Jan Długosz University of Częstochowa
Scientific Issues, Mathematics XI, Częstochowa 2006

On Some Crypto-Messages Parser*

Mirosław Kurkowski, Jacek Małek,
Maciej Orzechowski

Institute of Mathematics and Computer Science
Jan Długosz University of Częstochowa
al. Armii Krajowej 13/15, 42-200 Częstochowa, Poland
e-mail: m.kurkowski@ajd.czest.pl
j.malek@ajd.czest.pl
maciej@orzechowski.info

Abstract
Some methods of automatic verification of cryptographic protocols require creating specially designed formal languages based on suitable algebra of terms (called *crypto-terms* or *crypto-messages*). Sometimes in verification process it is essential to check whether a given crypto-term is a subterm of another one or in general whether some crypto-term belongs to the specified crypto-language (a set of some crypto-terms) or not. Another problem is creating the set of all antichains of subterms of a given term with respect to the order introduced by a subterm relation.

The main goal of this paper is to present a tool for verification whether a crypto-term belongs to the specified crypto-language or not. Our main purpose is to check lexical analysis and syntax analysis. Finally, after checking that a crypto-sentence is correct, we convert it into the special format,

*Extended version of a talk presented at the X Conference "Applications of Algebra in Logic and Computer Science", Zakopane, March 6–12, 2006. This research is partially supported by the Ministry of Science and Information Society Technologies under the grant number 3 T11C 011 28.

which can be transfomed to an atomic version of this sentence, and make an antichain. We can introduce formulas manually or as text files. The output may also be changed into a text file. The program is a text tool which is started in a Linux environment.

Keywords: Cryptographic authentication protocols, verification, grammar parsing.

1. Introduction

Cryptographic protocols are specialized tools to achieve authentication in large distributed computer network. These protocols are precisely defined sequences of actions (communication and computation steps) which use some cryptographic mechanism such as encryption and decryption.

In [2] a new method of verification of these protocols has been intro-
duced[1]. The main idea of this method is to create a suitable mathematical space of many different paralell executions of a given protocol. In this paper the authors proposed a specially designed algebra of terms, which can model messages sent during executions by participants who execute the protocol. Some constructions done in [2] require checking whether a given crypto-term is a subterm of another one or in general whether some crypto-term belongs to the specified crypto-language or not. Another goal, which is necessary in verification process, is to create the maximal set of all, in some sense independent, antichains of subterms of a given term. These antichains are created with respect to the order introduced by a subterm relation.

A rest of this paper is organized as follows: In Section 2 we give some basic definitions due to paper [2], in general the definition of investigated term's algebra. In the next section we present some informations about the tools used for checking (Bison and Flex) which we employ in our tool. Some examples are shown at the end of Section 2 and in the Section 3. In the last section we propose some remarks for future investigations.

[1] This method has been announced in [1].

2. Syntax

In this section we introduce syntax for dealing with untimed cryptographic protocols.

Let:

- $\mathcal{T}_P = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{n_P}\}$ be a set of symbols representing the participants of the computer network,

  $\mathcal{T}_I = \{\mathcal{I}_{\mathcal{P}_1}, \mathcal{I}_{\mathcal{P}_2}, \ldots, \mathcal{I}_{\mathcal{P}_{n_P}}\}$ be a set of symbols representing the identifiers of the participants,
- $\mathcal{T}_K = \bigcup_{i=1}^{n_P}\{\mathcal{K}_{\mathcal{P}_i}, \mathcal{K}_{\mathcal{P}_i}^{-1}\}$ be a set of symbols representing the cryptographic keys of the participants (public and private respectively),
- $\mathcal{T}_N = \bigcup_{i=1}^{n_P}\{\mathfrak{N}_{\mathcal{P}_i}^1, \ldots, \mathfrak{N}_{\mathcal{P}_i}^{n_N}\}$ [2] be a set of symbols representing *nonces* of participants,
- $\{"(", ")", "\langle", "\rangle", ","\}$ be a set of auxiliary symbols.

Definition. By a set of *crypto terms* $\mathcal{T}$ we mean the smallest set satisfying the following conditions:

[1] $\mathcal{T}_P \cup \mathcal{T}_I \cup \mathcal{T}_K \cup \mathcal{T}_N \subseteq \mathcal{T}$.

[2] If $X \in \mathcal{T}$ and $Y \in \mathcal{T}$, then the sequence $X \cdot Y \in \mathcal{T}$.

[3] If $X \in \mathcal{T}$ and $\mathcal{K} \in \mathcal{T}_{\mathcal{K}}$, then $\langle X \rangle_{\mathcal{K}} \in \mathcal{T}$.[3]

Next, we define some useful relations over the set $\mathcal{T}$.

Definition Let $\prec_{\mathcal{T}} \subseteq \mathcal{T} \times \mathcal{T}$ be the smallest relation which satisfies the following conditions:

[1] If $X, Y \in \mathcal{T}$, then $X \prec_{\mathcal{T}} X \cdot Y$ and $Y \prec_{\mathcal{T}} X \cdot Y$.

[2] If $X \in \mathcal{T}$ and $\mathcal{K} \in \mathcal{T}_{\mathcal{K}}$, then $X \prec_{\mathcal{T}} \langle X \rangle_{\mathcal{K}}$ and $\mathcal{K} \prec_{\mathcal{T}} \langle X \rangle_{\mathcal{K}}$.

By $\preceq_{\mathcal{T}}$ we mean the transitive and reflexive closure of $\prec_{\mathcal{T}}$. Next, for any $\mathcal{X} \subseteq \mathcal{T}$ we define a sequence of the sets $(\mathcal{X}^n)_{n \in \mathbf{N}}$ that are subsets of $\mathcal{T}$.

[2]We assume that n_P and n_N are some fixed natural numbers.

[3] $\langle X \rangle_{\mathcal{K}}$ is a term that is interpreted as a ciphertext containing the letter term X encrypted with the key $\mathcal{K}$.

- $\mathcal{X}^0 \overset{def}{=} \mathcal{X}$,
- $\mathcal{X}^{n+1} \overset{def}{=} \mathcal{X}^n \cup$
  $\cup \{Z \in \mathcal{T} \mid Z = X \cdot Y \vee Z = \langle X \rangle_{\mathcal{K}}$, for some $X, Y \in \mathcal{X}^n,\ \mathcal{K} \in \mathcal{X} \cap \mathcal{T}_{\mathcal{K}}\}$.

Intuitively, the set $\mathcal{X}^{n+1}$ corresponds to letter terms gradually built from $\mathcal{X}^n$ via the operations of composition and encryption.
In what follows, for any set Z we denote a set of all the finite subsets of Z by 2^Z_{fin}. For $\mathcal{X} \in 2^{\mathcal{T}}_{fin}$ let $h_{\preceq_{\mathcal{T}}}(\mathcal{X}) \overset{def}{=} \bigcup_{n \in \mathbf{N}} \mathcal{X}^n$ be a set composed of all the letter terms that can be constructed from elements of $\mathcal{X}$ only.[4] From the algebraical ponit of view the set $h_{\preceq_{\mathcal{T}}}(\mathcal{X})$ is the set of all terms which have some subterm in $h_{\preceq_{\mathcal{T}}}(\mathcal{X})$. Additionally, let $\eta_{\preceq_{\mathcal{T}}}(\{X\})$ be a set of all subterms of the term X.

Definition. Let $\boldsymbol{A}$ be a set of terms and X be a term. $\boldsymbol{A}$ is said to be a *maximal antichain* of subterms of X if the following properties hold:

[1] $(\forall Y \in \boldsymbol{A})(Y \preceq_{\mathcal{T}} X)$.

[2] $(\forall Y, Z \in \boldsymbol{A})\ \neg(Y \preceq_{\mathcal{T}} Z)$.

[3] $X \in h_{\preceq_{\mathcal{T}}}(\boldsymbol{A})$.

Definition. Let $X \in \mathcal{T}$ be a crypto-term and $\boldsymbol{G}$ be a subset of $\mathcal{T}$. $\boldsymbol{G}$ is said to be a set of *(independent) generators* of X (denoted by $\boldsymbol{G} \vdash X$) if the following conditions are met:

[1] $\boldsymbol{G} \subseteq \eta_{\preceq_{\mathcal{T}}}(\{X\})$,

[2] $X \in h_{\preceq_{\mathcal{T}}}(\boldsymbol{G})$,

[3] $(\forall Y \in \boldsymbol{G})(Y \notin h_{\preceq_{\mathcal{T}}}(\boldsymbol{G} \setminus \{Y\})$,

[4] $(\forall Y \in \boldsymbol{G})(X \notin h_{\preceq_{\mathcal{T}}}(\boldsymbol{G} \setminus \{Y\})$.

Intuitively, we have $\boldsymbol{G} \vdash X$ if all the elements of $\boldsymbol{G}$ are subterms of X, X can be composed out of the elements of $\boldsymbol{G}$, and $\boldsymbol{G}$ is such a minimal set.

[4]Decription is not allowed here.

Proposition. It is easy to observe that every maximal antichain of a given crypto-term X is a set of independent generators of X.

Example 1. Consider the crypto-term $L = \langle I_A, N_A \rangle_{K_B}$. Observe that the sets $\boldsymbol{G}_1 = \{I_A, N_A, K_B\}$ and $\boldsymbol{G}_2 = \{\langle I_A, N_A \rangle_{K_B}\}$ are sets of independent generators of L, i.e. we have $\boldsymbol{G}_1 \vdash L$ and $\boldsymbol{G}_2 \vdash L$.

3. Flex and Bison

Flex is a fast lexical analyser generator. It is a tool for generating scanners: programs which recognizes lexical patterns in input text. Flex is a free implementation of the well known Lex program. The description is in the form of pairs of regular expressions and C code called rules. The best choice is to link it with another great tool Bison. Lexical analyser created by Lex co-operates with the syntax analyser in the following way. After starting by the syntax analyser, the lexical analyser begins to read in the next symbols from the input, till it finds the longest file-prefix of input suitable for one of the models. Afterwards it does action which typically directs steering back to the syntax analyser. In a case of reading in the white symbols or comments the lexical analyser does not transfer any values to the syntax analyser, but returns to finding out the next lexem. The lexical analyser, as a result of starting, transfers a singular value, which is a lexical symbol, to the syntax analyser. For transferring values of an attribute consisting information of lexem, global variable `yylval` is being used.

Bison is a tool parser generator. It is a tool for converting a grammar description for an LALR context-free grammar into a C program to parse that grammar. It can specify one or more syntactic groupings and give rules for constructing them from their parts. The input to Bison is essentially machine-readable BNF (Backus-Naur Form). Parsers generated by Bison use in their work the auxiliary stack. The state of calculation made by parser is represented by the parser configuration. Such configuration is an ordered pair of sequences - stack and input. Stack element describes all symbols presented on the stack, and input consists of input symbols still not read. The most important actions of parser are shift and reduce with following meaning:

- shift - moves current terminal symbol to the top of the stack of parser,

- reduce - changes alfa production into A terminal

Antichain Parser tool description. The tool is combined of 3 parts. The first task is to check the lexical correctness and return to the next part of the recognized token. At the Flex level it is checked whether the adequate words at the input are from the set of the specified language or not. The second task rests with Bison, where proper grammar rules were defined which have to check the syntax correctness of the specified language.

At this step we build a special table which is genereted in the following code:

```
P : A NUM {sprintf(buf, "a%s", $2); $$ = strdup(buf);}
```

With every rule an operation, which aim is to build properly a structure within a table and to place it in a proper place of an output structure, is connected. With every rule an adequate action, which is suited by the longest match base, is related.

```
MM : P {$$=$1;}
   | NN {$$=$1;}
   | KK {$$=$1;}
   | LL {$$=$1;}
   ;
LL : I P {sprintf(buf, "i%s", $2); $$=strdup(buf);}
   ;
```

The last step is the adequate writing out the structure in the special format.

1. Input - standard input or text file (parameters given by the user).
2. Flex - lexical analyser generating tokens from input data.
3. Bison - parser grammar checking syntax analysis based on Flex tokens and rule.
4. C code building a tree using internal Bison stack.
5. C code generating output data.
6. Output - standard output or text file.

The tool for verification whether a formula belongs to the specified language cooperates with the second tool written in Python which executes C parser, puts a formula on standard input and generates all parts of a given cryptogram. We use Python in our programming process because Python provides us with a good environment for quickly developing an initial prototype. This allows us to get the overall program structure and logic right, and we can fine-tune small details in the fast development cycle that Python provides. Once the user is satisfied with program output, he can translate the Python code into C++, Fortran, Java or some other compiled language.

Example 2.

Consider the following step of some protocol:

```
a1 > a2 :  < <na1_1|ia>ka2 | <na2_1|ia2> ka1 > ka2
```

The output obtained with our tool is the following:

```
exec.a=1
exec.b=2
exec.s.n[1]=1
exec.s.i[1]=1
exec.s.k[2]=1
exec.s.n[2]=1
exec.s.i[2]=1
exec.s.i[2]=1
exec.s.k[1]=1
exec.s.k[2]=1
exec.gen.n[1]=1
exec.gen.n[2]=1
exec.sz.sz[0].co.n[1]=1
exec.sz.sz[0].co.i[1]=1
exec.sz.sz[0].k=2
exec.sz.sz[1].co.n[2]=1
exec.sz.sz[1].co.i[2]=1
exec.sz.sz[1].k=1
exec.sz.k=2
```

4. Conclusion

In this paper we have presented a new tool for verification whether a crypto-term belongs to the specified crypto-language. Our main purpose was to check lexical analysis and syntax analysis. For our investigation Flex and Bison – the well known grammar tools – have been used.

We believe that our tool will be usefull in research on the topic of symbolic verification of cryptographic protocols.

References

[1] M. Kurkowski. *Dedukcyjne metody weryfikacji poprawności protokołów uwierzytelniania, Deduction methods of verification of authentication protocols.* PhD dissertation, Institute of Computer Science of The Polish Academy of Sciences, Warsaw, Poland 2003.

[2] M. Kurkowski, W. Penczek. *Verifying cryptographic protocols modeled by networks of automata.* In: Proceedings of XV CS&P (Concurrency, Specification and Programming), Humboldt University Press, Berlin, pp. 292-303, 2006.