TCVN TIÊU CHUẨN QUỐC GIA

TCVN 11495-3:2016
ISO/IEC 9797-2:2011

CÔNG NGHỆ THÔNG TIN - CÁC KỸ THUẬT AN TOÀN - MÃ XÁC THỰC THÔNG ĐIỆP (MAC) - PHẦN 3: CƠ CHẾ SỬ DỤNG HÀM BĂM PHỔ BIẾN

Information technology - Security techniques - Message Authentication Codes (MACs) - Part 3: Mechanisms using a universal hash-function

HÀ NỘI - 2016

Mục lục **Trang**

Lời nói đầu

TCVN 11495-3:2016 hoàn toàn tương đương với ISO/IEC 9797-3:2011.

TCVN 11495-3:2016 do Tiểu ban kỹ thuật tiêu chuẩn quốc gia TCVN/JTC1/SC 27 *Kỹ thuật an ninh* biên soạn, Tổng cục Tiêu chuẩn Đo lường Chất lượng đề nghị, Bộ Khoa học và Công nghệ công bố.

Bộ tiêu chuẩn TCVN 11495 (ISO/IEC 9797) *Công nghệ thông tin – Các kỹ thuật an toàn – Mã xác thực thông điệp (MAC)* gồm các tiêu chuẩn sau:

- Phần 1: Cơ chế sử dụng mã khối;
- Phần 2: Cơ chế sử dụng hàm băm chuyên dụng;
- Phần 3: Cơ chế sử dụng hàm băm phổ biến;

Lời giới thiệu

Trong môi trường công nghệ thông tin, thường yêu cầu người ta có thể xác thực rằng dữ liệu điện tử đã không bị thay đổi theo một cách trái phép và người ta có thể cung cấp đảm bảo rằng thông điệp đã được khởi tạo bởi một thực thể mà nắm giữ khóa bí mật. Thuật toán Mã Xác thực Thông điệp (MAC – Message Authentication Code) là một cơ chế toàn vẹn dữ liệu thường được sử dụng mà có thể thỏa mãn những yêu cầu này.

Tiêu chuẩn này chỉ ra 4 thuật toán MAC dùng các hàm băm phổ biến: UMAC, Badger, Poly1305-AES và GMAC.

Các cơ chế này có thể được sử dụng như các cơ chế toàn vẹn dữ liệu để xác minh rằng dữ liệu này không bị thay đổi theo một cách trái phép. Chúng cũng có thể được sử dụng như các cơ chế xác thực thông điệp để đảm bảo tin chắc rằng một thông điệp đã được khởi nguồn bởi một thực thể nắm giữ khóa bí mật. Độ mạnh của cơ chế toàn vẹn dữ liệu và cơ chế xác thực thông điệp phụ thuộc vào độ dài (tính theo bit) và độ bí mật của khóa, vào độ dài (tính theo bit) của mã băm được tạo ra bởi hàm băm, vào độ mạnh của hàm băm, vào độ dài (tính theo bit) của MAC, và vào cơ chế cụ thể.

CHÚ THÍCH Khung cơ chế chung để chỉ ra các dịch vụ toàn vẹn được quy định ISO/IEC 10181-6 [7].

Công nghệ thông tin - Các kỹ thuật an toàn - Mã xác thực thông điệp (MAC) - Phần 3: Cơ chế sử dụng hàm băm phổ biến

Information technology - Security techniques - Message Authentication Codes (MACs) - Part 3: Mechanisms using a universal hash-function

1 Phạm vi áp dụng

Tiêu chuẩn này quy định các thuật toán MAC mà sử dụng một khóa bí mật và một hàm băm phổ biến cùng với một kết quả *n*-bit để tính ra MAC có *m*-bit dựa trên các mã khối đã quy định trong ISO/IEC 18033-3 và các mã dòng đã quy định trong ISO/IEC 18033-4.

a) UMAC;

b) Badger;

c) Poly1305-AES;

d) GMAC.

2 Tài liệu viện dẫn

Các tài liệu viện dẫn sau đây rất cần thiết cho việc áp dụng tiêu chuẩn này. Đối với các tài liệu ghi năm công bố thì áp dụng phiên bản được nêu. Đối với các tài liệu không ghi năm công bố thì áp dụng phiên bản mới nhất, bao gồm cả các sửa đổi, bổ sung (nếu có).

TCVN 11495-1 (ISO/IEC 9797-1), *Công nghệ thông tin —Các kỹ thuật an toàn – Các mã xác thực thông điệp (MAC) – Phần 1: Các cơ chế sử dụng mã khối (Information technology – Security techniques – Message Authentication Codes (MACs) – Part 1: Mechanisms using a block cipher)*;

ISO/IEC 18031, *Information technology- Security techniques- Random bit generation (Công nghệ thông tin – Các kỹ thuật an toàn – Sinh bit ngẫu nhiên);*

TCVN 11367-3 (ISO/IEC 18033-3), *Công nghệ thông tin – Các kỹ thuật an toàn – Thuật toán mã hóa – Phần 3: Mã khối (Information technology – Security techniques – Encryption algorithms – Part 3: Block ciphers).*

TCVN 11367-4 (ISO/IEC 18033-4), *Công nghệ thông tin – Các kỹ thuật an toàn – Thuật toán mã hóa – Phần 4: Mã dòng (Information technology- Security techniques- Encryption algorithms- Part 4: Stream ciphers).*

3 Thuật ngữ và định nghĩa

Tiêu chuẩn này áp dụng các thuật ngữ và định nghĩa sau đây.

3.1

chuỗi rỗng (empty string)

chuỗi gồm các ký hiệu có chiều dài là 0.

3.2

khóa (key)

dãy các ký tự mà để điều khiển hoạt động của một biến đổi mật mã.

3.3

nonce (nonce)

số được sử dụng một lần.

3.4

số nguyên tố (prime number)

số nguyên dương lớn hơn 1 mà không có các ước số nguyên khác với 1 và chính nó.

3.5

thẻ (tag)

kết quả của thuật toán MAC, được nối vào thông điệp có thể được mã hóa để cung cấp bảo vệ toàn vẹn.

3.6

hàm băm phổ biến (universal hash-function)

hàm mà ánh xạ các chuỗi bit vào các chuỗi bit có độ dài cố định, được đánh chỉ số bởi một tham số được gọi là khóa, thỏa mãn tính chất rằng đối với tất cả các đầu vào khác nhau, xác suất trên tất cả các khóa mà các đầu ra va chạm là nhỏ.

CHÚ THÍCH Các hàm phổ biến được giới thiệu bởi Carter và Wegman [4], và ứng dụng của chúng trong thuật toán MAC lần đầu được mô tả bởi Wegman và Carter [10].

4 Ký hiệu và thuật ngữ viết tắt

Tiêu chuẩn này sử dụng các ký hiệu và giải thích như sau:

bit(S, n)	Trả về số nguyên 1 nếu bit thứ n của chuỗi S là 1, ngược lại trả về số nguyên 0 (các chỉ số bắt đầu từ 1)
bitlength(S)	Độ dài của một chuỗi S tính theo bit
bitstr2unit(S)	Số nguyên không âm mà biểu diễn nhị phân của nó là chuỗi S. Hình thức hơn, nếu S dài t bit thì bitstr2unit(S) = 2^{t-1}*bit(S,1) + 2^{t-2}*bit(S,2) + … + 2^1*bit(S,t-1) + bit(S,t).

CHÚ THÍCH Các chuỗi bit được coi như big-endian, tức là, bit thứ nhất có nghĩa lớn nhất

blocklen	Độ dài khối của mã khối cơ sở tính theo octet
ceil	Phép toán làm tròn lên, tức là, nếu x là một số thập phân, thì ceil(x) là số nguyên nhỏ nhất n với $n \geq x$.
Enc(K, X)	Phép mã hóa của khối bản rõ X dưới khóa K dùng mã khối Enc
floor	Phép toán làm tròn xuống, tức là, nếu x là một số thập phân, thì floor(x) là số nguyên lớn nhất với $n \leq x$.
H	Giá trị băm
K	Khóa chủ
K_E	Khóa mã hóa
K_H	Khóa băm
keylen	Độ dài khóa của mã khối tính theo octet
$\log_2$	Hàm logarit cơ số 2
M	Thông điệp
MAC	Mã xác thực thông điệp
max	Giá trị lớn nhất trong số những giá trị được đưa ra như đối số
N	nonce (Điều 3.3)
octetlength(S)	Độ dài của chuỗi S tính theo octet (trong đó S được giả thiết là có độ dài theo bit là một bội của 8)
octetstr2unit(S)	Số nguyên không âm được định nghĩa như $S[0]+2^8 * S[1] + 2^{16} * S[2] + \ldots + 2^{8n-8} * S[n-1]$, trong đó n = octetlength(S).

CHÚ THÍCH Các chuỗi octet được coi như litte-endian, tức là, octet đầu tiên có ý nghĩa nhỏ nhất.

prime(n) Số nguyên tố lớn nhất nhỏ hơn so với 2^n, đối với số nguyên dương n bất kỳ

CHÚ THÍCH Các số nguyên tố được sử dụng trong tiêu chuẩn này được liệt kê trong Bảng 1.

Bảng 1 – Các số nguyên tố

n	prime(n)	prime(n) ở dạng hexa
32	$2^{32} - 5$	0x FFFFFFFB
36	$2^{36} - 5$	0x 0000000F FFFFFFFB
64	$2^{64} - 59$	0x FFFFFFFF FFFFFFC5
128	$2^{128} - 159$	0x FFFFFFFF FFFFFFFF FFFFFFFF FFFFFF61
130	$2^{130} - 5$	0x 00000003 FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFB

$S[i]$	octet thứ i của chuỗi S (các chỉ số bắt đầu từ 0)

CHÚ THÍCH Đặc tả của UMAC trong Điều 6.2 sử dụng chỉ số bắt đầu từ 1 thay cho 0.

$S[i...j]$	Chuỗi con của S bao gồm các octet từ i đến j.
taglen	Độ dài của thẻ, tính theo octet
uint2bitstr(x, n)	Chuỗi n-octet S sao cho bitstr2uint(S) = x.

uint2octetstr(x, n)	Chuỗi n-octet S sao cho x = octetstr2uint(S).
$X\|_s$	Cắt ngắn về bên trái của khối bit X: nếu X có độ dài lớn hơn hoặc bằng s, thì $X\|_s$ là khối s-bit bao gồm s bit bên trái nhất của X.
$X\|^s$	Cắt ngắn về bên phải của khối bit X: nếu X có độ dài lớn hơn hoặc bằng s, thì $X\|^s$ là khối s-bit bao gồm s bit bên phải nhất của X.
$X >> 1$	Dịch phải của khối bit X đi 1 vị trí: bit bên trái nhất của $Y = X >> 1$ luôn được đặt bằng 0.
$\|X\|$	Độ dài của X tính theo bit.
zeropad(S, n)	Đối với số nguyên dương n, chuỗi S được đệm bằng các bit 0 thành bội dương gần nhất của n octet. Một cách hình thức, zeropad(S, n) = $S \|\| T$, trong đó T là chuỗi ngắn nhất của các bit 0 (có thể rỗng) để mà $S \|\| T$ là không rỗng và n chia hết octetlength($S \|\| T$).
$\oplus$	Phép toán XOR theo từng bit trên các chuỗi bit. Nếu A, B là các chuỗi có cùng độ dài thì $A \oplus B$ là chuỗi bằng với XOR logic theo từng bit của A và B.
$\wedge$	Phép toán AND logic theo từng bit trên các chuỗi bit. Nếu A, B là các chuỗi có cùng độ dài thì $A \wedge B$ là chuỗi bằng với AND logic theo từng bit của A và B.
$+_{32}$	Phép cộng của 2 chuỗi 32-bit, mang lại một chuỗi 32-bit. Một cách hình thức hơn, $S +_{32} T$ = unit2bitstr(bitstr2unit(S) + bitstr2unit(T) mod 2^{32}, 4).
$+_{64}$	Phép cộng của 2 chuỗi 64-bit, mang lại một chuỗi 64-bit. Một cách hình thức hơn, $S +_{64} T$ = unit2bitstr(bitstr2unit(S) + bitstr2unit(T) mod 2^{64}, 8).
*	Phép toán nhân trên các số nguyên.
$*_{64}$	Phép nhân của 2 chuỗi 64-bit, mang lại một chuỗi 64-bit. Một cách hình thức hơn, $S *_{64} T$ = unit2bitstr(bitstr2unit(S) * bitstr2unit(T) mod 2^{64}, 8).

CHÚ THÍCH Các phép toán $+_{32}$, $+_{64}$ và $*_{64}$ thích hợp với các phép tính cộng và nhân mà được thực hiện một cách hiệu quả bởi các máy tính hiện đại.

$\|\|$	Phép ghép của hai chuỗi bit. Nếu A và B là các chuỗi bit có độ dài a và b tương ứng, thì $A \|\| B$ là chuỗi bit có độ dài $a + b$, a bit bên trái nhất (đầu tiên) của nó là các bit của A, và b bit bên phải nhất (cuối cùng) của nó là các bit của B.
0^n	Chuỗi bao gồm n bit 0.
1^n	Chuỗi bao gồm n bit 1.
{ }	Chuỗi bit với độ dài 0.
•	Phép nhân trong trường GF(2^{128}). Đa thức xác định mà định nghĩa phép biểu diễn của GF(2^{128}) là $1 + \alpha + \alpha^2 + \alpha^7 + \alpha^{128}$.

CHÚ THÍCH Giả sử U và V là các khối 128-bit. Khi đó khối 128-bit $W = U \bullet V$ có thể được tính như sau:

a) Đặt $W = 0^{128}$ và $Z = U$.

b) For i = 1, 2, ..., 128, thực hiện 2 bước sau:

1) Nếu bit(V, i) = 1 thì lấy $W = W \oplus Z$;

2) Nếu bit(Z, 128) = 0 thì lấy $Z = Z >> 1$; ngược lại lấy $Z = (Z >> 1) \oplus (11100001 \|\| 0^{120})$.

Các biến là các chữ cái viết hoa thì ký hiệu các chuỗi; các biến là các chữ cái viết thường thì ký hiệu các số nguyên.

5 Mô hình chung

Các mã xác thực thông điệp dựa trên hàm băm phổ biến sử dụng một thuật toán mã hóa (mã khối hoặc mã dòng). Các mã xác thực thông điệp kiểu này có tính chất đặc biệt là độ an toàn của chúng có thể là chứng minh được dưới giả thiết rằng thuật toán mã hóa là an toàn.

Các thuật toán MAC dựa trên hàm băm phổ biến yêu cầu một khóa chủ K, thông điệp M và giá trị nonce N như đầu vào. MAC được tính bằng cách sử dụng dãy các bước sau:

1) Tiền xử lý khóa. Khóa chủ K được sử dụng để sinh ra khóa băm K_H và khóa mã hóa K_E.
2) Tiền xử lý thông điệp. Thông điệp đầu vào M được mã thành khuôn dạng đầu vào cần thiết cho hàm băm.
3) Băm thông điệp. Thông điệp đã được mã được băm dưới điều khiển của khóa băm K_H dùng một hàm băm phổ biến. Kết quả là một giá trị băm H có độ dài ngắn, cố định.
4) Hoàn tất. Giá trị băm H được mã hóa dưới điều khiển của khóa mã hóa K_E. Kết quả là mã xác thực thông điệp MAC.

Đối với tất cả các cơ chế đã được trình bày trong tiêu chuẩn này, độ dài của thông điệp đầu vào được mong đợi là một số nguyên dạng octet.

CHÚ THÍCH Đối với tất cả các thuật toán MAC dựa trên băm phổ biến, cực kỳ quan trọng rằng một nonce khác sẽ được sử dụng cho mỗi thông điệp mới mà được xác thực dưới cùng một khóa. Nếu yêu cầu an toàn này không được thoả mãn, thì độ an toàn của thuật toán bị suy giảm nghiêm trọng.

6 Các cơ chế

6.1 Giới thiệu

Điều này quy định bốn cơ chế sử dụng một hàm băm phổ biến.

6.2 UMAC

6.2.1 Mô tả của UMAC

UMAC là một họ có 4 thuật toán MAC được tối ưu hóa cho 4 độ dài bit đầu ra khác nhau, được ký hiệu bởi UMAC-32, UMAC-64, UMAC-96 và UMAC-128. UMAC có thể được sử dụng cùng với mã khối bất kỳ từ ISO/IEC 18033-3. Nếu mã khối được sử dụng có độ dài khóa $|K|$ bit và độ dài khối $|B|$ bit, thì UMAC sử dụng một khóa $|K|$-bit K, và độ dài của nonce N là ở giữa 8 và $|B|$ bit. Phụ thuộc vào thành viên nào của họ UMAC được sử dụng, độ dài của MAC được tạo ra là 32, 64, 96 hoặc 128 bit. Nó được biểu diễn bởi tham số *taglen*, và có thể là 4, 8, 12 hay 16 octet, tương ứng. Độ dài của thông điệp đầu vào phải nhỏ hơn 2^{67} octet. Thông điệp đầu vào cho hàm UMAC phải chứa một số trọn vẹn các octet, tức là, độ dài tính theo bit của nó sẽ là một bội của 8. Nếu độ dài tính theo bit không là bội của 8, cơ chế này không được sử dụng.

CHÚ THÍCH 1 Phiên bản của UMAC được chỉ ra ở đây cần không được lẫn với các phiên bản sớm hơn của thuật toán UMAC, ví dụ [2].

CHÚ THÍCH 2 Nếu đầu vào của hàm MAC chứa một số trọn vẹn các byte, thì hàm được chỉ ra ở đây là giống với hàm đã được mô tả trong RFC 4418 [6].

6.2.2 Các yêu cầu

Trước khi sử dụng UMAC, các tham số sau cần phải được thống nhất:

- Một mã khối đã được chuẩn hóa trong ISO/IEC 18033-3. Việc lựa chọn của mã khối xác định độ dài khóa $|K|$ và độ dài khối $|B|$;
- Độ dài thẻ, *taglen*, nó cần phải là 4, 8, 12 hoặc 16 octet;
- Độ dài của nonce, nó cần phải nằm giữa 8 và $|B|$ bit.

6.2.3 Chú giải và các hàm bổ trợ

6.2.3.1 Các phép toán trên các chuỗi

Ngược lại với phần còn lại của tiêu chuẩn này, đặc tả của UMAC sử dụng bắt đầu của chỉ số từ 1 khi đánh số các phần tử trong một dãy. Do vậy, đối với UMAC, $S[i]$ ký hiệu octet thứ i của chuỗi S, trong đó $i \geq 1$.

6.2.3.2 Hàm bổ trợ KDF

Hàm dẫn xuất khóa này sinh ra các bit giả ngẫu nhiên. Nó trả về *numoctets* các octet đầu ra.

ĐẦU VÀO: Khóa chủ K, chuỗi (*keylen*)-octet

index, số nguyên không âm nhỏ hơn 2^{64}

numoctets, số nguyên không âm nhỏ hơn 2^{64}

ĐẦU RA: Y, chuỗi (*numoctets*)-octet

a) n = ceil(*numoctets* / *blocklen*)

b) Đặt Y là chuỗi rỗng

c) For i = 1 to n do

1) T = uint2bitstr(*index*, *blocklen* – 8) || uint2bitstr(i, 8)

2) T = Enc(K, T)

3) $Y = Y || T$

d) $Y = Y[1...numoctets]$

e) Output Y

CHÚ THÍCH Hàm dẫn xuất khóa KDF sử dụng một mã khối ở chế độ con đếm như được định nghĩa trong ISO/IEC 10116 [8].

6.2.3.3 Hàm bổ trợ PDF

Hàm dẫn xuất đệm này nhận một khóa và một nonce, trả về một dãy đệm giả ngẫu nhiên để sử dụng trong việc sinh thẻ. Một chuỗi đệm có độ dài 4, 8, 12 hoặc 16 octet có thể được sinh ra.

ĐẦU VÀO: Khóa chủ K, chuỗi (*keylen*)-octet

Nonce N, chuỗi có độ dài từ 1 tới *blocklen* octet.

Độ dài thẻ *taglen*, số nguyên bằng 4, 8, 12 hoặc 16.

ĐẦU RA: Y, chuỗi (*taglen*)-octet

a) *PDFnonce* = N

b) if (*taglen* = 4 hoặc *taglen* = 8)

1) *index* = bitstr2uint(N) mod (*blocklen* / *taglen*)

2) *PDFnonce* = $N \oplus$ unit2bitstr(*index*, octetlength(N))

c) padlen = blocklen – octetlength(*PDFnonce*)

d) *PDFnonce* = *PDFnonce* || $0^{padlen*8}$

e) *K'* = KDF(*K*, 0, *keylen*)

f) *T* = Enc(*K'*, *PDFnonce*)

g) if (*taglen* = 4 hoặc *taglen* = 8)

1) *Y* = *T*[(*index***taglen*) + 1...(*index***taglen*) + *taglen*]

h) else

1) *Y* = *T*[1...*taglen*]

i) Output *Y*

CHÚ THÍCH Các dãy đệm được sinh ra dùng các nonce mà chỉ khác nhau ở bit cuối cùng của chúng (khi sinh các chuỗi đệm 8-octet) hoặc hai bit cuối cùng (khi sinh các chuỗi đệm 4-octet) là được dẫn xuất từ cùng phép mã hóa của mã khối. Điều này cho phép lưu giữ và dùng chung một lần gọi mã khối duy nhất cho các nonce liên tiếp.

6.2.3.4 Hàm bổ trợ NH

NH ("Hàm băm phi tuyến - Non-linear Hash-function") là một hàm băm phổ biến.

CHÚ THÍCH Hàm băm phổ biến NH đã được giới thiệu bởi Black và các cộng sự [2].

ĐẦU VÀO: *Key*, một chuỗi 1024-octet

Msg, chuỗi các octet, độ dài octet của nó là bội nguyên của 32 và nhỏ hơn hay bằng 1024.

ĐẦU RA: *Y*, chuỗi 8-octet

Ngắt *Msg* và *Key* thành các khối 4-octet:

a) t = octetlength(*Msg*) / 4

b) Chia *Msg* thành các chuỗi 4-octet $M_1, M_2, ..., M_t$, để có $Msg = M_1 || M_2 || ... || M_t$.

c) Giả sử $K_1, K_2, ..., K_t$ là các chuỗi 4-octet để có $K_1 || K_2 || ... || K_t$ là một tiền tố của *Key* (4*t* octet trái nhất của *Key*)

d) $Y = 0^{64}$

e) $i = 1$

f) while($i < t$) do

1) $Y = Y +_{64} ((M_{i+0} +_{32} K_{i+0}) *_{64} (M_{i+4} +_{32} K_{i+4}))$

2) $Y = Y +_{64} ((M_{i+1} +_{32} K_{i+1}) *_{64} (M_{i+5} +_{32} K_{i+5}))$

3) $Y = Y +_{64} ((M_{i+2} +_{32} K_{i+2}) *_{64} (M_{i+6} +_{32} K_{i+6}))$

4) $Y = Y +_{64} ((M_{i+3} +_{32} K_{i+3}) *_{64} (M_{i+7} +_{32} K_{i+7}))$

5) $i = i + 8$

g) Return *Y*

CHÚ THÍCH Thủ tục này được áp dụng trực tiếp vào mỗi bit của dữ liệu đầu vào, vì thế cài đặt tối ưu của nó dẫn tới lợi ích lớn. Nó có thể được thực hiện trên các khối 4-octet, phân cặp các từ cho phép nhân mà được tách thành 4 để thích nghi song song hóa véctơ.

6.2.3.5 Hàm bổ trợ ENDIAN-SWAP

Hàm ENDIAN-SWAP chuyển đổi một chuỗi của các từ 4-octet từ little-endian thành big-endian, hoặc ngược lại.

ĐẦU VÀO: *S*, chuỗi có độ dài chia hết cho 4 octet

ĐẦU RA: *T*, chuỗi *S* cùng với mỗi từ 4-octet có endian đảo ngược lại.

a) n = octetlength(S) / 4

b) Giả sử $S_1, S_2, \ldots, S_n$ là các chuỗi có độ dài 4 octet để $S_1 \| S_2 \| \ldots \| S_n = S$

c) Đặt T là chuỗi rỗng

d) For i = 1 to n do

   1) Giả sử W_1, W_2, W_3, W_4 là các octet để $W_1 \| W_2 \| W_3 \| W_4 = S_i$

   2) $S_{Reversed} = W_4 \| W_3 \| W_2 \| W_1$

   3) $T = T \| S_{Reversed}$

e) Output T

6.2.3.6 Hàm bổ trợ POLY

Hàm POLY là một hàm băm đa thức được sử dụng trong hàm băm tầng thứ hai L2-HASH, xem Điều 6.2.7.2.

ĐẦU VÀO: *wordbits*, số nguyên 64 hoặc 128

maxwordrange, số nguyên dương nhỏ hơn $2^{wordbits}$

key, số nguyên trong dải 0...prime(*wordbits*) – 1

Msg, chuỗi cùng độ dài chia hết cho (*wordbits* /8) octet

ĐẦU RA: *y*, số nguyên trong dải 0...prime(*wordbits*) – 1

a) *wordoctets* = *wordbits* / 8

b) p = prime(*wordbits*)

c) $offset = 2^{wordbits} - p$

d) $marker = p - 1$

e) n = octetlength(*Msg*) / *wordoctets*

f) Giả sử $M_1, M_2, \ldots, M_n$ là các chuỗi có độ dài *wordoctets* octet để $Msg = M_1 \| M_2 \| \ldots \| M_n$

g) $y = 1$

h) for i = 1 to n do

   1) m = bitstr2uint(M_i)

   2) if ($m \geq maxwordrange$) then

      i) $y = (key * y + marker) \bmod p$

      ii) $y = (key * y + (m - offset)) \bmod p$

   3) else

      i) $y = (key * y + m) \bmod p$

i) Output y

6.2.4 Tiền xử lý khóa

UMAC sử dụng mã khối Enc. Mã khối cần được chọn sao cho *blocklen* ít nhất là 16 và lũy thừa của hai.

CHÚ THÍCH 1 Khuyến cáo sử dụng AES-128 cho UMAC. Trong trường hợp này, chúng ta có *blocklen* = 16 và *keylen* = 16.

CHÚ THÍCH 2 Nếu một số thông điệp phải được xác thực, có ý nghĩa để lưu trong bộ đệm khóa băm K_H, vì nó có thể được sử dụng lại. Chỉ có khóa mã hóa K_E phải được tính lại cho mỗi thông điệp mới.

ĐẦU VÀO: Khóa chủ *K*, chuỗi (*keylen*)-octet

Nonce *N*, chuỗi có độ dài từ 1 đến *blocklen* octet

Độ dài thẻ *taglen*, số nguyên bằng 4, 8, 12 hay 16.

ĐẦU RA: Khóa băm K_H = (*L1Key*, *L2Key*, *L3Key1*, *L3Key2*), chuỗi có độ dài thay đổi

Khóa mã hóa K_E, chuỗi (*taglen*)-octet

a) *iters* = *taglen* / 4

b) *L1Key* = KDF(*K*, 1, 1024 + (*iters* – 1) * 16)

c) *L2Key* = KDF(*K*, 2, *iters* * 24)

d) *L3Key1* = KDF(*K*, 3, *iters* * 64)

e) *L3Key2* = KDF(*K*, 4, *iters* * 4)

f) K_E = PDF(*K*, *N*, *taglen*)

g) Output K_H = (*L1Key*, *L2Key*, *L3Key1*, *L3Key2*), K_E

6.2.5 Tiền xử lý thông điệp

Các thông điệp được băm được xem như các chuỗi bit mà được đệm bởi các bit 0 vào bên trái để có độ dài octet thích hợp. Một khi thông điệp đã được đệm, tất cả các chuỗi được xem như các chuỗi octet.

CHÚ THÍCH Dữ liệu thông điệp được đọc theo kiểu little-endian để tăng tốc việc sinh thẻ trên các máy tính little-endian.

6.2.6 Băm thông điệp

ĐẦU VÀO: Khóa băm K_H = (*L1Key*, *L2Key*, *L3Key1*, *L3Key2*), chuỗi có độ dài thay đổi

Khóa mã hóa K_E, chuỗi (*taglen*)-octet

Thông điệp *M*, chuỗi có độ dài nhỏ hơn 2^{67} octet

taglen, số nguyên bằng 4, 8, 12 hay 16

ĐẦU RA: Thẻ *H*, chuỗi (*taglen*)-octet

a) Đặt *H* bằng chuỗi rỗng

b) For *i* = 1 to (*taglen*) / 4 do

1) $L1Key_i$ = *L1Key*[(*i* – 1) * 16 + 1 ... (*i* – 1)* 16 + 1024]

2) $L2Key_i$ = *L2Key*[(*i* – 1) * 24 + 1 ... *i* * 24]

3) $L3Key1_i$ = *L3Key1*[(*i* – 1) * 64 + 1 ... *i* * 64]

4) $L3Key2_i$ = *L3Key2*[(*i* – 1) * 4 ... *i* * 4]

5) *A* = L1-HASH($L1Key_i$, *M*)

6) if (bitlength(*M*) ≤ bitlength($L1Key_i$) then

i) $B = 0^{64} \| A$

7) Else

i) *B* = L2-HASH($L2Key_i$, *A*)

8) *C* = L3-HASH($L3Key1_i$, $L3Key2_i$, *B*)

9) $H = H \| C$

c) Output *H*

6.2.7 Các hàm băm phân tầng

6.2.7.1 Hàm băm tầng thứ nhất L1-HASH

Hàm băm tầng thứ nhất ngắt thông điệp thành các khối 1024-octet (đệm khối cuối cùng nếu cần thiết) và sau đó điều chỉnh thứ tự endian và băm mỗi khối bằng hàm NH. Ghép nối các kết quả tạo thành một chuỗi, ngắn hơn 128 lần so với chuỗi ban đầu.

ĐẦU VÀO: *L1Key*, chuỗi 1024-octet

L1Msg, chuỗi có độ dài nhỏ hơn 2^{67} octet

ĐẦU RA: H_1, chuỗi có độ dài (8 * ceil(bitlength(*L1Msg*) / 8192)) octet

a) t = max(ceil(bitlength(*L1Msg*) / 8192), 1)

b) Chia *L1Msg* thành các chuỗi M_1, M_2, ..., M_t, để có *L1Msg* = M_1 || M_2 || ... || M_t và octetlength(M_i) = 1024 đối với mọi $1 \le i \le t - 1$.

c) *Len* = uint2bitstr(1024 * 8, 8)

d) *H1* = ⟨empty string⟩

e) For i = 1 to $t - 1$ do

1) ENDIAN-SWAP(M_i)

2) *H1* = *H1* || (NH(*L1Key*, M_i) $+_{64}$ *Len*)

f) *Len* = unit2bitstr(bitlength(M_t), 8)

g) M_t = zeropad(M_t, 32)

h) ENDIAN-SWAP(M_t)

i) *H1* = *H1* || (NH(*L1Key*, M_t) $+_{64}$ *Len*)

j) Output *H1*

6.2.7.2 Hàm băm tầng thứ hai L2-HASH

Hàm băm tầng thứ hai băm lại đầu ra của L1-HASH sử dụng hàm băm đa thức được gọi là POLY. Nếu đầu ra của L1-HASH là dài, thì POLY được gọi 1 lần trên tiền tố của đầu ra L1-HASH và được gọi khi dùng các thiết lập khác nhau trên phần còn lại. Việc băm 2 bước này của đầu ra của L1-HASH chỉ cần đến nếu độ dài thông điệp là lớn hơn 16 megaoctet.

CHÚ THÍCH Một cài đặt cẩn thận của POLY là cần thiết để tránh tấn công đo thời gian có thể (xem [1] để biết thêm thông tin).

ĐẦU VÀO: *L2Key*, chuỗi 24-octet

L2Msg, chuỗi có độ dài nhỏ hơn 264 octet

ĐẦU RA : *H2*, chuỗi 16-octet

a) *Mask64* = uint2bitstr (0x, 01FFFFF 01FFFFFF, 8)

b) *Mask128* = unit2bitstr(0x 01FFFFFF 01FFFFFF 01FFFFFF 01FFFFFF, 16)

c) *k64* = bitstr2unit(*L2Key*[1...8] ∧ *Mask64*)

d) *k128* = bitstr2unit(*L2Key*[9...24] ∧ *Mask128*)

e) if (octetlength(*L2Msg*) ≤ 2^{17}) then

1) y = POLY(64, $2^{64} - 2^{32}$, *k64*, *L2Msg*)

f) else

1) M_1 = *L2Msg*[1...2^{17}]
2) M_2 = *L2Msg*[2^{17} + 1 ... octetlength(*L2Msg*)]
3) M_2 = zeropad(M_2 || unit2bitstr(0x80, 1), 16)
4) y = POLY(64, $2^{64} - 2^{32}$, *k64*, M_1)
5) y = POLY(128, $2^{128} - 2^{96}$, *k128*, uint2bitstr(y, 16) || M_2)

g) *H2* = uint2bitstr(y, 16)

h) Return *H2*

6.2.7.3 Hàm băm tầng thứ ba L3-HASH

Đầu ra từ L2-HASH gồm 16 octet. Hàm băm kết thúc này băm chuỗi 16-octet thành chuỗi có độ dài cố định 4 octet.

ĐẦU VÀO : *K1*, chuỗi 64-octet
K2, chuỗi 4-octet
Msg, chuỗi 16-octet

ĐẦU RA : *H3*, chuỗi 4-octet

a) $y = 0$

b) Ngắt *Msg* và *K1* thành 8 khối và chuyển thành các số nguyên:

1) For i = 1 to 8 do
   i) M_i = *Msg*[(i – 1) * 2 + 1... i * 2]
   ii) K_i = *K1*[(i – 1) * 8 + 1 ... i * 8]
   iii) m_i = bitstr2uint(M_i)
   iv) k_i = bitstr2uint(K_i) mod prime(36)

c) Băm tích trong, trích ra 32 bit cuối và dịch chuyển affine:

1) $y = (m_1 * k_1 + ... + m_8 * k_8)$ mod prime(36)
2) $y = y$ mod 2^{32}
3) *H3* = uint2bitstr(y, 4)
4) *H3* = *H3* ⊕ *K2*

d) Output *H3*

6.2.8 Hoàn tất

ĐẦU VÀO : Khóa mã hóa K_E, chuỗi có *taglen* octet
Giá trị băm *H*, chuỗi có *taglen* octet

ĐẦU RA : MAC có độ dài *taglen* octet

a) MAC = K_E ⊕ *H*

b) Output MAC

6.3 Badger

6.3.1 Mô tả của Badger

Badger là một thuật toán MAC mà sử dụng khóa 128-bit *K* và nonce 64-bit N. Nó xử lý một thông điệp có độ dài cho tới $2^{61} - 1$ octet thành một thẻ xác thực có độ dài *taglen*, giá trị này có thể là 4, 8, 12, 16 hay 20 octet. Thông điệp đầu vào bao gồm một số trọn vẹn các octet (tức là, độ dài bit của thông điệp sẽ là một bội của 8). Badger sử dụng một mã dòng (xem TCVN 11367-4 (ISO/IEC 18033-4)) hoặc bộ tạo số giả ngẫu nhiên PRG (xem ISO/IEC 18031).

CHÚ THÍCH Badger được đề xuất trong [3] và dựa trên kiến trúc cây băm kinh điển bởi Wegman và Carter [10].

6.3.2 Các yêu cầu

Trước khi sử dụng Badger, các tham số sau cần phải được thống nhất:

- Một mã dòng từ TCVN 11367-4 (ISO/IEC 18033-4) hoặc bộ tạo giả ngẫu nhiên mà tuân theo ISO/IEC 18031.
- Độ dài thẻ, *taglen*, là 4, 8, 12, 16 hay 20 octet.

6.3.3 Chú giải và các hàm bổ trợ

6.3.3.1 Hàm bổ trợ ENH

ENH (Hàm băm phi tuyến nâng cao – Enhanced Non-linear Hash-function) là một hàm băm phổ biến.

CHÚ THÍCH Hàm băm phổ biến ENH đã được giới thiệu bởi Boesgaard và các cộng sự [3]. Nó dựa trên hàm NH của Black và các cộng sự [2] mà đã được sử dụng trong UMAC.

ĐẦU VÀO : Khóa *LKey*, chuỗi 8-octet

Thông điệp *Left*, chuỗi 8-octet

Thông điệp *Right*, chuỗi 8-octet

ĐẦU RA : Giá trị băm *LHash*, chuỗi 8-octet

a) k_L = octetstr2uint(*LKey*[0...3], k_U = octetstr2uint(*LKey*[4...7])

b) $m1_L$ = octetstr2uint(*Right*[0...3], $m1_U$ = octetstr2uint(*Right*[4...7])

c) *m2* = octetst2uint(*Left*[0...7])

d) $h_L = (m1_L + k_L) \bmod 2^{32}$

e) $h_U = (m1_U + k_U) \bmod 2^{32}$

f) *Let h'* = $((h_U * h_L) + m2) \bmod 2^{64}$

g) *LHash* = unit2octetstr(*h'*, 8)

h) Output *LHash*

6.3.4 Tiền xử lý khóa

Độ dài khóa của Badger là 16 octet, và độ dài của nonce là 8 octet. Nếu bộ tạo yêu cầu các khóa và các nonce dài hơn, thì các octet còn lại có thể được đệm bằng các bit 0. Giá trị nonce cần phải khác với véctơ toàn 1. PRG được giả thiết có các giao diện sau:

- PRG_Init(*K*, *N*) khởi tạo trạng thái bên trong của PRG cùng với khóa *K* và nonce *N*.
- PRG_Next(*n*) tạo ra n bit đầu ra tiếp theo từ PRG.

Khi dùng các hàm này, các khóa băm và mã hóa được tính như sau.

CHÚ THÍCH Nếu một số thông điệp phải được xác thực, thì có ý nghĩa việc lưu trong bộ đệm khóa băm K_H, vì nó có thể được dùng lại. Chỉ có khóa mã hóa K_E phải được tính lại cho mỗi thông điệp mới.

ĐẦU VÀO: khóa chủ *K*, chuỗi 16-octet

Nonce *N*, chuỗi 8-octet

Độ dài theo bit *maxlen* của thông điệp đầu vào có thể dài nhất, bội số nguyên của 8 với $0 \leq maxlen \leq 2^{64} - 8$

Độ dài thẻ *taglen*, số nguyên (một trong số 4, 8, 12, 16, 20)

ĐẦU RA : Khóa băm $K_H = (KL, kf)$, chuỗi có độ dài thay đổi (trong đó *KL* là một véctơ của các chuỗi 8-octet và *kf* là một véctơ của các số nguyên 4-octet)

Khóa mã hóa K_E, chuỗi (*taglen*)-octet

a) PRG_Init(K, 1^{64})

b) *words_used* = 0

c) $u = taglen / 4$

d) $v = \max\{1, \text{ceil}(\log_2(maxlen)) - 6\}$

e) for $j = 1$ to 6 do

   1) for $i = 1$ to u do

      i) $kf_{j,i}$ = octetstr2uint(PRG_Next(32))

      ii) *words_used* = *words_used* + 1

f) for $j = 1$ to 6 do

   1) for $i = 1$ to u do

      i) while($kf_{j,i} \geq$ prime(32))

         I) $kf_{j,i}$ = octetstr2uint(PRG_Next(32))

         II) *words_used* = *words_used* + 1

g) while(*words_used* mod $4 \neq 0$)

   1) Loại bỏ PRG_Next(32)

   2) *words_used* = *words_used* + 1

h) for $j = 1$ to v do

   1) for $i = 1$ to u do

      i) $KL_{j,i}$ = PRG_Next(64)

i) PRG_Init(K, N)

j) K_E = PRG_Next(32 * u)

k) Output $K_H = (KL, kf)$, K_E

6.3.5 Tiền xử lý thông điệp

Tiền xử lý thông điệp là không cần thiết đối với Badger.

6.3.6 Băm thông điệp

Thông điệp được băm bằng cách tính biểu thức đa thức sau:

ĐẦU VÀO: khóa băm $K_H = (KL, kf)$, chuỗi có độ dài thay đổi

Thông điệp *M*, chuỗi có độ dài nhiều nhất $2^{61} - 1$ octet

Độ dài thẻ *taglen*, số nguyên (một trong số 4, 8, 12, 16, 20)

ĐẦU RA: giá trị băm *H*, chuỗi (*taglen*)-octet

a) *len* = bitlength(*M*) như số nguyên 64-bit

b) if *len* = 0

1) $M_1 = \ldots = M_u = 0^{64}$

c) else:

1) if *len* mod 64 ≠ 0:

i) Thêm các bit 0 vào các bit có nghĩa lớn nhất cho đến khi độ dài *len* của *M* là bội của 64

2) for $i = 1$ to u:

i) $M_i = M$

ii) $v' = \max\{1, \text{ceil}(\log_2(len)) - 6\}$

iii) for $j = 1$ to v':

I) $t = \text{octetlength}(M_i)$.

II) Chia M_i thành các khối 8-octet $B_1, .., B_t$ sao cho $M_i = B_t || \ldots || B_1$

III) nếu t là chẵn

a) $M_i = \text{ENH}(KL_{j,i}, B_t, B_{t-1}) || \ldots || \text{ENH}(KL_{j,i}, B_2, B_1)$

IV) ngược lại

a) $M_i = B_t || \text{ENH}(KL_{j,i}, B_{t-1}, B_{t-2}) || \ldots || \text{ENH}(KL_{j,i}, B_2, B_1)$

d) for $i = 1$ to u:

1) $Q_i = 0^7 || len || M_i$

2) Chia Q_i thành các khối 27-bit $B_1, .., B_5$ sao cho $Q_i = B_5 || \ldots || B_1$

3) Đệm mỗi khối $B_1, .., B_5$ bằng các bit 0 ở các bit có nghĩa lớn nhất sao cho có độ dài 4 octet.

4) For $j = 1$ to 5:

i) $b_j = \text{octetstr2uint}(B_j)$

5) $s_i = ((b_1 * kf_{1,i}) + \ldots + (b_5 * kf_{5,i}) + kf_{6,i}) \bmod \text{prime}(32)$

6) $S_i = \text{uint2octetstr}(s_i, 4)$

e) $H = S_u || \ldots || S_1$

f) Output *H*

6.3.7 Hoàn tất

ĐẦU VÀO : Khóa mã hóa K_E, chuỗi (*taglen*)-octet

Giá trị băm *H*, chuỗi (*taglen*)-octet

ĐẦU RA : Mã xác thực thông điệp MAC, chuỗi (*taglen*)-octet

a) MAC = $K_E \oplus H$

b) Output MAC

6.4 Poly1305-AES

6.4.1 Mô tả của Poly1305-AES

Poly1305 là một thuật toán MAC sử dụng khóa 256-bit K (với 22 bit được đặt bằng 0) và nonce 128-bit N. Nó chấp nhận các thông điệp có độ dài tính theo octet tùy ý l_0 và tạo ra MAC 128-bit. Thông điệp đầu vào gồm một số trọn vẹn các octet, tức là độ dài tính theo bit của thông điệp phải là một bội của 8.

CHÚ THÍCH Poly1305 đã được đề xuất trong [1] và dựa trên hàm băm đa thức. So sánh với một cài đặt thô, hiệu năng của Poly1305-AES có thể được cải tiến đáng kể bằng cách theo chỉ dẫn cài đặt được đưa ra trong [1].

6.4.2 Các yêu cầu

Việc sử dụng Poly1305-AES không yêu cầu các tham số bổ sung phải được thống nhất.

6.4.3 Tiền xử lý khóa

Khóa chủ 32-octet K có một khuôn dạng đặc biệt, trong đó một số bit phải bằng 0. Các bit đó là:

- 4 bit có nghĩa lớn nhất của $K[3]$, $K[7]$, $K[11]$, $K[15]$
- 2 bit có nghĩa nhỏ nhất của $K[4]$, $K[8]$, $K[12]$.

Khóa chủ khi đó được chia đơn giản thành khóa băm và khóa mã hóa, như sau:

ĐẦU VÀO : khóa chủ K, chuỗi 32-octet

ĐẦU RA : khóa băm K_H, chuỗi 16-octet
khóa mã hóa K_E, chuỗi 16-octet

a) $K_H = K[0...15]$

b) $K_E = K[16...31]$

c) Output K_H, K_E

6.4.4 Tiền xử lý thông điệp

Thông điệp được tiền xử lý như sau:

ĐẦU VÀO : Thông điệp M, chuỗi l_0-octet

ĐẦU RA: số các khối của thông điệp s, số nguyên
thông điệp đã được tiền xử lý c_1, .., c_s, dãy của các số nguyên 17-octet.

a) Let l_0 = octetlength(M)

b) Let s = ceil(l_0/16)

c) Let t = floor(l_0/16)

d) For $i = 0, \ldots, t - 1$:

1) c_{i+1} = octetstr2uint($M[16i...16i + 15]$) + 2^{128}

e) If $s > t$:

1) $r = l_0 \bmod 16$

2) c_s = octetstr2uint($M[16t \ldots l_0\text{-}1]$) + 2^{8r}

f) Output s; c_1, ..., c_s

6.4.5 Băm thông điệp

Thông điệp được băm bằng cách tính biểu thức đa thức sau:

ĐẦU VÀO: khóa băm K_H, chuỗi 16-octet

số các khối của thông điệp s, số nguyên

thông điệp đã được tiền xử lý, c_1, ..., c_s, dãy của các số nguyên 17-octet

ĐẦU RA: giá trị băm H, số nguyên 16-octet

a) r = octetstr2uint(K_H)

b) $H' = (c_1 * r^s + c_2 * r^{s-1} + ... + c_s * r_1) \bmod \text{prime}(130)$

c) $H = H' \bmod 2^{128}$

d) Output H

6.4.6 Hoàn tất

Cuối cùng, thông điệp được mã hóa dùng mã khối AES-128, đã được mô tả trong ISO/IEC 18033-3.

ĐẦU VÀO: giá trị băm H, chuỗi 16-octet

khóa mã hóa K_E, chuỗi 16-octet

nonce N, chuỗi 16-octet

ĐẦU RA: mã xác thực thông điệp MAC, chuỗi 16-octet

a) Let S = AES-128(K_E, N)

b) s = octetstr2uint(S)

c) Let $mac = (H + s) \bmod 2^{128}$

d) MAC = uint2octetstr(mac, 16)

e) Output MAC

6.5 GMAC

6.5.1 Mô tả của GMAC

GMAC có thể được sử dụng cùng với mã khối bất kỳ từ TCVN 11367-3 (ISO/IEC 18033-3) mà có độ dài khối bằng 128 bit. MAC kết quả có độ dài t bit, trong đó t là bội của 8 và thỏa mãn $64 \le t \le 128$. Độ dài của thông điệp đầu vào cần phải nhỏ hơn hay bằng 2^{64} khối.

CHÚ THÍCH Cơ chế này là một trường hợp đặc biệt của GCM (Galois/Counter Mode) được quy định trong ISO/IEC 19772, trong đó không có dữ liệu được mã hóa. GCM là thuộc về McGrew và Viega [9].

6.5.2 Các yêu cầu

Trước khi sử dụng GCM, các tham số sau cần phải thống nhất:

- Một mã khối từ TCVN 11367-3 (ISO/IEC 18033-3) cùng với độ dài khối bằng 128 bit. Lựa chọn của mã khối xác định độ dài khóa $|K|$;
- Độ dài thẻ t tính theo bit, trong đó $64 \le t \le 128$;
- Độ dài của nonce.

6.5.3 Chú giải và các hàm bổ trợ

6.5.3.1 Hàm bổ trợ GHASH

Hàm GHASH nhận như đầu vào là một khối 128-bit H và hai chuỗi bit có độ dài tùy ý W và Z, cho đầu ra là một khối 128-bit.

ĐẦU VÀO: khối 128-bit H

Các chuỗi bit độ dài tùy ý W và Z

ĐẦU RA : giá trị 128-bit X_{k+l+1}

a) Lấy k và u là các số nguyên duy nhất sao cho bitlength(W) = 128(k - 1) + u và 0 < u ≤ 128. Giả sử W_1, W_2, ..., W_k là dãy các khối 128-bit (với ngoại trừ có thể của W_k, nó chứa u bit cuối cùng của W) đã nhận được bằng cách phân chia W.

b) Lấy l và u là các số nguyên duy nhất sao cho bitlength(Z) = 128(l - 1) + v và 0 < v ≤ 128. Giả sử Z_1, Z_2, ..., Z_l là dãy các khối 128-bit (với ngoại trừ có thể của Z_l, nó chứa v bit cuối cùng của Z) đã nhận được bằng cách phân chia Z.

c) Tính giá trị 128-bit X_{k+l+1} dùng đệ quy sau:

1) $X_0 = 0^{128}$

2) $X_i = (X_{i-1} \oplus W_i) \bullet H$, $1 \le i \le k-1$ (bỏ qua bước này nếu $k \le 1$)

3) $X_k = (X_{k-1} \oplus (W_k \,||\, 0^{128-u})) \bullet H$ (bỏ qua bước này nếu $k = 0$)

4) $X_i = (X_{i-1} \oplus Z_{i-k}) \bullet H$, $k+1 \le i \le k+l-1$ (bỏ qua bước này nếu $l \le 1$)

5) $X_{k+l} = (X_{k+l-1} \oplus (Z_l \,||\, 0^{128-v})) \bullet H$ (bỏ qua bước này nếu $l = 0$)

6) $X_{k+l+1} = (X_{k+l} \oplus$ unit2bitstr(bitlength(W), 8) || unit2bitstr(bitlength(Z), 8)) • H.

d) Output X_{k+l+1}.

6.5.4 Tiền xử lý khóa

Khóa chủ K được sử dụng để dẫn xuất ra khóa băm K_H và khóa mã hóa K_E như sau.

ĐẦU VÀO: khóa chủ K

ĐẦU RA : khóa băm K_H

khóa mã hóa K_E

a) $K_H = \text{Enc}(K, 0^{128})$

b) $K_E = K$

c) Output K_H, K_E

6.5.5 Tiền xử lý thông điệp

Tiền xử lý thông điệp không cần cho GMAC.

6.5.6 Băm thông điệp

ĐẦU VÀO: thông điệp M

khóa băm K_H

ĐẦU RA : giá trị băm H

a) $H = \text{GHASH}(K_H, M, \{\})$

b) Output H

6.5.7 Hoàn tất

Một nonce độ dài thay đổi N cần phải được chọn. Giá trị này cần phải khác nhau cho mỗi thông điệp được bảo vệ, và được sẵn sàng đối với người nhận thông điệp. Tuy nhiên, giá trị này không cần thiết phải không dự đoán được hoặc bí mật.

CHÚ THÍCH Giá trị N có thể, ví dụ, được sinh ra bằng cách dùng một con đếm được duy trì bởi người khởi tạo, và được gửi đi ở dạng văn bản rõ cùng với thông điệp được bảo vệ.

ĐẦU VÀO: giá trị băm H

khóa mã hóa K_E

nonce N

ĐẦU RA: mã xác thực thông điệp MAC, chuỗi t-bit

a) Nếu bitlength(N) = 96 thì $Y_0 = N \,||\, 0^{31} \,||\, 1$. Ngược lại thì $Y_0 = \text{GHASH}(K_H, \{\}, N)$.

b) $\text{MAC} = (H \oplus \text{Enc}(K_E, Y_0))|_t$

c) Output MAC.

Phụ lục A

(quy định)

Các định danh đối tượng

Phụ lục này liệt kê các định danh đối tượng được gán cho các thuật toán được quy định trong tiêu chuẩn này.

```
MessageAuthenticationCodesPart3 {
  iso(1) standard(0) message-authentication-codes(9797)
  part3(3) asn1-module(0) algorithm-object-identifiers(0)
}
DEFINITIONS EXPLICIT TAGS ::= BEGIN
EXPORTS ALL;
IMPORTS
    BlockAlgorithms
        FROM EncryptionAlgorithm-3 {iso(1) standard(0)
          encryption-algorithms(18033) part(3)
          asn1-module(0) algorithm-object-identifiers(0)};
StreamCipherAlgorithms
        FROM EncryptionAlgorithm-4 {iso(1) standard(0)
          encryption-algorithms(18033) part(4)
          asn1-module(0) algorithm-object-identifiers(0)};

OID ::= OBJECT IDENTIFIER -- Alias

-- cách dùng khác
is9797-3 OID ::= {iso standard message-authentication-
        codes(9797) part3(3)}
-- Gán OID

id-umac   OID ::= {is9797-3 umac(1)}
id-badger OID ::= {is9797-3 badger(2)}
id-poly1305 OID ::= {is9797-3 poly1305-aes(3)}

-- kiểu định danh thuật toán MAC và tập các thuật toán MAC đã nhận diện
MessageAuthenticationCode ::=
         AlgorithmIdentifier{{MacAlgorithms}}
```

```
MacAlgorithms ALGORITHM ::= {
{ OID id-umac PARMS UmacParameters} |
{ OID id-badger PARMS BadgerParameters} |
{ OID id-poly1305 PARMS NullParameters} |
{OID id-gmac PARMS GmacParameters   },
... -- thuật toán bổ sung mong đợi --
}

-- định nghĩa các kiểu tham số MAC

UmacParameters ::= SEQUENCE {
   bcAlgo BlockAlgorithms,
   taglength INTEGER,
   noncelength INTEGER
}

BadgerParameters ::= SEQUENCE {
   scAlgo StreamCipherAlgorithms,
   taglength INTEGER,
}

GmacParameters ::= SEQUENCE {
   bcAlgo BlockAlgorithms,
   taglength INTEGER,
   noncelength INTEGER
}

END -- MessageAuthenticationCodesPart3--
```

Phụ lục B
(tham khảo)
Các véctơ kiểm tra

B.1 UMAC

Điều này chứa một số véctơ kiểm tra của UMAC, dùng AES-128 như mã khối. Bảng B.1 liệt kê các thẻ được sinh bởi UMAC khi dùng khóa 16-byte *K* và none 8-byte *N*.

- *K* = "abcdefghijklmnop"
- *N* = "bcdefghi"

Bảng B.1 – Các vectơ kiểm tra cho UMAC

Thông điệp	UMAC-32	UMAC-64	UMAC-96	UMAC-128
⟨empty⟩	113145FB	6E155FAD26900BE1	32FEDB100C79AD58F07FF764	32FEDB100C79AD58F07FF7643CC60465
'a' * 3	3B91D102	44B5CB542F220104	185E4FE905CBA7BD85E4C2DC	185E4FE905CBA7BD85E4C2DC3D117D8D
'a' * 2^{10}	599B350B	26BF2F5D60118BD9	7A54ABE04AF82D60FB298C3C	7A54ABE04AF82D60FB298C3CBD195BCB
'a' * 2^{15}	58DCF532	27F8EF643B0D118D	7B136BD911E4B734286EF2BE	7B136BD911E4B734286EF2BE501F2C3C

B.2 Badger

Điều này chứa một số véctơ kiểm tra của Badger, dùng Rabbit như mã dòng. Bảng B.2 liệt kê các thẻ được sinh bởi Badger khi dùng khóa *K* và *IV* sau.

- K = 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
- IV = 00 01 02 03 04 05 06 07

Bảng B.2 – Các véctơ kiểm tra cho Badger

Thông điệp	Thẻ Badger
⟨empty⟩	54 6D 3A 85 F8 CB FA D9 E0 58 50 58 2C AC 3D E4
00	5F AA AB 85 AC BE 04 48 1D D6 34 D0 FA D9 FA FA
01	47 EA 18 A1 99 AE 07 31 7C A5 AC C9 37 2F 55 85
00 01 02 03 04 05 06 07 08	F7 02 3D 65 CF 66 69 23 47 A0 8B 5F 93 55 84 27

B.3 Poly1305-AES

Điều này chứa một số véctơ kiểm tra cho Poly1305-AES trong Bảng B.3.

Bảng B.3 – Các véctơ kiểm tra cho Poly1305-AES

Véctơ kiểm tra #1	**Thông điệp**	<empty>
	Khóa mã hóa K_E	75 de aa 25 c0 9f 20 8e 1d c4 ce 6b 5c ad 3f bf
	Khóa băm K_H	a0 f3 08 00 00 f4 64 00 d0 c7 e9 07 6c 83 44 03
	Nonce *N*	61 ee 09 21 8d 29 b0 aa ed 7e 15 4a 2c 55 09 cc
	Thẻ Poly1305-AES	dd 3f ab 22 51 f1 1a c7 59 f0 88 71 29 cc 2e e7
Véctơ kiểm tra #2	**Thông điệp**	f3 f6
	Khóa mã hóa K_E	ec 07 4c 83 55 80 74 17 01 42 5b 62 32 35 ad d6
	Khóa băm K_H	85 1f c4 0c 34 67 ac 0b e0 5c c2 04 04 f3 f7 00
	Nonce *N*	fb 44 73 50 c4 e8 68 c5 2a c3 27 5c f9 d4 32 7e
	Thẻ Poly1305-AES	f4 c6 33 c3 04 4f c1 45 f8 4f 33 5c b8 19 53 de
Véctơ kiểm tra #3	**Thông điệp**	66 3c ea 19 0f fb 83 d8 95 93 f3 f4 76 b6 bc 24 d7 e6 79 10 7e a2 6a db 8c af 66 52 d0 65 61 36
	Khóa mã hóa K_E	6a cb 5f 61 a7 17 6d d3 20 c5 c1 eb 2e dc dc 74
	Khóa băm K_H	48 44 3d 0b b0 d2 11 09 c8 9a 10 0b 5c e2 c2 08
	Nonce *N*	ae 21 2a 55 39 97 29 59 5d ea 45 8b c6 21 ff 0e
	Thẻ Poly1305-AES	0e e1 c1 6b b7 3f 0f 4f d1 98 81 75 3c 01 cd be
Véctơ kiểm tra #4	**Thông điệp**	ab 08 12 72 4a 7f 1e 34 27 42 cb ed 37 4d 94 d1 36 c6 b8 79 5d 45 b3 81 98 30 f2 c0 44 91 fa f0 99 0c 62 e4 8b 80 18 b2 c3 e4 a0 fa 31 34 cb 67 fa 83 e1 58 c9 94 d9 61 c4 cb 21 09 5c 1b f9
	Khóa mã hóa K_E	e1 a5 66 8a 4d 5b 66 a5 f6 8c c5 42 4e d5 98 2d
	Khóa băm K_H	12 97 6a 08 c4 42 6d 0c e8 a8 24 07 c4 f4 82 07
	Nonce *N*	9a e8 21 e7 43 97 8d 3a 23 52 7c 71 28 14 9e 3a
	Thẻ Poly1305-AES	51 54 ad 0d 2c b2 6e 01 27 4f c5 11 48 49 1f 1b

B.4 GMAC

Điều này chứa một số véctơ kiểm tra cho GMAC khi dùng AES-128 như mã khối trong Bảng B.4.

Bảng B.4 – Các véctơ kiểm tra cho GMAC

Véctơ kiểm tra #1	**Thông điệp**	<empty>
	Khóa *K*	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	Nonce *N*	00 00 00 00 00 00 00 00 00 00 00 00
	Thẻ GMAC	58 e2 fc ce fa 7e 30 61 36 7f 1d 57 a4 e7 45 5a
Véctơ kiểm tra #2	**Thông điệp**	fe ed fa ce de ad be ef fe ed fa ce de ad be ef
	Khóa *K*	6a cb 5f 61 a7 17 6d d3 20 c5 c1 eb 2e dc dc 74
	Nonce *N*	ae 21 2a 55 39 97 29 59 5d ea 45 8b c6 21 ff 0e
	Thẻ GMAC	54 df 47 4f 4e 71 a9 ef 8a 09 bf 30 da 7b 1a 92
Véctơ kiểm tra #3	**Thông điệp**	fe ed fa ce de ad be ef fe ed fa ce de ad be ef ab ad da d2 42 83 1e c2 21 77 74 24 4b 72 21 b7
	Khóa *K*	fe ff e9 92 86 65 73 1c 6d 6a 8f 94 67 30 83 08
	Nonce *N*	ca fe ba be fa ce db ad de ca f8 88
	Thẻ GMAC	1c be 39 36 e5 53 b0 8f 25 c0 8d 7b 8d c3 9f db

Phụ lục C
(tham khảo)
Thông tin an toàn

Một số tấn công chống lại các thuật toán MAC dựa trên hàm băm phổ biến đã được mô tả trong [5]. Ngược lại với các kiểu thuật toán MAC khác, một số nhỏ các giả mạo có thể dẫn tới việc khôi phục khóa đối với các hàm MAC dựa trên hàm băm phổ biến, và do vậy dẫn tới sụp đổ hoàn toàn về độ an toàn.

Để bảo vệ chống lại các tấn công này, khuyến cáo mạnh cài đặt một hoặc nhiều biện pháp phòng chống sau:

a) Tăng mức an toàn (ví dụ, taglen) để giả mạo thứ nhất là không thể thực hành được.

b) Thường xuyên làm mới khóa đầy đủ được sử dụng cho thuật toán MAC. Nếu có thể, thậm chí người ta làm mới khóa cho mỗi thông điệp. Chú ý rằng không chỉ số các thông báo được xử lý dưới một khóa đơn lẻ cần phải bị hạn chế, mà tổng số các khối thông điệp được xử lý dưới cùng một khóa cũng bị hạn chế.

c) Cả người gửi và người nhận cần đảm bảo/kiểm tra tính duy nhất của các nonce được sử dụng với cùng một khóa MAC. Khi nonce được sử dụng lại, độ an toàn của thuật toán bị suy giảm nghiêm trọng.

d) Những người nhận cần phát hiện một số lớn các lần xác minh MAC bị thất bại như một nỗ lực tấn công, và xử lý với tình huống này một cách thích hợp..

Thư mục tài liệu tham khảo

[1] Bernstein, D. J., *The Poly1305-AES message-authentication code*. Proceedings of Fast Software Encryption 2005, LNCS 3557, pp. 32-49, Springer-Verlag, 2005

[2] Black, J., Halevi, S., Krawczyk, H., Krovetz, T. and Rogaway, P. *UMAC: Fast and provably secure message authentication*, Advances in Cryptology - CRYPTO '99, LNCS vol. 1666, pp. 216-233, Springer-Verlag, 1999

[3] Boesgaard, M., Christensen, T. and Zenner, E. Badger. *A fast and provably secure MAC*, Proceedings of Applied Cryptography and Network Security, LNCS vol. 3531, pp. 176-191, Springer-Verlag, 2005

[4] Carter, L. and Wegman, M. Universal classes of hash functions, *Journal of Computer and System Sciences*, 18 (1979), pp. 143-154

[5] Handschuh, H. and Preneel, B. *Key-Recovery Attacks on Universal Hash Function based MAC Algorithms*. Advances in Cryptology - CRYPTO '08, LNCS vol. 5157, pp. 144-161, Springer-Verlag, 2008

[6] IETF RFC 4418, *UMAC: Message Authentication Code using Universal Hashing, March 2006*

[7] ISO/IEC 10181-6, *Information technology — Open Systems Interconnection — Security frameworks for open systems: Integrity framework*

[8] ISO/IEC 10116, *Information technology — Security techniques — Modes of operation for an n-bit block cipher*.

[9] Mcgrew, D. and Viega, J. *The Galois/Counter Mode of Operation (GCM)*. National Institute of Standards and Technology [web page], http://csrc.nist.gov/CryptoToolkit/modes/proposedmodes/, May 2005

[10] Wegman, M. and Carter, L. New hash functions and their use in authentication and set equality, *Journal of Computer and System Sciences*, 22 (1981), pp. 265-279