# 基于计算模型的安全协议 Swift 语言实施安全性分析

孟博，何旭东，张金丽，尧利利，鲁金钿
（中南民族大学计算机科学学院，湖北 武汉 430074）

**摘 要**：分析 IOS 平台上的安全协议 Swift 语言实施安全性，对保障 IOS 应用安全具有重要意义。首先对已有安全协议 Swift 语言实施进行分析，确定 Swift 语言子集 SubSwift，并给出其 BNF；其次基于操作语义，建立 SubSwift 语言到 Blanchet 演算的映射模型，主要包含 SubSwift 语言的语句、类型到 Blanchet 演算的语句及类型的映射关系与规则；再次根据 SubSwift 语言到 Blanchet 演算的映射模型，提出从安全协议 SubSwift 语言实施生成安全协议 Blanchet 演算实施方法；最后应用 Antrl4 工具和 Java 语言开发安全协议 Blanchet 演算实施生成工具 SubSwift2CV，分析 OpenID Connect 协议、Oauth2.0 协议和 TLS 协议的 SubSwift 语言实施安全性。

**关键词**：安全协议；实施安全性；Swift 语言；形式化分析；模型抽取
**中图分类号**：TP309
**文献标识码**：A
**doi:** 10.11959/j.issn.1000−436x.2018165

# Security analysis of security protocol Swift implementations based on computational model

MENG Bo, HE Xudong, ZHANG Jinli, YAO Lili, LU Jintian
College of Computer Science, South Central University For Nationalities, Wuhan 430074, China

**Abstract:** Analysis of security protocol Swift implementations in IOS platform is important to protect the security of IOS applications. Firstly, according to the security protocol Swift implementations, the SubSwift language, which was a subset of Swift language, was widely used in IOS system, and its BNF were specified. Secondly, the mapping model from SubSwift language to Blanchet calculus based on the operational semantic was presented which consisted of mapping rules, relationship from the statements and types in SubSwift language to Blanchet calculus. And then, a method of generating security protocol Blanchet calculus implementations from SubSwift language implementations was developed. Finally, security protocol Blanchet calculus implementation generation tool SubSwift2CV was developed with Antrl4 and Java language. At the same time, OpenID Connect, Oauth2.0 and TLS security protocol SubSwift language implementations were analyzed with SubSwift2CV and CryptoVerif.

**Key words:** security protocol, implementations security, Swift language, formal analysis, model extraction

## 1 引言

安全协议既是网络空间安全[1-4]的重要组成部分，又是保障网络空间安全的关键。从安全协议设计、安全协议抽象规范安全性的分析与验证[5-6]到安全协议实施（安全协议代码）[7-10]，人们主要集中在安全协议抽象规范安全性的分析和验证方面[11]进行研究，其实用性较差。近几年来，人们对安全协议

**收稿日期**：2017−07−21；**修回日期**：2018−07−23
**通信作者**：孟博，mengscuec@gmail.com
**基金项目**：国家自然科学基金资助项目（No.61272497）；湖北省自然科学基金资助项目（No.2014CFB249, No.2018ADC150）；中南民族大学中央高校基本科研业务费专项资金资助项目（No.CZZ18003, No.QSZ17007）
**Foundation Items:** The National Natural Science Foundation of China (No.61272497), The Natural Science Foundation of Hubei Province (No.2014CFB249, No.2018ADC150), The Central University Basic Business Expenses Special Funding for Scientific Research Project (No.CZZ18003, No.QSZ17007)

的最终表现形式——安全协议实施越来越感兴趣。因为无论任何安全协议都必须进行安全协议实施安全性分析才能发挥作用，所以安全协议实施安全性分析对保障网络空间安全具有重要意义。

当前，基于得到的安全协议实施，分析其安全性的主要方法有程序验证法和模型抽取法。用程序验证方法对已有安全协议实施的安全性进行分析时，主要集中在基于逻辑、基于类型系统、类型系统与逻辑证明相结合等方面，直接分析安全协议实施的安全性。这些方法大部分既没有证明分析过程的正确性，又依赖于在安全协议实施中添加大量的代码注释与断言。文献[12-13]首先提出安全协议采用 C 语言和 Java 语言实施安全性分析方法：文献[14-15]基于符号模型，提出分析安全协议 C 语言实施认证性和保密性的方法。文献[16-21]基于 F 类语言类型检查器分析安全协议 F 类实施的认证性和保密性。模型抽取方法首先从安全协议实施中抽取安全协议抽象规范，并且证明抽取方法的正确性，然后用协议抽象规范安全性分析工具来分析其安全性。此方法被认为是有效且合理的，适合分析协议实施这类规模较小的代码。在程序验证方法中，也有部分涉及模型抽取的方法，分别是基于符号模型和计算模型。文献[22-24]抽取安全协议 F 类实施的抽象模型，并分析其安全属性；文献[25-27]抽取安全协议 C 语言实施的抽象模型，分别使用 ProVerif[28]和 CryptoVerif[29]分析其安全性和保密性；文献[30-31]抽取安全协议 Java 实施的抽象模型，并分析其安全属性。

目前，在 IOS 平台上，许多安全协议采用 Swift 语言实施，故其安全性的分析对保障 IOS 应用安全具有重要意义。同时，目前未见关于对安全协议 Swift 语言实施安全性进行分析的相关文献，因此本文基于计算模型，应用模型抽取方法来分析安全协议 Swift 语言实施的安全性。

本文主要贡献如下。

1) 定义与安全协议实施相关的 Swift 语言子集 SubSwift，并给出 SubSwift 语言的巴科斯范式（BNF, Backus-Naur form）。

2) 建立 SubSwift 语言到 Blanchet 演算的映射模型，其主要包含 SubSwift 语言的语句、类型到 Blanchet 演算的语句、类型的映射关系与规则等。

3) 提出从安全协议 SubSwift 语言实施生成安全协议 Blanchet 演算实施的方法，并开发安全协议 SubSwift 语言实施到 Blanchet 演算实施生成工具 SubSwift2CV。

4) 应用 CryptoVerif 和开发的安全协议 Blanchet 演算实施生成工具 SubSwift2CV，分析 OpenID Connect 协议、Oauth2.0 协议、TLS（transport layer protocol）协议等 SubSwift 语言实施的安全性。

## 2　相关工作

文献[22]提出验证安全协议 F#实施安全性的架构，支持密码原语的具体实施和符号化实施。该文献应用 FS2PV 工具把安全协议 F#实施转换为 ProVerif 的输入语言——Applied PI 演算，进而用 ProVerif 工具分析安全协议 F#语言实施的保密性和认证性。文献[30]开发 Elygah 系统，该系统首先把安全协议的 Java 语言实施转化成 Lysa 演算进程，然后得到安全协议 Java 语言实施的形式化模型，进而分析其认证性。但是该文献没有证明抽取方法的正确性。文献[25]首先应用符号执行技术，符号化执行安全协议 C 语言实施，获得安全协议 C 语言实施发送/接收消息的符号化描述；然后得到其形式化抽象模型；最后使用 ProVerif 分析其保密性和认证性。但目前该模型只支持顺序执行语句，不支持分支语句。文献[23-24]开发 ML 子集到 CryptoVerif 的编译器原型 FS2CV，该编译器把安全协议 F#语言实施中的密码原语、数据库、信道语句等转换成基于 Blanchet 演算的形式化模型，然后用 CryptoVerif 分析其 F#语言实施的保密性和认证性，并且采用手工方式证明抽取方法的正确性。该文献验证 TLS 协议 F#语言实施进行分析，证明其认证性和保密性。文献[26]从安全协议 C 语言实施中抽取其形式化模型，然后用 CryptoVerif 分析其认证性与保密性。首先，通过符号化执行从安全协议的 C 语言实施中抽取安全协议 C 语言实施抽象模型。然后，把抽取的模型转换成 CryptoVerif 语法描述，使用 CryptoVerif 分析安全协议 C 语言实施的认证性与保密性。但是目前只支持顺序执行协议，不支持分支语句。文献[31]首先，通过定义 SubJava 与 Blanchet 演算间的语法映射关系，基于模型抽取技术开发模型抽取工具 SubJava2CV。该工具对安全协议的 Java 语言实施首先进行分析并生成抽象语法树，然后化简抽象语法树，抽取出安全模型，将其转换为 Blanchet 演算的抽象语法树，最后生成 Blanchet 演算代码。最后，使用 SubJava2CV 抽取一个认证协议 Java 语言实施的安全模型，将其转换为 Blanchet

演算代码，应用自动化分析工具 CryptoVerif 分析安全属性，证明协议实施的认证性。文献[32]首先基于符号模型和计算模型，应用 ProVerif 和 CryptoVerif 分析 TLS1.3 安全协议抽象规范的安全性，然后给出 TLS1.0-1.3 安全协议实施——RefTLS。

## 3 SubSwift 语言与 Blanchet 演算

### 3.1 SubSwift 语言及其 BNF

Swift 语言子集 SubSwift 的定义主要考虑 2 个因素。1) Swift 语言是一种复杂的编程语言。若在定义 SubSwift 时，考虑 Swift 语言的所有语句，则太过复杂。另外，由于 Blanchet 演算是一种安全协议建模语言，其语言简单且功能不够强大，故不能建立从 Swift 到 Blanchet 的演算映射。2) 对已有的安全协议实施 Swift 进行分析，找出开发安全协议 Swift 实施所需要的核心语句。综合考虑这 2 个因素来定义 SubSwift 语言，并定义 SubSwift 语言的巴科斯范式，如图 1 所示。

SubSwift 语言主要包含的语句有：表达式语句（expression）、声明语句（declaration）、分支语句（branch_statement）、控制转移语句（control_transfer_statement）。表达式语句主要包含常量（value）、变量（variable）、赋值表达式（assignment）、算术运算表达式、逻辑运算表达式（logical_statement）等。由于 Blanchet 演算没有涉及算术运算，故没有考虑 SubSwift 语言的算术运算部分。SubSwift 语言表达式语句主要包含表达式的值、赋值表达式、逻辑运算表达式等。声明语句是对要进行安全协议实施中所需的常量、变量、函数、类、库等进行定义。SubSwift 语言声明语句主要包含导入声明（import_declaration）、常量声明（constant_ declaration）、变量声明（variable_ declaration）、函数声明（functions_declaration）和类声明（class_ declaration）。导入声明语句是指定 Swift 库文件数据分组名，该分组定义多种元素。控制转移语句用于控制代码的执行顺序，从而实现代码的跳转。SubSwift 语言控制转移语句包括 return 语句、throw 语句。return 语句用于从当前执行的函数返回到主调用函数，同时传回返回值。throw 语句用于创建异常和返回代码错误信息。分支语句用于某种特定条件下执行相应代码的情形。SubSwift 语言分支语句主要包括 if 语句和 guard 语句。if 语句如 ：当且仅当 if 后的<condition>值为 true 时，执行{statements1}；当<condition>值为 false 时，并且 if 语句中有 else 语句，就执行{statements2}，否则直接执行下一条语句。guard 语句如：与 If 语句实现相同的功能，当且仅当 guard 后的{statement}值为 true 时直接跳过 else 语句执行下一条语句。当条件表达式为 false 时，就执行 else 语句，并跳出 guard 语句代码段。

```
statement ::=
      expression
    | declaration
    | branch_statement
    | control_transfer_statement
    | do_statement;
expression ::=
    ⟨identifier⟩[: type]
  | assignment_expression
  | logical_expression;
assignment_expression ::=
  ⟨identifier⟩[: type] = ⟨identifier⟩;
logical_expression ::=
  ⟨expression⟩ == ⟨expression⟩;
  ⟨expression⟩ != ⟨expression⟩;
  ⟨expression⟩ && ⟨expression⟩;
  ⟨expression⟩ || ⟨expression⟩;
declaration ::=
    import_declaration
  | constant_declaration
  | variable_declaration
  | function_delcaration
  | class_declaration;
import_declaration ::=
    import ⟨class | var | func⟩⟨identifier⟩;
constant_declaration ::=
    let ⟨identifier⟩[: type][= ⟨identifier⟩];
variable_declaration ::=
    var ⟨identifier⟩[: type][= ⟨identifier⟩];
function_declaration ::=
    func ⟨identifier⟩{statements};
class_declaration ::=
    class ⟨identifier⟩{statements};
control_transfer_statement ::=
      return_statement
    | throw_statement ;
return_statement ::=
      return [expression];
throw_statement ::=
      throw expression ;
branch_statement ::=
      if_statement
    | guard_statement;
if_statement ::=
    if ⟨condition⟩ {statements} [else {statements}];
guard_statement ::=
    guard ⟨condition⟩ else {statements} ;
```

图 1 SubSwift 语言的巴科斯范式

### 3.2 Blanchet 演算及其 BNF

2008 年，Blanchet[27]提出一种概率进程演算——Blanchet 演算，它基于 PI 演算和其他几种演算的思想。在 Blanchet 演算中，消息是位串，密码原

语可看作位串到位串的函数的序列。这种演算具有概率学语义，并且所有进程的运行是多项式时间。观察等价[10]是证明安全属性过程中使用到的主要技术之一，其基本思想是：若攻击者能够区分进程 Q 和进程 Q′ 的可能性是可忽略的，则进程 Q 观察等价于进程 Q′。Blanchet 演算增加一种新功能，进程执行过程中所有变量的值都存储在数组中，这是协议分析过程实现自动化的关键，比如，表示进程索引编号 $i$ 中的 $x$ 的变量值。数组代替链表，而链表经常用在安全协议的手工证明方面。例如，考虑（MAC, message authentication code）安全性的定义。通常，MAC 安全性定义攻击者伪造 MAC 成功概率是可忽略的，因此，在安全性证明过程中，有个链表保存 MAC 算法中处理过的参数，当要证明消息 $m$ 的 MAC 时可以检查消息 $m$ 是否在链表中。在 Blanchet 演算中，MAC 生成算法，使用过的参数都保存在数组中，通过遍历该数组来寻找消息 $m$。相对于人工论证的链表，数组不需要对其中插入的值进行附加说明。这样消除了许多细微且难以自动化的语法转换，便于通过等式来表示数组元素之间的关系。Blanchet 演算是一种建立安全协议抽象模型的建模语言。图 2 给出了 Blanchet 演算的巴科斯范式。

## 4　安全协议 SubSwift 语言实施到安全协议 Blanchet 演算实施的映射模型

基于操作语义，建立安全协议 SubSwift 语言实施到安全协议 Blanchet 演算实施的映射模型，如图 3 所示。将安全协议 SubSwift 语言实施中的发送者类、接收者类分别转化为 Blanchet 演算中的发送者进程、接收者进程。套接字声明转化为 Blanchet 演算中的通道声明，消息发送、接收接口分别对应 Blanchet 演算中通道的输出、输入。

### 4.1　SubSwift 语言到 Blanchet 演算的语句映射

在 3.1 节中定义的 SubSwift 语言主要包含表达式语句、声明语句、分支语句、控制转移语句等。

表达式语句主要定义常量、变量、赋值表达式和逻辑表达式。表达式语句到 Blanchet 演算的 BNF 映射规则如图 4 所示。常量 a ∈ value 映射为 Blanchet 演算中的 a[:$T$]，变量 $x$ ∈ variable 映射为 Blanchet 演算中的项 $x$[:$T$]。赋值表达式 assignment：$x$ = a 映射为 let…in 语句 let y：$T$=simp leterm in；逻辑表达式映射为 Blanchet 演算中的（cond），逻辑与表达式 logicaland：$e_1$ && $e_2$ 映射为 ⟨simpleterm⟩ && ⟨simpleterm⟩，逻辑或 logicalor：$e_1 \| e_2$ 映射为 ⟨simpleterm⟩ ‖ ⟨simpleterm⟩，逻辑等于 Logical equan：$e_1 == e_2$ 映射为⟨simpleterm⟩=⟨simpleterm⟩，逻辑不等于 logicalunequal：$e_1 != e_2$ 映射为⟨simpleterm⟩ <> ⟨simpleterm⟩。

```
x, y | ident, T | type
simpleterm :=
        x[:T]
        | f(seq⟨x⟩)
cond :=
        ⟨simpleterm⟩
        | ⟨simpleterm⟩ = ⟨simpleterm⟩
        | ⟨simpleterm⟩ <> ⟨simpleterm⟩
        | ⟨simpleterm⟩ || ⟨simpleterm⟩
        | ⟨simpleterm⟩ && ⟨simpleterm⟩

term :=
        new x:T;
        | let y:T = simpleterm in
        | if ⟨cond⟩ then ⟨term⟩ [else ⟨term⟩]
        | find ⟨simpleterm⟩ suchthat defined ⟨cond⟩ then
event :=
        [inj] x[seq⟨T⟩].
queryterm :=
        ⟨event⟩
        | ⟨simpleterm⟩
        | ⟨queryterm⟩ && ⟨queryterm⟩
        | ⟨queryterm⟩ || ⟨queryterm⟩
query :=
        query secret x.
        | query secret1 x.
        | query ⟨event⟩ ==> ⟨queryterm⟩.
funtion :=
        fun ident(seq⟨T⟩):T [compos].
channel :=
        ⟨indent⟩[seq⟨term⟩]
channels :=
        channel seq⟨indent⟩.
pattern :=
        x
        |= ⟨term⟩
        | seq⟨x⟩
        | seq⟨pattern⟩
outprocess :=
        x[;outprocess]
        | event y[seq⟨x⟩][;outprocess]
        | new x:T[;outprocess]
        | let ⟨pattern⟩ = ⟨term⟩[in ⟨outprocess⟩]
[else ⟨outprocess⟩]
        | if ⟨cond⟩ then ⟨outprocess⟩ [else ⟨outprocess⟩]
        | find ⟨simpleterm⟩ suchthat defined ⟨cond⟩ t
hen[else ⟨outprocess⟩]
        | out(⟨channel⟩,⟨term⟩)[;⟨inprocess⟩]
inprocess :=
        0
        | in(⟨channel⟩,⟨pattern⟩)[;⟨outprocess⟩]
        | ⟨inprocess⟩ | ⟨inprocess⟩
        | !⟨ident⟩⟨inprocess⟩
```

图 2　Blanchet 演算的巴科斯范式

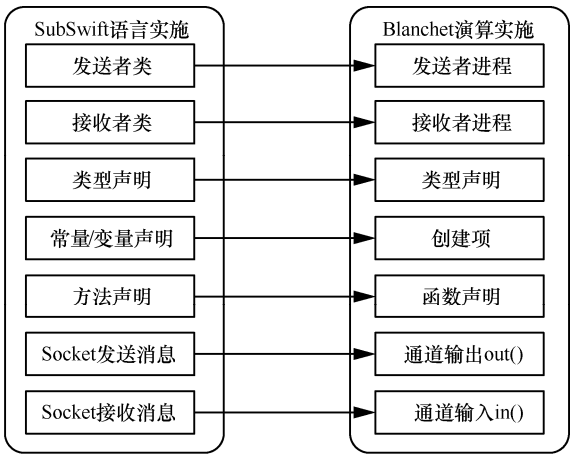


图 3　SubSwift 语言到 Blanchet 演算的映射模型

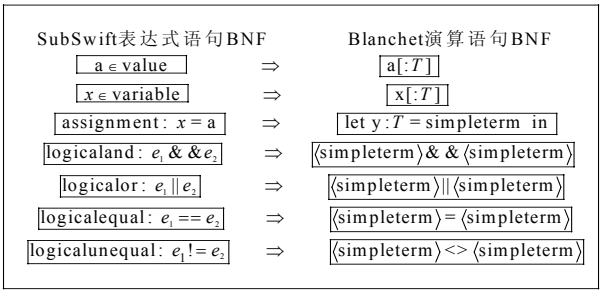


图 4　SubSwift 语言表达式到 Blanchet 演算的 BNF 映射规则

SubSwift 语言声明语句到 Blanchet 演算的 BNF 映射关系如图 5 所示。SubSwift 语言的声明语句主要包含导入声明、常量声明、变量声明、函数声明、类声明等。在 Blanchet 演算中，不存在与 SubSwift 中 import 语句对应的语句，因此，若安全协议 SubSwift 语言实施转换为 Blanchet 演算模型的过程中遇到 import 语句，则直接跳过。

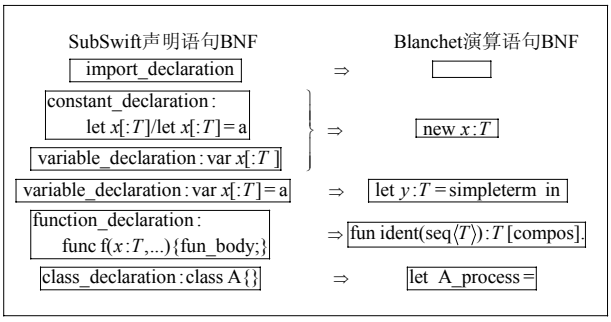


图 5　SubSwift 语言声明语句到 Blanchet 演算的 BNF 映射关系

Swift 语言基本语句到 Blanchet 演算的 BNF 映射关系如图 6 所示。SubSwift 语句映射为 Blanchet 演算语句时，任意一条 SubSwift 语句 statement : S 映射为 Blanchet 演算中输入或输出进程中的 ⟨outprocess / inprocess⟩，每个 SubSwift 语句块 code_block : {S*} 映射为 Blanchet 演算中输入或输出进程中的多条语句 ⟨outprocess / inprocess⟩*，if 语句 if_statement : if e {P} else {Q} 映射为 if ⟨cond⟩ then ⟨outprocess⟩ else ⟨outprocess⟩，guard 语句 guard_statement: guard e else {P} 映射为 if ⟨!cond⟩ then ⟨outprocess⟩，guard 语句映射为 Blanchet 演算中的 if 语句时，其条件表达式要进行取非操作。因为 else 后的语句块在 guard 语句条件不成立时执行，反之，else 后的语句块不会执行。创建类的实例 new_class : let $x$ = class($e_1, e_2, \cdots$)，映射为 Blanchet 演算中的 new 语句 new $x:T$，调用类方法 class_method: class.method($e_1, e_2, \cdots$)，映射为 Blanchet 演算中的函数 fun ident (seq⟨$T$⟩) :$T$ [compos]。SubSwift 语言用套接字等方法发送与接收消息，对应于 Blanchet 演算中 out 和 in 语句。Socket 中的 Send 方法 socket_send : socket.send($x_1, x_2, ...$) 发送消息，映射为 out 语句 out(⟨channel⟩, ⟨term⟩) [;⟨inprocess⟩；socket 中的 Receive 方法 socket_receive:socket. receive($x_1, x_2, \cdots$) 接收消息，映射为 in 语句 in (⟨channel⟩, ⟨term⟩)[;⟨outprocess⟩。

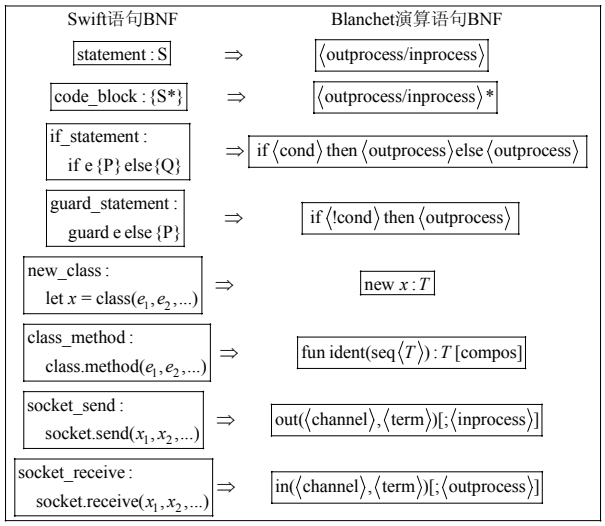


图 6　SubSwift 语言基本语句到 Blanchet 演算的 BNF 映射关系

### 4.2　SubSwift 语言到 Blanchet 演算的类型映射

SubSwift 语言和 Blanchet 演算是强类型语言，两者的主要区别在于，SubSwift 语言中的基本类型如整型 int、单精度浮点型 float、双精度浮点型 double 等都是 SubSwift 语言预先定义的；而在 Blanchet 演算中，除与密码体制、签名机制相关的部分数据类型是由 Blanchet 演算预先定义的之外，

其他类型是先声明后使用的。因此，在定义 SubSwift 语言到 Blanchet 演算的类型映射关系时，主要考虑以下 2 个方面：1) 对 SubSwift 语言中的基本数据类型，在 Blanchet 演算中声明一个同名的数据类型，直接进行调用；2) SubSwift 语言中与密码体制、签名机制相关的数据类型需映射到 Blanchet 演算中对应的密码体制、签名机制中的数据类型。

## 5　Blanchet 演算实施生成工具 SubSwift2CV

根据第 4 节定义的 SubSwift 语言到 Blanchet 演算的映射模型，Blanchet 演算实施首先基于计算模型，提出安全协议 SubSwift 语言实施生成安全协议 Blanchet 演算实施的方法，如图 7 所示。然后利用 Antrl4[34]工具和 Java 开发安全协议 Blanchet 演算实施生成工具 SubSwift2CV。

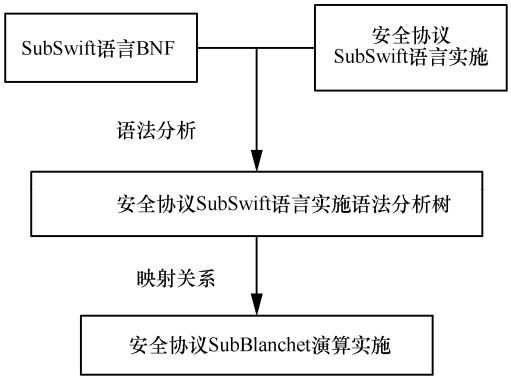


图 7　安全协议 Blanchet 演算实施生成工具开发模型

Antrl4 工具能够读取、处理、执行或翻译二进制文件或结构化文本，被广泛应用于构建各类语言、工具和框架。Antrl4 工具的监听器 Listener 机制将语法和语言应用进行隔离，用户只需在语法关系匹配短语开始和结束时添加相应的动作即可获取指定形式的数据。因此应用 Antrl4 工具及其监听器 Listener 来开发生成工具 SubSwift2CV，详细过程如图 8 所示。首先根据 SubSwift 语言的 BNF，应用 Antrl4 工具开发 SubSwift 语言词法分析器和语法分析器。然后输入安全协议 SubSwift 语言实施，并用词法分析器和语法分析器对其进行词法分析和语法分析，得到安全协议 SubSwift 语言实施语法分析树。再用 SubSwift 语言到 Blanchet 演算语言的映射关系，遍历所获取的安全协议 SubSwift 语言实施语法分析树，生成安全协议 Blanchet 演算实施。最后根据 Blanchet 演算语法规则完善生成的安全协议 Blanchet 演算实施。

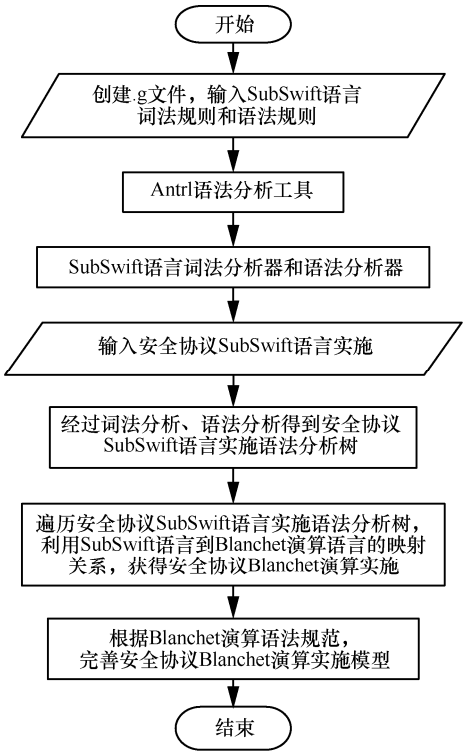


图 8　SubSwift2CV 开发过程

### 5.1　安全协议 SubSwift 语言实施语法分析

应用 Antrl4 工具对安全协议 SubSwift 语言实施进行语法分析，并获取安全协议 SubSwift 语言实施语法分析树。首先根据 SubSwift 语言的词法规则开发 SubSwift 语言词法分析器并对安全协议 SubSwift 语言实施进行词法分析，进而得到安全协议 SubSwift 语言实施的词法元素序列。然后根据 SubSwift 语言的语法规则开发 SubSwift 语言语法分析器，对获得的安全协议 SubSwift 语言实施词法元素序列进行语法分析，判断安全协议 SubSwift 语言实施是否符合 SubSwift 语言语法规则，若符合，则生成安全协议 SubSwift 语言实施的语法分析树。SubSwift 语言词法分析器和语法分析器开发原理如图 9 所示。

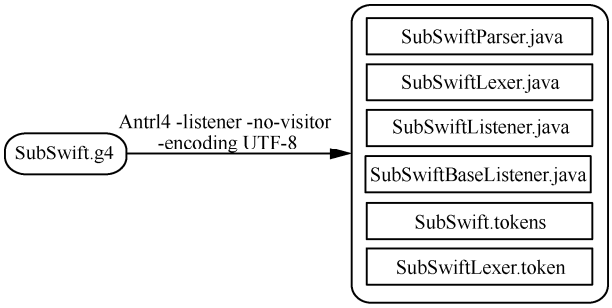


图 9　SubSwift 语言语法分析器开发原理

为开发 SubSwift 语言词法分析器和语法分析器，创建 SubSwift.g4 文件并存入 SubSwift 语言的词法规则和语法规则。执行 SubSwift.g4 文件生成的语法分析相关文件。SubSwiftLexer.java 是 SubSwift 词法分析器，将 SubSwift 代码转换为单词元素序列。SubSwiftParser.java 是 SubSwift 语法分析器，对单词元素序列进行语法分析。SubSwift BaseListener.java 由监听器 Listener 默认实现。

### 5.1.1 SubSwift 语言词法分析器

词法分析从左到右读入实施中的字符，并根据定义的词法规则识别并确定单词元素及其属性，从而将输入流中的字符串序列分割为单词元素序列，将其作为语法分析器的输入。每个单词元素序列是一个独立的序列 1，包括常量、关键字、标识符、运算符、分界符等。Antrl4 工具可以根据定义的 SubSwift 语言词法规则自动化生成 SubSwift 语言词法分析器 SubSwiftLexer.java，在 SubSwift.g4 文件中定义词法规则。在 SubSwift 语言中，常量包括布尔型、数值型等，在词法规则定义中，布尔型取值为 true 或 false，即为：boolean_literal:'true'|'false' 。整型可分为二进制数（binary_literal）、八进制数（octal_literal）等，其词法规则如图 10 所示。

```
integer_literal
   :Binary_literal
   |Octal_literal
   |Decimal_literal
   |Hexadecimal_literal;
Binary_literal:'0b'Binary_literal_character;
Binary_literal_character:[01][01]+;
Octal_literal:'0o'Octal_literal_character;
Octal_literal_character:[0-7][0-7]+;
Decimal_literal: [0-9][0-9]+;
Hexadecimal_literal:'0x'Hexadecimal_literal_character;
Hexadecimal_literal_character:[0-9a-fA-F][0-9a-fA-F]+;
```

图 10 SubSwift 语言中的整形词法规则

关键字是程序语言中具有固定意义的标识符，这些标识符只能作为保留字，不能被重新定义为一般标识符。在 SubSwift 语言中，定义的关键字如图 11 所示。

```
IMPORT:'import'      导入语句关键字
LET   :'let'         常量声明关键字
VAR   :'var'         变量声明关键字
FUNC  :'func'        函数声明关键字
CLASS :'class'       类声明关键字
IF    :'if'          } 分支语句关键字
ELSE  :'else'        }
GUARD :'guard'       }
RETURN:'return'      } 控制转换语句关键字
THROW :'throw'       }
```

图 11 SubSwift 语言中的关键字

安全协议 SubSwift 实施中标识符是程序语言中用来表示安全协议实体的字符串，如变量名、函数名、类名等。标识符通常由字母、数字和下划线组成，且不能以数字开头，SubSwift 语言中标识符的词法规则如下。

```
identifier:identifier_head identifier_character?;
identifier_head:[a-zA-Z]|'_';
identifier_character:[0-9]|identifier_head;
identifier_characters:identifier_character+
```

运算符一般分为算术运算符、逻辑运算符、赋值运算符、关系运算符及连接运算符。在 Blanchet 演算中，只有赋值运算和逻辑运算，因此主要考虑 SubSwift 语言运算符中的赋值运算符和逻辑运算符。SubSwift 语言中运算符的定义如图 12 所示。

```
assignment_operator :'='     //赋值运算符
logical_and         :'&&'    //逻辑与运算符
logical_or          :'||'    //逻辑或运算符
logical_equal       :'=='    //逻辑相等运算符
logical_unequal     :'!='    //逻辑不相等运算符
```

图 12 SubSwift 语言中的运算符

分界符是程序语言中一个重要组成部分，通常包括括号、分号、冒号、下划线等。SubSwift 语言的分界符定义如图 13 所示。

```
LCURLY  :'{';      //左大括号
LPAREN  :'(';      //左小括号
LBRACK  :'[';      //左中括号
RCURLY  :'}';      //右大括号
RPAREN  :')';      //右小括号
RBRACK  :']';      //右中括号
COMMA   :',';      //逗号
COLON   :':';      //冒号
SEMI    :';';      //分号
UNDERSCORE :'_';   //下划线
```

图 13 SubSwift 语言中的分界符

在 SubSwift 语言代码中，有些代码如空格、换行符、注释等在词法分析中可忽略，其具体词法规则定义如下。

```
WS : [ \n\r\t]+-> channel(HIDDEN) ;
block_comment : '/*' (block_comment|.)*? '*/'-> channel(HIDDEN) ;
line_comment :'//'.*? ('\n'|EOF)->channel(HIDDEN)
```

其中，WS 的作用是忽略换行符、空格、制表符，Block_comment 中定义的是多行注释，line_comment 中定义的是单行注释，channel(HIDDEN)的作用是跳过这些字符。

### 5.1.2　SubSwift 语言语法分析器

SubSwift 语言语法分析器的输入是经词法分析器后所得到的单词元素序列。根据给定的形式文法分析并确定单词元素序列的语法组织结构，判断源程序代码语法的正确性，若源程序在语法组织结构上正确，则生成语法分析树。

根据 Antrl4 工具的文法定义规则，在 SubSwift.g4 文件中添加定义的 SubSwift 语言的语法规则，编译执行之后生成的 SwiftParser.java 文件，即为 SubSwift 语言语法分析器。SubSwift 的 BNF，主要包括表达式语句、声明语句、分支语句和控制转换语句，根据 SubSwift 中语句的语法规范，SubSwift.g4 文件的文法规则定义如下。

```
statement :
    expression ';'                    //表达式语句
    |declaration ';'                  //声明语句
    |branch_statement ';'             //分支语句
    |control_transfer_statement ';' //控制转换语句
```

SubSwift 的表达式语句主要包含赋值语句和逻辑运算表达式语句。赋值语句如 variable[:Type] = identifier，逻辑运算表达式包含逻辑等于、逻辑不等于、逻辑与、逻辑或，它们分别对应于< identifier > == <identifier>、<identifier> != < identifier >、< identifier > & & < identifier > 及 <identifier > || < identifier >，根据其 BNF，可以在 SubSwift.g4 文件中将表达式的文法规则定义如下。

```
expression :
        identifier[: type]
        |assignment_expression
        |logical_expression
```

SubSwift 语言主要关注的声明语句有导入声明、常量声明、变量声明、函数声明及类声明。Sub Swift.g4 文件中声明语句的文法规则定义如图 14 所示。

```
declaration ::=
    import_declaration
    |constant_declaration
    |variable_declaration
    |function_delcaration
    |class_declaration;
import_declaration ::=
    import ⟨class | var | func⟩⟨identifier⟩;
constant_declaration ::=
    let ⟨identifier⟩[: type][= ⟨identifier⟩];
variable_declaration ::=
    var ⟨identifier⟩[: type][= ⟨identifier⟩];
function_declaration ::=
    func ⟨identifier⟩{statements};
class_declaration ::=
    class ⟨identifier⟩{statements};
```

图 14　SubSwift.g4 中声明语句的文法规则

在 SubSwift 语言中，除表达式语句、声明语句之外，其他语句还包括分支语句和控制转换语句，其中，分支语句主要包括 if 语句和 guard 语句，控制转换语句主要包括 return 语句和 throw 语句，在 SubSwift.g4 文件中，文法规则定义如下。

```
branch_statement :
        if_statement
       |guard_statement ;
if_statement : 'if ' condition_clause code_block
else_clause ;
else_clause : 'else' code_block | 'else' if_statement;
guard_statement : 'guard' condition_
clause 'else' code_block ;
condition_clause : expression ;
control_transfer_statement :
    return_statement
    |throw_statement ;
return_statement : 'return' expression ;
throw_statement : 'throw' expression
```

## 5.2　安全协议 Blanchet 演算实施生成

### 5.2.1　遍历安全协议 SubSwift 语言实施语法分析树

用 ParseTreeWalker 类遍历安全协议 SubSwift 语言实施语法分析树。Antrl4 工具的语法分析器编译 SubSwift.g4 文件时，会根据定义的 SubSwift 语言语法规则自动化生成语法树监听器 ParseTree

Listener 接口的子接口 SwiftListener 及其默认实现 SwiftBaseListener，其中包含 SubSwift 语言语法中每个规则对应的 Enter 方法和 Exit 方法。采用深度优先遍历 SubSwift 语言语法分析树。当树遍历器遇到 SubSwift 语言某个语法规则节点时，就会调用该规则所对应的 Enter 方法，并将该语法树节点的上下文传递给该规则对应的上下文对象；当树遍历器遍历完这个节点下所有的子节点之后，就会调用该规则对应的 Exit 方法。

#### 5.2.2 生成 Blanchet 演算实施

用 Antrl4 工具对 SubSwift 语句进行分析得到的语法分析树无法按照传统的方法通过对语法分析树的节点进行移动、删除或添加等操作来获得与之对应的 Blanchet 演算语法树，进而获得 Blanchet 演算语句。故采用 Antrl4 工具中特有的方式——注解语法树来获取 Blanchet 演算语句。

注解 SubSwift 语法分析树是为了将 SubSwift 语句对应的 Blanchet 演算语句存储在该 SubSwift 语句语法分析树的根节点中，只要访问根节点对应的映射值即可获得某条 SubSwift 语句所对应的 Blanchet 演算语句。

注解 SubSwift 语法分析树的具体方法是：通过使用 ParseTreeProperty 类型的字段 cv 及帮助方法 getCV()、setCV()，得到 SubSwift 语言输入短语所对应的 Blanchet 演算语句。在 SubSwift 语法分析树中，首先，将每棵子树转换为 Blanchet 演算语句，然后，把相关字符串关联在该子树的根节点上，在更高节点上捕获这些关联的字符串来获取更大的字符串，并关联在该节点上，最后，整个语句的根节点所关联的字符串即为整个语句转换为 Blanchet 演算后的结果。在 SubSwift2CV.java 文件中，定义转换器类 CVEmitter、字段 cv 和帮助方法 getCV()、setCV()，getCV()用于获取与当前根节点所关联的字符串值，setCV()用于注解当前根节点，其代码如下。

```
public static class CVEmitter extends SwiftBaseListener {
        ParseTreeProperty<String> cv =
        new ParseTreeProperty<String>();
        String getCV(ParseTree ctx) { return cv.get(ctx); }
        void setCV(ParseTree ctx, String s) { cv.put(ctx, s); }
```

以变量声明及初始化语句为例，说明注解语法分析树的过程。根据 SubSwift 语言到 Blanchet 演算的映射模型可知：变量声明未初始化语句 var $x[:T]$ 对应的 Blanchet 演算语句的 BNF 为 new $x:T$；而变量声明与初始化语句 var $x[:T]=a$ 对应的 Blanchet 演算语句的 BNF 为 let $y:T$ = simpleterm in。在变量声明语句语法分析树中，变量声明时是否初始化具体表现为 initializer 根节点的子树结构是否存在。此外，SubSwift 语法分析树结构中的 pattern_initializer 节点对应的 BNF 结构 $x[:T]=a$ 与 Blanchet 演算中的 $y:T$ = simpleterm 结构是对应的，通过在 pattern_initializer 节点对应的映射值中添加关键字 new 或 let…in 即可实现 SubSwift 变量声明语句到 Blanchet 演算语句的转换。

### 5.3 SubSwift2CV 界面设计与功能说明

SubSwift2CV 的主要功能包括：首先接收 SubSwift 语言实施并进行词法分析、语法分析，进而输出该 SubSwift 语言实施的语法分析树，最后遍历 SubSwift 语言实施的语法分析树，根据所建立的安全协议 SubSwift 语言到 Blanchet 演算的转换模型，生成并输出该 SubSwift 语言实施所对应的 Blanchet 演算实施。SubSwift2CV 界面主要包括 3 个区域，分别是显示输入的 SubSwift 语言实施、SubSwift 语言实施的语法分析树结构及对应的 Blanchet 演算实施。其界面设计如图 15 所示。

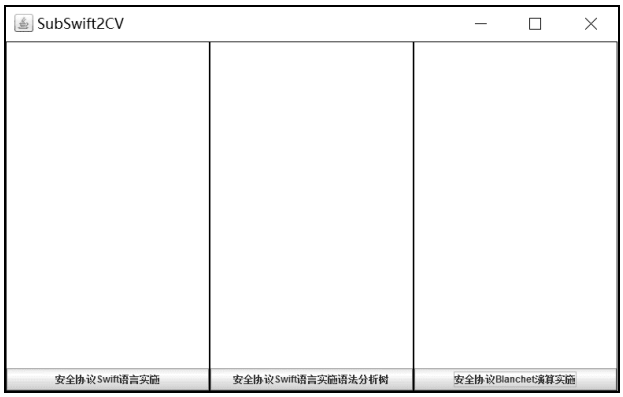


图 15 SubSwift2CV 的界面

## 6 应用案例

应用 SubSwift2CV 和 CryptoVerif 分析 OpenID Connect 协议[35]、Oauth2.0 协议[36]、TLS 协议[37]的 SubSwift 语言实施安全性，其原理如图 16 所示。

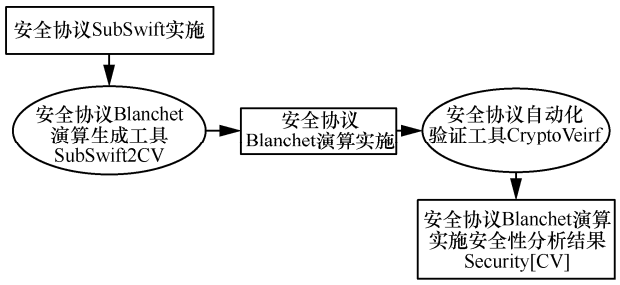


图 16　分析安全协议 SubSwift 实施安全性的框架

由图 16 可知，分析安全协议 SubSwift 语言实施安全性的主要思路为：首先，得到安全协议 SubSwift 语言实施；然后，将其导入 SubSwift2CV 中，生成安全协议 Blanchet 演算实施，最后，将该 Blanchet 演算实施转化为 CryptoVeirf 的输入，进而应用 CryptoVeirf 分析其安全性。下面以 OpenID Connect 安全协议 SubSwift 语言实施安全性分析为例，详细说明应用 SubSwift2CV 和 CryptoVeirf 分析安全协议 SubSwift 语言实施安全性的过程。

### 6.1　OpenID Connect 安全协议 SubSwift 语言实施安全性分析

OpenID Connect 安全协议包含客户端（client）、终端用户（end-user）和 OpenID 供应商（OpenID provider）3 个主体。 OpenID Connect 协议可通过隐式流（implicit flow）、授权码流（authorization code flow）和混合流（hybrid flow）3 种方式进行身份认证，不同的方式决定 ID Token 和 Access Token 返回到客户端的方式不同，其消息流程也会有所区别。本文选择授权码流的方式进行身份认证，ID Token 和 Access Token 从令牌终端中返回，客户端可以利用授权服务器发送的授权码向令牌终端请求 ID Token 和 Access Token，这种方式可有效防止令牌泄露。OpenID Connect 协议的消息结构如图 17 所示。

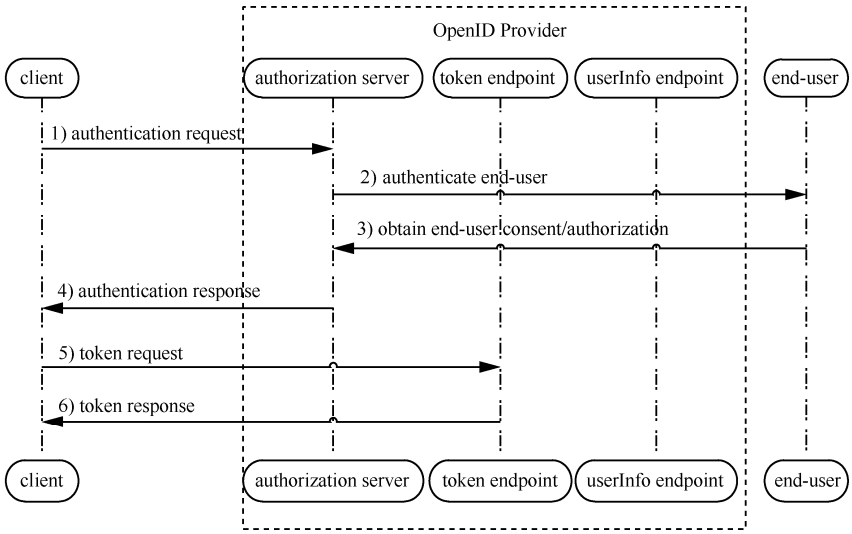


图 17　OpenID Connect 协议消息结构

授权码流进行身份验证的客户端首先将用户名、密码和重定向 URL 发送给 OpenID 供应商，然后接收 OpenID 供应商的公钥，最后收到来自 OpenID 供应商的令牌响应。因为令牌响应中的 ID Token 经过数字签名，客户端收到令牌响应之后，需要验证该数字签名，若验证结果为真，则表明客户端能认证 OpenID 供应商若验证为所示时。其中 OpenID 供应商首先要生成公钥和私钥，并在客户端发送令牌响应时使用生成的私钥对 ID Token 进行数字签名。当 OpenID 供应商收到客户端发送的身份信息时，将产生的公钥发送给客户端。当 OpenID 供应商收到发自客户端的令牌请求时，对客户端的身份信息进行验证，验证成功后生成 ID Token 和 Access Token 及相关参数，并向客户端发送令牌响应，执行 OpenID Connect 安全协议 OpenID 供应商、客户端 SubSwift 语言实施。结果如图 18 所示。结果表明客户端能够认证 OpenID 供应商。

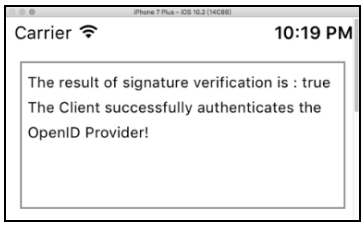


图 18　OpenID Connect 协议 SubSwift 语言实施安全性分析结果

### 6.2 OpenID Connect 安全协议 Blanchet 演算生成及其安全性分析

首先，将获取的安全协议 SubSwift 语言实施输入 SubSwift2CV。然后，经过词法分析及语法分析，进而生成安全协议 SubSwift 实施语法分析树，遍历语法分析树，从而生成安全协议 Blanchet 演算实施。最后，将生成的安全协议 Blanchet 演算实施转换为 CryptoVerif 工具的输入进而分析其安全性，同时获得其安全性分析结果。

实验采用的 OpenID Connect 源码[38]是由 Aero Gear 项目提供的，该项目致力于将企业与移动端结合起来使跨平台企业移动开发变得容易，目前，该项目获得 Twitter、Facebook 、Google 等公司支持，分析该项目的 OpenID Connect 实施对网络空间安全具有重要意义。

OpenID Connect 安全协议将 OpenID 供应商、客户端 SubSwift 语言实施分别导入 SubSwift2CV 工具中，输出结果如图 19 和图 20 所示。整理 SubSwift2CV 工具生成的 OpenID Connect 安全协议 Blanchet 演算实施，并将其转化为 CryptoVerif 语句后输入 Crypto Verif 工具中，进而得到的安全性分析结果如图 21 所示。结果表明客户端能够认证 OpenID 供应商。

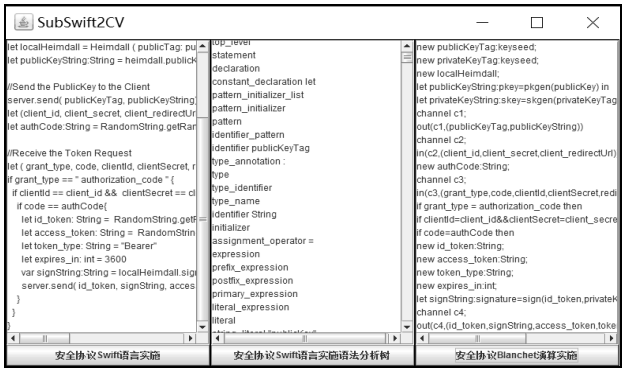


图 19 OpenID 供应商 Blanchet 演算实施结果

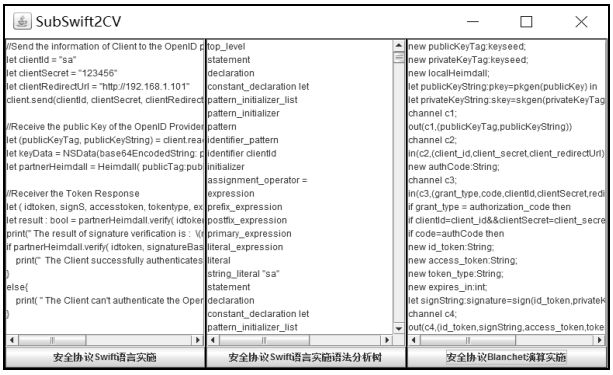


图 20 OpenID 客户端 Blanchet 演算实施

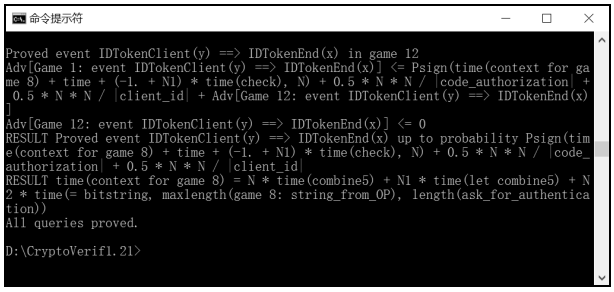


图 21 OpenID Connect 协议 Blanchet 演算实施安全性分析结果

### 6.3 Oauth2.0 和 TLS 安全协议 SubSwift 语言实施安全性分析

用同样的方法，对 Oauth2.0 安全协议 SubSwift 语言实施和生成的 Blanchet 演算实施的安全性分别进行分析，结果如图 22 和图 23 所示。

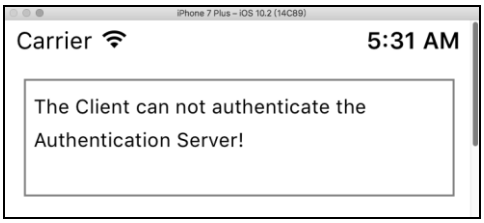


图 22 Oauth2.0 协议 SubSwift 语言实施安全性分析结果

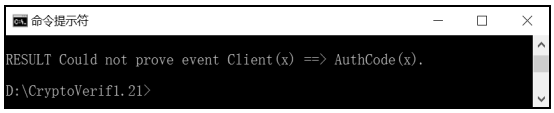


图 23 Oauth2.0 协议 Blanchet 演算实施安全性分析结果

由图 22 和图 23 可知，Oauth2.0 协议 SubSwift 语言实施和 Blanchet 演算实施的安全性分析结果是一致的，这表明客户端无法认证授权服务器，即当客户端收到授权码时不能够确定该授权码是否来自授权服务器。因为授权服务器向客户端发送授权码的过程中，没有采用数字签名机制，所以攻击者能获得其授权码，并对其进行篡改。

同样地，对 TLS 安全协议 SubSwift 语言实施和生成的 Blanchet 演算实施的安全性分别进行分析，结果如图 24 和图 25 所示。

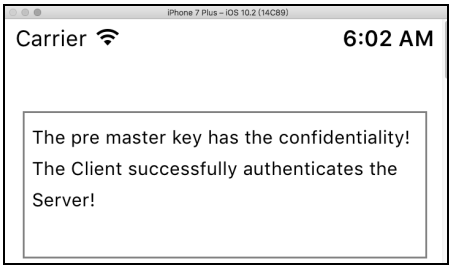


图 24 TLS 协议 SubSwift 语言实施安全性分析结果

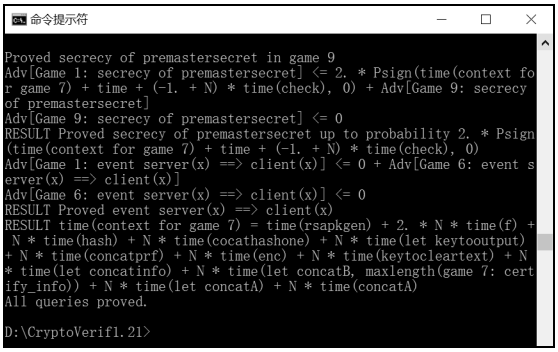


图 25　TLS 协议 Blanchet 演算实施安全性分析结果

由图 24 和图 25 可知，TLS 安全协议 SubSwift 语言实施和 Blanchet 演算实施的安全性分析结果是一致的，这表明在客户端与服务器通信的过程中，能够保证预主密钥的保密性，且客户端能够认证服务器。

## 7　结束语

基于计算模型对 IOS 平台上的安全协议 Swift 语言实施的安全性进行分析，对保障 IOS 应用安全具有重要意义。首先对已有的安全协议 Swift 语言实施进行分析，进而确定与安全协议 Swift 实施紧密相关的 Swift 语言子集 SubSwift。然后根据操作语义，建立从 SubSwift 语言到 Blanchet 演算的映射模型，提出从安全协议 SubSwift 语言实施中抽取安全协议 Blanchet 演算实施的方法，并开发安全协议 Blanchet 演算实施生成工具 SubSwift2CV，同时对 OpenID Connect 协议、Oauth2.0 协议及 TLS 协议的安全性进行分析。结果表明 OpenID Connect 协议、Oauth2.0 协议和 TLS 协议的 SubSwift 语言实施与安全协议 Blanchet 演算实施的安全性分析结果分别是“客户端能够认证 OpenID 供应商”和“客户端无法认证授权服务器”。在 SubSwift 客户端与服务器通信过程中能够保证预置密钥保密性，且客户端能认证服务器端。

开发的安全协议 Blanchet 演算实施生成工具 SubSwift2CV 不是复杂的编译器，故在当前的版本中存在优化等问题。未来计划展开以下 4 个方面的工作：1) 对 SubSwift 语言进行扩充，使其包含更多语句和特征；2) 基于互模拟技术，应用 Coq 来证明 SubSwift2CV 工具的正确性；3) 使用 SubSwift2CV 和 CryptoVerif 分析更多的安全协议 SubSwift 实施的安全性；4) 把本文提出的模型抽取方法和映射模型进行推广，分析 IOS 平台上大量存在的安全协议 Object C 语言实施的安全性。

## 参考文献：

[1] 张焕国, 韩文报, 来学嘉, 等. 网络空间安全综述[J]. 中国科学:信息科学, 2016, 46(2): 125-164.
ZHANG H G, HAN W B, LAI X J, et al. Survey on cyberspace security[J]. SCIENTIA SINICA Informationis, 2016, 46(2): 125-164.

[2] 王世伟. 论信息安全、网络安全、网络空间安全[J]. 中国图书馆学报, 2015(2):72-84.
WANG S W. On information security, network security and cyberspace security[J]. Journal of Library Science in China, 2015(2):72-84.

[3] MIN K S, CHAI S W, HAN M. An international comparative study on cyber security strategy[J]. International Journal of Security and its Applications, 2015, 9(2):13-20.

[4] 张焕国, 吴福生, 王后珍, 等. 密码协议代码执行的安全验证分析综述[J]. 计算机学报, 2018,41(2): 288-308.
ZHANG H G, WU F S, WANG H Z, et al. A survey: security verification analysis of cryptographic protocols implementations on real code[J]. Chinese Journal of Computers, 2018, 41(2): 288-308.

[5] 孟博, 张金丽, 鲁金钿. 基于计算模型的 OpenID Connect 协议认证性的自动化分析[J]. 中南大学民族大学学报（自然科学版）, 2016,35(3): 123-129.
MENG B, ZHANG J L, LU J T. Automatic analysis of authentication of OpenID Connect protocol based on the computational model[J]. Journal of South-Central University for Nationalities (Natural Science Edition), 2016, 35(3): 123-129.

[6] 牛乐园, 杨伊彤, 王德军, 等. 计算模型下的 SSHV2 协议认证性自动化分析[J]. 计算机工程, 2015, 41(10): 148-154.
NIU L Y, YANG Y T, WANG D J, et al. Automatic analysis on authentication of SSHV2 protocol in computational model[J]. Computer Engineering, 2015, 41(10): 148-154.

[7] AVALLE M, PIRONTI A, SISTO R. Formal verification of security protocol implementations: a survey[J]. Formal Aspects of Computing, 2014, 26(1): 99-123.

[8] MENG B, HUANG C T,YANG Y T, et al. Automatic generation of security protocol implementations written in Java from abstract specifications proved in the computational model[J]. International Journal of Network Security, 2017,19(1): 138-153.

[9] MENG B, YANG Y T, ZHANG J L, et al. PV2Java: automatic generator of security protocol implementations written in Java language from the applied PI calculus proved in the symbolic model[J]. International Journal of Security and its Applications, 2016, 10(11): 211-229.

[10] 孟博, 王德军.安全协议实施自动化生成与验证[M]. 北京: 科学出版社, 2016.
MENG B, WANG D J. Automatic generation and verification of security protocols' implements[M]. Beijing: Science Press, 2016.

[11] 雷新锋, 宋书民, 刘伟兵, 等. 计算可靠的密码协议形式化分析综述[J]. 计算机学报, 2014, 37(5): 993-1016.
LEI X F, SONG S M, LIU W B, et al. A Survey on computationally sound formal analysis of cryptographic protocols[J]. Chinese Journal of Computers, 2014, 37(5): 993-1016.

[12] GOUBAULT L J, PARRENNES F. Cryptographic protocol analysis on real C code[C]//International Workshop on Verification, Model Checking, and Abstract Interpretation. 2005: 363-379.

[13] JURJENS J. Automated security verification for crypto protocol im-

plementations: verifying the JESSIE project[J]. Electronic Notes in Theoretical Computer Science, 2009, 250(1): 123-136.

[14] CHAKI S, DATTA A. ASPIER: an automated framework for verifying security protocol implementations[C]//22nd IEEE Computer Security Foundations Symposium. 2009:172-185.

[15] DUPRESSOIR F, GORDON A D, JÜRJENS J, et al. Guiding a general-purpose C verifier to prove cryptographic protocols[C]// 24th IEEE Computer Security Foundations Symposium. 2011:3-17.

[16] BHARGAVAN K, GORDON A D. Modular verification of security protocol code by typing[C]//ACM Sigplan-Sigact Symposium on Principles of Programming Languages. 2010:445-456.

[17] BACKES M, MAFFEI M, UNRUH D. Computationally sound verification of source code[C]// 17th ACM Conference on Computer and Communications Security. 2010:387-398.

[18] BENGTSON J, BHARGAVAN K, FOURNET C, et al. Refinement types for secure implementations[J]. ACM Transactions on Programming Languages and Systems, 2011, 33(2): 8-45.

[19] SWAMY N, CHEN J, FOURENT C, et al. Secure distributed programming with value-dependent types[C]]// 16th ACM Sigplan International Conference on Functional Programming. 2011:266-278.

[20] SWAMY N, HRIŢCU C, KELLER C, et al. Semantic purity and effects reunited in F*[C]// 20th ACM SIGPLAN International Conference on Functional Programming. New York: ACM, 2015:12.

[21] SWAMY N, HRIŢCU C, KELLER C, et al. Dependent types and multi-monadic effects in F*[C]// 43rd annual ACM SIGPLAN-SIGACT Symp on Principles of Programming Languages. 2016: 256-270.

[22] BHARGAVAN K, FOURNET C, GORDON A D, et al. Verified interoperable implementations of security protocols[J]. ACM Transactions on Programming Languages and Systems. 2008, 31(1): 5.

[23] BHARGAVAN K, CORIN R, FOURNET C, et al. Automated computational verification for cryptographic protocol implementations[J]. Unpublished draft, Oct, 2009.

[24] BHARGAVAN K, FOURNET C, CORIN R, et al. Cryptographically verified implementations for TLS[C]// 15th ACM Conference on Computer and Communications Security. 2008:459-468.

[25] MIHHAIL A, GORDON A D, JÜRJENS J. Extracting and verifying cryptographic models from C protocol code by symbolic execution[C]//18th ACM Conference on Computer and Communications Security. 2011: 331-340.

[26] AIZATULIN M, GORDON A D, JURJENS J. Computational verification of C protocol implementations by symbolic execution[C]//19th ACM Conference on Computer and Communications Security. 2012: 712-723.

[27] BHARGAVAN K, BLANCHET K,KOBEISSI N. Verified models and reference implementations for the TLS 1.3 standard candidate[C]// 38th IEEE Symp on Security and Privacy. 2017:20.

[28] BLANCHET B. A computationally sound mechanized prover for security protocols[J]. IEEE Transactions on Dependable and Secure Computing, 2008, 5(4):193-207.

[29] BLANCHET B. An efficient cryptographic protocol verifier based on prolog rules[C]//14th IEEE Computer Security Foundations Workshop, Cape Breton.2001:82-96.

[30] O'SHEA N. Using ELYJAH to analyses Java implementations of cryptographic protocols[C]// FCS-ARSPA-WITS'08. 2008: 211-223.

[31] LI Z M, MENG B, WANG D J, et al. Mechanized verification of cryptographic security of cryptographic security protocol implementation in JAVA through model extraction in the computational model[J]. Journal of Software Engineering, 2015, 9(1): 1-32.

[32] 唐朝京, 鲁智勇, 冯超. 基于计算语义的安全协议验证逻辑[J]. 电子学报, 2014, 42(6):1179-1185.
TANG Z J, LU Z Y, FENG C. A verification logic for security protocols based on computational semantics[J]. Chinese Journal of Electronics, 2014, 42(6):1179-1185.

[33] Apple Inc. The Swift Programming Language[EB/OL]. [2017-4-1]

[34] TERENCE P. The definitive Antrl4 reference[M]. USA: The Pragmatic Bookshelf, 2012

[35] SAKIMURA N, BRADLEY J, JONES M, et al. OpenID connect core 1.0[EB/OL].[2017-1-10]

[36] XU X D, NIU L Y, MENG B. Automatic verification of security properties of OAuth 2.0 protocol with CryptoVerif in computational model[J]. Information Technology Journal, 2013, 12(12): 2273-2285.

[37] MENG B, NIU L Y, YANG Y T, et al. Mechanized verification of security properties of transport layer security 1.2 protocol with CryptoVerif in computational model[J]. Information Technology Journal, 2014, 13(4): 601-613.

[38] Client library for OAuth2/OpenID Connect [EB/OL]. [2016-10-1].

[作者简介]

**孟博**（1974-），男，河北行唐人，博士，中南民族大学教授、硕士生导师，主要研究方向为安全协议和形式化方法。

**何旭东**（1991-），男，湖北武汉人，中南民族大学硕士生，主要研究方向为安全协议实施安全。

**张金丽**（1991-），女，湖北随州人，中南民族大学硕士生，主要研究方向为安全协议实施安全。

**尧利利**（1993-），女，江西抚州人，中南民族大学硕士生，，主要研究方向为安全协议实施安全

**鲁金钿**（1991-），男，土家族，湖南湘西人，中南民族大学硕士生，主要研究方向为形式化方法和安全协议逆向分析。