doi: 10.7690/bgzdh.2020.05.003

SM3 算法在导弹数据数字化登记系统中的应用

丛林虎，方　轶
（海军航空大学岸防兵学院，山东 烟台 264001）

摘要：为解决目前部队在导弹数据记录、存储中经常出现的数据篡改、描涂、字迹不清等问题，设计一种基于 Hash 函数的导弹数据数字化登记系统。介绍 Hash 函数验证数据完整性的原理，对比几种常用的 Hash 函数，介绍我国自主研发的 SM3 算法，对其实现效率进行改进，设计导弹数据完整性验证方案，并使用 C#语言进行导弹数据数字化登记系统的开发。测试结果表明：该系统能有效检验数据是否完整，且具有可行性与可推广性。

关键词：Hash 函数；SM3 算法；数据完整性；导弹数据
中图分类号：TJ760　　**文献标志码：**A

Application of SM3 Algorithm in the Digital Registration of Missile Data

Cong Linhu, Fang Yi
(*College of Coastal Defense*, *Navy Aviation University*, *Yantai* 264001, *China*)

Abstract: In order to solve these problems such as data tampering, tracing and handwriting is not clear that often occur in the missile data recording and storage, to design a digital data registration system of missile based on Hash function. This paper introduces the principle of Hash function validation data integrity, compares of several common Hash functions, describes SM3 algorithm which designs by our country, improve the efficiency of its implementation, design missile data integrity verification plan, and use C # language to research and develop the missile data digital registration system. The test results show that the system can effectively test data integrity, and has the feasibility and extension.

Keywords: Hash function; SM3 algorithm; data integrity; missile data

0　引言

导弹的各种数据是反映导弹状态、性能的重要依据，也是导弹履历的重要组成部分。目前部队采用的导弹数据记录、存储方式仍然是手工纸质记录、存储。在数据的记录过程中，该方式难免会因为各种原因出现后期描涂修改数据的情况，甚至丢失原始数据，降低数据的可信度，为后续数据的统计、分析等工作带来不便。判断数据是否被篡改，即验证数据完整性非常有必要。

Hash 函数是一种能将任意长的消息构造成定长数据的函数，属于密码学 3 大类加密算法之一[1]。散列函数一般具有抗碰撞性和单向性，抗碰撞性又分为弱抗碰撞性和强抗碰撞性。弱抗碰撞性是指难以找到和该消息具有相同散列值的另外一条消息；强抗碰撞性是指难以找到散列值相同的 2 条不同消息；单向性是指无法通过散列值反推出该消息。由于 Hash 函数的特点，使其广泛应用于数据完整性检测、消息认证、数字签名、伪随机数生成和一次性口令等方面[2]。

笔者对我国国家密码管理局颁布的国密 SM3 散列函数进行了效率优化改进，并基于改进 SM3 算法设计数据完整性方案，最后使用 C#语言对方案进行实现，开发了导弹数据数字化登记系统。

1　Hash 函数

Hash 函数是一类函数的统称，其中包含很多具体算法，如 SHA−1、SHA−256、RIPEMD160、MD5、SM3、SHA−3 等[3]。其中 MD5 算法和 SHA−1 算法的强抗碰撞性已经被攻破，证明其并不安全[4]。

目前被广泛使用的 Hash 算法都是迭代 Hash 算法[5]。迭代 Hash 函数的输入和输出都是比特串。一般比特串 x 的长度记为 $|x|$，比特串 x 和 y 的级联记为 $x \| y$。设 $\{0,1\}^{m+t} \to \{0,1\}^m (t \geqslant 1)$，构造过程一般分为 3 步：1) 用一个公开算法对任意长度的输入消息 X 进行填充，将填充后的比特串记为 Y，满足 $|Y| = 0 (\mathrm{mod}\ t)$。2) 将 Y 分为长度为 t 比特的若干个分组，记为 $Y = Y_0 \| Y_1 \| \cdots \| Y_{L-1}$，其中，$|Y_i| = t (0 \leqslant i \leqslant L-1)$。3) 按以下步骤一次处理每一个消息分组：

$$H_0 = \mathrm{IV}; \tag{1}$$

$$H_{i+1} = f(H_i, Y_i) \quad 0 \leqslant L-1。 \tag{2}$$

收稿日期：2019−12−29；**修回日期：**2020−01−13
基金项目：国家自然科学基金(516054870)
作者简介：丛林虎(1986—)，男，山东人，博士，讲师，从事装备综合保障研究。E-mail: 342743812@qq.com。

其中：IV 为初始链接变量(initial value)；H_i 为中间链接变量；f 为压缩函数或轮函数。IV 的取值必须在杂凑算法说明中定义。填充规则必须保证 2 个不同的消息经过填充预处理后仍然不同，通常将填充前输入消息的长度填充在最后。

2　SM3 算法及其快速实现方案

2.1　SM3 算法

SM3 算法是我国国家密码管理局颁布的一种迭代 Hash 算法标准，其输入和输出都是比特串[6]。SM3 算法的输出长度为 256 比特，消息分组为 512 比特，采用 MD 结构。对于长度小于 2^{64} 比特的消息 X，SM3 算法经过填充和迭代压缩，生成杂凑值，杂凑值长度为 256 比特。具体算法描述[7]如下。

2.1.1　函数定义

SM3 算法中定义了布尔函数 $FF_j(X,Y,Z)$、$GG_j(X,Y,Z)$和置换函数 $P_0(X,Y,Z)$、$P_1(X,Y,Z)$，具体定义如下：

$$FF_j(X,Y,Z)=\begin{cases}X\oplus Y\oplus Z & 0\leqslant j\leqslant 15\\ (X\wedge Y)\vee(X\wedge Z)\vee(Y\wedge Z) & 16\leqslant j\leqslant 63\end{cases};\quad (3)$$

$$GG_j(X,Y,Z)=\begin{cases}X\oplus Y\oplus Z & 0\leqslant j\leqslant 15\\ (X\wedge Y)\vee(\neg X\wedge Z) & 16\leqslant j\leqslant 63\end{cases};\quad (4)$$

$$P_0(X)=X\oplus(X<<<9)\oplus(X<<<17);\quad (5)$$

$$P_1(X)=X\oplus(X<<<15)\oplus(X<<<23)。\quad (6)$$

式中的 X、Y、Z 均为长度为 32 的比特串。

2.1.2　填充

对于一个长度为 l 比特的消息，首先将比特“1”添加到消息的末尾，再添加 k 个“0”，其中 k 是满足 $l+1+k\equiv 448 \bmod 512$ 的最小非负整数，然后添加一个 64 位的长度 l 二进制比特串。经过填充后消息的长度就是 512 的倍数。

2.1.3　迭代压缩

将填充后的消息按每 512 比特的长度为一组进行分组，并对分组后的消息进行编号，得到 $B^{(0)}B^{(1)}\cdots B^{(n-1)}$，其中 $n=(l+k+65)\div 512$。对分组后已填充消息进行如下方式的迭代：

FOR　$i=0$　TO　$n-1$

　　$V^{(i+1)}=CF(V^{(i)},B^{(i)})$

　　END FOR

其中：CF 是压缩函数；$V^{(0)}$是初始值 IV，长度为 256 比特。最后一次迭代后的结果为 $V^{(n)}$。

2.1.4　消息扩展

将消息分组 $B^{(i)}$ 扩展，生成字 $W_0,W_1,\cdots,W_{67},W_0',W_1',\cdots,W_{63}'$，用于 CF 函数。首先将消息分组 $B^{(i)}$划分为 16 个字 $W_0,W_1,\cdots,W_{15}$。根据以下方法扩展：

FOR　$j=16$　TO 67

　　$W_j\leftarrow P_1\left(W_{j-16}\oplus W_{j-9}\oplus\left(W_{j-3}<<<15\right)\right)\oplus\left(W_{j-13}<<<7\right)\oplus W_{j-6}$

END FOR

FOR　$j=0$　TO 63

　　$W_j'=W_j\oplus W_{j+4}$

　　END FOR

2.1.5　压缩函数

压缩函数的计算过程如下：

　　$ABCDEFGH\leftarrow V^{(i)}$

FOR　$j=0$　TO 63

　　$SS1\leftarrow\left((A<<<12)+E+\left(T_j<<<j\right)\right)<<<7$

　　$SS2\leftarrow SS1\oplus(A<<<12)$

　　$TT1\leftarrow FF_j(A,B,C)+D+SS2+W_j'$

　　$TT2\leftarrow GG_j(E,F,G)+H+SS1+W_j$

　　$D\leftarrow C$

　　$C\leftarrow B<<<9$

　　$B\leftarrow A$

　　$A\leftarrow TT1$

　　$H\leftarrow G$

　　$G\leftarrow F<<<19$

　　$F\leftarrow E$

　　$E\leftarrow P_0(TT2)$

END FOR

　　$V^{(i+1)}\leftarrow ABCDEFGH\oplus V^{(i)}$

其中：A,B,C,D,E,F,G,H 是中间变量；SS_1,SS_2,TT_1,TT_2 是字寄存器，且字的存储方式为大端存储。最后输出的 256 比特杂凑值为 $y=ABCDEFGH$。

2.2　SM3 算法的快速实现方案

根据前文对 SM3 算法的描述，压缩函数主要过程是中间变量的赋值、移位运算以及寄存器中数据

的读写，且使用多个中间变量与多个寄存器，读写寄存器中的数据占用了大量的运算时间。可以通过优化中间变量、减少对寄存器中的数据读写次数的方式来达到降低压缩函数运算时间的目的，使用此方法进行改进的同时也不会破坏原有压缩函数结构，保证了结果仍然可靠，同时在对消息进行扩展时也可以进行一些优化改进。具体改进策略如下：

1) 扩展时，原始 SM3 算法对消息直接进行 121 次扩展，使用 122 个中间变量 W_j 和 W_j' 进行存储，即进行了 122 次赋值，并用于之后的压缩函数中。这里可以将消息扩展合并至压缩函数中，在使用时进行消息的字扩展，可以免去许多次赋值操作。在前文 SM3 算法描述的基础上，消息扩展优化方案如下：首先将消息分组 $B^{(i)}$ 划分为 16 个字：$W_0,W_1,\cdots,W_{15}$，并将前 4 个字存储在中间变量 W_0,W_1,W_2,W_3 中，用于压缩函数的第一次计算。

2) 压缩函数中的 SS_1,SS_2,TT_1,TT_2 都是中间变量，因此，需要减少这些中间变量的使用。优化方案主要集中在压缩函数循环体的前 4 步操作中，剩余相同部分不再给出。同时结合消息扩展部分的优化方案，在前文 SM3 算法描述的基础上，压缩函数优化改进后的描述如下：

$ABCDEFGH \leftarrow V^{(i)}$

FOR $j=0$ TO 63

$$TT1 \leftarrow FF_j(A,B,C)+D+\left(\left(\left((A<<<12)+E+\left(T_j<<<j\right)\right)<<<7\right)\oplus\left(A<<<12\right)\right)+\left(W_j\oplus W_{j+4}\right)$$

if $j=0$ TO 15

$$TT2 \leftarrow GG_j(E,F,G)+H+\left(\left((A<<<12)+E+\left(T_j<<<j\right)\right)<<<7\right)+P_1\left(W_{j-16}\oplus W_{j-9}\oplus\left(W_{j-3}<<<15\right)\right)\oplus\left(W_{j-13}<<<7\right)\oplus W_{j-6}$$

else

$$TT2 \leftarrow GG_j(E,F,G)+H+\left(\left((A<<<12)+E+\left(T_j<<<j\right)\right)<<<7\right)+W_j$$

$D \leftarrow C$

$C \leftarrow B<<<9$

……

3) 使用 Visual C#语言对原始 SM3 算法以及改进算法进行实现，运行环境为 CPU Intel Core i7-6700HQ 2.60 GHz，Windows7 SP1，Visual Studio 2012。使用 Stopwatch 方法统计各种运算的耗时，统计结果及 2 种实现方式的效果对比如表 1 所示。

表 1 2 种 SM3 算法实现效果对比

函数与操作	原 SM3 算法/ms	改进 SM3 算法/ms	效率对比/%
压缩函数	1 220	870	提高约 28.6
左移	300	200	提高约 33
布尔	80	90	降低约 12

从表中可知：笔者提出的方案能有效提高 SM3 算法的执行效率，总耗时减少了约 25%。

3 导弹数据数字化登记系统实现

3.1 导弹数据完整性验证方案设计

数据记录人员在导弹数据数字化登记系统上完成数据记录并提交后，系统自动调用笔者提出的改进 SM3 算法计算本次记录数据的 Hash 值，并存储于本地数据库中且不可更改。在对本次数据进行查询时，首先进行数据 Hash 值的验证，即重新计算查询数据的 Hash 值，并与之前提交数据时计算保存的 Hash 值进行比对[8]。根据 Hash 函数的特点，对同一组数据采用同一个 Hash 函数计算出的 Hash 值是相同的，如果比对结果相同，则说明该数据没有被修改过，是原始数据；否则说明该数据曾经被人修改，可以根据情况进行责任追究。具体方案如图 1 所示。

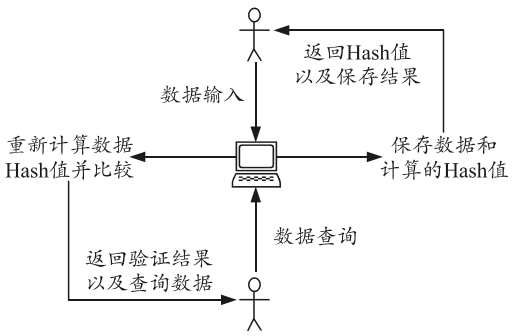

图 1 导弹数据完整性验证方案

3.2 导弹数据数字化登记系统实现

使用 Visual C#语言进行某型航空导弹业务数据数字化登记系统的开发，并应用基于笔者提出的改进 SM3 算法。开发完成后进行功能测试。如图 2 所示，完成数据登记后计算出 Hash 值。如图 3 所示，在查询时比对 Hash 值，并提示用户 Hash 值比对结果。

（下转第 27 页）

读、写、擦除操作，不仅实现了发动机试车数据的存储，而且可随时读取/擦除单次试车数据或指定地址的试车数据，对发动机关键数据的保存与分析具有较为重要的意义。

参考文献：

[1] 刘陵顺, 高艳丽, 张树团, 等. TMS320F28335 DSP 原理及开发编程[M]. 北京: 北京航空航天大学出版社, 2011: 103-118.

[2] 顾卫钢. 手把手教你学 DSP—基于 TMS320X281x[M]. 北京: 北京航空航天大学出版社, 2011: 127-148.

[3] 周世新. 大容量 Flash 与 DSP 接口技术的实现[J]. 无线电工程, 2006, 6(3): 57-58.

[4] 李娟, 王金海, 王敏, 等. DSP 外挂 FLASH 在线编程的研究与实现[J]. EIC, 2009(1): 114-116.

[5] 苏奎峰, 吕强, 常天庆, 等. TMS320X281x DSP 原理及 C 程序开发[M]. 北京: 北京航空航天大学出版社, 2008: 367-375.

****

（上接第 14 页）

图 2　系统生成 Hash 函数界面

图 3　数据完整性验证界面

4　结束语

笔者提出一种基于 Hash 函数的数据完整性验证方案，基于改进的 SM3 算法，设计、实现了导弹数据数字化登记系统。功能试用结果表明：该系统能够完成数据完整性验证，并将验证结果告知用户。下一步，可将数字签名技术应用于该系统，在验证数据完整性的基础上明确数据来源，防止可能出现的代签名导致责任人不明、数据来源不明确等问题。

参考文献：

[1] 赵军, 曾学文, 郭志川. 国产与国外常用杂凑算法的比较分析[J]. 网络新媒体技术, 2018, 7(5): 58-62.

[2] 徐津, 温巧燕, 王大印. 一种基于 Hash 函数和分组密码的消息认证码[J]. 计算机学报, 2015, 38(4): 793-803.

[3] 王小云, 于红波. 密码杂凑算法综述[J]. 信息安全研究, 2015, 1(1): 19-30.

[4] WANG X Y, YU H B. How to break MD5 and other hash functions[C]//Advances in Cryptology EUROCRYPT 2005. Heidelberg: Springer Berlin, 2005: 19-35.

[5] 朱宁龙, 戴紫彬, 张立朝, 等. SM3 及 SHA-2 系列算法硬件可重构设计与实现[J]. 微电子学, 2015, 45(6): 777-780, 784.

[6] 王小云, 于红波. SM3 密码杂凑算法[J]. 信息安全研究, 2016, 2(11): 983-994.

[7] 国家密码管理局. SM3 密码杂凑算法: GM/T0004—2012 [S]. 北京: 国家密码管理局, 2012.

[8] 郭小威, 向哲. 基于蚁群算法的舰载武器共架发射时域协调方法[J]. 兵工自动化, 2019, 38(10): 61-65.