doi:10.11887/j.cn.201706010　　http://journal.nudt.edu.cn

Merkle-Damgård Hash 结构并行扩展算法*

徐劲松[1,2],张民选[2],陈士伟[1],戴紫彬[1]
(1. 信息工程大学 密码工程学院,河南 郑州 450001;2. 国防科技大学 计算机学院,湖南 长沙 410073)

摘　要:利用松弛密码架构将 Merkle-Damgård 结构的 Hash 算法扩展为并行结构的算法,可以利用多核处理器提高杂凑效率。给出的证明过程表明松弛密码架构在处理不同长度的消息时不具备抗碰撞特性。提出的新并行扩展算法基于松弛密码架构设计,该算法弥补了其安全缺陷,并给出了新并行 Hash 结构的安全性分析。分析结果表明新结构抗碰撞特性不低于 Merkle-Damgård 结构的 Hash 算法。实验结果表明,新并行 Hash 结构处理长消息时有较高的处理性能。

关键词:Hash 算法;并行;松弛密码架构;密码分析;性能分析
中图分类号:TP309.7　　**文献标志码**:A　　**文章编号**:1001-2486(2017)06-059-05

Parallel algorithm for extending Merkle-Damgård Hash construction

XU Jinsong[1,2], *ZHANG Minxuan*[2], *CHEN Shiwei*[1], *DAI Zibin*[1]
(1. College of Cryptography Engineering, Information Engineering University, Zhengzhou 450001, China;
2. College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: Relaxed encryption framework which extends hash functions of Merkle-Damgård construction to a parallel construction can improve Hash performance by multi-core processor. A proving process was given to show that relaxed encryption framework has no property of collision resistance when processing messages of different size. A new parallel extending algorithm was proposed base on the design of relaxed encryption framework, which remedies the security flaws of the relaxed encryption framework, and the security of the new parallel Hash construction was also discussed. The cryptanalysis shows that the property of collision resistance of the new parallel construction is not weaker than the hash function of Merkle-Damgård construction. Experimental results indicate that the new Hash construction performs better when processing messages of large size.

Key words: Hash function; parallel; relaxed encryption framework; cryptanalysis; performance analysis

可信计算机在启动和运行时,要确保加载的操作系统、运行的程序以及数据的完整性,需要利用 Hash 算法对这些文件的完整性进行验证。目前常见的 Hash 算法有 MD5,SHA-1,SHA-2,SM3 等,这些 Hash 算法采用 Merkle-Damgård 结构[1-2],是一个串行的运算过程,其算法内部的并行度有限,算法的处理时间与消息的长度呈线性关系,当处理长消息时,需要很长时间,很难满足可信计算机的性能需求。

随着多核处理技术的发展,能够并行处理的 Hash 算法逐渐成为研究热点,出现了许多可并行处理 Hash 算法[3-11],这些并行 Hash 算法通常以现有的 Hash 算法或分组算法为基础,将其扩展为树形[3,5,7]或流水结构[6]等,有效地提高了消息杂凑的速度。这些算法一经提出就得到了广泛关注,并发现某些算法存在安全性问题[4,12-13]。本文对 Gan 提出的并行 Hash 扩展算法[4]以及最近基于此算法设计的 RSHA-1 算法[14]进行了分析,发现了其存在的安全问题,并在其基础上提出了新的并行扩展算法。

1　Gan 的并行扩展算法与 RSHA-1

Merkle-Damgård 结构 Hash 算法如图 1 所示,算法依次对每个消息分组 M_i 进行处理,且将每个分组的处理结果作为处理下一个分组的输入,形成一条数据链,当所有分组处理完成后,最终得到 m bit的 Hash 值 H。

Gan 提出的并行 Hash 扩展算法采用松散加密架构,可将任意串行 Hash 算法扩展为并行结构,该结构采用递归处理,每次递归将消息压缩一定比例,通过多次递归,最终将消息压缩至所需的长度。具体方法是将长消息经填充后拆分为 p 个

* **收稿日期**:2016-09-27
基金项目:国家自然科学基金资助项目(61404175)
作者简介:徐劲松(1977—),男,江苏连云港人,博士研究生,E-mail:pine_xu@sohu.com;
张民选(通信作者),男,教授,博士,博士生导师,E-mail:mxzhang@nudt.edu.cn

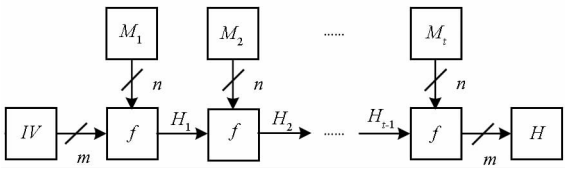


图 1 Merkle-Damgård Hash 结构

Fig. 1 Merkle-Damgård Hash construction

长度为 n 的短消息 M_i,用 Hash 函数对各个短消息 M_i 并行压缩得到 p 个长度为 m 的杂凑值 H_i,将 H_i 的并置作为下次递归的输入进行再次压缩,直到压缩结果等于消息摘要的长度 m。对于填充后长度为 np 的原始消息,经过$\lceil \log_{\frac{n}{m}} p \rceil + 1$ 次递归,得到最终的消息摘要。Kishore 采用此算法,对 SHA-1 算法进行并行化扩展,设计了 RSHA-1 算法[14],见算法 1。

算法 1 RSHA-1 算法

Alg. 1 RSHA-1 algorithm

输入:任意长度消息 M;
输出:160 bit 杂凑值 H;

1. 将消息 M 按 *SHA*-1 算法进行填充,填充得到消息 M';
2. $p = \frac{|M'|}{512}$,将消息 M'拆分成 p 个长度为512 bit的消息分组 $M_1, M_2, \cdots, M_p$;
3. **for** $i = 1$ to p
4. $H_i = f(IV, M_i)$,f 为 SHA-1 算法压缩函数;
5. **end for**
6. $H = H_1 \| H_2 \| \cdots \| H_p$;
7. **if** $|H| > 160$ **then**
8. M = H;
9. **goto** 第 1 行;
10. **else**
11. **return** H;
12. **end if**

RSHA-1 算法经过一次递归,可将消息压缩为输入消息长度的 31.25%(=160/512),通过多次递归压缩,最终得到 160 bit Hash 值。在第 4 行,$H_i = f(IV, M_i)$可并行计算,因此 RSHA-1 算法用多核处理能够提高处理性能。但在安全性方面,经过密码分析,RSHA-1 算法在处理不同长度的消息时会产生碰撞。

命题 1:RSHA-1 算法不具有抗碰撞特性。

证明:即证明对于消息 x 能够找到消息 y,使得 $H(x) = H(y)$,而 $x \neq y$。设原消息 M 经过 k 递归压缩可得到最终 Hash 值,按照算法设计,M 经过第一次递归得到 $H = \mathrm{recur}(M)$,且 $|H| > 160$,其中 recur 为 RSHA-1 算法的 1 ~ 6 行,则 H 需再经过 $k-1$ 次递归压缩得到最终的 Hash 值。因此有:

$$\begin{aligned}\text{RSHA-1}(M) &= \mathrm{recur}^k(M) = \mathrm{recur}^{k-1}[\mathrm{recur}(M)] \\ &= \mathrm{recur}^{k-1}(H) = \text{RSHA-1}(H)\end{aligned} \tag{1}$$

由式(1)可知,消息 M 和消息 H 得到了相同的 Hash 值,而 $M \neq H$,即消息 M 和消息 H 发生了碰撞。

由以上的证明过程不难得到推论 1。

推论 1:若消息 M 经过 k 递归压缩可得到最终 Hash 值,则 RSHA-1 算法前 $k-1$ 次递归生成的 H 与原消息 M 两两产生碰撞。

2 改进的并行 Hash 扩展算法

通过第 1 节的证明以及推论可知:尽管采用松散加密架构的 Hash 算法能够提高密码处理性能,但其并不具有抗碰撞特性,存在安全缺陷。因此在此基础上设计了一种新的并行 Hash 扩展算法来弥补其安全缺陷。

2.1 算法描述

算法 2 在松散加密架构的基础上主要做了三方面的调整:首先,原消息不论长短,都必须拆分为 p 个消息块,即消息块的长度由原始消息的长度以及并行度决定,而不是一个常量;其次,算法消除了递归处理,所有消息经过 2 次杂凑就能得到最终的杂凑值,相当于原算法在 $mp \leqslant n$ 时的特例;最后,算法用标准 Hash 算法替代了 Hash 算法的压缩函数,当发现标准 Hash 算法存在安全缺陷时,便于算法的更换。利用算法 2 得到的并行 Hash 结构如图 2 所示。

算法 2 新并行扩展算法

Alg. 2 New parallel extending algorithm

输入:任意长度消息 M,并行度 p;
输出:杂凑值 H;

1. 将消息 M 拆分成 p 个消息块 $M_1, M_2, \cdots, M_p$;
2. **for** $i = 1$ to p
3. $H_i = f(M_i)$,f 为任意 *Hash* 算法;
4. **end for**
5. $M' = H_1 \| H_2 \| \cdots \| H_p$;
6. **return** $H = f(M')$;

2.2 安全性分析

通过命题 1 的证明过程,不难看出 RSHA-1 算法造成消息 M 和 H 产生碰撞的原因在于将递

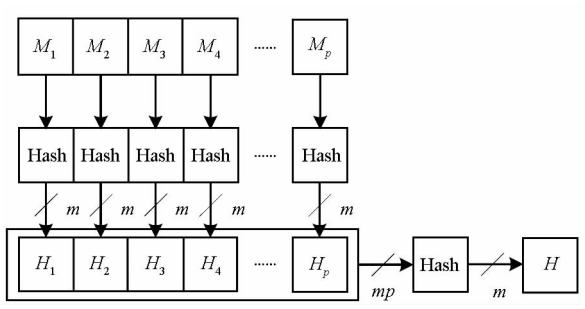


图 2　并行 Hash 结构

Fig. 2　Parallel Hash construction

归的输出作为下次递归的输入。在 RSHA-1 算法中,原消息 M 经填充后划分为若干长度为 n 的短消息,然后并行压缩,可获得较高的并行度。当再次递归时,压缩后的短消息 H 仍划分为若干长度为 n 的消息,其能够获得的并行度必然降低。因此,对于消息 M 和 H,算法相当于在不同的并行度上对长短消息进行处理,导致短消息和长消息之间产生碰撞。

根据上述分析,消除 RSHA-1 在处理消息 M 和 H 时发生的碰撞,可在每次压缩时,将所有消息都按相同的并行度进行处理,以消除命题 1 证明过程中消息替换的过程,即算法 2 对原扩展算法的第一点调整。

然而,不论杂凑消息的长短,Hash 算法总是输出长度为 m 的杂凑值。因此若原算法仅做第一点改进,将导致算法每次递归总是产生长度为 mp 的短消息 H,即原消息在经过一次压缩之后不能再次压缩消息长度。因此需要做第二点调整,消除递归,对短消息 H 直接进行杂凑,得到最终的杂凑值。

通过这两点调整,算法 2 使得命题 1 与推论 1 不再有效,弥补了原算法的安全缺陷。改进后并行 Hash 结构具备以下基本特性与指标。

1)抗碰撞攻击的能力。抗碰撞性是杂凑函数需要满足的基本安全原则之一。对于一个杂凑值长度为 m 的杂凑函数,若找到一对碰撞消息的计算复杂性低于 $O(2^{\frac{m}{2}})$,则称该杂凑函数不能抵抗碰撞攻击。在新并行结构中,先并行计算 p 个消息块的杂凑值,然后将 p 个杂凑值并置,最后对并置后的数据再次杂凑。由于算法内部采用了标准的 Hash 算法,因此该结构可以有效抵抗长度扩展攻击、二次碰撞攻击、不动点攻击,且该结构抵抗碰撞攻击的能力不低于单个杂凑函数抗碰撞攻击的能力。

2)无差别性。无差别性主要考查杂凑函数与随机 Oracle 的不可区分性。Hoch 证明了对于任意函数 α,询问次数低于 $O(2^{\frac{m}{2}})$时,$\alpha[H(M)\oplus G(M)]$ 与随机 Oracle 是无差别的[15],这里的 $H(M)$和 $G(M)$是两个不同的杂凑函数,但输入消息是一样的。在新结构中,杂凑函数是一样的,但输入消息不一定相同,且函数 α 为杂凑函数,故当询问次数低于 $O(2^{\frac{m}{2}})$ 时,该结构与随机 Oracle 是无差别的。

3)抗多碰撞攻击的能力。r 多碰撞是指多个不同的消息产生相同的杂凑值。输出 m bit 的杂凑函数,抗 r 多碰撞的理想计算复杂性为 $O(2^{\frac{m(r-1)}{r}})$。利用生日攻击,找到 n 对碰撞消息 $\{M_i,M_i'\}$ $(1\leqslant i\leqslant n)$,其计算复杂性约为 $O(2^{\frac{h}{2}})$,而这 n 对碰撞消息可以构造出 2^n 个并行杂凑模式的多碰撞,即 $N_1,N_2,N_3,\cdots,N_{2^n},N_i\in\{M_i,M_i'\}$ $(1\leqslant i\leqslant n^2)$产生相同的杂凑值,其计算复杂性为 $O(n2^{\frac{m}{2}})$,低于理想计算复杂性 $O(2^{\frac{(2^n-1)m}{2^n}})$。

2.3　性能分析

设 Hash 算法分组长度为 n,采用 Merkle-Damgård 结构 Hash 算法处理长度为 l 的消息,所需时间为:

$$T_s=\left[\frac{l}{n}+C(l)\right]T_{\mathrm{B}} \tag{2}$$

其中,T_{B} 为 Hash 算法处理一个分组所需的时间,$C(l)$为填充数据所增加的分组。

对于目前大多数 Hash 算法($n=64$ B),其取值范围在 0.140 625 ~ 1.125 之间,因此当消息长度 l 很大时,$C(l)$可忽略不计,此时:

$$T_s=\left[\frac{l}{n}+C(l)\right]T_{\mathrm{B}}\approx\frac{l}{n}T_{\mathrm{B}} \tag{3}$$

同样忽略填充数据对算法性能的影响,算法 2设计的并行 Hash 结构处理长度为 l 的消息,所需时间为:

$$T_p=\frac{l}{np}T_{\mathrm{B}}+\frac{mp}{n}T_{\mathrm{B}}=\left(\frac{l}{np}+\frac{mp}{n}\right)T_{\mathrm{B}} \tag{4}$$

其中,p 为并行度,m 为 Hash 算法输出数据的长度。由此可得新并行 Hash 结构的加速比为:

$$s=\frac{T_s}{T_p}=\frac{\frac{l}{n}}{\frac{l}{np}+\frac{mp}{n}}=\frac{lp}{mp^2+l} \tag{5}$$

图 3 显示了新并行 Hash 结构在处理不同长度消息时,加速比变化的情况($m=20$ B)。

由图 3 可知,当处理的消息很长时,算法加速比接近于算法的并行度 p,当处理的消息较短时,

加速比可能小于 1。

设 $s=\frac{lp}{mp^2+l}>1$,则可得到:

$$l>\frac{mp^2}{p-1} \tag{6}$$

即消息长度满足式(6)时,新结构将能够提高杂凑速度。

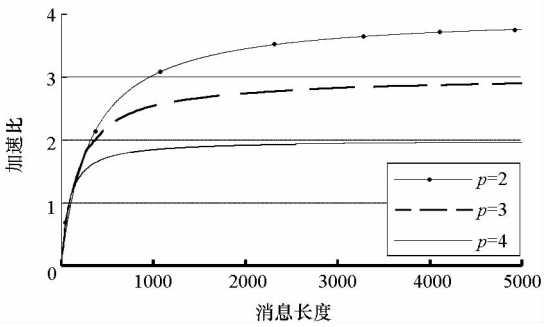


图 3 新结构处理性能

Fig. 3 Performance of new construction

3 实验测试

实验主要考察新并行 Hash 结构在实际应用中的性能。测试的杂凑算法包括标准 SHA-1 算法,以及采用算法 2 对 SHA-1 进行并行扩展后的算法。其中扩展算法的并行度分别选取 2,3 和 4。实验的软硬件环境如下:

处理器:Intel Core i5-3470,4 核处理器,3.20 GHz;

内存:8.00 GB;

操作系统:Windows7 SP1,32 位;

编译环境:Microsoft Visual Studio 2008,OpenMP。

3.1 短消息测试

根据理论分析,新并行 Hash 结构处理短消息时,加速比可能小于 1。对 8 个小文件进行了测试,表 1 记录了对这些文件进行杂凑所需的时间。

表 1 实验结果 1

Tab. 1 Experimental results 1

	SHA-1	$p=2$	$p=3$	$p=4$
64 B	0.000 008	0.000 030	0.000 035	0.000 043
128 B	0.000 009	0.000 031	0.000 036	0.000 043
256 B	0.000 010	0.000 033	0.000 038	0.000 043
512 B	0.000 012	0.000 034	0.000 039	0.000 045
1 KB	0.000 016	0.000 036	0.000 040	0.000 046
2 KB	0.000 023	0.000 042	0.000 040	0.000 048
4 KB	0.000 039	0.000 052	0.000 051	0.000 051
8 KB	0.000 078	0.000 079	0.000 076	0.000 075

测试结果表明,当并行 Hash 结构处理短消息时,存在性能上的损失。当文件长度增加至 8 KB 时,加速比接近于 1。

由算法分析可知:在理想情况下,当 $p=2$ 时,并行扩展后的 SHA-1 算法处理 64 B 消息较原算法没有增加处理时间,当 $p=3$,$p=4$ 时,增加一个分组的处理时间。实际测试结果表明:当 $p=2$ 时,并行扩展后的 SHA-1 算法处理 64 B 消息需要增加 24 μs;当 $p=3$ 时,需要增加 27 μs;当 $p=4$ 时,需要增加 35 μs。并行化后增加的时间远大于一个分组的处理时间(由 SHA-1 的测试结果可知,消息增加一个分组长度,需增加的处理时间小于 1 μs)。由此可知:当并行 Hash 结构处理短消息时,并行区域执行时间较短,OpenMP 开启、关闭线程等开销抵消了并行化改造后的收益,这是利用多核处理不可避免的开销,算法 2 本身不是造成处理短消息时加速比降低的主要因素。

3.2 长消息测试

同样,对 6 个大文件进行了测试,长度分别为 32 MB,64 MB,128 MB,256 MB,512 MB 和1 GB。表 2 记录了对这些文件进行杂凑所需的时间。

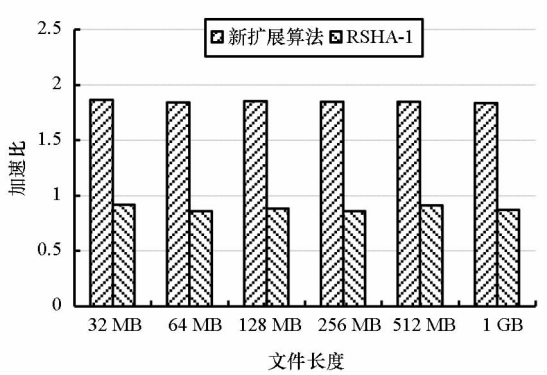
表 2 实验结果 2

Tab. 2 Experimental results 2

	SHA-1	$p=2$	$p=3$	$p=4$
32 MB	0.293 620	0.157 358	0.112 166	0.099 028
64 MB	0.584 113	0.317 113	0.224 237	0.198 560
128 MB	1.172 372	0.632 714	0.447 954	0.391 997
256 MB	2.346 699	1.270 856	0.893 883	0.791 761
512 MB	4.681 760	2.533 936	1.783 803	1.560 241
1 GB	9.388 636	5.111 856	3.568 136	3.122 568

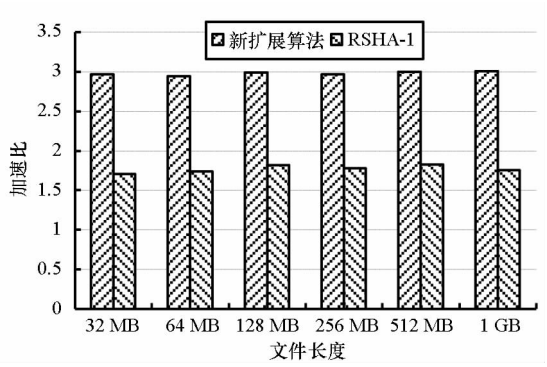
根据表 2 的测试结果可计算在提供的测试环境下新扩展算法对 SHA-1 进行并行扩展获得的加速比:当 $p=2$ 时(2 核),加速比约为 1.85;当 $p=3$时(3 核),加速比约为 2.62。当$p=4$时(4 核),加速比约为 2.99。

根据 RSHA-1 算法[14] 提供的测试结果同样可计算其加速比,由于该文献没有提供 3 核的测试数据,因此仅计算 2 核与 4 核的加速比,计算结果与本文的测试结果如图 4 所示。由图 4 可知,新扩展算法获得的加速比明显高于 RSHA-1 算法的。



(a) 2 核性能

(a) Performance using 2-core



(b) 4 核性能

(b) Performance using 4-core

图 4　性能比较

Fig. 4　Performance comparison

4　结论

并行 Hash 结构消除了 Merkle-Damgård 结构 Hash 算法中消息分组之间的数据相关性,可用多核处理器同时处理多个消息分组,大大提高了消息杂凑的效率。对现有的并行 Hash 结构进行了研究,指出 Gan 提出的基于松散加密架构的并行扩展算法存在一定的安全缺陷。提出新并行扩展算法并对其设计进行了三方面的调整。改进后的算法能够对任意 Hash 算法进行并行化扩展,有足够的安全强度,便于底层算法的更换。在实验环境下,该并行扩展算法用 4 核处理器对长消息进行杂凑,能获得 2.99 左右的加速比,较原算法有明显提高。同时实验结果表明,虽然在理论上算法不适合短消息的处理,但其并不是影响算法性能的主要因素。

参考文献(References)

[1] Damgård I B. A design principle for Hash functions[C]// Proceedings of International Cryptology Conference on Advances in Cryptology, 1989: 416 - 427.

[2] Merkle R C. A certified digital signature[C]// Proceedings on Advances in Cryptology, 1989: 218 - 238.

[3] Sarkar P, Schellenberg P J. A parallel algorithm for extending cryptographic Hash functions [C]//Proceedings of 2nd International Conference on Cryptology, 2001: 40 - 49.

[4] Gan X B, Wang Z Y, Shen L, et al. Parallelizing cryptographic Hash function using relaxed encryption framework [J]. Chinese Journal of Electronics, 2011, 20(4): 621 - 624.

[5] Palash S. Construction of universal one-way hash functions: tree hashing revisited[J]. Discrete Applied Mathematics, 2007, 155(16): 2174 - 2180.

[6] Al-Wesabi O, Samsudin A, Abdullah N. Fast hashing function based on multi-pipeline hash construction (MPHC)[J]. International Journal of Innovative Computing, Information and Control, 2012, 8(11): 7887 - 7907.

[7] Matsuo T, Kurosawa K. On parallel Hash functions based on block-cipher[C]// Proceedings on Australasian Conference on Information Security and Privacy, 2003: 510 - 521.

[8] Li Y T, Xiao D, Li H Q, et al. Parallel chaotic Hash function construction based on cellular neural network[J]. Neural Computing and Applications, 2012, 21(7): 1563 - 1573.

[9] Du M K, He B, Wang Y, et al. Parallel hash function based on block cipher [C]// Proceedings on International Conference on E-Business and Information System Security, 2009: 1 - 4.

[10] Li Y T, Xiao D, Deng S J, et al. Parallel Hash function construction based on chaotic maps with changeable parameters[J]. Neural Computing and Applications, 2011, 20(8): 1305 - 1312.

[11] Wang Y, Wong K W, Xiao D. Parallel hash function construction based on coupled map lattices [J]. Communications in Nonlinear Science and Numerical Simulation, 2011, 16(7): 2810 - 2821.

[12] Wang X M, Guo W, Zhang W F, et al. Cryptanalysis and improvement on a parallel keyed hash function based on chaotic neural network [J]. Telecommunication Systems, 2013, 52(2): 515 - 524.

[13] Wang X Y, Zhao J F. Cryptanalysis on a parallel keyed hash function based on chaotic neural network [J]. Neurocomputing, 2010, 73(16/17/18): 3224 - 3228.

[14] Kishore N, Kapoor B. An efficient parallel algorithm for hash computation in security and forensics applications [C]// Proceedings on IEEE International Advance Computing Conference, 2014: 873 - 877.

[15] Hoch J J, Shamir A. On the strength of the concatenated Hash combiner when all the Hash functions are weak[C]// Proceedings on International Colloquium on Automata, Languages, and Programming, 2008: 616 - 630.