SPI 演算规范的建模、实现验证研究*)

徐东红　齐　勇　侯　迪
(西安交通大学电子与信息工程学院　西安 710049)

摘　要　针对安全协议安全属性是否满足，缺乏有效性能评价方法的现状，大都使用 SPI 演算或相近的进程代数方法进行建模。利用这种方法不仅能够有效地形式化描述安全协议，并且能够对安全协议进行多方面的系统评价，但基本上没有说明怎么样寻找设计合适的验证工具，验证其安全属性实现的正确性。本文引入基于 SPI 演算的验证工具 SPRITE 来保证建模过程正确性，并设计给出实现映射的具体方法。本方法通过对典型的 WOO-LAM 单向认证协议予以说明，最后 SPRITE 产生的具体 JAVA 代码，给出了安全协议的安全属性，使形式化描述的协议的安全属性是否满足更接近于人的理解而不仅只是机器的解释。

关键词　形式化方法，进程代数，安全协议，SPI 演算，应用程序接口(API)

Research on Model and Implementation Authentication of SPI Calculus Specifications

XU Dong-Hong　QI Yong　HOU Di
(School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an 710049)

Abstract　Under the situations of effective approach being lack to evaluate the security properties of security protocols, an approach is proposed to construct the security protocol based on SPI calculus or similar process algebra in this paper. Applying this approach, the formal description of security protocol can be described effectively, furthermore, the SPI calculus can be implemented. However, there is no any traces to illustrate how to find or design an adapted tool to authenticate the correction of the security properties of the security protocol. In this paper we introduce SPRITE, SPRITE is a tool based on SPI calculus that can guarantee whether is the model and process describing right or not, and design the concrete ways to implement the reflection from abstract SPI calculus to concrete Java code. The approach is illustrated by a typed one way authentication protocol WOO-LAM security protocol. The authentication tool SPRITE generate concrete Java code that can be simple and concrete to describe the security property. It is more adapt to mankind's understand not just to adapt computer interpretation.

Keywords　Formal method, Process algebra, Security protocol, SPI calculus , API

随着互联网技术的迅猛发展，对于开放的分布式系统、电子商务、移动自主网络的广泛应用以及对普适计算和 Web 服务组合的理论研究，提出了适合各自特性和应用的安全协议。在工业界，对安全协议的正确性与一致性的研究越来越集中于形式化的方法。近几年来，用 SPI 演算来分析安全协议成为一个热点。主要因为 SPI 演算可以详细方便地描述安全协议的主体行为及主体之间的信息交换和安全属性，并且可以对一些系统推理。SPI 演算具有严格的语法、语义来描述安全协议。文[1,2]中对 SPI 演算进行了描述并扩展了两个基本原语，描述和验证在 Windows2000 系统中用于验证的 Kerberos 协议；文[2,3]中对标准的 SPI 演算的语法及语义进行了命题扩展，并利用扩展后的 SPI 演算对大型复杂协议 SSLV3.0 的安全性进行了形式化分析；文[4]中使用了类 pi 演算来验证电子支付协议的认证性和匿名性；文[5]应用 SPI 演算中的互模拟方法来说明安全协议的安全属性。文[6]定义了一个实时 SPI 演算来描述了攻击者模型，对保密性的认证性进行了研究。

这些方法对安全协议的安全属性进行了描述和推理。但研究发现，这些安全协议的安全属性都是基本 SPI 演算的语法、语义推理规则进行推理验证属性，验证结果正确与否不易被一般人理解，且没有找到一个合适的实现工具来实现。为了克服 SPI 演算的抽象问题，文[6]中 Stark 等曾把抽象解释引入到 SPI 演算的安全协议分析验证中，针对 SPI 演算在分析过程中标准语义过于抽象提出了其非标准语义，说明了非标准语义与标准语义的等价性的原理，并把这种方法应用在用 SPI 演算分析安全路由协议的方法中，但结果依然是比较抽象，不是很理解。

在本文中，设计了基于 SPI 演算的形式化描述的验证方法，该方法能实现规范验证的具体代码，即基于 SPI 演算的一种应用程序接口(API)，在形式化描述与具体代码中构架一座桥梁，使形式化描述的规范能映射到具体的可执行实现代码上，应用已有的编译器具体实现安全协议属性的正确与否，使对安全属性的描述和验证更接近于一般人的理解而不只是机器的解释。使用 SPI 演算的框架如图 1，由图 1 可以看出：

(1)接口层必须提供一个抽象地、能够信任地说明安全协议的具体行为属性。本文的方法是从高层的规范开始，逐步通过接口实现具体的实现代码。

(2)从抽象层到具体层间的映射之间的关系的正确性必

*)国家自然科学基金(No. 60473098)和国家高技术研究发展计划(863)(No. 2004AA112040)资助。徐东红　博士生，主要研究领域为形式化方法、PI 演算、Web 组合服务；齐　勇　教授，博士生导师，主要研究领域为计算机网络与分布计算、中间件技术。

须得以保证。

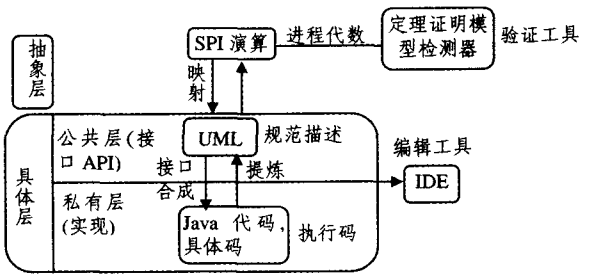

图 1 使用 SPI 演算分析的框架模型

1 安全协议的建模描述分析

在没有应用形式化方法以前,大多采用非形式化的 Alice/Bob 格式的概念来分析安全协议的安全属性问题。尽管这种方法对协议不同的主体之间交换的信息有合理的描述,但有时主体内部的行为对安全协议的安全性至关重要,如是否泄露了会话密钥和新鲜值。随着工业要求的发展,用形式化的方法来分析安全协议越来越广泛,先后出现了推理逻辑、GNY 安全、BAN 逻辑、串空间以及基于进程代数的 CCS、PI 演算和 SPI 演算来判定安全协议的安全属性是否保持。对于各种方法的介绍以及优缺点,文[8]进行了比较分析。在本文中,我们采用基于 SPI 演算建模描述。为了和具体实现工具对映,并且实现具体代码,我们对 SPI 演算做一个简单的介绍说明。SPI 演算是由 Abadi 和 Gordon 在[9,1]对 PI 演算的定义和进程进行安全的扩展并命名为 SPI 演算,即安全的 PI 演算,之所以近年来工业界采用 SPI 演算来描述、分析、验证安全协议,主要是因为 SPI 演算是在 PI 演算的基础为了详细说明如何能使用加密、解密而创立的,允许主体的进程之间在公共通道上发送、接收信息。有精确的语法、语义进行严格的推理,在 PI 演算的基础上,对于对称密钥、非对称密钥、数字签名,散列函数的协议分析更加方便。SPI 演算至少在下面两个阶段起建模分析作用:

(1)安全协议的发展设计阶段会产生众多的问题,SPI 演算可以起到描述发展中的安全协议的规范,给出基本的模型并通过推理工具进行验证,可以作为具体实现代码的参照。

(2)在相反的工程阶段,从具体的安全协议中转换代数的描述,这样可以利用推理技术来分析正在运行的安全协议。

1.1 适合于代码化的 SPI 演算的语法和语义

在 SPI 演算中,最基本的语法是项和进程。为了适合代码的生成,我们对其做了略微的改动,但并不对其分析构成影响。

项的定义如表 1。

表 1 SPI 演算的项的定义描述

L,M,N::=项	SK,SK′ SK 会话密钥
a,b,cN 名	$(M_1,M_2,\cdots,M_n)$ n 元组
x,y,zV 变量	$\{M\}_{SK}$ 会话密钥加密
A,B,CA 代理或主体	$\{[M]\}_{K^+}$ 公钥加密
No,No′No 新鲜值	$[\{M\}]_{K^-}$ 数字签名
$K^+,K^{+1}\in$Pub 公钥	Hash(M) 散列函数
$K^-,K^{-1}\in$Prk 私钥	

在标准的 PI 演算中,名是唯一的项。在 SPI 演算中,在安全协议描述中增加了对(M, N)来方便描述、发送数列、零、后继算子 suc(M),以说明产生的任意自然的数字。名一般可用来表示密钥或通道(仅被发送者和接收者所知)。为了和名区别,增加了变量 x 作为一个占位符来表示未知的值。当进程收到一个信息,这个未知的值就变成了已知的值,可以在语法上反映出动态通信的替代机制。为了说明加密信息,我们加入了基于共享密钥的加密$\{M\}_N$,项$\{M\}_N$表示用密钥 N 对项 M 加密的密文。如密钥可以使用基于共享密钥的加密算法(如 DES 等)、公钥(如 RSA 等)算法、数字签名和哈希函数等约定 $V\cap N\cap A\cap No\cap Pub\cap Prk\cap SK=\phi$

进程的定义如表 2。

表 2 进程的定义与描述

P,Q,R::=	进程	0 空进程
$\overline{M}\langle N\rangle$. P	输出	let(x,y)=M in P 拆对
m(x). P	输入	case m of 0:P suc(x): Q 分支
P|Q	复合	case L of$\{[M]\}_K^+$ in P 基于共享密钥的解密
(vn)P	限制	case L of$[\{M\}]_K^-$ in P 数字签名的认证
! P	复制	case T valid in P 时间戳或新鲜值的有效性
[m is n]P	匹配	

在(vn)P 中,P 的名 n 称为是受约束的名即新生成一个受限名 n,p 进程外的进程没有能力知道 n;在 M(x). P 中,P 中的变量 x 是受约束的,表示在通道中等待项的输入;在 case M of 0 : P suc(x) : Q 中变量 x 在第二个分支 Q 中是受约束的,称不受约束的变量(名)的出现是该变量(名)的自由出现。记进程 P 中所有自由出现的名所组成的集合为 fn(P),所有自由出现的变量组成的集合为 fv(P)。如果一个进程中没有任何自由出现的变量(即 fv(P)=φ),则称该进程是闭进程。记 P[M/x]为将 P 中所有 x 的自由出现用项 M 替换后所得到的进程,其中 $\overline{M}_{\langle N\rangle}$ 为 $\overline{M}_{\langle N\rangle}$. 0 的简写。输出进程 $\overline{m}\langle N\rangle$. P 表示通过通道 m 的一个输出,若有一个相互作用发生,那么项在通道 m 中通信,进程 P 运行;输入进程 m(x). P 表示进程准备从通道 m 输入,若一个相互作用发生,则 N 在 m 中发生通信,进程 P[N/x]运行; P|Q 表示进程 P 和 Q 并行运行,P 和 Q 可以在共知的通道中相互通信,也可以独立地和外界通信,! P 表示无限多个 P 进程的复制在并行运行;匹配[m is n]P 表示若项 M 和 N 相同,就运行 P,否则 P 阻塞;空进程 0 表示进程什么也不做;拆对 let(x,y)=M in P 表示若项 M 是对(N,L),则进程 P 执行 P[N/x][L/y],否则 P 阻塞;分支 case m of 0:P suc(x):Q 表示若项 M 为 0,则执行进程 P,否则若 M 是 suc(N),则运行 Q[N/x];基于共享密钥的解密 case L of$\{x\}_n$ in P 是 PI 演算中新增加的,它给我们一个解密的方法(并且检查项是不是正确的密文)表示进程试着用密钥 N 去解密项 L。若 L 是$\{M\}_N$形式的密文,则进程运行 P[M/x],否则进程阻塞。可以看出用以上的语法与语义可以方便地对安全协议的安全属性进行建模描述。

1.2 代码的产生的变化

为了方便代码的产生,我们对标准的 SPI 演算的语法做一点改变。对输出 $\overline{m}\langle N\rangle$. P 和输入 m(x). P 分别改变为定义在 prolog 语言中的"!"和"?",同时用 Pub(x),Priv(x)和 Hash(x)分别表示主体 x 的公钥部分、私钥部分以及主体的散列,对于新鲜值的有效性也进行了定义。

2 协议规范形式化描述的例子

为了说明描述过程,作为基于 SPI 演算安全协议规范的

例子，我们采用 Woo-Lam 单向认证协议。首先给出采用非形式化的表述，对其进行简单的分析，然后给出基于 SPI 演算的规范描述，并进行修改，以适宜可执行代码的产生。

2.1 非形式化描述 Woo-Lam 单向认证协议

Woo-Lam[9,10]是使用对称密钥和新鲜值的典型安全协议范例，该协议主要是对协议的发起者(Init)和接收者(Resp)的认证，并不对相反的过程协议认证，通过信任的服务器 S 来传递会话密钥、加密、解密密钥和新鲜值。该协议也称为大嘴蛙协议，其图示及非形式化描述如图 2 和表 3。

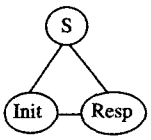

图 2 Woo-Lam 协议的模型

表 3 Woo-Lam 的非形式化描述

Message 1 Init→Resp ;Init8
Message 2 Resp→Init ;No
Message 3 Init→Resp ;Pub(Init)
Message 4 Resp→S ;Pub(Init),Pub(Resp),{Init,{N,K}}
Message 5 S→Resp ;Pub(Resp),{N,K}

其中 Init 和 Resp 分别为协议的发起者、接收者，N 为新鲜值，K 为会话密钥。

这种非形式化的描述，尽管看上去简单、直观，但并不能说明协议的行为。对其理解困难，在于对安全协议的实现理解，但并不是每个人都具有这样的经验。

相反，SPI 演算规范对同一个非形式化的安全协议给出了其精确的形式化描述。

2.2 形式化描述 Woo-Lam 单向认证协议

对于非形式化的 Woo-Lam 单向认证协议的描述，根据 SPI 演算的语法、语义给出如下形式化的描述：

$$Init(X,Y) \cong (\nu K_{XY})(\overline{C_Y}\langle X\rangle.\ C_X(x_{XY}).\ \overline{C_Y}\langle\{x_{XY},K_{XY}\}_{K_{XS}}\rangle.\ C_X(msg1'_{XY}).\ case\ msg1'_{XY}\ of\ \{msg1_{XY}\}_{K_{XY}}\ in\ F(msg1_{XY}))$$

$$Re\ c \cong (\nu N'_{XY})(\nu M'_{XY})(C_Y(z'_{XY})\ \text{if}\ z'_{XY}=X\ \text{then}\ \overline{C_X}\langle N'_{XY}\rangle.\ C_Y(x'_{XYB}).\ C_S\langle Y,\{X,x'_{XY}\}_{K_{YS}}\rangle\ C_Y(u'_{XY}).\ case\ u'_{XY}\ of\ \{w'_{XY}\}_{K_{YS}}\ i\ in\ let\ (r'_{XY},s'_{XY})=w'_{XY}\ in\ if\ r'_{XY}=N'_{XY}\ then\ \overline{C_X}\langle\{M'_{XY}\}s'_{XY}\rangle.\ F'$$

$$S \cong C_S(x_S).\ let(y,z_s)=x_s\ in\ case\ z_S\ of\ \{t_S\}_{K_{YS}}\ in\ let\ (x,u_S)=t_S\ is\ case\ u_S\ of\ \{r_{xy}\}_{K_{xs}}\ in\ let\ (v_{xy},w_{xy})=r_{xy}\ in\ \overline{C_y}\langle v_{xy},w_{xy}\rangle_{K_{ys}}\rangle$$

$$Protocol \cong (\nu K_{AS})!\ (Init\langle A,B\rangle \mid Init\langle A,I\rangle \mid Rec\langle B,A\rangle \mid Rec\langle I,A\rangle \mid S) \mid (\nu K_{BS})!\ (Init\langle B,A\rangle \mid Init\langle B,I\rangle \mid Rec\langle A,B\rangle \mid Rec\langle I,B\rangle \mid S) \mid (\mid (\nu K_{IS})(I \mid !S)$$

在式中，F 和 F' 分别为初始者和接收者的进程。

3 从 SPI 演算描述到具体实现代码之间的应用程序接口(API)

3.1 SPI 演算的 API

在实现应用程序接口(API)中，由图 3 知，把协议逻辑的执行实现和加密协议及网络通信分离开来，有利于安全协议规范的描述和安全协议与具体实现的转换。定义如下：协议逻辑是作为保持协议状态，密钥信息什么时候发送、是否发送及接收、输出信息是否是期望的信息、存储信息成分以及哪些成分需要代码的转换。

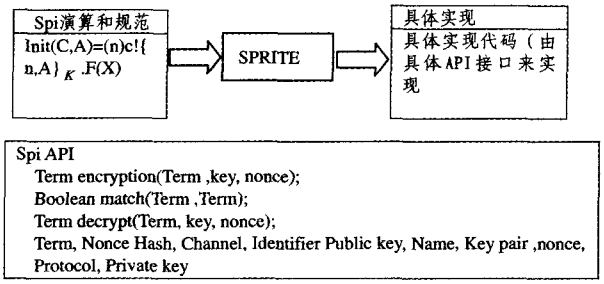

图 3 应用 Sprite 接口模型

3.2 SPI 进程到具体代码实现 Spritr

Sprite[11,12]是把网络安全协议具体实现的工具和环境。这个工具在安全协议和具体实现代码之间构建了一座桥梁，即在规范和具体实现之间形成了一种映射关系。这些映射的规范是由 Prolog[14]规则正确定义的，同时 Prolog 规则把 SPI 进程和类型与 Java 代码模板联系起来。Prolog 基于一个推理机对信息进行逻辑推理。除了在逻辑上发现给定问题的答案以外，它还可以处理备选方案且找到所有可能的解，而不只是一个解，是基于 Horn 子句书写程序，更容易表达为书面形式，比通常使用算法简单。Prolog 语法规则管理解析 SPI 进程，同时这些规则的规则从相应的代码模板中产生 Java[15,16]代码段。

Sprite 同时通过更新用户接口(UI)产生代码来追踪协议的进展和状态。通过跟踪定义的 SPI 进程，用户通过接口来控制协议主体的运行。为了清晰和方便，我们做出以下 Java 代码和 SPI 之间的映射关系：

输出，C! 〈N〉. P 与 void SPI. protocol. send(Spp. channel c,Spp. Term)相映射，其中 C 为通信的通道，N 为通信的项，这个进程的产生代码如下：

//C! 〈N〉. send(C,N);

其中参数 C 和 N 可以为 Java 变量或相对应于 Spp. channel 和 Spp. Term 的表达式的值。

输入，C? (X). P 与调用项 SPI. Protocol. Recv(Spp. channel)相对应，即信息通过通道 C 而产生一种新的项 Spp. Term，这个进程的代码产生如下：

C? (X). final〈Type〉x=getItem(c). unpack〈Type〉. (recv(c)));

其中〈Type〉是变量 X 的类型，并且是由规范中的变量类型所具体产生的。

受限，(Vn)一般是为协议中的新鲜值或者时间戳而创建的，与定义的类型 Spp. Nonce，Spi. Protocol. newNonce()或者 Spp. TimeStamp，Spi. Protocol. newTimeStamp()相映射。类型由与相对应的规范查找表所决定，nonce 类型的产生代码形式如下：

新鲜值，Nonce n=new Nonce()或者 TimeStamp n=new TimeStamp()

[M is N]. P 匹配，与一个返回状态相映射，这种状态调用 boolean Spi. protocol. match(Term，Term)的返回值所决定，其映射关系如下：

```
If(! match (m,n))
{return;
}
```

Let(x,y)=M in P 分支，该进程是从 SPI 项 M 相对应的实例 Spp. Term 中提取原始数据，项 M 被分裂成与 x 和 y 相对应的两项，且替代变量 x 与 y，即 P[L/x],p[N/y]。为了方便从提供者具体代码中实现、从原始数据中压缩和解压缩，约定拆对中的第一项为名或名变量，也就是必须是原子值。这种约定意味着在通信信道中发送者压缩和解压缩项时不需要存储额外的信息来保证信息的结构，因为这种分拆对可能是一任意深度的嵌套。如信息{A,B,C}pub(A)可由{(A,(B,C))pub(A)详细说明，该进程的映射关系如下：

```
Let(x,y)=M in P
Final〈Type〉x;
Final〈Type〉y;
{InputStream _is=new ByteArrayInputStream();
M. getDate();
X=getImpl(). unpack Nonce(_is);
Y=getTmpl(). unpack Identifier(_is);
}
```

其中〈Type x〉和〈Type y〉分别为变量 x 和 y 的类型。

解密，case L of $\{x\}_n$ in p 与输入流 SPI. protocol. decry (Spp. encryption,Spp. key)相映射。各调用方法可以由密钥的类型进行扩展，这种类型可以是对称的，公共的或者私有的。其映射关系如下：

Case L of $\{x\}_n$ in final 〈type x〉=getImple(). unpack 〈Type〉(decry(L,N);

4 Woo-Lam 的实现例子

为了说明 Sprite 的应用，我们从 Woo-Lam 的 SPI 演算规范产生具体代码。为了说明问题，我们详细地列出 Init 进程的产生码，Init 说明了协议发起者的作用。代码如下：

```
//Init(C,A,B)=
Public void Init(Channel C, Identifier A, Identifier B)
{ Nonce n=new Nonce();
  Send(C,encrypti(new pair(n,A),pub(B));
Encryption L=getImpl(). unpack Encryption (recv(c));
Term j=getImpl(). unpack Term(decrypt(l,priv(A));
Nonce x;final(Nonce m);
Input Stream_is=new Byte Array Input Stream (j. get Data());
X=get Impl(). unpack Nonce(_is);
Y=get Impl(). unpack None(_is);
}
If(! match(x,n))
Return;
Send(c,encrypt(m,pub(B)));
Return
}
```

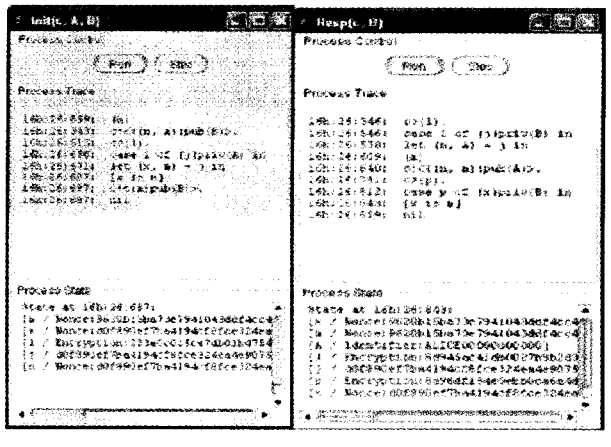

图 4　Sprite 验证运行结果

把以上程序代码运用 SPI2JAVA，该结果显示了 WOO-LAM 安全协议中并发运行的 Init 和 Resp 运行的结果(如图 4)[15~17]。结果分别描述了协议中的发起者、接收者及其变量变为实值的变化过程，它们随着变量被实例化而更新。可以看到，运行过以后，新鲜值 n 被暴露出来。这种情况说明该协议中若有攻击者进程，则攻击者进程可以利用新鲜值的漏洞，捕捉发送者的信息，对信息不做任何改动而重发一次信息。而发起者进程和接收者进程并不能发现攻击者进程的存在，这在一定的环境中会造成重大的损失。

结论　本文基于 SPI 演算对安全协议设计进行了建模，并应用 Sprite 在协议规范和具体实现代码之间构架了一座桥梁。应用这种方法，不但可以描述安全协议，并且可以验证安全协议属性是否满足。介绍了形式化描述语言 SPI 演算、逻辑语言 Prolog，列出了应用 Sprite 的 SPI 演算到 Java 代码的具体语法、语义映射，这种方法可以作为分析参考。最后，我们给出了大嘴蛙安全协议(Woo-Lam)的 SPI 演算形式化描述，并应用 Sprite 中的 SPI2JAVA 给出了具体的执行程序 Java 代码。我们计划进一步给出自动地正确映射的形式化机制，并将这些方法应用于 Web 服务组合的安全描述验证以及基于上下文的 Web 服务组合的安全描述验证。

参 考 文 献

1 Gu Yonggen, Fu Yuxi. A Simple Process Calculus for the analysis of Security Protocals. In: Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference, Dec. 2005. 110～114

2 李国强，顾永跟. 基于 SPI 演算的 Kerberos 认证协议形式化研究[J]. 计算机科学，2004(11)

3 赵宇，王亚弟. 基于 SPI 演算的 SSL3.0 协议安全性分析[J]. 计算机应用，2005(11)

4 顾永跟. 基于类 pi 演算的电子支付协议安全性形式化研究. 计算机应用研究，2006(3)

5 Sumii E, Pierce B C. A bisimulation for type abstraction and recursion. ACM SIGPLAN Notices, 2005, 40(1)

6 Haack C, Jeffrey A. Timed SPI-calculus with types for secrecy and authenticity. Lecture Notes in Computer Science, 2005

7 Stark I. A fully abstract domain model for the π-calculus[J]. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 1996. 36～42

8 卿斯汉. 安全协议 20 年研究进展[J]. 软件学报，2003(10)

9 Abadi M, Gordon A D. A calculus for cryptographic protocols: The SPI calculus[R]: [Technical Report]. Digital Systems Research Centre, 1998

10 Woo T Y C, Lam S S. Authentication for distributed systems [J]. Computer, 1992, 25(1): 39～52

11 Crazzolara F. Language, Semantics, and Methods for Security Protocols[D]: [PhD thesis]. 2003

12 Salaun G, Bordeaux L. Describing and Reasoning on Web Services Using Process Algebra[J]. In: 2nd International Conference on Web Services, 2004

13 http://people. cs. uct. ac. za/～btobler/sprite/

14 Milner R, Parrow J, Walker D. A calculus of mobile Processes [J]: [Technical Report]. 86, 1989

15 Sun Microsystems. Java 2 Enterprise Editon. http://java. sun. com/j2ee

16 Pozza D, Sisto R, Durante L. SPI2Java: automatic crytographiv rotocol Java code generation from SPI calculus [J]. In: Advanced Information Networking and Aplications, 2004. AINA 2004. 18th International Conference on, 2004, 1: 400～405