计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190100027

基于 FPGA 的 7-Zip 加密文档高能效口令恢复方法

陈晓杰[1]　周清雷[1]　李　斌[1,2]
1 郑州大学信息工程学院　郑州 450001
2 解放军信息工程大学数学工程与先进计算国家重点实验室　郑州 450001
(740248499@qq.com)

摘　要　随着 7-Zip 压缩软件的广范使用,破解 7-Zip 加密文档的口令对信息安全有着非常重要的意义。目前,破解 7-Zip 加密文档主要采用 CPU 和 GPU 平台,而潜在的口令空间大,计算复杂度高,在有限的时间内找到正确的口令需要更高性能的计算平台。因此,文中通过分析解密算法的 PMC 特性,采用可重构的 FPGA 硬件计算平台,使用流水线技术来实现数据拼接和 SHA-256 算法,并利用预计算和 CSA 方法优化 SHA-256 算法的关键路径,同时使用双端口 RAM 存储校验数据,从而满足算法的计算需求和存储需求,实现高效能的 7-Zip 解密算法。实验数据表明,文中提出的优化方法能大幅提升 SHA-256 算法的性能,使其吞吐量达到 110.080 Gbps,并且通过多种方法对解密算法进行优化,最终破解 10 位长度口令的速率达到了 10608 个/s,是 CPU 的 226 倍,GPU 的 1.4 倍,且能效比是 GPU 的 8 倍,极大地提升了算法的性能,降低了高功耗需求。

关键词:7-Zip 解密;高能效口令恢复;流水线;可重构;SHA-256;双端口 RAM
中图法分类号　TP309

Energy-efficient Password Recovery Method for 7-Zip Document Based on FPGA

CHEN Xiao-jie[1],ZHOU Qing-lei[1] and LI Bin[1,2]
1 School of Information Engineering,Zhengzhou University,Zhengzhou 450001,China
2 State Key Laboratory of Mathematical Engineering and Advanced Computing,Information Engineering University,Zhengzhou 450001,China

Abstract　With the wide range of 7-Zip compression software,7-Zip password cracking is very important for information security. Currently,cracking 7-Zip encryption documents mainly uses CPU and GPU platforms,and the potential for a large password space and high computational complexity requires a higher performance computing platform to find the correct password within a limited time. Therefore,by analyzing PMC characteristics of decryption algorithm,this paper adopted reconfigurable FPGA hardware computing platform,uses pipeline technology to realize data splicing and SHA-256 algorithm,used precomputation and CSA method to optimize the key path of SHA-256 algorithm,and used dual-port RAM to store verification data,thus satisfying the computational and storage requirements of the algorithm and realizing high-performance 7-Zip decryption algorithm. The experimental data show that the optimization method in this paper can greatly improve the performance of SHA-256 algorithm,making it throughput reach 110.080 Gbps. The decryption algorithm is optimized by various methods,and finally the 10 bit password is cracked to10608 per second,226 times that of the CPU,1.4 times that of the GPU,and 8 times that of the GPU, which greatly improves the performance and reduces the demand for high power consumption.

Keywords　7-Zip decryption,Energy-efficient password recovery,Pipeline,Reconfigurable,SHA-256,Dual port RAM

1　引言

7-Zip 是一款完全免费而且开源的压缩软件,能够支持多种语言且提供强大的压缩性能,适用于多种操作系统。7-Zip 文档加密使用了单向不可逆算法 SHA-256、对称加密算法 AES-256 和数据校验算法 CRC32。目前,哈希算法 SHA-256 仍然具有较高的安全性[1],攻击 SHA-256 函数的方法(如暴力破解、生日攻击和差分攻击)都需要很高的计算性能;AES-256 算法有着严谨的数学结构,目前还没有有效的攻击方式[2]。这两种算法在现有的计算能力下保障了 7-Zip 解密算法的安全性,但是,这也给需要对 7-Zip 进行快速口令恢复的信息安全和计算机取证带来了一定的困难。

到稿日期:2019-01-05　返修日期:2019-04-23　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划项目(2016YFB0800100);国家自然科学基金面上项目(61572444)
This work was supported by the National Key R&D Program of China (2016YFB0800100) and General Program of National Natural Science Foundation of China (61572444).
通信作者:周清雷(ieqlzhou@zzu.edu.cn)

近期有研究发现：口令服从 Zipf 分布，而不是以往假设的均匀分布[3]。这一发现使得在庞大的密码空间中破译离线口令成为了可能。文献[4-6]详细研究了“口令猜测算法”及“口令猜测模型”，并利用社会工程学等方法获得针对性的信息，从而分析出某些高概率口令的特征，或者针对某一用户的高概率口令。利用这些高概率口令可以破译很多用户密码，缩减了密码空间，提高了破译的效率。目前，破译 7-Zip 加密文档的有效方式是高概率口令字典与穷举结合的模式，但随着密码长度的增加，密码空间呈指数倍增长，需要高性能的计算平台进行破解。现阶段主要是基于 CPU 的通用服务器破解，但效率较低；或者基于 OpenCL 架构和 CUDA 架构的 GPU 服务器破解，但由于其受到访存的限制，计算性能受到影响，而且功耗较高。因此，需要具有更高性能的计算平台来提升计算的效率并减少电力的消耗。

经过近 30 年的发展，可重构的计算平台——现场可编程门阵列(Field Programmable Gate Array，FPGA)已成为实现数字系统的主流平台之一，其在高效的密码算法实现、神经网络算法加速、高速可重构双边滤波器等方面得到了较好的应用[7-9]。FPGA 具有较高的计算能力和较低的能耗，可实现高能效的口令恢复算法[10]，将其用于 7-Zip 的密码解密中，利用流水线并行工作的方式，可实现 7-Zip 密码破解算法的高效性。本文的主要工作是基于四核 FPGA 硬件平台，通过使用流水线优化 SHA-256 算法，并利用预计算和 CSA 策略优化关键路径，同时使用一种改进的循环队列实现数据拼接的流水线，以满足 7-Zip 算法的计算需求；再使用双端口 RAM 存储校验数据，并利用 FIFO 进行不同时钟域的数据隔离，从而减少算法的通信需求，满足存储需求，提升算法的性能。

2 算法分析

7-Zip 解密算法可以用一个 6 元组表示：

$$7\text{-}Zip=\{FS,P,\mathrm{DS},\mathrm{SHA\text{-}256},\mathrm{AES\text{-}256},\mathrm{CRC32}\} \quad (1)$$

其中：

FS(Feature String)表示从 7-Zip 加密文档中提取得到的特征串，包括向量值 FS_{iv}、校验数据 FS_{Cdata} 和对比数据 FS_{Vdata}，FS_{Cdata} 的大小由压缩文档大小决定。

P(Password)即为需要验证的密码，也可理解为包含所有可能密码空间的集合。

DS(Data Splicing)是数据拼接，将 P 扩展为宽字节与 4 字节的拼接次数 num 进行组合，再跟上 4 字节零组合成一次拼接，一共需要拼接 524 288 次。

SHA-256 是核心处理算法，运算次数 $count$ 由 DS 决定：

$$count=\frac{524\,288}{64\div(length(P)\times 2+4+4)} \quad (2)$$

其中，$length$ 为获取长度的函数。

AES-256 和 CRC32 是 7-Zip 解密过程对 FS_{Cdata} 进行处理的校验算法。

算法的处理、存储和通信(Processing Memory Communication，PMC)特性是算法的基本属性。因此，分析 7-Zip 解密算法的 PMC 特性对实现整体结构具有指导作用。

1)处理过程：算法的处理过程主要包括特征串提取、密码穷举、一次迭代、二次校验和最终验证 5 个模块，具体流程如算法 1 所示。

算法 1 7-Zip 处理过程

输入：加密 7-Zip 文档

输出：Result

1. FS=Extract(加密压缩文档)；//解析文档，获取数据
2. P=穷举+字典；//根据策略生成密码
3. for(num=0；num<524 288；num++) begin
4. DS(P，num)；
5. if(length(DS)==64)
6. Digest=SHA-256(DS)；//计算、存储
7. end for
8. Key= SHA-256(Digest)；//获取密钥
9. for(j=0；j<length(FS_{Cdata})/128；j++) begin
10. Out=AES-256(Key，FS_{Cdata})；//解密
11. Crc=CRC32(FS_{iv} ^ Out)，FS_{iv}=FS_{Cdata}；//校验
12. end for
13. Result=Match(Crc，FS_{Vdata})；//判断匹配并输出

上述算法中，步骤 1 是按规则从加密压缩文档中提取特征串 FS；步骤 2 是根据穷举和字典策略生成可能的密码集 P；步骤 3－7 是一次迭代，首先进行 DS 运算，然后每 64 byte 进行一次 SHA-256 计算；步骤 8－12 为二次校验，首先获得密钥，然后对每 128 位 FS_{Cdata} 进行 AES-256 运算，输出后再进行 CRC32 校验，直到 FS_{Cdata} 校验完毕；步骤 13 判断当前输入的 P 是否正确，并反馈结果。7-Zip 解密算法都是整型运算，属于计算密集型，要求算法可高度并行，具有较高的计算需求。

2)存储需求：算法的粗粒度存储主要有处理过程中传递的数据，具体需求如表 1 所列。

表 1 7-Zip 算法的存储需求

Table 1 Storage requirements of 7-Zip algorithm

存储类别	存储宽度/bit	存储深度
FS_{iv}	128	1
FS_{Cdata}	32	2048
FS_{Vdata}	32	1
P	160	≥1
DS	512	1
$Digest$	256	≥1

由于不同大小的加密压缩文档的 FS_{Cdata} 不同，因此将其设置为一个较大值。表 1 中 P 需要传递至一次迭代进行运算；$Digest$ 是一次迭代结束后的输出值，需要被二次校验计算得到 Key，P 和 $Digest$ 的深度与一次迭代的计算速度成正比。

3)通信需求：从算法的处理过程中可以看出算法需要对数据进行 3 次处理和传输。首先，需要从加密文档中获取特征串 FS，并且通过策略生成 P；然后，将这两部分传输到一次迭代进行计算，一次迭代的输出传递到二次校验；计算完毕后再一次对结果进行最终验证。

通过分析算法的 PMC 特性，可以发现算法中的数据拼接和 SHA-256 运算有较高的计算需求。7-Zip 解密算法与 WinRAR 解密算法都应用了数据拼接且拼接类似，因此优化 WinRAR 的数据拼接对 7-Zip 有重要的借鉴意义。文献[11]

对 WinRAR 的数据拼接部分进行优化，先利用 CPU 预先生成一个拼接表，再将此表的一部分放到共享内存中，从而减少全局内存开销访问，提升拼接的效率。文献[12]在 GPU 平台下通过增加 3 个数组的方式减少了状态跳变和数据链接的时间，再结合 GPU 本身的优势进行优化，提升了性能。哈希算法 SHA-256 需要大量的计算，如何提升运算效率，对加密、解密、数字签名等领域有重要意义。文献[13]设计了一种 ALSI 结构并在 FPGA 上进行实现，能够根据要求单独灵活地执行 SHA-2 算法，但是其最终频率只有 74 MHz，效率较低。文献[14]在 FPGA 平台上使用 3 种不同的预计算方式优化关键路径，并采用 CSA 对 3 种不同的方式优化加法运算，最终 3 种方案中频率最高为 115.46 MHz。文献[15]使用 CSA 和 CLA 优化关键路径加法运算，减少了资源占用，最终在 FPGA 上实现的频率为 41.97 MHz。文献[16]通过循环展开的方法将 64 轮循环迭代压缩为 32 轮，并且采用预计算、数据预提取、递归优化、加法压缩 CSA 等策略减少了算法的延时，提高了吞吐率，使算法频率达到了 172 MHz。

本文实现 DS 运算的平台与上述研究不同，通过可重构计算平台定制逻辑，从而满足高密度的计算需求，以达到最佳性能。哈希算法 SHA-256 通过全流水线优化方法，还能进一步提高计算的性能。

3 算法优化及实现

在 7-Zip 解密算法的 5 个处理过程中，特征串提取的计算量小、复杂度低，只需要在 CPU 端运行就可以满足计算需求；而其他部分计算需求较大，需要在高性能的 FPGA 中进行计算。因此，结合对算法的 PMC 特性分析，可得到其处理、存储、通信结构，如图 1 所示。

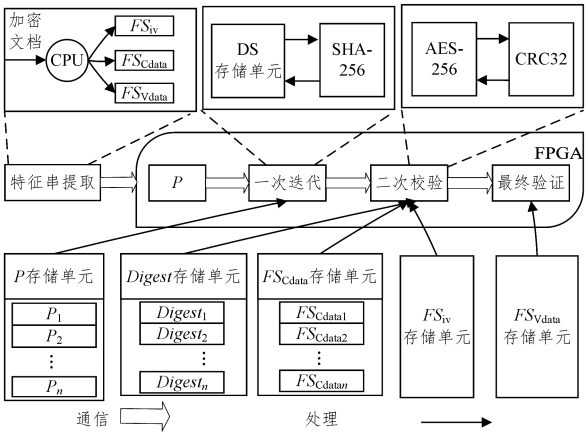


图 1 解密算法的 PMC 结构图

Fig. 1 Decryption algorithm PMC structure diagram

解密算法中计算量最大的是数据拼接和 SHA-256 运算，并且随着密码长度的增加，hash 运算的次数也越多，导致消耗的时间也越多；而 AES-256 和 CRC32 运算量相对较小。因此，对 SHA-256 运算和数据拼接 DS 操作的优化对算法计算性能的提升有着至关重要的作用。

3.1 SHA-256 算法优化

哈希算法 SHA-256 具有比 SHA-1 更高的复杂度和安全性，已被广泛应用于多种信息安全领域。其具体处理过程如下。

1)预处理。将输入数据按规则补位到 512 bit，然后分割为 16 组即 M_0-M_{15}，并初始化向量 K_0-K_{63} 和 h_0-h_7 用于后续计算。

2)循环迭代。将 A,B,C,D,E,F,G,H 赋初值 h_0-h_7，然后进行 64 轮(见图 2(a))更新计算。算法中更新 A 和 E 的值较为复杂，其计算公式如式(3)、式(4)所示：

$$A_{t+1}=H_t+\sum_1(E_t)+Ch(E_t,F_t,G_t)+K_t+W_t+\sum_0(A_t)+Maj(A_t,B_t,C_t) \tag{3}$$

$$E_{t+1}=H_t+\sum_1(E_t)+Ch(E_t,F_t,G_t)+K_t+W_t+D_t \tag{4}$$

其中，$\sum_1(E_t)$，$\sum_0(A_t)$，$Maj(A_t,B_t,C_t)$，$Ch(E_t,F_t,G_t)$是逻辑函数，W_t 更新如下：

$$W_t=\begin{cases}M_t, & 0\leqslant t\leqslant 15\\ \sigma_1(W_{t-2})+W_{t-7}+\sigma_0(W_{t-15})+W_{t-16}, & 16\leqslant t\leqslant 63\end{cases} \tag{5}$$

3)计算最终结果。将 h_0-h_7 与 $A_{63}-H_{63}$ 一一对应做加法运算，最终输出 256 位哈希值。

可以发现算法的耗时主要集中于更新 A 和 E 的值，需要进行多次加法操作，并且要经过 64 轮的循环迭代，因此对这两个操作数的优化对减少算法的延时有很重要的作用。算法的吞吐量决定了算法的计算性能，其计算公式如下[17]：

$$T=\frac{B\times f_{\max}\times N}{d} \tag{6}$$

其中，T 为吞吐量，B 为数据块大小，f 为最大时钟频率，N 为流水线级数，d 为计算延时。由式(6)可知流水线级数与频率和吞吐量成正比，因此为了提升算法的吞吐量，采用预计算和 CSA 策略来减少 A 和 E 的延时，并采用全流水线实现 SHA-256 算法。

3.1.1 预计算优化

SHA-256 运算主要集中在循环迭代部分，而其中较长的路径是更新 A 和 E 的值，且它们具有相同的加法操作，因此，可以采用预先计算的方法缩短路径的延时。由于 H_t,K_t,W_t 在 t 轮运算结束就可以获得，且不需要其他额外的逻辑运算，因此对这 3 个操作数进行预计算，并使用寄存器数组 S 存储结果值，用于 $t+1$ 轮的计算，其更新结构如图 2(b)所示。

经过预计算后，核心处理过程中更新的 A 和 E 值从式(3)、式(4)改变为式(8)、式(9)，使更新的 A,E 值分别从原来的 $6t_{ADD}$ 和 $5t_{ADD}$ 缩短至 $4t_{ADD}$ 和 $3t_{ADD}$：

$$S_t=H_t+K_t+W_t \tag{7}$$

$$A_{t+1}=\sum_1(E)+Ch(E_t,F_t,G_t)+S_t+\sum_0(A)+Maj(A_t,B_t,C_t) \tag{8}$$

$$E_{t+1}=\sum_1(E)+Ch(E_t,F_t,G_t)+S_t+D_t \tag{9}$$

3.1.2 加法器 CSA 优化

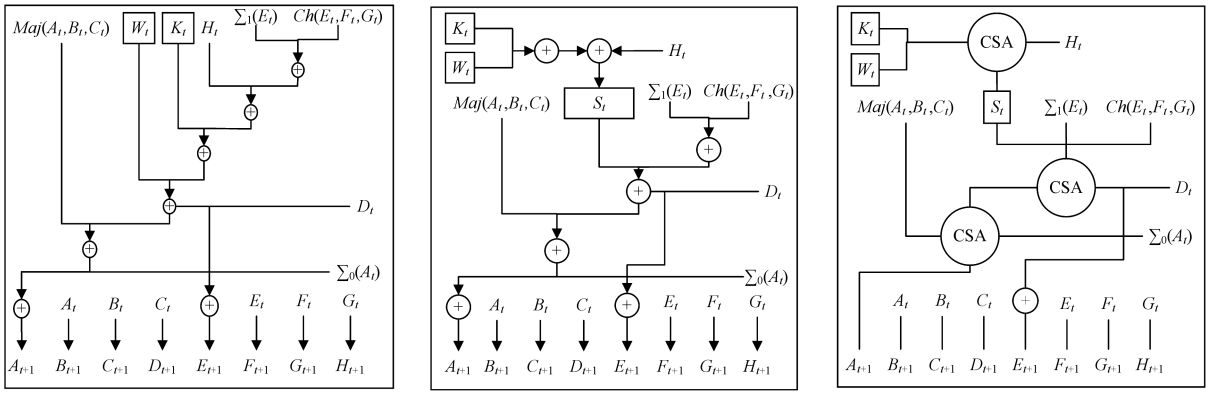
FPGA 适合于位运算，而加法运算的延时比位运算高，保留进位加法器(Carry Save Adders，CSA)策略能够减少加法运算，最小化关键路径的长度，保证流水吞吐率[18]。CSA 运算如下：

$$\begin{cases}S(a,b,c)=a\wedge b\wedge c\\ Ca(a,b,c)=[(a\&b)|(b\&c)|(a\&c)]\ll 1\\ CSA(a,b,c)=S(a,b,c)+Ca(a,b,c)=a+b+c\end{cases} \tag{10}$$

其中，a,b,c 为 n 位二进制数。

经过预计算优化后，算法中 S_t，A_{t+1} 和 E_{t+1} 分别需要

$2t_{ADD}$，$4t_{ADD}$和$3t_{ADD}$。通过CSA策略进一步减少加法延时，其结构改变为图2(c)所示。



(a)更新计算结构图 (b)预计算优化图 (c)CSA优化图

图2 SHA-256算法的结构图

Fig. 2 Structure diagram of SHA-256 algorithm

通过增加位运算来减少加法运算，使图2(b)中的预计算结构改变为图2(c)中的CSA结构，改变了算法的处理过程，使式(7)－式(9)的直接加法器处理替换为用式(11)－式(13)中CSA运算：

$$S_t=CSA(H_t,K_t,W_t) \tag{11}$$

$$A_{t+1}=CSA(CSA(\sum_1(E),Ch(E_t,F_t,G_t),S_t),\sum_0(A),Maj(A_t,B_t,C_t)) \tag{12}$$

$$E_{t+1}=CSA(\sum_1(E),Ch(E_t,F_t,G_t),S_t)+D_t \tag{13}$$

通过预计算和CSA策略，使A_{t+1}和E_{t+1}的运算都缩减至$2t_{ADD}$，从而提高算法的频率。

3.1.3 SHA-256算法的流水线优化

流水线方法是Intel首次使用于486芯片[19]，通过增大空间将多次重叠的操作进行展开的时空映射并行技术。SHA-256算法一共需要迭代64次，且在第一轮运算前需要消耗一个时钟获得预计算需要的值，最后输出也要一个时钟，因此，本文采用66级全流水线来实现流水线优化。

在具体实现时分为3个模块协同工作：W模块、预计算S模块和数据更新$(A-H)$模块。每一个模块都包含64个子模块，用于数据的存储传递。首先，将第一个数据输入到W的第一个子模块，输出$W[0]$传递到S模块，再将$S[0]$输入到数据更新模块，计算A_0，B_0，C_0，D_0，E_0，F_0，G_0，H_0，完成第一次迭代；然后，再计算W模块的第二个子模块，获得$W[1]$，再输入到S模块计算$S[1]$，紧接着做数据更新计算，完成第二次迭代；同时，第二个数据进行3个模块的第一个子模块操作，以此类推，直到第66个时钟完成第一个数据的输出，3个模块的数据流水线传递结构如图3所示。

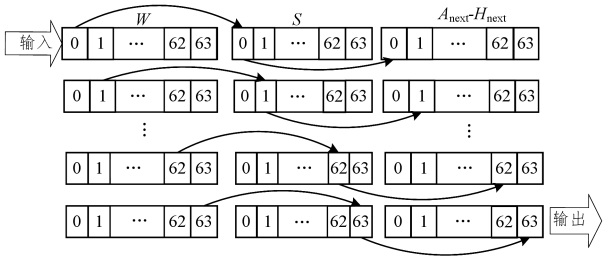


图3 数据流水线传递的结构图

Fig. 3 Structure diagram of data pipeline transfer

流水线实现方案既实现了数据内的串行传递，又达到了数据间的并行操作。当有大量的输入数据要进行SHA-256运算时，采用这种流水线的实现方式能够在第66个时钟之后的每个时钟都有结果输出，使算法的吞吐量和资源利用率得到很大的提升。

由于时空映射的流水线是一种空间换时间的技术，因此SHA-256算法的存储需求发生了较大的改变，如表2所列。

表2 SHA-256算法优化后的存储需求

Table 2 Storage requirements of optimized SHA-256 algorithm

存储类别	存储宽度/bit	存储深度
input	512	1
h_0-h_7	32	8
K_0-K_{63}	32	64
S	32	64
W	32	1024
$A-H$	32	8
$A_{next}-H_{next}$	32	512
output	256	1

3.2 基于循环队列的DS流水线优化

7-Zip解密算法需要进行524 288次DS操作，且拼接64 byte需要进行一次SHA-256运算。DS用到P和拼接次数num，SHA-256运算采用66级流水线，DS操作需要和SHA-256串联运算，也要采用66级流水线运算。相同密码长度的拼接次数num相同，且密码所在位置也相同，因此只需要解决密码的循环存储问题并保证拼接能在66个时钟周期内完成，然后进行链接组合，就可以实现数据拼接的流水线。

密码的循环存储可以采用类似循环队列的方式进行操作。将向量空间想象为一个首尾相接的圆环，这种向量为循环向量，存储在其中的队列称为循环队列，具有“先进先出”的特点。因此，采用改进的循环队列实现密码的循环存储，其实现如算法2所示。

算法2 P的循环队列存储

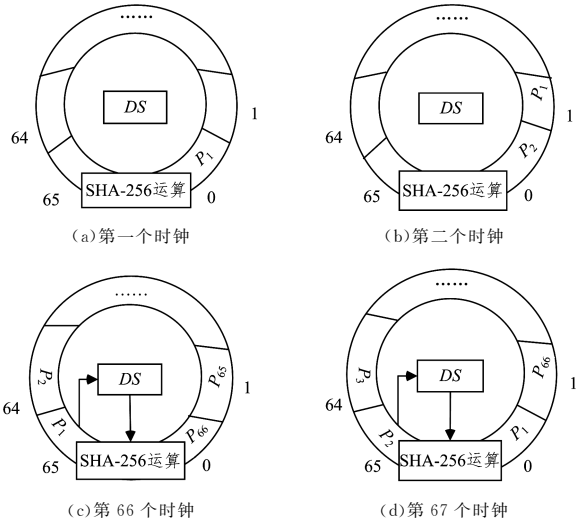
输入：P

输出：Result(P)

1. regP_array[66]；//存储队列

```
2. for(i=0;i<66;i++) begin //P 依次入队
3.    for(j=65;j>0;j--) begin
4.       P_array[j]= P_array[j-1];//push 操作
5.    end for
6.    P_array[0]= P_{i+1};// P1-P66入队
7. end for
8. while   begin//每个时钟都运算一遍
9.       Result(P)= P_array[65];//输出结果
10.      P_array[0]= P_array[65];//push 操作
11.      for(j=65;j>0;j--)
12.         P_array[j]= P_array[j-1];//push 操作
13.      end for
14. end while
```

算法 2 中，步骤 1 定义了一个寄存器数组，用于存储 P 的队列；步骤 2－7 是前 66 个时钟 P_1－P_{66} 依次进入队列，对应图 4 中(a)－(c)；步骤 8－14 是输出结果进行拼接并进行 SHA-256 运算，同时所有数据都向前推进一步，如图 4(d)所示。



(a)第一个时钟　(b)第二个时钟

(c)第 66 个时钟　(d)第 67 个时钟

图 4　循环结构图

Fig. 4　Loop structure diagram

从图 4 中可以看出，通过这种改进的循环队列，能够在第 66 个时钟后开始向 SHA-256 输入数据，并且在第 67 个时钟 P_1 重新进入 P_array[0]，开始新一轮的循环拼接，从而完成密码的循环存储，实现流水线的数据传递。

数据拼接需要将密码先转换为 2 byte 宽字节，且拼接总数也是 2 的整数倍。因此，可以每次拼接 2 byte，有效拼接操作只需要 32 个时钟周期，再加上缓存和数据传递，只需要 34 个时钟周期即可完成一次 64 byte 的拼接。由于不同密码拼接时其所在位置和拼接次数相同，因此可以采用两个 64 字节的数组进行拼接标记存储，间接链接到算法 2 的输出 P，可以供 66 组密码共同使用，实现了拼接的并行操作，并且能够在 66 级流水线运算下完成拼接操作，没有计算的延迟。以密码“abcd”为例，图 5 给出数组链接组合的部分结构(图中的值都是 16 进制)。

通过改进的循环队列将密码循环存储，并通过额外的数组进行数据标记拼接，从而完成整个拼接过程，实现 66 级流水线和 SHA-256 串联计算，提高了数据拼接的并行性和算法的计算性能。

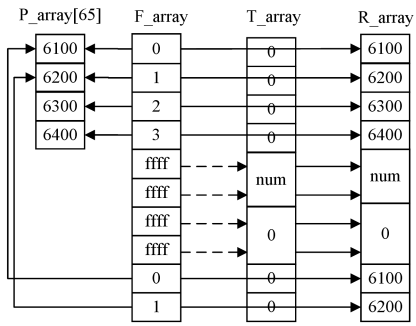


图 5　数据链接组合结构图

Fig. 5　Data link structure diagram

3.3　基于 IP 核的数据存储优化

双端口 RAM(Random Access Memory)是常见的共享式多端口存储器，其最大的特点是存储数据共享。一个存储器配备两套独立的地址、数据和控制线，允许两个独立的控制器同时异地访问存储单元，在非断电的情况下能够对同一地址进行多次读或者写。FPGA 片上 RAM 主要分为两类：分布式 RAM 和块 RAM。FPGA 器件上 RAM 资源的分布：分布式 RAM 分布在逻辑单元中，用逻辑单元内部的结构实现；块 RAM 是 FPGA 片上专有单元[20]。因此，算法实现采用块 RAM 存储需要校验的数据，减少了 FPGA 的逻辑资源消耗，且能够多次读取校验数据。

CPU 端一次写进 FPGA 中 4 字节数据，而数据校验为一次 16 byte，且 1 M 的加密压缩文档只需校验 240 byte，因此使用 IP Catalog 生成具有 4 byte 写端口和 16 byte 读端口的双端口 RAM，且设置深度为 2048，能够满足校验数据的存储。将校验数据通过双端口 RAM 存储在 FPGA 中，使数据校验运算不需要通过 CPU 再次进行传输，且能够多次使用，从而降低了算法的通信延时。双端口 RAM 的读写存储结构如图 6 所示。

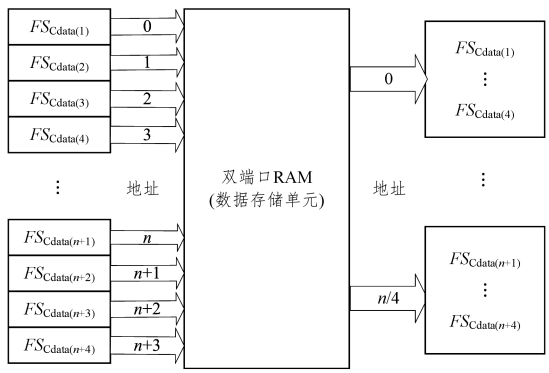


图 6　双端口 RAM 的读写结构图

Fig. 6　Read-write structure diagram of dual port RAM

FIFO(First In First Out)的全称是先进先出存储器，一般用于不同时钟域之间的数据传输，采用数据缓存实现数据传输速率的匹配，提高传输速度。FIFO 和双端口 RAM 都可以完成数据的缓存工作，但是 RAM 需要复杂的地址译码电路，用作数据缓存时不如 FIFO 方便[21]，并且 FIFO 仲裁控制简单。因此，FIFO 特别适合数据缓冲和突发传送数据。FPGA 功耗分为静态和动态两种。其中，静态功耗主要由电路决定；而动态功耗由多种因素决定，其公式如下[22]：

$$P_{dyn}=\alpha C_L V_{DD}^2 f \tag{14}$$

其中，α 表示电路信号转换速率，C_L 是电容负载，V_{DD} 表示电压，f 是时钟频率。

由式(14)可知，频率越高，功耗越高。从整个算法中可以发现一次迭代占用了解密算法的大部分运算，应该采用高频率时钟提高其计算性能，而 P 生成模块和二次校验计算需求较小，采用低频率时钟足以满足计算的需求。因此，P 模块和一次迭代的输出结果都采用 IP 核 FIFO 进行数据缓存，将 P 模块低频率运算的结果存储于 P-FIFO，一次迭代再从中获取 P 进行高频率并行计算，结果输出至 *digest*-FIFO 供二次校验的低频率串行计算。通过 FIFO 进行存储，分离了相同数据的不同时钟域，实现了不同速率的匹配，提升了效率且减少了功耗。

4 实验及结果分析

实验的硬件平台由四核可重构的 FPGA 加速卡，芯片型号为 Altera 公司 Cyclone V 系列的 5CEFA9，其逻辑资源为 113 560，寄存器资源有 454 240 个；软件平台由 Altera 公司专用软件 Quartus 进行设计、综合、编译。

由于哈希算法 SHA-256 的计算复杂性高，且迭代次数多，因此，本文采用流水线方法实现 SHA-256 算法，并采用预计算和 CSA 策略优化关键路径，实验结果如表 3 所列。

表 3 不同文献的 SHA-256 对比

Table 3 SHA-256 comparison of different references

对比文献	频率/MHz	吞吐量
文献[13]	74	592 Mbps
文献[14]	115.6	909.48 Mbps
文献[15]	41.97	1007.7 Mbps
文献[16]	172	11 008 Mbps
本文方法	215	110.080 Gbps

从表 3 中可以看出，本文方法的频率达到了 215 MHz，虽然与达到最高频率的文献[16]相比没有太大的提升，但吞吐量是其 10 倍，性能有明显的提升。本文采用了 3 种优化方法对 SHA-256 算法进行优化，为了显示优化后的结果，采用如下 3 种方案进行对比。

方案 1 串行实现，即不进行任何优化。

方案 2 只采用流水线进行优化。

方案 3 采用流水线、预计算进行优化。

3 种方案的实现平台与本文所用实现平台相同，实验结果如表 4 所列。

表 4 不同方案的 SHA-256 对比

Table 4 SHA-256 comparison of different schemas

对比方案	资源占用(ALMs)	吞吐量	吞吐量/资源占用
方案 1	1003	1696 Mbps	0.001 69
方案 2	8892	85.04 Gbps	0.009 56
方案 3	11 124	106.96 Gbps	0.009 61
本文	10 709.5	110.08 Gbps	0.010 27

表 4 增加了吞吐量与资源占用的比值，该比值能够反映吞吐量的增加是否大于资源的增长，即平均单位资源内吞吐量是否增长。通过比较可以看出，本文经过 3 种方法进行优化的效果最好。为了验证本文的设计方法是否适用于不同的 FPGA 结构，将本文的实现结果移植到 Xilinx 公司的 XCKU060 芯片上，性能对比结果如表 5 所列。

表 5 不同芯片的性能对比

Table 5 Performance comparison of different chips

芯片型号	资源占用	频率/MHz	吞吐量/Gbps
5CEFA9	10 709.5(ALMs)	215	110.080
XCKU060	21 556(LUT)	287	146.944

表 5 显示的结果表明了本文的方法可以在不同结构的 FPGA 芯片上实现，且吞吐量还能得到进一步提高。

通过分析算法的 PMC 特性，分别优化算法的通信、存储和处理过程：利用流水线技术实现数据拼接和 SHA-256 算法，从而提升算法的性能，满足计算需求，并利用双端口 RAM 存储校验数据，减少了通信需求，同时满足算法的存储需求，再利用 FIFO 隔离时钟域。经过综合布线后，算法各模块的资源使用情况如表 6 所列。

表 6 算法资源占用表

Table 6 Algorithm resource occupancy table

算法模块	ALMs	Registers	Block Memory
密码穷举	506.4	789	0
数据拼接	3177.2	3414	0
二次校验	3727.6	5146	0
匹配验证	45.7	170	0
SHA-256	11 289.6	37 390	28 896
digest-FIFO	28.5	18	32 769
P-FIFO	37.2	103	21 504
FS_{Cdata}(RAM)	0	0	65 536

从表 5 中可以看出，FIFO 和 RAM 主要消耗的是块存储器，片上资源使用较少；而其他模块主要消耗片上逻辑资源，且占比最高的是 SHA-256。由于 SHA-256 算法的输入不同，因此其布线结果会略有偏差。分析算法的资源占用情况，可实例化 4 个算子(完整的破译算法)，四核 16 个算子，且通过控制并行运行，最终实现算法的性能如图 7 所示。

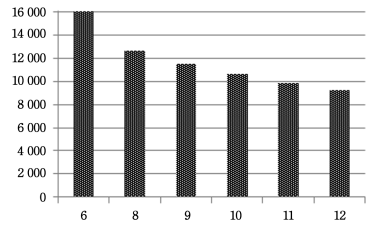


图 7 不同密码长度的破解性能

Fig. 7 Cracking performance of different password length

图 7 中显示了相同加密压缩文档的不同密码长度的破解速率，由于密码长度越长，需要进行的 SHA-256 算法的次数就越多，因此随着密码长度的增加，其破译性能越来越低。由于 7-Zip 解密算法的独特性，需要进行压缩数据的部分验证，因此对相同密码长度的不同加密压缩文档进行测试，性能结果如图 8 所示。

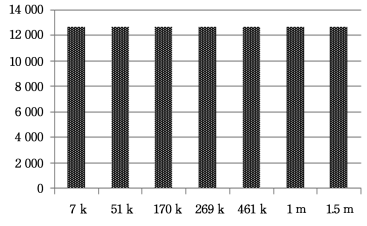


图 8 相同密码长度时不同文件大小的破译性能

Fig. 8 Cracking performance of different file sizes

从图 8 中可以看出，随着压缩文档的增大，其计算性能并没有明显的增大或减小，这是因为校验数据存储于 FPGA 内部的 RAM 单元中，无再次通信的限制，并且耗时主要集中在数据拼接和 SHA-256 运算，因此，通过存储优化，使算法的性能不受数据校验的影响。

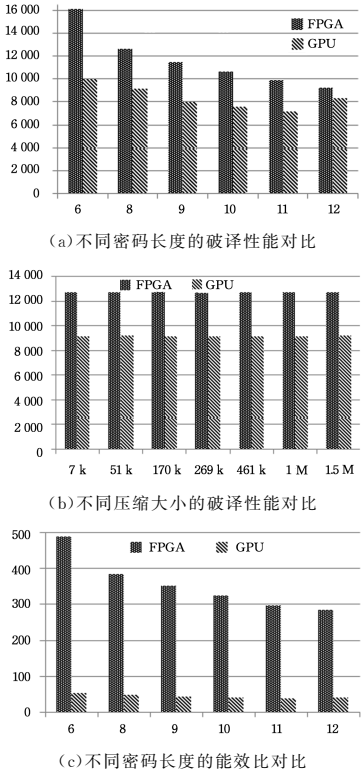
目前，破解 7-Zip 加密文档主要是在 CPU 平台和 GPU 平台。Hashcat 是一款开源的破解软件，同时支持 CPU 平台和 GPU 平台，因此，本文在 CPU(I5-7500)，GPU(GTX1080)上与 Hashcat 进行比验证，结果如表 7 所列。

表 7　不同平台的对比

Table 7　Comparison of different platforms

平台	速度(个/s)	功耗/W	能效比
CPU	56	7.61	7.36
GPU	9108	193	47.19
FPGA	12640	32.98	383.26

在不同平台的对比实验中，8 bit 密码长度 FPGA 的破解性能达到了 12 640 个/s，且功耗只有 32.98 W，为 GPU 的 0.18 倍。由表 7 可以看出，FPGA 的能效比是 GPU 的 8 倍，而 CPU 的性能较低，远远不能满足实际的需求，能效比是未来高性能计算机需要解决的重要问题[23]。为了进一步与 GPU 进行对比，分别在不同密码长度的破译性能、不同文件大小的破译性能、不同密码长度破解的能效等方面进行对比，比较结果如图 9 所示。



(a)不同密码长度的破译性能对比

(b)不同压缩大小的破译性能对比

(c)不同密码长度的能效比对比

图 9　FPGA 与 GPU 的对比结果

Fig. 9　Comparison results between FPGA and GPU

从比较结果可以看出，无论是不同密码长度还是不同文件大小，FPGA 的性能都要优于 GPU，并且最终的能效比也是 GPU 的 7～8 倍。

集群服务器是实际应用于密码破译的重要设施之一，而 GPU 服务器又是常用的一种。因此，本文将分别含有 8 块加速卡的 FPGA 服务器和 GPU 服务器进行对比验证，实验结果如图 10 所示。从图 10 可以看出，本文方案的能效比是 GPU 的 8 倍。针对目前集群服务器能耗过高的问题[24]，本文的优化方案与 GPU 相比具有较大的优势，也表明了该方案的可行性。

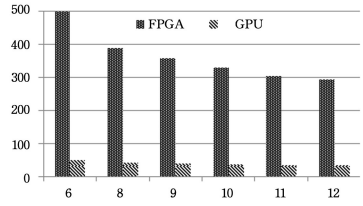


图 10　不同服务器的对比结果

Fig. 10　Comparison results between different servers

结束语　目前，破解 7-Zip 加密文档主要有通用服务器和 GPU 服务器。通用服务器基于操作系统实现，其计算速度受限于 CPU 构架，计算资源受限于系统调度，且指令之间属于串行执行，无法实现高速并行计算；而 GPU 平台涉及到访存问题，限制了破译的性能且功耗较高。因此，本文提出在低功耗的可重构计算平台 FPGA 上进行实现。在分析解密算法的 PMC 特性之后，采用流水线技术实现数据拼接和 SHA-256 算法，并优化 SHA-256 算法的关键路径，从而满足其高密度的核心计算部分。同时，使用双端口 RAM 存储校验数据，并使用 FIFO 缓存数据，隔离不同时钟域的数据，从而满足算法的存储需求。最终通过实验对比表明，本文方案在性能上有明显提升，且功耗较低，具有较高的实用价值。

参 考 文 献

[1] CHEN F T, YUAN J L. Enhanced Key Derivation Function of HMAC-SHA-256 Algorithm in LTE Network[C]// Fourth International Conference on Multimedia Information NETWORKING and Security. IEEE Computer Society, Washingdon, DC, USA, 2012: 15-18.

[2] ZHAO X J, GUO S Z, WANG T, et al. Improved Cache trace driven attack on AES and CLEFIA[J]. Journal on Communications, 2011, 32(8): 101-110.

[3] WANG D, JIAN G P, HUANG X Y, et al. Zipf's Law in Passwords[J]. IEEE Transactions on Information Forensics and Security, 2017, 12(11): 2776-2791.

[4] MA J, YANG W N, LUO M, et al. A Study of Probabilistic Password Models[C]// IEEE Symposium on Security and Privacy. USA: IEEE, 2014: 689-704.

[5] WANG D, ZHANG Z J, WANG P, et al. Targeted Online Password Guessing: An Underestimated Threat[C]// Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. New York: USA, ACM, 2016: 1242-1254.

[6] WANG P, WANG D, HUANG X Y. Advances in password security[J]. Journal of Computer Research and Development, 2016, 53(10): 2173-2188.

[7] KOZIEL B, AZARDERAKHSH R, KERMANI M M, et al. Post-Quantum Cryptography on FPGA Based on Isogenies on Elliptic Curves[J]. IEEE Transactions on Circuits and Systems

I: Regular Papers, 2017, 64(1): 86-99.
[8] ZHANG C, LI P, SUN G, et al. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks[C]// Proceesing of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. New York: ACM, 2015: 161-170.
[9] DABHADE S D, RATHNA G N, CHAUDHURY K N. A Reconfigurable and Scalable FPGA Architecture for Bilateral Filtering[J]. IEEE Transactions on Industrial Electronics, 2018, 65: 1459-1469.
[10] LIU P, LI S, DING Q. An Energy-Efficient Accelerator Based on Hybrid CPU-FPGA Devices for Password Recovery[J]. IEEE Transactions on Computers, 2019, 68(2): 170-181.
[11] ZHOU B, ZHANG Y Q, AN X J, et al. Optimization of RAR password brute-force cracking based on OpenCL [C]// High-Performance Computing China 2014. 2014: 871-874.
[12] AN X J, JIA H P, ZHANG Y Q. Optimized Password Recovery for Encrypted RAR on GPUs[C]// IEEE International Conference on High PERFORMANCE Computing and Communications. IEEE Computer Society, 2015: 591-598.
[13] LIU Z L, DONG X, LI D F. On the Hardware Implementations of the SHA-2(256, 384, 512) Hash Function[J]. Microelectronics & Computer, 2012, 29(12): 51-54.
[14] ALGREDO-BADILLO I, FEREGRINO-URIBE C, CUMPLIDO R, et al. FPGA-based implementation alternatives for the inner loop of the Secure Hash Algorithm SHA-256[J]. Microprocessors & Microsystems, 2013, 37(6/7): 750-757.
[15] JULIATO M, GEBOTYS C. A Quantitative Analysis of a Novel SEU-Resistant SHA-2 and HMAC Architecture for Space Missions Security[J]. IEEE Transactions on Aerospace & Electronic Systems, 2013, 49(3): 1536-1554.
[16] MICHAIL H E, ATHANASIOU G S, KELEFOURAS V, et al. On the exploitation of a high-throughput SHA-256 FPGA design for HMAC[J]. Acm Transactions on Reconfigurable Technology & Systems, 2012, 5(1): 1-28.
[17] TAN J, ZHOU Q L, SI X M, et al. Implementation and improvement of full-pipeline MD5 algorithm based on mimic compiter [J]. Journal of Chinese Computer Systems, 2017, 38(6): 1216-1220.
[18] LEI Y W, DOU Y, GUO S. High precision Scientific Computation Accumulator on FPGA[J]. Chinese Journal of Computers, 2012, 35(1): 112-122.
[19] WU Q, WANG X W, HUANG M. OpenFlow Switch Packets Pipeline Processing Mechanism Based on SDN[J]. Computer Science, 2018, 45(10): 295-299.
[20] LI Y, ZHANG D X, YU F. Technology Mapping of FPGA On-Chip-RAM in RTL Synthesis[J]. Acta Electronica Sinica, 2016, 44(11): 2660-2667.
[21] YU X F, LIU X B, HU B L, et al. Design of FIFO in High Speed Data Storage System Based on FPGA[J]. Nuclear Electronics & Detection Technology, 2010, 30(1): 59-62.
[22] LI B, ZHOU Q L, SI X M. Mimic computing for password recovery[J]. Future Generation Computer Systems, 2018, 84: 58-77.
[23] ZHANG K, GUO F, ZHENG W et al. Design of a Pipeline-Coupled Instruction Loop Cache for Many-Core Processors[J]. Journal of Computer Research and Development, 2017, 54(4): 813-820.
[24] LIN B, LI S S, LIAO X K, et al. Seadown: SLA-Aware Size-Scaling Power Management in Heterogeneous MapReduce Cluster [J]. Chinese Journal of Camputers, 2013, 36(5): 977-987.

CHEN Xiao-jie, born in 1993, postgraduate, is not member of China Computer Federation (CCF). His main research interests include information security.

ZHOU Qing-lei, born in 1962, Ph. D, professor, Ph. D supervisor, is member of China Computer Federation (CCF). His main research interests include information security, automata theory and computational complexity theory.