Cải tiến mã khối hạng nhẹ họ LED và Neokeon

Lê Phê Đô
Khoa Công nghệ thông tin
Đại học Công nghệ - Đại học Quốc gia Hà nội
Email: dolp.cntt@gmail.com

Lê Trung Thực
Khoa Công nghệ thông tin
Đại học Công nghệ - Đại học Quốc gia Hà nội
Email: thuclt12a@gmail.com

Mai Mạnh Trừng
Khoa Công nghệ thông tin
Đại học Kinh tế - Kỹ thuật Công nghiệp Hà nội
Email: mmtrung@ioit.ac.vn

Lê Thị Len
Khoa Công nghệ thông tin
Đại học Công nghệ - Đại học Quốc gia Hà nội
Email: phalebl@gmail.com

Nguyễn Khắc Hưng
Khoa Công nghệ thông tin
Đại học Công nghệ - Đại học Quốc gia Hà nội
Email: nguyenhung.tlh@gmail.com

Nguyễn Thị Hằng
Khoa Công nghệ thông tin
Đại học Công nghệ - Đại học Quốc gia Hà nội
Email: themoonit@gmail.com

Trần Văn Mạnh
Khoa Công nghệ thông tin
Đại học Công nghệ - Đại học Quốc gia Hà nội
Email: manhtv@outlook.com

Tóm tắt nội dung*—Họ mã khối Neokeon được đề xuất vào năm 2000 bởi bốn nhà khoa học Daemen, Peeters, Van Assche và Rijme. Môt trong các ưu điểm của Neokeon đó là nhỏ gọn, nhanh chóng triển khai trên phần cứng chuyên dụng, ngoài ra nó còn yêu cầu về bộ nhớ RAM rất thấp và hoạt động hiệu quả trên đa nền tảng. Nhưng trong thuật toán họ Neokeon sử dụng rất nhiều các ánh xạ tuyến tính và sử dụng khóa trực tiếp trong khi mã hóa mà không thông qua những phép biến đổi làm việc trước với khóa. Do đó, Neokeon dường như khá mong manh trước những dạng thám mã tuyến tính. LED là thuật toán được thiết kế và đề xuất bởi Guo, Peyrin, Poschmann và Robshaw vào năm 2011, là một trong những thuật toán mã hóa nhẹ khá mới. Kích thước khối tin 64-bit với hai biến thể của khóa lần lượt là 64-bit và 128-bit. Thiết kế của LED có nhiều điểm tương đồng với thuật toán AES. Trong bài báo này, chúng tôi thực hiện cải tiến thuật toán LED và Neokeon, đối với Neokeon chúng tôi ngẫu nhiên hóa dữ liệu tại những ánh xạ tuyến tính và thao tác làm việc với khóa, nhằm mục đích nâng cao độ an toàn của Neokeon trước các thám mã tuyến tính nhưng không làm thay đổi nhiều những ưu điểm sẵn có của Neokeon, đối với LED chúng tôi thực hiện cải tiến xử lý dữ liệu trong khối step của thuật toán này.

***Keywords*-mã khối hạng nhẹ; led; neokeon; tuyến tính.**

I. GIỚI THIỆU

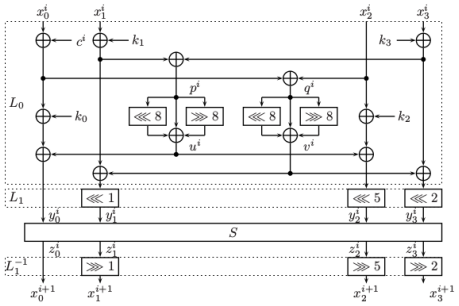
Neokeon [9, 10] là mã khối có kích thước bản tin 128-bits, sử dụng độ dài khóa 128-bits, được thiết kế bởi nhóm bốn nhà khoa học Daemon, Peeters, Van Assche và Rijme thuộc ESAT/COSIC, KU Leuven, Bỉ và đề xuất vào năm 2000. Các tiêu chí khi thiết kế thuật toán Neokeon:

- Chống lại các phân tích mật mã thông thường như vét cạn, thám mã vi sai, lượng sai...
- Tốc độ xử lý nhanh.
- Nhỏ gọn, dễ dàng triển khai.
- Hoạt động tốt trên đa nền tảng vi mạch.
- Thiết kế đối xứng và đơn giản.

Noekeon là thuật toán tự nghịch đảo, nó được coi như tiền thân của các thuật toán mã hóa khối hạng nhẹ hiện đại. Thuật toán gồm có 16 vòng, mỗi vòng sẽ thực hiện các bước mã hóa sau: Theta, Pi1, Gamma, và Pi2. Gamma là ánh xạ phi tuyến tính (S-layer), S-box hoạt động độc lập trên 32 bộ 4-bit. Pi1 và Pi2 là các phép thay đổi thứ tự bit đơn giản theo chu kỳ. Theta là một ánh xạ tuyến tính làm việc với khóa k. Do đó, Theta hoạt động một phần như AddRoundKey cùng với Pi1 và Pi2 tạo thành P-layer của các thuật toán mã hóa. Thủ tục mã hóa sử dụng khóa bằng khóa bí

mật. Với tính chất tự đảo nghịch hệ mật Noekeon có lợi thế lớn khi mã hóa và giải mã cần phải được thực hiện trên cùng một vi mạch. Phía dưới là chi tiết một vòng mã hóa của thuật toán Neokeon.

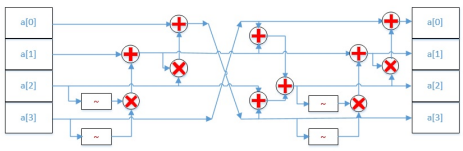
```
Round(Key, State, Constant1,
    Constant2) {
  State[0] ^= Constant1;
  Theta(Key, State);
  State[0] ^= Constant2;
  Pi1(State);
  Gamma(State);
  Pi2(State);
}
```



Hình 1: Tổng quan thuật toán Neokeon

Hàm Gamma phi tuyến tính xử lý các bít đầu vào:

```
Gamma(a){
  a[1] ^= ~a[3]&~a[2];
  a[0] ^= a[2]& a[1];
   tmp = a[3]; a[3] = a[0]; a[0] =
       tmp;
   a[2] ^= a[0]^a[1]^a[3];
   a[1] ^= ~a[3]&~a[2];
   a[0] ^= a[2]& a[1];
}
```



Hình 2: Sơ đồ hoạt động của Gamma

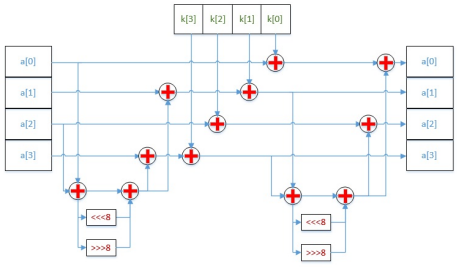
Các phép toán phi tuyến $\sim a[3]\& \sim a[2]$ có thể được thay thế bằng biểu thức tương đương $\sim (a[3] \mid a[2])$.
Gamma hoạt động độc lập trên 32 hộp 4 bít. Các hộp này bao gồm 4 bit trong mỗi đầu vào và có cùng vị trí. Ví dụ, hộp 9 bao gồm bit 9 của a [0], bit 9 của a [1], bit 9 của a [2] và bit 9 của a [3]. Giá trị của hộp 9 có thể được đại diện bởi một số hexa đại diện cho bốn bit theo thứ tự 3, 2, 1, 0.

S-boxes sử dụng trong hàm Gamma của thuật toán neokeon

Input	Output
0	7
1	A
2	2
3	C
4	4
5	8
6	F
7	0
8	5
9	9
A	1
B	E
C	3
D	D
E	B
F	6

Hàm phi tuyến tính Theta làm việc với khóa và các bit đầu vào:

```
Theta(k,a){
  temp = a[0]^a[2]; temp ^= temp>>>8
      ^ temp<<<8;
  a[1] ^= temp;
   a[3] ^= temp;
   a[0] ^= k[0]; a[1] ^= k[1]; a[2]
       ^= k[2]; a[3] ^= k[3];
   temp = a[1]^a[3]; temp ^= temp>>>8
       ^ temp<<<8;
   a[0] ^= temp;
   a[2] ^= temp;
}
```



Hình 3: Sơ đồ hoạt động của Theta

Nghịch đảo của hàm Theta chính là Theta (k', a), trong đó k' = Theta (NullVector, k). Tức là k' được

tạo ra từ chính hàm Theta nhận Nullvector làm khóa và khóa k làm giá trị bit đầu vào. Dễ thấy thiết kế hàm Theta đảm bảo các điều kiện:

- Sử dụng một khóa để mã hóa/ giải mã.
- Độ phức tạp thuật toán nhỏ.
- Khả năng khuếch tán thích hợp.
- Tính đối xứng.
- Tính đơn giản.

Theta hoạt động độc lập trên 8 khối 16 bits, được gọi là các cột. Các cột này bao gồm 4 bộ 4 bit trong các từ trạng thái ở các vị trí bằng modulo 8. Ví dụ, cột 3 gồm các bit 3, 11, 19, 27 của a [0], bit 3, 11, 19, 27 của a [1], bit 3, 11, 19, 27 của a [2] và bit 3, 11, 19, 27 của a [3].

Bảng 1: Bảng phân phối Hamming của hàm Theta

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1							16								
2		8				48				48				16	
3					112				448						
4				348				1152				224			
5			112				3136				1120				
6		48				3928				3984				48	
7	16				3136				7840				448		
8				1152				10556				1152			
9			448				7840				3136				16
10		48				3984				3928				48	
11					1120				3136				112		
12				224				1152				348			
13							448				112				
14		16				48				48				8	
15									16						

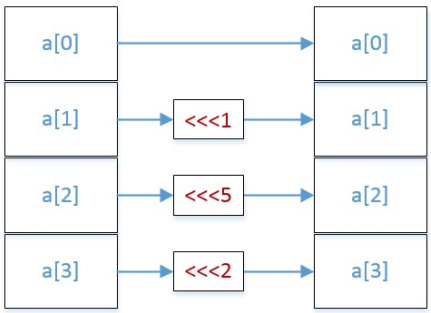
Các hàm Pi1, Pi2 là những hoán đổi bit: $Pi1(a)a[1] <<<= 1; a[2] <<<= 5; a[3] <<<= 2;$
$Pi2(a)a[1] >>>= 1; a[2] >>>= 5; a[3] >>>= 2;$

- Pi2 là nghịch đảo của Pi1: để có thể sử dụng cùng một phép chuyển cả trong mã hóa và giải mã.
- Bốn lần bù của Pi1 khác với modulo 8.
- Độ lệch của từ 0 là 0.
- Mảng bù đắp được lựa chọn từ bộ các mảng tối ưu hóa qua sự khuếch tán.

Mô phỏng tổng quan Neokeon khi thực hiện đầy đủ 16 vòng mã hóa:

```
Noekeon(WorkingKey, State) {
     For( i=0 ; i<Nr ; i++)
          Round(WorkingKey, State,
          Roundct[i],0);
     State[0] ^= Roundct[Nr];
     Theta(WorkingKey, State);
}
```

Trong đó, Roundct[i] được biểu diễn bởi



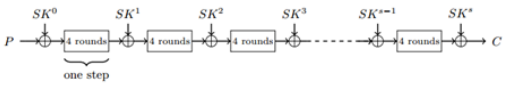
Hình 4: Sơ đồ hoán đổi, xáo trộn thứ tự bit

```
RC[0] = 0x80;
    if (RC[i]&0x80 != 0) RC[i+1] =
        RC[i]<<1 ^ 0x1B
    else RC[i+1] = RC[i]<<1;
```

Hàm tự nghịch đảo Neokeon:

```
InverseNoekeon(WorkingKey,State) {
    Theta(NullVector, WorkingKey);
    For( i=Nr ; i>0 ; i--)
        Round(WorkingKey,State,0,Roundct[i]);
    Theta(WorkingKey, State);
    State[0] ^= Roundct[0];
}
```

LED [11] được thiết kế dựa trên thuật toán AES đây là điểm xuất phát lý tưởng cho một thiết kế hạng nhẹ và an toàn. Thiết kế LED có nhiều điểm tương đồng các tính năng như S-boxes, ShiftRows, MixColumns. Chúng ta hãy theo dõi hình 1 dưới đây để xem quá trình mã hóa hoặc giải mã của LED.



Hình 5: Quy trình thực hiện LED

```
for i =0 to s -1 do {
   addRoundKey(STATE,SK[i]);
   step(STATE);
}
addRoundKey(STATE,SK[s]);
```

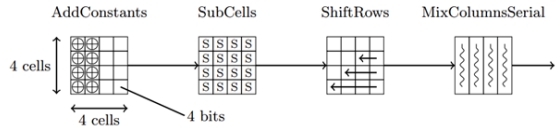
Trong việc giải mã, LED thực hiện việc đảo ngược các hàm và các tham số đầu vào thông qua khối step như hình 2 bên trên. Bây giờ, chúng ta sẽ

tìm hiểu từng cấu tử tham gia vào việc mã hóa và giải mã trong LED như khóa, hàm $addRoundKey$, và từng thành phần của khối step. Biểu diễn khóa bởi ma trận vuông cấp 4, 64 bit. Với 128 bit biểu diễn bằng 2 ma trận vuông cấp 4.

Hàm $addRoundKey(STATE, SK_i)$ là sự kết hợp từng thành phần trong khóa là SK^i với state thông qua toán tử XOR:

```
addRoundKey(state[4][4], keyBytes,
    step)
{
   for(i = 0; i < 4; i++)
      for(j = 0; j < 4; j++)
         state[i][j] ^=
             keyBytes[(4*i+j+step*16)
     \%(LED/4)];
}
```

LED = 64 hoặc 128 tùy thuộc vào việc chúng ta cài đặt LED-64 hay LED-128 Các thành phần trong khối step ở khâu mã hóa lần lượt là: $addConstants, subCell, shiftRow, mixColumn$. Việc giải mã ta chỉ cần đảo ngược lại chức năng và các tham số đầu vào của các thành phần này. Khối này được biểu diễn trong hình 3 phía bên dưới:



Hình 6: Quy trình thực hiện LED

Hàm $AddConstans$: Một vòng hằng số được định nghĩa như sau. Tại mỗi vòng, 6 bit $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ được dịch một vị trí sang trái với giá trị mới là rc_0 được tính bằng $rc_5 \bigoplus rc_4 \bigoplus 1$. Trong vòng lặp tiếp theo, sáu bit này được gán lại giá trị là 0 và được cập trước khi sử dụng. Hằng số, được sử dụng trong một vòng trong mảng.

```
sbox[16] = {12, 5, 6, 11, 9, 0, 10,
    13, 3, 14, 15, 8, 4, 7, 1, 2};
SubCell(state[4][4])
{
  for(i = 0; i < 4; i++)
    for(j = 0; j < 4; j++)
      state[i][j] = sbox[state[i][j]];
}
```

Hàm $SubCells$: Mỗi một thành phần trong mảng STATE được thay thế bởi một phần tử được sinh sau khi dùng S-box của hệ mật Present.

Hàm $ShiftRow$: Hàng thứ i của mảng STATE được dịch sang trái i lần với giá trị i lần lượt nhận là: 0, 1, 2, 3.

$MixColumns$: Mỗi cột của mảng STATE được hiển thị như là một véc-tơ cột và thay thế bởi véc-tơ cột. Số lượng khối lặp step trong suốt quá trình mã hóa hoặc giải mã phụ thuộc vào kích thước khóa. Đối với 64 bits khóa thì số vòng là 8, còn đối với 128 bit khóa thì số vòng là 12. Sau khi thực hiện tuần tự đủ số vòng và các thành phần bên trong vòng lặp ấy ta nhận được bản mã hoặc bản giải mã tương ứng.

II. THÁM MÃ

Ngày nay đã xuất hiện những cấu trúc mới cho mã khối như cấu trúc Matsui, cấu trúc bảng vuông... và sự phối ghép hỗn hợp các cấu trúc đó. Kể từ khi xuất hiện, các mã khối hạng nhẹ đã được các nhà mật mã phân tích đánh giá độ an toàn thông qua các tấn công cụ thể. Nổi tiếng nhất là hai tấn công: tấn công vi sai (do Biham và Shamir đề xuất 1991) và tấn công tuyến tính của Matsui (1993). Có thể tóm tắt ý tưởng của hai phương pháp tấn công này như sau.

Thám mã vi sai dựa trên yếu tố là hàm vòng f trong một mã khối lặp là một hàm yếu về mật mã. Cụ thể là nếu cặp bản mã là được biết và bằng cách nào đó có thể đạt được vi sai của cặp đầu vào tại vòng cuối cùng của mã khối lặp đó, thì tấn công vi sai có thể áp dụng để đạt được hoặc xác định được khoá hay một phần khoá con tại vòng cuối cùng. Trong thám mã vi sai, điều đó được thực hiện bằng cách chọn cặp bản rõ (X, X*) có vi sai là a sao cho vi sai DY(r -1) của cặp đầu vào tại vòng cuối cùng sẽ lấy giá trị cụ thể b với một xác suất cao. Vì có xác suất cao nên các khóa con tại vòng cuối được giải từ bộ ba (DY(r -1), Y (r), Y*(r)) sẽ thường tập trung vào một số phần tử có khả năng nhất. Từ các phần tử xuất hiện nhiều nhất, thám mã sẽ quyết định để tìm ra khóa con đúng tại vòng cuối cùng. Từ khóa con của vòng cuối này, có thể xác định được lại khóa bí mật ban đầu (nếu lược đồ tạo là đơn giản).

Còn mục đích của phương pháp thám mã tuyến tính trên một mã khối là tìm một biểu diễn xấp xỉ tuyến tính cho hệ này để có thể phá chúng nhanh hơn phương pháp tấn công vét cạn. Cụ thể để thực hiện thám mã tuyến tính cần phải tìm các biểu diễn tuyến tính hiệu quả có dạng P[i1, i2,.., ia] r C[j1, j2,.., jb] = K[k1, k2,.., kc], trong đó i1, i2,.., ia, j1,

j2,.., jb và k1, k2,.., kc là ký hiệu các vị trí bit cố định, và phương trình đúng với xác suất p 7 1/2 với giả thiết bản rõ P được lấy ngẫu nhiên còn C là bản mã tương ứng với khoá K cố định cho trước nào đó. Giá trị tuyệt đối |p-1/2| được xem như độ hiệu quả của phương trình trên. Nếu có thể thành công trong việc tìm một biểu diễn tuyến tính hiệu quả, thì khi đó có thể sử dụng nó để tìm ra được bit dạng khoá quan trọng K[k1, k2,..,kc] dựa trên phương pháp hợp lý cực đại.

Ngoài một số phương pháp truyền thống trên, ngày nay, nhờ tốc độ máy tính đã được cải thiện đáng kể, trong những bài toán mà không gian khoá không quá lớn người ta còn có thể áp dụng một phương pháp nữa đó là "vét cạn". Đối với không gian khoá lớn, đây thật sự là phương pháp kém hiệu quả nếu chúng ta chỉ thực hiện vét cạn một cách thông thường. Tuy nhiên nếu áp dụng đồng thời các kỹ thuật bổ trợ thì nó vẫn phát huy được hiệu quả tốt. Các kỹ thuật hỗ trợ được nói tới ở đây là xây dựng một thư viện phục vụ việc "vét cạn" bao gồm cơ sở dữ liệu về khoá và các tiêu chuẩn bản rõ tốt. Trên cơ sở đó tìm cách phân hoạch không gian khoá S thành hai tập con rời nhau là S1 và S2 sao cho khoá đúng sẽ "chắc chắn" thuộc một trong hai tập con đó. Từ đó tiến hành sử dụng thuật toán vét cạn trên tập con có chứa khoá đúng, khi đó việc "vét cạn" trong tập con nhanh chóng thể hiện tính hiệu lực của nó. Việc này cũng có thể thực hiện ngay đối với một số phương pháp truyền thống đã có được những kết quả đáng ngạc nhiên. Khi thám ra bản rõ ta chỉ cần đọc được lỗ chỗ đã là thành công vì lúc đó bằng quy luật ngôn ngữ ta sẽ khôi phục được bản rõ gốc như mong muốn.

Ngày nay, người ta đã có những công cụ tính toán cực nhanh nhờ công nghệ cluster. Từ đó người ta có thể xây dựng mạng tính toán song song với tốc độ tính toán đạt tới gần 100GF (một trăm tỷ phép tính dấu phảy động trên một giây). Như vậy người ta có phân rã bài toán để thực hiện việc tính toán song song cực kỳ có hiệu quả, đặc biệt đối với những bài toán có độ phức tạp tính toán lớn. Có thể nói rằng khá nhiều các mã khối trong đó có Neokeon đã bị phá thực tế bởi hai tấn công trên đây. Vượt lên trên các tấn công phân tích mã, lần lượt các hệ mã khối mới với ưu thế mới đã ra đời đáp ứng có độ an toàn chống lại được các tấn công thực tế và hiệu quả trong sử dụng. Tiêu chuẩn mã dữ liệu tiên tiến AES nhằm thay cho DES là một ví dụ. Nhìn chung từ các tấn công phân tích mã, ta có thể rút ra các đặc trưng an toàn cơ bản của một hệ mã khối.

III. CƠ SỞ CẢI TIẾN THUẬT TOÁN

Việc cải tiến nâng cao độ an toàn là cần thiết trước các tấn công thám mã. Chúng tôi đưa ra các nguyên tắc để cải tiến một hệ mã khối hạng nhẹ bao gồm:

Hệ mã phải có độ dài khối rõ, khối khoá đủ lớn [8] (không gian rõ và khoá lớn) để tránh tấn công vét cạn trên không gian rõ cũng như không gian khoá (thường độ dài cỡ khối lớn hơn hoặc bằng 128). Hệ mã phải có độ đo vi sai và độ đo độ lệch tuyến tính tối thiểu để tránh được hai kiểu tấn công nguy hiểm nhất là tấn công vi sai và tấn công tuyến tính theo các nguyên lý như đã trình bày trên. Các hộp thế, các phép biến đổi phi tuyến cần phải có bậc đại số cao để tránh tấn công nội suy, tấn công vi sai bậc cao. Tầng tuyến tính trong các hàm vòng cần phải được lựa chọn cẩn thận để khi phối hợp với tầng phi tuyến phải tạo ra hệ mã có tính khuyếch tán tốt để tránh các tấn công cục bộ trên các khối mã nhỏ. Lược đồ tạo khoá cần phải tránh được các lớp khoá yếu, không được tồn tại những quan hệ khoá đơn giản do tính đều, hay cân xứng trong lược đồ gây nên để tránh các kiểu tấn công khoá quan hệ, tấn công trượt khối dựa trên tính giống nhau trong các phân đoạn tạo khoá con.

Về phần ngẫu nhiên hóa dữ liệu [4]: Như đã nói, một mã khối hiện nay thường được thiết lập bởi cách lặp một số lần một hàm vòng nào đó. Các hàm vòng hiện nay thường được xây dựng theo cấu trúc thay thế - hoán vị, gồm các hộp thế cỡ 8x8 bit kết hợp với các phép hoán vị khối theo byte, các phép dịch vòng, các phép XOR giữa khóa và dữ liệu hoặc các phép toán cộng modulo. Giữa các tầng hộp thế trong hàm vòng thường sử dụng các phép biến đổi tuyến tính theo byte được lựa chọn cẩn thận để đảm bảo độ khuyếch tán tối ưu cho các hàm vòng. Như vậy có thể nói rằng, có ba yếu tố cơ bản liên quan đến phần ngẫu nhiên hóa dữ liệu theo thứ tự "từ trong ra ngoài" đó là: hộp thế phi tuyến, hàm vòng và cấu trúc mã/dịch. Rõ ràng là cần phải nghiên cứu các độ đo an toàn đối với các hộp thế, hàm vòng và thác triển ra toàn bộ cấu trúc mã khối. Để xây dựng hệ mã khối chống được các tấn công vi sai hay tuyến tính ta có thể chồng chất truy hồi các cấu trúc, nhưng điều bất lợi ở đây là hộp thế cố định cơ sở lại phải có kích cỡ quá lớn sẽ phi thực tế trong sử dụng. Cũng từ lý do đó, Knudsen đã đưa ra khái niệm và cách xây dựng các hệ mã khối an toàn thực tế. Trên quan điểm đó, người ta cần quan tâm đến các hộp thế nhỏ có các độ đo tốt, đến số lượng các hộp thế tích cực tương ứng

với các khái niệm lan truyền vi sai hay tuyến tính, và đến số lượng vòng lặp. Dựa trên quan điểm này người ta đã xây dựng được các hệ mã khối an toàn thực tế và hiệu quả. Giả sử rằng chỉ có cách tấn công mã Feistel bằng cách tìm các đặc trưng vi sai (hay đặc trưng tuyến tính) tốt nhất, tức là rất khó có thể tìm được các vi sai (hay xấp xỉ tuyến tính), thì các số liệu xác suất của đặc trưng vi sai (tuyến tính) 1 vòng không tầm thường tốt nhất và số vòng trong đặc trưng đó sẽ cho ta tính được cận dưới độ phức tạp của hai tấn công này. Cận dưới này có thể đủ để khẳng định độ an toàn thực tế của hệ mã, nếu như xác suất của đặc trưng vi sai (tuyến tính) 1 vòng không tầm thường tốt nhất là đủ nhỏ. Bằng cách sử dụng các hộp thế có các độ đo tốt ta có thể thiết lập các hàm vòng có các đặc trưng đủ tốt để từ đó thiết kế hệ mã khối chống được hai tấn công cơ bản hiện nay.

K.Nyberg đã đề xuất các phương pháp xây dựng các hộp thế dựa trên các phép lũy thừa cũng như phép nghịch đảo trên trường hữu hạn, các hộp thế này đã được ứng dụng thiết kế các hệ mã khối ngày nay như AES [7]... Một số nhà nghiên cứu mật mã cũng đã đưa ra cách xây dựng các hàm vòng có tác dụng khuyếch tán nhanh các độ đo vi sai hay độ đo độ lệch tuyến tính, như hàm vòng 2 tầng hộp thế (trong hệ mã khối E2 - Nhật), hàm vòng 3 tầng hộp thế (trong hệ mã khối SEED của Hàn Quốc)... Đặc biệt, J. Daemen và V. Rijnmen đã đưa ra cách xây dựng hàm vòng theo kiểu khuyếch tán hai chiều (dạng cấu trúc bảng vuông) và với cách lựa chọn cẩn thận các phép toán tuyến tính đảm bảo tính khuyếch tán rất tốt, làm cho hệ mã có thể đạt độ an toàn thực tế rất nhanh (tức sau số vòng không quá lớn như đối với DES hay GOST...).

Về phần lược đồ khóa [5, 6]: Các lược đồ khoá hiện tại có thể được chia thành hai kiểu. Ở kiểu 1, tri thức về một khoá con vòng sẽ cung cấp một cách duy nhất các bit khoá của các khoá con vòng khác hay của khoá chính. Còn ở kiểu 2, tri thức về một khóa con vòng bất kỳ đều không cho ta biết thêm bất kỳ bit nào về các khóa con vòng khác hay khóa chính. Các tác giả G. Carter, E. Dawson, và L. Nielsen đã đề xuất một dạng lược đồ khoá kiểu 2 như sau: Trước hết giả thiết tồn tại một hàm một chiều mạnh OWF. Giả sử MK là khoá chính của một hệ mã khối r- vòng, quá trình tạo các khoá con vòng như sau:

- Bước 1: OWF(MK) = Khoá con vòng (1)
- Bước 2: Với i =1 đến r-1

Thực hiện hoán vị bit MK để tạo ra MKi OWF(MKi) = Khoá con vòng (i+1). Chú ý rằng trong thuật toán trên là mỗi một khoá con vòng đều được tạo bởi toàn bộ các bit của khoá chính MK. Điều này có thể tạo ra hiệu ứng thác, tức là sự thay đổi một bit của khoá chính sẽ tạo ra sự thay đổi nhiều ở trong mỗi khoá con. Lược đồ trên nói chung khó áp dụng thực tế do phải sử dụng hàm một chiều. Hiện nay, để vận dụng ý tưởng này người ta thường thay OWF bởi phần cốt lõi phi tuyến của hàm vòng tương ứng trong phần ngẫu nhiên hóa dữ liệu cộng thêm một số phép toán điều khiển để vừa đảm bảo tính an toàn, vừa đảm bảo tính hiệu quả, tránh các tấn công khóa quan hệ, đồng thời cũng đảm đảm bảo sự gọn nhẹ của hệ mã khối đang thiết kế. Lược đồ khoá như thế sẽ là tốt theo nghĩa không chứa các khoá quan hệ và tri thức về một khoá con vòng nào đó không giúp gì trong xác định các thông tin về khoá con vòng khác cũng như khoá chính K.

Một số chú ý cần quan tâm: Để đảm bảo cho tính khuyếch tán dữ liệu hoàn toàn trong phần ngẫu nhiên hoá dữ liệu của một hệ mã khối, vấn đề xây dựng và lựa chọn hoán vị sử dụng trong các hàm vòng là hết sức quan trọng. Trên các tài liệu đã công khai, chúng tôi chưa thấy có tài liệu nào cho những chỉ dẫn cụ thể về vấn đề này. Tất nhiên, các hoán vị lựa chọn ở đây phải đảm bảo một số tính chất nào đó, như có khoảng cách trung bình là lớn, hay có khoảng cách đều hoặc khoảng cách ngẫu nhiên. Phép dịch bit trong chuẩn mã dữ liệu GOST của Nga có thể xem như một hoán vị có khoảng cách đều nhưng nhỏ. Do đó, theo một nghĩa nào đó, sử dụng hoán vị này cũng sẽ cho độ khuyếch tán tốt nhưng chậm, tức là phải với số vòng khá lớn. Hoán vị trong DES có khoảng cách trung bình khá lớn nhưng lại là hoán vị theo bit nên tốc độ chậm, ảnh hưởng trong thiết kế phần cứng. Có thể sử dụng hoán vị ngẫu nhiên hoặc giả ngẫu nhiên... có khoảng cách lớn. Tuy nhiên các lựa chọn cụ thể cần phải được tính toán cẩn thận để đảm bảo tối ưu yêu cầu khuyếch tán dữ liệu nhanh, đồng thời không ảnh hưởng lớn đến tốc độ của hệ mã khối.

Đối với việc lựa chọn các phép biến đổi đầu vào đầu ra cho một hệ mã khối kiểu Feistel, hay bất kỳ hệ mã khối nào có sử dụng các hàm vòng lặp, người thiết kế thuật toán cũng cần phải hết sức chú ý. Chẳng hạn DES [7] sử dụng các hoán vị cố định cho trước, đối với thám mã hoán vị này hoàn toàn không có ý nghĩa gì cả, nhưng nếu chọn một phép biến đổi nào đó phụ thuộc khóa, thì đương nhiên lược đồ khóa cần phải tính toán ra khóa đó và có nghĩa phần thiết kế lược đồ khóa cần phải đảm bảo

an toàn cho cả khóa con vòng cùng với khóa cho phép biến đổi đầu vào - đầu ra. Đây cũng là một yêu cầu không đơn giản. Một chú ý nữa là các phép biến đổi đầu vào - đầu ra trong hệ Feistel phải đảm bảo tính đối xứng thuận nghịch, khi đó mới không ảnh hưởng tới qui trình mã dịch. Có một số hệ mã khối đã đề cập đến vấn đề này như trong hệ E2, hệ Rijndael, Twofish... Tất nhiên, khi lựa chọn các phép biến đổi đầu vào đầu ra cần phải thiết kế ngay lược đồ khoá tương ứng đảm bảo tránh các tấn công đã có liên quan đến lược đồ khóa.

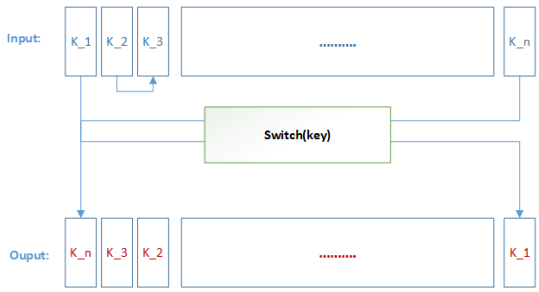
Lưu ý là số lượng các vòng lặp của mã khối cần cân đối giữa tốc độ mã dịch và độ đo độ an toàn của hệ mã. Ngoài việc dựa vào các căn cứ lý thuyết cần có các căn cứ thống kê số liệu thực hành về các độ đo an toàn đã nêu để lựa chọn số vòng lặp cụ thể.

IV. ĐỀ XUẤT CẢI TIẾN

A. Noekeon cải tiến

Đối với Noekeon cải tiến, chúng tôi sử dụng thêm 1 hàm $Switch(key)$ để làm việc với khóa 128bits trước khi sử dụng hàm $Theta$, nhằm mục đích xáo trộn dữ liệu khóa K, tăng độ mật cho thuật toán. Đảm bảo cho dù thám mã ra giá trị khóa K cũng sẽ không thám mã được nội dung bản rõ một cách dễ dàng. Chi tiết hàm $Switch(key)$ như sau:

- **Input:** Khóa K có độ dài 128 bits.
- **Xử lý:** Đổi chỗ vị trí bit thứ 1 và vị trí thứ n của khóa. Từ vị trí thứ 2, thực hiện đổi chỗ của cặp bit thứ (i;i+1).
- **Output:** Khóa K' có độ dài 128 bits.



Hình 7: Sơ đồ thuật toán Switch(key)

Như vậy, đối với hàm $Switch(key)$ có thể sử dụng trong cả mã hóa và giải mã mà không cần thay đổi. Giá trị output K' sẽ được sử dụng tại Noekeon cải tiến:

```
Round(Key, State, Constant1,
    Constant2) {
    State[0] ^= Constant1;
    Theta(Switch(Key), State);
    State[0] ^= Constant2;
    Pi1(State);
    Gamma(State);
    Pi2(State);
}
```

B. LED cải tiến

S-Box và vị trí của từng thành phần trong S-Box:

Pos		S[x]	x	
0	12	C	0	
1		5	1	
2		6	2	
3	11	B	3	
4		9	4	
5		0	5	
6	10	A	6	
7	13	D	7	
8		3	8	
9	14	E	9	
10	15	F	A	10
11		8	B	11
12		4	C	12
13		7	D	13
14		1	E	14
15		2	F	15

LED là mã khối nhẹ có độ an toàn cao, do đó ta có thể tiến hành cải tiến LED để áp dụng cho các thiết bị IoT có giới hạn lưu trữ và xử lý ở tầm trung cho đến tầm cao. Và những thiết bị đó yêu cầu một thuật toán mã hóa có độ bảo mật tốt. Để cải tiến LED ta nên đi theo hướng bù đắp các yếu điểm của thuật toán mã hóa khối. Có 3 hướng chính là tăng kích thước khối hoặc khóa, tăng độ hỗn loạn, tăng tính khuếch tán. Hiện tại độ dài khối và khóa của LED đang ở mức rất tốt là 64bit và 128bit do đó ta không nên phá vỡ cấu trúc này.

Ở bài báo này, chúng tôi lựa chọn phương pháp cải tiến là tăng độ hỗn loạn và rắc rối để kẻ tấn công từ bỏ việc dịch ngược đoạn mã. Vị trí cải tiến là S-Box và phương pháp cải tiến là áp dụng công thức của mã Ceasar trong việc thay thế từng thành phần trong S-Box. Áp dụng vào S-Box ta có công thức mã hóa:

$$Pos' = (Pos + k) mod 16.$$

Trong đó Pos là vị trí hiện tại của từng thành phần trong S-Box được đánh số như hình 1. Pos' là vị trí thay thế. Sau khi có Pos' ta chuyển đổi trả lại về mã Hecxa. k là số đơn vị dịch chuyển để thay thế. Tương tự như vậy, ta có công thức giải mã:

$$Pos' = (Pos - k) mod 16.$$

Mặt khác LED sử dụng S-Box của present và cố định nên ta cũng nên cố định k khi thiết kế. Mục đích tăng sự phức tạp khi có kẻ tấn công nhưng đồng thời không làm phức tạp quá trình giải mã. Ở đây bài báo chọn k bằng 10 theo tỉ lệ vàng 16/10 của Toán học. Như vậy thu được Pos, Pos' và S-Box' sau khi thực hiện quá trình gây nhiễu theo công thức Ceasar:

	S[x]'	**Pos'**	**Pos**
F	15	10	0
	8	11	1
	4	12	2
	7	13	3
	1	14	4
	2	15	5
C	12	0	6
	5	1	7
	6	2	8
B	11	3	9
	9	4	10
	0	5	11
A	10	6	12
D	13	7	13
	3	8	14
E	14	9	15

Với việc sử dụng thay thế Ceasar không làm tăng độ phức tạp của thuật toán nhưng lại gây ra sự nhiễu loạn không nhỏ trong giải mã khiến cho kẻ tấn công từ bỏ ý định giải mã để xem lén thông tin.

V. KẾT LUẬN

Hiện nay trong xu thế của IoT (Internet of Think), các tổ chức quốc tế đều rất quan tâm và chú trọng đến việc xây dựng, lựa chọn và phát hành những mã hạng nhẹ phù hợp. Điều này đã thúc đẩy sự ra đời hàng loạt các hệ mã khối được thiết kế đảm bảo nguyên tắc vừa an toàn, vừa hiệu quả và kinh tế. Có thể kể một số ví dụ như các hệ mã LED, PRESENT, SEA... Tuy nhiên trong quá trình sử dụng cần phải thường xuyên quan tâm nghiên cứu các phương pháp tấn công phân tích mã mới xuất hiện trên thế giới để có thể phòng tránh các điểm yếu tiềm tàng đối với các hệ mã đang sử dụng (vì thỏa mãn tối ưu các yêu cầu an toàn và hiệu quả kinh tế là việc không dễ dàng). Có thể hy vọng rằng, những đóng góp nghiên cứu của chúng tôi vừa đặt ra có thể giải quyết hoặc là cơ sở cho những ý tưởng giải quyết những bài toán về mật mã nhẹ ứng dụng trong IoT.

TÀI LIỆU

[1] G. Leander and A. Poschmann, *In Arithmetic of Finite Fields*, First International Workshop - WAIFI 2007, volume 4547 of Lecture Notes in Computer Science, pages 159 - 176, Springer 2007.

[2] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai. Piccolo, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of Lecture Notes in Computer Science, pages 342 - 357, Springer 2011.

[3] Miroslav Knezevic, Ventzislav Nikov, and Peter Rombouts, *Low Latency Encryption - Is "Lightweight = Light + Wait"*, NXP Semiconductors, Leuven, Belgium 2015.

[4] Miroslav Knezevic, *Lightweight Cryptography: from Smallest to Fastest*, NXP Semiconductors, July 2015.

[5] Nicky Mouha, *The Design Space of Lightweight Cryptography*, Dept. Electrical Engineering-ESAT/COSIC, KU Leuven, Leuven and iMinds, Ghent, Belgium.

[6] Gregor Leander, *Lightweight Block Cipher Design*, ECRYPT II School June 2011.

[7] International standard ISO/IEC 29192, *Information Technology - Security Techniques – Lightweight cryptography*.

[8] Axel York Poschmann, *Lightweight cryptography: cryptographic engineering for a pervasive world*, in Ph. D. Thesis. 2009. Citeseer.

[9] Joan Daemen, Michaël Peeters, Gilles Van Assche, Vincent Rijmen (2000). Nessie Proposal, *Noekeon*, First Open Nessie Workshop.

[10] Z'aba, M. R., Raddum, H., Henricksen, M., & Dawson, E. (2008, January), *Bit-pattern based integral attack*, In Fast Software Encryption (pp. 363-381).

[11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matt Robshaw, *The LED Block Cipher*, Institute for Infocomm Research, Singapore, 2011.