International Journal of Security and Its Applications
Vol. 10, No. 11 (2016), pp.211-228
http://dx.doi.org/10.14257/ijsia.2016.10.11.19

PV2JAVA: Automatic Generator of Security Protocol Implementations Written in Java Language from the Applied PI Calculus Proved in the Symbolic Model

Bo Meng[1], Yitong Yang[2], Jinli Zhang [3], Jintian Lu [4] and Dejun Wang[5*]

[1-5]*School of Computer, South-Central University for Nationalities*
MinYuan Road #182, Wuhan, Hubei, China, 430074
[1]*mengscuec@gmail.com*, [2]*yyt91@sina.cn*, [3]*jinlysa@163.com*, [4]*ljt45@hotmail.com*,
[5]*wangdj@whu.edu.cn*

Abstract

In order to get the security protocol implementations written in programming language from formal languages in secure way, firstly, the model of implementation generation from security protocol implementations written in formal language is presented; Apart from that, an automatic generator PV2JAVA is developed, which can transform security protocol implementations written in the Applied PI calculus proved in the symbolic model into security protocol implementations written in Java language ; Finally, the method of software testing is used to provide a strong confidence in the correctness of the automatic generator PV2JAVA through five typical security protocols.

Keywords*: implementation generation; security protocol; software security*

1. Introduction

Over the last years people including developers and users have done a lots of works on the formal analysis and verification of security protocol implementations written in formal languages. While we know that the extreme objective is to have security protocol implementations written in programming language, for example, Java language, to put it into practice in the information system. Hence we need to research the methods of generating security protocol implementations written in programming language because for the developers and users, just proof of securities of security protocol implementations written in formal language is not enough to give a strong confidence on its security properties. Although we proved the securities of security protocol implementations written in formal language, security protocol implementations written in programming language may be error and insecure. So it is important to analyze and prove security properties of security protocol implementations written in programming languages.

Generally, there are two methods to get secure security protocol implementations written in programming language. One is model extraction which is suitable to the legacy codes of security protocols. The method also can be divided into two types. One type mainly bases on the technologies of program analysis and verification, for example, logic, and type theory, which is directly used to automatically verify its cryptographic security. At the same time it is also depends on adding a lot of annotations and predicates/assertions to security protocol implementations written in formal language. The other type is model extraction which firstly extracts security protocol written in formal language from security protocol implementations written in programming language, then uses security protocol analyzer to analyze or prove the cryptographic security of security protocol implementations written in formal language. This method is difficult to be implemented correctly.

The other method is implementation generation which suits no-existing

ISSN: 1738-9976 IJSIA
Copyright ⓒ 2016 SERSC

implementations written in programming language for some security protocols. If we have gotten the security protocol implementations written in formal language and verified its security, then we can use the implementation generation method to get the secure security protocol implementations written in programming language. Implementation generation from security protocol written in formal language is that firstly the security protocol implementations in formal language, for example, the Applied PI calculus, are produced, then we use security protocol analyzer/prover, for example, ProVerif, to analyze the securities of security protocol implementations written in formal language, finally we use or develop the transformer to translate security protocol implementations written in formal language into security protocol implementations written in programming language.

About implementation generation, to our best knowledge, there does not exist the automatic generator of security protocol implementations in Java language from security protocol implementations written in the Applied PI proved in the symbolic model. Hence we research it. And the main contributions of this study are summarized in detail:

❁ The up-to-date automatic generation of security protocol implementations written in programming language is presented. We find that there does not exist the automatic generator of security protocol implementations in Java language from security protocol implementations written in the Applied PI proved in the symbolic model.

❁ Present a model of implementation generation in which the security protocol implementations FP [SP] written in formal language are translated to security protocol implementations PP[SP] written in programming language. It is also need to prove that for the adversary Adv[FP] based on the any adversary Adv[PP], if the security protocol implementations FP [SP] written in formal language is secure, then the security protocol implementations PP[SP] written in programming language is also secure for any adversary Adv[PP].

❁ Develop an automatic generator PV2JAVA which transforms security protocol implementations written in the Applied PI calculus that is inputs of ProVerif to the security protocol implementations written in Java language.

❁ Use the method of software testing to provide a strong confidence in the correctness of the automatic generator PV2JAVA. We use the automatic generator PV2JAVA and ProVerif to produce the Identity Federation security protocol based on SAML (http://saml.xml.org/saml-specifications), OAuth2.0, (http://tools.ietf.org/html/rfc674) improved OAuth2.0 [8], SSHV2 (http://www.snailbook.com/docs/userauth.txt), TLS (http://www.ietf.org/rfc/rfc5246.txt) security protocol implementations written in Java language from the their implementations written in the Applied PI calculus proved in the symbolic model. The security analysis results in the Applied PI calculus are compared to the security analysis results in JAVA language. The comparisons are successful. Hence the correctness of the automatic generator PV2JAVA is provided in a degree.

2. Related Work

The up-to-date mechanized generation of security protocol implementations written in programming language from security protocol implementations written in formal language proved in computational model and in symbolic model is presented.

In symbolic model, Pironti and Sisto [1] develop a tool Spi2Java which translates security protocol implementations written in typed Spi calculus to security protocol implementations based on some special libraries written in Java language. Apart from that, if the special libraries have met some requirements, then the generated Java implementation correctly simulates the typed Spi calculus specification. Following the work of Pironti and Sisto [1], Avalle *et al*. [2] design a framework called JavaSPI in which Java language is not only as a modeling language but also as the implementation language in security protocols. In essence, it translates security protocol implementations written in Java language to the inputs which is analyzed by ProVerif. Backes *et al*. [3] introduce a

 Copyright ⓒ 2016 SERSC

new implementation generator Expi2Java which translates security protocol implementations written in an extensible variant of the Spi calculus into security protocol implementations written in Java language which has the features of concurrency, synchronization between threads, exception handling and a type system. At the same time they formalize the translation algorithm of Expi2Java with the Coq proof assistant, and proved that if the original models are well-typed, then the generated programs are also well-typed. Li *et al*. [4] propose a multi-objective-language-oriented automatic implementation generation scheme for security protocols based on XML which describes the formal model of security protocols.

In computation model Cade and Blanchet [5,6] develop a compiler that takes a abstract specification of the protocol as the input language of the protocol verifier CryptoVerif and translates it into an OCaml implementation. They also prove that this compiler preserves the security properties proved by CryptoVerif and if the abstract protocol specification is proved secure in the computational model by CryptoVerif, Li *et al*.[7]develops an secure.an automatic verifier SubJAVA2CV is developed which is able to transform security protocol implementations written in SubJAVA language to security protocol implementations written in Blanchet calculus in the computational model.

According to the above related reference we can find, until now there does not existing that the automatic generator of security protocol implementations written in Java language from security protocol implementations written in the Applied PI proved in the symbolic model.

3. The Implementation Model from Security Protocol Implementations Written in Formal Language to Programming Language

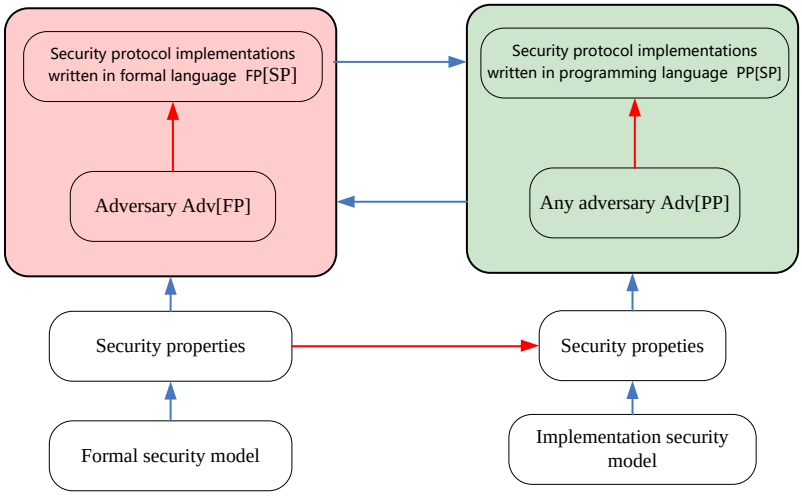

Figure 1. Implementation Model from Security Protocol Implementations Written in Formal Language to Programming Language

For the implementation model showed in "Figure 1", security protocol implementations FP [SP] written in formal languages, such as Pi calculus, Applied calculus, are generated firstly. Then security protocol implementations PP [SP] written in programming languages, for example, Java, Swift, C language, are produced. In other words the source languages are formal languages and the target languages are programming languages. If security protocol implementations written in programming language are generated from security protocol implementations written in formal language are secure, there are two conditions to be satisfied:

①. The semantic of formal language should simulate the semantic of

programming language.

②. According to the any adversary Adv[FP], we construct the adversary Adv[PP]. If the security protocol implementations FP [SP] written in formal language is secure, for any adversary Adv[PP], the security protocol implementations PP[SP] written in programming language is also secure .

The condition ① presents that the relationship is simulation or observational equivalence between the security protocol implementations PP [SP] written in programming language and the security protocol implementations FP[SP] written in formal language.

The condition ② presents that how to prove the security properties of the security protocol implementations PP[SP] written in programming language. For any adversaries in the context, our objective is to prove the security properties of the security protocol implementations PP[SP] written in programming language. Hence an adversary Adv [FP] in the security protocol implementations FP [SP] written in formal language is constructed based on the any adversary Adv[PP] in the security protocol implementations PP[SP] written in programming language. And then the conclusion is proved, if the security protocol implementations FP [SP] written in formal language is secure for the adversary Adv [FP], the security protocol implementations FP[SP] written in programming language is secure for any adversary Adv[PP].

4. The Mappings from Security Protocol Implementations Written in the Applied Pi Calculus to Java Language

Security protocol implementations written in Applied PI calculus is mainly composed of processes. Processes are mainly made up top process and communicating party processes. The top process is main process. The parties in security protocols are formalized as communicating party processes which consists of sender process and receiver process in template for security protocol implementations written in Applied Pi calculus.

The objective of the template based on the Applied calculus is used to analyze security protocols through ProVerif and is composed of the following sections: function declaration, equational theory, channel declaration, security properties, sender process, receiver process and top process. Function declaration is used to declare the functions used in formalizing security protocols. Channels are used to formalize the communication channels between the sender process and receiver process and is declared in channel declaration section. In the template of model of security protocols we assume that there are two roles. One is sender which is modeled as the sender process in Sender process section and the other is receiver that is modeled as the receiver process in Receiver process section. Top process is made up of the sender process and the receiver process.

Hence in the mappings in implementation generator showed in "Figure 3", the process is mapped into the main program in security protocol implementations written in Java which is a class. The names in the Applied PI calculus is mapped to the statement string new in Java language. About the type, as we know there is no type in the Applied PI calculus. At the same time there are a lot of message exchanged between the communicating parties. Hence we specify that the type of the variable in the Applied PI calculus as string class in Java language. The function in the Applied PI calculus is translated into the methods in the class in Java language. The special functions, for example, cryptographic primitive, in the Applied PI calculus is transformed into the security package in Java language. The communicating channel process which include input channels and output channels are mapped into the classes that have the responsibilities of sending and receiving messages between the communicating parties. The sender process and the receiver process are mapped into the sender class and receiver class, respectively in "Figure 2".

 Copyright ⓒ 2016 SERSC

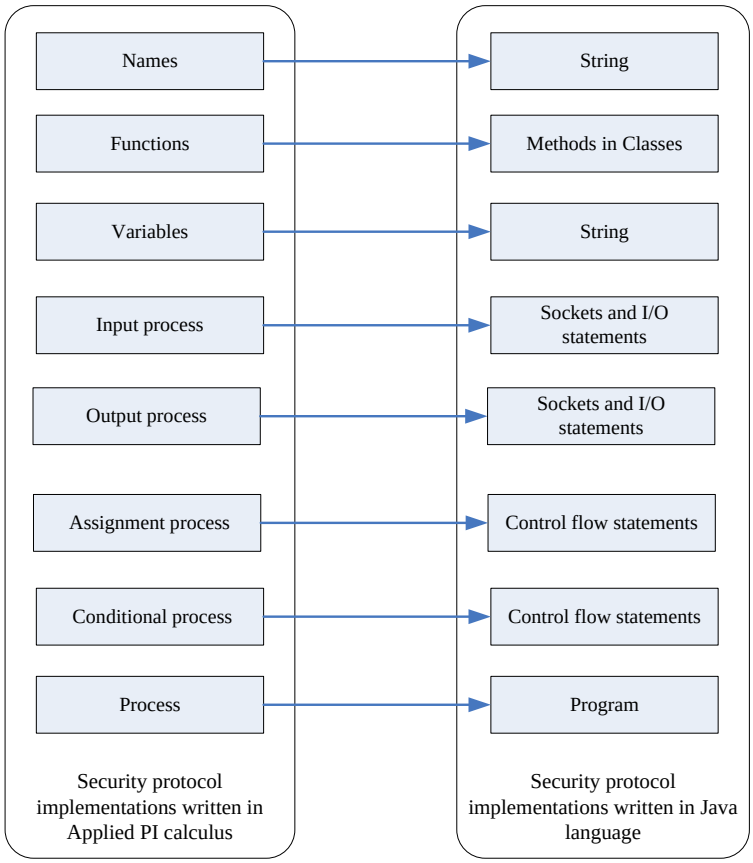

Figure 2. The Mappings from Security Protocol Implementations Written in the Applied PI Calculus to Java Language

❁ Names

Statement free a,b,c declares the free names a,b,c in the Applied PI calculus. When a name occurs free in the process,. the adversary knows the name. The BNF of statement free a,b,c is $free\ seq\langle ident\rangle$. The restriction new a;P in the Applied PI calculus creates a new name a, then executes P. The BNF of statement new a is $new\ ident$. According to the operational semantics relationships, we map the statements free and new to the statement string new in Java language in "Figure 3" and "Figure 4".

$$free\ seq\langle ident\rangle \Rightarrow \begin{Bmatrix} String\ ident1 = new\ String(); \\ String\ ident1 = new\ String(); \\ ... \\ String\ identn = new\ String(); \end{Bmatrix}$$

Figure 3. The Mapping from Statement free in the Applied PI Calculus to Statements in Java Language

$$new\ ident \Rightarrow String\ ident = new\ String()$$

Figure 4. The Mapping from Statement new in the Applied PI Calculus to Statements in Java Language

Copyright ⓒ 2016 SERSC

❁ Facts

In the Applied PI calculus, the operators is mainly composed of assignment, equality, relational, and Conditional operators and does not include the type transformation, bit operation and so on. Hence the operators in the Applied PI operators mostly can be directly translated into the operators in Java language. But there is a issue that we should notice. Because in the assignment expression in the Applied calculus, the right side of the assignment operator may be exist several terms. When we deal with the special assignment statements, for example, s= (a, b), it should be translated into String [] s= {a, b} in Java language. Operator ==> in the Applied PI calculus is used in query statement. The correspondent operator in Java language does not exist. The mapping from operators in the Applied PI calculus to Java statements is showed in "Table 1".

Table 1. The Mappings from Operators in the Applied PI Calculus to Statements in Java Language

Applied PI calculus	Java language	Operators
=	=	Assignment
<>	!=	Not equal to
=	==	Equal to
|	|	Conditional OR
&	&	Conditional AND
==>		Query agreement

❁ Input process

In the Applied PI calculus the input process in(c,p);P inputs a message on channel c, and executes P after matching the input message with p, and binding the variables contained in p. When a message does not match p, it cannot be input by this construct. The channel c can be any term. The BNF of the input process is $in(\langle term\rangle,\langle pattern\rangle)$. In Java language there are many classes and API that can be used to implement the message transition. Hence we transform the input process into the socket statements. And we translate the channel into ServerSocket classes. The mapping from statement the input process in the Applied PI calculus to Java statements is showed in "Figure 5".

$$in(\langle term\rangle,\langle pattern\rangle) \Rightarrow \begin{Bmatrix} ServerSocket\ term = \mathbf{new} ServerSocket(num); \\ BufferedReader\ br\ = \mathbf{new} BufferedReader(\mathbf{new} \\ InputStreamReader(socket.getInputStream())); \\ String\ \ pattern = br.readLine(); \end{Bmatrix}$$

Figure 5. The Mapping from Statement the input process in the Applied PI Calculus to Statements in Java Language

❁ Output process

The output out(c, M);P outputs the message M on the channel c, then executes P. The channel c can be any term. The BNF of the input process is $out(\langle term\rangle,\langle term\rangle)$. Owning to that the similarity of the function of the input process and the output process , we also transform the output process into the statements ServerSocket classes in Java language in "Figure 6" .

 Copyright ⓒ 2016 SERSC

$$out(\langle term\rangle, \langle term\rangle) \Rightarrow \left\{ \begin{array}{l} Socket\ term = \mathbf{new} Socket(addr, num); \\ PrintWriter\ pw = \mathbf{new} PrintWriter(socket.getOutputStream()); \\ pw.println(term); \end{array} \right\}$$

Figure 6. The Mapping from Statement the output process in the Applied PI calculus to Statements in Java Language

❁ Functions

fun f/n. declares a function symbol f of arity n. This function symbol is a constructor. When private is not present, the function can be applied by the attacker. When private is present, the function cannot be applied by the attacker. The BNF of the statement fun f/n is $fun\langle ident\rangle / n$. And it does not specify the type of the variables and names of the variables. The method of calling the functions is $ident(seq\langle itemname\rangle)$. Hence we can find that there is a big difference between the function declarations in the Applied PI calculus and in JAVA language. Because the function declaration in JAVA language is in the class and the types of the variables are presented at the same time. Hence we the correspondent transformation is $public\ void\ ident(seq\langle String\ itemname\rangle)\{...\}$ and the modifier of function declarations is specified as public. The type of the return value is void. At the same time we develop a class Custom that consists of the classes that is transformations of the functions fun f/n. The method of calling functions is $Custom.ident(seq\langle itemname\rangle)$, where Custom is the specified classes, ident is the name of the function fun f/n in the Applied PI calculus. The mapping from statement fun f/n in the Applied PI calculus to Java statements is showed in "Figure 7".

$$\begin{array}{l} fun\langle ident\rangle / n \Rightarrow public\ void\ ident(seq\langle String\ itemname\rangle)\{...\} \\ ident(seq\langle itemname\rangle) \Rightarrow Custom.ident(seq\langle itemname\rangle) \end{array}$$

Figure 7. The Mapping from Statement fun f/n in the Applied PI Calculus to Statements in Java Language

When we use the Applied PI calculus to model the security protocols, some functions that are closely related to information security are defined. For examples, encryption functions, signature functions, and so on. At the same time there are some closely related information security functions and classes in JAVA language. Hence we can directly transform the functions in the Applied PI calculus that are closely related to the information security into the functions and classed in Java language.

❁ The conditional process

The conditional process if f then P else Q executes P when the fact is true. Otherwise, it executes Q. The process if f then P is equivalent to if f then P else 0. The predicate calls are subject to an implementability condition (see the clauses declaration above). Equality and inequality tests are always implementable. The BNF of the statement if f then P else Q is $if\langle fact\rangle then\langle process\rangle[else\langle process\rangle]$ .According to the operational semantics of the conditional process if f then P else Q, it should be transformed into the statement $if(\langle factjava\rangle)\langle process\rangle[else\langle process\rangle]$ in JAVA language in Figure.10, where<process> is right recursion, factJava is transformed from fact in the Applied PI calculus, "==" is transformed from "=" in the fact in the Applied PI calculus, " ! =" is transformed from "<>" in the fact in the Applied PI calculus showed in "Figure 8".

$$\left\{ if \langle fact \rangle then \langle process \rangle \left[else \langle process \rangle \right] \Rightarrow if (\langle factjava \rangle) \langle process \rangle \left[else \langle process \rangle \right] \right\}$$

Figure 8. The Mapping from the Conditional Process if f then P else Q in the Applied PI Calculus to Statements in Java Language

❁ The assignment process

The let binding let p = M in P [else Q] executes P after matching the term M with the pattern p, and binding the variables contained in p. If the term M does not match the pattern p, the process blocks, or executes Q when the else clause is present. The BNF of the let statement is $let \langle pattern \rangle = \langle term \rangle in \langle process \rangle$. According to the operational semantics, it can be translated into the statement $\langle pattern \rangle = \langle term \rangle$ in Java language in "Figure 9".

$$\left\{ let \langle pattern \rangle = \langle term \rangle in \langle process \rangle \Rightarrow \langle pattern \rangle = \langle term \rangle \right\}$$

Figure 9. The Mapping from the assignment process let p = M in P [else Q] in the Applied PI Calculus to Statements in Java Language

❁ Types

As we know there is no type in the Applied PI calculus. At the same time there are a lot of message exchanged between the communicating parties. Hence we specify that the type of the variable in the Applied PI calculus as string class in Java language. In the Applied PI calculus the return value of the function is not existed, so we also specify the type of the return value of the function as string class in Java language in "Table 2".

Table 2. The Mappings from Types in the Applied PI Calculus to Types in Java Language

Statements in the Applied PI calculus	Statements in Java language	Types in the Applied PI calculus	Java class	Annotation
free ident	String ident=new String()	none	String class	Declare a name ident
new ident	String ident=new String()	none	String class	Create a name ident
fun ident/n	public void ident(String ident1…String identn)	none	String class	Declare a function
fun ident/n	public String ident()	none	String class	The definition of the return value
in(term,pattern)	String pattern=br.readline()	none	String class	The definition of input parameter term
out(term,term)	String term=new String()	none	String class	The definition of output parameter term

5. PV2JAVA: Automatic Generator from Security Protocol Implementations Written in the Applied PI Calculus to Java Language

According to mappings introduced in section 3 and 4, we develop an automatic generator PV2JAVA which accepts security protocol implementations written in the Applied calculus as input and produces security protocol implementations written in Java language as output. In the application of automatic generator PV2JAVA. Firstly we formalize security protocols with the Applied PI calculus, and then the tool ProVerif is used to execute the security analysis, finally the automatic generator PV2JAVA is used to generate the security protocol implementations written in Java language.

 Copyright ⓒ 2016 SERSC

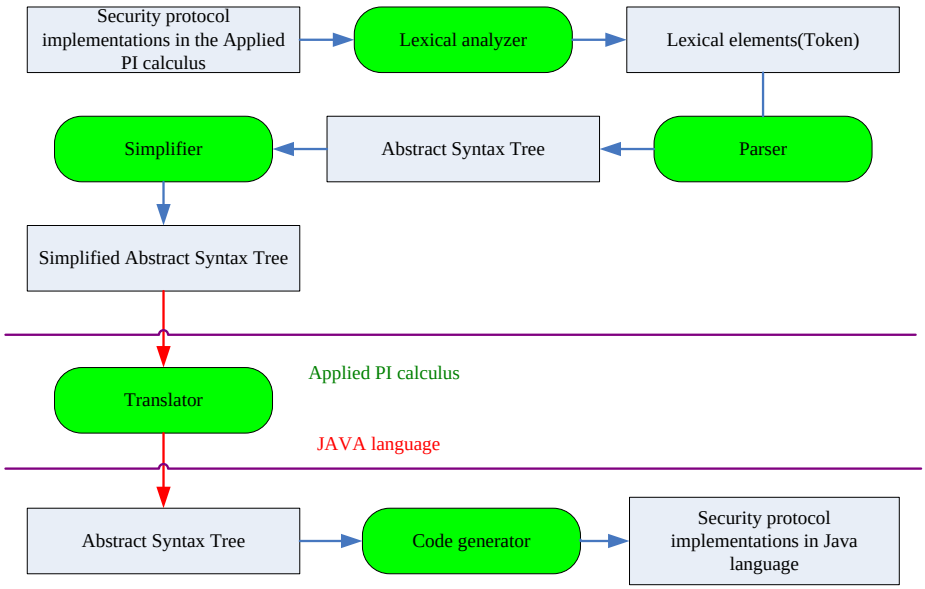

Figure 10. Idea of Development of Automatic Generator PV2JAVA

"Figure 10" shows the ideal of development of automatic generator PV2JAVA. Firstly, according to informal specification of security protocol, the security protocol implementations written in the Applied PI calculus is produced, then lexical analyzer developed by us based on syntax of the Applied PI calculus is used to analyze the correctness of security protocols implementations written in the Applied PI calculus. If verification is successful, lexical elements, for example, token, are produced. After that parser developed by us is to process tokens and produce abstract syntax tree to represent the structure of security protocol implementation written in the Applied PI calculus. The object is to generate the secure security protocol implementations written in Java language, so the structures and elements which are not related to implementation of security protocol, for example, events, are deleted and the simplified abstract syntax tree is generated. And then the translator mapping simplified abstract syntax tree in the Applied PI calculus into abstract syntax tree in Java is developed by us. After that, the implementation generator to produce the security protocol implementations written in Java language is developed. In the next section development of automatic generator PV2JAVA based on JavaCC as bellow. Firstly, we need to construct .jj file, and then use JavaCC to implement Lexical analyzer and parser for the Applied PI calculus. .jj file is mainly composed of options, function declarations, specification for lexical analysis and BNF notations for the Applied PI calculus.

❁ PV2JAVA Lexical analyzer

Lexical analyzer accepts security protocol implementation written in the Applied PI calculus as input and produces a sequence of tokens as output. The function of lexical analyzer is to verify grammar of security protocol implementation written in the Applied PI calculus based on abstract specification of the Applied PI calculus and separates a sequence of character into subsequence.

❁ PV2JAVA Parser

Using the JJtree, PV2JAVA Parser inputs the sequence of tokens for security protocol implementation written in the Applied PI calculus and generates the Abstract Syntax Tree for the Applied PI calculus. JJTree defines an Applied PI calculus interface Node that all parse tree nodes are generated. The interface support several methods for setting the parent of the node, and for adding children and retrieving nodes. JJTree can make some

Copyright ⓒ 2016 SERSC

operations in simple and multi nodes. In simple modes the type of each parse tree node is type SimpleNode that is the general interface for Abstract Syntax Tree nodes; in multi modes the parse tree node'type is derived from the name of the node. Each node is defined a node scope. Using the special identifier jjtThis, user actions in this scope can access the node under construction.

❁ PV2JAVA Simplifier

Owning to the specialty of Abstract Syntax Tree of security protocol implementation written in the Applied calculus, it is not easy to directly transform it into Abstract Syntax Tree of security protocol implementation written in Java language. At the same time it may be produce some mistakes in the transformations. In the following section we show the important simplifications of the Abstract Syntax Tree of security protocol implementation written in the Applied calculus.

❁ if -then-else

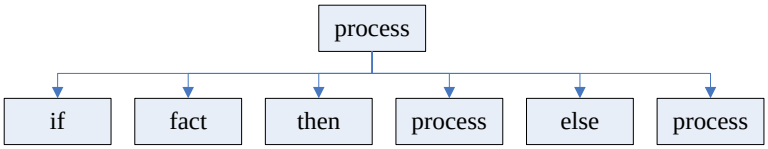

Figure 11. Abstract Syntax Tree of if-then-else in the Applied PI Calculus

According to the mapping $if\langle fact\rangle then\langle process\rangle$ 獚se$\langle process\rangle$ ƒ $if(\langle factjava\rangle)\langle process\rangle$ 獚se$\langle process\rangle$ from the statement if-then-else in the Applied PI calculus to the statement if-then-else in Java language, the statement if-then-else in the Applied PI calculus is translated into $if(\langle factjava\rangle)\langle process\rangle$ 獚se$\langle process\rangle$ .Hence we can get the abstract syntax tree of if-then-else statement in Java language in "Figure 12" through deleting the node then and changing the node fact into the node factJava in "Figure 11".

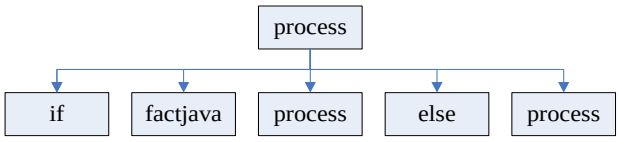

Figure 12. Abstract Syntax Tree of if-then-else in Java Language

❁ let-in

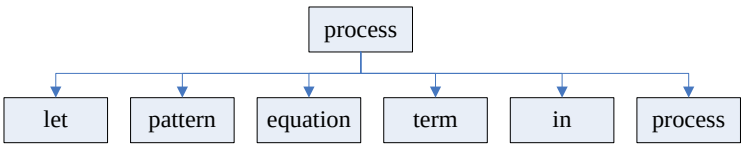

Figure 13. Abstract Syntax Tree of let-in in the Applied PI Calculus

According to the mapping $let\langle pattern\rangle = \langle term\rangle in\langle process\rangle$ ƒ $\langle pattern\rangle = \langle term\rangle$ from the statement let-in in the Applied PI calculus to the statement let-in in Java language, the statement let-in in the Applied PI calculus is translated into $\langle pattern\rangle = \langle term\rangle$. So we can get the abstract syntax tree of the statement let-in in Java language in "Figure 14" by deleting the nodes besides the nodes pattern, equation and term in "Figure 13".

 Copyright ⓒ 2016 SERSC

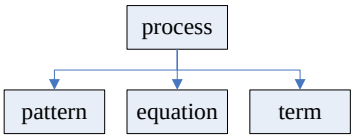

Figure 14. Abstract Syntax Tree of let-in in Java Language

❁ statement free

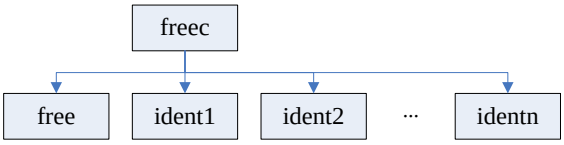

Figure 15. Abstract Syntax Tree of free in the Applied PI Calculus

According to the mapping $free\ seq\langle ident\rangle \int \{String\ ident1 = new\ String(); ...String\ identn = new\ String();\}$ from the statement free in the Applied PI calculus to the statement free in Java language, the statement free in the Applied PI calculus is translated into $\{String\ ident1 = new\ String(); ...String\ identn = new\ String();\}$. Hence we can get the statement free in Java language in Figure "16" through generating the correspondent new statement in Java language based on the ident in the Applied PI calculus in Figure "15".

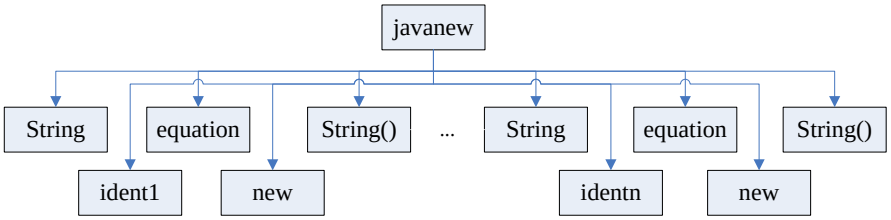

Figure 16. Abstract Syntax Tree of new in Java Language

❁ statement new

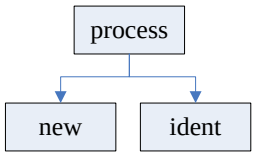

Figure 17. Abstract Syntax Tree of new in the Applied PI Calculus

According to the mapping $new\ ident \int String\ ident = new\ String()$ from the statement new in the Applied PI calculus to the statement new in Java language, the statement new in the Applied PI calculus is translated into $new\ String()$. So we can get the statement new in Java language in "Figure 18" through adding the node string and the node equation (=) in "Figure 17".

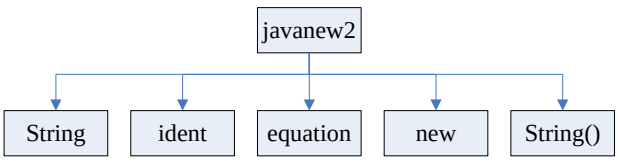

Figure 18. Abstract Syntax Tree of New in Java Language

Copyright ⓒ 2016 SERSC

❁ statement fun

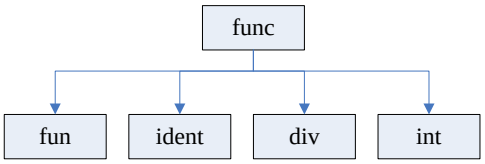

Figure 19. Abstract Syntax Tree of fun in the Applied PI Calculus

The mapping from the statement fun in the Applied PI calculus to the statement fun in Java language is $fun\langle ident\rangle / n \Rightarrow public\ void\ ident\big(seq\langle String\ itemname\rangle\big)\{...\}$. We specify the number of the parameters in statement fun in Java language based on the number of the statement fun in the Applied PI calculus. The statement fun in Java language in "Figure 20" can be gotten through adding the node public, void and variable nodes in "Figure19".

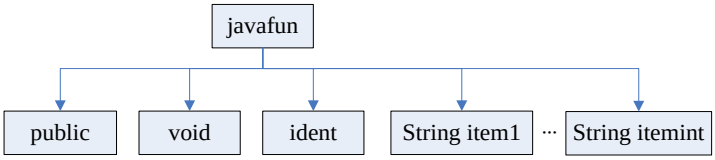

Figure 20. Abstract Syntax Tree of fun in Java Language

In order to reduce Abstract Syntax Tree of security protocol implementation written in the Applied PI calculus, we need to program a simplifier to perform the function. Simplifier accepts Abstract Syntax Tree as inputs and produces the simplified Abstract Syntax Tree. Here we implement the simplifier by using visitor pattern. JJTree has additional good support for the visitor design pattern. If we set the VISITOR option to true, then will an jjtAccept() method and childrenaccept() method are inserted into all of the node classes it generates by JJTree. Apart from that a visitor interface Visitor. Java is also produced that can be implemented and passed to the nodes to accept. Our simplifier is an instance of a visitor interface Visitor.Java.

❁ PV2JAVA Translator

PV2JAVA Translator is also a visitor which inputs the simplified Abstract Syntax Tree for the Applied PI calculus and outputs the Abstract Syntax Tree for Java language. That is said that translator is mapping function from language elements in the Applied PI calculus to language elements in Java language based on the definition. When translator has visited all the nodes in Abstract Syntax Tree for the Applied PI calculus, the correspondent Abstract Syntax Tree for the Applied PI calculus will be generated according to the model of Abstract Syntax Tree in the Applied PI calculus. Translator is developed based on SimpleNode of JavaCC.

❁ PV2JAVA Implementation generator

If we have gotten Abstract Syntax Tree of the Applied PI calculus, then the next work is to implement implementation generator to generate codes which is the security protocol implementation written in Java language. In the following we show the methods from files, fun function, free, new, expression and constant, if-then-else, let-in transformations.

❁ file

The security protocol implementations written in the Applied PI calculus is mainly composed of processes. At the same time we assume that there are two roles (sender and receiver) in the security protocols. Hence the security protocols implementations in JAVA language consists of two files. One is sender Java file. The other is receiver Java file. At the

 Copyright ⓒ 2016 SERSC

same time the import, package statements are added in those files.

❁ fun function

The mapping from the statement fun in the Applied PI calculus to the statement fun in JAVA language is $fun\langle ident\rangle/n \Rightarrow public\ void\ ident\big(seq\langle String\ itemname\rangle\big)\{...\}$. We specify the number of the parameters in statement fun in JAVA language based on the number of the statement fun in the Applied PI calculus. The statement fun in JAVA language can be gotten through adding the node public, void and variable nodes. We show the detail method through the example of fun fx/1 in the Applied PI calculus. The ident is instantiated as name fx of fun fx/1. String item1 is instantiated as String a. At the same time the correspondent notations () and {} are added. Thus public void fx(String a){} in JAVA language is generated.

❁ input process

The input process in the Applied PI calculus $in\big(\langle term\rangle,\langle pattern\rangle\big)$ can be directly translated into the following statements in JAVA language:

BufferedReaderbr=newBufferedReader(newInputStreamReader(socket.getInputStream ());

String a=br.readLine();

❁ output process

The output process in the Applied PI calculus $out\big(\langle term\rangle,\langle term\rangle\big)$ can be directly translated into the following statements in JAVA language: PrintWriter pw = new PrintWriter(socket.getOutputStream()) ; pw.println(b);

❁ free statement

There are two scenarios for free statements. We visit the leaf nodes of the abstract syntax tree of Javanew and get the nodes String、ident1、equation、new、String ()…String、identn、equation、new、String(). And then the node equation is instantiated as =. After that the new statements are listed. For example, free a,b in the Applied PI calculus, there are two variables that need to be defined, so the statements String a=new String() and String b=new String() are gotten.

When the leaf nodes of the abstract syntax tree of Javanew2 is visited and we can get the nodes String、ident、equation、new、String().And then the node equation is instantiated as =. ident is also instantiated as ident based on the different statements in the Applied PI calculus. For example, the statement new a in the Applied PI calculus, ident is instantiated as a and the statement String a=new String() is gotten by us.

❁ if-then-else statement

The leaf nodes of the abstract syntax tree of if-then-else in JAVA language is visited and the information of the nodes if, factJava, process, else, process are gotten. The notation () is add in content of the node factJava and the notation {} is also added in the content of the node process. At the same time the node process is instantiated. About the instantiation of the node factJava, when the operators are "=" and "<>", factJava should be instantiated as "==" and "!=" respectively; when the type of variables is String, factJava should be instantiated as str1.equals(str2). Here is an example to show the case. For the statement if a=b then in(c,str) else out(c,str) in the Applied PI calculus, the correspondent fact is instantiated as (a.equals(b)) and in(c,str) and out(c,str) in the Applied PI calculus are directly translated into JAVA language defined by us. At the same time the notation () is also added. Finally the statement if(a.equals(b)) {Socket statement…} else {Socket statement…} in JAVA language is generated.

❁ let-in statement

The leaf nodes of the abstract syntax tree of let-in in Java language is visited and the

Copyright ⓒ 2016 SERSC

information of the nodes pattern, equation, term. Owning to that the statement let-in is a simple assignment statement in the Applied PI calculus, the nodes pattern, equation, and term can be instantiated directly. Here is an example to show the procedure. For the statement let a=b in 0 in the Applied PI calculus, it is directly translated into the statement a=b in JAVA language. But there is a special scenario. For example, the statement let s= (a,b) in 0 in the Applied PI calculus, it is translated into the statement String[] s= {a,b} in Java language because there is not existing the statement s= (a,b) in Java language.

The automatic verifier PV2JAVA is composed of four functions: loading file of ProVerif, generating syntax tree for ProVerif, generating syntax tree for Java language and output file of Java language. In our current version of PV2JAVA , there are some limitations on it. At a time the only one process in the Applied PI calculus can be translated into classes in Java language. At the same time we also need to implement its method in the class.

6. PV2JAVA Applications on Typical Security Protocols

Here the method of software testing is used to provide a confidence on the correctness of the automatic generator PV2JAVA. We first present the security analysis of the five security protocols implementations written in the Applied PI calculus and the security analysis of the five security protocol implementations written in Java language which is generated by the automatic generator PV2JAVA. And then the five security protocols are Identity Federation Security Protocol Based on SAML, OAuth2.0, improved OAuth2.0 [8], SSHV2 and TLS 1.2. After that, the security analysis results in the Applied PI calculus are compared to the security analysis results in Java language. If the comparisons are successful, the correctness of the automatic generator PV2JAVA is provided in a degree. The results of comparison showed in Figure "Table 3".

Table 3. The comparisons between Security Protocol Implementations Written in Java Language and in the Applied PI Calculus

Security Protocols	Security Protocol implementations written in the Applied PI calculus	Result	Security Protocol implementations written in Java language	Result	Comparison
Identity Federation Security Protocol Based on SAML	Service provider authenticate identity provider	Yes	Service provider authenticate identity provider	Yes	Correspondence
	Identity Provider authenticate User Agent	No	Identity Provider authenticate User Agent	No	Correspondence
OAuth2.0	Authorization server authenticate end-user	No	Authorization server authenticate end-user	No	Correspondence
improved OAuth2.0	Authorization server authenticate end-user	Yes	Authorization server authenticate end-user	Yes	Correspondence
SSHV2	Server authenticates client	Yes	Server authenticates client	Yes	Correspondence
TLS1.2	Server authenticates client	Yes	Server authenticates client	Yes	Correspondence

For Identity Federation Security Protocol Based on SAML, Identity Provider does not authenticate User Agent because when the User Agent sends the password passwordidp in the way of plaintext to the Identity Provider. Hence the attacker can get the password passwordidp and launch an impersonation attack; Service Provider does not authenticate User Agent because when the User Agent sends the password passwordsp in the way of

 Copyright ⓒ 2016 SERSC

plaintext to the Service Provider. Hence the attacker can get the password $passwordsp$ and launch an impersonation attack; Identity Provider can authenticate Service Provider because when the Service Provider sends the digital signature $sign((id,version,issuestant,issuer,nameidpolicy),PR(keysp))$ to the Identity Provider. Hence the Identity Provider can authenticate Service Provider; Service Provider can authenticate Identity Provider because when the Identity Provider sends the digital signature $sign((aid,aversion,aissueinstant,aissuer,aauthnstatement,asubject),PR(KeyIdP2))$ to the Service Provider. Hence Service Provider can authenticate Identity Provider. When Identity Federation Security Protocol Based on SAML implementation written in is gotten, the user agent process, the service provider process and the identity provider process are inputted to the automatic generator PV2JAVA. And then Identity Federation Security Protocol Based on SAML implementation written in Java language is generated and is stored in the Java.txt.

For the authentication from authorization server to end-user in OAuth2.0 security protocol, authorization server does not authenticate end-user because when the end-user sends the password in the way of plaintext to the authorization server. Hence the attacker can get the password and launch an impersonation attack. We can find the analysis result is same to the analysis result with ProVerif in "Table 3".

For authentication from authorization server to end-user in improved OAuth2.0 security protocol, authorization server can authenticate end-user because when the end-user sends the password in the way of digital signature generated by the end-user to the authorization server. Hence the attacker cannot launch an impersonation attack. We can find the analysis result is same to the analysis result with ProVerif in "Table 3".

For authentication from server to client in SSHV2 security protocol, the result shows that server can authenticate client. We can find the analysis result is same to the analysis result with ProVerif in "Table 3".

For authentication from server to client in TLS1.2 security protocol is showed that server can authenticate client. We can find that server can authenticate client. We can find the analysis result is same to the analysis result with ProVerif in "Table 3".

7. Conclusion

Over the last years people including developers and users have done a lots of works on the formal analysis and verification of security protocol implementations written in formal language. While we know that the final objective is to have the security protocol implementations written in programming language to put it into practice in the information system. Hence we need to research the methods of generating the codes of security protocols because for the developers and users, just proof of securities of security protocol implementations written in formal language is not enough to give a strong confidence on its security properties. Although we proved the securities of security protocol implementations written in formal language, security protocol implementations written in programming language may be error and insecure. So it is important to analyze and prove security properties of security protocol implementations written in programming language.

So in this study firstly, the implementation model of implementation generation from security protocol implementations written in the Applied PI calculus is presented; Apart from that an automatic generator PV2JAVA is developed, finally the method of software testing is used to provide a strong confidence in the correctness of the automatic generator PV2JAVA.

One of our main work is to develop a generator form security protocols implementations written in the Applied PI calculus to security protocol implementations written in Java language. Our main objective is not to develop a complete complier.

Copyright ⓒ 2016 SERSC

Hence the compile time error and compiler optimization *etc*. have not been addressed in current work. About the correctness of translation from security protocols implementations written in the Applied PI calculus to security protocol implementations written in Java language, in our current version, we use the software testing method to provide a strong confidence in it by the five simplified typical security protocols. Owning to the differences of the Applied PI calculus and Java language, we will prove it from the view of operational semantics. Moreover, the operational semantics are different due to the different target in the near future.

In our current version of PV2JAVA , there are some limitations on it. At a time the only one process in the Applied PI calculus can be translated into classes in Java language. At the same time we also need to implement its method in the class.

In the near future we will use the proof assistant Coq to prove the correctness of translation from security protocol implementations written in the Applied PI calculus to security protocol implementations written in Java language and enhance the ability.

Acknowledgement

This study was supported in part by the natural science foundation of Hubei Province under the grants No. 2014CFB249, titled “Automatic Verification of Cryptographic Security in Security Protocol Code and Development of Software Tools”, conducted in Wuhan, China and by the foundation of China Scholarship Council.

References

[1] Pironti and R. Sisto, "Provably correct Java implementations of Spi Calculus security protocols specifications", Computers & Security.,vol.29, no.3,**(2010)**,pp. 302-314.
[2] M.Avalle, A. Pironti, R.Sisto and D. Pozza, "The Java SPI Framework for Security Protocol Implementation", Proceedings of the 2011 Sixth International Conference on Availability, Reliability and Security, Vienna University of Technology, Austria,**(2011)** August 22-26.
[3] M.Backes, A. Busenius, and C. Hritcu, "On the development and formalization of an extensible code generator for real life security protocols", Proceedings of the 4th international conference on NASA Formal Methods,Norfolk,Virginia,USA,**(2012)**April 3-5.
[4] X.Li, S.LI, D.Li and J.Ma,"Multi-language oriented automatic realization method for cryptographic protocols", Journal on Communications., vol.33,no.9,**(2012)**,pp.152-159.
[5] D.Cade and B.Blanchet, "From Computationally-proved Protocol Specifications to Implementations", Proceedings of the 2012 Seventh International Conference on Availability, Reliability and Security, Prague,**(2012)**, August 20-24.
[6] D.Cade and B.Blanchet, "Proved generation of implementations from computationally secure protocol specifications", Proceedings of the Second international conference on Principles of Security and Trust, Rome, Italy, **(2013)**,March 16-24.
[7] Z.Li, B. Meng, D. Wang and W. Chen, "Mechanized Verification of Cryptographic Security of Cryptographic Security Protocol Implementation in JAVA through Model Extraction in the Computational Model", Journal of Software Engineering., vol.9,no.1,**(2015)**,pp. 1-32.
[8] W.Chen, Y.T. Yang and L.Y.Yang, "Improved OAuth2.0 Protocol and Analysis of its Security", computer systems and applications., vol.23,no.3,**(2014)**,pp. 25-30.

 Copyright ⓒ 2016 SERSC

Authors

Bo Meng, he was born in 1974 in China. He received his M.S. degree in computer science and technology in 2000 and his Ph.D. degree in traffic information engineering and control from Wuhan University of Technology at Wuhan, China in 2003. From 2004 to 2006, he worked at Wuhan University as a postdoctoral researcher in information security. Currently, he is a full Professor at the school of computer, South-Center University for Nationalities, China. He has authored/coauthored over 50 papers in International/National journals and conferences. In addition, he has also published a book "secure remote voting protocol" in the science press in China. His current research interests include security protocols and formal methods.

Yitong Yang, she was born in 1991 and is now a postgraduate at the school of computer, South-Center University for Nationalities, China. Her current research interests include security protocols and formal methods.

Jinli Zhang, she was born in 1991 and is now a postgraduate at the school of computer, South-Center University for Nationalities, China. Her current research interests include security protocols and formal methods.

Jintian Lu, he was born in 1991 and is now a postgraduate at the school of computer, South-Center University for Nationalities, China. His current research interests include the formal and inverse analysis of security protocols.

Dejun Wang, he was born in 1974 and received his Ph.D. in information security at Wuhan University in China. Currently, he is an associate professor in the school of computer, South-Center University for Nationalities, China. He has authored/coauthored over 20 papers in international/national journals and conferences. His current research interests include security protocols and formal methods.

Copyright ⓒ 2016 SERSC

Copyright ⓒ 2016 SERSC