Секция 5

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

УДК 004.43, 004.056

CRYPTOGRAPHIC EXTENSION OF RUSSIAN PROGRAMMING LANGUAGE

G. P. Agibalov, V. B. Lipsky, I. A. Pankratova

An extension of the Russian programming language LYaPAS called LYaPAS-T is presented. The extension concerns the size of operands and the set of elementary operations over them. It is caused by the need of trustworthy and effective soft and hard implementations of contemporary cryptographic algorithms in secure computer systems applied for the logical control of critically important objects such as cosmic systems, nuclear weapons, energetic plants, submarines, etc. A LYaPAS-T compiler generating a load module for operating system Linux is presented too.

Keywords: *Russian programming language, cryptographic extension, LYaPAS-T, compiler.*

Introduction

Here by the Russian programming language is meant the algorithmic language LYaPAS elaborated at the beginning of the 1960th years at Tomsk State University (Russia) by the leadership of A. D. Zakrevskij and designed for the representation of logical combinatorial algorithms solving the problems of applied discrete mathematics appearing in the synthesis of discrete automata [1, 2]. The name of Russian programming language was given to it by American scientists [3]. Up to 1990th years, LYaPAS was applied in USSR [2], USA [4], Germany, Poland, Czechoslovakia and other countries. This time, LYaPAS is successfully reanimated at Tomsk State University by the Information Security and Cryptography Department especially for elaborating the trustworthy system and applied software destined for the computer-aided design of secure logical control computer systems and for the secure and effective implementation of cryptographic algorithms [5]. Among many programming languages known today, LYaPAS seems to be the most appropriate one for these purposes.

At the same time, there is an essential and perhaps single drawback of LYaPAS — the lack of many elementary operations that are widely used in contemporary cryptographic algorithms: for the long integer arithmetic, for calculating in many-dimensional spaces over finite fields and rings, for solving combinatorial problems on large sets, etc. By the way, this drawback is usual for all other programming languages including ones being much younger than LYaPAS. In some of them, the drawback is got over by writing classes of long integers, large discrete functions and others. As for LYaPAS, this its lack is more effectively got over by extending the language itself by spreading elementary operations in LYaPAS for logical complexes and by adding to it some new elementary operations defined for variables and logical complexes with the wanted purposes. The last version of LYaPAS known as LYaPAS-M [6, 7] slightly revised and then exTended in such a way is called LYaPAS-T.

The revision of LYaPAS-M concerns the symbol alphabet of the language and the arithmetic operations of multiplication and division for integers. The result of the revision is called vLYaPAS (from reVised LYaPAS). In it, small Latin letters are used instead of capital Russian ones, symbols of some operations are replaced by other more proper ones, and multiplication and division operations are defined saving the values of overflow and remainder respectively. For keeping them, a special variable — Z is inserted in the language.

The objective of the paper is to present an information about vLYaPAS, its extension LYaPAS-T motivated by the requirements of cryptographic algorithms, and a LYaPAS-T compiler generating code in the executive file format of the operating system (OS) Linux.

The project of a processor implementing LYaPAS-T programs in hardware, and a preprocessor translating LYaPAS-T programms to executive code for the processor are presented in [8], the program in vLYaPAS-T representing the cryptographic algorithm AES is demonstrated in [9].

1. Revised LYaPAS

A program in vLYaPAS is a series of sentences each starting with a pair §s where s is a non-negative integer called the number of the sentence. Every sentence is a sequence of operations applied to operands.

1.1. Operands

Operands in vLYaPAS are constants, variables, complexes and complex elements. They are used for representing non-negative integers, Boolean vectors, Unicode symbols and sequences of them. In vLYaPAS, non-negative integers are bounded by $2^{32} - 1$, the length of Boolean vectors does not exceed 32. Boolean vectors of length 32 are also called words. Components in a Boolean vector are numbered beginning with 0 in the direction from the right to the left end, and the vector itself can be considered as a non-negative integer (written in the binary form). A Boolean vector of any length $n \geqslant 1$ with only one component 1 is called a unit or identical vector.

There are natural, unit and symbol constants. Natural constants are written as decimal, hexadecimal, octal or binary numbers. A unit constant is a Boolean vector with only one component 1. It is denoted by Ii if the number of component 1 in it equals i. Symbol constant is a sequence of Unicode symbols.

Variables in vLYaPAS take values of Boolean vectors of length 32. The number of variables equals 27. They are denoted by letters a, b, ..., z, Z where Z is used for specialized purposes mentioned above. Besides, there is a virtual variable also called the own or internal variable in LYaPAS. Unlike the other variables, it is not written in LYaPAS programs, but it appears in them in implicit way as a result of any elementary operation and can be used as operand by any next operation in the program. For convenience of exposition, this variable is accepted to name τ.

Complex is a linearly ordered set of elements being symbols in a symbol complex or Boolean vectors of length 32 in a logical complex. Every complex has its own unique number from the series 0, 1, 2, ... The logical or symbol complex having number i is denoted by the symbol Li or Fi respectively. The real number and the greatest number of elements in a complex are the parameters of the complex and are called its cardinality and capacity respectively.

Unlike the other programming languages, the value types in vLYaPAS are not fixed. The value type (an integer or a vector) for a constant, variable and complex element is determined by the type of an operation (arithmetic or logical respectively) which is applied to this operand.

1.2. Operations

In vLYaPAS, there are elementary, complex, input-output and macro operations. Elementary operations are, in turn, of the following classes:

— value transfer: O — assigning 0s, $^-$ — assigning 1s, x — assigning x, $\Rightarrow$ — assigning τ, X — assigning an output value from a computer pseudorandom number generator (PRNG), $\Rightarrow$ X — assigning τ to to the seed of PRNG, T — assigning computer time, $=(x,y)$ — value swop;

— logical: $\neg$ — negation, $\vee$ — disjunction, & — conjunction, $\oplus$ — exclusive disjunction, $<$ — left shift, $>$ — right shift;

— arithmetic: ! — right 1 number calculation, % — Boolean vector weight calculation, $+$ — modulo 2^{32} addition, $-$ — modulo 2^{32} subtraction, $*$ — modulo 2^{32} multiplication (Z := overflow), : — double number division (Z := remainder), / — integer division quotient (Z := remainder), ; — integer division remainder (Z := quotient), $\triangle$ — increase by 1, $\bigtriangledown$ — decrease by 1;

— transition: $\rightarrow$ — unconditional transition, $\hookrightarrow$ — transition under condition $\tau = 0$, $\mapsto$ — transition under condition $\tau \neq 0$, $\rightarrow (x \diamond y)$ — transition under relation $\diamond \in \{=, \neq, <, >, \leqslant, \geqslant\}$, $\rightarrow$: — transition with return, $\rightarrow$! — return back, $\rightarrow (x)$ — transition in case if the running time exceeds x; $\rightarrow$Xabc — ones enumeration: if $a = 0$ then $b := 0$ and $\rightarrow c$, otherwise right 1 in a is replaced by 0, its number $\Rightarrow b$ and next operation;

— {assembly program} — assembly insertion (the assembly program pointed between { and } is executed).

Among complex operations there are forming (creating) a complex of a given cardinality, annihilation, decreasing complex capacity up to cardinality, complex clearance (without changing cardinality), addition of symbols to symbol complex, element insertion, element exclusion, subcomplex copying to another complex.

In future a complex created with a constant or variable capacity will be called respectively static or dynamic one.

Note that the operation of taking subcomplex existing in LYaPAS is not included in vLYaPAS because of its potential insecurity.

Input-output operations are the following: /F>C — output of symbol complex F to console; $/'\varsigma'$ >C — output of symbol constant ς to console; /F<C — symbol constant input from keyboard: symbols are added to a symbol complex F, complex cardinality is increased.

Macro operations are calls for subprograms with the given external (input/output) parameters.

2. LYaPAS extension

2.1. Long arithmetics

Natural constants in LYaPAS-T are integers $0, 1, \ldots, 2^n - 1$ where n is a multiple of 32 and depends on actual implementation of LYaPAS-T. Nowadays the value $n = 2^{14}$ seems to be quite sufficient for contemporary cryptographic applications.

By letting δ be 2^{32} a natural constant c may be expressed by the following series:

$$c = c_0 + c_1\delta + c_2\delta^2 + \ldots + c_{r-1}\delta^{r-1} \qquad (1)$$

for some $r > 0$ and $c_i \in \Omega = \{0, 1, \ldots, 2^{32} - 1\}$, $i = 0, 1, \ldots, r-1$. In their standard binary representation, the elements of the set Ω are Boolean vectors of length 32. Therefore in

LYaPAS-T, the sequence $c_0, c_1, \ldots, c_{r-1}$ is represented by a logical complex Lj of length r with c_i being the ith element of Lj.

All operations defined in vLYaPAS for variables can be used in LYaPAS-T for logical complexes. In case of arithmetic operation the sequence of complex elements is considered as a natural constant c expressed by the series in (1). Different operands for an arithmetic operation may be of different lengths and types (one of them — a variable, another — a complex). In case of logical operation the complex value is considered as a Boolean vector being the concatenation of the complex elements. Logical complexes of cardinality $n/32$ with values being unit vectors are unit constants Ii, $i = 0, 1, \ldots, n-1$, in LYaPAS-T.

2.2. Plurality of own variable

So, unlike LYaPAS, there are two types of operands for elementary operations in LYaPAS-T: variables of the length of one word and logical complexes of different lengths — from 1 to $n/32$ words. Accordingly, in LYaPAS-T, there are two types of the own variable — prime and complex. The first one is the traditional τ from LYaPAS. It has the length of one word and may take the values of any variable of the language. In any implementation of LYaPAS-T, soft or hard, it is kept in a processor register. The own variables of the 2nd type take values of logical complexes and have their lengths. In hard implementation of LYaPAS-T, each of them is kept in one and the same register of the maximal possible length — n. In soft implementation of LYaPAS-T, to exclude time-spending operations for transferring complex between a register and data memory, it is expediently, for the time of executing a series of operations beginning with the address to a logical complex, to pass the role of the complex own variable directly to this complex and to keep it in data memory where the complex is kept.

2.3. Additional operations

In addition to operations in vLYaPAS, the extension LYaPAS-T contains some new logical operations used in cryptographic algorithms including the following ones:

1) $\Updownarrow$L — permutation: components of Boolean vector τ are re-arranged according to the order of their serial numbers pointed in a logical complex L;

2) $\_(a, b)$ — projection: the part of Boolean vector τ with components having numbers in an interval (a, b) is chosen;

3) $\Uparrow b(i)$ — insertion: a Boolean vector b is put in τ before the ith element;

4) $\Downarrow (a, b)$ — reduction: the part of Boolean vector τ with components having numbers in an interval (a, b) is deleted from the vector;

5) $\mid b$ — concatenation: a Boolean vector b is added to τ;

6) $\ll i$ or $\gg i$ — left or right cyclic shift: Boolean vector τ is cyclically shifted i bits left or right respectively;

7) $\geqslant \varkappa(a)$ — most element: a maximal element is found in a complex $\varkappa$ whose elements are considered as non-negative integers, its value is given to τ, its number — to a;

8) $\leqslant \varkappa(a)$ — least element: analogously.

As for arithmetic operations modulo N (for any natural N) such as addition $(\mathrm{mod}\, N)$, multiplication $(\mathrm{mod}\, N)$, exponentiation $(\mathrm{mod}\, N)$ and others which are widely used in cryptography, there is no real possibility to include them in the list of the elementary operations in LYaPAS-T because of the existence of many algorithms implementing them with the different efficiencies in many cases. Instead, it is decided to implement these algorithms in an assembly language and (or) in LYaPAS-T and to include them in the LYaPAS-T library for applying them by users in LYaPAS-T programs as subprograms.

3. LYaPAS-T compiler

3.1. What is this?

For short, any LYaPAS-T program and its subprograms are called L-program and L-subprograms respectively. For being implemented by a computer, an L-program should be used as a parameter of LYaPAS-T compiler that converts it to a load module for the OS Linux.

The compiler starts under the command line of OS Linux

```
>$ lc <prog>.l
```

where `<prog>.l` is the name of a file with the L-program being a list of L-subprograms. (It is recommended to call an L-program file using the extension l but it is not necessarily.) The first L-subprogram in the list is the head one. OS Linux transfers the control to it after the file loading to RAM. The order of the other L-subprograms in the list does not matter. The file may contain not all necessary L-subprograms. In this case the compiler looks for them in the file `libl0.l` being the user library. It is the library where it is convenient to keep the often used L-subprograms.

The result of the compiler working is a load module which is kept with the name `<prog>` (without extension) at the same folder where the L-program is located. For executing it the following command is used:

```
>$ ./<prog>
```

Compiler is written in C++ with the using the library of regular expressions making it absolutely simple and transparent.

3.2. Load module structure

The load module consists of two segments — program segment `(.text)` and data segment `(.data)`. In turn, the first one consists of subprograms generated by the compiler for L-subprograms called in the process of the L-program execution. The data segment contains: the current address in the memory for placing new complexes, and the bound of memory granted for complexes by OS; the current state of the PRNG; unit constants; the weights of all Boolean vectors in $\{0,1\}^8$ (need for the implementation of the operation %); and all symbol constants met in the L-program.

3.3. Memory organization

For every L-subprogram, all its local variables, beginnings, cardinalities and capacities of its local complexes are placed in a stack forming a frame of 1420 bytes. An access to the local data is made by using fixed shifts from the frame beginning (register `ebp`). For the frame of a parental L-subprogram, the value of the register `ebp` is also saved in the frame. So, the list of frames is created for the L-subprograms called.

The local complexes of an L-subprogram are placed in a heap. The address of the free section in the heap at the moment of calling for the L-subprogram is also kept in the frame.

The operation of creating local complex is accompanied with the control of the free section of a necessary size. It is done by comparing the values of the complex capacity, address of the free area, and the memory bound granted by OS for complexes. In the case of enough place, the address of the complex beginning takes the value of the free section address, and the free section address is increased by the value of the complex capacity. If the place is not enough then an appeal to OS is made for increasing the accessible memory bound.

Under this organization, the memory is protected against attacks through the stack or heap overflow because, first, buffers (complexes) are taken away from the stack, and there is no possibility to rewrite the return address, and, second, there are no operations for setting memory free by means of OS.

BIBLIOGRAPHY

1. LYaPAS, a Programming Language for Logic and Coding Algorithms / eds. M. A. Gavrilov and A. D. Zakrevskii. New York, London: Academic Press, 1969. 475 p.
2. *Toropov N. R.* Programming language LYaPAS // Applied Discrete Mathematics. 2009. No. 2(4). P. 9–25. (in Russian).
3. *Nadler N.* User group for Russian programming language // IEEE, Newsletter for Computer-Aided Design. 1971. Iss. 3.
4. *Charles J. and Albright Jr.* An Interpreter for the Language LYaPAS. University of North Carolina at Chapel Hill: Department of Computer Science, 1974. 125 p.
5. *Agibalov G. P.* To reanimation of Russian programming language // Applied Discrete Mathematics. 2012. No. 3(17). P. 77–84. (in Russian).
6. *Zakrevskij A. D. and Toropov N. R.* Programming system LYaPAS-M. Minsk: Nauka i Technika, 1978. 240 p.(in Russian).
7. *Toropov N. R.* Dialogue programming system LES. Minsk: Nauka i Technika, 1985. 263 p. (in Russian).
8. *Agibalov G. P., Lipsky V. B., and Pankratova I. A.* Project of hardware implementation of Russian programming language // Applied Discrete Mathematics. Application. 2013. No. 6. P. 98–102.
9. *Broslavskiy O. V.* AES in LYaPAS // Applied Discrete Mathematics. Application. 2013. No. 6. P. 102–104.

УДК 004.43, 004.056

PROJECT OF HARDWARE IMPLEMENTATION OF RUSSIAN PROGRAMMING LANGUAGE

G. P. Agibalov, V. B. Lipsky, I. A. Pankratova

The projects of a LYaPAS-T processor implementing the programming language LYaPAS-T in hardware and of a preprocessor translating LYaPAS-T programs into the executive code of the processor are presented. It is also told that for a LYaPAS-T subset containing neither subprograms, nor operations over complexes and long operands, the architecture of the processor was described in VHDL, tested by means of a computer simulation, and implemented in a programmable logical integrated circuit obtained with the help of a computer-aided design.

Keywords: *Russian programming language, LYaPAS-T, hardware implementation, LYaPAS-T preprocessor.*

In [1] a cryptographic extension LYaPAS-T of Russian programming language and a compiler for its implementation in software were presented. The objective of this paper is to present an information about the project of a processor implementing LYaPAS-T programs in hardware, and a preprocessor translating LYaPAS-T programs to executive code for the processor.

1. Parameters

In LYaPAS-T implemented in hardware, the operand length, the largest number of a complex and the maximal quantity of subprograms in the hierarchical structure of a program are assumed to be bounded by natural n, m and k respectively. Hence, the quantities of