IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 29, 2020, accepted January 14, 2021, date of publication January 20, 2021, date of current version February 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3053068

Reconfigurable and High-Efficiency Password Recovery Algorithms Based on HRCA

BIN LI[1], FENG FENG[1], XIAOJIE CHEN[2], AND YAN CAO[1]
[1]School of Information Engineering, Zhengzhou University, Zhengzhou 450001, China
[2]State Key Laboratory of Mathematical Engineering and Advanced Computing, PLA Strategic Support Force Information Engineering University, Zhengzhou 450001, China

Corresponding author: Bin Li (cctvlibin@163.com)

ABSTRACT Cryptographic algorithms are widely used in information security fields such as network protocol authentication and commercial encryption software. Password recovery based on the hash algorithm is an important means of electronic forensics, encrypted information restoration, illegal information filtering, and network security maintenance. The traditional password recovery system is based mainly on the CPU and GPU and has a low energy efficiency ratio and cracking efficiency and cannot meet high-performance computing requirements. To further improve the computational efficiency and application flexibility of password recovery algorithms, this paper proposes a reconfigurable computing kernel design method based on a hybrid reconfigurable computing array (HRCA). Through in-depth analysis of the hash algorithm, the basic computing kernel set is extracted, and the combination design is carried out from the unit kernel, interconnection and storage structure to reconstruct the hash algorithm to match the application with the appropriate structure. Second, combined with the pipeline technology, the full pipeline hash and high-speed password attack algorithms are optimized and implemented to meet the needs of high-performance computing. Finally, an advanced computing kernel library is established, and the combination of a computing kernel map from the control and communication levels to achieve multidimensional reconfigurable computing and an overall placement strategy is used to make full use of the chip resources to improve computational efficiency. The experimental results and analysis show that compared with traditional CPU and GPU methods, the password recovery algorithm designed in this paper has the highest cracking speeds at 78.22 times and 2.65 times that of the CPU and GPU, respectively, and the highest energy efficiency ratio is 25.88 times and 3.16 times that of the CPU and GPU, respectively. Furthermore, the recovery efficiency has been significantly improved and meets the requirements of high-performance password recovery computing.

INDEX TERMS HRCA, computing kernel, reconfigurable, hash algorithm, password recovery.

I. INTRODUCTION

At present, password-based authentication methods are still widely used in various information systems. To prevent user information from being eavesdropped or leaked, the hash function is often used in an authentication protocol to authenticate a password [1]–[5]. To effectively recover network protocols and application passwords based on the hash algorithm; provide support for electronic forensics, information intelligence acquisition, criminal record review, and encrypted data behavior analysis; and evaluate the security of the new password hashing scheme [6], the optimization and implementation of hash algorithms have become a research hotspot. In many applications, processing speed and energy efficiency have become important indicators. However, the hash algorithm itself is very complicated and requires a large amount of calculation. Therefore, whether for a software or a hardware implementation, optimization of the execution frequency and energy consumption represents difficult points in the design.

In the software implementation, traditional password recovery based on the CPU architecture is limited by its computational speed for cryptographic algorithms and can crack only passwords with low complexity. For GPUs, high-throughput password recovery algorithms such as SHA1 and MD5 are implemented by means of multicore parallel computing [7]–[9]. However, the GPU has high power consumption and a low energy efficiency ratio. For FPGAs, different

The associate editor coordinating the review of this manuscript and approving it for publication was Mohamad Afendee Mohamed.

FPGA devices have different structures, the RTL-level design and placement requirements of the algorithm are different, and a variety of different schemes have been implemented [10]–[13]. However, most FPGA schemes do not consider the multimodule parallel situation and do not mention how to effectively partition and place the algorithm. The scalability is poor, which leads to variable algorithm performance. The password recovery task requires a large amount of calculation and obvious changes in the calculation cycle, which are difficult for existing computing structures to work with. It is necessary to seek a more efficient computing structure. Therefore, how to use innovative technology to construct a highly efficient and dynamically variable algorithm in the field of password recovery must be solved.

The development of computational granularity from complex instructions to today's instruction set and then to macro instructions shows that computational granularity is the most critical aspect to improve processing performance. To achieve high-efficiency and high-performance computing, mimic computing [14] proposes a multidimensional reconfigurable architecture based on a hybrid reconfigurable computing array (HRCA) from the perspective of architecture innovation. HRCA uses a coarse-grained basic computing kernel and reconfigurable interconnection to form a high-order computing segment, which can implement applications in a parallel and pipeline-efficient way, considering both computational performance and flexibility. Compared with bit-level operations, the computing kernel is stronger and more suitable for calculation-intensive applications. Compared with the instruction set, the computing kernel does not need instruction fetching and instruction decoding and thus has better execution efficiency. In addition, unlike CPU and GPU instruction processing, by adding a look-up table (LUT) and register hardware resources, the computing kernel can operate multiple serial instruction operations in one clock, simplifying and accelerating FPGA calculations. Second, the computing kernel can be reconstructed to form a unit kernel, and then the unit kernel can interconnect to form a functional module, which is coarse-grained reconfigurable. Finally, HRCA is equivalent to the ''dedicated instruction set'' of the reconfigurable FPGA, which provides technical support for the determination of special-purpose kernels and the generation of reconfigurable hardware functional units for domain applications. With FPGA-reconfigurable devices, HRCA can directly map the computing kernel to the hardware logic, which can make fuller use of FPGA resources, simplify the complexity of FPGA programming, increase the frequency of FPGA work, and improve the flexibility and scalability of FPGAs, thereby realizing efficient computing.

This paper is based on the HRCA design method, and through in-depth analysis of the features of the hash algorithm, the basic computing kernel set is extracted. In addition, a pipeline structure is formed by the computing kernel reconstruction approach to achieve the purpose of increasing the working frequency and parallel optimization. Thus, this approach improves the computational speed and reduces the energy consumption. At the same time, a high-speed password generation algorithm is realized by using the password guessing attack model, and a high-performance password recovery system is designed using the global asynchronous local synchronous (GALS) architecture, which provides strong support for password-based authentication and computer forensics.

The main contributions of this paper include the following:

1) A description of the model and structure of HRCA is provided that applies HRCA to the field of password recovery, gives a complete design method, and builds a multidimensional reconfigurable system.
2) The extraction algorithm of the basic computing kernel is given, and a method of kernel reconstruction is proposed from three aspects of the unit kernel structure, interconnection structure and storage structure. Moreover, the algorithm is matched with an appropriate structure.
3) To meet the computational requirements of the password recovery algorithm, the hash algorithm and password generation algorithm are designed using pipeline technology, including mask rules and high-speed dictionary parsing. A single module can generate 200 M passwords per second and complete the hash calculation.
4) An advanced computing kernel library is established for the algorithm kernel, control kernel and communication kernel, and the parallelism of the password recovery algorithm with the overall placement strategy is improved, greatly reducing the difficulty of hardware parallel program development.
5) Compared with the CPU, the method in this paper improves the cracking speed and energy efficiency ratio by more than 20 times and the energy efficiency ratio of the GPU by 3.16 times, thus improving password recovery significantly in terms of performance, energy efficiency, and scalability.

The organization of this paper is as follows. Section 2 introduces the research of HRCA and password recovery algorithms and presents the research motivation. Section 3 describes the design of password recovery algorithms based on HRCA in detail. Section 4 tests and evaluates the scheme to verify its effectiveness. Section 5 discusses the scope and limitations of the method. Finally, Section 6 summarizes this paper and outlines the prospects for ongoing work.

II. RELATED WORK

A. HRCA OVERVIEW

Based on reconfigurable computing, mimic computing conducts an in-depth analysis of the application, structure, and effectiveness relationships of high-performance computing in multiple typical fields and proposes the concept of ''application determines structure, structure determines efficiency''. Mimic computing breaks through the traditional concept of FPGA reconstruction and has been comprehensively

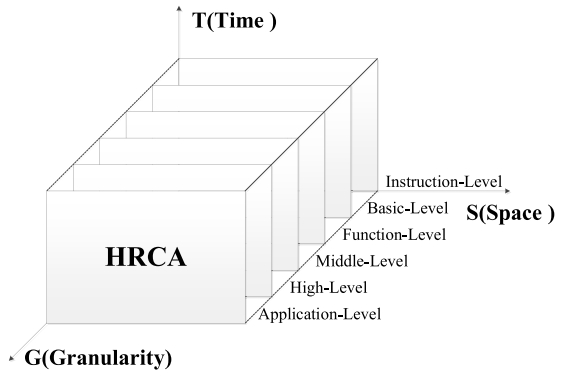


Fig. 1. TSG computing structure of HRCA.

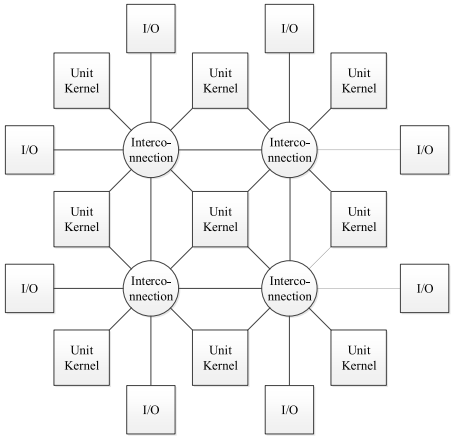


Fig. 2. HRCA structure of 9 unit kernels.

expanded in the dimensions of reconstruction granularity, timing breadth, fitness depth, and initiative. Logic gates, logic blocks, IP cores, and components that have been extended to subsystems are all multidimensional reconstruction components [15]. HRCA, as the key technology of mimic computing, introduces the concept of computing granulation based on time and space, that is, the time space granularity (TSG) structure, as shown in Fig.1. Computing granulation divides the computational processing into multiple execution fragments with different computing scales and aggregates these fragments to form the computing kernel. The computing kernel is the hardware implementation form of computing granulation. As the basic functional unit of HRCA, its scale is driven by the target application. It can be reconstituted into different numbers, granularities and functions of the arithmetic according to the application requirements, thereby achieving higher unit utilization and addressing efficient computing and flexibility. Therefore, HRCA is a computing structure with variable computing granularity, which is of great significance for improving the efficiency and resource utilization of computing systems.

At present, mimic computing has achieved good research results in the fields of Web services [16] and blockchain [17]. This paper is oriented to the field of password recovery, with an HRCA multidimensional variable computing structure, in terms of computational granularity, placement mapping, and cracking speed, to further improve the high efficiency and scalability of password recovery algorithms and better meet the diverse application requirements.

1) STRUCTURE OF HRCA

To achieve high-efficiency computing, HRCA uses the execution mode of the computing kernel to implement applications with flexible unit granularity and a fully parallel structure. The computing kernel implements a computing architecture with hardware circuits and then continuously inputs the data stream into the system and completes the calculation, which belongs to ''structural computing'' or ''spatial computing''. It is obviously different from the von Neumann structure based on the storage program, which can greatly improve the execution efficiency. Finally, through a basic computing kernel set, it is possible to construct a reconfigurable unit kernel structure with a hybrid of coarse and fine granularity. The global asynchronization is used between unit kernels, and local synchronization is used within the unit kernels, that is, the GALS on-chip system architecture, to achieve high scalability. In Fig.2, the HRCA structure of 9 unit kernels is shown.

For HRCA, the large-scale unit kernel structure forms a high-order computing segment that can support the various submodules of the application algorithm. In addition, the basic computing kernel set of HRCA is oriented to specific application fields, and a library-based mapping method can be adopted to solve the problem of efficient mapping from an application to the reconfigurable structure.

B. PASSWORD RECOVERY ALGORITHM

In recent years, with the development of hardware technology, many researchers have conducted substantial related research on the high-speed implementation of password recovery algorithms on GPUs and FPGAs.

In terms of GPUs, Qiu *et al.* [18] designed and optimized the SHA1 and MD5 password recovery algorithms on the GPU and improved the password generation algorithm to match the computing speed. Barbieri *et al.* [19] built a hierarchical and heterogeneous environment using multiple GPUs of different models, proposed a parallelized general exhaustive search model and optimized the MD5 and SHA1 algorithms to leverage GPU performance. Chen *et al.* [20] optimized the MD5 Crypt algorithm using CUDA programming. Aggarwal *et al.* [21] implemented the Bcrypt and SHA512 algorithms by using distributed cloud computing. Ge *et al.* [22] optimized the exhaustive attack of the SHA512 algorithm on the GPU. Dürmuth and Kranz [23] implemented two types of hash passwords, Bcrypt and Scrypt, on a GPU and FPGA, respectively. Dev [24] compared the performance of the SHA256 algorithm used in bitcoin mining on a CPU and GPU. Dev [25], [26] proposed

the use of botnets that were combined with the CPU and GPU to achieve large-scale hash password recovery.

In terms of FPGAs, Shi *et al.* [27] introduced hardware optimization technology and implemented MD5, SHA256 and SHA3 algorithms. Kim *et al.* [28] implemented a high-throughput SHA1 algorithm on Xilinx Virtex-6 LX240T through loop unfolding and preprocessing. Suhaili and Watanabe [29] designed two implementations of SHA1 on the Arria II GX EP2AGX45DF29C4, namely, the 80-cycle and 40-cycle unfolded SHA1 structure, with a maximum frequency of 274.2 MHz. Suhaili and Watanabe [30] implemented four MD5 algorithms with different pipeline stages, with a maximum frequency of 290.53 MHz. Ioannou *et al.* [31] optimized the SHA3 algorithm with a 2-stage pipeline on different FPGA devices. Mohamed and Nadjia [32] optimized the SHA256 algorithm through 6-input LUTs and the carry-save adder (CSA) and compared the delays of the two schemes. Maashri *et al.* [33] used reconfigurable hardware optimization to implement the XTEA and SHA512 algorithms. Huo and Liu [34] designed an application-specific instruction-set processor through hardware and software to realize various high-performance hash algorithms and meet the requirements of low cost and programmability. Mestiri *et al.* [35] optimized the SHA256 and SHA512 serial algorithms on the Xilinx Virtex-5 FPGA to improve the performance/area ratio. Rote *et al.* [36] optimized the structure of each round operation of SHA256 and SHA512 through the round pipelined technique to increase the algorithm frequency. Michail *et al.* [37], [38] aimed at the SHA1 and SHA2 series hash algorithms using a variety of FPGA optimization techniques to achieve high-throughput and area-efficient multimode architectures, which can support single- or multimode algorithms of SHA1, SHA256 and SHA512. Tyler *et al.* [39] introduced an expandable architecture targeted for an FPGA-based platform for recovering WPA/WPA2 passphrases. Kammerstetter *et al.* [40] presented a highly optimized low-cost FPGA cluster-based WPA2 personal password recovery system. Ding *et al.* [41] proposed a dual-granularity data path adjustment strategy to design accelerators for RAR3 password recovery. Bai *et al.* [42] proposed a highly scalable distributed structure based on Zynq SoC to implement a password recovery system for WinZip encrypted files.

In terms of high-level synthesis (HLS) programming, Jin and Finkel [43] implemented an 8-module parallel MD5 algorithm on an Intel Arria 10 GX1150 FPGA by using OpenCL programming. Vallina and Gilliland [44] optimized the SHA1 algorithm by OpenCL programming. Jacinto *et al.* [45] optimized the pipelined and nonpipelined SHA3 algorithms on the Xilinx Zynq-7000 SoPC through HLS technology combined with C language programming.

Regarding algorithms for password guessing attacks, Veras *et al.* [46] proposed a password cracking method based on semantics, which reduced the dependence on the training set. Han *et al.* [47] studied the difference between Chinese Pinyin and foreign English in setting passwords, optimized two password guessing methods based on probabilistic context-free grammar (PCFG) and a Markov model, and improved the hit rate to a certain extent. Ma *et al.* [48] compared and analyzed different password guessing models, noted that the Markov model is superior to the PCFG model, and proposed that the password probability threshold is an important indicator for evaluating the password set. Wang *et al.* [49] conducted a statistical analysis of many real password sets, noted that the frequency distribution of passwords conforms to a Zipf distribution, and gave the following formula: $f_r = C/r^s$, where r represents the password ranking, f_r represents the password frequency ranked r, and s and C are constants. The Zipf distribution effectively describes the relationship between password usage frequency and ranking in a password set, which provides a theoretical basis for evaluating password guessing attack models. Wang *et al.* [50] proposed a targeted password online guessing attack method. By using the user's relevant personal information, including name, birthday, mobile phone number and the user's leaked password, correlation analysis was performed to reduce the password search space.

As shown in TABLE 1, summarizes the related work of password recovery applications and their shortcomings. As seen in TABLE 1, due to high GPU power consumption and low energy efficiency ratio, as the scale continues to increase, it leads to a sharp increase in electricity and operating costs. Moreover, because the local memory on the GPU core is relatively small, the GPU is not good at hash algorithms that include S-box operations. For FPGA implementations, the hash algorithm has been optimized in different degrees, but the performance differs and remains low. Moreover, some schemes optimize and implement only a single hash module and do not make full use of chip resources to achieve parallelization of multiple hash modules. Determining how to improve the implementation efficiency of FPGAs and shorten the routing time is also a problem. For the implementation of HLS, additional PCIe channels and memory management are required. The implementation of these functions consumes considerable resources and compilation time, causing the hash algorithm to take up more resources and lose performance. Moreover, HLS programming has relatively fixed compilation and optimization strategies, and flexible development of algorithms, including serial-parallel conversion and collaboration, is difficult to achieve. Therefore, HLS programming cannot adequately support the flexible use of various applications. For password guessing attacks, the above solution increases the number of candidate passwords, occupying increasingly more space and thus resulting in low password coverage and a complex training process, and the matching speed cannot meet the needs of high-performance computing. Moreover, the efficiency of password cracking depends not only on the high-probability password itself but also on the support of high-performance algorithms.

The application range of hash algorithms is very wide, as shown in TABLE 2, including document encryption,

TABLE 1. Summary and comparison of password recovery-related technologies.

	Optimization techniques	Advantages	Shortcomings
GPU	Multicore parallel	Support for many types of algorithms, better flexibility	fetch and store limited, AES, DES and other S-box-dependent algorithm performance is low, energy efficiency is low
FPGA	Pipeline, multimodule parallel	Suitable for password recovery intensive computing, low power consumption	Fewer algorithms are supported, poor scalability, there is still space for improvement
HLS	FPGA high-level language programming	Easy to program, shorten development time	Occupies more resources, loses some performance, and is less flexible
Password guessing attack	Password heuristic attack algorithm	To some extent, the hit rate of password recovery is improved	The training process is more complex and lacks the support of high-performance algorithms

TABLE 2. Usage of hash algorithm in mainstream password recovery applications.

Algorithm	Application
SHA1	WinZip, WPA/WPA2, Oracle 11+, IPMI2 RAKP HMAC-SHA1, Blockchain My Wallet, PBKDF2-HMAC-SHA1
SHA256	RAR 5, 7-Zip, PDF 1.7 L3, Ms-AzureSync, SHA256 Crypt, PBKDF2-HMAC-SHA256, Ethereum Wallet
SHA512	Ms SQL 2012/2014, SSHA-512, SHA512 Crypt, Bitcoin/Litecoin Wallet.dat, PBKDF2-HMAC-SHA512
MD5	WordPress, PHPS, NTLMv2, Skype, MD5 Crypt, PBKDF2-HMAC-MD5

compression encryption, software encryption, database authentication, network protocol authentication, and blockchain. It can be seen in TABLE 2 that although SHA1 and MD5 have already undergone collision analysis, most software and network protocols still use these two algorithms. Therefore, it is necessary to optimize the implementation of these two algorithms. In addition, different applications have different requirements, structures, and implementations for hash computing, so a flexible architecture is needed to support them. This paper combines the idea of mimic computing and applies it to the field of password recovery through the HRCA computing model to design a high-performance reconfigurable algorithm. The design achieves multidimensional reconfigurability at the algorithm level and application level to improve computational efficiency, reduce power consumption, and improve system flexibility and scalability.

III. DESIGN OF THE RECONFIGURABLE PASSWORD RECOVERY ALGORITHM

A. OVERALL FRAMEWORK

The reconfigurable password recovery algorithm is implemented in the form of software and hardware collaborative computing between the host computer and the FPGA. The host computer is responsible for task scheduling and management, and the FPGA is mainly responsible for computing. The FPGA communicates with the host computer through PCIe and is divided into two transmission methods: BAR and DMA. The BAR channel mainly completes task configuration, status and result upload; the DMA channel mainly completes dictionary transmission. The password recovery algorithm is composed of password generation, dictionary analysis, password expansion, hash iteration and comparison verification function modules. To effectively improve the overall scalability and flexibility of the algorithm, the GALS architecture is used to form the hash computing process into a coarse-grained computing kernel and instantiate multiple algorithm submodules (ASs). Each AS is executed synchronously locally, and all ASs are executed asynchronously globally. When an AS finds the correct password, the arbitration module outputs according to the AS ID and transfers it to the host computer. The overall framework is shown in Fig.3.

In Fig.3, the hash iteration is a reconfigurable module, which can be reconfigured and loaded according to the user's configuration, and the corresponding hash computing kernel is executed in a pipelined manner. Second, according to the specific application, different tasks can be flexibly configured by the address LA and data LD fields, such as single-task parallel computing and multitask hybrid computing, which achieves better scalability. At the same time, the mask attack and dictionary attack are supported to meet the diversity of attacks. The mask attack is generated directly by the internal password generation module of each AS; the dictionary attack is distributed to each AS in turn through the PWD_FIFO after the parsed password. Finally, the asynchronous FIFO is placed among the modules in the AS, and the clock domain is divided, which can effectively improve the working

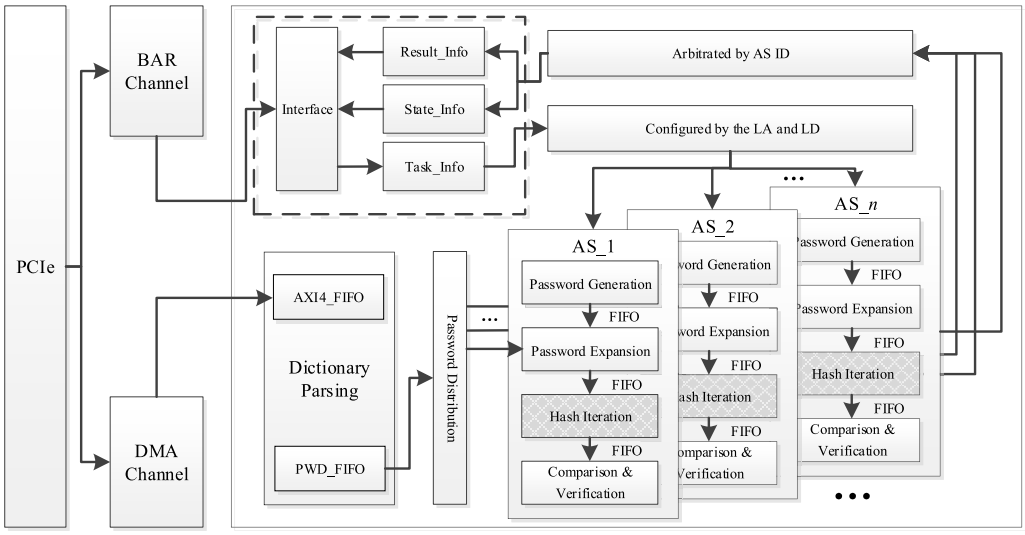


Fig. 3. The overall framework of the password recovery algorithm.

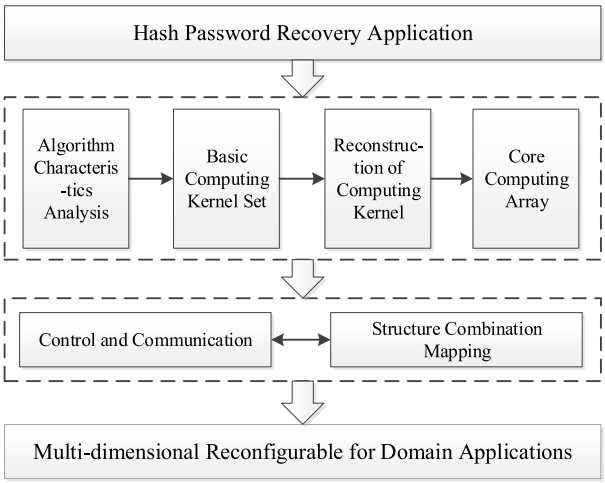


Fig. 4. Design process of password recovery algorithm based on HRCA.

frequency and resource utilization of the core hash iteration and meet the needs of high-performance computing.

B. EXTRACTION OF BASIC COMPUTING KERNEL

To realize the efficient computation of the password recovery algorithm, we first must analyze the characteristics of the algorithm, extract the basic computing kernel set, and describe the appropriate structure. The algorithm is reconstructed from the three aspects of computing, interconnection and storage to form the computing array of the core operation, which simplifies the difficulty of parallel programming. Finally, from the control and communication level, the GALS architecture is used to generate hardware structures by the combination mapping of computing kernels in order to improve the parallelism mapping of the application to a reconfigurable structure. A deeper, more extensive, domain-oriented multidimensional coarse-grained reconfigurable system is then built. The basic design process is shown in Fig.4.

1) EXTRACTION ALGORITHM

Let the target algorithm set supported by HRCA be $TAS = \{TA_1, TA_2, \ldots, TA_n\}$, where the basic block (BB) of the algorithm $TA_i (1 \le i \le n)$ divided by the control data flow graph (CDFG) is $BB_{TA_i} = \{BB_1, BB_2, \ldots, BB_m\}$. The basic operations contained in each BB are executed sequentially or concurrently, which can be executed completely independently. For example, according to the function of the hash algorithm, BB can be divided into key expansion, round iteration, nonlinear functions, etc. After all the BB_{TA_i} of TA_i are obtained, according to the code similarity or functional similarity, they are aggregated and combined to form a basic computing kernel set $BCKS = \{b_1 \times CK_1, b_2 \times CK_2, \ldots, b_k \times CK_k\}$, where $b_j (1 \le j \le k)$ is the number of kernels.

Suppose the resource occupied by CK_j is r_j; then, the total resource occupied by $BCKS$ is $R_{BCKS} = \sum_{j=1}^{k} b_j \times r_j$.

Second, because the reconstruction algorithm set TAS must consume resources, let us take public resources as R_{pub} and the private resources taken for algorithm TA_i reconstruction as $R_{pvt}(TA_i)$; then, the total reconstruction cost of TAS is $R_{TAS} = R_{BCKS} + R_{pub} + \sum_{i=1}^{n} R_{pvt}(TA_i)$.

The available resources of the FPGA chip are R, and R_{TAS} should satisfy the constraint condition C_{area}: $R_{TAS} \le R$.

Finally, the BB for the same function can be implemented by many independent FPGA methods, and its reconfigurable form is limited, so there are many combinations of $BCKS$. Then, define the delay constraint condition of the computing core as C_{time} and the power consumption as C_{power}, and adjust the $BCKS$ by the reconstruction function F. Then, evaluate the solved $BCKS$, select the most suitable $BCKS$ from them, and match the application with the optimal structure.

The specific algorithm is described in Algorithm 1.

C. RECONSTRUCTION OF CORE HASH COMPUTING

1) UNIT KERNEL STRUCTURE

After the basic computing kernel set is obtained, it must be mapped and reconstructed to form a coarse-grained unit

Algorithm 1 Computing Kernel Extraction Algorithm

Input: $TAS = \{TA_1, TA_2, \ldots, TA_n\}$

Output: $BCKS_{opt} = \{b_1 \times CK_1, b_2 \times CK_2, \ldots, b_k \times CK_k\}$

1: **for** ($i = 1$ to n) **do**
2: Divide the basic block BB_{TA_i} by CDFG for TA_i;
3: **end for**
4: Under the conditions of C_{area}, statistically merge BB_{TA_i}
5: $BCKS_{cur} = BB_{TA_1} \cap BB_{TA_2} \cap \ldots \cap BB_{TA_n}$
$= \{b_1 \times CK_1, b_2 \times CK_2, \ldots, b_k \times CK_k\}$
6: Reconstitute $BCKS_{cur}$ to generate a limited number of versions
$\{BCKS_1, BCKS_2, \ldots, BCKS_l\} = F(BCKS_{cur})$;
7: Let $BCKS_{opt} = BCKS_{cur}$;
8: **for** ($i = 1$ to l) **do**
9: Under the conditions of satisfying C_{area}, C_{time} and C_{power};
10: If $BCKS_i$ is better than $BCKS_{opt}$ in terms of resources, delay and power consumption, then $BCKS_{opt}$=$BCKS_i$;
11: **end for**

kernel, and a hybrid granularity structure is established to form an optimal structure that meets current application requirements. Because domain-oriented applications generally process relatively limited types of data, the processing flow is similar and relatively fixed, and the target structure that can be reconstructed is limited. Therefore, the reconfigurable schemes of the hash algorithm are also limited.

After analyzing the CDFG of the hash algorithm, it was found that the process of each round of iterative operations is basically the same. Therefore, each iteration of the hash algorithm can be refined into a unit kernel. For example, for the SHA1 algorithm, its iteration operation is:

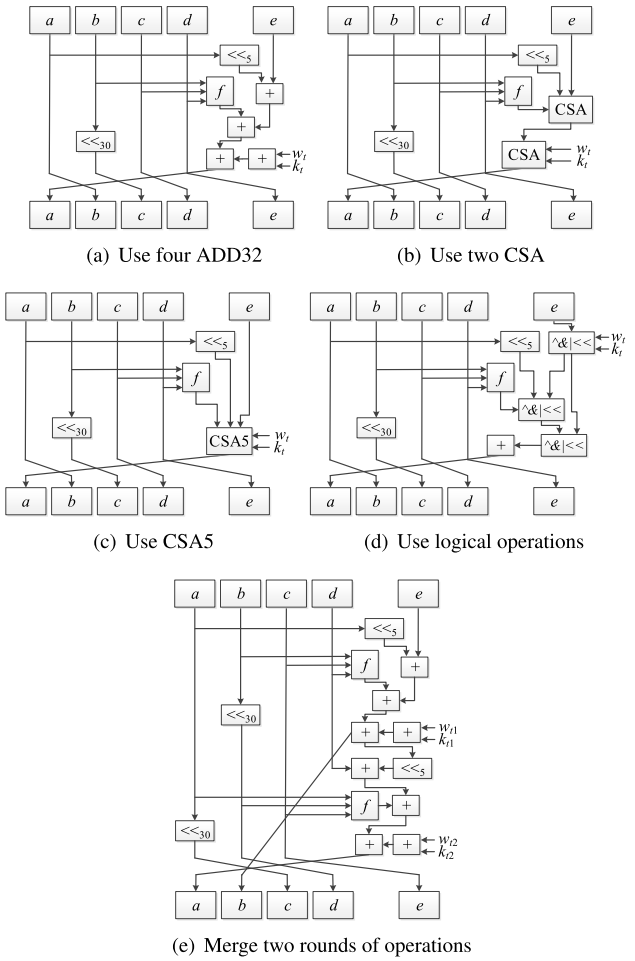
$$\begin{aligned} a &= a\_next;\ b = a;\ c = c\_next;\ d = c;\ e = d;\\ a\_next &= \{a[26:0],\ a[31:27]\} + f + e + k_t + w_t;\\ c\_next &= \{b[1:0], b[31:2]\}; \end{aligned}$$

where a, b, c, d, and e are initialization variables, f is a nonlinear function for each round of iteration, k_t is a constant for each round, w_t is a data block for each round, and $0 \le t < 80$.

Because the calculation of a_next belongs to the critical path, it can be completed directly using four ADD32 kernels; two CSA kernels; CSA5 kernels; or multiple XOR32, AND32, OR32, and S32 with one ADD32 kernel; alternatively, the merge method can be used to combine two rounds of operations into one round. Here, CSA and CSA5 are function-level arithmetic kernels composed of the basic computing kernel, and the formula is as follows:

$$\begin{aligned} &CSA(x, y, z)\\ &\quad = (x\hat{}y\hat{}z) + (((x\&y)|(x\&z)|(y\&z)) \ll 1);\\ &CSA4 = CSA((x\hat{}y\hat{}z), (((x\&y)|(x\&z)|(y\&z)) \ll 1), u);\\ &CSA5 = CSA4((x\hat{}y\hat{}z), (((x\&y)|(x\&z)|(y\&z)) \ll 1), u, v). \end{aligned}$$

For each iteration of SHA1, the unit kernel structure formed is shown in Fig.5.



(a) Use four ADD32 (b) Use two CSA

(c) Use CSA5 (d) Use logical operations

(e) Merge two rounds of operations

Fig. 5. **Unit kernel structure of each iteration of SHA1.**

In the same way, MD5 also has five unit kernel structures similar to SHA1. For SHA256 and SHA512, there are six kinds of unit kernel structures. In addition to the above five kinds of structures, the 32-bit or 64-bit CSA operations using CSA6 and CSA7 are also included.

It can be seen that there are many ways to construct the unit kernel structure of SHA1, SHA256, SHA512, MD5 and other algorithms, and each unit kernel structure has great differences in latency, resource consumption and other aspects. Therefore, according to the specific device characteristics and resource distribution, an appropriate unit kernel structure can be selected to build a high-efficiency algorithm that meets the requirements.

2) INTERCONNECT STRUCTURE

After the unit kernel is constructed, there are many paths between registers and registers, and between registers and look up tables, corresponding to different interconnect structures. The logic level of each unit kernel structure after placement and routing is also different, resulting in different delays. In this way, the interconnection depth can be balanced by logical association and movement, or the logic topology of the interconnection can be rewritten to minimize the delay.

For example, for the four ADD32 unit kernel structures of a_next, in the hardware programming environment, in addition to directly calculating

assign $a\_next = \{a[26:0], a[31:27]\} + f + e + k_t + w_t$;

the addition of 5 operands can also be divided into 3, such as

assign $temp = \{a[26:0], a[31:27]\} + f + e$;

assign $a\_next = temp + k_t + w_t$;

to reduce the logic level delay by logical association. Moreover, we can use the precomputation method to calculate

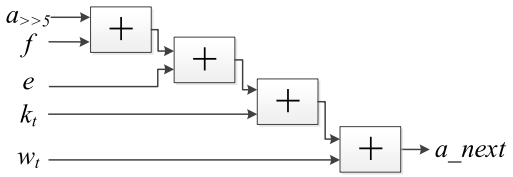
$temp <= e + k_t + w_t$;

in advance and then calculate

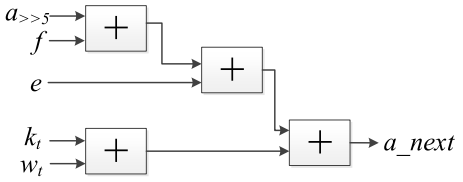
assign $a\_next = \{a[26:0], a[31:27]\} + f + temp$;

to move the operation on the critical path to the noncritical path.

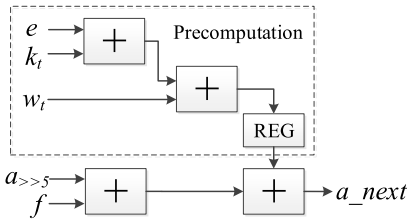
The interconnect structure of these three implementations is shown in Fig.6.



(a) Add one by one



(b) Add after split



(c) Precomputed addition

Fig. 6. **Three interconnection methods for calculating *a_next*.**

Since the delay of the cyclic shift is much less than that of the nonlinear function and the addition, the delay boundaries for the three schemes in Fig.6 are, respectively, calculated by the following formulas:

For the delay of Fig.6(a): $T_1 = t_f + 4 \times t_{ADD32}$;

For the delay of Fig.6(b): $T_2 = t_f + 3 \times t_{ADD32}$;

For the delay of Fig.6(c): $T_3 = t_f + 2 \times t_{ADD32}$;

where t_f and t_{ADD32} are the delays of the nonlinear functions f and ADD32, respectively. Obviously, inserting a register in the middle of the calculation by dividing the original logic into two parts through a precomputation method can effectively reduce the interconnection level and delay. At the same time, interconnecting two close operands, such as preferentially calculating a and f, k_t and w_t, is more conducive to placement and routing.

Second, the calculation of a_next and e_next of SHA256 and SHA512 can also change the interconnection structure by rewriting the logic to further reduce the delay. Taking SHA256 as an example, there are:

$$\begin{aligned} a\_next &= \sum\nolimits_0 (a) + \mathrm{Maj}(a,b,c)_t \\ &\quad + \sum\nolimits_1 (e) + \mathrm{Ch}(e,f,g) + h + w_t + k_t; \\ e\_next &= d + \sum\nolimits_1 (e) + \mathrm{Ch}(e,f,g) + h + w_t + k_t; \end{aligned}$$

where $\sum_0$, $\sum_1$, Maj and Ch are functions of SHA256. Since the calculation of w_t is related only to the input message and k_t is a known initialization parameter, the sum of w_t and k_t can be calculated 2 clocks in advance, as follows:

$$wk_t^{(n)} = w_{t+1}^{(n-1)} + k_t^{(n)}.$$

Then, logically rewriting a_next and e_next, there are

$$\begin{aligned} s_1 &<= wk_t^{(n-1)} + h^{(n-1)}; \\ s_2 &<= s_1 + \sum\nolimits_1 (e^{(n-1)}) + \mathrm{Ch}\left(e^{(n-1)}, f^{(n-1)}, g^{(n-1)}\right); \\ s_3 &<= \sum\nolimits_0 (a^{(n-1)}) + \mathrm{Maj}\left(a^{(n-1)}, b^{(n-1)}, c^{(n-1)}\right); \\ &\text{assign} \quad a\_next = s_2 + s_3; \\ &\text{assign} \quad e\_next = s_2 + d^{(n-1)}. \end{aligned}$$

Since $wk_t^{(n)}$ can be calculated by 2 clocks in advance and s_1 can be calculated by 1 clock in advance, s_2 and s_3 can be calculated at the current clock, and a_next and e_next are directly connected by being assigned to the output. Before and after the optimization of the interconnection structure, the a_next and e_next delays are as follows:

$$\begin{aligned} T_{a\_next} &= T_{\sum_0} + T_{\mathrm{Maj}} + T_{\sum_1} + T_{\mathrm{Ch}} + 6 \times t_{\mathrm{ADD32}}; \\ T_{e\_next} &= T_{\sum_1} + T_{\mathrm{Ch}} + 5 \times t_{\mathrm{ADD32}}; \\ T_{a\_next}{}' &= T_{e\_next}{}' = T_{\sum_1} + T_{\mathrm{Ch}} + 3 \times t_{\mathrm{ADD32}}. \end{aligned}$$

Obviously, $T_{a\_next}{}'$, $T_{e\_next}{}'$ are better than T_{a_next}, T_{e_next}.

In addition, it is possible to cascade CSA and ADD32 computing kernels or nest the use of CSA to form a hybrid interconnection for data flow and logic rewriting and sorting. For the merge operation, in addition to merging 2 rounds into 1 round, we can merge 3 rounds, 4 rounds, or 5 rounds into 1 round. Although multiple rounds of merging increases the delay and reduces the frequency, it still has a certain advantage because it occupies a small chip area. For example, for the pipeline SHA1 algorithm, merging 2 rounds into 1 round increases the delay of $t_f + 2 \times t_{ADD32}$ but reduces the register resources by approximately 20% and the power consumption by approximately 36%.

3) STORAGE STRUCTURE

For the hash algorithm, due to the need to cache the intermediate results of some operations, a variety of methods such as registers, RAM or FIFO can be used. Each storage method corresponds to a different storage structure and has an important impact on the frequency, resource consumption and throughput of the algorithm. Through the analysis of CDFG, the number of data, data bit width and data dimension required for each step of the algorithm can be obtained, and the required storage space for data can be calculated from this. Combined with the distribution and capacity of the FPGA on-chip storage resources, the appropriate storage method is then selected. Finally, we use storage strategies such as storage partition and replication to optimize the storage structure at a fine-grained level to reduce the use of logical resources and improve the working frequency.

For the serial implementation of the hash algorithm, in addition to the necessary input and output, each iteration requires multiple parameters to participate in the calculation. The specific storage requirements of SHA1 are given in TABLE 3.

TABLE 3. SHA1 serial algorithm storage requirements.

Storage Type	Storage Width (bit)	Storage Depth
Input	512	1
Initialize variables $h_0 - h_4$	32	5
Constant $k_0 - k_3$	32	4
Grouping data blocks $w_0 - w_{15}$	32	16
Intermediate variables $a - e$	32	5
Intermediate variables a_next, c_next	32	2
Output	160	1

It can be seen in TABLE 3 that the serial hash algorithm does not have high storage requirements during operation and can completely use the storage method of the register. However, the serial hash algorithm has low computational efficiency, cannot meet the computing needs, and must be accelerated in a pipelined parallel manner. The resource consumption then increases exponentially with the increasing number of pipeline stages. As shown in TABLE 4, the storage requirements of the SHA1 algorithm for the 80-stage pipeline are given.

It can be seen in TABLE 4 that if the storage is still in registers, a total of 1280 32-bit-wide registers are required for the SHA1 pipeline algorithm to store the value of the data block w_t. Moreover, the initialization variables $h_0 - h_4$ and intermediate variables $a - e$ and $a\_next - e\_next$ all need 400 32-bit-wide registers to be cached. Although there are many register resources in the FPGA, if too many local registers are consumed, there will still be logic overlap and increased delay.

TABLE 4. Storage requirements of the SHA1 pipeline algorithm.

Storage Type	Storage Width (bit)	Storage Depth
Input	512	1
Initialize variables $h_0 - h_4$	32	5×80
Constant $k_0 - k_3$	32	4
Grouping data blocks $w_0 - w_{15}$	32	16×80
Intermediate variables $a - e$	32	5×80
Intermediate variables $a\_next - e\_next$	32	5×80
Output	160	1

Therefore, through data classification and hybrid storage, the data dependency can no longer cross the boundary of the data block, and the communication between pipeline computing kernels is more balanced. According to the data characteristics of the hash algorithm, the following strategies are given to optimize the storage structure.

Strategy 1: direct assignment. Hash algorithms all contain a list of constants, and the scale is small, but the values are scattered. Using the direct assignment strategy and replacing BRAM with registers can effectively save the on-chip memory module interface.

Strategy 2: BRAM storage. For the initialization variables $h_0 - h_4$, because multiple groups of different data can be input each time, they must be stored until output. However, in the whole hash calculation process, $h_0 - h_4$ participates only in the first and last steps. Therefore, BRAM can be used for storage, and after the first step is involved in the calculation, it is sent to storage and then read out and participates in the calculation in turn in the last step.

Strategy 3: reduce data space. For the storage of block w_t, a two-dimensional register array can be used. In one-dimensional space, the value can be assigned by cyclic shift, and in two-dimensional space, the value can be copied by the register. However, in the last 15 rounds of the hash operation, the amount of w_t involved in the calculation decreases in turn. Therefore, in the last 15 rounds, the storage space of w_t can be reduced. In addition, because w_t is shifted and copied in one-dimensional space, a cyclic shift RAM can be used to replace the register in order to achieve a balance between resources and performance.

Strategy 4: data cascading. For the intermediate variables $a - e$ and $a\_next - e\_next$, the five 32-bit data can be combined into a whole 160-bit-wide data as input and output. Participating in operations by cascading data is more advantageous to restrict data to a logical area. Moreover, because $b\_next = a$; $d\_next = c$; $e\_next = d$; that is, some variables can be directly assigned to obtain the result.

Strategy 5: data reuse. For operations with precomputation, there is data overlap, and the value can be passed through the register during the calculation process to achieve data multiplexing.

For FPGA storage resources, registers are widely distributed, and BRAM is distributed in a fixed area. Combined with the above various strategies, the use of hybrid storage can make full use of FPGA resources and effectively shorten the critical path delay. Obviously, for more complex calculations of SHA256, SHA512 and HMAC, the combination of various storage strategies can further improve its performance.

4) RECONSTRUCTION METHOD

The reconstruction of the computing kernel must adopt a heuristic strategy to find the optimal solution for different hardware environments, unit kernel structures, interconnection structures and storage structures. Here, we can complete the reconstruction of the kernel by means of equivalent code transformation through a sliding window and then select the appropriate code and structure through energy efficiency assessment and comparison. The process is shown in Fig.7.

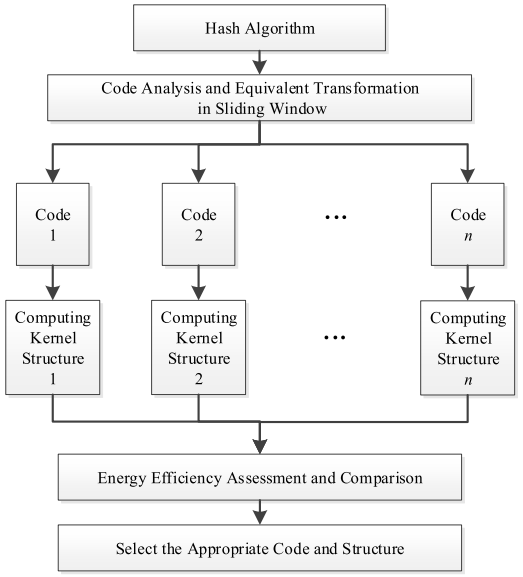


Fig. 7. Reconstruction process of hash algorithm.

Under the condition that the area constraints are met, the iterative process of the hash algorithm is divided into k consecutive basic blocks, denoted as $BLK = \{BLK_1, BLK_2, \ldots, BLK_k\}$. Each $BLK_i (1 \le i \le k)$ contains the operation that is performed sequentially or in parallel, and it is a coarse-grained block that can be performed completely independently. BLK is interconnected by the CDFG of the hash algorithm. Then, define a sliding window of size l, such that each time l iterative groups are selected to continuously replace $BLK_j (1 \le l, j \le k)$ for functional equivalent replacement, which generates BLK'_j. The area, delay, power consumption, or energy efficiency ratio of BLK'_j may be better than those of BLK_j. Finally, the simulated annealing algorithm is used to iteratively optimize BLK, bring the newly generated BLK'_j back to BLK, update it to form BLK', calculate the cost of BLK' in all aspects, and compare with the cost of the original BLK. If the cost of BLK' is less than the cost of the original BLK, accept the new reconstruction scheme. Otherwise, accept the new solution with probability $\exp(-\Delta d/t)$, where Δd is the cost difference in area, delay, power consumption, or energy efficiency ratio, and t is the current temperature value. Then, slide the window forward and perform the next code equivalent transformation and iterative optimization. The specific reconstruction algorithm is as follows.

Algorithm 2 Reconfigurable Algorithm of Computing Kernel

Input: $BLK = \{BLK_1, BLK_2, \ldots, BLK_k\}$
Output: $BLK'_{opt} = \{BLK'_1, BLK'_2, \ldots, BLK'_k\}$

```
t = Initial(t); l = Initial(l);
while t > t_min do
    BLK_sub = {BLK_i1, BLK_i2, ..., BLK_il} = ContSel(BLK);
    for (j = 1 to l) do
        BLK'_ij = FunRepl(BLK_ij);
    end for
    BLK'_sub = {BLK'_i1, BLK'_i2, ..., BLK'_il};
    BLK' = Update(BLK, BLK'_sub);
    Δd = Comparison(BLK', BLK);
    if ((Δd < 0) or (Random(0,1) < exp(−Δd/t))) then
        BLK = BLK';
    end if
    t = t × Δt;
    Continue to slide the window forward; if the current window is completely transformed, adjust the window size and start a new round of code equivalent transformation;
end while
```

In Algorithm 2, t_{min} is the termination of low temperature; Δt is the cooling coefficient; ContSel represents the continuous selection of l basic blocks from BLK; FunRepl indicates that the computing kernel function is replaced equivalently; Update means to update the interconnect and storage structure; and comparison means the difference in area, delay, power consumption, or energy efficiency ratio.

D. PIPELINE TECHNOLOGY

HRCA implements applications efficiently in a pipelined and parallel manner. Pipeline technology is a kind of time-parallel technology. It divides a repetitive process into several subprocesses, and each subprocess runs in parallel with others. For the password recovery algorithm, in order to find the correct password, a large number of passwords must be attempted. The pipeline structure can not only improve the utilization of hardware resources but also improve the execution speed with a high acceleration ratio. Here, according to the number of iterations of the hash algorithm, all loops are unfolded to form a full pipeline structure. When it is working at full load, the overall pipeline can calculate a set of hash values at every clock. The full pipeline architecture is shown in Fig.8.

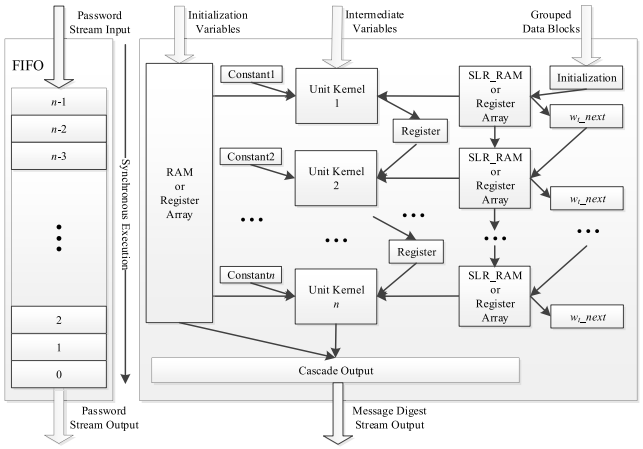


Fig. 8. Full pipeline architecture of hash algorithm.

In Fig.8, the password stream uses FIFO for buffering, the initialization variables are cached using RAM or the register array, the constants are directly assigned, the intermediate variables are passed in registers after unit kernel calculation, and the grouped data blocks are transferred by RAM or the register array after initialization. Second, the password stream is executed in parallel with the hash operation to form a synchronous pipeline of linear operations. Third, the unit kernel is reconstructed with the basic computing kernel, and the data width is changed; any one of the algorithms of SHA1, SHA256, SHA512 and MD5 can be implemented, which have good scalability. Finally, for different FPGA chips, through the reconstruction of the unit kernel, a full pipeline hash algorithm with different performance, power consumption and area can also be formed to match the application with a suitable structure.

E. PASSWORD ATTACK ALGORITHM

1) MASK RULE ALGORITHM

The Zipf distribution of passwords provides a theoretical basis for password recovery attacks to reduce the password space and improve the hit rate. The traditional password exhaustion algorithm tries all possible combinations of passwords in turn, and the cracking time depends on the length of the password and the character set used. This method can theoretically crack all passwords, but due to the limitations of the device performance and time, brute force attacks are often unsuccessful. Here, the password structure partition method of probabilistic context-free grammar is adopted, and D_n, U_n, L_n and S_n are designated as digits, uppercase letters, lowercase letters, and special characters with length n, respectively. These special symbols represent the password structure; for example, the structure of "Pass@123" is expressed as $U_1L_3S_1D_3$. Further, the corresponding character sets of D_n, U_n, L_n and S_n are specified, and the characters in each character set are arranged in descending order of probability. The mask rules are formed by various combinations of D_n, U_n, L_n and S_n, thus improving the efficiency of password guessing attacks. The specific process is shown in Algorithm 3.

Algorithm 3 Mask Rule Password Generation Algorithm

Input: Password character set; *pwd_mask*; *pwd_length*;
Output: Candidate password

```
Configure the character set for each byte of the password
for (i = 0 to (pwd_length − 1)) do
    password[i] = Null;
    index[i] = 0;
end for
while (True) do
    for (i = 0 to (pwd_length − 1)) do
        if (pwd_mask[i] ==‘D’) then
            password[i] =DigitCharSet(index[i]);
        else if (pwd_mask[i] ==‘U’) then
            password[i] =UpperCharSet(index[i]);
        else if (pwd_mask[i] ==‘L’) then
            password[i] =LowerCharSet(index[i]);
        else if (pwd_mask[i] ==‘S’) then
            password[i] =SpecialCharSet(index[i]);
        end if
        Determine whether index[i] needs to carry or return to 0;
    end for
    Generate password;
    If index[0 . . . pwd_length − 1] is all 0, jump out of the while loop;
end while
```

The password generation algorithm judges whether each byte of the mask rule is a digit, uppercase letter, lowercase letter, or special character; finds the characters in the corresponding character set according to the index; and then generates a password by combining characters. Furthermore, the algorithm also judges whether the index carry is out of bounds or returns to zero, ensuring that the character currently indexed is always in the corresponding D_n, U_n, L_n and S_n character set. This guarantees that all generated passwords satisfying the given mask rule. In addition, to meet the needs of high-performance computing, the mask rule algorithm must generate candidate passwords within one clock and pass it to the hash algorithm to complete the operation. For this purpose, a dual-port RAM is allocated, and the corresponding character set is stored in the RAM, which can be indexed in one clock according to the address. Moreover, because each character of the password mask is independent, a unit kernel structure can be formed, as shown in Fig.9.

In Fig.9, after the RAM is configured, the address and counter are initialized by the password mask character, and the address is updated by the index carry. When the counter accumulates to a certain degree, the next index carry flag is generated. At the same time, multiple unit kernels can be executed in parallel in the form of a pipeline. The carry is passed between kernels through the index and generates a password of length n within one clock. The corresponding overall structure is shown in Fig.10.

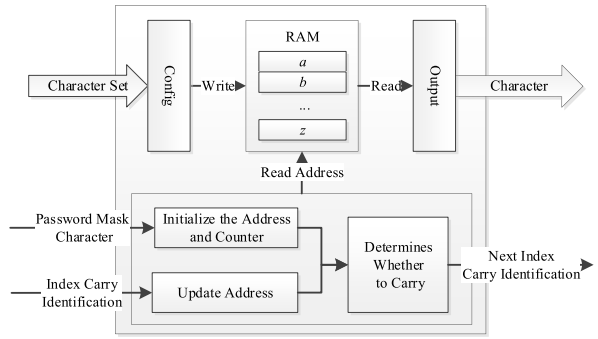


Fig. 9. Unit kernel structure for password generation.

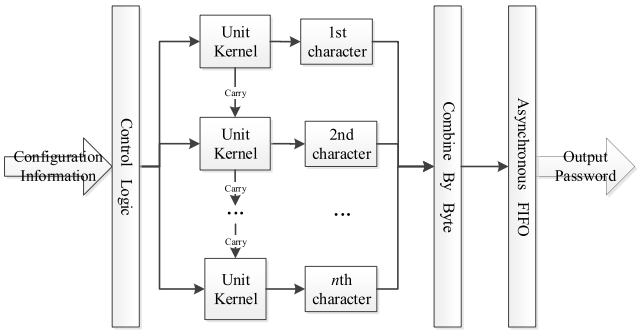


Fig. 10. Password generation pipeline structure.

2) HIGH-SPEED DICTIONARY PARSING

To meet the demand of dictionary-mode high-performance computing, the dictionary is transmitted by DMA. First, the data are transferred to the AXI4 FIFO in burst read-write mode via PCIe. Then, the data are read from the AXI4 FIFO, the position of all passwords is determined, and they are marked according to the newline character. Finally, the data are truncated according to the mark flag and assembled into a password of length n, and there is arbitration control to write the password into the asynchronous FIFO. Since only one group of data is read from the AXI4 FIFO each time, there may be a case where the end data do not satisfy a password. At this time, it is necessary to splice the current valid data with the next group of data to make up a complete password and parse it again. The dictionary parsing structure based on the computing kernel is shown in Fig.11.

Since the AXI4 FIFO bit width is 64 bytes, it is assumed that the maximum password length is 20 bytes. According to this, define a 1-bit-wide identification *dic_flag*[63:0] and a processing identification *dic_proc_flag*[84:0], define a 1-byte-wide cache data *dic_data*[63:0] and a processing data *dic_proc_data*[84:0], and then define a byte counter *byte_cnt*. The specific steps of dictionary parsing are as follows:

1) Read the data from the AXI4 FIFO and assign it to *dic_data*[63:0]. Determine whether the $i(0 \le i < 64)$ character is a new line character within 1 clock. If so, *dic_flag*[i]=0, and otherwise, *dic_flag*[i]=1;
2) Assign *dic_data* and *dic_flag* to *dic_proc_data* and *dic_proc_flag*, and initialize the counter *byte_cnt*=64;

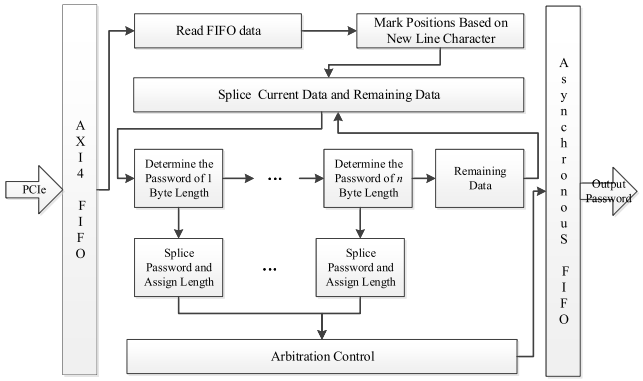


Fig. 11. Dictionary parallel parsing structure.

3) For a password of length n, if the first n bits of *dic_proc_flag* are all 1—that is, *dic_proc_flag*[$n-1:0$]==n'b1, and *dic_proc_flag*[n]==0—then use *dic_proc_data*[$n:0$] to splice the password of current length n;
4) *dic_proc_flag* and *dic_proc_data* are shifted to the right by n, $byte\_cnt = byte\_cnt - n$;
5) If *byte_cnt* is less than 22, read the data from the AXI4 FIFO again, assign *dic_data* and *dic_flag*, and jump to 6); otherwise, jump to 3) to continue execution;
6) Splice *dic_data* and *dic_flag* behind the remaining valid data of *dic_proc_data* and *dic_proc_flag*, and there is $byte\_cnt = byte\_cnt + 64$, then jump to 3) to continue execution.

For steps 3) and 4), the judgment of the password with length of 1-20 bytes can be completed within 1 clock and output after splicing. For steps 5) and 6), if the number of remaining bytes is less than 22, the data must be read again. Obviously, the remaining 22 bytes contain a complete password so that it can be parsed while reading the data. In this way, when the frequency is 200 MHz, 200 M passwords can be parsed every second, which greatly improves the efficiency of dictionary attacks.

F. MAPPING AND PLACEMENT OF COMPUTING KERNEL

1) ADVANCED COMPUTING KERNEL LIBRARY

According to the granularity of the partition, the computing kernel can complete the reconstruction from fine-grained to coarse-grained. The fine-grained reconstruction embodies the flexibility, and the coarse-grained reconstruction embodies the characteristics of high-performance computing. To speed up hardware design and implementation, the computing kernel is divided into the algorithm, control and communication to form an advanced password recovery computing kernel library. The hardware structure is then generated in the form of the combined mapping, multiplexing pipeline and serial-parallel hybrid computing for various algorithms, as shown in Fig.12.

The algorithm computing kernel includes the core hash algorithm, mask rule algorithm, dictionary high-speed

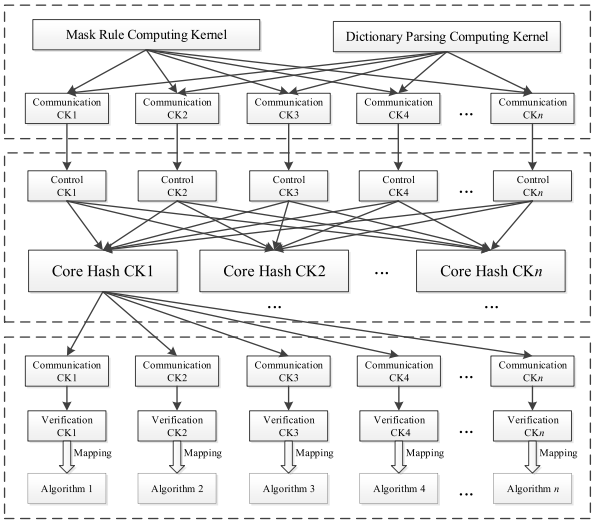


Fig. 12. Schematic diagram of computing kernel structure mapping.

parsing and verification module. They are mainly implemented in the unit kernel reconfigurable form, which is executed in the pipelined parallel manner through fine-grained optimization and has high performance.

The control computing kernel includes loop control, branch jump, and input/output selection and can be realized by the execution sequence and jump result of the CDFG dynamic detection algorithm. Their control logic is realized by the parameters of loop start point, loop count, loop boundary and judgment conditions.

The communication computing kernel is mainly used to cache data and connect between modules to achieve IO balance. Through the CDFG analysis of the data dependencies and the working frequency of the serial-parallel modules in the algorithm, the goal is to maximize data throughput and realize customized data paths.

The algorithm computing kernel has been described in detail in the previous section. The following will be optimized from the control and communication plane to improve the overall computational performance while satisfying the flexibility of the password recovery application.

2) CONTROL COMPUTING KERNEL STRUCTURE

There are many password recovery algorithms, and each algorithm calls the hash function in different loop iterations, ranging from one to hundreds of thousands. Second, the input of each iteration is not the same: some are fixed character splicing, some are different in the order of password and salt splicing, and some need circular splicing. Finally, the structure of the called hash function is different: some need to call the hash function only once per iteration, some need to call multiple times, and some have a mixed call of multiple hash functions. Therefore, it is necessary to control and optimize the core hash calculation of the pipeline; otherwise, it is unable to give full play to its performance.

Through the analysis of various types of password recovery algorithms, three control optimization structures of fixed splicing, circular splicing and multihash combination are given, and the control computing kernel is formed. It can complete the control of cycle, selection and jump through parameter configuration.

(1) Fixed splicing control. The input of this kind of hash pipeline must be spliced at a fixed position according to the password and the salt length, and then the output of this time is spliced again at a fixed position and used as the input for the next time. Such algorithms include Oracle 11+, PDF 1.7 L3, SSHA-512, and WordPress.

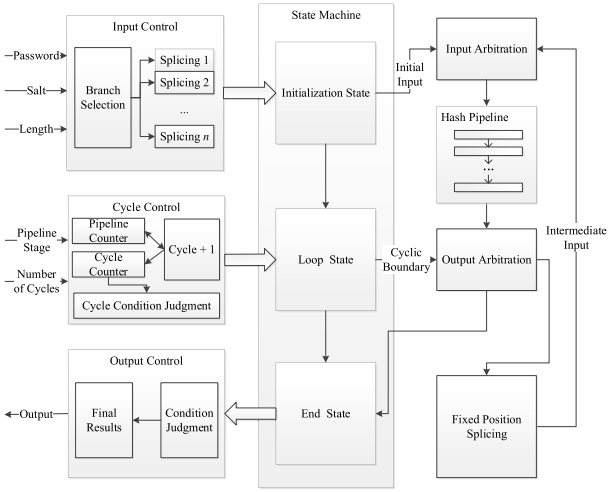


Fig. 13. Pipeline fixed position splicing control.

As shown in Fig.13, the entire control flow is divided into three states: initialization, loop and end. Each state has its own control operation. In particular, the input control is spliced by the branch selection according to the password or the salt length; the cycle control counts according to the pipeline stage and the number of cycles and judges whether to jump out of the loop; the output control judges to output the final result according to the condition. The corresponding pipeline control is as follows: ① after the initialization is completed, the initial input and the intermediate input are arbitrated by the input arbitration and input to the pipeline for calculation; ② the output arbitration judges whether to jump to the end state according to the cyclic boundary and sends the result to the next cycle for fixed splicing or output.

(2) Circular splicing control. The input of this type of hash pipeline requires the circular splicing of the password, salt and other parameters. When the splicing length reaches a hash input group, it can be calculated. Therefore, before each hash iteration, the input must be pieced together in advance; otherwise, the pipeline will be broken. Its control flow is shown in Fig.14. Such algorithms include RAR 3.x-4.x, 7-Zip, SHA512 Crypt, and MD5 Crypt.

Because the hash pipeline is batch input, it is necessary to splice the salt and other parameters with all passwords in turn; other parameters include byte length, last loop result,

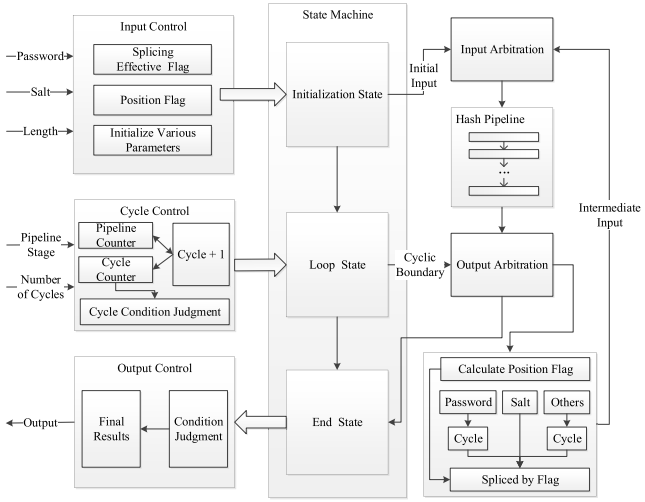


Fig. 14. Pipeline circular splicing control.

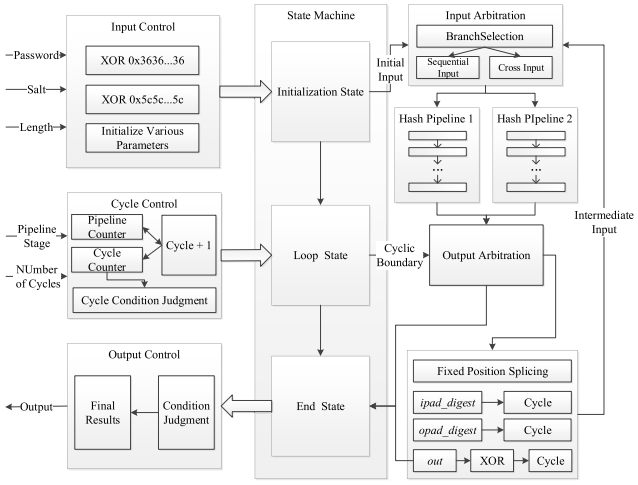


Fig. 15. Multihash pipeline combination splicing control.

and 0x80. Here, the splicing effective flag and the position flag bit array are added to the input control to precalculate the password, salt and other parameters where the next splicing is and mark it. After the output arbitration, the batch of passwords, salts and other parameters are spliced according to the position flag and input into the hash pipeline. The main difference between circular splicing and fixed splicing is the splicing processing of characters, and other operations are basically similar.

(3) Multihash combination control. For the HMAC algorithm, many hash calculations must be called during one operation. The formula is

$$\mathrm{HMAC}(P, M) = \mathrm{Hash}(P \oplus opad || \mathrm{Hash}(P \oplus ipad || M)),$$

where P is the password, M is the input message, and *ipad* and *opad* are external padding, whose initial values are 0x3636 …36 and 0x5c5c …5c, respectively. The HMAC algorithm must XOR the filled P with *ipad* and *opad*, and then two hash calculations are performed to generate *ipad_digest* and *opad_digest*, respectively. Next, *ipad_digest* and *opad_digest* are used as the initial hash values to perform hash operations twice again. In this loop iteration, the control flow is shown in Fig.15. These algorithms include HMAC-SHA1, RAR 5, HMAC-SHA512, and HMAC-MD5.

Here, to optimize the HMAC algorithm, two hash pipelines are used for combined processing. In the initial state, the batch passwords are XOR *ipad* and *opad* and are sequentially input into hash pipelines 1 and 2, respectively. In the loop state, hash pipelines 1 and 2 alternately use the result of the other side to splice at a fixed position to form the current message input. In the end state, the results of *out* XOR on hash pipeline 2 are output in turn.

By combining hash algorithms, not only is the control complexity reduced but also the routing is more conducive to improve the performance. In addition to the HMAC algorithm, a similar structure can be adopted for the algorithms that nest using hash operations including PHPS, sha($salt.sha1($pass)), and sha1(md5($pass)).

3) COMMUNICATION COMPUTING KERNEL OPTIMIZATION

The password recovery algorithm consists of multiple modules, some of which are calculated serially and some of which are calculated in parallel. To improve data throughput and avoid insufficient or excessive computation, I/O balance between modules is critical. The communication computing kernel mainly forms the buffer area through RAM, FIFO or registers and performs an input/output connection on each serial-parallel module.

Suppose that serial module X and parallel module Y communicate with each other, and X transfers data to Y. The clock frequency of X is $freq_x$, the calculation period is cyc_x, and one output is generated at a time. The clock frequency of Y is $freq_y$, the period of one iteration is cyc_y, the number of iterations is $iter_y$, and n groups of data must be input for one calculation. To give full play to the parallel performance of Y, the following condition must be satisfied:

$$(freq_x/cyc_x) \geq (freq_y/(cyc_y \times iter_y)) \times n.$$

Moreover, the general pipeline frequency is higher than the serial frequency, such that

$$freq_x \leq freq_y.$$

The longer the iteration period of Y, the greater the number of iterations and the lower the output requirements of X. When Y is fully pipelined and iterates only once, X must also be pipelined or executed in parallel. Similarly, when the parallel module Y transfers data to the serial module X, the processing of X must also satisfy the above conditions.

For the communication optimization between the serial and parallel modules, we can use CDFG by the greedy strategy to calculate the frequency satisfied by other modules forward and backward on the premise of ensuring the core

hash performance. At the same time, for the communication between serial and parallel modules in different clock domains, asynchronous RAM or FIFO is inserted to complete the data synchronization. Finally, the clock frequency of some modules is unified to shorten the routing time and improve the placement quality, as shown in Algorithm 4.

Algorithm 4 Communication Computing Kernel Optimization

Input: $F = \{f_1, f_2, \ldots, f_n\}$

Output: Frequency of each functional module

1: Establish $CDFG = (V, E)$ from $F = \{f_1, f_2, \ldots, f_n\}$;
2: Initialize the $freq_{fix}$ of the on-chip component frequency, and initialize the $freq_{low}$ of the module frequency with lower data throughput;
3: Determine the frequency $freq_{core}$ of the node v_h where the core hash operation module is located;
4: **for** (From node v_h to breadth-first search for nodes traversing each layer in the *CDFG* downward) **do**
5: **if** (The current node frequency is not determined) **then**
6: **if** (There is a serial-parallel conversion between the current node and the parent node) **then**
7: Calculate the frequency that this node meets, and insert asynchronous RAM or FIFO between the modules;
8: **else**
9: The node is serial or parallel with the parent node, and its frequency is the same as the parent node;
10: **end if**
11: **else if** (The current node has a frequency that is different from the frequency of the parent node) **then**
12: Insert asynchronous RAM or FIFO between modules;
13: **end if**
14: **end for**
15: Similarly, from node v_h to breadth-first search to traverse the nodes of each layer in the *CDFG* up to the root node, calculate the node frequency of each layer;
16: Traverse the nodes of each layer from the root node, and adjust the frequency of some nodes appropriately to unify and reduce the number of clock frequencies to improve the routing quality.

4) OVERALL PLACEMENT STRATEGY

As the FPGA scale increases, the reconfigurable resources become more intensive, which results in more time-consuming routing of complex algorithms. To effectively reduce the routing complexity and achieve the flexibility of hash iteration, the FPGA is divided into multiple regions, and there are $FSP_{region} = \{fsp_1, fsp_2, \ldots, fsp_n\}$. For the target hash function set TAS_{hash}=(SHA1, SHA256, SHA512, MD5, ...), for any $fsp_i (1 \leq i \leq n)$, there is Implement$(fsp_i) = \forall hash \in TAS_{hash}$; that is, each region can serve as the placement with the same or different password recovery algorithms.

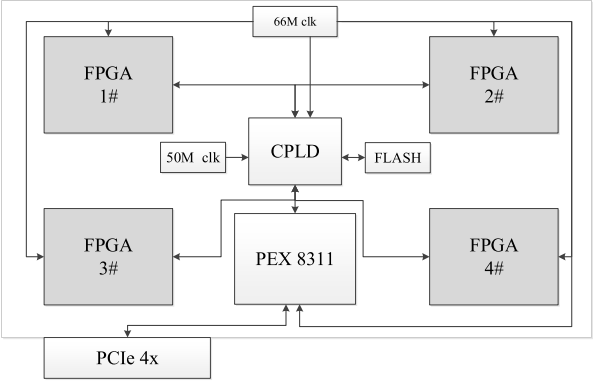


Fig. 16. High-performance reconfigurable board.

To accomplish this, we first establish a two-dimensional FPGA region model (H, W), where H and W are the number of CLBs contained in a single column and row of the FPGA, respectively; that is, H represents the height, and W represents the width. Second, using a region-based division strategy, multiple ASs are placed and executed in the GLAS mode so that they can work in parallel. The resource consumption of the initial placement of a single AS can be used to calculate the number of submodules that can be placed. The FSP_{region} is then divided according to the number of submodules. Third, according to the number of CLB overlaps and conflict areas that appear when arranging multiple submodules, the analysis is performed from multiple perspectives of logic, storage and interconnection. The compilation strategy is then changed accordingly to perform multiple placements and summarize the results of each placement. Finally, we choose one of the better placement schemes, adopt the greedy strategy to prioritize the computing kernel with the spatial division method, optimize the conflict areas locally, constantly adjust the CLB conflict areas, and find the global optimal solution. The details are shown in Algorithm 5.

IV. EXPERIMENTAL RESULTS AND ANALYSIS

The experimental hardware platform in this paper is a high-performance reconfigurable board integrated by 4 large-scale FPGAs. The FPGA chip type is the xcku060-ffva 1156-2-i. Its on-chip resources include 726 K logic units, 331,680 LUTs, 663,360 REGs, 1080 BRAMs and 9180 Kb DisRAMs. Its structure is shown in Fig.16. The programming software is Xilinx Vivado 2019.2.

In Fig.16, four large-scale FPGAs are mainly responsible for computing. CPLD communicates with FPGAs and PEX 8311 through address and data buses and can dynamically load FPGA configuration files from FLASH. PEX 8311 integrates a PCIe communication module and communicates with the host computer through the PCIe bus.

The experiments are mainly based on the password recovery algorithms. First, the simulated annealing algorithm is

Algorithm 5 Computing Kernel Placement Strategy

Input: Initial routing of a single AS
Output: Overall placement scheme

1: The initial routing of a single AS determines the occupancy of resources R_{AS}, such as LUT, registers and BRAM;
2: From the overall FPGA resource R, calculate the number of ASs that can be placed $n = R/R_{AS}$;
3: Perform an initial placement of n AS, and change the strategy to choose the best result;
4: If the current strategy meets the timing requirements, then finish;
5: **if** (The current solution does not meet the timing requirements, but there are still FPGA resources remaining) **then**
6: Split the AS into core, high-frequency, low-frequency and communication computing kernels;
7: According to n, divide the FPGA resources from top to bottom and left to right in a rectangular way from the space, which is recorded as $FSP_{region} = \{fsp_1, fsp_1, \ldots, fsp_n\}$
8: **for** ($i = 0$ to n) **do**
9: Adopt the minimum partition method, and preferentially place the i-th core computing kernel in fsp_i;
10: Then, place the i-th high-frequency and low-frequency computing kernels into fsp_i;
11: Finally, place the i-th communication computing kernel into fsp_i;
12: **end for**
13: From a global perspective, use the breadth-first search algorithm to work with overlapping regions;
14: If there are still remaining areas, move the overlapping logic to the nearest area first;
15: If there is no redundant area, move and convert adjacent *fsp* logic blocks to ensure that all *fsp* logic density is balanced;
16: **end if**
17: **if** (The current solution does not meet the timing requirements and exceeds the overall FPGA resource R) **then**
18: Number of ASs $n = n - 1$, and jump 3 to continue execution;
19: **end if**

used to complete the reconstruction of different basic computing kernels. The different reconstruction schemes are compared and analyzed from the aspects of performance, resources, and power consumption to illustrate the high scalability of HRCA. Second, the serial-parallel structure, routing time, energy efficiency ratio, and performance of other schemes are compared to illustrate the efficiency of HRCA. Third, the implementations of different password recovery algorithms are compared and analyzed, indicating the practicality of the HRCA design. Finally, from the three dimensions of hash function-level, application-level and board-level reconfiguration, the cracking efficiency of password recovery is analyzed to illustrate the flexibility of the HRCA architecture.

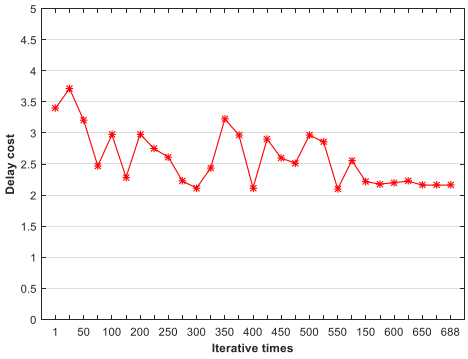


Fig. 17. SHA1 reconstruction delay cost curve.

A. RECONSTRUCTION SCHEME ANALYSIS

1) SHA1 RECONSTRUCTION ANALYSIS

Taking the SHA1 algorithm as an example, the overall structure can be divided into 4 main modules: input preprocessing module, w_t storage module, 4 groups of 20 rounds for a total of 80 rounds of core operation modules, and output modules. It is implemented in a full pipeline architecture, mainly for w_t storage and 80 rounds of core operation modules, and adopts the simulated annealing algorithm for iterative optimization. Let the scale of reconstruction be 80 rounds, l=20, t=1000, t_{min}=1, Δt=0.99, and use ADD32, AND32, OR32, NOT32, XOR32, CSA and other computing kernels to reconstruct $a\_next - e\_next$. For the reconstruction of w_t storage, REGs, BRAMs, and DisRAMs are used. The computing kernel used is formally described. Its resource and delay costs are shown in TABLE 5, where the resource cost is calculated as LUTs+REGs/2.

After 688 iterations, the algorithm is terminated, and the reconstruction delay cost of SHA1 is shown in Fig.17. It can be seen in Fig.17 that the reconstruction delay cost of the SHA1 computing kernel is between 2.11 and 3.71.

As shown in TABLE 6, the typical SHA1 reconstruction scheme is realized emphatically.

It can be seen in TABLE 6 that the SHA1 algorithm formed by the computing kernel reconstruction method has a frequency above 225 MHz, a delay of 2.08 - 4.44 ns, and a maximum power consumption of only 4.709 W. At the same time, it can be seen that the coarse-grained algorithms constructed by different reconstruction schemes are not the same regarding delay, structure and resources. In particular, the scheme of using the register storage and CSA is the best in terms of performance and resources, and merging two rounds into one round takes up the fewest resources.

2) COMPARISON OF SERIAL AND PARALLEL STRUCTURES

If the hash algorithm is implemented in a serial structure, each password must wait for the last password to be processed.

TABLE 5. Computing kernel reconstruction resources and delay cost of SHA1.

Computing Kernel	Description	Resource Cost	Delay Cost
ADD32	32-bit addition	80	1.388
AND32	32-bit and	79	0.62
OR32	32-bit or	45.5	1.449
NOT32	32-bit not	63	0.778
XOR32	32-bit xor	79	0.62
S32	32-bit shift	24	0.798
LF32	32-bit nonlinear function	96	0.689
SC512	512-bit string concatenation	256	0.696
CSA	32-bit carry-save adders	126	1.766
CSA5	CSA operation of 5 operands	252	1.9
REG	Register store w_t	108.5	1.874
BRAM	BRAM store w_t	40	1.858
DisRAM	DisRAM store w_t	137	1.764

TABLE 6. Comparison of different SHA1 computing kernel reconstruction schemes.

Scheme	Composition	LUTs	REGs	Pipeline Stage	Highest Frequency (MHz)	Power Consumption (W)
SHA1_1	Register, ADD32	12604	22243	82	325	3.049
SHA1_2	Register, 2 CSA	12358	23860	83	480	3.994
SHA1_3	Register, CSA5	16515	23700	82	365	3.712
SHA1_4	Register, AND32, OR32, XOR32, S32 and ADD32	16228	22359	82	425	4.079
SHA1_5	2 rounds merged into 1 round	11029	18816	42	250	2.142
SHA1_6	RAM, CSA	16591	13214	82	225	3.133
SHA1_7	DisRAM, CSA	23757	13214	82	260	3.036

If a multiserial structure is adopted in parallel, it is assumed that the parallel number is p and the number of input passwords is n. If $p \geq n$, there is no need to wait for consecutively entered passwords, and it can directly enter the hash operation; if $p < n$, the first p passwords can enter the hash operation in sequence. For the $p + 1$ password, it still must wait. Although the computational efficiency of this parallel structure can be increased by p times, it is obvious that n is much larger than p, and the efficiency of the entire hash operation process is still low. In addition, although the serial structure takes up fewer resources, due to the complicated logic operations of each round and the long hash operation process, most logic operations are tightly coupled, which is not conducive to routing. For multimodule parallelism, this leads to FPGA node overlap.

In the pipeline mode, the stage is n, and any number of passwords can be processed continuously. When the first password is calculated on the pipeline, the second password can directly enter the pipeline without waiting for the first password to be completed. Its computational efficiency is n times that of serial structure. As shown in TABLE 7, the serial and pipeline structure of each hash algorithm are compared. Among them, the computational efficiency increase times=pipeline frequency × stage/serial frequency; the resource utilization improvement factor = serial resource × stage/pipeline resource.

According to the comparison in TABLE 7, the pipeline structure has higher execution efficiency and better resource usage than the serial structure. Not only can a group of hash values be calculated in each clock, greatly improving the data processing speed, but also the whole hash algorithm can be ensured to work at a higher frequency.

3) COMPARISON OF SYNTHESIS AND ROUTING TIME

Using the HRCA design method simplifies the mapping of code to the hardware structure and reduces the complexity of module parallel design. Moreover, the GALS architecture is adopted, which effectively shortens the development time. In the case of multiple modules in parallel, taking 200 MHz as the performance index and balancing resources as the optimization goal, the traditional design method and the HRCA reconstruction design method are used for analysis and comparison, as shown in TABLE 8.

It can be seen in TABLE 8 that the HRCA design method not only shortens the synthesis and routing time but also meets the timing constraints and has higher performance.

TABLE 7. Comparison of serial and pipeline structures.

Algorithm	Serial Structure			Pipeline Structure				Increase of Computational Efficiency	Increase in Resource Utilization
	Frequency(MHz)	LUTs	REGs	Frequency (MHz)	Stage	LUTs	REGs		
SHA1	260 MHz	933	922	480	83	12358	23860	153.23	4.76
SHA256	205	1395	1119	287	67	21556	24547	93.80	3.87
SHA512	165	2717	2179	200	84	57604	71006	101.82	3.43
MD5	210	757	444	315	66	8684	8301	99.00	5.03

TABLE 8. Comparison of synthesis and routing time between traditional and HRCA design methods.

			Traditional Design			HRCA Design		
	Number of Modules	Frequency	Synthesis Time	Routing Time	Meet Timing Constraints	Synthesis Time	Routing Time	Meet Timing Constraints
SHA1	16	250 MHz	0:13:49	7:11:47	No	0:18:20	5:18:03	Yes
SHA256	11	200 MHz	0:13:33	2:26:52	Yes	0:09:33	2:13:46	Yes
SHA512	3	200 MHz	0:11:47	5:27:42	No	0:10:11	4:10:33	Yes
MD5	32	200 MHz	0:15:17	4:31:20	Yes	0:11:46	1:51:12	Yes

TABLE 9. Comparison of peak password speeds generated by the CPU and FPGA.

Character Set	CPU (Mp/s)	GPU (Mp/s)	FPGA (Mp/s)
digits + letters	349.5	8211.9	20000
digits + special characters	350.3	8433.3	20000
letters + special characters	355.5	9039.5	20000

The traditional method has also optimized the algorithm design, including the pipeline, register cache, and CSA. However, the same method is adopted for all algorithms, and the specific FPGA structure and application are not analyzed in depth in combination with the characteristics of different algorithms.

B. PERFORMANCE COMPARISON AND ANALYSIS

1) MASK PASSWORD GENERATION SPEED

In the case of a given password strategy, mask rules are used to exhaust the password. A character set for each byte of the password is allocated to generate various password combinations, including digits + letters, digits + special characters, letters + special characters, etc. In the case where the password length is 20, the peak speed of the password generated by the CPU, GPU, and FPGA is compared, as shown in TABLE 9. The CPU type is Intel i5-7500, and the main frequency is 3.40 GHz; the GPU type is GeForce GTX 1080.

It can be seen in TABLE 9 that the FPGA can generate a password more than 56.26 and 2.21 times faster than the CPU and GPU, respectively, which has obvious advantages. The reason is that the FPGA uses pipeline technology, which can read the characters in the corresponding position within one clock and complete the splicing to generate the password. Second, the FPGA single-password generation module takes up only 470 LUTs and 270 REGs, with a maximum frequency of 375 MHz. Finally, when $n(n \leq 100)$ modules are parallel, the speed is increased by dozens of times. Therefore, the FPGA is more suitable for password recovery of the hash algorithm than the CPU.

2) DICTIONARY EFFICIENCY ANALYSIS

In dictionary mode, the supply of passwords becomes a bottleneck for algorithms with faster speed. Here, we take the PDF 1.7 L3 algorithm as an example to compare the impact of the CPU, GPU and FPGA on the speed under different dictionary sizes, as shown in TABLE 10.

It can be seen in TABLE 10 that the speed of the FPGA dictionary password is much higher than that of the CPU and GPU, and the highest speed is 24.33 and 5.58 times, respectively. This is mainly because FPGA transfers the dictionary through DMA, can read data while parsing, and can verify the password at the same time. However, the CPU and GPU are limited by storage and require frequent interaction with memory, which affects the speed of dictionary parsing. In addition, the larger the dictionary, the greater the impact on the GPU, and its speed continuously decreases. The FPGA is not affected by the size of the dictionary and is basically maintained at a stable speed.

3) RECOVERY EFFICIENCY ANALYSIS

Assuming that a password pwd with a length of n corresponds to 95 ASCII characters in the set, the password

TABLE 10. The speed of each computing component under different dictionary sizes.

Dictionary File	Size	Number of Passwords	CPU (p/s)	GPU (p/s)	FPGA (p/s)
Dict_1	67.9 MB	11881376	5051600	47628800	118813780
Dict_2	162 MB	14365003	4920400	33835700	119708358
Dict_3	551 MB	44044861	5066900	26222200	108752743
Dict_4	953 MB	100000000	5279900	21839100	121951219
Dict_5	1.44 GB	119040000	5173200	20773200	111224402

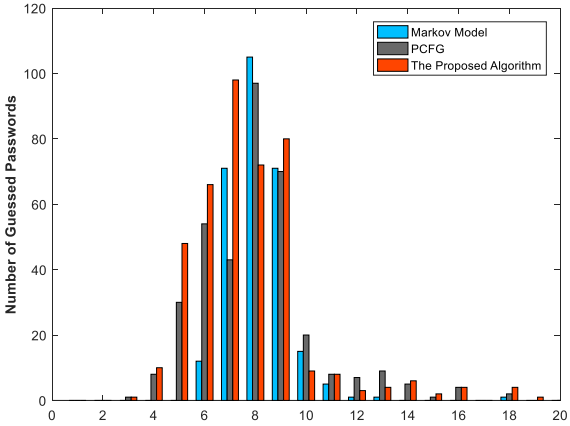


Fig. 18. Guessing numbers of three password guessing algorithms.

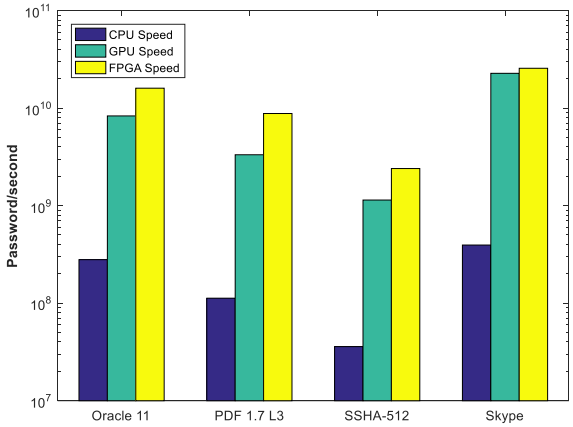


Fig. 19. Comparison of the CPU, GPU and FPGA algorithm speed.

space is 95^n. Obviously when n is large, the password search space becomes huge. However, due to the Zipf distribution of password frequencies, most users choose a password length of 8-14, and the corresponding password space can be reduced to 95^8-95^{14}. Moreover, according to the rules of user password statistics, the password space can be further reduced, and the recovery rate can be improved. At present, most users use a combination of letters + digits + special characters to set passwords; that is, $U_{n1}L_{n2}D_{n3}S_{n4}$, and $n1 + n2 + n3 + n4 = n$. At this time, the corresponding password space is $26^{n1} \times 26^{n2} \times 10^{n3} \times 33^{n4}$, which is obviously smaller than 95^n. For example, suppose that the user password structure is a $U_1L_5D_6S_1$ 13-digit password, and one reconfigurable board can exhaust all passwords in 7.38 days.

In practical application, 5000 SSHA-512 tasks are obtained, and 10^9 passwords are generated by Markov model, PCFG, and the proposed mask rule algorithm. The cracking results are shown in Fig.18.

It can be seen from Fig.18 that the three algorithms have their own advantages. When the password length is 8, the number of guessed passwords by Markov model is higher than the other two methods. When the password length is 3-7 and 9-19, PCFG and our algorithm are higher than Markov model, and our algorithm is higher than PCFG in most cases. In the end, our algorithm guessed a total of 416 passwords, Markov model guessed 282 passwords, and PCFG guessed 359 passwords. The password recovery accuracy of our algorithm is higher than the other two methods. This is because our algorithm and PCFG perform better when generating medium-scale passwords, while Markov model needs more training data to generate larger-scale passwords and obtain better guessability. Then in the case of massive tasks, using the proposed mask rule algorithm can give priority to recover more passwords in a short time.

In addition, due to a user's password reuse behavior, that is, using the same password to log in to multiple accounts, when faced with different decryption tasks, the same mask rule file can be used to crack different tasks to recover most passwords. Obviously, a multialgorithm and task parallel architecture is more suitable for applications in the field of hash password recovery.

4) ENERGY EFFICIENCY RATIO ANALYSIS

To verify the effectiveness of the HRCA design method, the algorithm speed and energy efficiency ratio of the CPU, GPU and FPGA are compared here. The software used is Hashcat v3.60. The comparison algorithms are Oracle 11 (SHA1), PDF 1.7 L3 (SHA256), SSHA-512 (SHA512) and Skype (MD5). The energy efficiency ratio (EER) is calculated as follows: EER = performance/power consumption. The comparison results are shown in Fig.19 and Fig.20.

As seen in Fig.19 and Fig.20, the password recovery algorithms designed based on HRCA have not only a high speed but also a high energy efficiency ratio. In particular, the

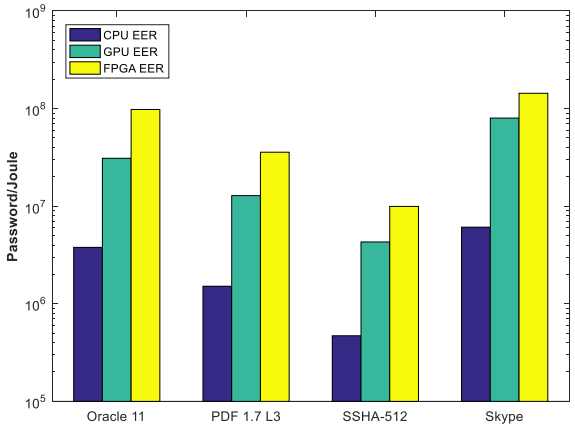


Fig. 20. Comparison of the CPU, GPU and FPGA energy efficiency ratio.

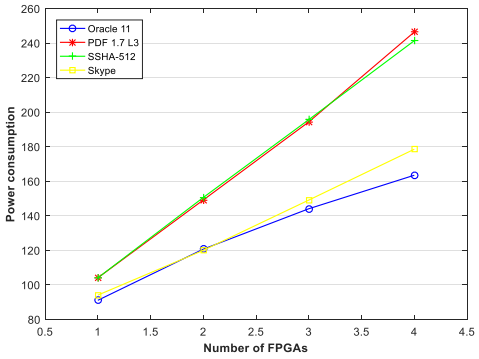


Fig. 21. Power consumption of each algorithm changes with increasing FPGA number.

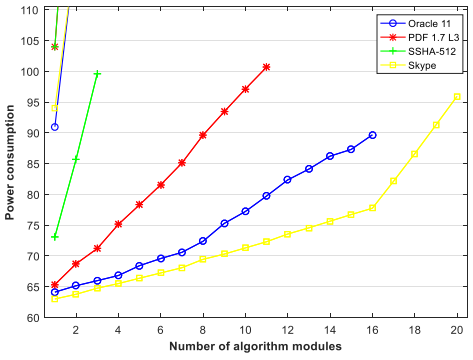


Fig. 22. Power consumption of each algorithm changes with increasing number of modules in FPGA.

highest speed is 78.22 times and 2.65 times that of the CPU and GPU, respectively, and the highest energy efficiency ratio is 25.88 times and 3.16 times that of the CPU and GPU, respectively.

Further, the power consumption curves of each algorithm with the increase in the number of FPGAs and modules are given, as shown in Fig.21 and Fig.22.

As seen in Fig.21 and Fig.22, the overall power consumption of the FPGA is low, and according to the computing needs, the FPGA is dynamically called, and the number of FPGAs or modules is appropriately increased or decreased to obtain a better energy efficiency ratio. For example, if the computational requirements are not high, only one FPGA can be used, and the other FPGAs are in the unloaded state, or only some modules of one FPGA are used to further reduce power consumption.

5) COMPARISON WITH OTHER SCHEMES

The performance and throughput of the SHA1, SHA256, SHA512 and MD5 algorithms of other FPGA schemes and ours are compared below, as shown in TABLE 11. The calculation formula of the throughput is as follows:

$$T = B \times f \times m/C$$

where T is the throughput, B is the size of the data block, f is the clock frequency, m is the number of modules, and C is the clock cycle required to process the data block.

It can be seen in TABLE 11 that the frequency and throughput of the hash algorithm implemented in this paper are higher than those of most other schemes. This is due to the use of the HRCA design idea, the realization of the hash algorithm and password generation algorithm with a full pipeline architecture, and the placement of multiple modules in parallel, greatly improving the computational performance.

The performance of the SHA1, SHA256, SHA512 and MD5 algorithms of other GPU schemes and the performance this scheme are compared, as shown in TABLE 12.

It can be seen in TABLE 12 that the computational speed of the hash algorithm implemented in this paper is much higher than that of other GPUs. This result is mainly due to the fixed hardware structure of the GPU, which is not suitable for all algorithms, especially for the complex calculation of SHA256 and SHA512 with a large number of parameters. The high-performance reconfigurable board in this study contains 4 large-scale FPGAs with variable structures, which is are suitable for scientific computing in the field.

C. APPLICATION COMPARATIVE ANALYSIS

1) HMAC PERFORMANCE ANALYSIS

The HMAC algorithm includes three basic types: the HMAC(key=$pass), HMAC(key=$salt) and PBKDF2-HMAC algorithms. Among them, the passwords and salts involved in HMAC(key=$pass) and HMAC(key=$salt) calculations are in different orders, and only one calculation is performed. PBKDF2-HMAC first performs the HMAC(key=$pass) operation and then repeats the loop calculation according to the number of iterations. Obviously, the three calculation modes of HMAC are very similar, which is very suitable for the shared computing kernel. Here, through the state machine judgment, the three algorithms are fused into one. In addition, MD5, SHA1, SHA256 and SHA512 are further optimized to improve the computational performance. As shown in TABLE 13, the implementations of different HMAC algorithms and computing kernel structures are given.

TABLE 11. Comparison with other FPGA schemes.

Algorithm	Other Schemes				This Scheme	
	References	Hardware	Frequency (MHz)	Throughput (Mbps)	Frequency (MHz)	Throughput (Mbps)
SHA1	Kim et al. [28]	Virtex-6 LX240T	150.7	7351	250	8192000
	Suhaili et al. [30]	Arria II GX: EP2AGX45DF29C4	274.2	1754		
	Michail et al. [37]	Virtex-7	139.6	14300		
	Michail et al. [38]	TSMC 90 nm ASIC	589	15078		
SHA256	Mohamed et al. [32]	Virtex-5	170	1359	200	4505600
	Mestiri et al. [35]	Virtex-5	202	1580		
	Rote et al. [36]	Virtex-6	271	2040		
	Michail et al. [37]	Virtex-7	159	20100		
	Michail et al. [38]	TSMC 90 nm ASIC	521	16672		
SHA512	Maashri et al. [33]	Cyclone IV	50	1444	200	2457600
	Mestiri et al. [35]	Virtex-5	176	2200		
	Rote et al. [36]	Virtex-6	192.57	2347		
	Michail et al. [37]	Virtex-7	159	16300		
MD5	Suhaili et al. [29]	Arria II GX: EP2AGX45DF29C4	290.53	74375	200	13107200
	Jin Z et al. [43]	Arria 10 GX1150	201	813056		

TABLE 12. Comparison with other GPU schemes.

Algorithm	Other Schemes			This Scheme
	References	Computing Part	Speed (p/s)	Speed (p/s)
SHA1	Qiu et al. [18]	AMD HD 7970	2615 M	16000 M
	Barbieri et al. [19]	NVIDIA Geforce GT 540M, GTX 660, GTX 550Ti, 8600M GT, 8800 GTS	950.1 M	
	Huo et al. [34]	ASIP (1000 MHz)	24.42 M	
SHA256	Huo et al. [34]	ASIP (1000 MHz)	30.26 M	8800 M
SHA512	Aggarwal et al. [21]	GRID K520 × 4	1120 M	2400 M
	Ge et al. [22]	AMD R9 290 × 2	1055 M	
	Huo et al. [34]	ASIP (1000 MHz)	11.92 M	
MD5	Qiu et al. [18]	AMD HD 7970	6877 M	25600 M
	Barbieri et al. [19]	NVIDIA Geforce GT 540M, GTX 660, GTX 550Ti, 8600M GT, 8800 GTS	3258.4 M	
	Huo et al. [34]	ASIP (1000 MHz)	30.86 M	

It can be seen in TABLE 13 that after the fusion of the three HMAC algorithms, a high computing speed can still be maintained. In the HMAC single-iteration process, except for HMAC-SHA512, the calculation of *ipad* and *opad* of other algorithms can be calculated in parallel by two hash modules, and then the two hash modules are connected in series to calculate the salt, which speeds up the calculation process. Similarly, for the PBKDF2-HMAC calculation with 1000 iterations, alternately using two hash modules for parallel calculation not only improves the resource utilization rate but also shortens the calculation cycle.

TABLE 13. Computing kernel structure of HMAC algorithms.

Fusion Algorithm	Computing Structure	Control Structure	Storage Structure	Number of Modules	Frequency (MHz)	Power Consumption (W)	Speed of Single Iteration (p/s)	Speed of 1000 Iterations (p/s)
HMAC-SHA1	2 SHA1	multihash combination	167 sets of register	7	200	18.365	399 M	1397890
HMAC-SHA256	2 SHA256	multihash combination	135 sets of register	5	200	24.292	285 M	998491
HMAC-SHA512	1 SHA512	fixed splicing	84 sets of registers and FIFO	3	200	23.496	149 M	299550
HMAC-MD5	2 MD5	multihash combination	133 sets of register	14	200	19.081	797 M	2795774

TABLE 14. FPGA reconstruction explanations of different applications.

Typical Applications	Explanation
WinZip, WPA/WPA2	Because the WinZip, WPA2 calculation process must use SHA1 circularly and alternately, resulting in increased peripheral control and storage logic, the use of ADD32 and multihash combined control can effectively reduce resource consumption
RAR 5, 7-Zip PDF 1.7 L8	Xilinx FPGA registers are rich in resources, and the use of registers to cache intermediate results is conducive to placement and routing and improves the clock frequency
Office 2013, SSHA-512	Since SHA512 has a calculated bit width of 64, if it is implemented using CSA, it occupies many logic resources, the timing drops sharply, and the routing fails
WordPress, PHPS, Skype	Since the calculation of b_next contains a string left shift operation, using logical operations can appropriately merge some resources to reduce area occupation

TABLE 15. Implementation of Xilinx FPGA in different applications.

	Number of Modules	Frequency (MHz)	LUTs	REGs	RAMs
WPA/WPA2	6 / 6	200 / 250	209300 / 205201	428551 / 389665	216.5 / 229
RAR 5	5 / 5	200 / 200	280116 / 266164	307977 / 289290	56 / 31
Office 2013	3 / 3	150 / 200	251185 / 246955	341272 / 342505	39 / 39
WordPress	25 / 30	200 / 200	268448 / 271529	381321 / 325176	176 / 211

2) COMPARISON OF PASSWORD RECOVERY APPLICATIONS

To verify the computing kernel of FPGA implementation under different password recovery applications, different reconstruction schemes are used, as shown in TABLE 14.

Here, we selected four applications, WPA/WPA2, RAR 5, Office 2013 and WordPress, and the comparisons before and after optimization using computing kernels are given. The results are shown in TABLE 15.

It can be seen in TABLE 15 that the comparative applications that are not implemented using the HRCA architecture have lower performance. The reason is that the traditional design method does not make full use of chip resources, and the control logic implemented by the architecture is slightly complicated, which is not suitable for hardware placement and routing, resulting in poor computational performance.

A comparison with the password recovery algorithm of other schemes is shown in TABLE 16.

It can be seen in TABLE 16 that the computing speed of the password recovery algorithm designed by HRCA is much higher than those of other schemes.

D. MULTIDIMENSIONAL RECONFIGURABLE ANALYSIS

Here, the reconstruction scheme of the hash algorithm is divided into function-level, application-level and board-level reconstruction, and a multidimensional reconfiguration is formed to apply to different applications.

1) FUNCTION-LEVEL RECONSTRUCTION

In this scheme, 10 passwords are randomly selected from the dictionary library; 10 sets of SHA1, SHA256 and

TABLE 16. Comparison with the password recovery algorithm of other schemes.

Other Schemes				This Scheme
References	Algorithm	Hardware	Speed (p/s)	Speed (p/s)
Liu et al. [13]	RAR5	XC7Z030-3 × 16	5711	109820
	WPA/WPA2		57780	714624
	Office 2007		19635	277788
	Office 2010		9837	138600
	Office 2013		979	23952
Tyler et al. [39]	WPA/WPA2	Virtex-5 LX330 FPGA	8931	714624
Kammerstetter et al. [40]		Kintex-7 XC7K410T-3	210783	
Ding et al. [41]	RAR 3	XC7Z030-3	6340	82640
Bai et al. [42]	WinZip	Zynq-7020 × 48	50000	4290088

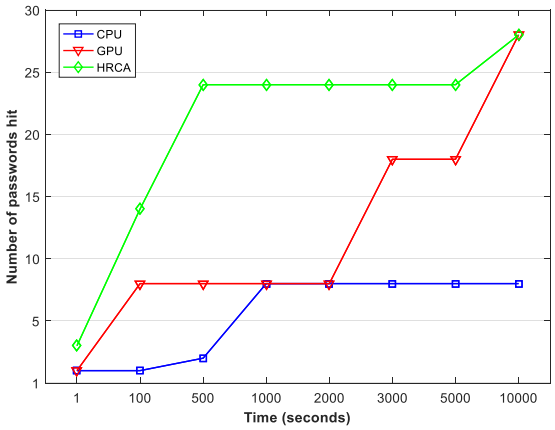
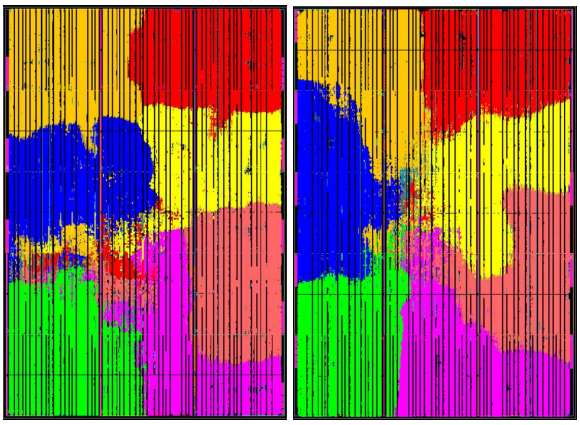


Fig. 23. Comparison of CPU, GPU and FPGA password recovery efficiency.



(a) Office 2007 (b) Office 2010

Fig. 24. FPGA placement of Office 2007 / 2010.

SHA512 verification digests are accordingly generated; and the CPU, GPU and FPGA are used for testing. In the high-performance reconfigurable board, one chip is configured as SHA1; one chip is configured as SHA256; two chips are configured as SHA512; and the CPU and GPU sequentially crack the tasks of SHA1, SHA256 and SHA512. The test results are shown in Fig.23.

As seen in Fig.23, the password hit rate of the GPU and FPGA is much higher than that of CPU in the same time. Although both the GPU and FPGA hit 28 passwords in 10,000 s, in the 100 - 5000 s interval, the number of FPGA hit passwords is higher than that of the GPU, which is more time sensitive. This result is due to the use of FPGA reconfigurable features, which can support multiple algorithms or multiple tasks in parallel. Because the hardware circuit has spatial parallelism, each module has its own circuit, and the circuits do not affect each other. Although the CPU has multiple cores and supports multithreading, it is limited by the architecture, and the computational efficiency of each core is low. If each core executes an algorithm and runs independently, the expected computing speed cannot be achieved. The GPU has many cores, but it is difficult to control. During operation, it can implement only one algorithm and support only one task, which is more suitable for parallel processing of data. Further, from the Zipf distribution of the password, it is clear that the multialgorithm and task-parallel architecture is more suitable for the application of hash password recovery.

2) APPLICATION-LEVEL RECONSTRUCTION

Here, application-level reconstruction of Office 2007 and Office 2010 was performed. Both applications used the SHA1_4 scheme in TABLE 6 to compute the kernel reconstruction. The number of parallel modules was 7, and the frequency was 250 MHz. The FPGA placement after reconstruction is shown in Fig.24.

As seen in Fig.24, the placement of Office 2007 and Office 2010 using the HRCA design method and GALS architecture is very similar. This is because we divided the applications reasonably in the computing kernel approach; reconstructed only the control logic of the algorithm, the communication module and the verification comparison module; and realized

TABLE 17. Algorithm implementation of other boards.

Algorithm	Intel Cyclone V	Xilinx xcku060	Huawei FX600	Xilinx U280
SHA1	8/135	16/250	32/250	48/250
SHA256	6/150	11/200	22/200	38/200
SHA512	2/100	3/200	8/200	12/200
MD5	20/150	32/200	60/200	98/200

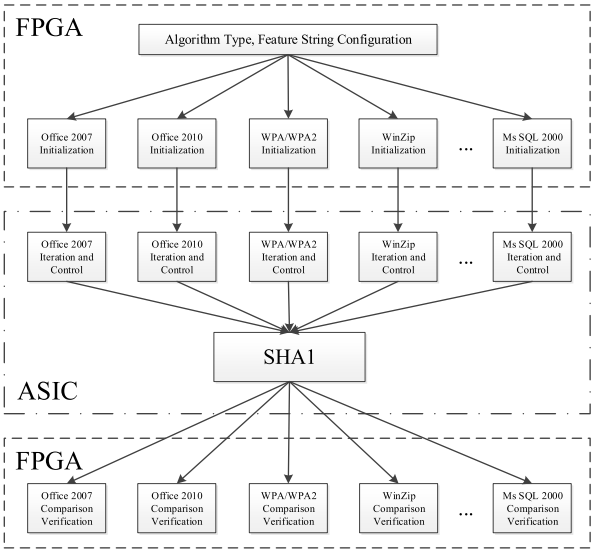


Fig. 25. HRCA-based FPGA + ASIC password recovery algorithm architecture.

two different applications at a lower cost. This approach simplifies the development of hardware and highlights the effectiveness of the HRCA design method.

Similarly, algorithms with the same core hash algorithms, such as WinZip 128 and WinZip 256, RAR5 and PBKDF2-HMAC-SHA256, Ms SQL 2012/2014 and SSHA-512, WordPress, PHPS, and Skype, can be reconstructed in the same way.

3) BOARD-LEVEL RECONSTRUCTION

In addition, in order to verify the effectiveness of the HRCA design, the hash algorithm was transplanted to other boards such as Intel Cyclone V, Huawei FX600 and Xilinx U280. The specific results are shown in TABLE 17 (the data format in TABLE 17 is module number/frequency MHz). Among them, Intel Cyclone V is a low-end board; Xilinx xcku060 is a mid-end board; and Huawei FX600 and Xilinx U280 are high-end boards.

It can be seen in TABLE 17 that the hash algorithms of all boards work at a higher frequency, and 6-98 modules can be placed for parallel calculation. The resource utilization rate of each board reaches more than 70%, among which the resource utilization rate of Cyclone V MD5 is 89.69%; xcku060 MD5 is 88.20%; FX600 SHA1 is 70.46%; and U280 SHA256 is 80.88%. The algorithm designed by HRCA has good portability and can give full play to the computational performance of different FPGA boards.

V. APPLICABILITY DISCUSSION

This paper mainly studies the hash password recovery algorithm, which is based on logical operation, bit operation and addition. In addition, its structure is compact, so it is very suitable for the implementation of HRCA. Second, based on the HRCA configuration of different computing kernels, hash algorithms with different models, structures and types can be realized in very flexible applications. Again, for symmetric cryptographic algorithms such as AES, SM4, Twofish, and Serpent, each round can be unfolded to analyze the key expansion, S-box and data replacement, etc. Using logical resources, RAMs or LUTs to form a unit kernel in parallel, the approach can form a variety of pipeline algorithms with different stages. For asymmetric encryption algorithms including RSA, ECC, and SM2, analysis of modular multiplication, modular inversion, double point, point addition, etc., can be used to form a unit kernel with large number of operations, and different bit-width algorithms can be implemented with control logic. Finally, to further improve the computational performance, it can also be combined with HRCA design ideas to implement heterogeneous computing of FPGAs + ASICs with application-level reconstruction, as shown in Fig.25. By making full use of the reconfigurability of FPGAs and the ultrahigh computational power of ASICs, each module of the algorithm can be reasonably divided, the initialization and comparative verification of various password recovery algorithms on FPGAs can be realized, and the core iteration of various algorithms can be implemented on ASICs to fully save ASIC resources and improve its computing flexibility.

Cryptographic algorithms are the cornerstone of information security, and design methods based on HRCA can also be used in image and data encryption. For image encryption, including image parallel encryption [51], a chaotic system can be used to generate random keys [52] and chaotic image encryption methods [53], [54] to improve the security and resistance of the algorithm. In order to improve the speed of image encryption, a variety of schemes such as the chaotic communication system [55], chaotic pseudo random number generator [56], chaos blend DNA coding [57], and chaotic map image encryption [58]–[61] have been implemented on FPGAs. Based on this approach, the HRCA method can be

used to design and implement various reconfigurable algorithms in the field of image encryption to form a computing kernel library. In the hardware mode, different computing kernel combinations are called each time to generate diversified and variable encryption schemes to improve the security of image encryption. Moreover, the process of image encryption is accelerated by hardware to realize a more efficient and flexible algorithm. For data encryption, the HRCA high-performance reconfigurable board can be used as the computing component, and cryptographic algorithms can be used to coordinately accelerate user authentication, data encryption transmission and storage. Moreover, embedded platforms can be built to protect data security and user privacy with cryptographic algorithms to promote the secure development of the Internet.

Although the HRCA design method has many advantages, there remain many problems that have not been solved. First, HRCA is more suitable for computationally intensive applications, mainly to achieve high-performance scientific computing. Communication-intensive application focus more on data communication and transmission, and the I/O bandwidth between modules is the primary consideration. Second, the extraction and analysis of computing kernels and the reconstruction and reorganization of unit kernels require a deep grasp and understanding of domain applications and mastery of certain hardware technologies. This paper mainly validates and implements the HRCA design method on an FPGA. In the domain-oriented application, if the reconfigurable chip is specially customized, the computing performance can be further improved.

VI. CONCLUSION

A reconfigurable password recovery algorithm based on HRCA is proposed in this paper. First, through in-depth analysis of the characteristics of the hash algorithm, the basic computing kernel set is extracted, and the combination design is carried out with the unit kernel structure, interconnection structure and storage structure. Second, the reconstruction of the computing kernel is completed by using the simulated annealing algorithm and functional equivalent replacement. The full pipeline design architecture of the hash algorithm is given, and the mask rule password algorithm and dictionary high-speed parsing are used to improve the overall speed of the password recovery algorithm. Third, the password recovery advanced computing kernel library is presented. With the algorithm and a control and communication computing kernel that is multidimensional reconfigurable, an algorithm structure suitable for the current application is mapped and generated. Finally, the FPGA is divided into regional resources to optimize placement and routing in depth, further improving resource utilization and clock frequency. The experimental results show that this method can reconstruct the computing kernel scheme with advantages in performance, resources, power consumption, etc., and can reconstruct a suitable structure for different FPGAs or applications, reducing the threshold for parallel hardware design and shortening the development cycle. Moreover, the designed system has multidimensional reconfigurability, and the variable structure significantly improves the recovery efficiency. The password recovery algorithm based on HRCA has a high hardware utilization rate, considering the efficiency and flexibility of the computing.

However, due to the numerous password recovery algorithms, this paper implements only part of the encryption recovery algorithm. The next step is to complete the implementation of other password recovery algorithms in order to increase the types of algorithms supported by the HRCA. The heterogeneous computing collaboration of FPGA + ASIC should be further studied to establish an ASIC hardware computing kernel library and realize a higher-performance and scalable password recovery algorithm. In addition, how to effectively apply HRCA to other fields, such as image encryption and big data security, has yet to be further studied and explored.

REFERENCES

[1] N. Paladi, C. Gehrmann, and A. Michalas, "Providing user security guarantees in public infrastructure clouds," *IEEE Trans. Cloud Comput.*, vol. 5, no. 3, pp. 405–419, Jul. 2017.

[2] S. A. Kumar, T. Vealey, and H. Srivastava, "Security in Internet of Things: Challenges, solutions and future directions," in *Proc. 49th Hawaii Int. Conf. Syst. Sci. (HICSS)*, Jan. 2016, pp. 5772–5781.

[3] S. A. Chaudhry, M. S. Farash, H. Naqvi, and M. Sher, "A secure and efficient authenticated encryption for electronic payment systems using elliptic curve cryptography," *Electron. Commerce Res.*, vol. 16, no. 1, pp. 113–139, Mar. 2016.

[4] N. Katende, C. Wilson, and A. Kibe, "Enhancing trust in cloud computing using md5 hashing algorithm and rsa encryption standard," *Int. J. Sci. Eng. Res.*, vol. 8, no. 3, pp. 550–566, 2017.

[5] V. Nicolls, N.-A. Le-Khac, L. Chen, and M. Scanlon, "IPv6 security and forensics," in *Proc. 6th Int. Conf. Innov. Comput. Technol. (INTECH)*, Aug. 2016, pp. 743–748.

[6] G. Hatzivasilis, I. Papaefstathiou, and C. Manifavas, "Password hashing competition—Survey and benchmark," *IACR Cryptol. ePrint Arch.*, vol. 2015, pp. 1–30, Mar. 2015.

[7] A.-D. Vu, J.-I. Han, H.-A. Nguyen, Y.-M. Kim, and E.-J. Im, "A homogeneous parallel brute force cracking algorithm on the GPU," in *Proc. ICTC*, Sep. 2011, pp. 561–564.

[8] K.-C. Lu, A. F. M. Huang, A. Y. S. Su, T.-J. Ding, and C.-N. Su, "Information password recovery with GPU," in *Proc. Int. Carnahan Conf. Secur. Technol. (ICCST)*, Sep. 2015, pp. 1–5.

[9] R. Hranický, M. Holkovič, and P. Matoušek, "On efficiency of distributed password recovery," *J. Digit. Forensics, Secur. Law*, vol. 11, no. 2, pp. 79–96, 2016.

[10] A. Abbas, R. Voss, L. Wienbrandt, and M. Schimmler, "An efficient implementation of PBKDF2 with RIPEMD-160 on multiple FPGAs," in *Proc. 20th IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2014, pp. 454–461.

[11] F. Wiemer and R. Zimmermann, "High-speed implementation of bcrypt password search using special-purpose hardware," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Dec. 2014, pp. 1–6.

[12] X. Li, C. Cao, P. Li, S. Shen, Y. Chen, and L. Li, "Energy-efficient hardware implementation of LUKS PBKDF2 with AES on FPGA," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 402–409.

[13] P. Liu, S. Li, and Q. Ding, "An energy-efficient accelerator based on hybrid CPU-FPGA devices for password recovery," *IEEE Trans. Comput.*, vol. 68, no. 2, pp. 170–181, Feb. 2019.

[14] J. Wu, "Meaning and vision of mimic computing and mimic security defense," (in Chinese), *Telecommun. Sci.*, vol. 30, no. 7, pp. 1–7, 2014.

[15] B. Li, Q. Zhou, and X. Si, "Mimic computing for password recovery," *Future Gener. Comput. Syst.*, vol. 84, pp. 58–77, Jul. 2018.

[16] Z. Fan and L. Xing-Guo, "Design and verification of reconfigurable Web cloud access device based on smart hybrid storage," in *Proc. Int. Conf. Comput. Inf. Sci.*, Jun. 2013, pp. 1483–1488.
[17] J. Fu, S. Qiao, Y. Huang, X. Si, B. Li, and C. Yuan, "A study on the optimization of blockchain hashing algorithm based on PRCA," *Secur. Commun. Netw.*, vol. 2020, pp. 1–12, Sep. 2020.
[18] W. Qiu, Z. Gong, Y. Guo, B. Liu, X. Tang, and Y. Yuan, "GPU-based high performance password recovery technique for hash functions," *J. Inf. Sci. Eng.*, vol. 32, pp. 97–112, Jan. 2016.
[19] D. Barbieri, V. Cardellini, and S. Filippone, "Exhaustive key search on clusters of GPUs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, May 2014, pp. 1160–1168.
[20] R. Chen, Y. Zhang, J. Zhang, and J. Xu, "Design and optimizations of the MD5 crypt cracking algorithm based on CUDA," in *Proc. Int. Conf. Cloud Comput.*, 2014, pp. 155–164.
[21] A. Aggarwal, P. Chaphekar, and R. Mandrekar, "Cryptanalysis of bcrypt and SHA-512 using distributed processing over the cloud," *Int. J. Comput. Appl.*, vol. 128, no. 16, pp. 13–16, Oct. 2015.
[22] C. Ge, L. Xu, W. Qiu, Z. Huang, J. Guo, G. Liu, and Z. Gong, "Optimized password recovery for SHA-512 on GPUs," in *Proc. 7 IEEE Int. Conf. Comput. Sci. Eng. (CSE) IEEE Int. Conf. Embedded Ubiquitous Comput. (EUC)*, Jul. 2017, pp. 226–229.
[23] M. Dürmuth and T. Kranz, "On password guessing with GPUs and FPGAs," in *Proc. Int. Conf. Passwords*, vol. 9393, Dec. 2014, pp. 19–38.
[24] J. Anish Dev, "Bitcoin mining acceleration and performance quantification," in *Proc. IEEE 27th Can. Conf. Electr. Comput. Eng. (CCECE)*, May 2014, pp. 1–6.
[25] J. Anish Dev, "Usage of botnets for high speed MD5 hash cracking," in *Proc. 3rd Int. Conf. Innov. Comput. Technol. (INTECH)*, Aug. 2013, pp. 314–320.
[26] J. A. Dev, "On the imminent advent of botnet powered cracking," in *Proc. IEEE 2nd Int. Conf. Collaboration Internet Comput. (CIC)*, Nov. 2016, pp. 188–195.
[27] Z. Shi, C. Ma, J. Cote, and B. Wang, "Hardware implementation of hash functions," in *Introduction to Hardware Security and Trust*, vol. 10. New York, NY, USA: Springer, 2012, pp. 27–50.
[28] J.-w. Kim, H.-u. Lee, and Y. Won, "Design for high throughput SHA-1 hash function on FPGA," in *Proc. 4th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2012, pp. 403–404.
[29] S. B. Suhaili and T. Watanabe, "High throughput evaluation of SHA-1 implementation using unfolding transformation," *ARPN J. Eng. Appl. Sci.*, vol. 11, no. 5, pp. 3350–3355, 2016.
[30] S. B. Suhaili and T. Watanabe, "High-throughput message digest (MD5) design and simulation-based power estimation using unfolding transformation," *J. Signal Process.*, vol. 21, no. 6, pp. 233–238, 2017.
[31] L. Ioannou, H. E. Michail, and A. G. Voyiatzis, "High performance pipelined FPGA implementation of the SHA-3 hash algorithm," in *Proc. 4th Medit. Conf. Embedded Comput. (MECO)*, Jun. 2015, pp. 68–71.
[32] A. Mohamed and A. Nadjia, "SHA-2 hardware core for virtex-5 FPGA," in *Proc. IEEE 12th Int. Multi-Conf. Syst., Signals Devices (SSD)*, Mar. 2015, pp. 1–5.
[33] A. Al Maashri, L. Pathuri, M. Awadalla, A. Ahmad, and M. Ould-Khaoua, "Optimized hardware crypto engines for XTEA and SHA-512 for wireless sensor nodes," *Indian J. Sci. Technol.*, vol. 9, no. 29, pp. 1–7, Aug. 2016.
[34] Y. Huo and D. Liu, "A high-throughput processor for cryptographic hash functions," *J. Commun.*, vol. 11, no. 7, pp. 702–709, 2016.
[35] H. Mestiri, F. Kahri, B. Bouallegue, and M. Machhout, "Efficient FPGA hardware implementation of secure hash function SHA-2," *Int. J. Comput. Netw. Inf. Secur.*, vol. 7, no. 1, pp. 9–15, Dec. 2014.
[36] M. D. Rote, V. N, and D. Selvakumar, "High performance SHA-2 core using the round pipelined technique," in *Proc. IEEE Int. Conf. Electron., Comput. Commun. Technol. (CONECCT)*, Jul. 2015, pp. 1–6.
[37] H. E. Michail, G. S. Athanasiou, G. Theodoridis, and C. E. Goutis, "On the development of high-throughput and area-efficient multi-mode cryptographic hash designs in FPGAs," *Integration*, vol. 47, no. 4, pp. 387–407, Sep. 2014.
[38] H. E. Michail, G. S. Athanasiou, G. Theodoridis, A. Gregoriades, and C. E. Goutis, "Design and implementation of totally-self checking SHA-1 and SHA-256 hash functions' architectures," *Microprocessors Microsyst.*, vol. 45, pp. 227–240, Sep. 2016.
[39] J. Tyler, R. Daniel, H. Phillip, and Z. Joseph, "An FPGA architecture for the recovery of WPA/WPA2 keys," *J. Circuits Syst. Comput.*, vol. 24, no. 7, pp. 1–26, 2015.
[40] M. Kammerstetter, M. Muellner, D. Burian, C. Kudera, and W. Kastner, "Efficient high-speed wpa2 brute force attacks using scalable low-cost fpga clustering," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2016, pp. 559–577.
[41] Q. Ding, Z. Zhang, S. Li, and P. Liu, "Energy-efficient RAR3 password recovery with dual-granularity data path strategy," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.
[42] X. Bai, L. Jiang, J. Yang, Q. Dai, and M. Z. A. Bhuiyan, "Password recovery for ZIP files based on ARM-FPGA cluster," in *Proc. Int. Conf. Secur., Privacy Anonymity Comput., Commun. Storage*, 2017, pp. 405–414.
[43] Z. Jin and H. Finkel, "Evaluation of MD5Hash kernel on OpenCL FPGA platform," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2018, pp. 1026–1032.
[44] F. M. Vallina and S. Gilliland, "Performance optimization for a SHA-1 cryptographic workload expressed in OpenCL for FPGA execution," in *Proc. 3rd Int. Workshop OpenCL IWOCL*, 2015, p. 7.
[45] H. S. Jacinto, L. Daoud, and N. Rafla, "High level synthesis using vivado HLS for optimizations of SHA-3," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2017, pp. 563–566.
[46] R. Veras, C. Collins, and J. Thorpe, "On the semantic patterns of passwords and their security impact," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–16.
[47] W. Han, Z. Li, L. Yuan, and W. Xu, "Regional patterns and vulnerability analysis of chinese Web passwords," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 2, pp. 258–272, Feb. 2016.
[48] J. Ma, W. Yang, M. Luo, and N. Li, "A study of probabilistic password models," in *Proc. IEEE Symp. Secur. Privacy*, May 2014, pp. 689–704.
[49] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 11, pp. 2776–2791, Nov. 2017.
[50] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1242–1254.
[51] X. Wang, L. Feng, and H. Zhao, "Fast image encryption algorithm based on parallel computing system," *Inf. Sci.*, vol. 486, pp. 340–358, Jun. 2019.
[52] X. Wang and S. Gao, "Image encryption algorithm for synchronously updating Boolean networks based on matrix semi-tensor product theory," *Inf. Sci.*, vol. 507, pp. 16–36, Jan. 2020.
[53] X. Wang and S. Gao, "Image encryption algorithm based on the matrix semi-tensor product with a compound secret key produced by a Boolean network," *Inf. Sci.*, vol. 539, pp. 195–214, Oct. 2020.
[54] Y. Xian and X. Wang, "Fractal sorting matrix and its application on chaotic image encryption," *Inf. Sci.*, vol. 547, pp. 1154–1169, Feb. 2021.
[55] E. Tlelo-Cuautle, V. H. Carbajal-Gomez, P. J. Obeso-Rodelo, J. J. Rangel-Magdaleno, and J. C. Nuñez-Pérez, "FPGA realization of a chaotic communication system applied to image processing," *Nonlinear Dyn.*, vol. 82, no. 4, pp. 1879–1892, Dec. 2015.
[56] A. A. Rezk, A. H. Madian, A. G. Radwan, and A. M. Soliman, "Reconfigurable chaotic pseudo random number generator based on FPGA," *AEU Int. J. Electron. Commun.*, vol. 98, pp. 174–180, Jan. 2019.
[57] V. Muralidharan, S. Arumugham, S. Rethinam, S. Janakiraman, H. N. Upadhyay, and S. Rajagopalan, "Chaos blend DNA coding for image encryption on FPGA," in *Proc. Int. Conf. Comput. Commun. Informat. (ICCCI)*, Jan. 2018, pp. 1–6.
[58] A. J. Mansor, H. N. Abdalla, and H. T. Ziboon, "Digital image scrambling using chaotic systems based on FPGA," in *Proc. 3rd Sci. Conf. Electr. Eng. (SCEE)*, Dec. 2018, pp. 19–24.
[59] O. A. Aboulseoud and S. M. Ismail, "FPGA floating point fractional-order chaotic map image encryption," in *Proc. 31st Int. Conf. Microelectron. (ICM)*, Dec. 2019, pp. 134–137.
[60] H. A. Abdullah and H. N. Abdullah, "Fpga implementation of color image encryption using a new chaotic map," *Indonesian J Electr Eng Comput Sci*, vol. 13, no. 1, pp. 129–137, 2019.
[61] E. Tlelo-Cuautle, J. D. Díaz-Mu noz, A. M. González-Zapata, R. Li, W. D. León-Salas, F. V. Fernández, O. Guillén-Fernández, and I. Cruz-Vega, "Chaotic image encryption using hopfield and hindmarsh–rose neurons implemented on fpga," *Sensors*, vol. 20, no. 5, pp. 1–22, 2020.

BIN LI received the Ph.D. degree in software engineering from PLA Strategic Support Force Information Engineering University, Zhengzhou, China, in 2018. He is currently working with the School of Information Engineering, Zhengzhou University, Zhengzhou. His main research interests include high-performance computing and information security.

FENG FENG received the M.S. degree in computer science and technology from Zhengzhou University, Zhengzhou, China, in 2019, where he is currently pursuing the Ph.D. degree. His main research interests include high-performance computing and information security.

XIAOJIE CHEN received the M.S. degree in computer science and technology from Zhengzhou University, Zhengzhou, China, in 2019, where he is currently pursuing the Ph.D. degree with PLA Strategic Support Force Information Engineering University. His main research interests include high-performance computing and information security.

YAN CAO received the Ph.D. degree in computer application technology from PLA Strategic Support Force Information Engineering University, Zhengzhou, China, in 2013. He is currently working with the School of Information Engineering, Zhengzhou University, Zhengzhou. His main research interests include network security and parallel computing.