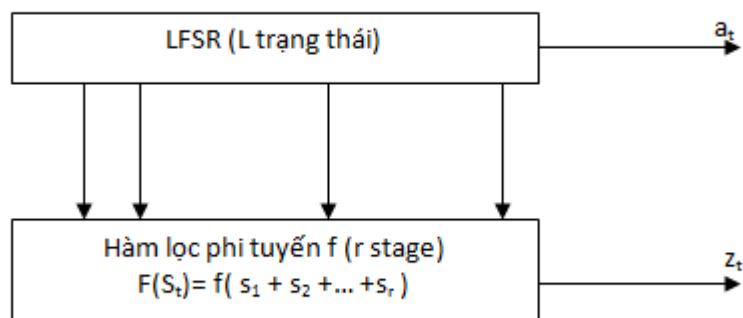
Hình 16. Mô hình của Generator phép cộng.

Hình 16 diễn đạt một mô hình generator phép cộng để sinh ra dòng khóa trên trường  $GF(2)$ .

#### 2.4.4.3. Generator lọc phi tuyến

Generator lọc là loại generator sử dụng một thanh ghi và một hàm phi tuyến  $f$  để tạo ra keystream. Hàm phi tuyến  $f$  có đầu vào trạng thái của thanh ghi nhưng thông thường hàm này chỉ dùng một số stage cố định trên thanh ghi. Đầu ra của hàm  $f$  là một bit hay một dãy bit để làm keystream. Như vậy độ an toàn của generator lọc sẽ phụ thuộc vào tính chất của hàm  $f$ . Loại generator này cũng được áp dụng trong các phương pháp mã hóa dựa trên generator ZUC (xem **Phần 3.2**).





Hình 17. Mô hình generator lọc.

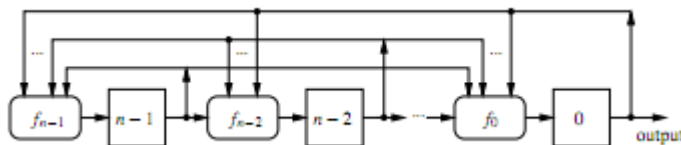
#### 2.4.4.4. Thanh ghi dịch chuyển hồi tiếp phi tuyến

**Thanh ghi dịch chuyển hồi tiếp phi tuyến** (nonlinear feedback shift register – NLFSR) cũng có cấu trúc giống như LFSR: gồm một dãy các stage và thay đổi trạng thái khi có tín hiệu định thời. Nhưng khác biệt ở chỗ **hàm hồi tiếp** (feedback function) trong NLFSR là một hàm phi tuyến. Ví dụ với  $x_i$  là thông tin các stage  $i$  của thanh ghi, hàm hồi tiếp trong LFSR sẽ có dạng tuyến tính, ví dụ:

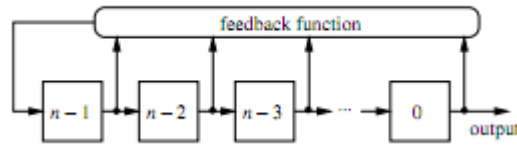
$$f(x) = x_1 + x_2 + x_3 + x_4.$$

Trong khi với NLFSR thì hàm hồi tiếp sẽ là một hàm phi tuyến, ví dụ:

$$f(x) = x_1 x_2 x_3 + x_4.$$



Hình 18. Mô hình NLFSR Galois.



Hình 19. Mô hình NLFSR Fibonacci.

Do tính chất biến đổi phi tuyến nên thanh ghi phi tuyến có độ an toàn hơn thanh ghi tuyến tính, các generator sử dụng thanh ghi phi tuyến không cần dùng các phương pháp “phi tuyến hóa” như kết hợp các thanh ghi hay lọc thanh ghi. Nhưng bù lại thanh ghi phi tuyến khó cài đặt hơn và chạy chậm hơn. Trong thực tế người ta thường kết hợp cả 2 loại thanh ghi này để tạo generator, như trong loại mã dòng Grain người ta sử dụng một thanh ghi LFSR 80-bit và một thanh ghi NLFSR 80-bit [27].

## 2.5. Trường hữu hạn $GF(p)$ và $GF(p^m)$

Phần này trình bày lý thuyết căn bản về các trường hữu hạn (finite field) cần thiết trong việc định nghĩa các dãy (sequence) cũng như *keystream* được tạo bởi các generator. Thật vậy, bản chất của dãy mà ta khảo sát được sinh ra bởi generator là bao gồm các phần tử (element), một phần tử có thể là một keyword trong dãy *keystream* (keystream sequence). Các phần tử này có thể thuộc về một trong hai trường hữu hạn  $GF(p)$  hay  $GF(p^m)$ .

Phần này được xem như là *một cơ sở toán học của mã dòng*.

### 2.5.1. Trường hữu hạn (trường Galois)

Trường với một số hữu hạn các phần tử được gọi là trường hữu hạn, với định nghĩa:

**Định nghĩa 2.5.1** [14]: Một trường hữu hạn  $\{F, +, \bullet\}$  gồm có một tập hữu hạn  $F$ , và hai phép toán  $+$  và  $\bullet$  thỏa mãn các tính chất sau:

$$1. \forall a, b \in F, a + b \in F, a \bullet b \in F$$

$$2. \forall a, b \in F, a + b = b + a, a \bullet b = b \bullet a$$

$$3. \forall a, b, c \in F, (a + b) + c = a + (b + c), (a \bullet b) \bullet c = a \bullet (b \bullet c)$$

$$4. \forall a, b, c \in F, a \bullet (b + c) = a \bullet b + a \bullet c$$

$$5. \exists 0, 1 \in F, a + 0 = 0 + a = a, a \bullet 1 = 1 \bullet a = a$$

$$6. \forall a \in F, \exists (-a) \in F \text{ sao cho } a + (-a) = (-a) + a = 0$$

$$\forall a \neq 0 \in F, \exists a^{-1} \in F \text{ sao cho } a \bullet a^{-1} = a^{-1} \bullet a = 1$$

Trường hữu hạn còn có tên gọi khác là **trường Galois** (Galois Field). Số phần tử trong một trường Galois có thể là **một số nguyên tố hoặc lũy thừa của một số nguyên tố**. Chẳng hạn  $GF(7)$ ,  $GF(8) = GF(2^3)$  và  $GF(2^8)$  có số phần tử tương ứng là 7, 8 và 256 là các trường Galois, còn  $GF(6)$  có số phần tử là 6 không phải là một trường Galois. Từ đây trở đi ta ký hiệu  $p$  là số nguyên tố.  $GF(p^m)$  là trường hữu hạn với  $p^m$  phần tử, còn được gọi là **trường mở rộng của  $GF(p)$**  và  $p$  được gọi là **đặc số** (characteristic) [14][21].

Một số định nghĩa khác liên quan đến trường hữu hạn  $GF(p^m)$ :

**Định nghĩa 2.5.2** [14]: *Bậc của một trường hữu hạn là số phần tử trong trường hữu hạn đó.*

**Định nghĩa 2.5.3** [14]: *Cho  $\alpha$  là một phần tử khác 0 của  $GF(p^m)$ , bậc của  $\alpha$  là số nguyên dương nhỏ nhất, ký hiệu  $ord(\alpha)$ , sao cho  $\alpha^{ord(\alpha)}$  là phần tử đơn vị của  $GF(p^m)$ .*

**Định nghĩa 2.5.4** [14]: Khi  $\text{ord}(\alpha) = p^m - 1$ ,  $\alpha$  được gọi là một **phần tử cơ bản** của  $GF(p^m)$ .

**Định nghĩa 2.5.5** [14]: Đa thức với các hệ số của nó là các phần tử của  $GF(p^m)$  được gọi là đa thức trên  $GF(p^m)$ .

**Định nghĩa 2.5.6** [14]: Một đa thức trên  $GF(p^m)$  là **đa thức bất khả quy** nếu nó không thể được phân tích thành nhân tử của các đa thức không tầm thường (bậc  $> 0$ ) trên trường tương tự ( $GF(p^m)$ ).

Trong một số tài liệu tiếng Việt, **đa thức bất khả quy** còn có tên gọi khác là **đa thức nguyên tố**.

## 2.5.2. Cách biểu diễn phần tử trong trường hữu hạn

Để biểu diễn một phần tử trong trường Galois, có nhiều cách khác nhau như: *biểu diễn lũy thừa* (power representation), *biểu diễn cơ sở thông thường* (normal basis), *biểu diễn cơ sở chuẩn* (standard basis) [14], .... Đây là các cách biểu diễn cho trường  $GF(p^m)$ , còn đối với trường  $GF(p)$  như ta đã biết các phần tử của nó là tập hợp  $\{1, 2, \dots, p-1\}$ . Ngoài ra  $GF(p)$  còn có cách ký hiệu khác là  $Z_p$ .

- Trong cách **biểu diễn lũy thừa**, tập hợp các phần tử của  $GF(p^m)$  có thể được biểu diễn như sau [14]:

$$\{0, 1, \alpha, \alpha^2, \dots, \alpha^{p^m-2}\}$$

ở đây  $\alpha$  là một phần tử cơ bản của  $GF(p^m)$ .

- Trong cách **biểu diễn cơ sở thông thường**, mỗi phần tử cơ sở được liên hệ đến bất kỳ một phần tử nào trong các phần tử cơ sở, bằng cách áp dụng *ánh xạ lũy*

thừa bậc  $p$  lặp đi lặp lại, ở đây  $p$  là *đặc số* (characteristic) của trường, điều đó nói rằng [14]:

Cho  $GF(p^m)$  là trường với  $p^m$  phần tử, và  $\beta$  là một phần tử của nó, sao cho  $m$  phần tử:

$$\{\beta, \beta^p, \beta^{p^2}, \dots, \beta^{p^{m-1}}\}$$

độc lập tuyến tính.

- Còn **biểu diễn cơ sở chuẩn** là một cách biểu diễn tự nhiên của các phần tử trường hữu hạn như các đa thức trên một *trường nền*, còn được gọi là biểu diễn đa thức. Định nghĩa cụ thể như sau [14]:

Cho  $\alpha \in GF(p^m)$  là **nghiệm** của một **đa thức bất khả quy** (irreducible polynomial) bậc  $m$  trên  $GF(p)$ . *Chuẩn* hay *cơ sở đa thức* của  $GF(p^m)$  là:

$$\{0, 1, \alpha, \dots, \alpha^{m-1}\}$$

Vì vậy trong biểu diễn này, mỗi phần tử của  $GF(p^m)$  được biểu diễn như một đa thức  $c_0 + c_1\alpha + c_2\alpha^2 + \dots + c_{m-1}\alpha^{m-1}$  trên  $GF(p)$ . *Trường nền* ở đây là  $GF(p)$ . Như vậy **cách biểu diễn cơ sở chuẩn dựa trên biểu diễn đa thức thông qua một đa thức bất khả quy**. Mỗi đa thức bất khả quy bậc  $m$  trên  $GF(p)$  định nghĩa duy nhất một trường  $GF(p^m)$ . Phần tiếp theo (2.5.3) giúp hiểu rõ vai trò của đa thức này trong cách biểu diễn này.

(Trong cách biểu diễn này, ta có thể không cần quan tâm đến nghiệm cụ thể  $\alpha$  của đa thức bất khả quy là gì, mà có thể đại diện bằng một biến  $x$ . Do đó ta còn có cách nói, mỗi phần tử của  $GF(p^m)$  được biểu diễn như một đa thức  $c_0 + c_1x + c_2x^2 + \dots + c_{m-1}x^{m-1}$  trên  $GF(p)$ ).

Do tính đơn giản của nó, nên cách **biểu diễn cơ sở chuẩn** được sử dụng rộng rãi [14]. Và trong luận văn này chỉ đề cập đến áp dụng của cách biểu diễn này cho các dãy được sinh ra bởi generator.

### 2.5.3. Tính toán trên trường hữu hạn

Thông thường hai trường  $GF(p)$  và  $GF(2^m)$  hay được ứng dụng trong mật mã, nên ta chú ý các phép tính trên hai trường này.

Ở đây, luận văn chỉ đề cập các phép tính căn bản trên trường  $GF(2^m)$ .

Cho hai phần tử  $A, B \in GF(2^m)$ , với biểu diễn cơ sở chuẩn tương ứng là  $A(x) = \sum_{i=0}^{m-1} a_i x^i$

và  $B(x) = \sum_{i=0}^{m-1} b_i x^i$  thông qua  $f(x)$  là đa thức bất khả quy trên  $GF(2)$  bậc  $m$ .

○ **Phép cộng:**  $C(x) \equiv (A(x) + B(x)) \bmod f(x) \equiv \sum_{i=0}^{m-1} (a_i \oplus b_i) x^i$ .

○ **Phép nhân:**  $C(x) = A(x) \bullet B(x) = \sum_{i=0}^{m-1} c_i x^i \equiv A(x) \cdot B(x) \bmod f(x)$

○ **Phép nghịch đảo:** Tính  $A^{-1} \bmod f(x)$ . Ta có thuật toán Euclid nhị phân mở rộng [15]:

**Thuật toán (tính nghịch đảo) Euclid nhị phân mở rộng (Binary Extended Euclidean Algorithm - BEA)**

**Input:**  $A \in GF(2^m), A \neq 0$

**Output:**  $A^{-1} \bmod f(x)$

---

1 :  $b \leftarrow 1, c \leftarrow 0, u \leftarrow A, v \leftarrow f.$

2 : **while**  $x$  divides  $u$  **do**

3 :            $u \leftarrow u/x.$

4 :           **if**  $x$  divides  $b$  **then**

5 :                    $b \leftarrow b/x.$

6 :           **else**

7 :                    $b \leftarrow (b+f)/x.$

8 :           **end if**

9 : **end while**

10 : **if**  $u = 1$  **then**

11 :           **return**  $b$

12 : **end if**

13 : **if**  $\deg(v) < \deg(u)$  **then**

14 :            $u \leftrightarrow v, b \leftrightarrow c.$

15 : **end if**

16 :  $u \leftarrow u + v, b \leftarrow b + c.$

17 : **goto** step 2

---

Thuật toán BEA chứa hai đẳng thức bất biến là  $bA + df = u$  và  $cA + ef = v$ , với  $d$  và  $e$  không được tính rõ ràng (không cần quan tâm trong thuật toán). Thuật toán kết

thức khi  $\deg(u) = 0$ , với trường hợp  $u = 1$  và do đó  $bA + df = 1$ , hay  $Ab \equiv 1 \pmod{f}$ . Vì thế  $b = A^{-1} \pmod{f(x)}$  [15].

**Ví dụ:** Xét trên trường  $GF(2^8)$ , cho đa thức  $f(x) = x^8 + x^6 + x^5 + x + 1$  bất khả quy trên  $GF(2)$ ,  $A(x) = x^7 + x^3 + x + 1$  và  $B(x) = x^4 + 1$  là hai phần tử thuộc  $GF(2^8)$  được biểu diễn dưới dạng đa thức thông qua  $f(x)$ . Ta tính:

$$\textbf{Cộng: } A(x) + B(x) = x^7 + x^4 + x^3 + x + (1 \oplus 1)x^0 = x^7 + x^4 + x^3 + x$$

**Nhân:**

$$A(x) \bullet B(x) = (x^7 + x^3 + x + 1) \cdot (x^4 + 1) \pmod{f(x)} = x^{11} + 2x^7 + x^5 + x^4 + x^3 + x + 1 \\ \pmod{x^8 + x^6 + x^5 + x + 1} = x^7 + x^2 + x$$

**Nghịch đảo  $B(x)$ :** Áp dụng thuật toán BEA, ta sẽ có được:

$$(x^4 + 1) \cdot (x^6 + x^4 + x^3 + x^2 + 1) + (x^8 + x^6 + x^5 + x + 1) \cdot x^2 = 1. \text{ Hay:}$$

$$(x^4 + 1) \cdot (x^6 + x^4 + x^3 + x^2 + 1) \equiv 1 \pmod{f(x)}. \text{ Suy ra:}$$

$$B^{-1} \pmod{f(x)} = x^6 + x^4 + x^3 + x^2 + 1.$$

Nếu xem trường  $GF(2^8)$  là tập hợp bao gồm các bytes. Ta có:

$$A(x) = 10001011 = A, \quad B(x) = 00010001 = B.$$

Cũng cùng với các kết quả như trên, khi thực hiện phép cộng, nhân (hai dãy bits) và nghịch đảo (của  $B$ ), ta sẽ thu được:

$$A + B = 10001011 \oplus 00010001 = 10011010,$$

$$A \bullet B = 10000110,$$

$$B^{-1} = 01011101.$$



## 2.6. Các khía cạnh mật mã của Sequence

Các *khía cạnh mật mã* (cryptographic aspect) của *dãy* (hay *dòng khóa*) là các đặc tính của dãy, là các nhân tố có thể cần thiết cho sự an toàn của một mã dòng nào đó.

Đối với dãy của *mã dòng đồng bộ cộng*, có một số độ đo mật mã về sức mạnh của nó như: *độ phức tạp tuyến tính* (linear complexity), *độ phức tạp cầu* (sphere complexity), *phân phối mẫu* (pattern distribution) và *tính tự tương quan* [4].

Sau đây là các khía cạnh mật mã của dãy trên trường hữu hạn. Ta nói một dãy trên một trường nghĩa là tất cả các phần tử của dãy đó thuộc trường đó.

### 2.6.1. Độ phức tạp tuyến tính và đa thức cực tiểu

#### A. Khái niệm chung về độ phức tạp tuyến tính và đa thức cực tiểu:

Cho một đa thức  $f(x)$  thuộc  $GF(q)[x]$  ( $q$  nguyên tố) có toán tử đa thức  $f(E)$  được định nghĩa như dưới. Nếu một dãy trên trường hữu hạn  $GF(q)$ , và  $f(x)$  trên  $GF(q)$  được cho bởi [4]:

$$f(x) = c_0 + c_1x + \dots + c_{L-1}x^{L-1},$$

ta định nghĩa:

$$f(E)s_j = c_0s_j + c_1s_{j-1} + \dots + c_{L-1}s_{j-L+1}.$$

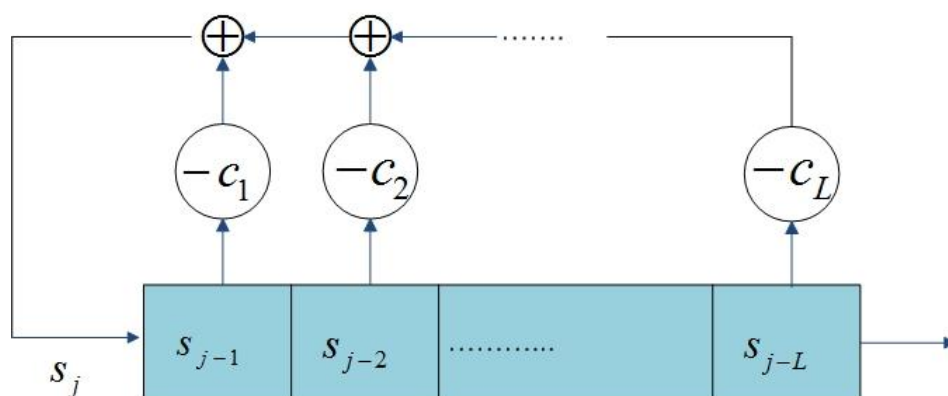
Cho  $s^n$  ký hiệu một dãy  $s_0s_1 \dots s_{n-1}$  với chiều dài  $n$  trên trường hữu hạn  $GF(q)$ . Nếu  $n$  là số hữu hạn, ta gọi dãy là *dãy hữu hạn*. Ngược lại ( $n = \infty$ ), ta gọi dãy là *dãy nửa vô hạn* (semi-infinite sequence). Một đa thức  $f(x) \in GF(q)[x]$  có  $\deg(f(x)) \leq l$  và  $c_0 \neq 0$  được gọi là *đa thức đặc trưng* của dãy  $s^n$  nếu:

$$f(E)s_j = 0 \text{ với mọi } j \text{ với } j \geq l.$$

Nếu các phương trình trên đúng cho  $l$ , thì chúng cũng đúng cho  $l+1$ . Vì vậy với mỗi đa thức đặc trưng, tồn tại một giá trị nhỏ nhất  $l \geq \deg(f)$  sao cho các phương trình trên đúng. Ta gọi  $l$  nhỏ nhất đó là *độ dài liên kết hồi tiếp* (associated recurrence length) của  $f(x)$  cho dãy. Ngoài ra *đa thức đặt trung* được gọi là *đa thức cực tiểu* (minimal polynomial) và *độ dài liên kết hồi tiếp* được gọi là *độ phức tạp tuyến tính* (linear complexity) của dãy. Độ phức tạp tuyến tính được ký hiệu là  $L(s^n)$ .

**Nhận xét (\*):** Nếu một dãy nữa vô hạn  $s^\infty$  là tuần hoàn, thì đa thức cực tiểu của nó là duy nhất với điều kiện  $c_0 = 1$ . Nếu dãy  $s^n$  tuần hoàn có đa thức cực tiểu là  $f$ , ta luôn có  $L(s^n) = \deg(f)$ .

**B. Độ phức tạp tuyến tính và đa thức cực tiểu của thanh ghi dịch chuyển hồi tiếp tuyến tính (Linear Feedback Shift Register – LFSR):**



Hình 20. LFSR tổng quát thể hiện sự đệ quy.

Trong LFSR với chiều dài  $L$  ở Hình 8, các khối đang chứa  $s_{j-1}$ ,  $s_{j-2}$ , ...,  $s_{j-L+1}$ ,  $s_{j-L}$  là những đơn vị nhớ hay đoạn (stage), và với mỗi nhịp đồng hồ (clock tick) giá trị của đơn vị nhớ phải nhất được xuất ra, trong khi các giá trị của các đơn vị nhớ khác được dịch chuyển qua bên đơn vị nhớ ngay bên phải một cách tuần tự [4]. Các giá trị khởi

tạo  $s_0, s_1, \dots, s_{L-1}$  của  $L$  đoạn trùng với  $L$  ký số (digit hay keyword) xuất ra đầu tiên của LFSR, các ký số xuất ra còn lại được tính duy nhất thông qua biểu thức đệ quy:

$$s_j = -\sum_{i=1}^L c_i s_{j-i}, \quad j = L, L+1, L+2, \dots.$$

Các ký số xuất ra và các hệ số hồi tiếp  $c_1, c_2, \dots, c_L$  được cho nằm trong cùng một trường, có thể là một trường hữu hạn  $GF(q)$  hoặc một trường vô hạn (như trường số thực). LFSR được gọi là không suy biến khi  $c_1 \neq 0$  [17].

**Định lý 2.6.1** [17]: *Nếu có một số LFSR với chiều dài  $L$  sinh ra dãy  $s^N = s_0, s_1, \dots, s_{N-1}$  nhưng không sinh ra được dãy  $s^{N+1} = s_0, s_1, \dots, s_{N-1}, s_N$ , thì bất kỳ LFSR có chiều dài  $L'$  sinh ra dãy  $s^{N+1}$  luôn thỏa mãn:*

$$L' \geq N+1-L.$$

Đa thức cực tiểu được gọi là đa thức hồi tiếp đối với LFSR. Các hệ số  $c_i$  như trong Hình 8 là các hệ số của đa thức:  $C(D) = c_0 + c_1D + c_2D^2 + \dots + c_LD^L$ . Đa thức này là đa thức hồi tiếp hay còn được gọi là đa thức kết nối của LFSR. Ta còn viết  $\langle L, C(D) \rangle$  là ký hiệu của một LFSR, với chiều dài  $L$  và đa thức kết nối là  $C(D)$  [18]. Theo Nhận xét (\*) ở trên, nên để LFSR có duy nhất một đa thức kết nối người ta cho  $c_0 = 1$ . Nên  $C(D) = 1 + c_1D + c_2D^2 + \dots + c_LD^L$ .

Nếu  $C(D) \in GF(q)[D]$  là một đa thức cơ bản (primitive polynomial) với bậc  $L$ , thì  $\langle L, C(D) \rangle$  được gọi là một maximum-length LFSR. Đa thức cơ bản  $C(D)$  là đa thức cực tiểu có nghiệm là phần tử cơ bản  $\alpha$  của  $GF(q^L)$ . Kết quả của một maximum-length LFSR với trạng thái khởi tạo khác không (non-zero) được gọi là *m-sequence*. *m-sequence* thỏa mãn các tiên đề ngẫu nhiên Golomb (xem Phụ lục) [18]. Ví dụ: Cho

$C(D) = 1 + D + D^4$  là một đa thức cơ bản trên  $GF(2)$ , LFSR  $\langle 4, 1 + D + D^4 \rangle$  là một *maximum-length* LFSR. Từ đó dãy kết quả của LFSR này là một *m-sequence* với chu kỳ tối đa có thể là  $N = 2^4 - 1 = 15$ .

Lưu ý rằng, ***một đa thức cơ bản cũng đồng thời là một đa thức bất khả quy, nhưng điều ngược lại thì không đúng.***

Ký hiệu  $L_N(s)$  là chiều dài nhỏ nhất của LFSR sinh ra dãy  $s^N$ .

**Định lý 2.6.2** [17]: *Nếu có một số LFSR với chiều dài  $L_N(s)$ , sinh ra dãy  $s^N = s_0, s_1, \dots, s_{N-1}$  và cả dãy  $s^{N+1} = s_0, s_1, \dots, s_{N-1}, s_N$ , thì  $L_{N+1}(s) = L_N(s)$ . Ngược lại, nếu có một số LFSR với chiều dài  $L_N(s)$  sinh ra dãy  $s^N$  nhưng không sinh được dãy  $s^{N+1}$ , thì  $L_{N+1}(s) = \max[L_N(s), N + 1 - L_N(s)]$ .*

**Định lý 2.6.2** làm cơ sở để thiết lập thuật toán Berlekamp-Massey [17] để tìm ra một LFSR ngắn nhất với chiều dài  $L_n(s)$  sinh ra dãy  $s_0, s_1, \dots, s_{n-1}$ . Cũng **theo Nhận xét (\*)**, vì do LFSR sinh ra dãy tuần hoàn nên  $L(s^n) = \deg(C(D)) = L_n(s)$ . Điều này có được là do bậc của đa thức kết nối luôn bằng chiều dài của LFSR. Từ đó có thể hiểu rằng thuật toán sau đây ***cũng đồng thời xác định được độ phức tạp tuyến tính của dãy  $s_0, s_1, \dots, s_{n-1}$ .***

### Thuật toán Berlekamp – Massey

**Input:** Dãy  $s^n = s_0, s_1, \dots, s_{n-1}$  với chiều dài  $n$ .

**Output:** LFSR ngắn nhất sinh ra  $s^n$  với *độ phức tạp tuyến tính*  $L(s^n)$  (chiều dài LFSR ngắn nhất) và *đa thức kết nối*  $C(D)$ .

1: **Khởi tạo:**  $1 \rightarrow C(D) \quad 1 \rightarrow B(D) \quad 1 \rightarrow x$

$$0 \rightarrow L \quad 1 \rightarrow b \quad 0 \rightarrow N$$

2: **Nếu**  $N = n$ , dừng. **Ngược lại**, tính:

$$d = s_N + \sum_{i=1}^L c_i s_{N-i}.$$

3: **Nếu**  $d = 0$ ,  $x+1 \rightarrow x$ , và **đi tới** bước 6.

4: **Nếu**  $d \neq 0$  và  $2L > N$

$$C(D) - db^{-1}D^x B(D) \rightarrow C(D)$$

$x+1 \rightarrow x$ , và **đi tới** bước 6.

5: **Nếu**  $d \neq 0$  và  $2L \leq N$

$$C(D) \rightarrow T(D); \quad [T(D) \text{ là biến tạm của } C(D)]$$

$$C(D) - db^{-1}D^x B(D) \rightarrow C(D)$$

$$N+1-L \rightarrow L$$

$$T(D) \rightarrow B(D)$$

$$d \rightarrow b$$

$$1 \rightarrow x$$

6:  $N+1 \rightarrow N$  và **quay về** bước 2.

Nếu độ phức tạp tuyến tính của một dòng khóa là  $L$ , thì chỉ cần một dòng khóa con của nó có chiều dài ít nhất  $2L$  (đóng vai trò là Input của thuật toán Berlekamp-Massey) là đủ để xác định được LFSR với chiều dài  $L$  sinh ra đầy đủ dòng khóa ban

đầu [18]. Vì vậy, độ phức tạp tuyến tính lớn là yếu tố mật mã cần nhưng không đủ của các dòng khóa đối với *mã dòng cộng*. Điều này sẽ được làm rõ khi ta giới thiệu về *độ phức tạp cầu* (sphere complexity). Tuy nhiên như ta đã có nói trong **Phần 2.3.3**, đối với một số *mã dòng không cộng* thì độ phức tạp tuyến tính lớn của mã dòng không phải là một yêu cầu mật mã cần thiết.

Từ **Định lý 2.6.2** và thuật toán Berlekamp-Massey, ta có được định lý sau:

**Định lý 2.6.3** [17]: *Giả sử thuật toán Berlekamp-Massey được áp dụng với dãy  $s^n = s_0, s_1, \dots, s_{n-1}$  và cho  $L, C(D), x$  và  $B(D)$  ký hiệu các giá trị khi thuật toán kết thúc. Nếu  $2L \leq n$ , thì  $C(D)$  là đa thức kết nối của LFSR **duy nhất** với chiều dài nhỏ nhất  $L$  sinh ra dãy. Nếu  $2L > n$ , thì tập các đa thức là tập các đa thức kết nối cho tất cả LFSR với chiều dài nhỏ nhất  $L$  sinh ra dãy.*

Một vấn đề nữa, theo *Khái niệm ở phần A* thì cả đa thức kết nối, cũng như dãy  $s^n$  đều đang xét trên cùng một trường  $GF(q)$ . Nhưng nếu xét trường hợp  $s^n = s_0, s_1, \dots, s_{n-1}$  là dãy trên trường  $GF(q^m)$ . Độ phức tạp tuyến tính của  $s^n$  đối với trường con  $GF(q)$ , ký hiệu  $L_{GF(q)}(s^n)$ , được định nghĩa là số tự nhiên nhỏ nhất  $L$  sao cho tồn tại các hệ số  $c_1, c_2, \dots, c_L \in GF(q)$  thỏa:

$$s_j + c_1 s_{j-1} + \dots + c_L s_{j-L} = 0, \text{ với mọi } L \leq j < n.$$

Lúc đó, theo các nhà nghiên cứu thì độ phức tạp tuyến tính  $L_{GF(q)}(s^n)$  là sự tổng quát hóa của độ phức tạp tuyến tính thông thường. Bất đẳng thức sau hiển nhiên đúng [4]:

$$L(s^n) \leq L_{GF(q)}(s^n).$$

(ta hiểu **độ phức tạp tuyến tính thông thường** là khi cả đa thức kết nối, cũng như dãy  $s^n$  đều đang xét trên cùng trường  $GF(q)$ )

➤ **Kết quả thực nghiệm thuật toán Berlekamp-Massey:**

Chúng tôi đã hiện thực thuật toán Berlekamp-Massey đối với dãy nhị phân (trên trường  $GF(2)$ ) để tính độ phức tạp tuyến tính và tìm ra LFSR sinh ra dãy nhị phân được cho trước. Sau đây là thuật toán ***Berlekamp-Massey đối với dãy nhị phân*** [18]:

**Input:** Một dãy nhị phân  $s^n = s_0, s_1, s_2, \dots, s_{n-1}$  với chiều dài  $n$ .

**Output:** Độ phức tạp tuyến tính  $L$  ( $0 \leq L \leq n$ ) của dãy và đa thức hồi tiếp  $C(D)$  của LFSR sinh ra dãy.

1. **Khởi tạo:**  $C(D) \leftarrow 1$ ,  $L \leftarrow 0$ ,  $m \leftarrow -1$ ,  $B(D) \leftarrow 1$ ,  $N \leftarrow 0$ .

2. **While** ( $N < n$ ) do

$$d \leftarrow (s_N + \sum_{i=1}^L c_i s_{N-i}) \bmod 2.$$

**If**  $d = 1$  then

$$T(D) \leftarrow C(D)$$

$$C(D) \leftarrow C(D) + B(D) \cdot D^{N-m}.$$

**If**  $L \leq N/2$  then

$$L \leftarrow N+1-L, \quad m \leftarrow N, \quad B(D) \leftarrow T(D).$$

$N \leftarrow N + 1.$

3. **Return**  $L, C(D).$

**Ví dụ** cho dãy nhị phân  $\mathbf{A} = 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,$   
 $1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1,$   
 $0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,$   
 $1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,$   
 $1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,$   
 $1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0,$   
 $0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0,$   
 $0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0.$

Kết quả chạy thuật toán Berlekamp-Massey cho được độ phức tạp tuyến tính của  $\mathbf{A}$  là **9**.

Theo như trên thì ta có thể chỉ dùng một dãy con với chiều dài ít nhất là  $2 \cdot 9 = 18$  để tìm ra được đa thức hồi tiếp  $C(D)$  của LFSR sinh ra  $\mathbf{A}$ . Ta thử kiểm chứng.

Ở đây ta dùng một dãy con bất kỳ có chiều dài 18, chẳng hạn dãy (trong phần in đậm trên của dãy  $\mathbf{A}$ )  $\mathbf{B} = 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0.$

Kết quả chạy thuật toán cho ta được độ phức tạp tuyến tính vẫn là 9 và đa thức kết nối  $C(D) = 1 + D^9.$

Như vậy thông qua kết quả thực nghiệm ta thấy, việc tấn công (chẳng hạn biết trước bản rõ) LFSR là hoàn toàn có thể thực hiện một cách đơn giản bằng thuật toán Berlekamp-Massey.

### **C. Một số vấn đề về đa thức cực tiểu:**

Như đã có đề cập, **độ phức tạp tuyến tính của dãy tuần hoàn trên trường hữu hạn thì đúng bằng bậc đa thức cực tiểu của nó.** Về mặt mặt mã, ta cần biết đến không chỉ



độ phức tạp tuyến tính của dãy mà còn đa thức cực tiểu [4]. Sau đây là một số kết quả của các nhà nghiên cứu về đa thức cực tiểu của dãy tuần hoàn.

**Khái niệm hàm sinh** (generating function hay formal power) [4]: *Hàm sinh của một dãy hữu hạn  $s^\infty$  trên  $GF(q)$  được định nghĩa bởi:*

$$s(x) = \sum_{i=0}^{\infty} s_i x^i.$$

Nếu  $s^\infty$  tuần hoàn với chu kỳ  $N$ , ta có:

$$(1 - x^N)s(x) = s^N(x) = \sum_{i=0}^{N-1} s_i x^i.$$

Các mệnh đề sau đây đúng:

**Mệnh đề 2.6.4** [4]: *Hàm sinh của mỗi dãy tuần hoàn  $s^\infty$  có thể được biểu diễn như:*

$$s(x) = \frac{g(x)}{f(x)} \quad (2.6.1)$$

với  $f(0) \neq 0$  và  $\deg(g) < \deg(f)$ .

Biểu thức trong (2.6.1) được gọi là *dạng tỉ số* (rational form) của hàm sinh  $s(x)$  của dãy  $s^\infty$ . Nếu  $\gcd(g(x), f(x)) = 1$ , thì nó được gọi là *dạng tỉ số rút gọn* (reduced rational form).

Ký hiệu  $f_s$  là đa thức cực tiểu của một dãy  $s^\infty$ . **Hai mệnh đề kinh điển** sau rất hữu ích:

**Mệnh đề 2.6.5** [4]: *Cho  $s^\infty$  là một dãy tuần hoàn trên  $GF(q)$  và:*

$$s(x) = \frac{r(x)}{f(x)}, \quad f(0) = 1$$

là một dạng tỉ số của hàm sinh của  $s^\infty$ . Khi đó  $f(x)$  là đa thức cực tiểu của dãy nếu  $\gcd(r(x), f(x)) = 1$ .

Mệnh đề sau áp dụng cho tổng của hai dãy tuần hoàn.

**Mệnh đề 2.6.6** [4]: Cho các dạng tỉ số rút gọn của hai dãy tuần hoàn  $s^\infty$  và  $t^\infty$  riêng biệt:

$$s(x) = \frac{r_s(x)}{f_s(x)}, \quad t(x) = \frac{r_t(x)}{f_t(x)}.$$

Khi đó đa thức cực tiểu của **dãy tổng của hai dãy** được xác định như:

$$f_{s+t} = \frac{f_s f_t}{\gcd(f_s f_t, r_s f_t + r_t f_s)}.$$

## 2.6.2. Phân phối mẫu của dòng khóa

Cho  $s^\infty$  là một dãy với chu kỳ  $N$  trên  $GF(q)$ , ở đây  $N$  không nhất thiết là chu kỳ nhỏ nhất. Vector  $(s_t, s_{t+\tau_1}, \dots, s_{t+\tau_{k-1}})$  được gọi là một *mẫu của độ dài  $k$*  (pattern of length  $k$ ) với các khoảng  $(\tau_1, \tau_2 - \tau_1, \dots, \tau_{k-1} - \tau_{k-2})$ . Một mẫu  $(s_t, s_{t+\tau})$  còn được gọi là một  *$\tau$ -bigram*. [4]

**Phân phối mẫu** (pattern distribution) là một yếu tố mật mã quan trọng của dòng khóa. Để thấy được tầm quan trọng của một phân phối mẫu nào đó, ta có luật bảo toàn của mẫu như bên dưới [4].

Bây giờ xem xét các mẫu với chiều dài  $k$  và các khoảng  $(\tau_1, \tau_2 - \tau_1, \dots, \tau_{k-1} - \tau_{k-2})$ . Cho một dãy với chu kỳ nhỏ nhất  $N$  trên  $GF(q)$ , biến vector  $(s_t, s_{t+\tau_1}, \dots, s_{t+\tau_{k-1}})$  nhận giá trị trên không gian  $GF(q)^k$  khi  $t$  chạy từ 0 đến  $N-1$ . Cho  $n((s_t, s_{t+\tau_1}, \dots, s_{t+\tau_{k-1}}) = a)$

là ký hiệu số lần mà biến vector  $(s_t, s_{t+\tau_1}, \dots, s_{t+\tau_{k-1}})$  nhận giá trị  $a \in GF(q)^k$  khi  $t$  chạy từ 0 đến  $N-1$ . [4]

**Định lý 2.6.7** (*Luật bảo toàn của mẫu*) [4]: Cho các ký hiệu như trên, ta luôn có:

$$\sum_{a \in GF(q)^k} n((s_t, s_{t+\tau_1}, \dots, s_{t+\tau_{k-1}}) = a) = N.$$

Rõ ràng định lý này có nghĩa là  $n((s_t, s_{t+\tau_1}, \dots, s_{t+\tau_{k-1}}) = a)$  được bảo toàn. Hằng số  $\frac{n((s_t, s_{t+\tau_1}, \dots, s_{t+\tau_{k-1}}) = a)}{N}$  được ký hiệu là  $\Pr(a)$ , là xác suất để  $(s_t, s_{t+\tau_1}, \dots, s_{t+\tau_{k-1}})$  nhận giá trị  $a$ . Kéo theo:

$$\sum_{a \in GF(q)^k} \Pr(a) = 1.$$

Nhìn chung, các **mẫu xấu** (bad pattern) thường có xác suất nhỏ trong dòng khóa. Nếu trong một dãy với chu kỳ  $N$  trên  $GF(q)$ , phần lớn phân phối mẫu có thể có với độ dài  $k$  và các khoảng  $(\tau_1, \tau_2 - \tau_1, \dots, \tau_{k-1} - \tau_{k-2})$  được yêu cầu sao cho  $n((s_t, s_{t+\tau_1}, \dots, s_{t+\tau_{k-1}}) = a)$  xấp xỉ bằng một hằng số, đó là  $\frac{N}{q^k}$ . Sự cần thiết của *phân phối mẫu* có liên quan đến *tấn công sai phân* (thăm mã sai phân) trên *generator dãy tự nhiên*. Sau khi thực hiện tấn công, những mẫu xấu cho nhiều thông tin về khóa hơn là các mẫu khác (tốt hơn) [4].

### 2.6.3. Hàm tương quan

Đầu tiên ta xem xét định nghĩa đặc trưng của nhóm (character of group) để nắm bắt được nội dung của các hàm tương quan.

**Định nghĩa 2.6.8** (*Đặc trưng của một nhóm*) [19]: Cho  $G$  là nhóm bất kỳ. Một đặc trưng (character) của  $G$  là một đồng cấu nhóm:

$$\chi : G \rightarrow C^*.$$

Thỏa mãn các sự kiện đúng sau:

(1) Tập hợp  $\hat{G}$  của các đặc trưng của  $G$  là một nhóm với phép nhân từng phần:

$$(\chi_1 \cdot \chi_2)(x) = \chi_1(x)\chi_2(x),$$

phép nghịch đảo được định nghĩa như  $(\chi^{-1})(x) = \chi(x)^{-1}$ , và đơn vị đồng cấu thường  $x \rightarrow 1$ .

(2) Nếu  $G$  là một nhóm hữu hạn với bậc  $n$ , tất cả đặc trưng nhận giá trị trong tập:

$$\mu_n = \{z \in C \mid z^n = 1\},$$

trong đó  $C$  là tập số phức, và nghịch đảo  $\chi^{-1}$  của một đặc trưng còn là liên hợp phức  $\bar{\chi}$ .

Một khái niệm nữa là **đặc trưng bổ sung** của trường hữu hạn (Additive characters of finite field) [19]: nếu  $GF(q)$  là một trường hữu hạn với  $q = p^\nu$  ( $p$  nguyên tố) phần tử, ta có một sự đẳng cấu của các nhóm  $GF(q) \cong GF(p^\nu) \cong (Z/pZ)^\nu$ . Các đặc trưng của  $(Z/pZ)^\nu$  được gọi là các đặc trưng bổ sung của  $GF(q)$ , ký hiệu là  $\hat{GF}(q)$  hay  $\hat{F}_q$ . Ở đây trường  $Z/pZ$  còn có cách ký hiệu thông dụng khác mà ta quen thuộc đó là  $Z_p$  [21].

**Định lý 2.6.9** [20]: Cho  $b \in GF(q)$ , hàm  $\chi_b(c) = e^{\frac{2\pi i \text{Tr}(bc)}{p}}$  đối với tất cả  $c \in GF(q)$  là một đặc trưng bổ sung của  $GF(q)$ , và mỗi đặc trưng bổ sung của  $GF(q)$  được thu theo cách này.

Trong đó  $\text{Tr}$  là ánh xạ được định nghĩa như sau [19]:

Tr :

$$GF(q) \rightarrow GF(p)$$

$$x \mapsto x + x^p + x^{p^2} + \cdots + x^{\frac{q}{p}}$$

Đặc biệt  $\text{Tr}(x)^p = \text{Tr}(x)$ .

Cho  $GF(q)$  là một trường hữu hạn. Cho  $\chi$  là một đặc trưng bổ sung của  $GF(q)$ .  $s^\infty$  và  $t^\infty$  là hai dãy với chu kỳ tương ứng là  $N$  và  $M$ , và  $P = \text{lcm}\{M, N\}$  (bội chung nhỏ nhất).

**Hàm hồ tương tuần hoàn** (periodic crosscorrelation function) của hai dãy được định nghĩa bởi [4]:

$$CC_{s,t}(l) = \sum_{i=0}^{P-1} \chi(s_i - t_{i+l}) = \sum_{i=0}^{P-1} \chi(s_i) \overline{\chi(t_{i+l})}. \quad (2.6.3.1)$$

Nếu hai dãy đồng nhất, thì  $P = M = N$  và **hàm hồ tương** được gọi là **hàm tự tương quan tuần hoàn** (periodic autocorrelation function) của  $s^\infty$ , được định nghĩa bởi:

$$AC_s(l) = \sum_{i=0}^{N-1} \chi(s_i - s_{i+l}) = \sum_{i=0}^{N-1} \chi(s_i) \overline{\chi(s_{i+l})}. \quad (2.6.3.2)$$

Nếu  $q = 2$ , thì  $\chi(a) = (-1)^a$  là một đặc trưng bổ sung của  $GF(2)$ , ở đây ta đồng nhất  $GF(2)$  với  $\mathbb{Z}_2$ . Lúc đó (2.6.3.1) và (2.6.3.2) tương ứng là các **hàm hồ tương** và **tự tương quan** thông thường của **dãy nhị phân**.

Cho  $s^\infty$  và  $t^\infty$  là hai dãy với chu kỳ tương ứng là  $N$  và  $M$ , và  $P = \text{lcm}\{M, N\}$ , **hàm hồ tương phi tuần hoàn** (aperiodic crosscorrelation function) của hai dãy được định nghĩa bởi:

$$ACC_{s,t}(l, u, v) = \sum_{i=u}^v \chi(s_i - t_{i+l}) = \sum_{i=u}^v \chi(s_i) \overline{\chi(t_{i+l})}.$$

Nếu hai dãy đồng nhất, thì  $P = M = N$  và *hàm hỗ tương* được gọi là ***hàm tự tương quan phi tuần hoàn*** (aperiodic autocorrelation function) của  $s^\infty$ , được định nghĩa bởi:

$$ACC_s(l, u, v) = \sum_{i=u}^v \chi(s_i - s_{i+l}) = \sum_{i=u}^v \chi(s_i) \overline{\chi(s_{i+l})}.$$

Các kết quả *tự tương quan phi tuần hoàn* có lẽ có tính mật mã quan trọng hơn so với các kết quả *tự tương quan tuần hoàn*. Nói chung tự tương quan tuần hoàn thì có nhiều liên hệ để dễ dàng kiểm soát hơn so với tự tương quan phi tuần hoàn [4].

#### 2.6.4. Độ phức tạp cầu

Cho  $x$  là một dãy hữu hạn với chiều dài  $n$  trên  $GF(q)$ . ***Độ phức tạp trọng số*** (weight complexity) của dãy hữu hạn được định nghĩa bởi [4]:

$$WC_u(x) = \min_{WH(y)=u} L(x+y),$$

ở đây  $WH(y)$  là ký hiệu ***trọng số Hamming*** của  $y$  ( $y$  cũng trên  $GF(q)$ ), nghĩa là *số các phần tử của  $y$  khác “không”(zero)* [22] (ta xem  $x$  và  $y$  đóng vai trò như hai vector thuộc không gian  $GF(q)^n$ ).

Bây giờ xét không gian  $GF(q)^n$  với ***khoảng cách Hamming***  $d_H$ . Khoảng cách Hamming giữa hai vector là số vị trí (tọa độ) mà hai phần tử tương ứng trên hai vector khác nhau [22], ***vector*** ở đây là một khái niệm tổng quát còn trong trường hợp cụ thể có thể là ***dãy bit*** khi xét trên không gian  $GF(2)^n$ . **Ví dụ:** Xét 2 dãy bit cùng chiều dài là  $a = \mathbf{01111010}$ ,  $b = \mathbf{10111011}$ , ta có  $d_H(a, b) = 3$ .

Ký hiệu  $S(x, u) = \{y : d_H(x, y) = u\}$ , theo định nghĩa ta có [4]:

$$WC_u(x) = \min_{y \in S(x, u)} L(y).$$

Điều này nói lên rằng *độ phức tạp trọng số* là *biên nhỏ nhất* của *độ phức tạp tuyến tính* của tất cả các dãy với chiều dài  $n$  trên *mặt cầu*  $S(x, u)$ .

Cho  $O(x, u) = \{y : 0 < d_H(x, y) \leq u\}$  là hình cầu với tâm  $x$ . **Độ phức tạp cầu** (sphere complexity) được định nghĩa bởi:

$$SC_u(x) = \min_{y \in O(x, u)} L(y) = \min_{0 < v \leq u} WC_v(x).$$

Tương tự, cho  $s^\infty$  là một dãy với chu kỳ  $N$  (không nhất thiết là chu kỳ nhỏ nhất) trên  $GF(q)$ . *Độ phức tạp trọng số* và *độ phức tạp cầu* của dãy được xác định tương ứng như:

$$WC_u(s^\infty) = \min_{WH(t^N)=u} L(s^\infty + t^\infty),$$

$$SC_u(s^\infty) = \min_{0 < v \leq u} WC_v(s^\infty)$$

trong đó  $t^\infty$  cũng có chu kỳ là  $N$ .

Cơ sở mật mã của các độ phức tạp này là: một số dòng khóa (keystream) với độ phức tạp tuyến tính lớn có thể được tính xấp xỉ bởi một số dãy với độ phức tạp tuyến tính thấp hơn. **Độ phức tạp cầu và độ phức tạp trọng số được dựa trên mô hình xấp xỉ LFSR**. Trái ngược với *độ phức tạp tuyến tính* được dựa trên LFSR ngắn nhất sinh ra một dãy, độ phức tạp cầu  $SC_k(s^\infty)$  thì được dựa trên LFSR ngắn nhất sinh ra *một dãy khác* với một *xác suất giống nhau* (phù hợp) không nhỏ hơn  $1 - \frac{k}{N}$ , ở đây  $N$  là chu kỳ của dãy  $s^\infty$  ứng với độ phức tạp cầu. Độ phức tạp trọng số  $WC_k(s^\infty)$  được dựa trên LFSR ngắn nhất sinh ra *một dãy khác* với *xác suất giống nhau* đúng bằng  $1 - \frac{k}{N}$ .

Ta sẽ thấy tầm quan trọng của hai độ phức tạp dựa trên mô hình xấp xỉ LFSR này trong trường hợp có một thuật toán hiệu quả để tìm LFSR cho việc xấp xỉ generator gốc như dưới đây.

Ta xem xét một cách tấn công như sau: giả sử rằng dòng khóa ở dạng nhị phân và độ phức tạp tuyến tính của dòng khóa của đối phương rất kém bền vững, *nhà phá mã* (cryptanalyst) ***có thể cố gắng xây dựng một LFSR để xấp xỉ generator dòng khóa gốc***, theo các bước sau [4]:

### **Bước 1:** Khởi đầu

- Dùng thuật toán Berlekamp-Massey để xây dựng một LFSR sinh ra dãy 
$$z^n = z_0 z_1 \cdots z_{n-1}.$$
- Sau đó dùng LFSR đã được xây dựng để giải mã một mẫu lớn bản mã.
- Nếu chỉ có  $\varepsilon$  phần trăm (hằng số này có thể linh hoạt, chẳng hạn như nhỏ hơn 15) bản mã được giải mã *không thể cảm nhận được*, thì chấp nhận LFSR này và dùng. Ngược lại đi đến Bước 2.

### **Bước 2:** Chạy vòng lặp

- Cho  $i = 0$  tới  $n - 1$ , thực hiện:

$$+ z_i = z_i \oplus 1$$

+ Áp dụng thuật toán Berlekamp-Massey đối với dãy mới (sau khi thực hiện phép  $\oplus$  ở trên) để xây dựng một LFSR sinh ra dãy mới.

+ Sau đó dùng LFSR đã được xây dựng để giải mã một mẫu lớn bản mã.

+ Nếu chỉ có  $\varepsilon$  phần trăm (hằng số này có thể linh hoạt, chẳng hạn như nhỏ hơn 15) bản mã được giải mã *không thể cảm nhận được*, thì chấp



nhận LFSR này và dừng. Ngược lại, lặp lại bước này đối với  $i = i + 1$  nếu  $i < n - 1$ , và đi đến Bước 3 nếu  $i = n - 1$ .

### Bước 3: Chạy vòng lặp

- Cho cặp  $(i, j)$  với  $i < j$  và  $i, j \in \{0, 1, \dots, n - 1\}$ , thực hiện:

+  $z_i = z_i \oplus 1$  và  $z_j = z_j \oplus 1$ .

+ Áp dụng thuật toán Berlekamp-Massey đối với dãy mới (sau khi thực hiện phép các  $\oplus$  ở trên) để xây dựng một LFSR sinh ra dãy mới.

+ Sau đó dùng LFSR đã được xây dựng để giải mã một mẫu lớn bản mã.

+ Nếu chỉ có  $\varepsilon$  phần trăm (hằng số này có thể linh hoạt, chẳng hạn như nhỏ hơn 15) bản mã được giải mã *không thể cảm nhận được*, thì chấp nhận LFSR này và dừng. Ngược lại, lặp lại bước này đối với cặp tiếp theo  $(i, j)$  với  $i < j$  nếu đây là một cặp trong số còn lại, và xuất “thất bại” (fail) rồi dừng nếu đây không phải là một cặp còn lại (chạy hết vòng lặp).

Bởi vì độ phức tạp của thuật toán Berlekamp-Massey đối với các dãy có chiều dài  $n$  là  $O(n^2)$ , nên *độ phức tạp* của tấn công trên là  $O(n^4)$ . Vì vậy nếu  $s^\infty$  là một keystream sao cho *độ phức tạp tuyến tính* của nó là rất lớn (chẳng hạn ví dụ như  $2^{40}$ ) và  $SC_k(s^\infty)$  đủ nhỏ (chẳng hạn ví dụ như nhỏ hơn 1000) với  $k$  rất nhỏ, thì tấn công này có thể thành công. Ý tưởng cơ bản của tấn công này là, ta mong đợi rằng dãy keystream có thể được biểu diễn như:

$$z^\infty = u^\infty + v^\infty$$

sao cho  $u^\infty$  và  $v^\infty$  có chu kỳ là  $N$ , tỉ số  $\frac{WH(v^N)}{N}$  rất nhỏ và dãy  $u^\infty$  có *độ phức tạp tuyến tính* nhỏ. Điều này có thể được thực hiện khi *độ phức tạp tuyến tính* của keystream rất kém bền vững. Trong trường hợp này, ta mong đợi rằng keystream được biết  $z^n$  có thể được biểu diễn như:

$$z^n = u^n + v^n$$

với  $WH(v^n) \leq 2$  nếu  $n < \frac{2N}{k}$ .

Do đó dẫn đến vấn đề là *nhà thiết kế* của một *mã dòng đồng bộ cộng* phải đảm bảo rằng  $k$  rất nhỏ, *độ phức tạp cầu*  $SC_k(s^\infty)$  *đủ lớn*. Nói cách khác, nhà thiết kế của một mã dòng đồng bộ cộng sẽ đảm bảo rằng *dòng khóa* được sinh ra không thể bị xấp xỉ bởi một dãy với độ phức tạp tuyến tính nhỏ, bởi vì *thuật toán thời gian đa thức trên* có thể được dùng để tìm một LFSR để xấp xỉ dãy keystream gốc nếu độ phức tạp tuyến tính của nó rất kém bền vững. **Điều này nói lên rằng độ phức tạp cầu là yếu tố mật mã quan trọng của dòng khóa.**

## 2.7. Tính an toàn của mô hình mã dòng

Những phần trên cũng đã có lúc đề cập đến vấn đề an toàn của mô hình mã dòng. Trong mật mã nói chung và mã dòng nói riêng, vấn đề an toàn luôn gắn liền với vấn đề *tấn công*, nên khi bàn về tính an toàn ta cũng sẽ đề cập đến các yếu tố tấn công này. Rõ ràng nếu một mô hình mã dòng vô hiệu được các tấn công thì mô hình đó thực sự mạnh. Tuy nhiên do giới hạn của một luận văn đại học, nên luận văn này sẽ không đi quá sâu vào các tấn công cũng như các phương pháp thám mã, có nhắc thì cũng chỉ đề cập về sự hiện diện của các tấn công trong các mô hình mã dòng cụ thể. Như **Phần 2.3. Một số kiến trúc mã dòng** có đề cập, sự an toàn của một mô hình mã dòng phụ thuộc vào *kiến trúc mã dòng* được dùng. **Các khía cạnh mật mã** của dãy hay dòng khóa

được sinh ra bởi generator cũng ảnh hưởng đến tính an toàn của mô hình mã dòng, bởi vì có các tấn công xuất phát từ đây và phụ thuộc vào các khía cạnh mật mã này. Bản thân **kiến trúc của generator** cũng đóng vai trò rất lớn trong việc đảm bảo tính an toàn của mô hình mã dòng tương ứng, ví dụ **tính phi tuyến** hay việc sử dụng **hàm Boolean** trong kiến trúc của generator. Trong kiến trúc của generator không chỉ có các hàm Boolean mà còn có thể có các thành phần phức tạp hơn như S-box, một thành phần cũng hay gặp thấy trong kiến trúc của các hệ mã khối. Các đặc tính mật mã của hàm Boolean và S-box ảnh hưởng lớn đến độ an toàn của generator.

Một vấn đề nữa trong mật mã, đó là việc cân nhắc giữa tính an toàn và tốc độ cũng như độ khó cài đặt của một thuật toán hay phương pháp mật mã. Mã dòng cũng không ngoại lệ. Một số hệ thống mã dòng dễ cài đặt nhưng không an toàn, một số thì khó cài đặt nhưng tính an toàn của chúng lại cao, một số khác có thể có cả sự cài đặt dễ dàng và tính an toàn lý tưởng nhưng lại *chậm*. Điều này làm cho nhà thiết kế mô hình mã dòng phải cân nhắc tùy theo yêu cầu đòi hỏi, hoặc nên thiết kế một cách có sự cân bằng giữa các yêu cầu về độ khó cài đặt, tính an toàn và tốc độ.

Phần này cũng chỉ đi sâu nghiên cứu tính an toàn của các mô hình mã dòng dùng generator dựa trên thanh ghi dịch chuyển, do tính phổ dụng của nó trong thực tế mã dòng.

### 2.7.1. Tính an toàn dựa trên kiến trúc mã dòng

Khi mô hình mã dòng áp dụng kiến trúc **mã dòng đồng bộ cộng**, đây là một loại mô hình hay được dùng trong thực tế. Công việc chính của mô hình là sự làm việc của generator để sinh ra dòng khóa, việc còn lại chỉ đơn giản là phép XOR. Nên tính an toàn của mô hình dựa vào kiến trúc này phụ thuộc vào tính an toàn của generator, điều này sẽ được đề cập ở **Phần 2.7.3** bên dưới.

Khi mô hình mã dòng áp dụng kiến trúc ***mã dòng tự đồng bộ cộng***, thì tính an toàn được tăng hơn so với mã dòng đồng bộ cộng do dòng khóa sinh ra phụ thuộc vào bản rõ. Tuy nhiên, mô hình loại này khó phân tích và thiết kế do liên quan đến sự phản hồi của các ký tự bản mã tới generator.

Khi mô hình mã dòng áp dụng kiến trúc ***mã dòng đồng bộ không cộng***, nếu được thiết kế tốt, dường như những tấn công nhằm vào mã dòng cộng và mã khối không áp dụng được cho mô hình này. Ngoài ra nếu áp dụng generator và thuật toán mã khối nhanh, sẽ cải thiện rất nhiều tốc độ của loại mô hình này.

Còn đối với mô hình mã dòng áp dụng kiến trúc ***phân phối hợp tác*** (CD), tính an toàn phụ thuộc vào việc chọn các thành phần mã khối và generator điều khiển. Nếu các thành phần mã khối và generator được chọn càng an toàn thì mô hình mã dòng càng được an toàn. Thậm chí nếu kiến trúc của mô hình này được thiết kế tốt, ta vẫn có thể sử dụng được các mã khối yếu cho mô hình, mà vẫn chống được các tấn công lên các thành phần mã khối và generator.

➤ **Tóm lại** các kiến trúc mã dòng khác nhau thì có độ an toàn khác nhau tùy thuộc vào chất lượng của thiết kế. Những kiến trúc khác kiến trúc mã dòng cộng có vẻ an toàn hơn, tuy nhiên lại phức tạp hơn so với nó. Do tính đơn giản của mã dòng cộng nên trong thực tế các mô hình mã dòng thường áp dụng kiến trúc này. Độ an toàn lúc đó phụ thuộc vào generator, với kiến trúc của nó và dòng khóa mà nó sinh ra. Chính vì vậy mà sự phát triển trọng tâm của mã dòng là chủ yếu nghiên cứu về generator.

### **2.7.2. Tính an toàn dựa trên các khía cạnh mật mã của dòng khóa**

Các dòng khóa là sản phẩm được sinh ra bởi các generator, các đặc tính (khía cạnh) của nó có thể nói lên nhiều điều về tính an toàn của generator hay cả mô hình mã dòng tương ứng, bởi vì có các tấn công dựa vào chính các dòng khóa này, diễn hình như tấn

công dựa vào thuật toán Berlekamp-Massey để truy ra được kiến trúc của generator (LFSR) và cả dòng khóa khi biết một phần của dòng khóa với chiều dài ít nhất  $2L$  ( $L$  là độ phức tạp tuyến tính). Nên **độ phức tạp tuyến tính** lớn không phải là yêu cầu đủ để cho một *mã dòng cộng* được an toàn, nó chỉ ở mức cần thiết. Trong khi đó đối với *mã dòng không cộng* thì không cần quan tâm tới độ phức tạp tuyến tính là lớn hay nhỏ, vì nó không cần thiết (xem **Phần 2.6.1**).

Đối với đặc tính **phân phối mẫu**, các dòng khóa có phân phối mẫu xấu tạo điều kiện cho các tấn công sai phân, điều này có khả năng để lộ thông tin về khóa rất nhiều sau tấn công (xem **Phần 2.6.2**). Trong mô hình mã dòng dựa trên *kiến trúc phân phối hợp tác*, SG phải được thiết kế sao cho dòng khóa được sinh ra có các phân phối mẫu tốt để sự *hợp tác* trong kiến trúc này được phát huy "*sức mạnh*" của nó (xem **Phần 2.3.5**).

Còn với **độ phức tạp cầu**, nếu các dòng khóa có một độ phức tạp tuyến tính rất lớn trong khi độ phức tạp cầu đủ nhỏ, thì generator (LFSR) của mô hình mã dòng chắc chắn sẽ bị tấn công. Các nhà thiết kế phải xây dựng được generator sao cho dòng khóa sinh ra có một độ phức tạp cầu đủ lớn. Như ta đã nói ở trên, độ phức tạp tuyến tính lớn chưa đủ để generator được an toàn, thì lúc này dường như có thể hiểu rằng với chính độ phức tạp cầu lớn sẽ bổ sung và làm cho generator được an toàn hơn. Do đó độ phức tạp cầu rất quan trọng (xem **Phần 2.6.4**).

Một điều cần lưu ý, xét trường hợp các mô hình mã dòng không được công bố cụ thể kiến trúc của nó, như không cho biết **độ phức tạp tuyến tính** và *đa thức kết nối* của LFSR. Điều này có vẻ không phải là tiêu chí thiết kế hay được dùng trong thực tế, bởi vì trong thực tế cấu trúc của các phương pháp mã phải được công bố, độ an toàn của phương pháp mã chỉ phụ thuộc vào khóa bí mật và phương pháp mã đó an toàn khi khóa bí mật đó khó bị tấn công để tìm ra được. Giả sử khi công bố kiến trúc generator của một mô hình mã dòng dùng generator LFSR, nếu kẻ tấn công biết trước một phần bản rõ, anh ta có thể dễ dàng tấn công thành công và tìm ra được khóa cũng như cả

dòng khóa được sinh ra bởi generator bằng thuật toán Berlekamp-Massey. Nên có thể nói ***LFSR là một generator rất yếu***. Do đó ta cần phải có những generator được thiết kế với những kiến trúc "*phức tạp*" hơn để đảm bảo tính an toàn của nó, điều này sẽ được đề cập chi tiết trong phần ngay dưới đây.

### 2.7.3. Tính an toàn dựa trên kiến trúc của generator

Có nhiều cách để khắc phục được điểm yếu của LFSR trong kiến trúc generator của mô hình mã dòng, đó là các cách: thay thế LFSR bằng ***NLFSR*** với *hàm hồi tiếp* (feedback function) ở dạng phi tuyến, dùng đến kiến trúc generator phép cộng hay tổng quát hơn là ***generator kết hợp phi tuyến***, kiến trúc ***generator lọc*** với việc sử dụng các đặc tính liên quan đến *hàm Boolean* (Boolean function) trong nó.

Về hàm Boolean trong thiết kế kiến trúc của generator, luận văn cũng đề cập đến các thuộc tính của hàm Boolean liên quan đến tính an toàn của generator như: ***độ phi tuyến***, tiêu chuẩn ***SAC*** (Strict Avalanche Criterion). ***Độ phi tuyến của hàm Boolean*** là khái niệm cơ sở để xây dựng khái niệm tổng quát hơn là ***độ phi tuyến của S-box***, một thành phần cấu tạo quan trọng trong generator ZUC.

Tiêu chuẩn SAC cũng là yếu tố để đánh giá độ an toàn của S-box được dùng trong kiến trúc của generator. Ngoài ra luận văn còn đề cập đến ***tính đồng nhất sai phân*** (differential uniformity) của S-box.

#### 2.7.3.1. Khi dùng ***NLFSR***

Dòng khóa được sinh ra có thể có độ phức tạp tuyến tính rất lớn [4]. Điều này đã cho thấy nó tiến bộ hơn so với LFSR. Tuy nhiên như đã từng đề cập trong phần nói về *tính an toàn dựa trên các khía cạnh mật mã của dòng khóa*, độ phức tạp tuyến tính lớn chỉ là một điều kiện cần chứ không đủ để đảm bảo cho generator an toàn.

### 2.7.3.2. Khi dùng generator kết hợp phi tuyến, generator lọc phi tuyến

Dùng đến **generator kết hợp phi tuyến** là một cách để khắc phục rất nhiều điểm yếu của LFSR. Tuy nhiên loại generator này luôn tiềm ẩn khả năng bị các *tấn công tương quan* (correlation attack). Nên hàm kết hợp  $f$  sẽ được lựa chọn cẩn thận sao cho không có sự phụ thuộc xác suất nào giữa **bất kỳ tập con nhỏ nào của  $n$  dãy LFSR và dòng khóa**. Điều kiện này có thể được thỏa mãn nếu  $f$  được chọn là *miễn tương quan bậc  $m$*  ( $m^{\text{th}}$ -order correlation immune). (hàm phi tuyến  $f$  như đã giới thiệu trong Phần 2.4.4.2.4).

**Định nghĩa 2.7.1** (miễn tương quan bậc  $m$ ) [18]: Xét dãy trên trường  $GF(2)$ . Cho  $X_1, X_2, \dots, X_n$  là các biến nhị phân độc lập, mỗi biến nhận giá trị 1 hay 0 đều với xác suất  $\frac{1}{2}$ . Một hàm luận lý (Boolean)  $f(x_1, x_2, \dots, x_n)$  là **miễn tương quan bậc  $m$**  nếu với mỗi tập con của  $m$  biến ngẫu nhiên  $X_{i_1}, X_{i_2}, \dots, X_{i_m}$  với  $1 \leq i_1 < i_2 < \dots < i_m \leq n$ , biến ngẫu nhiên  $Z = f(X_1, X_2, \dots, X_n)$  là độc lập xác suất của vector ngẫu nhiên  $(X_{i_1}, X_{i_2}, \dots, X_{i_m})$ , tương đương với  $I(Z; X_{i_1}, X_{i_2}, \dots, X_{i_m}) = 0$ .

ở đây  $I$  là lượng tin (mutual information) (xem thêm **Phụ lục**).

**Ví dụ**, hàm  $f(x_1, x_2, \dots, x_n) = x_1 \oplus x_2 \oplus \dots \oplus x_n$  là miễn tương quan bậc  $(n - 1)$ . Ta sẽ dễ thấy rằng ví dụ này thỏa mãn định lý sau.

**Định lý 2.7.2** [18]: Nếu một hàm Boolean  $f(x_1, x_2, \dots, x_n)$  là miễn tương quan bậc  $m$ , ở đây  $1 \leq m < n$ , thì **bậc phi tuyến** của  $f$  có giá trị **lớn nhất** là  $n - m$ . Hơn nữa, nếu  $f$  được cân bằng (balanced), nghĩa là có đúng **một nửa** các giá trị sinh ra của  $f$  là 0, thì bậc phi tuyến của  $f$  có giá trị lớn nhất là  $n - m - 1$  với  $1 \leq m \leq n - 2$ .

Đối với **generator phép cộng**, mặc dù dòng khóa được sinh ra có chu kỳ, độ phức tạp tuyến tính và miền tương quan lớn, nhưng nó có thể bị tổn thương do các *tấn công tương quan* chắc chắn và các *tấn công biết trước bản rõ dựa trên miền 2-adic* [18].

Tương tự đối với việc dùng đến **generator lọc phi tuyến**. Một ví dụ của loại generator này là **knapsack generator** với hàm lọc luận lý phi tuyến là  $f(x) = \sum_{i=1}^L x_i a_i \bmod Q$ . Ở đây  $a_1, a_2, \dots, a_L$  là khóa bí mật,  $L$  là chiều dài của LFSR,  $x = [x_L, \dots, x_2, x_1]$  là một trạng thái của thanh ghi,  $Q = 2^L$ . Rõ ràng  $f(x)$  là phi tuyến vì, nếu  $x$  và  $y$  là hai trạng thái thì trong trường hợp tổng quát  $f(x \oplus y) \neq f(x) + f(y)$ .

### 2.7.3.3. Độ phi tuyến và tiêu chuẩn SAC của hàm Boolean

#### A. Hàm Boolean và độ phi tuyến của hàm Boolean:

Như trên đã có đề cập đến hàm Boolean. Các tiêu chuẩn cho một hàm Boolean mạnh về mặt mã như là: tính *được cân bằng* (balancedness), *độ phi tuyến* (nonlinearity), *tiêu chuẩn lan truyền* (propagation criterion) hay SAC (Strict Avalanche Criterion) [32]. Sau đây là các vấn đề liên quan đến hàm Boolean.

**Định nghĩa 2.7.3 (Hàm Boolean)** [32]: Hàm Boolean là hàm  $f$  ánh xạ  $GF(2)^n$  thành  $GF(2)$ . Còn gọi đơn giản  $f$  là hàm trên  $GF(2)^n$ .

Rõ ràng tổng quát hơn hàm Boolean là hàm ánh xạ  $GF(p)^n$  thành  $GF(p)$ .

Cho  $f$  là hàm trên  $GF(2)^n$ . Dãy định nghĩa bởi  $((-1)^{f(\alpha_0)}, (-1)^{f(\alpha_1)}, \dots, (-1)^{f(\alpha_{2^n-1})})$  được gọi là **sequence** của  $f$ . Dãy nhị phân định nghĩa bởi  $(f(\alpha_0), f(\alpha_1), \dots, f(\alpha_{2^n-1}))$  được gọi là **bảng chân trị** (truth table) của  $f$ . Ở đây  $\alpha_i$  với  $0 \leq i \leq 2^n - 1$  là các vector



trong  $GF(2)^n$  mà số nguyên do nó biểu diễn là  $i$  (xem vector như chuỗi bits). Lưu ý rằng bảng chân trị có *tính thứ tự*.

**Ví dụ** về bảng chân trị của một hàm Boolean  $f$  với các biến đầu vào là  $x_1, x_2, x_3$ :

$x_1$	$x_2$	$x_3$	$f(x)$
0	0	0	<b>0</b>
0	0	1	<b>1</b>
0	1	0	<b>0</b>
0	1	1	<b>0</b>
1	0	0	<b>0</b>
1	0	1	<b>1</b>
1	1	0	<b>0</b>
1	1	1	<b>1</b>

Như vậy bảng chân trị của  $f$  là dãy bit: **01000101**.

**Bảng chân trị gọi là được cân bằng** (balanced) nếu nó chứa số các bit 1 bằng với số các bit 0. Hàm được cân bằng nếu bảng chân trị của nó được cân bằng. Ta thấy, chiều dài của bảng chân trị sinh ra là  $2^n$ , do đó nếu hàm được cân bằng thì số bit 1 của nó là  $2^{n-1}$  hay nói cách khác, **trọng số hamming** của bảng chân trị bằng  $2^{n-1}$ . **Generator mà dùng hàm Boolean cân bằng thì dãy tạo ra sẽ có sự ngẫu nhiên tốt hơn.**

Hàm *affine*  $f$  trên  $GF(p)^n$  là hàm có dạng  $f(x) = a_1x_1 \oplus a_2x_2 \oplus \dots \oplus a_nx_n \oplus c$ , ở đây  $a_j, c \in GF(2); j = 1, 2, \dots, n$ . Ngoài ra  $f$  được gọi là *hàm tuyến tính* nếu  $c = 0$ . Sequence của hàm affine (hoặc tuyến tính) được gọi là *affine (linear) sequence*.

Cho hai hàm  $f$  và  $g$  trên  $GF(2)^n$ , ***khoảng cách Hamming*** giữa chúng được định nghĩa như  $d_H(f, g) = d_H(\xi_f, \xi_g)$ , ở đây  $\xi_f$  và  $\xi_g$  tương ứng là các *bảng chân trị* của  $f$  và  $g$ .

**Định nghĩa 2.7.4** (Độ phi tuyến) [32]: **Độ phi tuyến** (nonlinearity) của  $f$ , ký hiệu bởi  $N_f$ , là *khoảng cách Hamming nhỏ nhất* giữa  $f$  và tất cả hàm affine trên  $GF(2)^n$ . Nghĩa là  $N_f = \min_{i=0,1,\dots,2^{n+1}-1} d_H(f, \varphi_i)$ , ở đây  $\varphi_0, \varphi_1, \dots, \varphi_{2^{n+1}-1}$  là các ký hiệu của các hàm affine trên  $GF(2)^n$ .

Nếu generator có kiến trúc áp dụng hàm Boolean cân bằng có **độ phi tuyến càng lớn** thì độ an toàn của generator đó càng cao để **chống lại các tấn công thám mã tuyến tính** (Linear Cryptanalysis).

Có nhiều cách để cải thiện độ lớn độ phi tuyến của hàm Boolean như: **kết nối, phân chia, điều chỉnh** các dãy [32].

**Một số kết quả về hàm cân bằng với độ phi tuyến lớn** như sau:

Trong [32], bằng cách áp dụng một *thủ tục lặp* (áp dụng các cách kết nối, điều chỉnh dãy) để cải thiện hơn độ phi tuyến của hàm được xây dựng, các nhà nghiên cứu đã xây dựng hàm thỏa định lý sau.

**Định lý 2.7.5** [32]: Cho bất kỳ số chẵn  $n \geq 4$ , tồn tại một hàm cân bằng  $f^*$  trên  $GF(2)^n$  mà độ phi tuyến của nó là:

$$N_{f^*} \geq \begin{cases} 2^{2^m-1} - \frac{1}{2}(2^{2^{m-1}} + 2^{2^{m-2}} + \dots + 2^{2^2} + 2 \cdot 2^2); & n = 2^m, \\ 2^{2^s(2t+1)-1} - \frac{1}{2}(2^{2^{s-1}(2t+1)} + 2^{2^{s-2}(2t+1)} + \dots + 2^{2(2t+1)} + 2^{2t+1} + 2^{t+1}), & n = 2^s(2t+1). \end{cases}$$

Độ phi tuyến của các hàm cân bằng trên  $GF(2)^4$ ,  $GF(2)^6$ ,  $GF(2)^8$ ,  $GF(2)^{10}$ ,  $GF(2)^{12}$  và  $GF(2)^{14}$  được xây dựng bằng cách này cho trong bảng sau [32]:

<b>Không gian vector</b>	$GF(2)^4$	$GF(2)^6$	$GF(2)^8$	$GF(2)^{10}$	$GF(2)^{12}$	$GF(2)^{14}$
<b>Cực đại</b>	4	26	118	494	2014	8126
<b>Bằng cách điều chỉnh</b>	4	26	116	492	2010	8120
<b>Bằng cách kết nối</b>	4	24	112	480	1984	8064

Bảng 1. Các độ phi tuyến của các hàm cân bằng.

➤ **Kết quả thực nghiệm độ phi tuyến:**

Trong phạm vi luận văn, chúng tôi cũng tiến hành thực nghiệm đo đặc độ phi tuyến của hàm Boolean. Rõ ràng có thể dựa trực tiếp vào **Định nghĩa 2.4.7** để xác định độ phi tuyến của hàm Boolean bằng một cơ chế lập trình khá phức tạp. Nhưng thực ra có một cách thuận lợi hơn, đó là sử dụng **biến đổi Walsh-Hadamard** (Walsh-Hadamard transform) dựa trên ma trận Walsh-Hadamard.

**Ma trận Walsh-Hadamard** được định nghĩa như [32]:

$$H_0 = 1, H_n = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes H_{n-1}, n = 1, 2, \dots$$

ở đây  $\otimes$  là **tích Kronecker**, được định nghĩa khi  $A \otimes B$  với  $A$  là ma trận  $m \times n$  và  $B$  là ma trận  $s \times t$  có kết quả là ma trận  $ms \times nt$ :

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & \dots & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{bmatrix} \text{ với } a_{ij} \text{ là phần tử ở dòng } i \text{ và cột } j$$

của ma trận  $A$ .

**Ta có:**

$$H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix};$$

$$H_3 = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

Quá trình này cứ tiếp diễn và có thể gọi nó là **biến đổi Walsh-Hadamard**.

Trong khi đó các bảng chân trị của tất cả các hàm tuyến tính bậc 3 như sau:

0 :	0	0	0	0	0	0	0	0
$x_0$ :	0	1	0	1	0	1	0	1
$x_1$ :	0	0	1	1	0	0	1	1
$x_1 + x_0$ :	0	1	1	0	0	1	1	0
$x_2$ :	0	0	0	0	1	1	1	1
$x_2 + x_0$ :	0	1	0	1	1	0	1	0
$x_2 + x_1$ :	0	0	1	1	1	1	0	0
$x_2 + x_1 + x_0$ :	0	1	1	0	1	0	0	1

Gọi các bảng chân trị này là **bảng các bảng chân trị tuyến tính 3 biến**.

Nếu thay các phần tử có giá trị 1 thành 0 và -1 thành 1 trong  $H_3$  thì ta thấy lúc đó  $H_3$  trùng với bảng chân trị trên. Tương tự ta có thể kiểm chứng rằng với mỗi  $H_n$  ( $n > 3$ ) luôn có một bảng các bảng chân trị  $n$  biến tương ứng với nó. Từ đây ta có ý tưởng, **dùng biến đổi Walsh-Hadamard để xác định các bảng chân trị của các hàm Boolean tuyến tính  $n$  biến khi cần tính độ phi tuyến của một hàm Boolean  $f$  nào đó có  $n$  biến**. Với một lưu ý rằng, sau biến đổi Walsh-Hadamard ta vẫn chưa xác định được các truth table của các hàm affine  $n$  biến còn lại, tuy nhiên đây là một việc hết sức dễ dàng. Bởi vì các hàm affine còn lại đó là các hàm tuyến tính tương ứng **XOR** với 1, nên chỉ cần lấy tất cả các phần tử của các bảng chân trị của các hàm tuyến tính tương ứng **XOR** với 1 là ta có được các bảng chân trị của các hàm affine còn lại. Để rồi sau đó tiếp tục dùng cho việc so sánh các **khoảng cách Hamming** trong quá trình tính độ phi tuyến của  $f$ . Với tư tưởng đó, các thủ tục cần thiết (có dùng biến đổi Walsh-Hadamard) được hiện thực (lập trình) để tính độ phi tuyến của hàm Boolean.

**Sau đây là bảng thống kê độ phi tuyến của các hàm Boolean trên  $GF(2)^8$  với bảng chân trị được chọn ngẫu nhiên:**

STT	Bảng chân trị	Độ phi tuyến
-----	---------------	--------------

1	101011101000011101101111101000101011100001011111111100 000001000011011111110000000110111001100001111100001100 010011110001011000110001001010101000000000100100000010 011100000010010010100011011011101011111111111011000010 0110111111111101010001110110000100111000	<b>100</b>
2	001000011110000001001100010100100100000111001110011011 111001011111111011001100001010101111101110101111000110 000011111111010111100001110011111111110010011001111000 010110001111110000100100100001010011011011110111000100 1101011001011111011110100010111001001001	<b>102</b>
3	100110111010111010110001011110010111010011010100000001 110100111000011001111011011111011111010001111010110001 110010010001011111100100111011110100111000011001111110 000111101110011001100010110110101111111111100101011001 0011100011111100011011010010100100000011	<b>100</b>
4	111011001101111101111011110011001110010000010011011111 011111100101111001100101101000010110000010000100011001 110111010101000111111100110001111100111110001010111111 111001110110101100010001111101000011000101011100110110 1010010101110100010111101000011010110010	<b>103</b>
5	110111010000011011100001010010011001010011001000101101 101011010000101101011011100110010011110011111100011010 101111111010101011000001010000010101100100001010001101 101000101000101001001101000001000001000101000111111100	<b>107</b>

	1001011110110111011101110011010111101100	
6	001100010101001100110010010000101101001101010000001010 011110010000000101101100100000011100110000101110110101 000101100001110011010010110110111111001000110101101111 001100000000000001001001110010011100001000110000011110 1101000111000101010101000110011110011010	<b>103</b>
7	011110000000101111001011000010110011011010010001000010 000001101100000001111011100110100100011100111000010000 001001011000101010110100001110101011000011101101110010 110011011000001110000101100001111101111010101001000010 0101010001100000011011011011111110010111	<b>105</b>

Ta nhận thấy các độ phi tuyến cho trong bảng này rõ ràng nhỏ hơn độ phi tuyến cực đại với giá trị là **118** ở các hàm Boolean trên  $GF(2)^8$  cho trong **Bảng 1**.

## B. Tiêu chuẩn SAC của hàm Boolean:

**Định nghĩa 2.7.6** [32]: Một hàm  $f$  trên  $GF(2)^n$  được gọi là thỏa mãn tiêu chuẩn SAC nếu  $f(x) \oplus f(x \oplus \alpha)$  là một hàm cân bằng với bất kỳ  $\alpha \in GF(2)^n$  mà trọng số Hamming của nó là 1.

Nếu xét hàm  $f$  trên  $GF(p)^n$ , ta có định nghĩa tổng quát về tiêu chuẩn SAC như sau:

**Định nghĩa 2.7.7** (tổng quát SAC) [33]: Hàm  $f(x) : GF(p)^n \rightarrow GF(p)$  thỏa tiêu chuẩn SAC khi và chỉ khi  $\Pr(f(x+a) = f(x) + a) = \frac{1}{p}, \forall a \in GF(p)^n$  thỏa  $WH(a) = 1$ .

Trong kiến trúc của generator có thể bao gồm các **S-box** (bảng thay thế), generator ZUC là một ví dụ. Tiêu chuẩn SAC cũng ảnh hưởng đến độ an toàn của S-box.

Xét  $m \times n$  S-box gồm  $m$  bit đầu vào và  $n$  bit đầu ra. *Có thể xem S-box này gồm  $n$  hàm  $f_0, f_1, \dots, f_{n-1}$ , trong đó hàm  $f_j : GF(2)^m \rightarrow GF(2)$  xác định bit thứ  $j$  trong kết quả của S-box ( $0 \leq j < n$ ) [33]. Các hàm này là các hàm Boolean và được gọi là các hàm thành phần của S-box.*

Một trong những tiêu chí đánh giá độ an toàn của S-box là từng hàm  $f_j$  phải đạt hay “gần đạt” tiêu chuẩn SAC, tức là nếu 1 bit đầu vào của S-box bị thay đổi thì mỗi bit đầu ra sẽ bị thay đổi với xác suất xấp xỉ  $\frac{1}{2}$  [33].

Chẳng hạn trong *Bảng 2* [33], phần tử tại dòng  $i$  cột  $j$  là số trường hợp giá trị của hàm  $f_j$  bị thay đổi khi bit đầu vào thứ  $i$  bị thay đổi đối với S-box trong thuật toán AES. Mặc dù tất cả các hàm  $f_j$  thì S-box của AES không thỏa tiêu chí SAC nhưng số trường hợp kết quả của  $f_j$  bị thay đổi khi bit đầu vào thứ  $i$  bị thay đổi xấp xỉ 128. Như vậy, khi 1 bit đầu vào bị thay đổi, mỗi bit đầu ra sẽ thay đổi với xác suất xấp xỉ  $\frac{1}{2}$  [33]. Do đó các hàm thành phần của S-box gần đạt tiêu chuẩn SAC.

	$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$
<b>bit 0</b>	132	132	116	144	116	124	116	128
<b>bit 1</b>	120	124	144	128	124	116	128	136
<b>bit 2</b>	132	132	128	120	144	128	136	128
<b>bit 3</b>	136	136	120	116	128	136	128	140
<b>bit 4</b>	116	128	116	132	128	128	140	136



<b>bit 5</b>	116	132	132	120	120	140	136	136
<b>bit 6</b>	136	136	120	132	120	136	136	124
<b>bit 7</b>	132	144	132	136	124	136	124	132

Bảng 2. Khảo sát sự thay đổi của các hàm nhị phân thành phần  $f_j$  khi bit đầu vào thứ  $i$  bị thay đổi đối với S-box trong AES.

#### 2.7.3.4. Tính đồng nhất sai phân của S-box

**Định nghĩa 2.7.8** (tính đồng nhất sai phân) [34]: Cho  $G_1$  và  $G_2$  là các nhóm Abel hữu hạn. Ánh xạ  $f : G_1 \rightarrow G_2$  được gọi là **đồng nhất sai phân** (differential uniformity) mức  $\delta$  nếu:

$$\forall \alpha \in G_1, \alpha \neq 0, \forall \beta \in G_2, |\{z \in G_1 \mid f(z + \alpha) - f(z) = \beta\}| \leq \delta \quad (2.7.3.4)$$

ở đây  $\delta$  được gọi là mức đồng nhất sai phân của  $f$ .

Nếu  $f$  là một  $n \times m$  S-box,  $G_1 = GF(2^n)$  và  $G_2 = GF(2^m)$ , thì biểu thức (2.7.3.4) trong định nghĩa trên có thể viết là:  $\forall \alpha \in G_1, \alpha \neq 0, \forall \beta \in G_2, |\{z \in G_1 \mid f(z + \alpha) \oplus f(z) = \beta\}| \leq \delta$ . Vì trên  $GF(2^m)$ , phép cộng ( $\oplus$ ) cũng chính là phép trừ. Để thấy điều này dựa vào định nghĩa phép cộng trên trường  $GF(2^m)$  (xem **Phần 2.5.3**), và với một lưu ý rằng trên trường  $GF(2)$  phép cộng cũng chính là phép trừ (chú ý:  $0 - 1 = -1 = 1 \bmod 2$ ).

Giá trị  $\delta$  càng nhỏ thì ánh xạ  $f$  càng an toàn đối với **tấn công mật mã sai phân** và **tấn công mật mã tuyến tính** [33].

Đối với  $n \times m$  S-box, mức đồng nhất sai phân bị chặn dưới là  $\delta_{\min} = 2^{n-m+1}$ . S-box đạt được mức đồng nhất  $\delta_{\min}$  được gọi là Almost Perfect Nonlinear (APN). Tuy nhiên, không tồn tại APN S-box có  $n$  bit đầu vào và  $n$  bit đầu ra với  $n$  chẵn. Vì vậy,

trong thuật toán AES sử dụng  $8 \times 8$  S-box, mức đồng nhất sai phân tối thiểu (lý tưởng) là  $\delta = 2^2 = 4$  [33].

# Chương 3. MÃ DÒNG TRÊN MẠNG DI ĐỘNG

## Tóm tắt chương:

✚ Chương 3 hệ thống và khảo sát các vấn đề liên quan đến ứng dụng của mã dòng trên mạng di động. Nội dung chương này trình bày các vấn đề chính sau:

- Hệ thống hóa về mạng di động, các thuật toán bảo mật đã có trên mạng di động.
- Hệ thống lại mô hình mã dòng ZUC với cấu tạo của ZUC cũng như kiến trúc và hoạt động của 3 lớp: LFSR, BR, hàm phi tuyến  $F$ ; trình bày hoạt động của ZUC.
- Trình bày hai ứng dụng của ZUC là thuật toán mã hóa 128-EEA3 và thuật toán chứng thực thông điệp 128-EIA3.
- Hệ thống, phân tích các tiêu chí thiết kế và tính an toàn của ZUC: trình bày và phân tích tiêu chí thiết kế của lớp LFSR; trình bày tiêu chí thiết kế của lớp BR; hệ thống và phân tích tiêu chí thiết kế và tính an toàn của hàm phi tuyến  $F$ , đặc biệt đi sâu phân tích và thực nghiệm đo đạc để kiểm tra các đặc tính mật mã quan trọng của hai S-box  $S_0$  và  $S_1$  là: tính phi tuyến của S-box, tính đồng nhất sai phân của S-box, tiêu chuẩn SAC và tính cân bằng (balance) của các hàm thành phần của S-box.

## 3.1. Giới thiệu về mạng di động

### 3.1.1. Các chuẩn mạng di động

**GSM** là tên viết tắt của *Hệ thống thông tin di động toàn cầu* (Global System for Mobile Communications). Đây là một chuẩn dành cho mạng thông tin di động và hiện nay được sử dụng rất phổ biến trên thế giới. GSM được công bố vào năm 1982 và được xem là chuẩn mạng *thế hệ thứ hai* (Second Generation – 2G). GSM dùng mô hình *mạng chia ô* (cellular network). Mạng này được phân thành nhiều ô (cell) với năm loại kích thước ô khác nhau. Điện thoại di động sẽ được kết nối vào mạng GSM bằng cách tìm kiếm ô gần nó nhất. Để được mạng GSM xác nhận, mỗi điện thoại phải có một *modun xác nhận người đăng ký* (Subscriber Identity Module) hay còn được gọi đơn giản là thẻ SIM. Mạng GSM hoạt động trên nhiều băng tần khác nhau. Các băng tần được sử dụng nhiều nhất là 900 MHz và 1800 MHz.

**UMTS** ra đời sau GSM và được xem là chuẩn mạng thuộc thế hệ thứ ba (Third Generation – 3G). UMTS là tên viết tắt của *Hệ thống viễn thông di động toàn cầu* (Universal Mobile Telecommunications System). So với GSM mạng UMTS có tốc độ truyền tải cao hơn do sử dụng kỹ thuật *trải phổ* (wideband). Về lý thuyết tốc độ truyền tải tối đa của mạng UMTS có thể lên đến 45Mbit/s. UMTS sử dụng cặp dải băng tần riêng cho thao tác tải lên (upload) và tải xuống (download). Cặp băng tần này thay đổi tùy vào mỗi quốc gia và chuẩn loại UMTS sử dụng.

**ESP** là chuẩn mạng được phát triển từ UMTS và hiện vẫn còn đang được nghiên cứu và xây dựng. EPS là tên của *hệ thống gói tin tiến hóa* (Evolved Packet System). Chuẩn mạng này thuộc thế hệ thứ tư (4G) và kế thừa các ưu điểm từ hai chuẩn mạng GSM và UMTS. Có hai chuẩn được quan tâm nhiều trong EPS là chuẩn **LTE** (Long Term Evolution) và chuẩn **SAE** (Service Architecture Evolution). LTE quan tâm đến sóng

truyền và *giao tiếp* (Interface) với thiết bị, trong khi SAE quan tâm đến việc xây dựng *mạng lõi* (Core Network).

### 3.1.2. Bảo mật trên mạng di động

**Mạng GSM** được thiết kế cho các ứng dụng bảo mật không quá phức tạp. Trong chứng thực, GSM sử dụng *khóa qui ước trước* (pre-shared key) và phương pháp *thách thức – trả lời* (challenge-response authentication). Cặp thuật toán chứng thực A3 và A8 sẽ được cài sẵn trong SIM để giải mã gói tin thách thức và tạo khóa bí mật để truyền dữ liệu. Khi SIM gửi yêu cầu kết nối đến mạng, hệ thống mạng sẽ tạo một số ngẫu nhiên RAND rồi gửi lại cho SIM. SIM và hệ thống mạng cùng sử dụng thuật toán A3 với đầu vào là khóa bí mật K1 (khóa bí mật của SIM này và hệ thống mạng) và số ngẫu nhiên RAND còn đầu ra là giá trị SRES dài 32-bit. Nếu hai giá trị SRES của SIM và hệ thống mạng giống nhau thì SIM được chứng thực thành công. Khi đó SIM cùng hệ thống mạng dùng thuật toán A8 để tạo ra khóa phiên nhằm mã hóa dữ liệu trao đổi [28]. GSM dùng nhóm thuật toán A5 để mã hóa gói tin. Trong nhóm thuật toán A5 này có hai thuật toán đang được áp dụng rộng rãi là A5/1 và A5/2 và một thuật toán đang được phát triển dựa trên mã hóa Kasumi là A5/3.

Để bảo mật EPS sử dụng cặp thuật toán là EEA và EIA. **EEA là thuật toán mã hóa trên EPS (EPS Encryption Algorithms)** còn **EIA là thuật toán chứng thực trên EPS (EPS Integrity Algorithms)**. Cặp thuật toán đầu tiên mà EPS sử dụng là 128-EEA1 và 128-EIA1. Các thuật toán này được xây dựng dựa trên mã dòng SNOW 3G và sử dụng khóa dài 128 bit. Sau đó cặp thuật toán 128-EEA2 và 128-EIA2 [41] được xây dựng và phát triển dựa trên mã hóa khối AES. Hiện nay hướng nghiên cứu thuật toán bảo mật trên EPS là phát triển vào mã dòng ZUC [31] làm cơ sở cho cặp thuật toán 128-EEA3 và 128-EIA3.

## 3.2. Mã dòng ZUC

### 3.2.1. Cấu tạo của ZUC

ZUC sử dụng bộ sinh phi tuyến. Bộ sinh trong ZUC bao gồm một thanh ghi LFSR và một hàm phi tuyến  $F$ . Hình dưới đây mô tả cấu trúc tổng quát của ZUC. Ta chia ZUC ra làm 3 lớp chính như sau: lớp trên cùng là **thanh ghi LFSR** có 16 stage, lớp ở giữa được gọi là lớp **tái cấu trúc dây bit** (Bit-reorganization - BR), lớp ở dưới cùng là **hàm phi tuyến  $F$**  [31]:



**Ở dạng thiết lập**, thanh ghi LFSR nhận dữ liệu đầu vào là một từ (word)  $u$  dài 31 bit. Từ này được tạo ra bằng cách lấy word  $w$  từ đầu ra của hàm phi tuyến  $F$  và bỏ đi bit thấp nhất ( $u = w \gg 1$ ). Quy trình bao gồm các bước sau:

LFSRWithInitialisationMode( $u$ ) {

1.  $v = 2^{15}s_{15} + 2^{17}s_{13} + 2^{21}s_{10} + 2^{20}s_4 + (1+2^8)s_0 \bmod (2^{31}-1);$
2.  $s_{16} = (v+u) \bmod (2^{31}-1);$
3. If  $s_{16} = 0$ , then set  $s_{16} = 2^{31}-1;$
4.  $(s_1, s_2, \dots, s_{15}, s_{16}) \rightarrow (s_0, s_1, \dots, s_{14}, s_{15});$

}

**Ở dạng hoạt động**, thanh ghi thực hiện các thao tác sau:

LFSRWithWorkMode() {

1.  $s_{16} = 2^{15}s_{15} + 2^{17}s_{13} + 2^{21}s_{10} + 2^{20}s_4 + (1+2^8)s_0 \bmod (2^{31}-1);$
2. If  $s_{16} = 0$ , then set  $s_{16} = 2^{31}-1;$
3.  $(s_1, s_2, \dots, s_{15}, s_{16}) \rightarrow (s_0, s_1, \dots, s_{14}, s_{15});$

}

### 3.2.3. Tái cấu trúc dãy bit

Trong bộ sinh phi tuyến, hàm phi tuyến  $F$  thường không sử dụng hết các stage của thanh ghi LFSR mà chỉ sử dụng một số stage chọn sẵn. Việc chọn stage được thực hiện ở lớp **tái cấu trúc dãy bit** (the bit-reorganization). Ở lớp này các stage  $s_0, s_2, s_5, s_7, s_9,$



$s_{11}, s_{14}, s_{15}$  của thanh ghi sẽ được kết hợp lại tạo thành bốn word 36-bit là  $X_0, X_1, X_2, X_3$ . Ba word đầu sẽ được sử dụng trong hàm phi tuyến  $F$ , word  $X_3$  còn lại dùng để tạo ra keystream. Hoạt động của lớp này được miêu tả bởi hàm sau:

```

Bitreorganization() {
    1.     $X_0 = s_{15H} \parallel s_{14L}$ ;
    2.     $X_1 = s_{11L} \parallel s_{9H}$ ;
    3.     $X_2 = s_{7L} \parallel s_{5H}$ ;
    4.     $X_3 = s_{2L} \parallel s_{0H}$ .
}

```

Trong hàm Bitreorganization(),  $s_{xL}$  là 16 bit cao của stage  $s_x$ ,  $s_{xH}$  là 16 bit thấp của stage  $s_x$ . Còn phép biến đổi  $a \parallel b$  là phép nối hai dãy bit  $a$  và  $b$  thành một dãy bit duy nhất trong đó dãy  $a$  nằm về phía bên trái còn dãy  $b$  nằm phía bên phải của dãy bit mới này.

### 3.2.4. Hàm phi tuyến $F$

Hàm phi tuyến  $F$  nhận đầu vào là 3 word  $X_0, X_1, X_2$  từ lớp trên. Đầu ra của hàm là 2 word  $W$ . Trong lớp này có hai biến nhớ là  $R_1$  và  $R_2$ . Hoạt động của hàm  $F$  bao gồm các bước sau:

```

 $F(X_0, X_1, X_2)$  {
    1.  $W = (X_0 \text{ XOR } R_1) + R_2 \text{ mod } 2^{32}$ ;
    2.  $W_1 = R_1 + X_1 \text{ mod } 2^{32}$ ;
    3.  $W_2 = R_2 \text{ XOR } X_2$ ;
}

```

$$4. R_1 = S(L_1(W_{1L} || W_{2H}));$$

$$5. R_2 = S(L_2(W_{2L} || W_{1H}));$$

}

Trong hàm  $F$  có sử dụng các hàm con  $S()$ ,  $L_1()$  và  $L_2()$ . Hàm  $S()$  là một 32x32 S-box nhận vào một word (32 bit) và trả về một word tương ứng.  $L_1()$  và  $L_2()$  là các hàm biến đổi tuyến tính. Cấu trúc của từng hàm được trình bày chi tiết trong phần tiếp theo.

### 3.2.4.1. S-box S

32x32 S-box trong ZUC gồm bốn 8x8 S-box ghép lại là  $S_0$ ,  $S_1$ ,  $S_2$ ,  $S_3$ . Trong đó  $S_0 = S_2$  và  $S_1 = S_3$ . Mô hình của các S-box con  $S_0$  và  $S_1$  được thể hiện qua các sơ đồ bên dưới.

Để tính  $S(X)$  với  $X$  là một word 32 bit ta phải tách  $X$  thành 4 byte khác nhau:  $X = X_0 || X_1 || X_2 || X_3$ . Gọi  $Y = S(X)$ , khi đó  $Y = S_0(X_0) || S_1(X_1) || S_2(X_2) || S_3(X_3)$ . Để tính giá trị qua S-box con, ví dụ  $S_0(X_0)$ , ta tách  $X_0$  thành hai phần mỗi phần 4 bit được thể hiện ở dạng thập lục phân, ví dụ  $X_0 = H_0 || L_0$ . Khi đó giá trị  $S_0(X_0)$  sẽ nằm tại hàng thứ  $H_0$  và cột thứ  $L_0$  trong bảng S-box  $S_0$ . Ví dụ với  $X = 0x12345678$  thì  $Y = S(X) = S_0(0x12) || S_1(0x34) || S_2(0x56) || S_3(0x78) = 0xF9C05A4E$ .

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	3E	72	5B	47	CA	E0	00	33	04	D1	54	98	09	B9	6D	CB
1	7B	1B	F9	32	AF	9D	6A	A5	B8	2D	FC	1D	08	53	03	90
2	4D	4E	84	99	E4	CE	D9	91	DD	B6	85	48	8B	29	6E	AC
3	CD	C1	F8	1E	73	43	69	C6	B5	BD	FD	39	63	20	D4	38
4	76	7D	B2	A7	CF	ED	57	C5	F3	2C	BB	14	21	06	55	9B
5	E3	EF	5E	31	4F	7F	5A	A4	0D	82	51	49	5F	BA	58	1C
6	4A	16	D5	17	A8	92	24	1F	8C	FF	D8	AE	2E	01	D3	AD
7	3B	4B	DA	46	EB	C9	DE	9A	8F	87	D7	3A	80	6F	2F	C8
8	B1	B4	37	F7	0A	22	13	28	7C	CC	3C	89	C7	C3	96	56
9	07	BF	7E	F0	0B	2B	97	52	35	41	79	61	A6	4C	10	FE
A	BC	26	95	88	8A	B0	A3	FB	C0	18	94	F2	E1	E5	E9	5D
B	D0	DC	11	66	64	5C	EC	59	42	75	12	F5	74	9C	AA	23

C	0E	86	AB	BE	2A	02	E7	67	E6	44	A2	6C	C2	93	9F	F1
D	F6	FA	36	D2	50	68	9E	62	71	15	3D	D6	40	C4	E2	0F
E	8E	83	77	6B	25	05	3F	0C	30	EA	70	B7	A1	E8	A9	65
F	8D	27	1A	DB	81	B3	A0	F4	45	7A	19	DF	EE	78	34	60

Bảng 3. S-box  $S_0$ .

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	55	C2	63	71	3B	C8	47	86	9F	3C	DA	5B	29	AA	FD	77
1	8C	C5	94	0C	A6	1A	13	00	E3	A8	16	72	40	F9	F8	42
2	44	26	68	96	81	D9	45	3E	10	76	C6	A7	8B	39	43	E1
3	3A	B5	56	2A	C0	6D	B3	05	22	66	BF	DC	0B	FA	62	48
4	DD	20	11	06	36	C9	C1	CF	F6	27	52	BB	69	F5	D4	87
5	7F	84	4C	D2	9C	57	A4	BC	4F	9A	DF	FE	D6	8D	7A	EB
6	2B	53	D8	5C	A1	14	17	FB	23	D5	7D	30	67	73	08	09
7	EE	B7	70	3F	61	B2	19	8E	4E	E5	4B	93	8F	5D	DB	A9
8	AD	F1	AE	2E	CB	0D	FC	F4	2D	46	6E	1D	97	E8	D1	E9
9	4D	37	A5	75	5E	83	9E	AB	82	9D	B9	1C	E0	CD	49	89
A	01	B6	BD	58	24	A2	5F	38	78	99	15	90	50	B8	95	E4
B	D0	91	C7	CE	ED	0F	B4	6F	A0	CC	F0	02	4A	79	C3	DE
C	A3	EF	EA	51	E6	6B	18	EC	1B	2C	80	F7	74	E7	FF	21
D	5A	6A	54	1E	41	31	92	35	C4	33	07	0A	BA	7E	0E	34
E	88	B1	98	7C	F3	3D	60	6C	7B	CA	D3	1F	32	65	04	28
F	64	BE	85	9B	2F	59	8A	D7	B0	25	AC	AF	12	03	E2	F2

Bảng 4. S-box  $S_1$ .

### 3.2.4.2. Hàm biến đổi tuyến tính

$L_1$  và  $L_2$  là hai hàm biến đổi tuyến tính. Hai hàm này ánh xạ một word 32-bit này thành một word 32-bit khác. Cú pháp cụ thể của hàm như sau:

$$L_1(X) = X \oplus (X \lll_{32} 2) \oplus (X \lll_{32} 10) \oplus (X \lll_{32} 18) \oplus (X \lll_{32} 24),$$

$$L_2(X) = X \oplus (X \lll_{32} 8) \oplus (X \lll_{32} 14) \oplus (X \lll_{32} 22) \oplus (X \lll_{32} 30).$$

### 3.2.5. Hoạt động của ZUC

Để xây dựng mã dòng từ bộ sinh phi tuyến của ZUC ta phải hoàn thành các công việc sau đây:

- Đưa thông tin của khóa và của *vector khởi tạo* (instant vector – IV) vào bộ sinh.
- Thiết lập trạng thái ban đầu cho thanh ghi LFSR.
- Thực hiện lại nhiều lần hoạt động generator cho đến khi thu được keystream có độ dài phù hợp.

Mã dòng ZUC sử dụng tiến trình nạp khóa để thiết lập thông tin ban đầu cho LFSR và tiến trình thực thi để phát sinh keystream. Tiến trình thực thi trong ZUC lại được chia làm hai bước: ***bước khởi tạo*** và ***bước hoạt động***.

### 3.2.5.1. Nạp khóa

ZUC sử dụng khóa  $k$  dài 128 bit và vector khởi tạo  $iv$  dài 128 bit. IV và khóa khi đưa vào ZUC sẽ được khai triển thành thông tin ban đầu cho thanh ghi LFSR. Tiến trình gọi là ***nạp khóa*** (loading key procedure). Trong tiến trình, một chuỗi hằng số  $D$  được sử dụng. Hằng số  $D$  dài 240 bit và được chia thành các dãy con, mỗi dãy dài 15 bit như sau:

$$D = d_0 || d_1 || \dots || d_{15},$$

$$d_0 = 100010011010111_2, \quad d_1 = 010011010111100_2,$$

$$d_2 = 110001001101011_2, \quad d_3 = 001001101011110_2,$$

$$d_4 = 101011110001001_2, \quad d_5 = 011010111100010_2,$$

$$d_6 = 111000100110101_2, \quad d_7 = 000100110101111_2,$$

$$d_8 = 100110101111000_2, \quad d_9 = 010111100010011_2,$$

$$d_{10} = 110101111000100_2, \quad d_{11} = 001101011110002_2,$$

$$d_{12} = 101111000100110_2, \quad d_{13} = 011110001001101_2,$$

$$d_{14} = 111100010011010_2, \quad d_{15} = 100011110101100_2.$$

Để tạo thông tin cho các stage của thanh ghi LFSR,  $k$  và  $iv$  lần lượt được chia nhỏ thành 16 dãy con, mỗi dãy dài 8 bit.

$$k = k_0 \parallel k_1 \parallel k_2 \parallel \dots \parallel k_{15}$$

$$iv = iv_0 \parallel iv_1 \parallel iv_2 \parallel \dots \parallel iv_{15}$$

Khi đó trạng thái của stage thứ  $i$  trong thanh ghi LFSR sẽ được khởi tạo như sau:

$$s_i = k_i \parallel d_i \parallel iv_i.$$

### 3.2.5.2. Bước khởi tạo

Trong **bước khởi tạo** (initialization stage) này, thuật toán ZUC gọi tiến trình nạp khóa (phần 3.2.5.1) để phát sinh dữ liệu ban đầu cho generator. Đồng thời các thanh ghi nhớ  $R_1, R_2$  được gán giá trị 0. Việc khởi tạo được thực thi 25 lần bằng các hàm sau:

1. Bitreorganization();
2.  $w = F(X_0, X_1, X_2)$ ;
3. LFSRWithInitialisationMode( $w \gg 1$ ).

Sau đoạn lệnh trên thanh ghi LFSR sẽ thay đổi trạng thái các thanh ghi nhớ  $R_1$  và  $R_2$  được cập nhật. Lúc bấy giờ ZUC chưa tạo ra keystream.

### 3.2.5.3. Bước hoạt động

Sau khi thực hiện xong bước khởi tạo, ZUC chuyển sang ***bước hoạt động*** (working stage). Ở bước này hàm giá trị đầu ra  $w$  của hàm phi tuyến  $F$  không được sử dụng để thay đổi trạng thái thanh ghi LFSR. Bước hoạt động bao gồm chuỗi lệnh sau:

1. Bitreorganization();
2.  $F(X_0, X_1, X_2)$ ;
3. LFSRWithWorkMode().

Chuỗi lệnh này được ZUC thực hiện một lần và không tạo ra keystream. Sau khi thực hiện xong chuỗi lệnh trên, ZUC mới tạo ra keystream bằng cách thực thi chuỗi lệnh dưới đây:

1. Bitreorganization();
2.  $Z = F(X_0, X_1, X_2) \oplus X_3$ ;
3. LFSRWithWorkMode().

Chuỗi lệnh này tạo ra một word  $Z$  dài 32 bit.  $Z$  này chính là keyword đầu ra của ZUC. Chuỗi lệnh sẽ được thực hiện lặp lại nhiều lần để tạo keystream có kích thước đáp ứng được yêu cầu mã hóa.

### 3.3. Ứng dụng của ZUC

#### 3.3.1. Mã hóa 128-EEA3

##### 3.3.1.1. Giới thiệu

Mã dòng 128-EEA3 dùng để mã hóa và giải mã dữ liệu với các dòng dữ liệu liên tục có độ dài từ 1 đến 20000 bit. Đầu vào và đầu ra của giải thuật này như sau:

	Tham số	Kích thước (bits)	Ghi chú
Đầu vào	COUNT	32	Biến đếm
	BEARER	5	Xác định sóng mang (bearer)
	DIRECTION	1	Xác định hướng chuyển dữ liệu là upload hay download
	CK	128	Khóa mật
	LENGTH	32	Chiều dài (bits) của dữ liệu đầu vào
	M	LENGTH	Chuỗi bit đầu vào (bản rõ)
Đầu ra	C	LENGTH	Chuỗi bit đầu ra (bản mã)

Trong bảng trên các biến COUNT, BEARER, DIRECTION là các tham số dùng để thiết lập generator. Bộ ba biến này kết hợp lại tạo thành biến *nonce* cho quá trình mã hóa, giải mã và được đính kèm vào bản mã.

##### 3.3.1.2. Tạo keystream

Trong phần thiết lập, 128-EEA3 sẽ tạo ra các đối số để chạy ZUC là: khóa và vector khởi tạo. Biến CK được dùng làm khóa của ZUC. Quá trình tạo vector khởi tạo được bắt đầu bằng việc tách các biến COUNT ra thành từng byte:

$$\text{COUNT} = \text{COUNT}_0 \parallel \text{COUNT}_1 \parallel \text{COUNT}_2 \parallel \text{COUNT}_3$$

Dùng biến IV 128-bit làm vector khởi tạo sử dụng cho ZUC. IV sẽ được tách nhỏ thành từng byte và được gán giá trị cho từng byte này:

$$IV = IV_0 \parallel IV_1 \parallel IV_2 \parallel \dots \parallel IV_{15}$$

$$IV_0 = COUNT_0, IV_1 = COUNT_1, IV_2 = COUNT_2,$$

$$IV_3 = COUNT_3, IV_4 = BEARER \parallel DIRECTION \parallel 00_2,$$

$$IV_5 = IV_6 = IV_7 = 00000000_2,$$

$$IV_8 = IV_0, IV_9 = IV_1, IV_{10} = IV_2, IV_{11} = IV_3,$$

$$IV_{12} = IV_4, IV_{13} = IV_5, IV_{14} = IV_6, IV_{15} = IV_7.$$

Lúc bấy giờ hai biến CK và IV sẽ được nạp để ZUC hoạt động. Sau mỗi vòng lặp, ZUC sẽ tạo ra một word 32-bit. Để tạo đủ keystream để mã hóa dữ liệu có kích thước  $LENGTH$  thì cần thực hiện số vòng lặp  $L = \lceil LENGTH/32 \rceil$ . Sau  $L$  vòng lặp ZUC sinh ra keystream  $z$  có kích thước  $LENGTH$  bit.

### 3.3.1.3. Mã hóa và giải mã

Trong quá trình mã hóa, ta XOR từng bit của thông tin cần mã hóa  $M$  với keystream  $z$  mà ZUC vừa tạo ra. Do đó **128-EEA3 dùng kiến trúc mã dòng đồng bộ cộng**. Quá trình mã hóa và giải mã được tiến hành như sau:

- Mã hóa:  $C = M \text{ XOR } z$ .
- Giải mã:  $M = C \text{ XOR } z$ .

Quá trình mã hóa và giải mã được thực hiện trên từng bit một.



### 3.3.2. Chứng thực 128-EIA3

#### 3.3.2.1. Giới thiệu

Giải thuật 128-EIA3 dùng để chứng thực một thông điệp có độ dài từ 1 đến 20000 bit. 128-EIA3 sẽ dùng một khóa IK để tạo ra *mã xác thực thông điệp* (Message authentication code – MAC). Đầu vào và đầu ra của giải thuật này như sau:

	Tham số	Kích thước (bits)	Ghi chú
Đầu vào	COUNT	32	Biến đếm
	BEARER	5	Xác định sóng mang (bearer)
	DIRECTION	1	Xác định hướng chuyển dữ liệu là upload hay download
	IK	128	Khóa
	LENGTH	32	Chiều dài (bits) của thông điệp đầu vào
	M	LENGTH	Thông điệp đầu vào
Đầu ra	IM	32	MAC - Mã chứng thực thông điệp

#### 3.3.2.2. Tạo keystream

Giống như 128-EEA3, giải thuật 128-EIA3 cũng cần truyền cho ZUC khóa và vector khởi tạo. Biến IK được dùng làm khóa của ZUC. Biến COUNT cũng được tách thành từng byte:

$$\text{COUNT} = \text{COUNT}_0 \parallel \text{COUNT}_1 \parallel \text{COUNT}_2 \parallel \text{COUNT}_3$$

Dùng biến IV 128-bit làm vector khởi tạo sử dụng cho ZUC. IV sẽ được tách nhỏ thành từng byte và được gán giá trị cho từng byte này. Cách gán giá trị trong giải thuật này khác với trong 128-EEA3:

$$\text{IV}_0 = \text{COUNT}_0, \text{IV}_1 = \text{COUNT}_1, \text{IV}_2 = \text{COUNT}_2,$$

$$IV_3 = \text{COUNT}_3, IV_4 = \text{BEARER} \parallel 000_2, IV_5 = 00000000_2,$$

$$IV_6 = 00000000_2, IV_7 = 00000000_2, IV_8 = IV_0 \oplus (\text{DIRECTION} \ll 7),$$

$$IV_9 = IV_1, IV_{10} = IV_2, IV_{11} = IV_3, IV_{12} = IV_4,$$

$$IV_{13} = IV_5, IV_{14} = IV_6 \oplus (\text{DIRECTION} \ll 7), IV_{15} = IV_7$$

Lúc bấy giờ hai biến CK và IV sẽ được nạp để ZUC hoạt động. Số lần word đầu ra của ZUC trong giải thuật này là  $L = \lceil \text{LENGTH} / 32 \rceil + 2$ . Tách keystream  $z$  mà ZUC vừa sinh ra thành từng bit:

$$z = z_0 \parallel z_1 \parallel \dots \parallel z_{32 \times L - 1}.$$

Định nghĩa  $z[i]$  với  $i$  trong khoảng  $[0, \dots, 32 \times (L-1)]$

$$z[i] = z_i \parallel z_{i+1} \parallel \dots \parallel z_{i+31}.$$

### 3.3.2.3. Tính giá trị MAC

Cho  $T$  là một word 32 bit và được gán giá trị 0. Giá trị MAC được tính bằng đoạn mã giả sau:

```
For each i=0,1,2,...,LENGTH-1
```

```
    if M[i] = 1, then
```

$$T = T \oplus z[i]$$

```
    end if
```

```
end for
```

$$T = T \oplus z[\text{LENGTH}]$$

$$IM = T \oplus z[32 \times (L-1)]$$

Biến IM chính là giá trị MAC đầu ra của giải thuật 128-EIA3.

### 3.4. Tiêu chí thiết kế và tính an toàn của ZUC

Các nhà nghiên cứu gần đây đã đề nghị các tiêu chí cũng như cách thức thiết kế cả 3 lớp của ZUC. Tiêu chí thiết kế nào cũng có nguyên nhân của nó, đặc biệt tiêu chí và cách thiết kế của *hàm phi tuyến F đóng vai trò rất quan trọng* trong việc quyết định đến tính an toàn của ZUC.

#### 3.4.1. Tiêu chí thiết kế LFSR

Tiêu chí thiết kế LFSR bản chất nằm ở chỗ việc lựa chọn *đa thức cơ bản  $f(x)$* , là đa thức kết nối của LFSR. Sau đây là tiêu chí để lựa chọn  $f(x)$  [35]:

1. **Trọng số Hamming** của mỗi *hệ số* khác không là thấp nhất có thể. Phải hiểu trọng số Hamming của hệ số là trọng số Hamming của biểu diễn nhị phân của hệ số đó.
2. Tổng các trọng số Hamming của các *hệ số* khác không (trừ số hạng  $x^{16}$ ) là một số chẵn.
3. **Hệ số** của *số hạng* với bậc cao thứ hai phải khác không.
4. Các bậc của *số hạng* với các *hệ số* khác không (trừ số hạng  $x^{16}$ ) phải khác nhau đôi một.
5. Các **vị trí** của '1' của các *hệ số* khác không trong biểu diễn 2-adic của chúng phải khác nhau đôi một.

Trong đó biểu diễn 2-adic của một hệ số  $u \in GF(p)$  là biểu diễn có dạng:

$$u = u_0 + u_1 2 + u_2 2^2 + \dots + u_{30} 2^{30}$$

ở đây  $p = 2^{31} - 1$  là số nguyên tố [35];  $u_i$  nhận giá trị 0 hoặc 1,  $i = 0, 1, \dots, 30$ . **Biểu diễn 2-adic** cũng chính là **biểu diễn nhị phân** như ta đã biết.

**Đa thức** sau được chọn thỏa tiêu chí nêu trên:

$$f(x) = x^{16} - (2^{15}x^{15} + 2^{17}x^{13} + 2^{21}x^{10} + 2^{20}x^4 + (2^8 + 1)).$$

Ta nhận thấy cách thức tính các phần tử dãy  $s_j$  ( $j \geq 16$ ) được sinh ra bởi LFSR này có phần ngược so với LFSR được nghiên cứu trong **Chương 2**. Cụ thể trong Chương 2, thì  $s_j$  được tính theo:

$$s_j = -\sum_{i=1}^{16} c_i s_{j-i} = -c_1 s_{j-1} - \dots - c_{16} s_{j-16}.$$

Còn trong trường hợp lớp LFSR này thì:

$$s_j = -\sum_{i=15}^0 c_i s_{j-(16-i)} = -c_{15} s_{j-1} - \dots - c_0 s_{j-16}. \quad (3.4.1)$$

Điều này có thể được giải thích, bởi vì toán tử đa thức  $f(E)$  (xem **Phần 2.6.1**) đối với lớp LFSR này là  $f(E)s_j = c_{16}s_j + c_{15}s_{j-1} + \dots + c_0s_{j-16}$ . Trong khi đó đối với LFSR được nghiên cứu ở **Chương 2** là  $f(E)s_j = c_0s_j + c_1s_{j-1} + \dots + c_{16}s_{j-16}$ . Mỗi hệ số  $c_i$  ( $0 \leq i \leq 16$ ) là hệ số tương ứng với số hạng  $x^i$  của đa thức  $f(x)$  trên.

Lý do tại sao  $s_j$  được tính theo công thức (3.4.1) trên trong **Phần 2.6.1** có đề cập, nhằm đảm bảo  $f(E)s_j = 0$  với mọi  $j$  với  $j \geq l$ .  $l$  ở đây là độ phức tạp tuyến tính của dãy được sinh ra, cũng đồng thời là chiều dài của LFSR ( $l = 16$ ).

Ta dễ dàng kiểm tra được  $f(x)$  thỏa mãn tiêu chí trên với biểu diễn nhị phân của các hệ số như sau:

$$2^{15} = \mathbf{1000000000000000}.$$

$$2^{17} = \mathbf{10000000000000000}.$$

$$2^{21} = \mathbf{10000000000000000000}.$$

$$2^{20} = \mathbf{10000000000000000000}.$$

$$2^8 + 1 = \mathbf{100000001}.$$

Vì  $f(x)$  là một đa thức cơ bản trên  $GF(p)$ , nên LFSR sẽ sinh một  $m$ -sequence với chu kỳ  $p^{16} - 1 \approx 2^{496}$  và thỏa mãn các tiên đề ngẫu nhiên **Golomb** (xem **Phụ lục**), điều này đảm bảo về độ ngẫu nhiên của dãy sinh ra bởi lớp LFSR này.

### 3.4.2. Tiêu chí thiết kế của BR

Lớp **BR** là khớp nối giữa LFSR và hàm phi tuyến  $F$ , nó trích 128 bit từ các ô (cell) của LFSR và tạo thành bốn 32-bit word. Thiết kế chính của BR dựa vào tiêu chí sau [35]:

1. Phù hợp cho cài đặt phần mềm.
2. Bốn 32-bit word từ BR có tính ngẫu nhiên tốt về mặt thống kê.
3. Số các bit phủ nhau của bốn 32-bit word trong các thời điểm liên tiếp nhỏ.

Dựa trên tiêu chí trên, BR làm việc như đã đề cập:

$$\text{a) } X_0 = s_{15H} \parallel s_{14L};$$

$$\text{b) } X_1 = s_{11L} \parallel s_{9H};$$

$$\text{c) } X_2 = s_{7L} \parallel s_{5H};$$

$$d) X_3 = s_{2L} \parallel s_{0H},$$

ở đây  $X_i$  ( $i = 0, 1, 2, 3$ ) là đầu ra của BR, và chỉ số  $H$  và  $L$  của một số ô  $s$  cho biết tương ứng 16 bit cao và 16 bit thấp của  $s$ .

Trong thuật toán ZUC, vì các phần tử trong  $GF(p)$  được xác định trong tập  $\{1, 2, \dots, p\}$ , cho nên trong suốt quá trình hồi tiếp của 16-stage LFSR, giá trị 0 sẽ được thay thế bởi  $p$ . Chú ý rằng LFSR sẽ sinh một  $m$ -sequence với chu kỳ  $p^{16} - 1$ . Nên trong một chu kỳ của sequence như vậy, sẽ không có trạng thái nào mà khi tất cả giá trị các ô đều bằng  $p$ . Các kết quả sau được trích dẫn trong [35]:

**Định lý 3.4.1** [35]:  $\Pr(s_i=p) = (p^{15} - 1)/(p^{16} - 1)$ , và cho bất kỳ  $1 \leq a \leq p - 1$  ta có:

$$\Pr(s_i = a) = p^{15} / (p^{16} - 1)$$

**Định lý 3.4.2** [35]:

$$\Pr(s_{iH} = 0) = \Pr(s_{iL} = 0) = (2^{15} - 1)p^{15} / (p^{16} - 1),$$

$$\Pr(s_{iH} = 2^{16} - 1) = \Pr(s_{iL} = 2^{16} - 1) = (2^{15} p^{15} - 1) / (p^{16} - 1),$$

$$\Pr(s_{iH} = a) = \Pr(s_{iL} = a) = 2^{15} p^{15} / (p^{16} - 1),$$

Trong đó  $a$  là một chuỗi nhị phân 16-bit mà không là all-zero (tất cả các bit đều là 0) hay all-one (tất cả các bit đều là 1).

Cho  $X_0, X_1, X_2, X_3$  là bốn 32-bit word được thu bởi xử lý của BR.

Từ **Định lý 3.4.2** và các thuộc tính thống kê tiêu biểu của  $m$ -sequence, [35] cũng kết luận sau:

**Hệ quả 3.4.3** [35]: Cho  $a$  và  $b$  là hai chuỗi nhị phân 16-bit bất kỳ mà không là all-zero hay all-one, thì:

$$\Pr(X_0=(a,b))= 2^{15}p^{15}/(p^{16}-1) \times 2^{15}p^{15}/(p^{16}-1)$$

**Hệ quả 3.4.4** [35]: Cho  $a$  là một chuỗi nhị phân 16-bit bất kỳ mà không là all-zero hay all-one,  $\mathbf{1}$  là một chuỗi nhị phân 16-bit all-one, thì ta có:

$$\Pr(X_0=(a,\mathbf{1}))= \Pr(X_0=(\mathbf{1}, a))= 2^{15}p^{15}/(p^{16}-1) \times (2^{15}p^{15}-1)/(p^{16}-1)$$

**Hệ quả 3.4.5** [35]: Cho  $a$  là một chuỗi nhị phân 16-bit bất kỳ mà không là all-zero hay all-one,  $\mathbf{0}$  là một chuỗi nhị phân 16-bit all-zero, thì ta có:

$$\Pr(X_0=(a,\mathbf{0}))= \Pr(X_0=(\mathbf{0}, a))= 2^{15}p^{15}/(p^{16}-1) \times (2^{15}-1)p^{15}/(p^{16}-1)$$

**Hệ quả 3.4.6** [35]: Cho  $\mathbf{1}$  là một chuỗi 16-bit all-one,  $\mathbf{0}$  là một chuỗi 16-bit all-zero, thì ta có:

$$\Pr(X_0=(\mathbf{0},\mathbf{1}))= \Pr(X_0=(\mathbf{1},\mathbf{0}))= (2^{15}p^{15}-1)/(p^{16}-1) \times (2^{15}-1)p^{15}/(p^{16}-1)$$

**Hệ quả 3.4.7** [35]: Cho  $\mathbf{0}$  là một chuỗi 16-bit all-zero, thì ta có:

$$\Pr(X_0=\mathbf{0})= (2^{15}-1)p^{15}/(p^{16}-1) \times (2^{15}-1)p^{15}/(p^{16}-1)$$

**Hệ quả 3.4.8** [35]: Cho  $\mathbf{1}$  là một chuỗi 16-bit all-one, thì ta có:

$$\Pr(X_0=\mathbf{1})= (2^{15}p^{15}-1)/(p^{16}-1) \times (2^{15}p^{15}-1)/(p^{16}-1)$$

**Hệ quả 3.4.9** [35]: Các biến ngẫu nhiên  $X_0, X_1, X_2$  và  $X_3$  có cùng phân phối xác suất.

Dựa vào các hệ quả trên, ta có thể xem như  $X_0, X_1, X_2, X_3$  có phân phối xác suất đều.

### 3.4.3. Thiết kế và tính an toàn của hàm phi tuyến $F$

Tính an toàn của ZUC phụ thuộc đáng kể vào hàm phi tuyến  $F$ . Trong thực tế được sử dụng ở các mô hình generator, LFSR phải kết hợp với các thanh ghi (LFSR) khác hay dùng *lọc thanh ghi* để *phi tuyến hóa* nhằm đem đến sự an toàn hơn cho mô hình

generator. Và trong kiến trúc của ZUC, sự lọc thanh ghi (lớp LFSR) được thực hiện nhờ (lớp) hàm phi tuyến  $F$ .

Hàm phi tuyến  $F$  là một hàm nén từ 96 bit còn 32 bit. Thiết kế của nó thông qua một số cấu trúc lấy từ thiết kế của *mã khối* (block cipher). Xem xét cả những yêu cầu an toàn và hiệu năng, *thiết kế của hàm phi tuyến  $F$  chủ yếu dựa vào tiêu chí sau* [35]:

1. Nhận đầu vào 96 bit và đầu ra là một 32-bit word.
2. Hàm phi tuyến  $F$  phải mang bộ nhớ.
3. Hàm phi tuyến  $F$  phải sử dụng các S-box để có được *độ phi tuyến cao* và *các thuộc tính mật mã khác*.
4. Hàm phi tuyến  $F$  phải sử dụng phép biến đổi tuyến tính với sự khuếch tán tốt.
5. Dãy sinh ra của hàm phi tuyến  $F$  phải *cân bằng* và khó có thể đoán ra được (*khả năng không thể đoán ra được cao*).
6. Hàm phi tuyến  $F$  phải phù hợp đối với cài đặt phần mềm lẫn phần cứng.
7. Chi phí cài đặt phần cứng của hàm phi tuyến  $F$  phải thấp.

#### 3.4.3.1. Thiết kế và tính an toàn của các S-box $S_0$ và $S_1$

Hai S-box được dùng trong hàm phi tuyến  $F$  tương ứng là  $S_0$  và  $S_1$ . Vì lớp LFSR được thiết kế tốt để nhằm chống lại các tấn công đại số trên  $GF(2)$ , do đó *độ miễn đại số* (algebraic immunity) (xem **Phụ lục**) không là ưu tiên cao nhất trong thiết kế của các S-box [35].

Ở **Phần 2.7.3.3** đã đề cập đến định nghĩa về *độ phi tuyến của hàm Boolean*. Trong khi đó, ta được biết S-box có thể xem như được cấu tạo từ nhiều hàm Boolean (gọi là các hàm thành phần của S-box), và ta cũng có *định nghĩa về độ phi tuyến của S-box*:



**Định nghĩa 3.4.10** (độ phi tuyến của S-box) [37] [38]: **Độ phi tuyến của một S-box** là độ phi tuyến nhỏ nhất trong số các độ phi tuyến của các tổ hợp tuyến tính khác không của các hàm thành phần S-box. Nghĩa là:

$$N_F = \min_g \{N_g \mid g = \bigoplus_{j=1}^m c_j f_j, (c_1, c_2, \dots, c_m) \neq (0, 0, \dots, 0)\}.$$

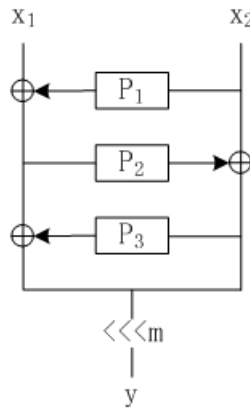
trong đó,  $F = (f_1, \dots, f_m)$  là hàm ánh xạ từ  $GF(2)^n$  thành  $GF(2)^m$  (các  $f_i$  với  $i = 0, 1, \dots, m$  là các hàm Boolean ánh xạ từ  $GF(2)^n$  thành  $GF(2)$ ).  $c_1, c_2, \dots, c_m \in GF(2)$ .  $F$  chính là một  $n \times m$  S-box.

#### A. Thiết kế của S-box $S_0$ :

Thiết kế của S-box  $S_0$  chủ yếu dựa vào 3 tiêu chí sau [35]:

1. Chi phí cài đặt phần cứng thấp.
2. **Độ phi tuyến cao.**
3. **Mức đồng nhất sai phân thấp.**

Dựa trên sự xem xét trên, S-box  $S_0$  được thiết kế sử dụng một **kiến trúc Feistel**, như hình sau:



Hình 22. Kiến trúc của S-box  $S_0$ .

Trong Hình 22, cả hai  $x_1$  và  $x_2$  là các chuỗi 4-bit,  $m = 5$ . Và  $P_1$ ,  $P_2$ ,  $P_3$  là các biến đổi trên trường  $GF(16)$ , được định nghĩa tương ứng trong các bảng Bảng 5, Bảng 6, Bảng 7.

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	9	15	0	14	15	15	2	10	0	4	0	12	7	5	3	9

Bảng 5. Biến đổi  $P_1$ .

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	8	13	6	5	7	0	12	4	11	1	14	10	15	3	9	2

Bảng 6. Biến đổi  $P_2$ .

Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output	2	6	10	6	0	13	10	15	3	3	13	5	0	9	12	13

Bảng 7. Biến đổi  $P_3$ .

➤ **Kết quả thực nghiệm:**

Trong phạm vi của luận văn, dựa vào định nghĩa **độ phi tuyến** và **tính đồng nhất sai phân** của S-box, những thủ tục cần thiết đã được hiện thực (lập trình) để xác định giá

trị các đại lượng này đối với  $S_0$ . Cụ thể **độ phi tuyến** và **mức đồng nhất sai phân** của  $S_0$  đã được đo đạc và có kết quả tương ứng là **96** và **8**. Ngoài ra theo DACAS, **bậc đại số** và **độ miễn đại số** (xem **Phụ lục**) của  $S_0$  tương ứng là **5** và **2**.

Luận văn cũng đã khảo sát **tiêu chuẩn SAC** đối với các hàm thành phần của  $S_0$ . Sau đây là bảng thể hiện số liệu về sự thay đổi giá trị đầu ra của các hàm thành phần  $f_j$  khi bit đầu vào thứ  $i$  thay đổi đối với  $S_0$  (xét tại dòng  $i$  cột  $j$ ):

	$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$
<b>bit 0</b>	96	116	136	132	104	128	120	120
<b>bit 1</b>	128	88	124	144	116	96	128	120
<b>bit 2</b>	120	120	128	128	140	120	96	160
<b>bit 3</b>	136	124	136	132	112	128	128	128
<b>bit 4</b>	128	112	144	144	112	128	128	128
<b>bit 5</b>	128	160	144	112	144	128	128	128
<b>bit 6</b>	128	112	128	128	112	128	128	128
<b>bit 7</b>	128	128	144	160	144	128	128	128

*Bảng 8. Khảo sát sự thay đổi của các hàm nhị phân thành phần  $f_j$  khi bit đầu vào thứ  $i$  bị thay đổi đối với S-box  $S_0$  của hàm phi tuyến  $F$ .*

Ta thấy số trường hợp kết quả của  $f_j$  bị thay đổi khi bit đầu vào thứ  $i$  bị thay đổi rất xấp xỉ 128. Như vậy, khi 1 bit đầu vào bị thay đổi, mỗi bit đầu ra sẽ thay đổi với xác suất xấp xỉ  $\frac{1}{2}$ . Do đó các hàm thành phần của  $S_0$  gần đạt tiêu chuẩn SAC.

## B. Thiết kế của S-box $S_1$ :

Thiết kế của S-box  $S_1$  chủ yếu dựa vào 2 tiêu chí sau [35]:

1. ***Độ phi tuyến của S-box  $S_1$  lớn nhất có thể.***
2. ***Mức đồng nhất sai phân của S-box  $S_1$  là thấp nhất có thể.***

Theo tiêu chí trên, S-box  $S_1$  có công thức như sau:

$$S_1 = Mx^{-1} + B$$

ở đây  $x^{-1}$  là nghịch đảo của  $x$  trên trường  $GF(2^8)$  được định nghĩa thông qua ***đa thức bất khả quy***  $f(x) = x^8 + x^7 + x^3 + x + 1$  trên  $GF(2)$  bậc 8.  $B = 0x55$ ,  $M$  là một ma trận kích thước  $8 \times 8$  và được định nghĩa như sau:

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Vì S-box  $S_1$  tương đương với S-box của chuẩn mã hóa nâng cao AES, do đó  $S_1$  có nhiều thuộc tính giống với S-box của AES, bao gồm ***độ phi tuyến, mức đồng nhất sai phân, bậc đại số*** và ***độ miễn đại số***.

➤ **Kết quả thực nghiệm:**

***Độ phi tuyến, mức đồng nhất sai phân*** của  $S_1$  tương ứng là ***112, 4***. Ta thấy giá trị độ phi tuyến là 112 gần đạt ngưỡng cực đại là 118 (đối với hàm Boolean trên  $GF(2)^8$ ), mức đồng nhất sai phân 4 là giá trị tối tiểu (lý tưởng) của một  $8 \times 8$  S-box như đã đề cập trong **Phần 2.7.3.4**. Trong phạm vi của luận văn, giá trị của ***độ phi tuyến*** và ***mức***

**đồng nhất sai phân** của  $S_1$  đã được kiểm chứng thông qua việc đo đạc bằng các cơ chế lập trình dựa vào định nghĩa của chúng. Ngoài ra theo DACAS, **bậc đại số** và **độ miễn đại số** của  $S_1$  tương ứng là **7** và **2**.

Đối với **tiêu chuẩn SAC**, sau đây là bảng thể hiện số liệu đo đạc được về sự thay đổi giá trị đầu ra của các hàm thành phần  $f_j$  khi bit đầu vào thứ  $i$  thay đổi đối với  $S_1$  (xét tại dòng  $i$  cột  $j$ ):

	$f_0$	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$
<b>bit 0</b>	140	124	120	120	144	124	136	132
<b>bit 1</b>	120	120	136	132	136	124	140	136
<b>bit 2</b>	132	136	144	132	124	136	124	136
<b>bit 3</b>	120	144	128	136	124	128	144	120
<b>bit 4</b>	140	128	140	128	132	136	136	144
<b>bit 5</b>	116	140	136	120	120	124	124	136
<b>bit 6</b>	120	136	136	132	112	124	124	124
<b>bit 7</b>	132	136	124	128	124	132	132	136

*Bảng 9. Khảo sát sự thay đổi của các hàm nhị phân thành phần  $f_j$  khi bit đầu vào thứ  $i$  bị thay đổi đối với S-box  $S_1$  của hàm phi tuyến  $F$ .*

Ta thấy số trường hợp kết quả của  $f_j$  bị thay đổi khi bit đầu vào thứ  $i$  bị thay đổi xấp xỉ 128. Như vậy, khi 1 bit đầu vào bị thay đổi, mỗi bit đầu ra sẽ thay đổi với xác suất xấp xỉ  $\frac{1}{2}$ . Do đó các hàm thành phần của  $S_1$  cũng gần đạt tiêu chuẩn SAC.

➤ **Nhận xét:**

Hai S-box  $S_0$  và  $S_1$  được thiết kế dựa trên *kiến trúc Feistel* và *S-box trong AES* tương ứng. *Kiến trúc Feistel* là một kiến trúc nổi tiếng trong việc thiết kế các thuật toán mã khối, điển hình là thuật toán *DES*. Việc mô phỏng theo cách thức *S-box trong AES*, đã giúp cho  $S_1$  đạt được sự an toàn cần thiết, điều này được giải thích thêm trong phần Phụ lục của phần S-box trong AES về vai trò của *ánh xạ nghịch đảo* (xem **Phụ lục**). ZUC nói chung và hàm phi tuyến  $F$  nói riêng đã sử dụng cả hai S-box  $S_0$  và  $S_1$ . Trong khi AES chỉ sử dụng một S-box tương đương với  $S_1$  và được công nhận là an toàn hiện nay, nên ta có thể đặt niềm tin vào độ an toàn của thành phần phi tuyến của ZUC. Với một độ phi tuyến cao và mức đồng nhất sai phân thấp, đặc biệt là đối với S-box  $S_1$ , hàm phi tuyến  $F$  giúp cho ZUC có khả năng chống lại các tấn công thám mã tuyến tính và thám mã sai phân. Ngoài ra, các hàm thành phần của  $S_0$  và  $S_1$  cũng được *cân bằng*, cụ thể qua thực nghiệm cho được kết quả là tất cả các bảng chân trị của các hàm thành phần của chúng đều cân bằng (có 128 phần tử giá trị ‘1’). Nhờ tính cân bằng này mà kết quả đầu ra của hàm phi tuyến  $F$  có *độ ngẫu nhiên cao*, do vậy dòng khóa mà ZUC sinh ra cũng có độ ngẫu nhiên cao.

Sau đây là bảng so sánh các tính chất của S-box trong AES và hai S-box  $S_0$  và  $S_1$  trong hàm phi tuyến  $F$ :


S-box	Độ phi tuyến	Mức đồng nhất sai phân	SAC
S-box trong AES	112	4	$\sim 1/2$
S-box $S_0$ trong $F$	96	8	$\sim 1/2$
S-box $S_1$ trong $F$	112	4	$\sim 1/2$
Giá trị tối ưu	118	4	$1/2$

*Bảng 10. So sánh các tính chất của S-box trong AES và hai S-box  $S_0$  và  $S_1$  trong hàm phi tuyến  $F$ .*

Trong đó, các giá trị của các tính chất đối với S-box trong AES đã được đo đạc kiểm chứng lại.

## Chương 4. CHƯƠNG TRÌNH THỰC HIỆN

### Tóm tắt chương:

 Chương 4 trình bày kết quả của ứng dụng do chúng tôi thực hiện, với mô hình ứng dụng thử nghiệm mã dòng thông qua thuật toán mã hóa 128-EEA3 sử dụng generator ZUC. Nội dung chương này trình bày các vấn đề chính sau:

- Giới thiệu tổng quan về ứng dụng Voice Chat dùng thuật toán mã hóa 128-EEA3 để đảm bảo tính bí mật của dữ liệu cuộc hội thoại trên đường truyền.
- Trình bày mô hình của ứng dụng với các nội dung: các yêu cầu chức năng của chương trình; mô hình hoạt động của chương trình bao gồm: mô hình hoạt động ở chế độ công khai, mô hình hoạt động ở chế độ riêng tư; giao diện chương trình và hướng dẫn thực thi.
- Thực nghiệm so sánh tốc độ giữa thuật toán mã hóa 128-EEA3 và thuật toán mã hóa AES.
- Tổng kết, đánh giá các kết quả đạt được và chưa đạt được của chương trình thực hiện.



## 4.1. Giới thiệu

Trong phạm vi của luận văn, chúng tôi đã xây dựng thử nghiệm chương trình ứng dụng phương pháp mã dòng. Cụ thể là chúng tôi đã chọn mã dòng **ZUC** là phương pháp được dùng trong mô hình ứng dụng này. Thuật toán mã hóa 128-EEA3 được dùng đến, sử dụng generator ZUC.

Chúng tôi đã chọn ứng dụng **Voice Chat** để hiện thực việc áp dụng mã dòng vào đảm bảo bí mật dữ liệu trên đường truyền. Việc chọn Voice Chat nhằm minh họa hai ý tưởng chính: (1) Chỉ người có khóa mới hiểu nhau và (2) việc mã hóa – giải mã không ảnh hưởng đến quá trình đàm thoại.

Mã dòng là một phương pháp mã đối xứng. Do đó những người muốn nói chuyện với nhau thông qua ứng dụng Voice Chat mà dữ liệu đã được mã hóa, phải dùng chung một khóa mật cho trước. Đây chính là khóa đối xứng nói chung trong phương pháp mã hóa đối xứng. Việc dùng khóa đối xứng này đảm bảo rằng chỉ có những người có được khóa mật mới có thể hiểu được nội dung của cuộc hội thoại, còn những kẻ khác không có khóa cho dù có cố gắng nghe lén cũng không thể nào hiểu được nội dung là gì. Đó cũng chính là ý nghĩa hướng đến và nhằm mang lại của chương trình thực hiện này.

**ZUC** là viết tắt của **Zu Chongzhi**, một nhà toán học và thiên văn học người Trung Quốc vào thế kỷ thứ 5. Generator ZUC và các ứng dụng của nó (128-EEA3, 128-EIA3) được thiết kế bởi **Trung tâm nghiên cứu an toàn tuyến thông và bảo mật dữ liệu** (Data Assurance and Communication Security Research Center – DACAS) của Viện hàn lâm khoa học Trung Quốc.

## 4.2. Mô hình ứng dụng

### 4.2.1. Yêu cầu chức năng chương trình

Ứng dụng được xây dựng để đáp ứng yêu cầu tạo ra một chương trình Voice Chat trực tuyến có hỗ trợ mã hóa dữ liệu. Chương trình theo cấu trúc Client – Server. Thông qua một *máy chủ* (Server), *các máy trạm* (Client) có thể kết nối và trao đổi dữ liệu âm thanh cho nhau. Các máy trạm có thể hoạt động ở hai chế độ: chế độ công khai (public mode) và chế độ riêng tư (private mode). Trong chế độ công khai, dữ liệu được gửi mà không được mã hóa. Điều này dẫn đến nguy cơ dữ liệu có thể bị nghe lén bởi người trung gian. Trong chế độ riêng tư các dữ liệu sẽ được mã hóa bằng thuật toán 128-EEA3 (xem chi tiết ở **Phần 3.3.1**) trước khi gửi đi nhằm đảm bảo không bị nghe lén bởi *kẻ trung gian* (Man in the middle).

Chương trình được viết bằng ngôn ngữ C++ và có thể chạy trên các phiên bản của hệ điều hành Window.

### 4.2.2. Phương pháp tạo keystream

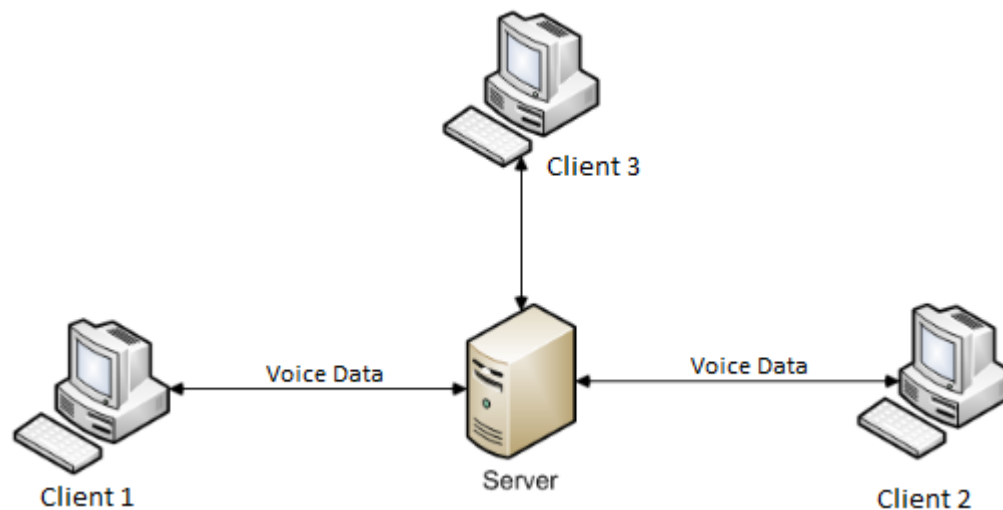
Để sử dụng phương pháp mã hóa 128-EEA3 ta phải tạo ra một keystream từ ZUC để mã hóa từng bit trong gói tin cần mã hóa. Nhưng nếu việc tạo keystream diễn ra cùng lúc với quá trình mã hóa dữ liệu thì tốc độ mã hóa sẽ bị chậm lại. Điều này làm hạn chế ưu điểm về tốc độ của mã hóa dòng. Phương án thiết kế của nhóm chúng tôi là tạo keystream tĩnh từ khóa người dùng trước khi thực hiện mã hóa. Như vậy khi gửi gói tin ta chỉ việc kết hợp từng bit của gói tin với từng bit của keystream có sẵn.

Vấn đề của phương án thiết kế trên là kích thước keystream tĩnh cần phải đạt giá trị bao nhiêu để đảm bảo an toàn cho việc mã hóa. Do kích thước một chu kỳ của keystream do ZUC tạo ra là rất lớn  $((2^{31} - 1)^{16} - 1 \text{ bit})$  nên ta không thể lưu được tất cả thông tin keystream của ZUC. Nhóm chúng tôi đề xuất thêm phương án sử dụng hai keystream

tính khác nhau. Kích thước của mỗi keystream tính là 10MB. Khi mã hóa ta sử dụng một trong hai keystream trên và sau mỗi khoảng thời gian xác định thì ta lại chuyển đổi keystream.

### 4.2.3. Mô hình hoạt động của chương trình

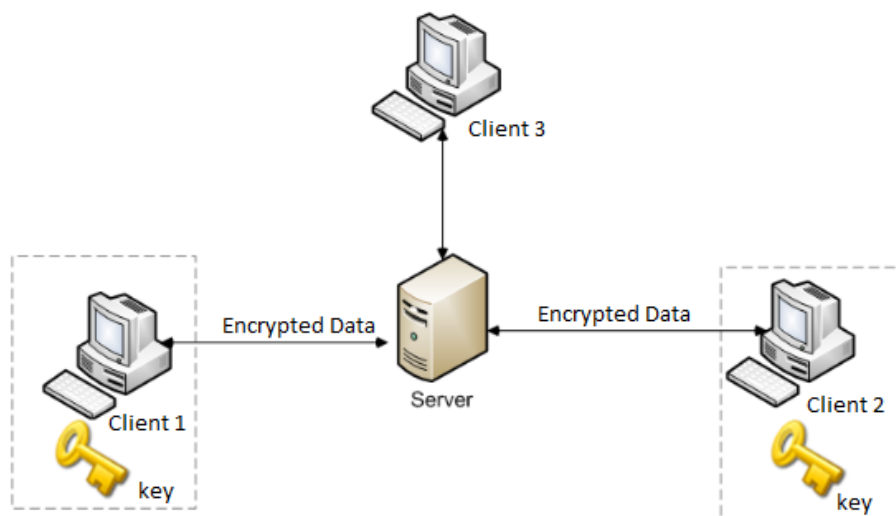
- Mô hình hoạt động ở chế độ công khai:



Hình 23. Mô hình hoạt động của ứng dụng Voice Chat ở chế độ công khai.

Trong mô hình trên **Client1** và **Client 2** là hai máy khách kết nối và trao đổi dữ liệu âm thanh với nhau thông qua **Server**. **Client 3** là máy khách có thể kết nối với server và **nghe trộm được thông tin** trao đổi giữa hai máy trên.

- Mô hình hoạt động ở chế độ riêng tư:

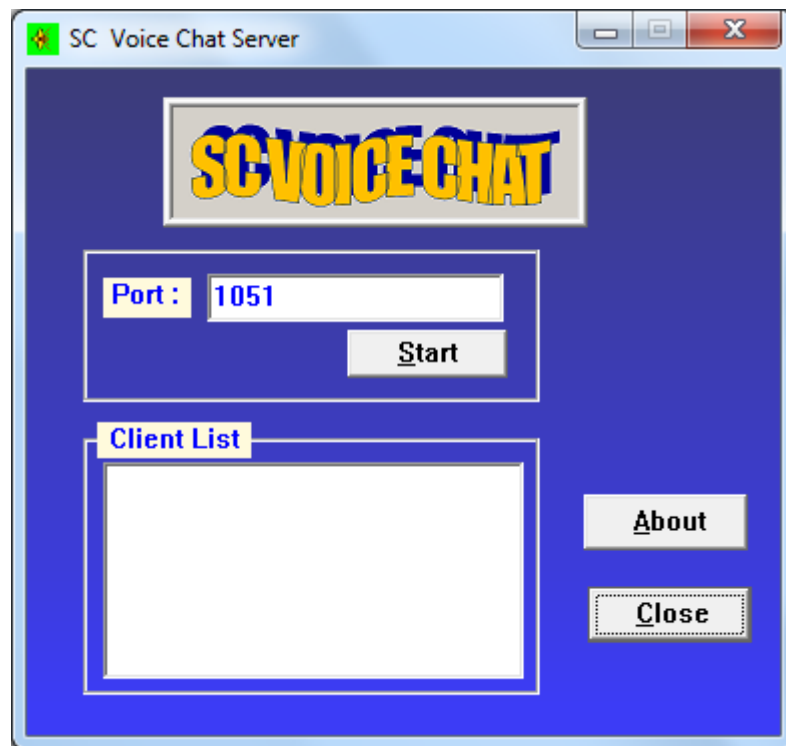


Hình 24. Mô hình hoạt động của ứng dụng Voice Chat ở chế độ riêng tư.

Trong mô hình ở chế độ riêng tư này, hai máy **Client 1** và **Client 2** trao đổi cho nhau trước một khóa cho trước (pre-shared key). Các dữ liệu trước khi gửi sẽ được mã hóa bằng thuật toán 128-EEA3 dựa trên khóa này. Máy khách **Client 3** có thể bắt được dữ liệu trên đường truyền nhưng không giải mã được do không có khóa, nên **không thể nghe trộm được thông tin** trao đổi giữa hai máy trên. Nhờ có khóa mà khi dữ liệu được gửi đến bên kia, nó sẽ được giải mã để Client 1 (Client 2) có thể hiểu được nội dung thông điệp Voice Chat giữa hai máy.

#### 4.2.4. Giao diện chương trình và hướng dẫn thực thi

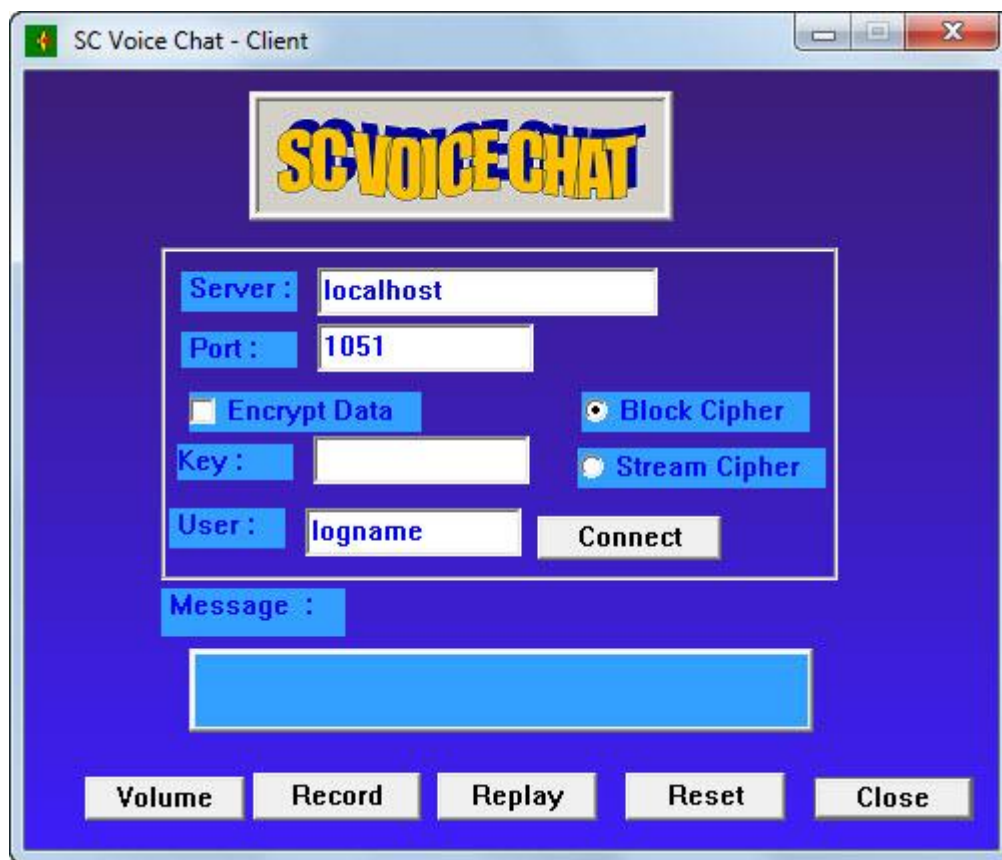
Tại máy chủ, chạy chương trình **SCVoiceChat-server.exe**. Chương trình này có giao diện như sau:



Hình 25. Giao diện chương trình *SCVoiceChat-server.exe*.

Để tạo hệ thống Voice Chat, ta chọn cổng dịch vụ rồi nhấn nút **Start**, danh sách các thông tin máy khách truy cập sẽ được hiển thị trong bảng **Client List**.

Để đăng nhập máy chủ ta chạy chương trình *SCVoiceChat-client.exe* ở máy khách. Chương trình có giao diện như sau:



Hình 26. Giao diện chương trình *SCVoiceChat-Client.exe*

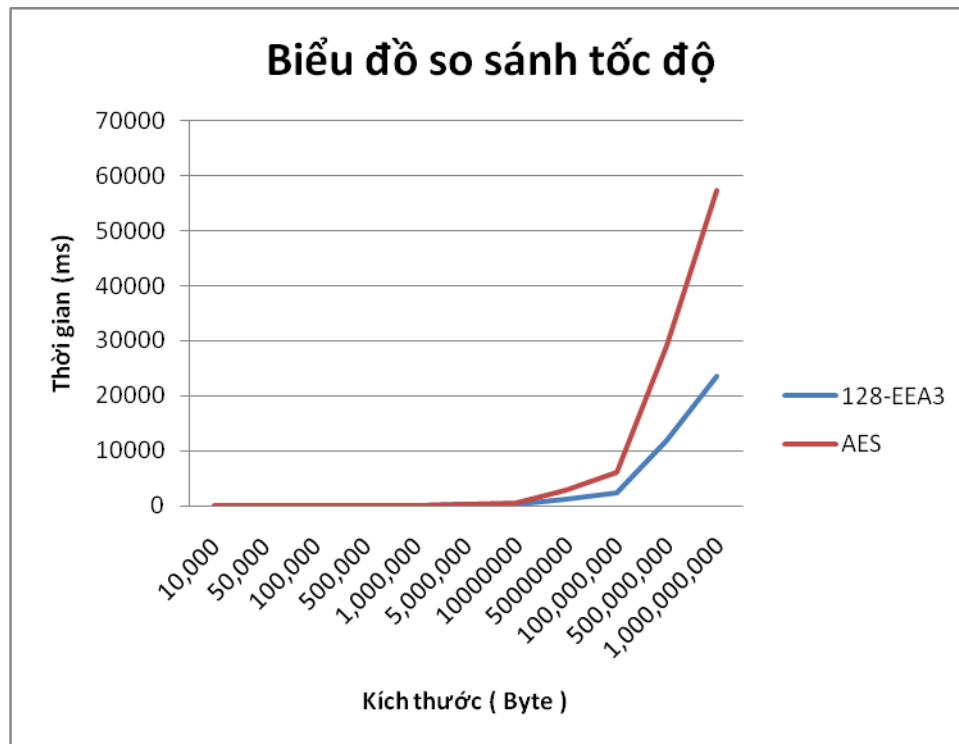
Ta chọn tên đăng nhập, điền địa chỉ mạng của máy chủ và cổng tương ứng. Nếu sử dụng chế độ riêng tư ta chọn chức năng mã hóa dữ liệu và nhập khóa vào. Sau khi kết nối thành công với server ta có thể chọn một máy khách trong hệ thống để trò chuyện. Chương trình này hỗ trợ hai phương pháp mã hóa dữ liệu là mã dòng với thuật toán 128-EEA3 và mã hóa khối sử dụng thuật toán AES ở kiểu hoạt động *CTR* (CTR mode of operation).

### 4.3. Kết quả thực nghiệm

Để minh họa sự khác biệt giữa mã khối và mã dòng, nhóm chúng tôi so sánh tốc độ mã hóa và giải mã của giải thuật 128-EEA3 trong ứng dụng trên với giải thuật AES dựa trên kích thước của khối dữ liệu. Kết quả so sánh được thể hiện trong bảng sau:

STT	Kích thước (byte)	Thời gian thực thi (miligiây)			
		Mã hóa 128-EEA3	Mã hóa AES	Giải mã 128-EEA3	Giải mã AES
1	100	0	0	0	0
2	500	0	0	0	0
3	1,000	0	0	0	0
4	5,000	0	0	0	0
5	10,000	0	0	0	0
6	50,000	0	1	0	1
7	100,000	1	3	0	3
8	500,000	4	14	7	13
9	1,000,000	9	29	15	24
10	5,000,000	57	140	67	139
11	10,000,000	121	276	112	286
12	50,000,000	587	1389	569	1419
13	100,000,000	1252	2937	1203	3024
14	500,000,000	5893	14445	6005	14745
15	1,000,000,000	11774	28401	11852	28860

*Bảng 11. So sánh tốc độ thực thi giữa giải thuật 128-EEA3 và giải thuật AES.*



Hình 27. Biểu đồ so sánh tốc độ thực thi giữa 128-EEA3 và AES.

Kết quả thực nghiệm cho thấy: khi dữ liệu càng lớn thì tốc độ mã hóa dữ liệu của giải thuật 128-EEA3 càng cao hơn của mã khối AES.

Thực nghiệm trên đã thực hiện trên máy có cấu hình: CPU Intel Core 2 Duo 2.10GHz; DDRAM2 1GB; Processor 32 bit.

#### 4.4. Tổng kết chương

Sau khi được hiện thực, chương trình đã được thử nghiệm để kiểm tra hiệu năng mang lại. Dữ liệu âm thanh được mã hóa và giải mã bằng 128-EEA3 sử dụng generator ZUC với thời gian không đáng kể đối với khả năng nghe của tai người. Nghĩa là độ trễ của ứng dụng Voice Chat không đáng kể, và âm thanh nghe được khá trong (clear), đáp ứng được yêu cầu về tốc độ của một ứng dụng Voice Chat. Thực nghiệm dựa trên ứng dụng này cho thấy mã hóa dòng có tốc độ cao hơn mã hóa khối. Sự chênh lệch về tốc



độ càng thể hiện rõ khi ta mã hóa trên bản rõ càng lớn. Đặc biệt với các ứng dụng hiện đại có lưu lượng truyền tải lớn như video chat, xem phim trực tuyến mã dòng ZUC sẽ có ưu thế hơn mã khối do tốc độ thực thi nhanh hơn. Ngoài ra với những vấn đề đã nghiên cứu về độ an toàn của generator ZUC (xem **Phần 3.4**), ứng dụng này mang đến sự an toàn cho cuộc hội thoại Voice Chat.

Tuy nhiên ứng dụng còn một số hạn chế như chưa giải quyết được bài toán đồng bộ dữ liệu giữa hai máy khi giải mã trong trường hợp có một hay nhiều gói tin bị sai hoặc thất lạc trên đường truyền (xem **Phần 2.2** và **2.3.1**).

# KẾT LUẬN

**Luận văn đã đạt được các kết quả sau:**

- Luận văn đã khảo sát và hệ thống hóa khá đầy đủ các lý thuyết quan trọng về mã dòng.
- Chúng tôi đã khảo sát, phân tích cũng như thực nghiệm và đo đạc các đặc tính mật mã quan trọng của mã dòng ZUC, điển hình nhất là hàm phi tuyến  $F$  trong kiến trúc của ZUC. Với những khảo sát đó, chúng tôi đã làm sáng tỏ các yếu tố quyết định đến tính an toàn của một mô hình mã dòng nói chung và mã dòng ZUC nói riêng.
- Chúng tôi cũng đã xây dựng thành công ứng dụng Voice Chat sử dụng mã dòng ZUC để đảm bảo tính bí mật dữ liệu trên đường truyền. Qua đó minh họa được ưu thế của mã dòng ZUC so với mã khối AES về mặt tốc độ bằng những thực nghiệm với chương trình đã được xây dựng.

Song cùng với các kết quả đạt được đó, luận văn của chúng tôi cũng chưa thực sự hoàn hảo do những **kết quả chưa đạt được** sau:

- Do thời gian thực hiện hạn hẹp nên chúng tôi chưa đề xuất ra được một mô hình mã dòng nào như mong muốn trước khi bắt tay vào đề tài.
- Tính an toàn của một hệ mã nói chung và mã dòng nói riêng sẽ được làm sáng tỏ hơn nếu nó được trải qua sự thử nghiệm của các phương pháp thám mã. Do độ khó và sâu của các phương pháp thám mã nên chúng tôi cũng chưa nghiên cứu trong luận văn đại học này.
- Chương trình thực hiện của chúng tôi có một số điểm còn tồn tại như: chưa giải quyết được bài toán đồng bộ dữ liệu giữa hai máy khi giải mã trong trường hợp

có một hay nhiều gói tin bị sai hoặc thất lạc trên đường truyền; chương trình chỉ được hiện thực để chạy trên máy tính thường (máy để bàn và xách tay) mà chưa thể chạy trên máy điện thoại di động.

# HƯỚNG PHÁT TRIỂN

Dựa trên các kết quả đã đề cập, chúng tôi đưa ra hướng phát triển của luận văn như sau:

- Phát triển ứng dụng Voice Chat cho điện thoại di động dùng mã dòng ZUC để đảm bảo tính bí mật của dữ liệu trên đường truyền, ứng dụng chạy trên các hệ điều hành như Window Mobile, Android.
- Nghiên cứu các phương pháp thám mã trên mã dòng, áp dụng các phương pháp thám mã này để kiểm định tính an toàn của mã dòng ZUC.
- Nghiên cứu các phương pháp kiểm định tính ngẫu nhiên của dãy (sequence), nhằm áp dụng cho dòng khóa được sinh ra bởi ZUC.

# TÀI LIỆU THAM KHẢO

- [1] M.J.B. Robshaw, “*Stream Ciphers*”, RSA Laboratories Technical Report TR-701, 1995, pp. 1 – 3.
- [2] Trang web của hiệp hội GSMA, “*GSM Security Algorithms*”,  
[http://gsmworld.com/our-work/programmes-and-initiatives/fraud-and-security/gsm\\_security\\_algorithms.htm](http://gsmworld.com/our-work/programmes-and-initiatives/fraud-and-security/gsm_security_algorithms.htm)
- [3] Majithia Sachin, Dinesh Kumar, “*Implementation and Analysis of AES, DES and Triple DES on GSM Network*”, IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.1, January 2010, pp. 1 – 2.
- [4] Thomas W.Cusick, Cunsheng Ding, Ari Renvall , “*Stream Ciphers and Number Theory*”, North-Holland Mathematical Library, 2003.
- [5] Tom Carter, “*An introduction to information theory and entropy*”, Complex Systems Summer School, June – 2007, pp. 55 – 58.
- [6] Adi Shamir, “*Stream Ciphers: Dead or Alive?*”, ASIACRYPT, 2004, pp. 22 – 41.
- [7] Steve Babbage, “*Stream Ciphers – What does industry want?*”, The State of the Art of Stream Ciphers, Thursday October 14, 2004, pp. 9 – 11.
- [8] Franz Pichler, “*Finite state machine modeling of cryptographic systems in loops*”, Springer, 1998, pp. 1 – 2.
- [9] W. Diffie, M. Hellman, “*Privacy and authentication – An introduction to cryptography*”, Proc. IEEE 67(3), 1979, pp. 415 – 417.

- [10] Joseph Lano, “*CRYPTANALYSIS AND DESIGN OF SYNCHRONOUS STREAM CIPHERS*”, Katholieke Universiteit Leuven – Faculteit Ingenieurswetenschappen Arenbergkasteel, B-3001 Heverlee (Belgium), 2006.
- [11] Joan B. Plumstead, “*Inferring a sequence generated by a linear congruence*”, Springer, 1998, pp. 317 – 318.
- [12] Chung-Chih Li, Bo Sun, “*Using Linear Congruential Generators for Cryptographic Purposes*”, Computer Science Department – Lamar University – Beaumont, TX 77710, pp. 2 – 3.
- [13] Werner Alexi, Benny Chor, Oded Goldreich, Claus P. Schnorr, “*RSA and Rabin functions: certain parts are as hard as the whole*”, Society for Industrial and Applied Mathematics Philadelphia, PA, USA, ISSN: 0097-5397, 1988, pp. 197 – 208.
- [14] Edgar Ferrer, “*Acceleration of Finite Field Arithmetic with an Application to Reverse Engineering Genetic Networks*”, University of Puerto Rico at Mayaguez, 2008.
- [15] J. Guajardo, S. S. Kumar, C. Paar, J. Pelzl, “*Efficient Software-Implementation of Finite Fields with Applications to Cryptography*”, Springer Science + Business Media B.V. 2006, pp. 3 – 9.
- [16] Richard A. Mollin, “*An Introduction to Cryptography – 2nd ed*”, Taylor & Francis Group, LLC, 2007.
- [17] James L. Massey, “*Shift-Register Synthesis and BCH Decoding*”, IEEE TRANSACTIONS ON INFORMATION THEORY, 1969, pp. 122 – 125.

- [18] A. Menezes, P. van Oorschot, S. Vanstone, “*Handbook of Applied Cryptography*”, CRC Press, 1997.
- [19] E. Kowalski, “*Exponential sums over finite fields, I: elementary methods*”, ETH Zurich – D-MATH, Ramistrasse 101, 8092 Zurich, Switzerland, pp. 1 – 15.
- [20] János Folláth, “*Pseudorandom Binary Sequences Over Fields of Characteristic 2*”, International Conference on Uniform Distribution Marseille, CIRM, 21-25/01/2008, pp. 8 – 11.
- [21] Nguyễn Chánh Tú, “*Lí thuyết mở rộng trường và Galois*”, Giáo trình điện tử, Khoa Toán ĐHSP Huế, 12 – 2006.
- [22] Randy Yates, “*A Coding Theory Tutorial*”, Digital Signal Labs, 19–Aug–2009.
- [23] T.Beth and F.Piper. “*The stop-and-go generator*”, T. Beth and N. Cot and I. Ingemarsson, editors, *Advances in Cryptology – Eurocrypt '84*, pp. 88-92, Springer-Verlag, Berlin, 1984, pp. 88 – 92.
- [24] D. Gollmann, “*Pseudo-random properties of cascade connections of clock controlled shift registers*”, T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology – Eurocrypt '84*, pp. 93-98, Springer-Verlag, Berlin, 1985, pp. 93 – 98.
- [25] W. Meier and O. Staffelbach, “*The self-Shrinking generator*”, *Advances in Cryptology – Eurocrypt '94*, Springer-Verlag, 1995, pp. 205 – 214.
- [26] Dong Hoon Lee, Jaeheon Kim, Jin Hong, Jae Woo Han, Dukjae Moon, “*Algebraic Attacks on Summation Generators*”, *Fast Software Encryption 2004*, 2004, pp. 34 – 48.

- [27] Martin Hell, Thomas Johansson, Willi Meier, “*Grain - a stream cipher for constrained environments*”, International Journal of Wireless and Mobile Computing, Vol. 2, No. 1, 2007, pp. 86 – 93.
- [28] Paul Yousef, “*GSM-Security a Survey and Evaluation of the Current Situation*”, Master's thesis, Linkoping Institute of Technology, 5-Mar-2004.
- [29] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, “*Specification of the A5/3 Encryption Algorithms for GSM and ECSD, and the GEA3 Encryption Algorithm for GPRS*”. Document 1: “*A5/3 and GEA3 Specifications*” (Release 6), Sep-2003.
- [30] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, “*Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3*”. Document 1: “*128-EEA3 and 128-EIA3 Specification*”, 4-Jan-2011.
- [31] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, “*Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3*”. Document 2: “*ZUC Specification*”, 4-Jan-2011.
- [32] Jennifer Seberry, Xian-Mo Zhang, Yuliang Zheng, “*Nonlinearity and Propagation Characteristics of Balanced Boolean Functions*”, Department of Computer Science – The University of Wollongong, pp. 2 – 25.
- [33] Trần Minh Triết, “*Nghiên cứu và phát triển các phương pháp bảo vệ thông tin dựa trên AES*”, Luận án Tiến sĩ, Đại học Khoa học Tự nhiên Tp.HCM, 2009.
- [34] K. Nyberg, “*Differentially uniform mappings for cryptography*”, EUROCRYPT '93, LNCS vol. 765, Springer-Verlag, 1993, pp. 57 – 65.



- [35] 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, “*Specification of the 3GPP Confidentiality and Integrity Algorithms 128-EEA3 & 128-EIA3*”. Document 4: “*Design and Evaluation Report*”, 18-Jan-2011.
- [36] Claude Carlet, “*Boolean Functions for Cryptography and Error Correcting Codes*”, University of Paris 8, France.
- [37] Josef Pieprzyk, Chris Charnes, Jennifer Seberry, “*On the Immunity of S-boxes against Linear Cryptanalysis*”, Center for Computer Security Research, Department of Computer Science, University of Wollongong, pp. 1 – 9.
- [38] Xian-Mo Zhang, Yuliang Zheng, “*On Nonlinear Resilient Functions*”, EUROCRYPT’95, France, May 1995, pp. 3 – 15.
- [39] Simon Fischer, Willi Meier, “*Algebraic Immunity of S-boxes and Augmented Functions*”, FHNW, CH-5210 Windisch, Switzerland.
- [40] Hồ Văn Quân, “*Lý thuyết thông tin*”, Khoa CNTT – ĐHBK TPHCM, pp. 45 – 53.
- [41] Ghizlane ORHANOU, Saïd EL HAJJI, Youssef BENTALEB, Jalal LAASSIRI “*EPS Confidentiality and Integrity mechanisms Algorithmic Approach*”, IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 4, No 4, July 2010, pp. 15 – 22.

# Phụ lục A. Một số thuộc tính mật mã khác của hàm Boolean

Ngoài các thuộc tính mật mã quan trọng của hàm Boolean, ảnh hưởng đến tính an toàn của generator như: tính cân bằng (balancedness), độ phi tuyến (nonlinearity), tiêu chuẩn SAC, miễn tương quan (correlation immunity), còn có các thuộc tính khác của hàm Boolean cũng ảnh hưởng đến tính tan toàn của generator đó là **bậc đại số** (algebraic degree), **độ miễn đại số** (algebraic immunity). Giống như độ phi tuyến, khái niệm bậc đại số và độ miễn đại số không chỉ có ở hàm Boolean mà còn có ở S-box, phụ lục còn trình bày khái niệm của chúng đối với S-box.

## A.1. Bậc đại số của hàm Boolean

Một cách biểu diễn của hàm Boolean hay được dùng trong mật mã là biểu diễn đa thức  $n$  biến trên  $GF(2)$ , có dạng [36]:

$$f(x) = \bigoplus_{I \in P(N)} a_I \left( \prod_{i \in I} x_i \right) = \bigoplus_{I \in P(N)} a_I x^I,$$

ở đây  $P(N)$  là ký hiệu *tập lũy thừa* (power set) của  $N = \{1, 2, \dots, n\}$  (tập lũy thừa của  $N$  là tập bao gồm tất cả các tập con của  $N$ ). Dạng biểu diễn này được gọi là *dạng chuẩn đại số* (Algebraic Normal Form – ANF) của hàm Boolean. Mỗi hàm Boolean tồn tại duy nhất một ANF [36].

Bậc của ANF được ký hiệu là  $d^\circ f$  và được gọi là **bậc đại số** (algebraic degree) của hàm  $f$ :  $d^\circ f = \max\{|I| \mid a_I \neq 0\}$ , ở đây  $|I|$  là ký hiệu kích thước của  $I$ . Bậc đại số còn có tên gọi khác là *bậc phi tuyến* (nonlinear order).

**Ví dụ [36]:** Cho hàm  $f$  với *truth-table* là:

$x_1$	$x_2$	$x_3$	$f(x)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Nó là tổng của các *hàm nguyên tử* (atomic function)  $f_1, f_2$  và  $f_3$  với các *truth-table* tương ứng là:

$x_1$	$x_2$	$x_3$	$f_1(x)$	$f_2(x)$	$f_3(x)$
0	0	0	0	0	0
0	0	1	1	0	0

0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	0	1

Hàm  $f_1(x)$  nhận giá trị 1 nếu và chỉ nếu  $1 \oplus x_1 = 1$ ,  $1 \oplus x_2 = 1$  và  $x_3 = 1$ , điều này xảy ra nếu và chỉ nếu  $(1 \oplus x_1)(1 \oplus x_2)x_3 = 1$ . Vì vậy ANF của  $f_1$  có thể thu được bằng triển khai của tích  $(1 \oplus x_1)(1 \oplus x_2)x_3$ . Tương tự đối với  $f_2$  và  $f_3$ , từ đó ta xem ANF của  $f$  bằng  $(1 \oplus x_1)(1 \oplus x_2)x_3 \oplus x_1(1 \oplus x_2)x_3 \oplus x_1x_2x_3 = x_1x_2x_3 \oplus x_2x_3 \oplus x_3$ . Từ đây dễ dàng nhận ra bậc đại số của  $f$  là 3.

Trong trường hợp hàm  $f$  có nhiều bit đầu vào hơn (tương đương với truth-table phức tạp hơn), có một thuật toán giúp ta tính được ANF của  $f$  từ truth-table của nó, đó là thuật toán *Fast Mobius Transform*. Đây là một thuật toán dùng phương pháp chia để trị [36].

Các hàm mật mã phải có bậc đại số cao. Thực vậy, tất cả các hệ thống mật mã sử dụng các hàm Boolean phải mang lại sự hỗn độn cho phương pháp mã, như kết hợp hoặc lọc các hàm trong mã dòng, các hàm Boolean ẩn trong S-box của mã khối [36], hay S-box được dùng thuật toán ZUC.

Trong trường hợp dùng kiểu **generator lọc trong mã dòng**, nếu  $L$  là chiều dài của LFSR và nếu đa thức hồi tiếp (kết nối) là đa thức cơ bản, thì độ phức tạp tuyến tính của dãy đầu ra của generator thỏa:

$$LC \leq \sum_{i=0}^{d \circ f} \binom{L}{i}$$

Như vậy **bậc đại số** của  $f$  có giá trị cao sao cho **độ phức tạp tuyến tính** có thể có giá trị cao. **Nếu phương pháp mã sử dụng hàm Boolean với bậc đại số thấp có thể gây ra khả năng bị tấn công sai phân cao** [36].

Bậc đại số của hàm Boolean là cơ sở để xây dựng khái niệm về **bậc đại số của S-box**, như sau:

**Định nghĩa bậc đại số của S-box** [38]: *Bậc đại số của S-box là bậc đại số nhỏ nhất trong số các bậc đại số của các tổ hợp tuyến tính khác không của các hàm thành phần S-box. Nghĩa là:*

$$d \circ F = \min_g \{d \circ g \mid g = \bigoplus_{j=1}^m c_j f_j, (c_1, c_2, \dots, c_m) \neq (0, 0, \dots, 0)\}.$$

trong đó,  $F = (f_1, \dots, f_m)$  là hàm ánh xạ từ  $GF(2)^n$  thành  $GF(2)^m$  (các  $f_i$  với  $i = 0, 1, \dots, m$  là các hàm Boolean ánh xạ từ  $GF(2)^n$  thành  $GF(2)$ ).  $F$  chính là một  $n \times m$  S-box.

## A.2. Độ miễn đại số của hàm Boolean

**Định nghĩa** [36]: Cho hàm Boolean  $f$ , tìm hàm  $g$  với bậc nhỏ nhất  $d$ , sao cho  $f \cdot g = 0$  hoặc  $(f + 1) \cdot g = 0$ . **Độ miễn đại số** (algebraic immunity) của  $f$  là  $d$ .

**Định lý** [36]: Cho bất kỳ hàm Boolean  $f$  với  $n$  biến, độ miễn đại số lớn nhất là  $\lceil n/2 \rceil$ .

Độ miễn đại số càng lớn thì phương pháp mã sử dụng hàm Boolean càng an toàn chống lại các **tấn công đại số** (Algebraic attacks).

Cho một S-box  $S$ , ta có ý niệm về các **phương trình ẩn** (implicit equations) có dạng  $F(x, y) = 0$  với mỗi  $x \in GF(2)^n$  và  $y = S(x)$ . Phương trình này có dạng ANF là:  $F(x, y) = \sum c_{\alpha, \beta} x^\alpha y^\beta = 0 \pmod 2$ , với các hệ số  $c_{\alpha, \beta} \in GF(2)$ ;  $\alpha, \beta \in GF(2)^n$ ;  $x^\alpha = (x_1^{\alpha_1} \cdots x_n^{\alpha_n})$ . Nếu biết trước  $y$ , phương trình có thể được viết  $F_y(x) = 0$ , gọi là **phương trình điều kiện** (conditional equation). Ta có ý niệm bậc  $d$  của phương trình điều kiện theo  $x$  là:  $d = \max\{WH(\alpha), c_{\alpha, \beta} = 1\} \leq n$  với  $WH(\alpha)$  là trọng số Hamming của  $\alpha$ . Vấn đề đặt ra là phương trình  $F_y(x) = 0$  nào có bậc nhỏ nhất tương ứng với một đầu ra  $y$  cho trước, lúc đó ta gọi phương trình điều kiện với bậc nhỏ nhất.

Sau đây cũng giới thiệu về định nghĩa độ miễn đại số của S-box:

**Định nghĩa độ miễn đại số của S-box [39]:** Xét một S-box  $S: GF(2)^n \rightarrow GF(2)^m$ . Cho một số đầu ra  $y$  cố định, xác định  $d$  là bậc nhỏ nhất của một **phương trình điều kiện**  $F_y(x) = 0$  đúng với mọi  $x \in S^{-1}(y)$ . **Độ miễn đại số của  $S$**  được định nghĩa là giá trị  $d$  nhỏ nhất trên mọi  $y \in GF(2)^m$ .

## Phụ lục B. S-box trong AES

Trong AES, mỗi byte  $y$  được thay thế sử dụng bảng thay thế (cố định) S-box được xác định như sau [33]:

- Lấy nghịch đảo  $z = y^{-1} \in GF(2^8)$  với quy ước  $0^{-1} = 0$ .
- Cho  $(z_0, z_1, \dots, z_7)$  là biểu diễn nhị phân của  $z$ . Thực hiện ánh xạ affine trên trường  $GF(2^8)$  với biểu diễn nhị phân  $z$ . Kết quả  $t = (t_0, t_1, \dots, t_7)$  được xác định như sau:

$$\begin{pmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \\ z_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Thành phần chính của S-box trong AES là **ánh xạ nghịch đảo** trên trường  $GF(2^8)$ . Nhờ có ánh xạ nghịch đảo này, đã giúp S-box đạt được tính an toàn tối ưu đối với phương pháp thám mã sai phân và phương pháp thám mã tuyến tính [33].

Trong S-box này, ánh xạ affine trên  $GF(2^8)$  được sử dụng làm bước hậu xử lý nhằm loại bỏ các điểm bất biến ( $0 \rightarrow 0, 1 \rightarrow 1$ ) trong ánh xạ nghịch đảo [33].

## Phụ lục C. Một số khái niệm khác

### C.1. Lượng tin

**Lượng tin** (measure of information) là một khái niệm trong Lý thuyết thông tin dùng để so sánh định lượng các tin tức với nhau [40].

Nếu số tin trong tập các tin càng nhiều thì sẽ mang lại một lượng tin càng lớn khi nhận được một tin (giả sử các tin có khả năng xuất hiện bình đẳng như nhau). Một tin có xác suất xuất hiện càng nhỏ thì có lượng tin càng lớn. Sau đây là định nghĩa của lượng tin.

Xét một nguồn  $A = \{a_1, a_2, \dots, a_m\}$  với các xác suất xuất hiện là  $\Pr(a_i)$ ,  $i = 1, 2, \dots, m$ . Ký hiệu lượng tin của mỗi tin  $a_i$  là  $I(a_i)$

**Định nghĩa** (lượng tin) [40]:

- *Lượng tin của một tin được đo bằng logarit của nghịch đảo xác suất xuất hiện của tin đó:*

$$I(x) = \log \frac{1}{\Pr(x)} = -\log \Pr(x)$$

- *Lượng tin chứa trong một dãy  $x = a_1 a_2 \dots a_n$  với  $a_i \in A$  là:*

$$I(x) = \log \frac{1}{\Pr(x)} = -\sum_{i=1}^n \log \Pr(a_i)$$

Đơn vị của lượng tin tùy thuộc vào cách chọn cơ số logarit. Nếu cơ số là 2 thì đơn vị là **bits**, cơ số là  $e$  thì đơn vị là **nats**, cơ số là 10 thì đơn vị là **Hartley**. Ta không cần quan tâm đến cơ số, thông thường nếu được ghi là  $\log(x)$  ta hiểu là  $\log_2(x)$ .

Lượng tin riêng của một tin  $a_i \in A$  chỉ có ý nghĩa đối với chính tin đó chứ không phản ánh được giá trị tin tức của nguồn  $A$ . Từ đó dẫn đến khái niệm **lượng tin trung bình** như sau.

**Định nghĩa** (lượng tin trung bình) [40]: *Lượng tin trung bình của một nguồn tin  $A$  là lượng tin trung bình chứa trong một ký hiệu bất kì của nguồn tin. Nó được ký hiệu là  $I(A)$  và công thức tính:*

$$I(A) = \sum_{a_i \in A} \Pr(a_i) I(a_i) = -\sum_{a_i \in A} \Pr(a_i) \log \Pr(a_i).$$

## C.2. Các tiên đề ngẫu nhiên Golomb

Ta khảo sát trên dãy nhị phân.

Cho  $s$  là một **dãy** (sequence). Một **run** của  $s$  là một dãy con của  $s$  bao gồm liên tiếp các phần tử 0 hoặc 1. Run của 0 gọi là **gap**, của 1 gọi là **block**.



**Ví dụ:** Dãy 0100111, có 4 run là: “0”, “1”, “00”, “111”.

Ngoài những gì đã đề cập về hàm tự tương quan của dãy trên trường  $GF(q)$  trong **Phần 2.6.3**, ta còn có định nghĩa về hàm tự tương quan của dãy nhị phân như sau:

**Định nghĩa** (hàm tự tương quan của dãy nhị phân) [18]: Cho  $s = s_0, s_1, s_2, \dots$  là một dãy tuần hoàn với chu kỳ  $N$ . **Hàm tự tương quan** của  $s$  là hàm  $C(t)$  được định nghĩa như:

$$C(t) = \frac{1}{N} \sum_{i=0}^{N-1} (2s_i - 1) \cdot (2s_{i+t} - 1), \text{ với } 0 \leq t \leq N-1.$$

Sau đây là đến định nghĩa các tiên đề ngẫu nhiên Golomb:

**Định nghĩa các tiên đề ngẫu nhiên Golomb** [18]: Cho  $s$  là một dãy tuần hoàn với chu kỳ  $N$ . Các tiên đề ngẫu nhiên Golomb như sau:

1. Trong chu kỳ  $s^N$  của  $s$ , số phần tử ‘1’ khác số phần tử ‘0’ nhiều nhất 1 đơn vị.
2. Trong chu kỳ  $s^N$ , có ít nhất  $1/2$  các run có chiều dài 1, ít nhất  $1/4$  có chiều dài 2, ít nhất  $1/8$  có chiều dài 3,.... Lưu ý số lượng các run tương ứng với mỗi chiều dài nhỏ nhất là 1. Hơn nữa, đối với mỗi chiều dài này, số các gap và các block gần như nhau. Như vậy tiên đề 2 này có ý bao hàm cả tiên đề 1 ở trên.
3. Hàm tự tương quan  $C(t)$  nhận hai giá trị. Cho một số số nguyên  $K$ ,

$$N \cdot C(t) = \sum_{i=0}^{N-1} (2s_i - 1) \cdot (2s_{i+t} - 1) = \begin{cases} N, & t = 0 \\ K, & 1 \leq t \leq N-1 \end{cases}$$

Một dãy nhị phân thỏa mãn các tiên đề ngẫu nhiên Golomb được gọi là một **pseudo-noise sequence** hoặc một **pn-sequence**. Trong thực tế, các **pn-sequence** như các dãy **m-sequence** được sinh ra từ *maximum-length* LFSR (xem **Phần 2.6.1**).

**Ví dụ:** Xem xét một dãy tuần hoàn  $s$  có chu kỳ  $N = 15$ . Với dãy chu kỳ:

$$s^{15} = 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1.$$

Xem xét  $s$  với các tiên đề Golomb:

1. Trong  $s^{15}$ , số các bit 0 là 7, các bit 1 là 8.
2. Trong  $s^{15}$ , có 8 run. Có 4 run chiều dài 1, có 2 run chiều dài 2, có 1 run chiều dài 3, có 1 run chiều dài 4.
3. Hàm tự tương quan  $C(t)$  nhận hai giá trị:  $C(0) = 1$  và  $C(t) = -1/15$  với  $1 \leq t \leq 14$ .

Vì vậy  $s$  là một ***pn-sequence***.