

# A Protection Method of Target Codes

Ge Hongmei<sup>1</sup>, Xu Chao<sup>1,2</sup> & Shi Qian<sup>2</sup>

<sup>1</sup> XuZhou College of Industrial Technology, Xuzhou 221000, China

<sup>2</sup> State Key Laboratory of Software Engineering of Wuhan University, Wuhan 430072, China

Correspondence: Xu Chao, XuZhou College of Industrial Technology, Xuzhou 221000, China; State Key Laboratory of Software Engineering of Wuhan University, Wuhan 430072, China. E-mail: xuch@whu.edu.cn

Received: September 21, 2012 Accepted: October 11, 2012 Online Published: October 18, 2012

doi:10.5539/cis.v5n6p43

URL: <http://dx.doi.org/10.5539/cis.v5n6p43>

## Abstract

In order to protect the target codes generated by compiler, this paper puts forward a scheme combining compiler with hardware platform on the base of the existing protection method of executable code. First, classify codes of C Language into four security levels to determine the protection strategies for different codes. Secondly, design and implement the corresponding encryption module, decryption module and decompression module in VHDL Language after studying the structure and the principle of FPGA and the method of development software. Thirdly, decompress and decode the target codes through FPGA modules when target codes is being executed. Run the target codes only when they are qualified. Finally, implement and simulate the scheme through FPGA. The simulation experiment verifies the correctness and effectiveness of encryption module, decryption module and decompression module.

**Keywords:** target codes protection, FPGA, AES algorithm, BCC algorithm

## 1. Introduction

With the development of modern information society, human depends on the software more than the computer. As a result of the rapid popularization of computer in the social life, demand for software application is increasing day after day. At the same time the complexity of software is also rising, demand for reliability is gradually enhanced. However, the larger software systems has become more fragile, sometimes they work in an unexpected way and cause tremendous damage directly or indirectly to the users, for example, *the incident of Aliya No.5 Rocket in Europe in 1996 and ambulance events in France in 1999 and so on*. Software fault has been gradually the main source of failure in safety critical systems. In order to develop more secure and reliable computer software, many countries invest a lot of manpower and material resources in software reliability research.

The software is composed by codes, its reliability fundamentally can be manifested as code reliability, so it is an important way that the security protection for codes guarantees the reliability of software.

At present code protection methods can be classified into two types: hardware protection method and software protection method. Only one method is hard to keep balance among security intensity, system cost and implementation cost. Hardware protection method adopts cryptographic technique, so security intensity can be increased and secure coprocessor can increase code execution speed, but the requirement for hardware increases the system's implementation cost. Software protection method such as watermark or code obfuscation can reduce the cost, but system security and code execution efficiency can be reduced as well.

This paper puts forward a scheme combining compiler with hardware platform on the base of the existing protection method of executable code. The scheme is implemented and simulated through FPGA(Field Programmable Gate Array) reduce. The method not only implements the high security protection for codes but also reduces the cost of system implementation.

The structure of this paper is presented as follows: introduction related to technical theory in Section 2; introduction to the scheme of target code protection in Section 3; how to implement encryption/decryption module and decompression module through FPGA in Section 4; related work and conclusion in the last section.

## 2. Related Theory

### 2.1 Instruction to Hardware and Software of Design Platform

#### 2.1.1 Integrated Development Environment of Quartus

Quartus II software is an integrated development tool for logic circuit design provided by Altera company. It not only includes a series of tools for FPGA development process, but also provides a third party software interface to facilitate the enhancement of function.

Design process of Quartus II usually starts with compiling RTL codes, then verifies if RTL codes can achieve the correct function and if the results can meet the time requirements of the design after the layout, finally configures the programming files to the target FPGA devices to complete the entire development process.

#### 2.1.2 FPGA

FPGA (Field - Programmable Gate Array) is a further development product based on PAL, GAL, CPLD and other programmable devices. It appears as a semi custom circuit in the field of an application-specific integrated circuit (ASIC), so it can solve the lack of custom circuit and overcome that the numbers of original programmable device gate circuit are limited FPGA basically can be divided into six parts: the programmable input / output unit, basic programmable logic unit, embedded block RAM, rich routing resources, the embedding function unit and embedded special hardcore etc.

### 2.2 Introduction to Algorithm

#### 2.2.1 AES Algorithm

AES(Advanced Encryption Standard)is also called Rijndael encryption method, a block encryption standard adopted by the United States federal government. This standard has replaced DES and been widely analyzed and used in the world.

This paper adopts the AES symmetric encryption algorithm to encrypt the data processing. In this way it can not only protect code security better, but also implement hardware conveniently.

#### 2.2.2 Compression Algorithm

The compression algorithm is mainly divided into two categories: lossless compression and lossy compression. Lossless compression refers to a process through which the raw data can be restored after the decompression of the compressed information. Lossy compression refers to a process through which the raw data is slightly at variance with the reprocessed data, but its expression remains intact. This paper compresses the target codes to save storage space, adopt to the equipment with low storage capacity and further reduce the amount of encryption, decryption and verification. The paper designs a simple compression algorithm (BCC algorithm), which encodes the binary data directly to save the cost of mapping table and dictionary, adapt to smaller process compression; on the other hand, reduce complexity of system security enhancement module through software and hardware platform.

## 3. Protection Scheme

### 3.1 models of Code Security Level

According to the statistics of CNVD, leaks ratio caused by input validation error is up to 29.82%, the ratio of design error followed by is 21.56%, the ratio of boundary conditions error is 19.91% from Jan of 2001 to July of 2010.The input validation error is due to the failure to check the validity of input data provided by users, resulting in a buffer overflow, illegal pointer and so on. The proportion of buffer overflow in all leaks accounts for 2.4%. Thus, buffer overflow and pointer problem are the main problems, Leaks caused by boundary conditions error rank second. Therefore, by analyzing the statistical information and the fault in C language, this paper divides C language into four security levels as shown in Table 1.

Table 1. Models of code security level

C Language	Security Level
buffer overflow, illegal pointer	3
Array bounds	2
conditional statement, type conversion memory leaks, void type	1
etc	0

### 3.2 Strategies for Codes Protection

Strategies for codes protection are divided into three stages:

Stage 1: After compiler divides source program into basic blocks, the paper marks the security level for each statement on the base of models of code security level. According to  $S_{bi} = \text{Max}(S_{c_{bi},k})$ , security level of each basic block is calculated (where  $b_i$  represents basic block  $I$ ,  $c_{bi}$  represents statement  $j$  in basic block  $b_i$ ,  $S_k$  represents the security level of  $K$ , which is the basic block and the statement in basic block). Finally, write the mark to the corresponding configuration file.

Stage 2: When the target code of compiler is generated, the target code is compressed with compression algorithm. According to the actual results of compression, security control file will be modified. Finally, encrypt the target code and verify the mark insertion, at the same time, encrypt and verify the security control file according to Table 2.

Table 2. Security code level and process mode

Code Level	Process Mode
0	No handling
1	Insert verification mark
2	encryption
3	Encryption + verification

Stage 3: The target codes will be run after decompressed and decrypted through FPGA. The speed of decompression and decryption is faster than that of software by using hardware implementation. Illegal invaders are prevented from snooping on bus signals to obtain the corresponding instructions for the part communicates with the processor directly. General framework is described in Figure 1. Before running the program, decrypt and verify the data of security control file. Only when the data is qualified can it be loaded into CBB Module of FPGA to control the execution of program as shown in Figure 2. When a block is loaded into a cache running, security mark should be checked. The data is decrypted and verified in terms of security mark information. Only when the data is qualified can it be transferred to decompression device. After being decompressed, the data will be transferred to the processor. Exit the program and stop executing, provided one link malfunctions.

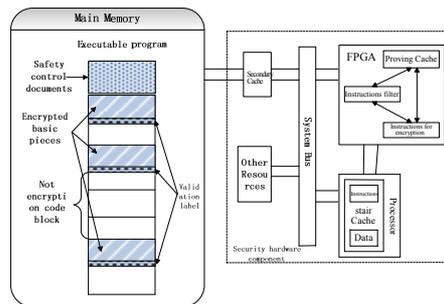


Figure 1. General framework

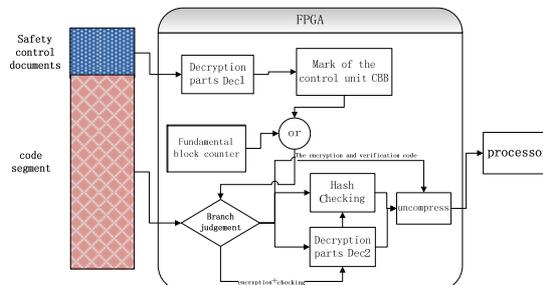


Figure 2. A blueprint for inner module of FPGA

## 4. Specific Implementation

### 4.1 FPGA Implementation of Encryption / Decryption Module

General framework of AES algorithm is shown in Figure 3.

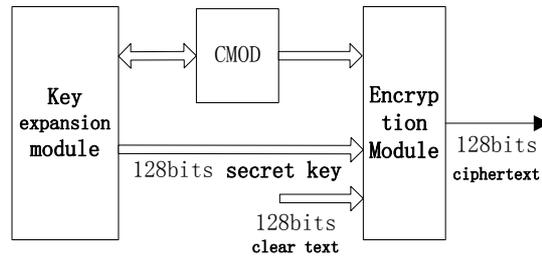


Figure 3. General framework of AES

#### 4.1.1 Key Expansion Module

Key Expansion Module is used for the generation of circulating keys. It is a process that a 128 bit key as an initial input key generates a key needed in the back 10 rounds' operation through RotWord, SubByt and Rcon XOR operation. pio[17..0] is the corresponding 18 keyboards' signal when the initial key is inputted. pio[15..0] represents hexadecimal numeral f, e, d...1, 0 of keyboard input. pio[16], a flag bit of input end, represents enter key of keyboard input. pio[17], a flag bit deleting input error, represents back key of keyboard input. Wren is the enabling signal of keyboard input, effective on low level. rst represents module reset signal, effective on low level. 'rdaddress[3..0]' represents reading Key RAM address signal. 'rden' is the enabling signal. 'rdclock' is reading key clock signal. 'ready\_finish' is key expansion finish signal. 'outkey[127..0]' is circulating key output signal representing a 128 bit key after expansion. As shown in Figure 4.

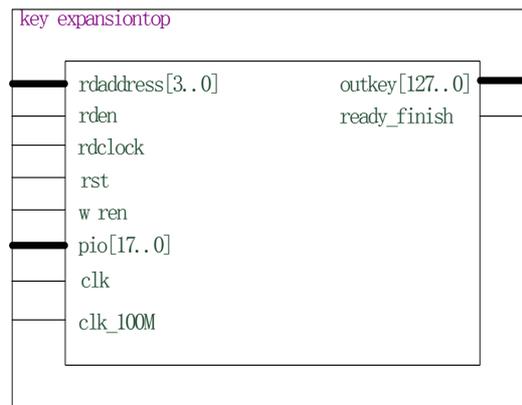


Figure 4. Key expansion module diagram

#### 4.1.2 Control Module

Control module is used to control the execution sequence of key expansion module and encryption module so that seal expansion module can correctly generate 10 rounds' keys needed by encryption module. Control module diagram is shown in Figures 4-5, in which rdena, rdclk and rdaddress are respectively enabling signal for reading keys, clock signal and address signal, keydrdy is clock signal for encryption module to read keys, countout is indication signal for key rounds' number, keyeffect is effective signal for key finish. Control module diagram is shown in Figure 5.

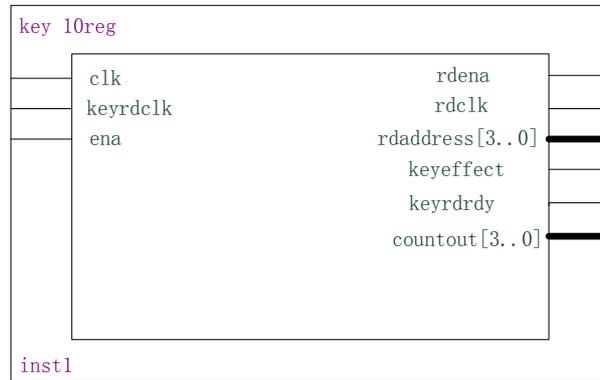


Figure 5. Control module diagram

#### 4.1.3 Encryption/Decryption Module

Connect S-box transformation module, line transformation module, column transformation module and key XOR module in logic execution sequence. Modules are cooperated through 128 bit register so that each part can operate in the light of clock signal to generate encryption module. In the module, *clk* is encryption clock signal, *ena* is encryption enabling signal with effectiveness on high level. When *ena* is connected with *keyeffect*, key signal is effective and encryption module can be encrypted. *Countkey* represents current round number of input key. *Keyrdclk* signal connecting with *keyrdrdy* signal of control module becomes clock signal for reading key. *Keyinstart* is flag signal finishing key expansion. Data input [127..0] is the 128 bit input data signal which need be encrypted. Round key [127..0] is the 128 bit round key signal needed in encryption. Data output [127..0] is the 128 bit output data signal after encryption. Available is flag signal finishing 10 rounds' encryption, effective on high level. *Roundi* [3..0] is flag signal of current encryption rounds' number in the process of encryption. General module diagram of encryption module is shown in Figure 6.

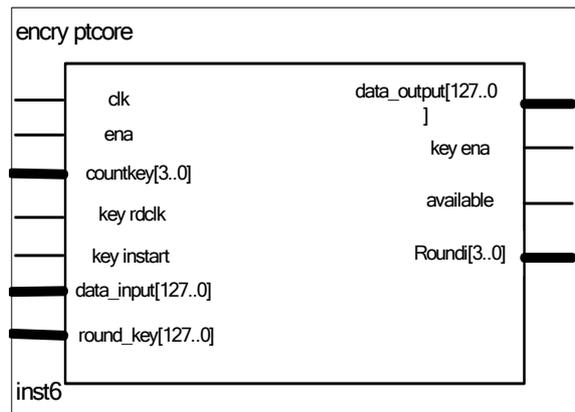


Figure 6. Encryption module diagram

#### 4.2 FPGA Implementation of Decompression Module

Decompression module decompresses the compressed data of basic block to and then restore to raw data under the control of clock signal. Basic block is divided in terms of 64 B (512 bit), so the data of the compressed basic block must be less than 64 B. Based on the principle that no enough width should be supplied with zero at low position, the data of basic block can be expanded into 521 bit for program to operate. In the module *CLK* is clock signal of decompression, *ena* is enabling signal of decompression, *datain*[511..0] represents the compressed data of basic block after expansion, *available* is effective signal of the decompressed data, *dataout*[511..0] represents the decompressed data of basic block. Decompression module diagram is shown in Figure 7.



support of hardware platform the codes generated by the compiler can be encrypted. But the development of this technique which is based on codes system structure is limited by the high cost of the security hardware platform.

In view of these, this paper proposes a scheme combining the compiler with hardware platform. The protection method that combines software with hardware not only makes up for the deficiency of algorithm secrecy level but also decreases the cost of security protection. When target codes are compressed, the length of flag codes will be reduced to a minimum. In this way the cost of security protection can be reduced. Finally, the scheme is implemented and simulated through FPGA to form an effective protection framework for target codes.

## 6. Conclusion

This paper puts forward a target codes protection scheme combining hardware protection with software protection. In the scheme the compiler compresses the generated target codes and parts of codes are encrypted through FPGA. Then the compiler inserts a verification mark into codes. When executed, the target codes will be decompressed through FPGA. Before executed through CPU, the target codes will be decrypted and verified in lights of block mark information. Since the length of the basic block codes is short, a binary compression algorithm is designed. Finally, design and simulate encryption/decryption module of AES algorithm and decompression module of BCC algorithm through FPGA. Simulation results oscillogram shows that the algorithm function implemented through FPGA is correct and effective.

## Reference

- Christian, S. C., & Clark, D. T. (2002). Watermarking, tamper-proofing, and obfuscation - tools for software protection. *Software Engineering*, 28(8), 735-746. <http://dx.doi.org/10.1109/TSE.2002.1027797>
- Collberg, C., & Thomborson, C. (2002). Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection. *IEEE Transactions on Software Engineering*, 28(6), 735-746. <http://dx.doi.org/10.1109/TSE.2002.1027797>
- Fan, D., Li, Z., & Jiang, X. (2011). Code Optimization and Program Criterion Conversion in Certifying Compiler. *Micro computer system*, 7, 1400-1405.
- Gelbart, O., Narahari, B., & Simha, R. (2006). SPEE: A Secure Program Execution Environment Tool Using Static and Dynamic Code Verification. *Journal of High Speed Networks*, 15(1), 21-32.
- Gelbart, O., Ott, P., Narahari, B., Rahul Simha, Choudhary, A., & Zambreno, J. (2005). CODESSEAL: Compiler/FPGA Approach to Secure Applications. *Lecture Notes in Computer Science*, 3495, 530-535. [http://dx.doi.org/10.1007/11427995\\_54](http://dx.doi.org/10.1007/11427995_54)
- Han, X., & Wang, G. (2004). An Attack to Multisignature Schemes Based on Discrete Logarithm. *Chinese Journal of Computers*, 8, 1147-1152.
- Horne, B., Matheson, L., Sheehan, C., & Tarjan, R. (2002). Dynamic self-checking techniques for improved tamper resistance. In *Proc. 1st ACM Workshop on Digital Rights Management (DRM2001)*, pp. 141-159, Springer.
- Kiyomoto, & Shinsaku, A. (2008). A Security Protocol Compiler Generating C Source Codes. *International Conference on Information Security and Assurance, Busan, Korea*. pp. 20-26. <http://dx.doi.org/10.1109/ISA.2008.13>
- Lin, C., Chen, Y., Li, L., & Hua, B. (2007). Garbage collector verification for proof-carrying code. *Journal of Computer Science and Technology*, 22(3), 426-437. <http://dx.doi.org/10.1007/s11390-007-9049-z>
- Michael, H., & David, L. (2005). *Writing Secure Code*. Redmond, Washington, USA: Microsoft Press.
- Software Engineering Research and Practice. (SERP'05). (2005). June, Las Vegas.
- Sun, S., Lu, Z., & Niu, X. (2004). *Digital Watermarking Technology and Application* (pp. 36-37). Beijing: Science press.
- Wang, C. X. (2000). *A security architecture for survivability mechanisms*. PhD thesis, University of Virginia, Department of Computer Science.
- William, Z., Clark, D., Thomborson, & Wang, F. (2006). *Applications of Homomorphic Functions to Software Obfuscation*. WISI: pp. 152-153.
- Zhang, H., & Wang, Z. (2009). *Cryptography Introduction* (pp. 23-25). Wuhan University Press.