ICS 35.040
L 80

中华人民共和国国家标准

GB/T 15852.3—2019

信息技术　安全技术　消息鉴别码 第3部分:采用泛杂凑函数的机制

Information technology—Security techniques—Message authentication codes (MACs)—Part 3:Mechanisms using a universal hash-function

(ISO/IEC 9797-3:2011, MOD)

2019-08-30 发布　　2020-03-01 实施

国家市场监督管理总局
中国国家标准化管理委员会　发布

前　言

GB/T 15852《信息技术　安全技术　消息鉴别码》分为以下3个部分：

——第1部分：采用分组密码的机制；

——第2部分：采用专用杂凑函数的机制；

——第3部分：采用泛杂凑函数的机制。

本部分为GB/T 15852的第3部分。

本部分按照GB/T 1.1—2009给出的规则起草。

本部分使用重新起草法修改采用ISO/IEC 9797-3:2011《信息技术　安全技术　消息鉴别码　第3部分：采用泛杂凑函数的机制》。

本部分与ISO/IEC 9797-3:2011相比存在结构变化，将6.3.3.1调整为6.3.3，6.5.3.1调整为6.5.3。

本部分与ISO/IEC 9797-3:2011的主要技术性差异及其原因如下：

——关于规范性引用文件，本部分做了具有技术性差异的调整，以适应我国的技术条件，调整的情况集中反映在第2章“规范性引用文件”中，具体调整如下：

- 用等同采用国际标准的GB/T 15852.1—2008代替了ISO/IEC 9797-1；
- 删除了ISO/IEC 18031、ISO/IEC 18033-3、ISO/IEC 18033-4；
- 增加了GB/T 32907—2016、GB/T 33133.1—2016、GB/T 36624—2018。

——在第3章中删除了密钥、素数两个常规性术语和定义。

——在第4章中增加缩略语部分。

——在6.3.1中根据ZUC算法对初始向量的要求，将Badger的64位全1初始向量修改为128位全1初始向量。

——删除规范性附录A对象标识符(因为缺少国内相关对象标识符定义)。

本部分做了下列编辑性修改：

——调整资料性附录B为资料性附录A，列举了在底层采用ZUC或SM4算法的MAC算法所生成的测试向量；

——调整资料性附录C为资料性附录B，介绍了泛杂凑函数的安全性信息；

——增加资料性附录C，介绍了ZUC和SM4算法的抗攻击能力。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

本部分由全国信息安全标准化技术委员会(SAC/TC 260)提出并归口。

本部分起草单位：中国科学院软件研究所、中国科学院数据与通信保护研究教育中心、北京邮电大学、成都卫士通信息产业股份有限公司。

本部分主要起草人：张立廷、吴文玲、张蕾、眭晗、温巧燕、王鹏、金正平、彭真、秦体红。

信息技术　安全技术　消息鉴别码
第3部分:采用泛杂凑函数的机制

1　范围

GB/T 15852的本部分规定了4种采用泛杂凑函数的消息鉴别码算法:UMAC、Badger、Poly1305和GMAC。这些算法基于GB/T 33133.1—2016中规定的序列密码算法和GB/T 32907—2016中规定的分组密码算法,或符合国家规定的其他序列密码算法和分组密码算法,使用一个密钥和一个泛杂凑函数处理一个长度为m位的比特串,输出一个长度为n位的比特串作为MAC。

本部分适用于安全体系结构、进程及应用的安全服务。这些算法可以作为数据完整性机制,用于检验数据是否在未经授权的方式下被更改。也可以作为消息鉴别机制,确保消息来自于拥有密钥的实体。数据完整性机制和消息鉴别机制的强度由以下指标决定:密钥的长度(按比特)与保密性、泛杂凑函数产生的杂凑码的长度(按比特)、泛杂凑函数的强度、MAC的长度(按比特),以及具体的机制。

注:提供完整性服务的一般框架在ISO/IEC 10181-6[7]中指定。

2　规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件,仅注日期的版本适用于本文件。凡是不注日期的引用文件,其最新版本(包括所有的修改单)适用于本文件。

GB/T 15852.1—2008　信息技术　安全技术　消息鉴别码　第1部分:采用分组密码的机制(ISO/IEC 9797-1:1999,IDT)

GB/T 32907—2016　信息安全技术　SM4分组密码算法

GB/T 33133.1—2016　信息安全技术　祖冲之序列密码算法　第1部分:算法描述

GB/T 36624—2018　信息技术　安全技术　可鉴别的加密机制

3　术语和定义

GB/T 15852.1—2008界定的以及下列术语和定义适用于本文件。

3.1

空串　empty string

长度为零的比特串。

3.2

临时值　nonce

使用一次的值,用于向MAC算法提供新鲜输入。

3.3

标签　tag

MAC算法的结果,附加一个可能的加密消息以提供完整性保护。

3.4

泛杂凑函数　universal hash-function

由密钥确立的映射,将一定范围内任意长比特串映射到定长比特串,满足:对于所有不同的输入,其

输出在密钥均匀随机的前提下发生碰撞的概率极小。

注：泛杂凑函数由 Carter 和 Wegman 提出[4]，其在 MAC 算法中的应用最早由 Wegman 和 Carter 描述[11]。

4 符号和缩略语

4.1 符号

下列符号适用于本文件。

bit(S,n)	若比特串 S 的第 n 个比特是 1 则输出整数 1，否则输出整数 0(索引从 1 开始)
bitlength(S)	比特串 S 的比特长度
bitstr2uint(S)	一个非负整数，其二进制表示为比特串 S。形式化地，若 S 的长度为 t 比特，则 bitstr2uint(S)$=2^{t-1}*$bit(S,1)$+2^{t-2}*$bit(S,2)$+\cdots+2^{1}*$bit(S,$t-1$)$+$bit(S,t)

注 1：比特串是以高位顺序排列的，也就是说，第一个比特为最高位。

blocklen	底层分组密码的分组长度(按字节计)
ceil	向上取整操作，也就是说，若 x 是一个浮点数，则 ceil(x)是满足 $n\geqslant x$ 的最小的整数 n
Enc(K,X)	明文分组 X 在密钥 K 的作用下通过分组密码 Enc 进行加密
floor	向下取整操作，也就是说，若 x 是一个浮点数，则 floor(x)是满足 $n\leqslant x$ 的最大的整数 n
H	杂凑值
K	主密钥
K_E	加密密钥
K_H	杂凑密钥
keylen	分组密码的密钥长度(按字节计)
$\log_2$	二进制对数函数
M	消息
max	指定参数中的最大值
N	临时值
octetlength(S)	比特串 S 按字节计的长度(假定 S 的比特长度是 8 的倍数)
octetstr2uint(S)	定义为 $S[0]+2^{8}*S[1]+2^{16}*S[2]+\cdots+2^{8n-8}*S[n-1]$ 的非负整数，其中 $n=$octetlength(S)

注 2：字节串是以低位顺序排列的，即第一个字节为最低位

prime(n)	对任意正整数 n，小于 2^{n} 的最大素数值

注 3：本部分使用的素数如表 1 所示。

表 1　素数值

n	prime(n)	prime(n)的十六进制表示
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

$S[i]$　　比特串 S 的第 i 个字节(索引从 0 开始)

注 4：6.2 中关于 UMAC 的条款使用了一个起始为 1 而不是 0 的索引。

$S[i\cdots j]$　　由 S 的第 i 个到第 j 个字节组成的子串

taglen　　标签的字节长度

uint2bitstr(x,n)　　长度为 n 的字节串 S,满足 bitstr2uint(S)$=x$

uint2octetstr(x,n)　　长度为 n 的字节串 S,满足 x=octetstr2uint(S)

$X|_s$　　比特分组 X 的左截断:若 X 的比特长度大于或等于 s,则$X|_s$ 是由 X 最左侧的 s 个比特组成的长度为 s 位的比特分组

$X|^s$　　比特分组 X 的右截断:若 X 的比特长度大于或等于 s,则$X|^s$ 是由 X 最右侧的 s 个比特组成的长度为 s 位的比特分组

$X\gg1$　　比特分组 X 右移一位:$Y=X\gg1$ 最左侧的比特恒为 0

$|X|$　　X 的比特长度

zeropad(S,n)　　对于非负整数 n,用零比特对比特串 S 进行填充,直到其长度是最接近的 n 个字节的整数倍。形式化地,zeropad(S,n)$=S\parallel T$,其中 T 是满足 $S\parallel T$ 非空并且 n 可以整除 octetlength($S\parallel T$)的最短的零比特串(T 可能为空)

$\oplus$　　比特串的比特级逻辑异或运算。若 A,B 是长度相等的比特串,则 $A\oplus B$ 表示 A 和 B 的比特级逻辑异或所形成的比特串

$\wedge$　　比特串的比特级逻辑与运算。若 A,B 是长度相等的比特串,则 $A\wedge B$ 表示 A 和 B 的比特级逻辑与所形成的比特串

$+_{32}$　　两个 32 位的比特串的加法运算,得到一个 32 位的比特串。形式化地,$S+_{32}T$=uint2bitstr(bitstr2uint(S)+bitstr2uint(T)mod2^{32},4)

$+_{64}$　　两个 64 位的比特串的加法运算,得到一个 64 位的比特串。形式化地,$S+_{64}T$=uint2bitstr(bitstr2uint(S)+bitstr2uint(T)mod2^{64},8)

$*$　　整数的乘法运算

$*_{64}$　　两个 64 位的比特串的乘法运算,得到一个 64 位的比特串。形式化地,$S*_{64}T$=uint2bitstr(bitstr2uint(S)$*$bitstr2uint(T)mod2^{64},8)

注 5：$+_{32}$,$+_{64}$ 和 $*_{64}$ 运算与在现代计算机上可以高效执行的加法和乘法运算有着很好的对应。

$\|$	两个比特串的级联。若 A 和 B 分别是长度为 a 位和 b 位的比特串，则 $A \| B$ 是长度为 $(a+b)$ 位的比特串，其中最左侧的 a 位是比特串 A，最右侧的 b 位是比特串 B
0^n	由 n 个 0 组成的比特串
1^n	由 n 个 1 组成的比特串
$\{\}$	空串
$\cdot$	域 $GF(2^{128})$ 上的乘法，定义 $GF(2^{128})$ 的多项式为 $1+\alpha+\alpha^2+\alpha^7+\alpha^{128}$

注 6：令 U 和 V 是两个 128 位的比特串。则 128 位的比特串 $W=U \cdot V$ 可以如下计算：

a) 令 $W=0^{128}$，$Z=U$。

b) 对于 $i=1,2,\cdots,128$，执行以下两个步骤：

1) 若 $\mathrm{bit}(V,i)=1$，则令 $W=W \oplus Z$；

2) 若 $\mathrm{bit}(Z,128)=0$，则令 $Z=Z \gg 1$；否则令 $Z=(Z \gg 1) \oplus (11100001 \| 0^{120})$。

4.2 缩略语

下列缩略语适用于本文件。

ENH：增强的非线性杂凑函数(Enhanced Non-linear Hash-function)

GMAC：伽罗华消息鉴别码(Galois Message Authentication Code)

KDF：密钥流生成函数(key-derivation function)

MAC：消息鉴别码(Message Authentication Code)

NH：非线性杂凑函数(Non-linear Hash-function)

PDF：填充生成函数(pad-derivation function)

UMAC：泛杂凑消息鉴别码(Universal-Hash Message Authentication Code)

5 一般模型

采用泛杂凑函数的消息鉴别码算法使用了一种加密算法(分组密码算法、序列密码算法或伪随机生成器算法)。这类消息鉴别码算法具有一个特性，即：在假定加密算法安全的前提下，可以证明这类消息鉴别码算法的安全性。

附录 A 列举了在底层采用 ZUC 或 SM4 算法的 MAC 算法所生成的测试向量，附录 B 给出了泛杂凑函数的安全性信息，附录 C 给出了加密算法 ZUC 和 SM4 算法的安全性说明。

采用泛杂凑函数的 MAC 算法需要一个主密钥 K，一个消息 M 和一个临时值 N 作为输入。依照下列步骤序列可以计算得到 MAC：

a) 密钥预处理。利用主密钥 K 生成一个杂凑密钥 K_H 和一个加密密钥 K_E。其中，在 UMAC 和 Badger 中还使用临时值 N 作为输入。

b) 消息预处理。将输入的消息 M 编码为杂凑函数所需的输入格式。

c) 消息杂凑。编码后的消息在杂凑密钥 K_H 的控制下经一个泛杂凑函数进行杂凑。其结果为一个固定且长度较短的杂凑值 H。

d) 终止化操作。将杂凑值 H 在加密密钥 K_E 的控制下进行加密，其结果即为消息鉴别码 MAC。其中，在 Poly1305 和 GMAC 中还使用临时值 N 作为输入。

对于本部分中提出的所有机制，默认输入消息的长度是字节的整数倍。

对于所有采用泛杂凑函数的 MAC 算法，最重要的是，当使用相同的密钥时，鉴别每个新消息应使用不同的临时值。如果该安全性要求没有满足，算法的安全性将严重降低。

6 机制

6.1 概述

本章规范了四种采用泛杂凑函数的消息鉴别码算法。凡涉及密码算法的相关内容，按国家有关法规实施；凡涉及采用密码技术解决保密性、完整性、真实性、不可否认性需求的应遵循密码相关国家标准和行业标准。

6.2 UMAC

6.2.1 UMAC 的描述

UMAC 是一个由 4 个 MAC 算法组成的算法族，它们针对不同的输出比特长度做了效率优化，分别是 UMAC-32、UMAC-64、UMAC-96 和 UMAC-128。UMAC 使用一个长度为 128 比特的密钥 K，并且临时值 N 的长度介于 8 比特和 128 比特之间。根据使用的 UMAC 族中算法的不同，产生的 MAC 的长度分别为 32 比特、64 比特、96 比特或 128 比特。该指标由参数 *taglen* 表示，相应地为 4 个、8 个、12 个或 16 个字节。输入消息的长度应小于 2^{67} 个字节。输入 UMAC 函数的消息应包含整数个字节，也就是说，其比特长度应是 8 的倍数。若其比特长度不是 8 的倍数，则该机制不适用。

附录 A 的 A.1 中列举了在底层采用 SM4 算法的 UMAC 所生成的测试向量。

注 1：这里指定的 UMAC 的版本避免与 UMAC 算法较早的版本（如参考文献[2]等）相混淆。

注 2：若 MAC 函数的输入包含整数个字节，则这里指定的 UMAC 函数与 IETF RFC 4418[6] 中描述的函数相同。

6.2.2 要求

在使用 UMAC 之前，需要首先协商以下参数：

——分组密码算法，具体分组密码算法的选择决定了密钥长度 $|K|$ 和分组长度 $|B|$。

——标签长度，*taglen*，应为 4、8、12 或 16 之一（以字节计）。

——临时值的长度，应介于 8 比特和 128 比特之间。

6.2.3 符号和辅助函数

6.2.3.1 比特串操作

关于 UMAC 的条款在对序列中的元素标记时使用以 1 为起始的索引。从而，对于 UMAC，$S[i]$ 定义为比特串 S 的第 i 个字节，其中 $i \geqslant 1$。

6.2.3.2 辅助函数 KDF

该密钥流生成函数输出伪随机比特。它返回长度为 *numoctets* 的字节串作为输出。

输入：　主密钥 K（长度为 *keylen* 字节的比特串）

index（小于 2^{64} 的非负整数）

numoctets（小于 2^{64} 的非负整数）

输出：　Y（长度为 *numoctets* 字节的比特串）

a) $n = \mathrm{ceil}(numoctets / blocklen)$。

b) 将 Y 设为空串。

c) 对于 $i = 1$ 到 n，令：

1) $T = \mathrm{uint2bitstr}(index, blocklen - 8) \| \mathrm{uint2bitstr}(i, 8)$；

2) $T = \mathrm{Enc}(K, T)$；

3） $Y=Y \| T$。

d） $Y=Y[1 \cdots numoctets]$。

e） 输出 Y。

注：根据 GB/T 17964—2008[8] 中的定义，密钥流生成函数 KDF 采用 CTR 模式调用一个分组密码算法。

6.2.3.3 辅助函数 PDF

该填充生成函数调用一个密钥和一个临时值，返回一个用以生成标签的伪随机填充序列，长度为 4 个、8 个、12 个或 16 个字节。

输入： 主密钥 K（长度为 $keylen$ 字节的比特串）
临时值 N（长度介于 1 和 $blocklen$ 之间的字节串）
标签长度 $taglen$（整数 4、8、12 或 16）

输出： Y（长度为 $taglen$ 字节的比特串）

a） $\mathrm{PDFnonce}=N$。

b） 若（$taglen=4$ 或 $taglen=8$）：

1） $index=\mathrm{bitstr2uint}(N)\mathrm{mod}(blocklen/taglen)$；

2） $PDFnonce=N \oplus \mathrm{uint2bitstr}[index, \mathrm{octetlength}(N)]$。

c） $\mathrm{padlen}=\mathrm{blocklen}-\mathrm{octetlength}(\mathrm{PDFnonce})$。

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

e） $K'=\mathrm{KDF}(K, 0, keylen)$。

f） $T=\mathrm{Enc}(K', \mathrm{PDFnonce})$。

g） 若（$taglen=4$ 或 $taglen=8$）：

$Y=T[(\mathrm{index} * \mathrm{taglen})+1 \cdots (\mathrm{index} * \mathrm{taglen})+\mathrm{taglen}]$。

h） 其他情况：

$Y=T[1 \cdots taglen]$。

i） 输出 Y。

注：若使用的临时值只有最后一个比特（当生成长度为 8 字节的填充时）或最后两个比特（当生成长度为 4 字节的填充时）不同，则生成的填充序列可由相同的分组密码加密获得。从而，连续的临时值可以通过缓存和共享同一次分组密码调用生成填充序列。

6.2.3.4 辅助函数 NH

NH（非线性杂凑函数）是一个泛杂凑函数。

注 1：非线性杂凑函数 NH 由 Black 等提出[2]。

输入： Key（长度为 1 024 字节的比特串）
Msg（字节串，其长度是 32 的倍数并且小于或等于 1 024）

输出： Y（长度为 8 字节的比特串）

将 Msg 和 Key 分割成长度为 4 的字节分组：

a） $t=\mathrm{octetlength}(Msg)/4$。

b） 将 Msg 分割成长度为 4 的字节串 $M_1, M_2, \cdots, M_t$，从而 $Msg=M_1 \| M_2 \| \cdots \| M_t$。

c） 令 $K_1, K_2, \cdots, K_t$ 为长度为 4 的字节串，即 $K_1 \| K_2 \| \cdots \| K_t$ 是 Key 的一个前缀（Key 最左侧的 $4t$ 个字节）。

d） $Y=0^{64}$。

e） $i=1$。

f） 当（$i<t$）时，令：

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) 返回 Y。

注 2：这一程序直接作用于输入数据的每一个比特，所以对该程序的优化实现将产生极大的效益。该程序可作用于长度为 4 的字节分组，也可以作用于乘法中的两个字，这两个字宜各分为 4 个部分以便于向量并行化操作。

6.2.3.5 辅助函数 ENDIAN-SWAP

函数 ENDIAN-SWAP 将一个长度为 4 的字节串从按低位排序转换为按高位排序，或反之。

输入： S（长度可被 4 个字节整除的比特串）

输出： T（每 4 个字节尾数逆转的比特串 S）

a) $n=\mathrm{octetlength}(S)/4$。

b) 令 $S_1,S_2,\cdots,S_n$ 为长度为 4 的字节串，满足 $S_1\parallel S_2\parallel\cdots\parallel S_n=S$。

c) 将 T 设为空串。

d) 对于 $i=1$ 到 n：

1) 令 W_1,W_2,W_3,W_4 为字节，满足 $W_1\parallel W_2\parallel W_3\parallel W_4=S_i$。

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

3) $T=T||S_{Reversed}$。

e) 输出 T。

6.2.3.6 辅助杂凑函数 POLY

函数 POLY 是一个多项式杂凑函数，于第二层杂凑函数 L2-HASH 中使用，参见 6.2.7.2。

输入： $wordbits$（整数 64 或 128）

$maxwordrange$（小于 $2^{wordbits}$ 的正整数）

key[取值范围为 0…prime($wordbits$)-1 的整数]

Msg[长度可被($wordbits$/8)个字节整除的比特串]

输出： y（取值范围为 0…prime($wordbits$)-1 的整数）

a) $wordoctets=wordbits/8$。

b) $p=\mathrm{prime}(wordbits)$。

c) $offset=2^{wordbits}-p$。

d) $marker=p-1$。

e) $n=\mathrm{octetlength}(Msg)/wordoctets$。

f) 令 $M_1,M_2,\cdots,M_n$ 为长度为 $wordoctets$ 的字节串，满足 $Msg=M_1\parallel M_2\parallel\cdots\parallel M_n$。

g) $y=1$。

h) 对于 $i=1$ 到 n：

1) $m=\mathrm{bitstr2uint}(M_i)$；

2) 若($m\geqslant maxwordrange$)，则：

i) $y=(key*y+marker)\bmod p$；

ii) $y=(key*y+(m-offset))\bmod p$。

3） 其他情况：

$y=(key*y+m)\bmod p$。

i） 输出 y。

6.2.4 密钥预处理

UMAC 使用的分组密码算法应当满足其分组长度 *blocklen* 至少为 16，并且是 2 的方幂。

注：如果有若干个消息需要进行鉴别，杂凑密钥 K_H 将被重复使用，缓存该密钥是有必要的。只有加密密钥 K_E 在每次处理新消息时被重新计算。

输入： 主密钥 K（长度为 *keylen* 字节的比特串）

临时值 N（长度介于 1 和 *blocklen* 之间的字节串）

标签长度 *taglen*（整数 4、8、12 或 16）

输出： 杂凑密钥 $K_H=(L1Key,L2Key,L3Key1,L3Key2)$（任意长度的比特串）

加密密钥 K_E（长度为 *taglen* 字节的比特串）

a） $iters=taglen/4$。

b） $L1Key=\mathrm{KDF}[K,1,1024+(iters-1)*16]$。

c） $L2Key=\mathrm{KDF}(K,2,iters*24)$。

d） $L3Key1=\mathrm{KDF}(K,3,iters*64)$。

e） $L3Key2=\mathrm{KDF}(K,4,iters*4)$。

f） $K_E=\mathrm{PDF}(K,N,taglen)$。

g） 输出 $K_H=(L1Key,L2Key,L3Key1,L3Key2),K_E$。

6.2.5 消息预处理

在需要被杂凑的消息右侧填充零串使其达到字节长度整数倍。消息一旦被填充，所有的比特串都被视为字节串。

注：在按低位顺序排列的计算机上，消息数据按从低位顺序排列读取以加速标签的生成。

6.2.6 消息杂凑

输入： 杂凑密钥 $K_H=(L1Key,L2Key,L3Key1,L3Key2)$（任意长度的比特串）

加密密钥 K_E（长度为 *taglen* 字节的比特串）

消息 M（长度小于 2^{67} 字节的比特串）

taglen（整数 4、8、12 或 16）

输出： 标签 H（长度为 *taglen* 字节的比特串）

a） 将 H 设为空串。

b） 对于 $i=1$ 到 $(taglen/4)$，令：

1） $L1Key_i=L1Key[(i-1)*16+1\cdots(i-1)*16+1\ 024]$；

2） $L2Key_i=L2Key[(i-1)*24+1\cdots i*24]$；

3） $L3Key1_i=L3Key1[(i-1)*64+1\cdots i*64]$；

4） $L3Key2_i=L3Key2[(i-1)*4+1\cdots i*4]$；

5） $A=\text{L1-HASH}(L1Key_i,M)$；

6） 若 $(\mathrm{bitlength}(M)<=\mathrm{bitlength}(L1Key_i))$，则：

$B=0^{64}\parallel A$；

7） 其他情况：

$B=$L2-HASH$(L2Key_i, A)$；

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

9） $H=H \| C$。

c） 输出 H。

6.2.7 分层杂凑函数

6.2.7.1 第一层杂凑函数 L1-HASH

第一层杂凑将消息分割成长度为 1 024 的字节分组（按需填充最后一个分组），然后调整排序并用函数 NH 对每一个分组进行杂凑。将结果串联形成一个比特串，其长度小于或等于 128 与一个小于原始值的值相乘。

输入： $L1Key$（长度为 1024 字节的比特串）

$L1Msg$（长度小于 2^{67} 字节的比特串）

输出： $H1$（长度为 8 * ceil(bitlength($L1Msg$)/8192)字节的比特串）

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

b） 将 $L1Msg$ 分割成比特串 $M_1, M_2, \cdots, M_t$，满足 $L1Msg=M_1 \| M_2 \| \cdots \| M_t$，并且对于所有的 $1 \leqslant i \leqslant t-1$ 都有 octetlength(M_i)=1 024 成立。

c） $Len=$uint2bitstr(1024 * 8,8)。

d） $H1=$<empty string>。

e） 对于 $i=1$ 到 $t-1$，令：

1） ENDIAN-SWAP(M_i)；

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

f） $Len=$uint2bitstr(bitlength(M_t),8)。

g） $M_t=$zeropad(M_t,32)。

h） ENDIAN-SWAP(M_t)。

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

j） 输出 $H1$。

6.2.7.2 第二层杂凑函数 L2-HASH

第二层利用一个称为 POLY 的多项式杂凑函数将 L1-HASH 的输出重新进行杂凑。若 L1-HASH 的输出较长，则对 L1-HASH 输出的前缀调用一次 POLY，再用不同的设置对其余部分调用。只有当消息长度大于 16 兆字节时，才需要对 L1-HASH 的输出执行这两步杂凑。

注：对 POLY 的谨慎调用是必要的，用以避免可能的时间攻击（更多信息参见[1]）。

输入： $L2Key$（长度为 24 字节的比特串）

$L2Msg$（长度小于 2^{64} 字节的比特串）

输出： $H2$（长度为 16 字节的比特串）

a） $Mask64=$uint2bitstr(0x 01FFFFFF 01FFFFFF, 8)。

b） $Mask128=$uint2bitstr(0x 01FFFFFF 01FFFFFF 01FFFFFF 01FFFFFF, 16)。

c） $k64=$bitstr2uint($L2Key[1 \cdots 8] \wedge Mask64$)。

d） $k128=$bitstr2uint($L2Key[9 \cdots 24] \wedge Mask128$)。

e） 若(octetlength($L2Msg$)$\leqslant 2^{17}$)，则：

$y=\mathrm{POLY}(64, 2^{64}-2^{32}, k64, L2Msg)$。

f) 其他情况：

1) $M_1 = L2Msg[1 \cdots 2^{17}]$；

2) $M_2 = L2Msg[2^{17}+1 \cdots \text{octetlength}(L2Msg)]$；

3) $M_2 = \text{zeropad}(M_2 \parallel \text{uint2bitstr}(0\text{x}80,1),16)$；

4) $y = \text{POLY}(64, 2^{64}-2^{32}, k64, M_1)$；

5) $y = \text{POLY}(128, 2^{128}-2^{96}, k128, \text{uint2bitstr}(y,16) \parallel M_2)$。

g) $H2 = \text{uint2bitstr}(y,16)$。

h) 返回 $H2$。

6.2.7.3 第三层杂凑函数 L3-HASH

L2-HASH 的输出是长度为 16 的字节串。作为最后一层杂凑函数，L3-HASH 将长度为 16 的字节串杂凑成一个固定长度为 4 的字节串。

输入： $K1$(长度为 64 字节的比特串)
$K2$(长度为 4 字节的比特串)
Msg(长度为 16 字节的比特串)

输出： $H3$(长度为 4 字节的比特串)

a) $y=0$。

b) 将 Msg 和 $K1$ 分割成 8 个分组并转化成整数：

对于 $i=1 \sim 8$：

1) $M_i = Msg[(i-1)*2+1 \cdots i*2]$；

2) $K_i = K1[(i-1)*8+1 \cdots i*8]$；

3) $m_i = \text{bitstr2uint}(M_i)$；

4) $k_i = \text{bitstr2uint}(K_i) \bmod \text{prime}(36)$。

c) 内积杂凑，提取最后 32 个比特做仿射变换：

1) $y = (m_1 * k_1 + \cdots + m_8 * k_8) \bmod \text{prime}(36)$；

2) $y = y \bmod 2^{32}$；

3) $H3 = \text{uint2bitstr}(y,4)$；

4) $H3 = H3 \oplus K2$。

d) 输出 $H3$。

6.2.8 终止化操作

输入： 加密密钥 K_E(长度为 $taglen$ 字节的比特串)
杂凑值 H(长度为 $taglen$ 字节的比特串)

输出： 消息鉴别码 MAC(长度为 $taglen$ 字节的比特串)

a) $\text{MAC} = K_E \oplus H$。

b) 输出 MAC。

6.3 Badger

6.3.1 Badger 的描述

Badger 是一个 MAC 算法，它使用一个 128 比特的密钥 K 和一个 128 比特的临时值 N。它可以将一个长度小于或等于 $2^{61}-1$ 个字节的消息处理成一个长度为 $taglen$ 的鉴别标签，其中 $taglen$ 可以取 4 个、8 个、12 个、16 个或 20 个字节。输入的消息由整数个字节构成(也就是说，消息的比特长度应该

是 8 的倍数)。

A.2 中列举了在底层采用 ZUC 算法的 Badger 所生成的测试向量。

注：Badger 在参考文献[3]中被提出，以由 Wegman 和 Carter 建立的经典杂凑树[11]为基础。

6.3.2 要求

在使用 Badger 之前，需要首先协商以下参数：

——序列密码算法或伪随机生成器算法。

——标签长度，$taglen$，应为 4 个、8 个、12 个、16 个或 20 个字节之一。

6.3.3 辅助函数 ENH

ENH(增强的非线性杂凑函数)是一个泛杂凑函数。

注：增强的非线性杂凑函数 ENH 由 Boesgaard 等人提出[3]。它基于 Black 等人提出的非线性杂凑函数 NH 建立，NH 应用在 UMAC 中。

输入：　密钥 $LKey$(长度为 8 字节的比特串)
　　消息 $Left$(长度为 8 字节的比特串)
　　消息 $Right$(长度为 8 字节的比特串)

输出：　杂凑值 $LHash$(长度为 8 字节的比特串)

a) $k_L=\mathrm{octetstr2uint}(LKey[0\cdots3])$，$k_U=\mathrm{octetstr2uint}(LKey[4\cdots7])$。

b) $m1_L=\mathrm{octetstr2uint}(Right[0\cdots3])$，$m1_U=\mathrm{octetstr2uint}(Right[4\cdots7])$ 。

c) $m2=\mathrm{octetstr2uint}(Left[0\cdots7])$。

d) $h_L=(m1_L+k_L)\mathrm{mod}2^{32}$。

e) $h_U=(m1_U+k_U)\mathrm{mod}2^{32}$。

f) 令 $h'=((h_U*h_L)+m2)\mathrm{mod}2^{64}$。

g) $LHash=\mathrm{uint2octetstr}(h',8)$。

h) 输出 $LHash$。

6.3.4 密钥预处理

Badger 使用的密钥长度为 16 个字节，临时值的长度为 16 个字节。若生成器需要较长的密钥或临时值，则其余的字节可以用零进行填充。临时值应与全 1 向量的值不相同。假设该 PRG 具有以下接口：

——PRG_Init(K，N)使用密钥 K 和临时值 N 将 PRG 的内部状态初始化。

——PRG_Next(n)从 PRG 产生下一组长度为 n 比特的输出。

利用这些函数，杂凑密钥和加密密钥可以按如下步骤进行计算：

注：如果有若干个消息需要进行鉴别，则杂凑密钥 K_H 将被重复使用，所以缓存该密钥是有必要的。只有加密密钥 K_E 在每次处理新消息时被重新计算。

输入：　主密钥 K(长度为 16 字节的比特串)
　　临时值 N(长度为 16 字节的比特串)
　　所有可能输入的消息的最长比特长度 $maxlen$，满足 $0\leqslant\mathrm{max}len\leqslant2^{64}-8$ 并且是 8 的整数倍
　　标签长度 $taglen$(整数 4、8、12、16、20 之一)

输出：　杂凑密钥 $K_H=(KL,kf)$(任意长度的比特串，其中 KL 是一个由长度为 8 字节的比特串构成的向量，kf 是一个由长度为 4 字节的整数构成的向量)

加密密钥 K_E(长度为 $taglen$ 字节的比特串)

a) PRG_Init(K,1^{128})。

b) $words\_used=0$。

c) $u=taglen/4$。

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

e) 对于 $j=1$ 到 6:

对于 $i=1$ 到 u:

1) $kf_{j,i}=\mathrm{octetstr2uint}(\mathrm{PRG\_Next}(32))$;

2) $words\_used=words\_used+1$。

f) 对于 $j=1$ 到 6:

对于 $i=1$ 到 u:

当($kf_{j,i}\geqslant\mathrm{prime}(32)$)时:

1) $kf_{j,i}=\mathrm{octetstr2uint}(\mathrm{PRG\_Next}(32))$;

2) $words\_used=words\_used+1$。

g) 当($words\_used \bmod 4\neq 0$)时:

1) 抛弃 PRG_Next(32);

2) $words\_used=words\_used+1$。

h) 对于 $j=1$ 到 v:

对于 $i=1$ 到 u:

$KL_{j,i}=\mathrm{PRG\_Next}(64)$。

i) PRG_Init(K,N)。

j) $K_E=\mathrm{PRG\_Next}(32*u)$。

k) 输出 $K_H=(KL,kf)$,K_E。

6.3.5 消息预处理

Badger 不需要对消息进行预处理。

6.3.6 消息杂凑

消息的杂凑依照以下多项式表达计算:

输入: 杂凑密钥 $K_H=(KL,kf)$(任意长度的比特串)

消息 M(长度至多为 $2^{61}-1$ 字节的比特串)

标签长度 $taglen$(整数 4、8、12、16、20 之一)

输出: 杂凑值 H(长度为 $taglen$ 字节的比特串)

a) $len=\mathrm{bitlength}(M)$为 64 比特的整数。

b) 若 $len=0$:

$M_1=\cdots=M_u=0^{64}$

c) 其他情况:

1) 若 $len \bmod 64\neq 0$:

在最高比特位添加零比特直到 M 的长度 len 是 64 比特的整数倍。

2) 对于 $i=1$ 到 u:

ⅰ) $M_i=M$;

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

ⅲ) 对于 $j=1$ 到 v':

Ⅰ） $t=\text{octetlength}(M_i)/8$；

Ⅱ） 将 M_i 分割成长度为 8 的字节分组 $B_1,\cdots,B_t$，满足 $M_i=B_t \| \cdots \| B_1$；

Ⅲ） 若 t 为偶数：

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

Ⅳ） 其他情况：

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

d） 对于 $i=1$ 到 u：

1） $Q_i=0^7 \| len \| M_i$；

2） 将 Q_i 分割成长度为 27 比特的分组 $B_1,\cdots,B_5$，满足 $Q_i=B_5 \| \cdots \| B_1$；

3） 将每个分组 $B_1,\cdots,B_5$ 的最高比特位用零填充使其长度为 4 个字节；

4） 对于 $j=1$ 到 5：

$b_j=\text{octetstr2uint}(B_j)$；

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

6） $S_i=\text{uint2octetstr}(s_i,4)$。

e） $H=S_u \| \cdots \| S_1$。

f） 输出 H。

6.3.7 终止化操作

输入： 加密密钥 K_E（长度为 *taglen* 字节的比特串）

杂凑值 H（长度为 *taglen* 字节的比特串）

输出： 消息鉴别码 MAC（长度为 *taglen* 字节的比特串）

a） $\text{MAC}=K_E \oplus H$。

b） 输出 MAC。

6.4 Poly1305

6.4.1 Poly1305 的描述

Poly1305 是一个 MAC 算法，它使用一个 256 比特的密钥 K（其中有 22 比特设为零）和一个 128 比特的临时值。该算法可以处理任意字节长度 l 的消息，生成 128 比特的 MAC。输入的消息应包含整数倍个字节，也就是说消息的比特长度应该是 8 的整数倍。

A.3 中列举了在底层采用 SM4 算法的 Poly1305 所生成的测试向量。

注：Poly1305 在参考文献[1]中被提出，以多项式杂凑为基础。相较于最一般的实现，依照[1]中给出的建议可以使 Poly1305 的性能得到显著提高。

6.4.2 要求

使用 Poly1305 无须协商任何额外参数。

6.4.3 密钥预处理

长度为 32 的字节主密钥 K 具有特殊的格式，个别的比特位应为零。这些比特位包括：

——$K[3]$，$K[7]$，$K[11]$，$K[15]$的最高 4 个比特位。

——$K[4]$，$K[8]$，$K[12]$的最低 2 个比特位。

主密钥之后直接分割成一个杂凑密钥和一个加密密钥，如下：

输入： 主密钥 K（长度为 32 字节的比特串）

输出：　　杂凑密钥 K_H（长度为 16 字节的比特串）
　　加密密钥 K_E（长度为 16 字节的比特串）

a) $K_H = K[0 \cdots 15]$。

b) $K_E = K[16 \cdots 31]$。

c) 输出 K_H, K_E。

6.4.4 消息预处理

消息按如下方式进行预处理：

输入：　　消息 M（长度为 l_0 字节的比特串）

输出：　　消息分组的个数 s（整数）
　　经过预处理的消息 $c_1, \cdots, c_s$（长度为 17 字节整数的序列）

a) 令 $l_0 = \mathrm{octetlength}(M)$。

b) 令 $s = \mathrm{ceil}(l_0/16)$。

c) 令 $t = \mathrm{floor}(l_0/16)$。

d) 对于 $i = 0, \cdots, t-1$：
　　$c_{i+1} = \mathrm{octetstr2uint}(M[16i \cdots 16i+15]) + 2^{128}$。

e) 若 $s > t$：

　　1) 令 $r = l_0 \mathrm{mod} 16$。

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

f) 输出 $s; c_1, \cdots, c_s$。

6.4.5 消息杂凑

消息的杂凑依照以下多项式表达计算：

输入：　　杂凑密钥 K_H（长度为 16 字节的比特串）
　　消息分组的个数 s（整数）
　　经过预处理的消息 $c_1, \cdots, c_s$（长度为 17 字节整数的序列）

输出：　　杂凑值 H（长度为 16 字节的比特串）

a) $r = \mathrm{octetstr2uint}(K_H)$。

b) $H' = (c_1 * r^s + c_2 * r^{s-1} + \cdots + c_s * r^1)\mathrm{modprime}(130)$。

c) 令 $H = H' \mathrm{mod} 2^{128}$。

d) 输出 H。

6.4.6 终止化操作

最后，利用分组密码算法对消息进行加密。

输入：　　杂凑值 H（长度为 16 字节的比特串）
　　加密密钥 K_E（长度为 16 字节的比特串）
　　临时值 H（长度为 16 字节的比特串）

输出：　　消息鉴别码 MAC（长度为 16 字节的比特串）

a) 令 $S = \mathrm{Enc}(K_E, N)$。

b) $s = \mathrm{octetstr2uint}(S)$。

c) 令 $mac = (H + s)\mathrm{mod} 2^{128}$。

d) $\mathrm{MAC} = \mathrm{uint2octetstr}(mac, 16)$。

e) 输出 MAC。

6.5 GMAC

6.5.1 GMAC 的描述

GMAC 与分组长度为 128 比特的分组密码算法配合使用，得到 MAC 的长度为 t 比特，其中 t 是 8 的倍数并且满足 $96 \leqslant t \leqslant 128$（在特定场合下，$t=32$ 和 $t=64$ 仍允许使用）。输入消息的长度应小于或等于 2^{64} 分组。

GMAC 特指 GB/T 36624—2018 中指定的 GCM（指 Galois/Counter Mode）在没有需要加密的数据时的特殊情况。GCM 由 McGrew 和 Viega 提出[9]。

A.4 中列举了在底层采用 SM4 算法的 GMAC 所生成的测试向量。

6.5.2 要求

在使用 GMAC 之前，需要首先协商以下参数：

——分组长度为 128 比特的分组密码算法。分组密码算法的选择决定密钥长度 $|K|$。

——标签（按比特）长度 t，其中 $96 \leqslant t \leqslant 128$（在特定场合下，$t=32$ 和 $t=64$ 仍允许使用，但是需谨慎地参考文献[10]的附录 C）。

——临时值的长度。

6.5.3 辅助函数 GHASH

函数 GHASH 以长度为 128 比特的分组 H 和两个任意长度的比特串 W 与 Z 为输入，输出一个长度为 128 比特的分组。

输入：　长度为 128 比特的分组 H
　　任意长度的比特串 W 和 Z

输出：　长度为 128 比特的值 X_{k+l+1}

a) 令 k 和 u 为唯一确定的整数，满足：$\mathrm{bitlength}(W)=128(k-1)+u$ 并且 $0<u\leqslant 128$。令 $W_1, W_2, \cdots, W_k$ 为分解 W 得到的长度为 128 比特的分组序列（可能存在例外，即 W_k 包含 W 的最后 u 比特）。

b) 令 l 和 v 为唯一确定的整数，满足：$\mathrm{bitlength}(Z)=128(l-1)+v$ 并且 $0<v\leqslant 128$。令 $Z_1, Z_2, \cdots, Z_l$ 为分解 Z 得到的长度为 128 比特的分组序列（可能存在例外，即 Z_l 包含 Z 的最后 v 比特）。

c) 依照以下递归计算长度为 128 比特的值 X_{k+l+1}：

1) $X_0=0^{128}$；

2) $X_i=(X_{i-1}\oplus W_i)\cdot H$　　$1\leqslant i\leqslant k-1$（若 $k\leqslant 1$ 则略去该步骤）；

3) $X_k=(X_{k-1}\oplus(W_k \parallel 0^{128-u}))\cdot H$　　（若 $k=0$ 则略去该步骤）；

4) $X_i=(X_{i-1}\oplus Z_{i-k})\cdot H$　　$k+1\leqslant i\leqslant k+l-1$（若 $l\leqslant 1$ 则略去该步骤）；

5) $X_{k+1}=(X_{k+l-1}\oplus(Z_l \parallel 0^{128-v}))\cdot H$　　（若 $l=0$ 则略去该步骤）；

6) $X_{k+l+1}=(X_{k+l}\oplus \mathrm{uint2bitstr}(\mathrm{bitlength}(W),8) \parallel \mathrm{uint2bitstr}(\mathrm{bitlength}(Z),8))\cdot H$。

d) 输出 X_{k+l+1}。

6.5.4 密钥预处理

通过如下方式，使用主密钥 K 派生杂凑密钥 K_H 和加密密钥 K_E：

输入：　主密钥 K

输出：　杂凑密钥 K_H

加密密钥 K_E

a) $K_H = \mathrm{Enc}(K, 0^{128})$。

b) $K_E = K$。

c) 输出 K_H, K_E。

6.5.5 消息预处理

GMAC 中无需对消息进行预处理。

6.5.6 消息杂凑

输入：消息 M

杂凑密钥 K_H

输出：杂凑值 H

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

b) 输出 H。

6.5.7 终止化操作

选取任意长度的临时值 N。对于需要保护的不同消息，该临时值应互不相同，并且可以由消息的接收方获知。该值无需是不可预测或秘密的。

注：N 可以由消息发送方通过一个计数器生成，然后以明文的形式与受保护的消息一同发送。

输入：杂凑值 H

加密密钥 K_E

临时值 N

输出：消息鉴别码 MAC(长度为 t 位的比特串)

a) 若 $\mathrm{bitlength}(N) = 96$，则令 $Y_0 = N \| 0^{31} \| 1$。否则，令 $Y_0 = \mathrm{GHASH}(K_H, \{\}, N)$。

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

c) 输出 MAC。

附 录 A
（资料性附录）
测 试 向 量

A.1 UMAC

本条包括若干 UMAC 的测试向量，这里使用 GB/T 32907—2016 中规定的 SM4 算法作为其底层的分组密码算法。表 A.1 列举了使用 16 字节密钥 K 和 8 字节临时值 N 时 UMAC 生成的标签。

K ="abcdefghijklmnop"

N ="bcdefghi"

表 A.1 UMAC 测试向量

消息	UMAC-32	UMAC-64	UMAC-96	UMAC-128
〈empty〉	330d0fde	92a7ab5a4db03535	5e72819955fc948b79aa5a1a	5e72819955fc948b79aa5a1a53d8fdf6
$'a' * 3$	e80d10e6	49a7b462dd820446	85729ea1c5cea5f8697120fb	85729ea1c5cea5f8697120fb46cb5ff4
$'a' * 2^{10}$	28e39d7f	894939fbecda9bb5	459c1338f4963a0bd1428ea6	459c1338f4963a0bd1428ea69dad30f5
$'a' * 2^{15}$	d67dfc5a	77d758de45be2be8	bb02721d5df28a56401bef4b	bb02721d5df28a56401bef4b9f308025

A.2 Badger

本条包括若干 Badger 的测试向量，这里使用 GB/T 33133.1—2016 中规定的 ZUC 算法作为其底层的序列密码算法。表 A.2 列举了使用以下密钥 K 和 IV 时 Badger 生成的标签。

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 08 09 0a 0b 0c 0d 0e 0f

表 A.2 Badger 测试向量

消息	Badger 标签
〈empty〉	77 65 d7 8f 3a 69 ab e3 c0 70 60 61 f6 ae 99 2d
00	03 a1 c2 6a 05 e9 1e 42 68 c5 c5 9a 62 47 16 9e
01	6f 39 bb 1d 9d 41 15 b7 a0 e9 ab 56 0a b9 70 44
00 01 02 03 04 05 06 07 08	55 ff eb df 82 27 9e 74 b9 a7 17 70 57 bd 13 ab

A.3 Poly1305

本条包括若干 Poly1305 的测试向量，这里使用 GB/T 32907—2016 中规定的 SM4 算法作为其底层的分组密码算法。参见表 A.3。

表 A.3　Poly1305 测试向量

测试向量	输入	内容
测试向量＃1	消息	〈empty〉
	加密密钥 K_E	75 de aa 25 c0 9f 20 8e 1d c4 ce 6b 5c ad 3f bf
	杂凑密钥 K_H	a0 f3 08 00 00 f4 64 00 d0 c7 e9 07 6c 83 44 03
	临时值 N	61 ee 09 21 8d 29 b0 aa ed 7e 15 4a 2c 55 09 cc
	Poly1305 标签	15 30 55 7e 5d a6 ad 58 3e 34 cb 41 3a b9 f3 d4
测试向量＃2	消息	f3 f6
	加密密钥 K_E	ec 07 4c 83 55 80 74 17 01 42 5b 62 32 35 ad d6
	杂凑密钥 K_H	85 1f c4 0c 34 67 ac 0b e0 5c c2 04 04 f3 f7 00
	临时值 N	fb 44 73 50 c4 e8 68 c5 2a c3 27 5c f9 d4 32 7e
	Poly1305 标签	ab 51 6f 5d 11 cc f3 3e 18 43 21 ed c8 75 7b 22
测试向量＃3	消息	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
	加密密钥 K_E	6a cb 5f 61 a7 17 6d d3 20 c5 c1 eb 2e dc dc 74
	杂凑密钥 K_H	48 44 3d 0b b0 d2 11 09 c8 9a 10 0b 5c e2 c2 08
	临时值 N	ae 21 2a 55 39 97 29 59 5d ea 45 8b c6 21 ff 0e
	Poly1305 标签	c0 be 41 5f b7 48 bc 07 96 d0 cb 83 a5 c4 60 e4
测试向量＃4	消息	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
	加密密钥 K_E	e1 a5 66 8a 4d 5b 66 a5 f6 8c c5 42 4e d5 98 2d
	杂凑密钥 K_H	12 97 6a 08 c4 42 6d 0c e8 a8 24 07 c4 f4 82 07
	临时值 N	9a e8 31 e7 43 97 8d 3a 23 52 7c 71 28 14 9e 3a
	Poly1305 标签	af ee 4c 3c c1 bf d5 89 6d 2e 7f 44 6a 0f f8 a1

A.4　GMAC

本条包括若干 GMAC 的测试向量，这里使用 GB/T 32907—2016 中规定的 SM4 算法作为其底层的分组密码算法。参见表 A.4。

表 A.4　GMAC 测试向量

测试向量	输入	内容
测试向量＃1	消息	〈empty〉
	密钥 K	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
	临时值 N	00 00 00 00 00 00 00 00 00 00 00 00
	GMAC 标签	23 2f 0c fe 30 8b 49 ea 6f c8 82 29 b5 dc 85 8d
测试向量＃2	消息	fe ed fa ce de ad be ef fe ed fa ce de ad be ef
	密钥 K	fe ff e9 92 86 65 73 1c 6d 6a 8f 94 67 30 83 08
	临时值 N	ca fe ba be fa ce db ad de ca f8 88
	GMAC 标签	9d 63 25 70 f9 30 64 26 4a 20 91 8e 30 81 b4 cd
测试向量＃3	消息	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
	密钥 K	fe ff e9 92 86 65 73 1c 6d 6a 8f 94 67 30 83 08
	临时值 N	ca fe ba be fa ce db ad de ca f8 88
	GMAC 标签	1e ea eb 66 9e 96 bd 05 9b d9 92 91 23 03 0e 78

附　录　B
（资料性附录）
泛杂凑函数的安全性信息

针对采用泛杂凑函数的 MAC 算法的若干攻击在[5]中有描述。相对于其他类型的 MAC 算法，极少量的伪造就可以构成针对采用泛杂凑函数的 MAC 算法的密钥恢复攻击，并最终导致安全性完全崩溃。

为抵御这些攻击，强烈建议实施以下一项或多项对策：

a）增强安全性等级（例如，*taglen*），从而使得第一次伪造在实际应用中是不可行的。

b）有计划地更新 MAC 算法使用的整个密钥。如果可能，使用者甚至应该为每个消息更新密钥。注意到，不仅是在单一密钥下处理的消息个数应该被限制，在同一密钥下处理的消息分组的总和也应该被限制。

c）发送方与接收方需要保证/验证在相同的 MAC 密钥下使用的临时值是否唯一。当临时值重复使用时，算法的安全性将严重降低。

d）接收方应该将大量的失败的 MAC 鉴别作为一种攻击企图进行检测，并对这种情况恰当处理。

附 录 C
（资料性附录）
ZUC 和 SM4 算法的抗攻击能力

ZUC(祖冲之序列密码算法)于 2011 年正式进入 3GPP LTE 国际标准,并于 2012 年成为国家密码行业标准 GM/T 0001—2012,于 2016 年成为国家标准 GB/T 33133.1—2016。ZUC 算法能够抵抗常见的各种序列密码攻击方法,目前尚无有效攻击出现。

SM4(原 SMS4)分组密码算法于 2006 年由国家商用密码管理办公室正式发布,并于 2012 年成为密码行业标准 GM/T 0002—2012,于 2016 年成为国家标准 GB/T 32907—2016。自发布以来,SM4 的安全强度经受住了实践的考验。在密码分析领域,目前对 SM4 最好的分析结果是 23 轮的差分分析(时间复杂度 $2^{126.7}$,数据复杂度 2^{118}[12])和 23 轮的线性分析(时间复杂度 2^{122},数据复杂度 $2^{126.54}$[13])。考虑到 SM4 全轮总共有 32 轮迭代,所以 SM4 尚有充足的安全冗余。

本部分规定的 4 种采用泛杂凑函数的 MAC 算法:UMAC、Badger、Poly1305 和 GMAC,在附录 A 中的测试向量分别使用了 GB/T 32907—2016 中规定的分组密码算法 SM4 和 GB/T 33133.1—2016 中规定的序列密码算法 ZUC 作为其底层的密码算法。在其底层算法 ZUC 和 SM4 具备伪随机特性的基础上,本部分规定的 4 种采用泛杂凑函数的 MAC 算法都具备可证明安全的理论分析保障,能够为各种现实应用提供高效安全的消息鉴别服务。

参 考 文 献

[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] GB/T 17964—2008 信息安全技术 分组密码算法的工作模式

[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] National Institute of Standards and Technology. NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode(GCM) and GMAC. November 2007.

[11] 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.

[12] Bozhan Su, Wenling Wu, and Wentao Zhang. Security of the sms4 block cipher against differential cryptanalysis. Journal of Computer Science and Technology, 26(1):130-138, 2011.

[13] Mingjie Liu, Jiazhe Chen. Improved Linear Attacks on the Chinese Block Cipher Standard. Journal of Computer Science and Technology. November 2014, Volume 29, Issue 6, pp. 1123-1133.