计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200500040

基于 FPGA 集群的 Office 口令恢复优化实现

李 斌[1] 周清雷[1] 斯雪明[2] 陈晓杰[2]
1 郑州大学信息工程学院 郑州 450001
2 数学工程与先进计算国家重点实验室 郑州 450001

摘 要 口令恢复是口令找回和电子取证的关键技术,而加密的 Office 文档被广泛使用,实现 Office 加密文档的有效恢复对信息安全具有重要的意义。口令恢复是计算密集型任务,需要硬件加速来实现恢复过程,传统的 CPU 和 GPU 受限于处理器结构,大大限制了口令验证速度的进一步提升。基于此,文中提出了基于 FPGA 集群的口令恢复系统。通过详细分析 Office 加密机制,给出了各版本 Office 的口令恢复流程。其次,在 FPGA 上以流水线结构优化了核心 Hash 算法,以 LUT(Look Up Table)合并运算优化改进了 AES(Advanced Encryption Standard)算法,以高速并行实现了口令生成算法。同时,以多算子并行设计了 FPGA 整体架构,实现了 Office 口令的快速恢复。最后,采用 FPGA 加速卡搭建集群,配合动态口令切分策略,充分发掘了 FPGA 低功耗高性能的计算特性。实验结果表明,无论在计算速度还是能效比上,优化后的 FPGA 加速卡都是 GPU 的 2 倍以上,具有明显的优势,非常适合大规模部署于云端,以缩短恢复时间找回口令。

关键词:FPGA;Office 加密文档;口令恢复;SHA1(Secure Hash Algorithm 1);AES;信息安全
中图法分类号 TP309

Optimized Implementation of Office Password Recovery Based on FPGA Cluster

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

Abstract Password recovery is the key technology of password back and electronic forensics. While encrypted office documents are widely used,it is of great significance to achieve the effective recovery of office encrypted documents for information security. Password recovery is a computation-intensive task and requires hardware acceleration to implement the recovery process. Traditional CPUs and GPUs are limited by the processor structure,which greatly limits the further increase in password verification speed. In view of this,this paper proposes a password recovery system based on FPGA cluster. Through detailed analysis of the office encryption mechanism,the password recovery process of each version of office is given. Secondly,the core Hash algorithm is optimized with a pipeline structure on FPGA,the AES algorithm is improved by LUT merging operation,and the password generation algorithm is implemented in parallel at high speed. At the same time,the architecture of FPGA is designed with multiple algorithm sub-modules in parallel,which realizes the fast recovery of office password. Finally,the FPGA accelerator card is used to build the cluster,and the dynamic password segmentation strategy is used to fully explore the low-power and high-performance computing features of FPGAs. The experimental results show that the optimized FPGA accelerator card is more than twice the GPU in terms of computing speed and energy efficiency ratio,which has obvious advantages and is very suitable for large-scale deployment in the cloud to shorten the recovery time and retrieve the password.

Keywords FPGA,Office encrypted document,Password recovery,SHA1,AES,Information security

1 引言

随着信息技术的快速发展,大量的电子文档在网络上传输,不乏含有个人隐私和敏感信息的重要加密文件。Office 办公软件是日常生活和办公使用得最多的工具,包含 Word,Excel,PowerPoint 等若干组件。为了有效地保证 Office 的安全性,微软提供了口令机制限制访问权限,为用户的数据提供安全保障[1]。但是任何事物都具有两面性,口令保护尽管提供了一种很好的安全保护方案,但是用户难免会忘记口令。一旦口令丢失,尽管文档属于自己,但它已经限制了自己访问

到稿日期:2020-05-11 返修日期:2020-08-22 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(2016YFB0800100,2016YFB0800101);国家自然科学基金面上项目(61572444)
This work was supported by the National Key Research and Development Program of China (2016YFB0800100,2016YFB0800101) and General Program of National Natural Science Foundation of China (61572444).
通信作者:李斌(iebinli@zzu.edu.cn)

的权利，给用户带来了极大的困扰，甚至是经济损失。同时，加密文档也给国家安全部门的侦查取证工作带来了诸多不便[2]。因此，Office 加密文档的快速恢复成为了研究的热点。

口令恢复是口令分析的一个重要方向，已经成为密码学研究的重点，在计算机数据和电子取证等领域有着重要的应用。口令恢复技术从最初单纯使用暴力方式，枚举口令空间的所有组合，发展到利用人类记忆的局限性和社会统计学，结合暴力和字典恢复优势的时空折中恢复方式[3-4]。即在已知字典的基础上进行多种规则变动，以此得到更为真实的口令，提高了破解成功率。但是随着口令长度的增加和口令字符集的扩大，口令空间以指数的速度增大，从而使得在有效的时间内实现口令空间的完全搜索变得越来越困难，提高口令验证速度也成为了主要的制约因素。

随着对口令恢复研究的深入，口令概率模型[5]逐步完善，包括 Markov，PCFG(Probabilistic Context-Free Grammars)等概率攻击理论模型。它们通过对口令结构的划分与频次分布的统计，来增加高概率候选口令的组合生成空间，具有更好地扩展性和灵活性。在此基础上，利用个人信息构造口令的定向攻击[6]应运而生，即通过姓名、生日、电话等攻击对象的相关信息，来增强口令猜测的针对性。此外，还可根据不同地域的口令特点[7]，深度挖掘口令元素组成，以此优化口令攻击模型。近期的研究更是表明口令服从 Zipf 分布[8]，该分布有效地刻画了口令使用频率和排名之间的关系，揭示了口令的重用行为和脆弱性，为缩减口令空间、提高命中率提供了严谨的科学支撑。

高性能计算平台的发展为口令恢复提供了更多便利，但对依赖时间瓶颈来保证安全性的口令机制产生了威胁。随着可编程器件的发展，FPGA 已经广泛应用于并行计算的各个领域[9]。相比 CPU 和 GPU，FPGA 摆脱了冯诺依曼架构[10]，它将某种计算架构用硬件电路实现，然后持续地将数据流输入系统并完成计算，属于“结构计算”或“空间计算”。FPGA 可以根据需要通过软件编程来定义器件的硬件功能，具有功耗低、延迟低、吞吐率高和灵活性好等优势。

由此，本文利用 FPGA 并行计算的优势，在一个板卡上集成多个 FPGA 芯片，形成加速卡，并以一定规模形成 FPGA 服务器集群，为 Office 搭建高性能口令恢复平台。其次，通过对 Office 加密机制的分析，深度优化 SHA1 和 AES 核心算法，采用流水线技术和多算子并行技术提高口令的并发验证，极大地提高了计算速度。同时，结合口令概率攻击，在 FPGA 内部以并行的方式实现口令的高速生成，满足口令的供给需求。最后，在 FPGA 上实现了 Office 各种版本的高效快速的口令恢复算法，提高了整体恢复效率。

2　研究背景

2.1　Office 加密机制

Office 作为使用得最为广泛的办公软件，主要通过自身提供的加密功能来实现对文档内容的加密，并分为打开文件时加密和修改文件时加密，为文档内容不被泄露和篡改提供了双重保护。目前，微软公开了 MS-CFB 和 MS-OFFCRYPTO 两个材料[11-12]，为研究 Office 文档结构以及口令验证机制提供了参考，其中 MS-CFB 主要涉及 Office 文档结构，包括存储格式、组织方式等；MS-OFFCRYPTO 主要涉及 Office 文件加密机制以及口令验证过程的加密参数等。本文结合 Office 文件格式的分析，对 Office 各版本的加密机制进行了深入研究，阐明了加密参数的含义以及加解密过程。

根据微软官方的定义，加密 Office 文档主要需要 Salt，Hash，KeyGeneration，Encryption，Decryption，EncryptedVerifier 和 EncryptedVerifierHash 等 7 种信息。这 7 种信息以明文数据流的形式保存在 Office 文档中，其各项含义如表 1 所列。

表 1　加密 Office 文档的数据流信息

Table 1　Encrypted Office document data flow information

序号	标识	内容说明
1	Salt	16 字节随机数，与用户口令一起生成加解密密钥
2	Hash	Hash 函数，用于迭代计算，不同 Office 版本使用不同的函数
3	KeyGeneration	密钥生成函数，产生的 Key 被当作加解密的密钥
4	Encryption	加密函数，使用 Key 进行文件加密，不同 Office 版本的加密函数不同
5	Decryption	解密函数，使用 Key 进行文件解密
6	EncryptedVerifier	加密后的 Verifier 值，验证口令时读此数据
7	Encrypted-VerifierHash	EncryptedVerifier 的 Hash 值，16/32 字节，验证口令时读该数据

Office 各个版本的加密流程基本相同，具体描述为：

1)生成随机数 Salt，并将其保存在文件的加密信息流中。

2)用户输入加密的口令 Pwd，与 Salt 一起生成加密密钥 Key＝KeyGeneration(Salt，Pwd)。

3)生成验证随机数 Verifier，并进行加密运算 EncryptedVerifier＝Encryption(Key，Verifier)，将结果保存在文档的加密信息流中。

4)对 Verifier 进行 Hash 运算 VerifierHash＝Hash(Verifier)，再使用 Key 进行加密运算 EncryptedVerifierHash＝Encryption(Key，VerifierHash)，将结果保存在文档的加密信息流中。

5)使用 Key 对文件内容加密，得到加密后的 Office 文档。

Office 解密过程为加密的逆过程，其前两步基本相同，在步骤 3)使用 Key 对 EncryptedVerifier 进行解密，得到 Verifier＝Decryption(Key，EncryptedVerifier)，再对 Verifier 进行 Hash 运算得到 VerifierHash＝Hash(Verifier)，并使用 Key 对 EncryptedVerifierHash 进行解密得到 VerifierHash′＝Decryption(Key，EncryptedVerifierHash)，如果 VerifierHash 和 VerifierHash′相等，则使用 Key 解密文档，否则口令错误。

2.2　FPGA 加速卡

传统的口令恢复计算部件主要有 CPU 和 GPU。CPU 是一种通用的架构，需要指令集和读写数据，并周而复始地操作，称为“时域计算”。当并发操作量太大时，CPU 不能进行高效的处理，即便使用更多的线程也无济于事。对于 GPU，在芯片内部有一堆相同的计算核心，可以处理类似但并不完全相同的数据集，擅长“单指令多数据流(SIMD)”的并行处理。但是，由于 GPU 单核上的 Local Memory 较少，不适合加密算法的数据通路，对于 AES，DES 和 RC4 的加速，GPU 都不擅长。同时 GPU 功耗高、能效比低，对于使用 CPU＋GPU

搭建的口令恢复计算平台，具有“功耗墙”的问题[13]。

FPGA 具有面向单个或多个领域的重构加速能力，在领域范围内具有一定的应用灵活性和远高于 CPU 和 GPU 的性能。FPGA 的本质是无指令、无需共享内存的体系结构，可通过编程配置其中的逻辑单元和逻辑阵列互连结构，进而以接近“专用电路”的方式进行计算。FPGA 以查找表和寄存器实现复杂的组合逻辑操作，且片上内存 BRAM 属于各自的逻辑区域，无需不必要的仲裁和缓存。因此 FPGA 的运算速度足够快，具有很强的计算能力和灵活性。

FPGA 加速卡主要满足计算密集型应用，由 4 片大规模可重构 FPGA 构成高阶可重构计算核心，通过底板总线互连以及时钟单元进行时钟同步，并由 PEX8311 芯片通过控制单元 CPLD 进行 4 片 FPGA 的选择、烧录和数据读写。FPGA 加速卡可根据应用需求，重构 FPGA 处理核、I/O 接口、片上互连网络等，达到高效能计算的目的。其结构如图 1 所示。

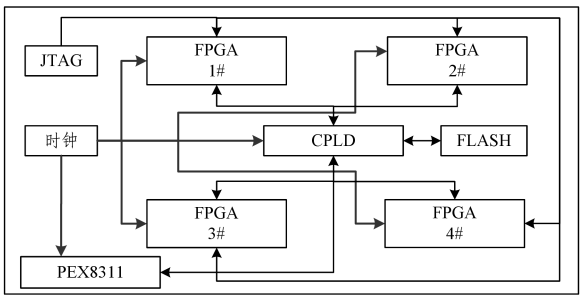

图 1 FPGA 加速卡结构图

Fig. 1 Structure diagram of FPGA acceleration card

由此，针对 Office 的口令恢复，包括 Hash 迭代和 AES 算法，主要以逻辑运算、位运算和加法为主，且算法结构规则紧凑，非常适合使用 FPGA 实现。其次，通过对 FPGA 逻辑区域的划分，可制定合适的硬件结构和数据通路，并以多模块并行的架构提高计算的灵活性和可扩展性。然后，将 FPGA 加速卡与云计算系统进行异构融合，形成 FPGA 集群，并通过调度管理和口令切分策略实现负载均衡。最终以 FPGA 集群的高效能计算，来满足使 Office 口令恢复得到实际应用的服务需求。

3 方案实现

3.1 Office 破解流程

通过对 Office 加密文档的分析可知，只要输入用户口令 Pwd 和 Salt 计算出正确的 Key，即可解密。因此密钥的生成是最为关键的一步。其次，本文方案对密钥生成进行了分析，将其分为口令扩展和 Hash 迭代，以优化控制流程。最后，由于 AES 解密更为耗时，本文方案将 EncryptedVerifier 依次进行解密、Hash 运算、再-加密，并与 EncryptedVerifierHash 进行比对，以缩短计算时间。因此整个 Office 口令破解可以细化为 5 个部分：文档解析、口令扩展、Hash 迭代、AES/RC4 加解密、对比验证，如图 2 所示。其中，主要的计算部分是高次 Hash 迭代，占整个过程计算量的 99%以上。

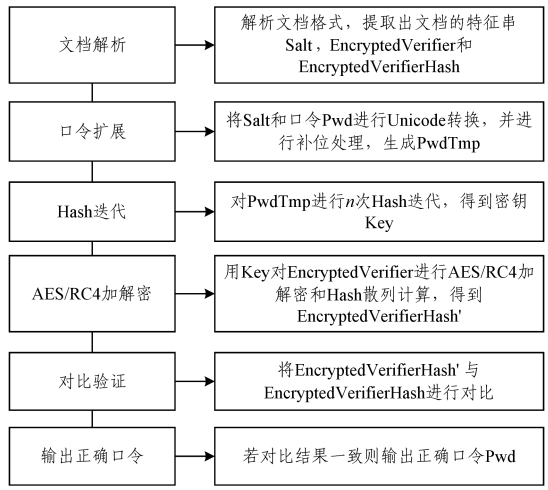

图 2 Office 加密文档的破解流程图

Fig. 2 Office encrypted document cracking flowchart

表 2 列出了 Office 2003，Office 2007，Office 2010，Office 2013－Office 2019 各版本主要步骤的详细说明。

表 2 Office 各版本的主要步骤分析

Table 2 Analysis of main steps for each version of Office

算法类型	主要步骤分析				
	文档解析	口令扩展	Hash 迭代	AES/RC4 加解密	对比验证
Office 2003	提取 48 字节特征串	Unicode 转码，MD5	7 次 MD5	MD5，RC4 加解密	验证结果，输出正确口令
Office 2007	提取 64 字节特征串	Unicode 转码，SHA1	5000 次 SHA1	SHA1，AES128 加解密	验证结果，输出正确口令
Office 2010	提取 64 字节特征串	Unicode 转码，SHA1	100000 次 SHA1	SHA1，AES128 加解密	验证结果，输出正确口令
Office 2013－Office 2019	提取 64 字节特征串	Unicode 转码，SHA512	100000 次 SHA512	SHA512，AES256 加解密	验证结果，输出正确口令

在 Office 2003 版本中，以 doc 格式存储文件，默认采用 RC4 加密算法，密钥长度为 40 bit。那么无论用户设置的口令长度如何，最终都将转换为 40 bit 长度的密钥。因此只需要预先构造密钥表，穷举 2^{40} 种 RC4 密钥，即可百分之百得到原始口令。在 Office 2007—Office 2019 版本中，以 docx 格式存储文件，采用 AES 加密算法，密钥长度强制为 128 位或 256 位，需要尝试 2^{128} 或 2^{256} 个口令空间。如果想要破解出正确口令，则有效的口令攻击手段是必不可少的。

另外，Office 口令恢复核心计算为 Hash 迭代。以 Office 2010 为例，需要 10 万次 SHA1 迭代运算，每次 SHA1 迭代需要 80 轮，每轮包括复杂的加法和逻辑运算，计算量非常大。在验证海量口令时，传统 CPU（Intel i5-7500）的速度仅为 2000 个每秒，远远无法满足实际应用的需求。其次，AES128 以 SHA1 迭代结果和 EncryptedVerifier 为输入进行加解密操作，如果采用并行架构，需要及时处理多组数据，以提高计算速度。最后，设最长口令长度为 20 字节，每次输入多组口令，模块间的通信带宽和存储也需要优化设计，以提高口令吞吐量。

综上，本文以 Office 2010 为例，在 FPGA 上对其进行优化改进，以搭建高性能 FPGA 集群，并拓展到 Office 2007 和 Office 2013－Office 2019。而对于 Office 2003，目前已可秒破，不再对其进行赘述。

3.2　核心算法的实现

3.2.1　SHA1 的改进与优化

SHA1 是 Office 2007/Office 2010 的核心运算段，占整个运算量的 99%以上，这里以全流水架构实现对 SHA1 算法的硬件加速。流水线结构把一个重复的过程分解为若干子过程，每个子过程可以和其他子过程同时进行，从而提高了系统的执行效率，具有较好的并行性[14]。图 3 给出了具有 80 级流水线结构的 SHA1 算法工作示意图，每个时钟可输入 1 个数据，如果连续输入 80 组，则 80 个时钟周期后会依次产生 80 组输出。显然，当 SHA1 流水线满负荷工作时，其效率非常高。

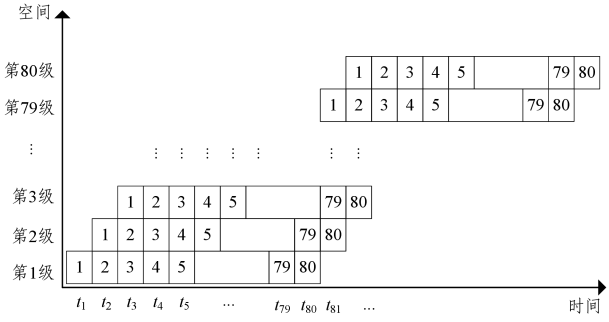

图 3　流水线结构示意图

Fig. 3　Schematic diagram of pipeline structure

SHA1 将初始信息填充至 512 比特位[15]，并初始化为 16 个 32 比特位分组 $w[15:0]$，以 H0 = 0x67452301，H1 = 0xEFCDAB89，H2 = 0x98BADCFE，H3 = 0x10325476，H4 = 0xC3D2E1F0 为初始链接变量，对 5 个中间变量 a,b,c,d,e 进行 80 轮迭代运算。每轮迭代公式如下：

$$a=a\_next;b=a;c=c\_next;d=c;e=d \tag{1}$$

$$a\_next=\{a[26:0],a[31:27]\}+f_t(b,c,d)+e+k_t+w_t \tag{2}$$

$$c\_next=\{b[1:0],b[31:2]\} \tag{3}$$

其中，$f_t(b,c,d)$为每轮迭代的非线性函数，k_t为每轮的常量，w_t为每轮的数据块。f_t，k_t和 w_t对应的表达式如下：

$$f_t(b,c,d)=\begin{cases}(b\wedge c)\vee(\sim b\wedge d), & 0\leqslant t\leqslant 19\\ b\oplus c\oplus d, & 20\leqslant t\leqslant 39\\ (b\wedge c)\vee(b\wedge d)\vee(c\wedge d), & 40\leqslant t\leqslant 59\\ b\oplus c\oplus d, & 60\leqslant t\leqslant 79\end{cases} \tag{4}$$

$$k_t=\begin{cases}0x5A827999, & 0\leqslant t\leqslant 19\\ 0x6ED9EBA1, & 20\leqslant t\leqslant 39\\ 0x8F1BBCDC, & 40\leqslant t\leqslant 59\\ 0xCA62C1D6, & 60\leqslant t\leqslant 79\end{cases} \tag{5}$$

$w_t=S^1(w_{t-3}\oplus w_{t-8}\oplus w_{t-14}\oplus w_{t-16})$，$t\geqslant 16$，$S^1$ 表示循环左移 1 位。

最后，以 $a=a+$H0，$b=b+$H1，$c=c+$H2，$d=d+$H3，$e=e+$H4 的级联输出 160 位散列值。

显然，b,c,d,e 可以直接通过传值得到，而 a 需要经过复杂运算得到，则延迟消耗都集中在关键路径 a 上。对于 FPGA，加法比位运算延迟大得多，为了减少加法器的使用，定义进位保留加法器 CSA，具体表示如下：

$$CSA(x,y,z)=(x\wedge y\wedge z)+(((x\&y)|(x\&z)|(y\&z))\ll 1) \tag{6}$$

同时，引入变量 g 进行预计算。正常计算时有：

$$g=CSA(e,k_t,w_t) \tag{7}$$

而当提前一轮计算时，有：

$$g=\mathrm{CSA}(d,k_{t+1},w_{t+1}) \tag{8}$$

即提前使用下一轮 e 的结果 d,k_{t+1},w_{t+1}，计算出 g。则 a 的计算可简化为：

$$a\_next=\mathrm{CSA}(\{a[26:0],a[31:27]\},f_t(b,c,d),g) \tag{9}$$

通过加入 CSA 和 g，最终使得 5 个模 32 加法运算简化为 2 个模 32 加法运算，对于 a 值的运算过程来说，不仅缩短了关键路径，降低了延迟，还减小了硬件面积。优化后的 SHA1 单次迭代流程如图 4 所示。

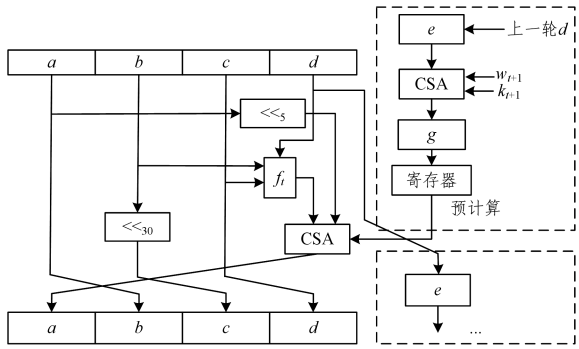

图 4　SHA1 单次迭代的结构图

Fig. 4　Structure diagram SHA1 single iteration

进一步，为了优化流水线结构，提高计算频率，将每轮运算和 w_t的运算独立开来。首先，建立 80 个子模块 Rnd0-Rnd79，每个模块会单独计算本次迭代下的 a,b,c,d,e 值，并级联输出。然后，在流水线主结构中，定义 32 比特位宽的二维寄存器数组 $w_t[0:79][0:15]$，为每一轮迭代计算提供参数。对于 w_t的更新，定义 32 比特位宽的寄存器数组 $w\_next[63:0]$，计算公式如下：

$$\{w\_next[i][0],w\_next[i][31:1]\}=w_t[i-1][1]\wedge w_t[i-1][3]\wedge w_t[i-1][9]\wedge w_t[i-1][14] \tag{10}$$

$$w_t[i][j]=w_t[i-1][j+1] \tag{11}$$

$$w_t[i][15]=w\_next \tag{12}$$

其中，$0\leqslant i<64$，$0\leqslant j<15$。而余下的 $w_t[64:79][0:15]$，其参与运算的参数已在第 48—63 轮计算出，直接用 $w_t[i][j]=w_t[i-1][j+1]$ $(64\leqslant i<79)$传值即可。另外，定义 32 比特位宽的寄存器数组 GI[0:80]，用来进行预计算，使用前一轮的 e 值、本轮的 w_t和 k_t进行 CSA 计算，并将结果传递给对应的轮运算。最后，在流水线主结构中定义 128 比特位宽的寄存器数组 RI[0:79]和 128 比特位宽的线型数组 WO[0:79]，$WO[i]$ $(0\leqslant i\leqslant 79)$作为 SHA1 本次计算的结果将值传递给 $RI[i]$，$RI[i]$作为下一次计算的输入。如此，通过 $RI[i]$和

$WO[i]$，将 SHA1 流水线的第 i 级与第 $i+1$ 级有机地连接起来，完成整个计算。

SHA1 流水计算的过程如图 5 所示。

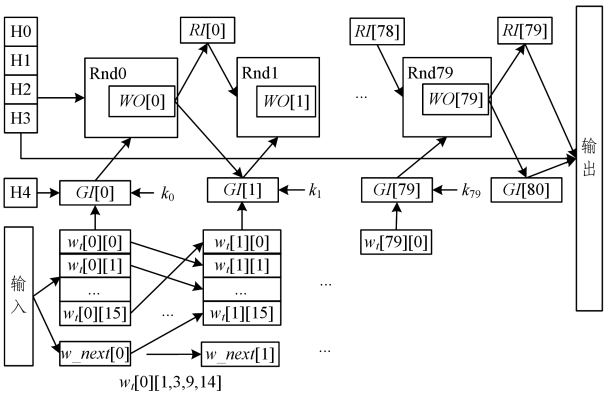

图 5 SHA1 流水线结构图

Fig. 5 Structure diagram SHA1 pipeline

3.2.2 AES 的改进与优化

AES 具有 128 比特的分组长度、3 种可选的密钥长度(128 字节、192 字节和 256 字节[16])，其轮数 N_r 依赖于密钥长度，分别为 10，12 和 14。AES 算法主要由密钥扩展、AddRoundKey 轮密钥加、SubBytes 字节代换、ShiftRows 行移位和 MixColumn 列混淆组成[17]，其中 SubBytes 变换实现混淆原则，ShiftRows 和 MixColumns 混合运算主要实现扩散原则。

对于密钥扩展，将输入的 128 位 Key 划分为 4 个 32 位寄存器数组 $w[0]$，$w[1]$，$w[2]$和 $w[3]$，经循环左移、SubBytes、异或运算后，产生本轮的输出，并将其作为下一轮的输入，计算公式如下：

$$\begin{aligned}&w[3]=w[3]\wedge SubByte(R^8(w[0]))\wedge rc[i];\\&w[2]=w[2]\wedge w[3];\\&w[1]=w[1]\wedge w[2];\\&w[0]=w[0]\wedge w[1];\end{aligned}\tag{13}$$

其中，R^8 表示 1 字节的左循环移位，$rc[i]$表示轮常数，$rc[10]=\{0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1B,0x36\}$，$0\leqslant i\leqslant 9$。

对于轮加密模块，SubBytes 采用 16 个 S 盒进行并行计算，其运算过程如下：

$$sb[i]=SBox(sa[i]),0\leqslant i\leqslant 15\tag{14}$$

然后，将 SubBytes 的结果传递给 ShiftRows，并完成行移位置换操作，置换过程如下：

$$\begin{aligned}&\{sr[0],sr[1],sr[2],sr[3]\}=\{sb[0],sb[1],sb[2],sb[3]\}\\&\{sr[4],sr[5],sr[6],sr[7]\}=\{sb[5],sb[6],sb[7],sb[4]\}\\&\{sr[8],sr[9],sr[10],sr[11]\}=\{sb[10],sb[11],sb[8],sb[9]\}\\&\{sr[12],sr[13],sr[14],sr[15]\}=\{sb[15],sb[12],sb[13],sb[14]\}\end{aligned}\tag{15}$$

下面，将 ShiftRows 的结果传递给 MixColumn 子模块，完成域 $GF(2^8)$上的列混合变换操作，对应的数学表达式为：

$$\begin{bmatrix}b_{00}&b_{01}&b_{02}&b_{03}\\b_{10}&b_{11}&b_{12}&b_{13}\\b_{20}&b_{21}&b_{22}&b_{23}\\b_{30}&b_{31}&b_{32}&b_{33}\end{bmatrix}=\begin{bmatrix}2&3&1&1\\1&2&3&1\\1&1&2&3\\3&1&1&2\end{bmatrix}\begin{bmatrix}a_{00}&a_{01}&a_{02}&a_{03}\\a_{10}&a_{11}&a_{12}&a_{13}\\a_{20}&a_{21}&a_{22}&a_{23}\\a_{30}&a_{31}&a_{32}&a_{33}\end{bmatrix}\tag{16}$$

其在有限域上乘 0x02 和 0x03 的过程如下：

$$\begin{aligned}&mc2(x)=(x<<1)\wedge(8'h1b\&\{8\{x[7]\}\})\\&mc3(x)=mc2(x)\wedge x\end{aligned}\tag{17}$$

其中，$mc2(x)$表示域乘 0x02 后的结果，$mc3(x)$表示域乘 0x03 后的结果。那么对于其中第一列的操作如下：

$$\begin{aligned}&mc[31:24]=mc2(sr[0])\wedge mc3(sr[1])\wedge sr[2]\wedge sr[3]\\&mc[23:16]=sr[0]\wedge mc2(sr[1])\wedge mc3(sr[2])\wedge sr[3]\\&mc[15:8]=sr[0]\wedge sr[1]\wedge mc2(sr[2])\wedge mc3(sr[3])\\&mc[7:0]=mc3(sr[0])\wedge sr[1]\wedge sr[2]\wedge mc2(sr[3])\end{aligned}\tag{18}$$

其中，mc 表示 32 位宽的输出，第 2—4 列操作相同。

最后，将中间加密的结果与密钥扩展的结果经 AddRoundKey 子模块，完成异或操作并输出。为减少时钟周期数，AES 单轮加密采用 LUT 实现，并在 1 个时钟周期内计算出结果。AES 单轮加密的结构如图 6 所示。

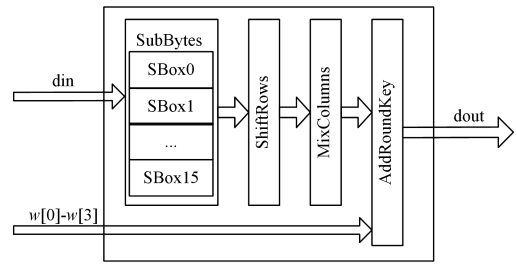

图 6 AES 单轮加密的结构

Fig. 6 Structure of AES single-round encryption

AES128 加密算法的前 9 轮结构相同，除了第 10 轮由字节变换、行移位变换、轮密钥加 3 步组成外，其他 9 轮都由字节变换、行移位变换、列混淆变换、轮密钥加 4 步构成。通过状态机以串行方式实现 AES128 加密，其整体结构如图 7 所示。而 AES128 解密算法[18]的结构基本相同，但需要逆序使用加密算法中各种变换的逆变换，并逆序使用各轮密钥，将密文转换为明文。

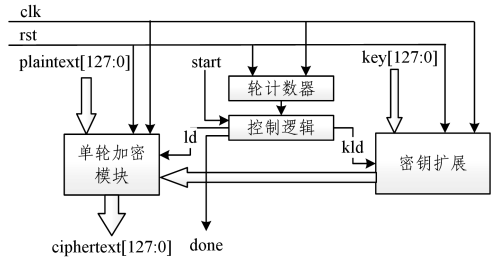

图 7 AES 串行加密结构

Fig. 7 Structure of AES serial encryption

3.2.3 高速口令的生成

随着用户安全意识的增强，口令长度日益增大，穷举口令空间的方式很难奏效[19]。而且加密算法的强度大大加强，新

的安全散列算法标准也已经确立，口令破解难度越来越大，这是一个无法回避的问题。这里采用基于口令结构字典穷举口令的方法，即将概率口令攻击和字典攻击结合在一起，以设定好的口令排列规则，在 FPGA 内部解析即时的生成口令，然后使用对应的目标算法进行验证，既提高了破解效率又降低了穷举量。

FPGA 高速口令的生成主要包括口令结构解析模块和口令生成模块，如图 8 所示。其中，口令结构解析模块将读取到的口令结构字典解析为能识别的符号，并将其缓存到 FIFO 中；口令生成模块根据解析后的规则对口令的每一位进行配置，然后生成口令。

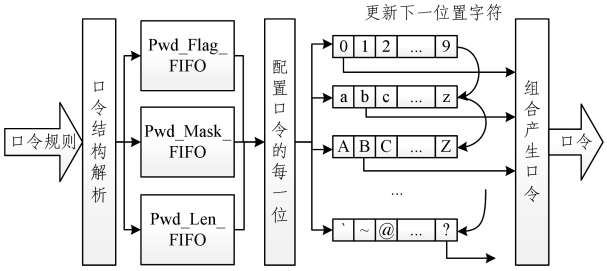

图 8　FPGA 高速口令生成结构图

Fig. 8　FPGA high-speed password generation structure diagram

具体地，对于口令结构解析模块，以“＃”表示明文、“?”表示掩码，则“?d”表示数字、“?l”表示小写字母、“? u”表示大写字母、“?s”表示特殊字符、“＃x”表示明文 x，“0x0a”为结束符。然后将解析后的数字、小写字母、大写字母和特殊字符分别标记为 1，2，3 和 4，并将对应口令位的穷举标志位置 1。明文为不需要穷举的字符直接传输，并将穷举标志位置 0。例如：“＃1＃2＃3?d?d?d”表示以 123 为前缀，穷举后 3 位数字，其口令空间为 123000－123999，则解析后对应的 FPGA 规则为“123111”，穷举标志位为“000111”。

口令生成模块在获取到掩码规则后，首先进行初始化操作，然后并行遍历口令每一位的字符，最后将所有字符按位置排列，生成口令。对应的流程如算法 1 所示。

算法 1　FPGA 口令生成算法

输入：口令标志位 pwd_flag，口令掩码 pwd_mask，口令长度 pwd_len

输出：候选口令 password

1. For i=0 to (pwd_len－1) do
2.　由 pwd_mask[i]初始化每一位字符配置；
3. End for
4. While (True) do
5.　For i=0 to (pwd_len－1) do
6.　　并行产生口令对应位置的字符；
7.　End for
8.　For i=0 to (pwd_len－1) do
9.　　由 pwd_flag[i]选择拼接明文还是掩码产生的字符；
10.　　将每一位字符拼接，生成 password；
11.　End for
12.　如果口令的最后一位字符全部更新完毕，跳出 While 循环；
13. End While

在算法 1 中，对应口令的每一位字符产生的流程如算法 2 所示。

算法 2　FPGA 口令字符生成算法

输入：口令掩码 pwd_mask_i，更新使能 pwd_update

输出：候选字符 pwd_char，下一字符更新使能 next_pwd_update

1. 由 pwd_mask_i 初始化 next_char 为{‘0’‘a’‘A’首个特殊字符}中的一个字符；
2. If(pwd_update)
3.　If (next_char ＝＝‘9’) next_char ＝‘0’；
4.　Else If (next_char ＝＝‘z’) next_char ＝‘a’；
5.　Else If (next_char ＝＝‘Z’) next_char ＝‘A’；
6.　Else If (next_char ＝＝当前特殊字符) next_char ＝下一特殊字符；
7.　Else next_char＝next_char＋0x01；
8. End If
9. pwd_char＝next_char；
10. If(pwd_char ＝＝ 最后 1 个字符) next_pwd_update＝1；
11. Elsenext_pwd_update＝0；

算法 2 通过更新使能 pwd_update 和下一字符更新使能 next_pwd_update，将口令每一位字符前后关联。当前位置字符枚举完毕，则更新下一位置字符。如此循环往复，直至口令最后一位字符更新完毕，整个口令生成算法结束。

3.3　FPGA 整体结构

3.3.1　整体结构组成

基于 FPGA 的 Office 口令破解算法主要由 6 个模块组成，如图 9 所示。其中上位机通过 PCIE 与 FPGA 进行通信，并对 FPGA 进行配置。同时，FPGA 将自身状态实时上报给上位机，且一旦找到正确口令，就立马通知上位机并停止计算。这里采用 GALS(Global Asynchronous Local Synchronous)架构，放置多个恢复模块，每组恢复模块独立执行，可支持同一掩码多个任务或不同掩码同一任务等多种恢复模式。

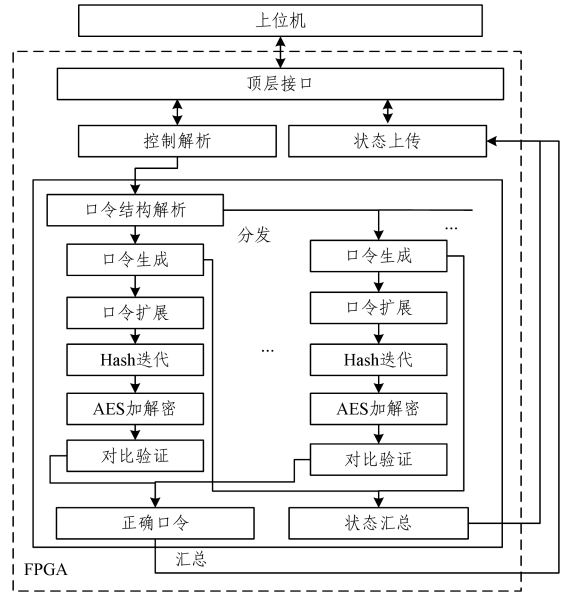

图 9　FPGA 的整体结构

Fig. 9　Overall structure of FPGA

各个模块的功能如下：

1）顶层接口模块。上位机输入数据到顶层接口，把数据传给寄存器，并将数据依次分发给需要的模块。

2)控制解析模块。控制各个模块的初始化配置和数据通信。

3)状态上传模块。收集 FPGA 各个模块的工作状态,并上传给上位机。

4)口令结构解析模块。将口令结构字典解析后的掩码规则依次分发给各口令生成模块。

5)口令生成模块。由掩码规则生成口令并传递给核心算子进行运算。

6)核心算子模块。核心算子模块由口令扩展、Hash 迭代、AES 加解密和对比验证子模块组成,提供完整的 Office 破解运算。

3.3.2 I/O 控制优化

Office 核心算子模块的 Hash 迭代采用流水线实现,一次可输入多组口令。为充分发挥流水线的性能,需要对 Hash 迭代前后关联的模块进行 I/O 和控制优化。其次,为防止口令传递过程中发生溢出或丢失,需要通过 FIFO 和状态机进行控制。再次,由于 Hash 迭代每次都需要对上一轮结果进行字符拼接,并将其作为本次迭代的输入,因此需要对 Hash 流水线进行循环迭代计数,并据此控制流水线的初始化、中间缓存和最后的结果输出。最后,为进一步提高计算速度,在满足 Hash 迭代吞吐量的情况下,可适当降低其他模块的频率,减小布线复杂度以提高 Hash 迭代的频率。

图 10 给出了 Hash 流水线的字符拼接控制。其通过流水线计数器控制 Hash 流水线的输入和输出,当完成一次完整的流水线运算时,循环迭代计数器加 1。当循环迭代计数器累加到一定次数时,判断是否输出。

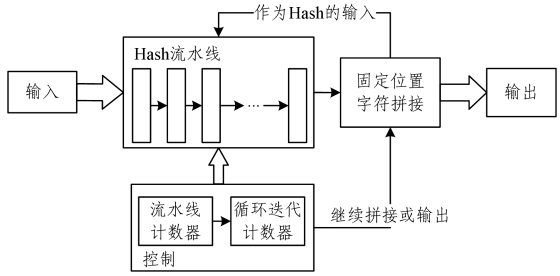

图 10 Hash 迭代字符拼接控制

Fig. 10 Character splicing control of Hash iteration

图 11 给出了各模块间的 I/O 数据流和控制。各模块间通过 FIFO 松耦合互连,并配合 FIFO 读写控制,完成数据的传递,其中口令 FIFO 和口令缓存 FIFO 为同步 FIFO,口令扩展 FIFO 和 Hash 结果 FIFO 为异步 FIFO。其次,通过异步 FIFO 将时钟域隔离,使口令计数器和 Hash 迭代工作在高频,而其他模块工作在低频,互不干扰。再根据口令计数器的控制,每次从口令扩展 FIFO 中读取一组数据并将其送入 Hash 流水线进行计算。最后,每次从 Hash 结果 FIFO 中读取 1 个数据进行处理,并在验证对比时同步地从口令缓存 FIFO 中读取 1 个口令,将口令与验证结果绑定。如果验证通过,则输出正确口令。

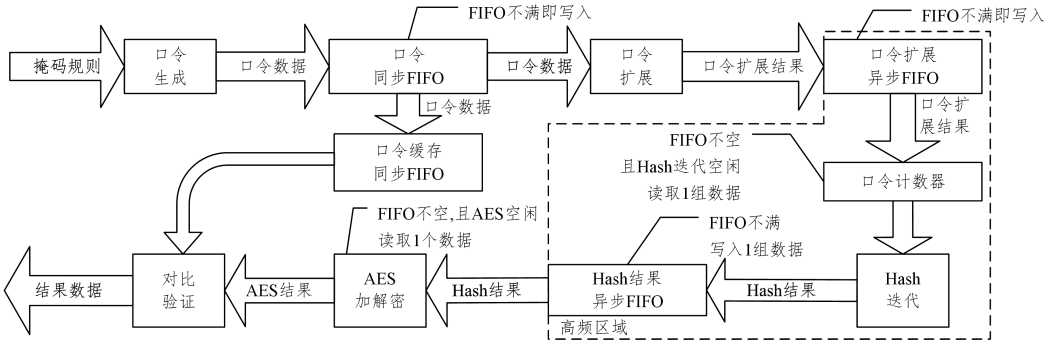

图 11 各模块间的 I/O 数据流和控制

Fig. 11 I/O data flow and control between modules

3.4 多核 FPGA 集群

FPGA 集群通过识别 Office 口令恢复应用的需求、应用的变化,依据尽可能高效的原则,构建出适合于应用需求的处理结构。图 12 描述了 FPGA 集群的整体设计框架,包含客户端特征串提取单元、任务分发单元、口令结构字典单元、掩码规则分发单元、状态管理单元、结果入库单元、FPGA 服务器烧录和配置单元、状态结果汇报单元。特征串提取和任务分发单元主要负责任务的解析与分发。口令结构字典和掩码规则分发单元主要负责读取口令规则与分发。状态管理单元主要负责收集 FPGA 运行时的状态。结果管理单元将 FPGA 计算出的结果进行汇总并入库。烧录配置单元会根据任务类型选择相应配置文件对 FPGA 进行烧录。

FPGA 集群系统的运行流程如下:

1)用户提交待恢复的 Office 加密文档,并配置口令结构字典。

2)解析加密 Office 文档,提取特征串并添加到任务队列。

3)客户端从任务队列读取任务,并将其下发到各 FPGA 服务器。

4)FPGA 服务器根据配置文件选择对应的 bitstream 文件,并对 FPGA 加速卡进行烧录。烧录完成后,通知客户端。

5)客户端依次读取口令结构字典中的掩码规则,并分发到各 FPGA 服务器。

6)FPGA 服务器根据掩码规则、计算速度和 FPGA 数量,对掩码规则进行二次切分,并再依次分发到各 FPGA。

7)FPGA 接收到掩码后开始计算,并上报中间状态到 FPGA 服务器。FPGA 服务器汇总后,再上报给客户端。

8)当找到正确口令时,系统停止计算;否则,跳转步骤 5),直至口令结构字典读取完毕。

由于分发的掩码规则对应的口令空间可能过大,导致部分 FPGA 长时间处于工作状态,而其他 FPGA 却空闲。因

此，在步骤 6）中对掩码规则进行二次切分，以平衡负载，切分的具体规则如下：

1）由掩码规则 $Mask=(c_n,\cdots,c_2,c_1)$ 计算口令空间 $pwd_{total}=\prod_{i=1}^{n}c_i$。

2）根据单算子计算速度 s、切分掩码工作时间 t，对 pwd_{total} 进行划分，有 $pwd_{\text{single}}=s\times t$。

3）根据 pwd_{single} 从 $Mask$ 中取至少 k 个字符 $Mask_{\text{part}}=(c_k,\cdots,c_2,c_1)$，使 $\prod_{i=1}^{k}c_i\leqslant pwd_{\text{signle}}$，并取 k 的最大值。

4）将 $Mask$ 表示为 $Mask'=(c_n,c_{n-1},\cdots,c_{k+1},Mask_{\text{part}})$，每次按照 $Mask_{\text{part}}$ 口令空间大小对 $Mask'$ 进行切分。

5）根据 FPGA 数 n、算子数 m，将切分好的掩码进行分组，生成 n 个口令结构字典，每个字典含有 m 个掩码规则，并配置到对应的 FPGA，直到掩码全部分配完毕。

最后，每当有新的 FPGA 服务器加入集群时，新加入的 FPGA 服务器会主动向客户端发送自己的 IP 地址，并将其加入到计算中。同时，客户端可同时配置多个不同的任务，并下发到不同的 FPGA 服务器。并且 FPGA 服务器也可同时烧录不同的 bitstream 文件，即每个 FPGA 加载不同的 Office 口令恢复算法。因此，FPGA 集群支持多任务的混合计算，具有更好的灵活性和可扩展性。

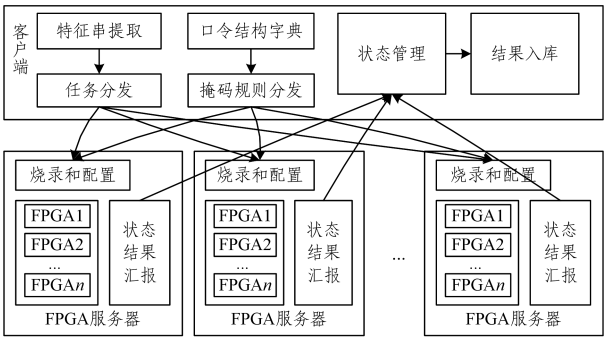

图 12　FPGA 集群的整体架构图

Fig. 12　Overall architecture of FPGA cluster

4　实验结果与分析

本文实验的硬件平台为 FPGA 加速卡，芯片型号为 Xilinx 公司的 xcku060-ffva1156-2-i，其查找表 LUT 资源是 331680，FlipFlops 寄存器资源是 663360；GPU 型号为 NVIDIA GTX 1080；CPU 为 Intel i5-7500 处理器。软件平台为集设计、仿真、综合、布线、生成于一体的 Vivado 2018.2 软件。实验通过对算法的各模块进行深度优化，给出了资源占用，并与 CPU、GPU 和其他方案进行了性能对比分析。其次，给出了口令切分测试。最后，针对不同规模的 FPGA 集群服务器，给出了恢复效率的分析。

4.1　各模块的实现

Office 各版本核心算法的实现情况如表 3 所列。其中 SHA1 和 SHA512 采用流水线方式实现，AES128 和 AES256 采用串行方式实现，以串并混合操作，提高整体的计算性。

表 3　Office 各版本核心算法的实现情况

Table 3　Implementation of core algorithms in each version of Office

模块	实现方式	计算周期	LUTs	REGs	RAMs
SHA1	流水线	83	15588	26225	0
SHA512	流水线	83	70840	102274	0
AES128 加解密	串行	12/23	771/765	665/668	4/4.5
AES256 加解密	串行	16/33	1775/2865	1192/2735	0

表 4 以 Office 2010 为例，列出了口令生成、口令扩展、AES 加解密、Hash 迭代和对比验证的实现情况。

表 4　Office 2010 各模块的实现情况

Table 4　Implementation of Office 2010 modules

模块	实现方式	LUTs	REGs	RAMs
口令结构解析	串行	263	808	4
口令生成	并行	1174	845	2.5
口令扩展	串行	2295	2131	0
Hash 迭代	流水线	15754	27644	5
AES 加解密	串行	3279	3772	8.5
对比验证	串行	1	169	0

从表 3 可以看出，单个 SHA1，SHA256，AES128，AES256 模块占用资源适中，完全满足 FPGA 设计需求。从表 4 中看出，Office 2010 各模块以串并混合结构实现，其中口令生成和 Hash 迭代分别以并行、流水线方式实现，保证了口令验证速度，其他模块采用串行实现。由于 Office 核心运算的迭代次数多，运算时间长，其他模块采用串行方式实现，在满足计算需求的情况下，不仅减少了资源占用量，而且更有利于提高核心 Hash 迭代的时钟频率。表 5 列出了 Office 2010 各模块间的 FIFO 配置和频率。

表 5　Office 2010 各模块间的 FIFO 配置

Table 5　FIFO configuration between Office 2010 modules

模块	通信方式	写频率/MHz	读频率/MHz	位宽	深度
口令 FIFO	同步	50	50	168	32
口令缓存 FIFO	同步	50	50	168	512
口令扩展 FIFO	异步	50	250	160	128
Hash 结果 FIFO	异步	250	50	160	128

表 5 中，口令 FIFO 深度最小，主要用于局部缓存口令，最长支持 20 字节的口令。口令缓存 FIFO 由于需要缓存口令扩展对应的口令、流水线中运算的口令、Hash 结果对应的口令，因此深度最大。而口令扩展 FIFO 和 Hash 结果 FIFO 只需存储一组数据，因此深度为 128。

Office 各版本口令恢复算法的实现情况如表 6 所列。其中 Office 2007/2010 每个算子均由 2 个 SHA1 流水线组成，Office 2013－Office 2019 由 1 个 SHA512 流水线组成。

表 6　Office 各版本口令恢复算法的实现情况

Table 6　Implementation of password recovery algorithms in each version of Office

算法类型	算子数	Hash 迭代频率/MHz	LUTs	REGs	RAMs
Office 2007	7	250	233133	331680	24.12
Office 2010	7	250	272478	472310	256.50
Office 2013－Office 2019	3	200	247090	331680	42.5

从表 6 可以看出，Office 口令恢复算法均采用了多算子

并行的方式实现，并运行在较高的频率，保证了计算的高效能，并充分利用了 FPGA 的资源。

4.2 性能对比分析

描述口令恢复硬件平台的性能指标有性能、功率、能效比。其中性能指平台单位时间(s)内的口令恢复速率，简记为 pwd_speed。功率用于计算硬件设备工作时的能耗，简记为 u_power。能效比[20]表示平台性能与设备功率之比，简记为 EER(Energy Efficiency Ratio)，其计算公式如下：

$$EER=\frac{pwd\_speed}{u\_power} \tag{19}$$

表 7 列出了 FPGA 加速卡的 Office 各项性能指标。可以看出，FPGA 加速卡不仅性能较高，且功耗较低，具有很高的能效比，适合大规模使用。

表 7 FPGA 加速卡的 Office 性能指标

Table 7 Office performance indicators of FPGA accelerator card

算法类型	性能/(个/秒)	功耗/W	能效比/(个/焦耳)
Office 2007	277788	148	1876.95
Office 2010	138600	171	810.53
Office 2013—Office 2019	23952	112	213.86

本文将本文方案与其他方案的性能进行了对比，结果如表 8 所列。

表 8 本文方案与其他方案的对比

Table 8 Comparisonof proposed schemes and other schemes

方案	算法	计算部件	速度
文献[21]	Office 2007	NVIDIA GTX 1080	134188
	Office 2010		66925
	Office 2013		8860.5
文献[22]	Office 2007	XC7Z030-3	19635
	Office 2010		9837
	Office 2013		979
本文	Office 2007	xcku060×4	277788
	Office 2010		138600
	Office 2013		23952

显然，本文方案较其他方案更具有优势，充分发挥了 FPGA 的计算优势。文献[22]的 Office 2007/2010/2013 的能效比分别为 1589.88，783.82 和 91.92，而本文的能效比分别为 1876.95，810.53 和 213.86，因此本文方案明显高于文献[22]。

进一步，从计算部件、口令策略、加速技术、集群管理和负载均等方面，将本文方案与其他方案进行了综合实现对比，结果如表 9 所列。可以看到，在相同的计算部件和加速技术下，本文方案较其他方案更为完善，不仅支持掩码规则的口令策略，还以两次的口令切分实现了更为均衡的负载，能更好地在实际中应用。

表 9 本文方案与其他方案的综合对比

Table 9 Comprehensive comparison of proposed schemes and other schemes

方案	计算部件	加速技术	口令策略	集群管理	负载均衡
文献[23]	FPGA	流水线	穷举	支持	简单
文献[24]	FPGA	流水线	—	支持	—
本文	FPGA	流水线	掩码规则	支持	两次切分

[1] https://hashcat.net/hashcat/

4.3 口令切分测试

为保证 FPGA 集群的高效工作，本节对口令切分进行了测试。假设掩码为?l?l?l?l?d?d?d?d?d?l，以 20 min 为标准，对 1 块加速卡进行切分，结果如表 10 所列。

表 10 掩码口令切分

Table 10 Mask password split

算法类型	掩码总数	掩码示例	掩码空间	单算子计算时间
Office 2007	456976	#a#a#a#a?d?d?d?d?d?l	2600000	约 4.4 min
Office 2010	456976	#a#a#a#a#a?d?d?d?d?d?l	2600000	约 8.7 min
Office 2013—Office 2019	4569760	#a#a#a#a#a#0?d?d?d?d?l	260000	约 2.2 min

从表 10 可以看出，切分后的掩码单算子的计算时间都在 20 min 以内，如果再多切分一位口令，则会超出 20 min。显然，如果?l?l?l?l?d?d?d?d?d?l 没有经过切分直接送入 FPGA，则对于单算子来讲，需要至少几年的时间才能全部计算完毕。

进一步，如果使用 2 台服务器，每台 4 块加速卡，共 8 块 FPGA 加速卡，共 32 个 FPGA，口令切分会产生 32 个口令结构字典，每个字典含有 3～7 组掩码规则。这样，每台服务器接收 16 个字典，每个加速卡配置 4 个字典，每个 FPGA 配置 3～7 组掩码，如此循序切分，完成整个集群的负载均衡。

4.4 FPGA 集群分析

本节通过 2 台 FPGA 服务器搭建口令恢复集群系统，以及 2 台 CPU+GPU 服务器(8 块 GPU)使用 Hashcat v3.60[1]搭建口令恢复集群系统，以 Office 为恢复对象，对比 2 套系统的性能，其结果如图 13 所示。

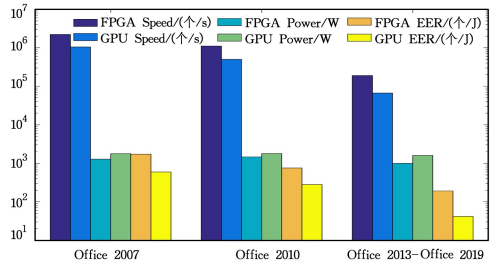

图 13 FPGA 集群与 GPU 对比结果

Fig. 13 Comparison between FPGA cluster and GPU

从图 13 可以看出，Office 各版本的 FPGA 集群系统性能分别是 GPU 集群性能的 2.11 倍、2.20 倍和 2.85 倍，功耗是 0.72 倍、0.82 倍和 0.62 倍，能效比是 2.91 倍、2.69 倍和 4.57 倍，较 GPU 更有优势。由于 FPGA 具有很强的并行计算和位运算能力，可定义任意宽度的寄存器，且由门电路组成的逻辑有着更高的计算能力。显然，使用 FPGA 加速卡搭建的集群系统更具有实用性。

结束语 本文实现了基于 FPGA 集群的 Office 口令恢复系统。通过对 Office 加密机制的分析，在 FPGA 上优化了核心算法，并设计了多算子并行的结构，提高了 FPGA 的整体性能。相比 GPU，本文实现的 FPGA 加速卡在口令恢复速度上提高了 2.11 倍以上，在能效比上提高了 2.69 倍以上，具

有很明显的优势。同时 FPGA 加速卡功耗低,更适合在云端搭建 FPGA 集群,为口令恢复提供了更强的算力。

但是随着 FPGA 集群规模的扩展,任务的调度管理和系统结构将成为制约因素。因此,如何实现 FPGA 集群的高效管理,搭建合适的网络拓扑结构并实现异构计算,充分发挥 FPGA 的计算性能,提高工作效率,将是下一步研究的主要工作。同时,如何进一步提高系统的口令恢复效率,对基于统计学的社工字典也亟待研究。

参 考 文 献

[1] HONG J,CHEN Z,HU J. Analysis of encryptionmechanism in Office 2013[C]// 2015 IEEE 9th International Conference on Anti-counterfeiting,Security,and Identification (ASID). IEEE,2015:29-32.

[2] HRANICK R,MATOUŠEK P,RYŠAV O,et al. Experimental evaluation of password recovery in encrypted documents[C]// Proceedings of ICISSP. 2016:299-306.

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

[4] KAKARLA T,MAIRAJ A,JAVAID A Y. A Real-World Password Cracking Demonstration Using Open Source Tools for Instructional Use[C]// 2018 IEEE International Conference on Electro/Information Technology (EIT). IEEE,2018:0387-0391.

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

[6] WANG D,ZHANG Z J,WANG P,et al. Targeted Online Password Guessing:An Underestimated Threat[C]// ACM Sigsac Conference on Computer and Communications Security. ACM,2016:1242-1254.

[7] WANG D,WANG P,HE D,et al. Birthday,Name and Bifacial-security:Understanding Passwords of Chinese Web Users[C]// 28th USENIX Security Symposium. 2019:1537-1554.

[8] 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.

[9] WIRTHLIN M. High-reliability FPGA-based systems:space,high-energy physics,and beyond[J]. Proceedings of the IEEE,2015,103(3):379-389.

[10] LI B,ZHOU Q,SI X. Mimic computing for password recovery [J]. Future Generation Computer Systems,2018,84:58-77.

[11] MS-CFB:Compound File Binary File Format (v2018091) [EB/OL]. https://docs. microsoft. com/zh-cn/openspecs/ windows_protocols/ms-cfb.

[12] MS-OFFCRYPTO:Office Document Cryptography Structure (v20181211) [EB/OL]. https://docs. microsoft. com/en-us/openspecs/office_file_formats/ms-offcrypto/.

[13] SONG J,SUN Z Z,LI T T,et al. Research Advance on Code Oriented Optimization of Software Energy Consumption[J]. Chinese Journal of Computers,2016,39(11):2270-2290.

[14] MICHAIL H E,ATHANASIOU G S,KELEFOURAS V I,et al. Area-throughput trade-offs for SHA-1 and SHA-256 hash functions' pipelined designs[J]. Journal of Circuits,Systems and Computers,2016,25(4):1-27.

[15] SUHAILI S,WATANABE T. High throughput evaluation of SHA-1 implementation using unfolding transformation [J]. ARPN Journal of Engineering and Applied Sciences,2016,11(5):3350-3355.

[16] WONG M M,WONG D M L,ZHANG C,et al. Circuit and system design for optimal lightweight AES encryption on FPGA [J]. IAENG International Journal of Computer Science,2018:45(1):52-62.

[17] HAFSA A,SGHAIER A,MACHHOUT M,et al. A New security Approach to Support the operations of ECC and AES Algorithms on FPGA[C]// 2019 19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA). IEEE,2019:95-100.

[18] RAO M,KAKNJO A,OMERDIC E,et al. An efficient high speed AES implementation using Traditional FPGA and LabVIEW FPGA platforms[C]// 2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). IEEE,2018:932-937.

[19] DING Q,ZHANG Z,LI S,et al. Energy-Efficient RAR3 Password Recovery with Dual-Granularity Data Path Strategy[C]// 2019 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE,2019:1-5.

[20] BAI X,JIANG L,YANG J,et al. Password Recovery for ZIP Files Based on ARM-FPGA Cluster[C]// 2017 International Conference on Security,Privacy and Anonymity in Computation,Communication and Storage (SpaCCS). 2017:405-414.

[21] HRANICK R,ZOBAL L,VEEA V,et al. Distributed Password Cracking in a Hybrid Environment[C]// Proceedings of SPI. 2017:75-90.

[22] LIU P,LI S,DING Q. An energy-efficient accelerator based on hybrid CPU-FPGA devices for password recovery[J]. IEEE Transactions on Computers,2018,68(2):170-181.

[23] HAN Y,ZHOU Q L,LI B,et al. High-performance VPN Password Recovery Method on Multiple FPGAs[J]. Journal of Chinese Computer Systems,2019,40(4):79-84.

[24] CHEN X J,ZHOU Q L,LI B. EnergyGefficient Password Recovery Methodfor7GZip Document Basedon FPGA[J]. Computer Science,2020,47(1):321-328.

LI Bin,born in 1986,Ph.D,lecturer. His main research interests include high-performance computing and information security.