Design and Implementation of a Domain Specific Rule Engine

Mengdong Chen(✉), Xinjian Zhou, Dong Wu, and Xianghui Xie

State Key Laboratory of Mathematical Engineering and Advanced Computing, Henghua Science and Technology Park, Wuxi 214125, Jiangsu, China
chen.mengdong@meac-skl.cn

Abstract. Security strings are often needed in identity authentication mechanism. Security strings recovery is a reverse process, which does much calculations on a large amount of possible strings to find the right one, so that we can recover lost or forgotten strings and regain access to valuable information. In this reverse process, we need first process basic strings based on transformation rules, so as to generate new ones quickly. Rule processing is complex, which has high requirements for computing power, processing time, especially system power consumption. In response to the above requirements, this work puts forward the idea of accelerating the processing of rules using hardware for the first time, and a domain specific rule engine is designed and implemented on the existing FPGA platform. The experimental results show that the performance of the rule engine on a single Xilinx Zynq 7z030 FPGA is better than that of CPU, its performance power ratio is 3 times higher than that of GPU, and 50 times higher than that of CPU. The speed and energy efficiency of the rule processing is improved effectively.

Keywords: String · Rule · Engine · Domain specific

1 Introduction

With the development of computer technology and the expansion of Internet scale, identity authentication mechanism has gradually become an important way for people to protect their information [1]. The authentication process requires an identity information consists of a username and a security string. The HASH algorithm is usually used to calculate the digests of secure strings and the digests are stored together with user credentials. When the user authenticates his identity, the authentication system receives the security string inputed by the user and uses the HASH algorithm to convert the string into a digest and compares it with the digest value stored in the system to complete the authentication process. The forgetting of security strings can cause inconvenience and loss [2]. The analysis technology of authentication protocol is just to solve this problem.

In the analysis of authentication protocol, a large number of to-be-tested strings need to be quickly generated in a short period of time for the subsequent HASH algorithm to calculate the digest value. And the digest value is then compared with the stored digest, so that the correct string can be found. In the process of generating of

W. Xu et al. (Eds.): NCCET 2018, CCIS 994, pp. 108–119, 2019.
https://doi.org/10.1007/978-981-13-5919-4_11

possible strings, using dictionary and transformation rules is a very accurate and effective way [3–5]. Based on transformation rules and existing dictionaries consisted of basic strings, it is possible to generate a large number of strings with a higher probability, which in turn can increase the speed of analysis and improve accuracy.

As there are many kinds of transformation rules, and the calculation is complex, rule processing is a task with great demand for computation power and processing time. To the best of our knowledge, the public implementation methods are all based on CPU and GPU now [6, 7], which have many shortcomings in processing speed and system power consumption. Aiming at the rule processing in analysis of authentication protocol, this article presents a hardware-implemented, energy-efficient, reconfigurable rule processing architecture, and implements a domain specific rule processing engine. The research in this article is based on Xilinx Zynq FPGA. The experimental results show that the engine performs well in terms of processing performance and system power consumption.

2 Rule and Its Implementation Platform

2.1 Rules in the Analysis of Identity Authentication Protocal

When setting a secure string, people often set up a new string based on a simple transformation, such as adding a prefix, adding a suffix, etc., this transformation is called a transformation rule [6, 7]. This rule-based approach provides an idea for the analysis of identity authentication protocols. By collecting known security strings, a dictionary can be formed. The analysis can be attempted in the dictionary. Compared to the full-character search space, the amount of calculation here can be significantly reduced and a higher probability of hits can be ensured. At the same time, by applying rules to the dictionary, new strings can be generated, which expands the coverage of the dictionary, and improves the hit rate. The exquisitely set dictionaries and rules can significantly increase the hit rate of the analysis when satisfying the limitations of search scale, time limit, and the like.

In the analysis of authentication protocol, there are many string transformation rules accumulated. Multiple tools have their own supported rules and provide a dictionary plus rule analysis mode.

John the Ripper [6] is an open source and free analysis software, its main purpose is to analyze the weak Unix passwords. It now supports more than 100 kinds of algorithms, and provides support for many different types of system architectures, including Unix/Linux, Windows/DOS and OpenVMS. It supports dictionary analysis mode, and supports more than 40 kinds of string transformation rules and their handling. The rules are processed on CPU.

Hashcat [7] is a widely used multiplatform free analysis kit, which supports various platforms with OpenCL runtime, including CPU, GPU (supporting NVIDIA GPU and AMD GPU), DSP, FPGA, etc. It supports multiple operating systems, including Linux, Windows, MacOS, etc. It supports distributed processing, nearly 200 algorithms, and multiple analysis modes. It supports the processing of dictionaries and rules, and its rules are processed mainly on CPU and GPU.

Based on the rules used by Hashcat, this article studies 41 common basic transformation rules and implements their acceleration engine. Table 1 lists several typical transformation rules. Each rule takes a visible character as its mnemonic, some rules need parameters, and the number of parameters varies from 0 to 3. Table 1 illustrates the transformation results of the rule by taking the string *p@ssW0rd* as an example.

Table 1. Rules and their meanings

Mnemonic	Description	Example	Transform result
u	Uppercase all letters	u	P@SSW0RD
r	Reverse the entire word	r	dr0Wss@p
pN	Append duplicated word N times	p2	p@ssW0rdp@ssW0rdp@ssW0rd
{	Rotates the word left	{	@ssW0rdp
DN	Deletes character at position N	D3	p@sW0rd
iNX	Inserts character X at position N	i4!	p@ss!W0rd
sXY	Replace all instances of X with Y	ss$	p@$$W0rd
*XY	Swaps character at position X with character at position Y	*34	p@sWs0rd
+N	Increment character @ N by 1 ascii value	+2	p@tsW0rd

In actual use, several individual rules can be combined together to carry out one transform, such as *uD3ss$3*, which is combined of 3 individual rules. A new string is generated after all 3 rules are processed

2.2 Rule Processing Platform with High Efficiency

The analysis process of identity authentication protocol needs to search and calculate the string space made up of visible characters to find the correct string. When the length of the string increases, the space of search and the amount of computation all increase exponentially [8, 9]. Moreover, the analysis and calculation include a large number of computationally intensive modules. The computational power of a single computing node cannot meet the requirements. Even in the dictionary and rules mode which is a certain targeted analysis pattern, the number of rules and basic strings is also very large. Take a dictionary file with 50 million entries and a rule file with 100 thousand entries as an example, only the processing of rules will need to generate 5×10^{13} to-be-tested new strings, which contains huge amount of computation. Even we use MD5, the simplest HASH algorithm an example, it still takes more than ten days to finish the analysis process on a single common CPU. It can be seen that the computing power of a single computing node is still far from the analysis task of identity authentication protocol. The usual practice is to build large-scale computing clusters, divide the search space into different computing tasks, and each node conducts search and calculation in its own task space to speed up the entire analysis process [10].

When constructing large-scale analysis clusters, performance and power consumption are two main concerns. Rule processing and authentication protocol analysis are both computationally complex tasks. Common CPUs have encountered bottlenecks in performance improvement, and their computational capabilities have fallen far short of the requirements. The advent of GPU acceleration units has made them clearly superior in performance. And the theoretical computational performance in dealing with high integration, computationally intensive issues, etc. has substantially exceeded that of general-purpose processors [11, 12]. However, the GPU also has problems as an acceleration device. Especially when building a large-scale computing system, its construction cost, frequency wall, power-consumption wall, and storage wall have made the GPU's high cost and high power consumption intolerable [13].

The dedicated ASIC has a high degree of integration and high processing performance. However, the development is complex and the cost is high. Once the function is implemented, it cannot be changed, and it is not suitable for the acceleration of rule processing.

FPGA has the characteristics of low power consumption and high parallelism. It can not only accelerate the computing speed, but also keep power consumption within acceptable range [13–15]. Its wide application and reconfigurable characteristics provide a basis for its application in accelerating rule processing. Its low power consumption, high parallelism and strong expansibility make it suitable for accelerating the processing of rules and the analyzing of identity authentication protocol. Focusing on processing performance and energy efficiency. This article studies the rule processing techniques, designs and implements a rule engine with FPGA.

3 Design of the Rule Engine

The rule engine accelerates the parsing of the rules in a fully hardware-implemented manner. When processing, the software only needs to configure the size and location of the rule and dictionary files. The rule engine can automatically obtain rules and dictionaries from the off-chip, parse them, and generate new strings. The newly generated strings can be written back to the off-chip memory space through a high-speed bus for use by other applications. It can also integrate HASH authentication algorithms on-chip to directly verify the correctness of the string and complete the analysis of the entire identity authentication protocol.

3.1 Structure of the Hardware Platform

The rule engine of this article is based on an identity authentication protocol analysis system. This analysis system is a large-scale reconfigurable computing cluster. Its computing power comes from a large number of low-power reconfigurable Xilinx Zynq XC7Z030 chips. The chip is a hybrid core processor that includes a general-purpose embedded computing core (a dual-core ARM CortexTM-A9 processor running at 1 GHz) and an FPGA-based reconfigurable compute core [16]. The two heterogeneous computing resources are tightly coupled through a high-speed interconnection bus, which can support the parallel collaborative execution of general-purpose computing tasks and accelerated computing tasks. The hybrid processing platform integrates 1 GB

of low-power DDR memory, 32 GB of flash memory, Gigabit Ethernet interfaces, and high-speed ring network interfaces and so on. The structure diagram of the computing platform based on Zynq XC7Z030 is shown in Fig. 1. The platform is visible in Fig. 2.

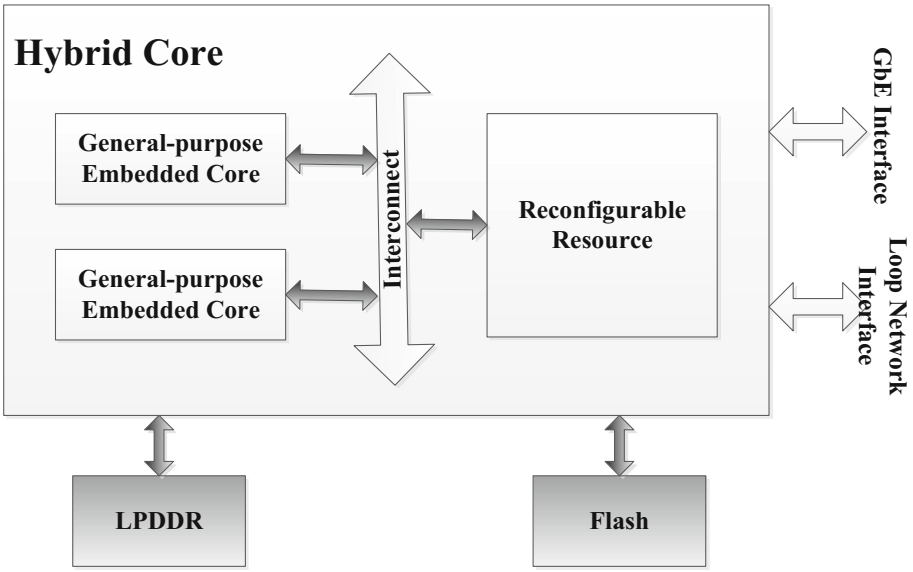

Fig. 1. Structure of the hardware platform

The rule engine is implemented on the reconfigurable FPGA of a single Zynq XC7Z030, and the engine can be integrated in each FPGA in the large-scale system. Rule files and dictionary files are divided into smaller parts according to computing tasks, and stored in off-chip low-power DDR memories.

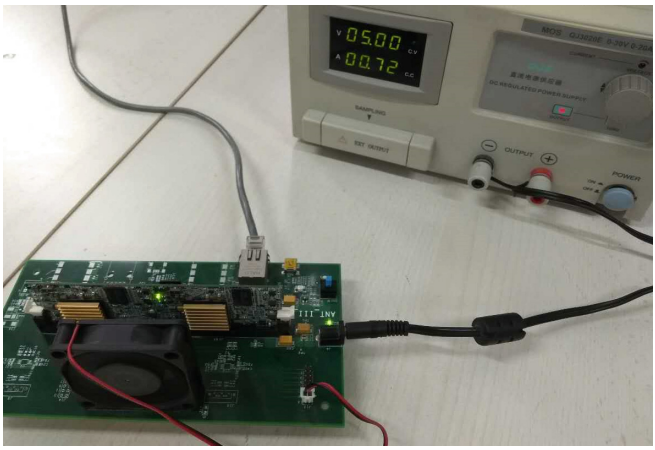

Fig. 2. Physical picture of the hardware platform

3.2 Design of the Data Format

Rule-based identity authentication protocol analysis requires that a large number of transformation rules be organized into rule files and common strings be organized into dictionary files for storage and usage. In the hardware implementation of the rule engine, expansion requirements, storage space limitations, and the readability of the rules should be considered. Each rule is coded with a specific length of 8 bits. We directly encode the rules with ASCII code of their mnemonic. The 8-bit code can theoretically support up to 256 rules, leaving room for new extensions for future rules. The rules are coded and stored by ASCII code, which ensures the readability of rule files and reduces the workload of translation and conversion between hardware and software.

When performing string transformation in reality, several individual rules are often combinated together to perform one transform. In designing the hardware storage space, considering the rule combination and its parameters, 40 bytes of storage space is allocated for each transformation. Generally, the length of the string in each dictionary file is the same. In the hardware logic, 32 bytes of storage space is allocated for each dictionary entry, that is, the length of the string is a maximum of 32. The rule file and dictionary file format, as well as its storage form in hardware, are shown in Fig. 3.

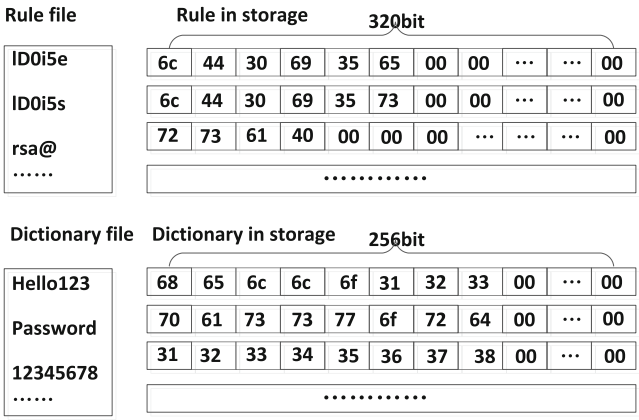

Fig. 3. The storage format of rule file and dictionary file

3.3 Design of the Engine Core

Rule processing is the process of decoding the rules, and then transforming the strings to get new ones. Because different transformation rules will change the length of the string, even if the input string has the same length, the output string will still be of different lengths, which will cause difficulties for subsequent usage. The solution is to categorize the dictionary files, and each time the processed dictionary file has the same string length. When processing, all the dictionary entries in the dictionary file are first looped for one rule, and thus, these new generated strings are of a same length, which facilitates

the use for subsequent HASH pipelines. Then we use another rule, reacquire dictionary file and loop through it. This process continues, change another rule, reacquire dictionary and loop through the dictionary until all rules in the rule file are processed.

For the case where rules are combined together to apply a transformation to a string, the processing of the latter rules depends on the processing result of the preceding ones, the rules can only be executed in order. During the design process, this article optimized the execution time of each rule, and processed each rule within one clock cycle, including the analysis of the rule and the transformation of string. In this way, one rule is processed every clock cycle. Figure 4 depicts the detailed processing of 3 transformations, each time the transformation is a combination of 3 rules and the number of dictionary entries is k. Because each transformation is made up of 3 basic rules, 1 new string can be generated in every 3 clock cycles.

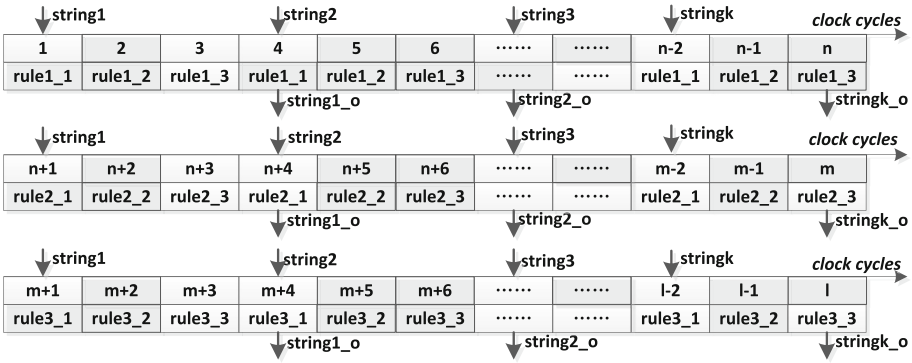

Fig. 4. The procedure of rule processing. The horizontal axis is the time axis and represents the clock period. *string1*, *string2*, *string3*, …… *stringk* are k entries of 1 dictionary. *rule1_1*, *rule1_2*, *rule1_3* are 3 basic rules, which together constitute 1 transformation. r*ule2_1*, *rule2_2*, *rule2_3, rule3_1*, *rule3_2*, *rule3_3* are similar to this. *string1_o*, *string2_o*, ……, *stringk_o* are k outputs of the transformations. The clock cycles *1* to *n* complete the first transformation of the k dictionary entries, and *n + 1* to *m*, *m + 1* to *l* complete the second and third transformations, respectively.

The brief structure of the engine core is demonstrated in Fig. 5. Each of the 41 basic rules is designed as a single rule processing element (RPE). The periphery is a preprocessing circuit, a rule decoding circuit, and a memory management circuit. The preprocessing circuit is responsible for dividing the continuously stored rule and dictionary files into entries. The rule decoding circuit decodes the basic rules of each transformation one by one, and then selects the corresponding rule acceleration unit to perform operations. Each RPE completes the transformation according to the input string and the string length, and calculates the length of the newly generated string. The storage management circuit is responsible for using the high-speed bus to obtain rule and dictionary files from off-chip, form on-chip caches at various levels, and output the generated new strings off-chip as needed. All of this work is done automatically by hardware.

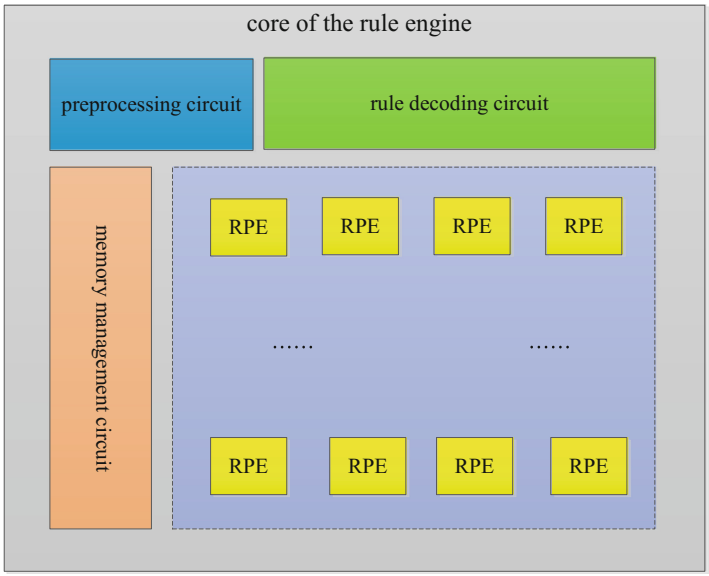

Fig. 5. Brief structure of the engine core

The processing logic of 41 basic rules constitutes a processing core. According to the limitation of hardware resources, one or more processing cores can be placed at the same time within the rule engine, and each core takes its own rule and dictionary to transform.

3.4 The Storage Architecture

The rules engine automatically accesses rules and dictionaries. To meet the speed requirements of the high-speed engine for rules and dictionaries, a total of three levels of storage structures are set.

The first level of storage: off-chip DDR. The rule and dictionary files are stored in DDR. The CPU in Zynq configures the start address and size information of the rule and dictionary files in DDR to the FPGA logic, and the hardware automatically obtains the rules and dictionary. At the same time, if the new strings generated by the rule engine need to be passed off-chip for use by other applications, they are also automatically transferred to the DDR memory by the rule engine.

Second level storage: On-chip RAM. The rule engine in FPGA pre-fetches the rules and dictionary into the FPGA through the AXI (Advanced eXtensible Interface) bus and caches them in the on-chip RAM. The four high-performance AXI_HP interfaces of the AXI bus can achieve a total bandwidth of 4.8 GB/s when operating at 150 MHz, which can guarantee the speed demand of the rule engine. As processing logic continues to consume data in RAM, the rule engine continuously acquires data from off-chip and guarantees the demand.

Third-level storage: On-chip FIFO. The processing logic acquires dictionary data from the on-chip RAM and performs preprocessing, and then stores the dictionary in the on-chip FIFO buffer for high-speed processing by the core processing logic.

4 Experiments and Results

In order to verify the correctness and performance of the designed rule engine, this article develops and implements it on the hardware platform based on Zynq 7z030 chip, through the Vivado (v2015.2) tool suite. The number of different processing cores, different string lengths, and different combinations of rules were tested, and their performance and power consumption were analyzed. The results were compared and analyzed with those of other platforms.

4.1 Results of the Implementation

The design is synthesized and implemented through the Vivado tool. Based on the constraints of hardware resources, the number of processing cores that can be placed and the overall performance are analyzed. Performance is calculated as the number of new strings that can be generated per second when the transformation is combined of 1 rule.

The results show that the maximum implementation frequency is 150 MHz and the maximum number of cores that can be put on chip is 2, so the maximum processing performance is to process 300M basic rules and generate 300M new strings per second. In the case of placing a single processing core on chip, the resource occupancy is 42%, and there are still enough resources to place the HASH algorithm pipeline, which can be used for on-chip verification. If two engine cores are placed on the chip, the resource consumption is about 80%, at this time, the HASH algorithm pipeline cannot be placed on the chip, and the rule engine can be used to transfer the generated new strings to the off-chip for other applications.

4.2 Comparison with Other Platforms

As far as we know, this is the first work that realizes the work of rule processing using hardware.

The comparisons of performance and power consumption are mainly performed with the software implementations on the CPUs and GPUs. The same rule and dictionary files are run on CPU and GPU, respectively. And the results are compared with this work. The two aspects of comparison are performance and power consumption.

Software implementation uses the latest hashcat 4.1.0, which is the industry's fastest analysis tool and supports both CPU and GPU platforms [5]. The result of software has a great relationship with its running platform, for example, in NVIDA GPUs, its desktop products and products specifically designed for high-performance computing have a huge gap in computing power. This work selects two mainstream

product platforms for experimentation. The adopted CPU is: Intel(R) Core(TM) i7-6700 CPU @ 3.40 GHz with 32G memory. The adopted GPU is: NVIDIA GeForce GTX 970, with 1664 processing cores, running at 1.18 GHz. The performance comparison results are shown in Fig. 6.

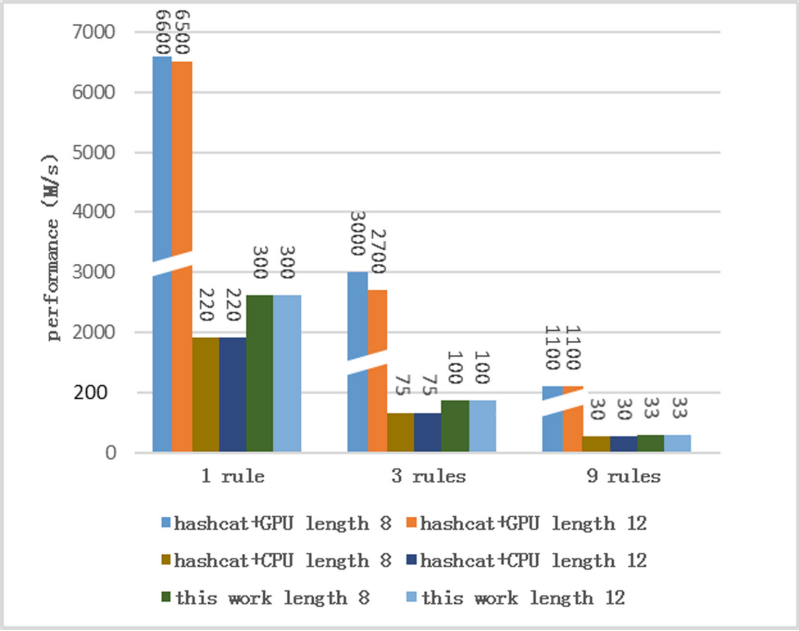

Fig. 6. The comparison of performance between different platforms. For the rule combination, we tested the combination of 1 rule, 3 rules and 9 rules on each platform. For the length of the strings in the dictionary, we tested the situation of 8 bytes and 12 bytes on each platform. Performance is calculated as the number of new strings generated per second.

Through analysis, it can be found that for different string lengths, the processing performance of the three platforms is not affected. The processing performance is greatly affected by the combination of rules. The more rules are combined, the more complex the processing is. Our rule engine can achieve the performance of 300M per second when the transformation is combined of 1 rule. The performance of this work is better than that of the CPU implementation, and is worse than that of the GPU implementation. However, when using the rule engine of this article to build a large-scale, low-power computing cluster, its computing power will increase significantly. But this is not the work of this article.

In actual operation, the running power consumption of the rule engine and the GPU platform is observed in real time. The power consumption of the CPU is calculated as 65 W and the performance power ratio is calculated (the number of rules that can be processed per second per watt). The results are shown in Fig. 7. The operating power of the rule engine is only 2 W, and its performance power ratio is 3 times higher than

that of the GPU, and it is 50 times higher than that of CPU. We can see that the processing speed of the rule engine is fast enough, and its power consumption is not large, which is very suitable for building large-scale processing systems.

Fig. 7. The performance power ratio of different platforms

5 Conclusion

The processing of rules is an important part of identity authentication protocol analysis. Its process is complex, and it has high requirements for processing performance and system power consumption. This work proposes the acceleration of the rule processing with all hardware approach for the first time. We build a domain-specific rule engine using FPGA's high parallelism and low power consumption, and implement it on Zynq 7z030 FPGA. The experimental results show that the running performance of the rule engine is better than that of Intel i7-6700 CPU. The performance power ratio is 3 times higher than that of NVIDIA GeForce GTX 970 GPU, and about 50 times higher than that of CPU platform.

It effectively improves the speed and energy efficiency of rule processing. The rule engine designed in this paper has high processing performance, low system cost and low operating power consumption. It is particularly suitable for constructing large-scale, distributed, and reconfigurable rule processing systems, thereby providing a basis for the design and implementation of the entire identity authentication protocol analysis system.

Acknowledgements. This research was supported by National Natural Science Foundation of China (No. 91430214; 61732018). I am also grateful to my tutor and colleagues who had helped me in this project.

References

1. Kammerstetter, M., Muellner, M., Burian, D., Kudera, C., Kastner, W.: Efficient high-speed WPA2 brute force attacks using scalable low-cost FPGA clustering. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 559–577. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_27
2. Houshmand, S., Aggarwal, S., Flood, R.: Next gen PCFG password cracking. IEEE Trans. Inf. Forensics Secur., 1776–1791, August 2015. https://doi.org/10.1109/tifs.2015.2428671
3. Wang, D., Zhang, Z., Wang, P., Yan, J., Huang, X.: Targeted online password guessing: an underestimated threat. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, pp. 1242–1254. ACM, October 2016
4. Huh, J.H., Oh, S., Kim, H., Beznosov, K., Mohan, A., Rajagopalan, S.R.: Surpass: system initiated user-replaceable passwords. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 170–181. ACM, October 2015
5. Melicher, W., et al.: Fast, lean, and accurate: modeling password guessability using neural networks. In: USENIX Security Symposium, pp. 175–191, August 2016
6. John the Ripper password cracker. http://www.openwall.com/john/doc/
7. Hashcat advanced password recovery. https://hashcat.net/hashcat/
8. Ji, S., Yang, S., Hu, X., Han, W., Li, Z., Beyah, R.: Zero-sum password cracking game: a large-scale empirical study on the crackability, correlation, and security of passwords. IEEE Trans. Dependable Secure Comput. **14**(5), 550–564 (2017)
9. Brumen, B., Makari, T.: Resilience of students' passwords against attacks. In: 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, pp. 1275–1279. IEEE, May 2017
10. Chang, D., Jati, A., Mishra, S., Sanadhya, S.K.: Cryptanalytic time–memory trade-off for password hashing schemes. Int. J. Inf. Secur., 1–18 (2016)
11. Qiu, W., Gong, Z., Guo, Y., Liu, B., Tang, X., Yuan, Y.: GPU-based high performance password recovery technique for hash functions. J. Inf. Sci. Eng. **32**(1), 97–112 (2016)
12. Akleylek, S., Tok, Z.Y.: Efficient arithmetic for lattice-based cryptography on GPU using the CUDA platform. In: 2014 22nd Signal Processing and Communications Applications Conference (SIU), pp. 854–857, April 2014 (2014)
13. Li, X., Cao, C., Li, P., Shen, S., Chen, Y., Li, L.: Energy-efficient hardware implementation of LUKS PBKDF2 with AES on FPGA. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 402–409 (2016). https://doi.org/10.1109/trustcom.2016.0090
14. Wang, C., Mao, J., Leng, T., Zhuang, Z.Y., Wang, X.M.: Efficient acceleration for total focusing method based on advanced parallel computing in FPGA. Int. J. Acoust. Vib. **22**, 536 (2017)
15. Abbas, A., Voss, R., Wienbrandt, L., Schimmler, M.: An efficient implementation of PBKDF2 with RIPEMD-160 on multiple FPGAs. In: 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pp. 454–461 (2014). https://doi.org/10.1109/padsw.2014.7097841
16. Zynq-7000 All Programmable SoC. https://www.xilinx.com/support/documentation/product-briefs/zynq-7000-product-brief.pdf