doi:10.11959/j.issn.1000-436x.2017208

SSMCI：以攻击者为中心的安全协议验证机制

谷文，韩继红，袁霖
（解放军信息工程大学三院，河南 郑州 450004）

摘 要：提出一种能对安全协议进行分析的自动化验证机制。提出需求的概念，认为需求是攻击者未知但又对攻击者合成目标项至关重要的知识集合，并建立了以需求为中心的攻击者模型；设计一种以攻击者为中心的状态搜索方式，按需添加协议会话实例，为避免新增协议会话实例引起攻击者推理出现时序矛盾，引入回溯机制以确保状态转移过程中攻击者知识能正确增长。实验表明，该系统能正确验证协议的安全性，状态空间数目略优于 Scyther 工具。

关键词：安全协议；攻击者模型；状态扩展
中图分类号：TP393.08 **文献标识码**：A

SSMCI: verification mechanism for security protocols centered on the attacker

GU Wen, HAN Ji-hong, YUAN Lin
(The Third College, PLA Information Engineering University, Zhengzhou 450004, China)

Abstract: A automatic verification mechanism for security protocols analysis was proposed. The attacker model was proposed and the concept of ‘need’ was defined, a knowledge set which was necessary for the attacker to compose a target message term but unknown to the attacker. The attacker model was established as needed. The mechanism centered on the attacker was designed, in which whether add a protocol session was determined by the attacker. This might cause contradiction in time sequence, so some back-track algorithm was adopted to solve this contradiction. Experiments show that the system can verify the security of the protocol, and the number of state space is slightly better than the Scyther tool.

Key words: security protocol, attacker model, state expansion

1 引言

在网络安全体系中，安全协议对于确保通信和数据安全具有重要作用。随着互联网的发展，安全协议的设计日趋复杂，安全协议分析技术愈发重要。近年来，安全协议形式化分析研究受到国内外研究人员越来越多的关注，并取得了很大的研究进展，一些成熟的形式化分析方法也在实际网络环境中得到应用。在这些方法中，基于模型检测的安全协议验证方法取得了极大的成功。但仍然发现不少协议中已知或未知的漏洞，该方法存在状态空间爆炸的问题。为解决基于模型检测的密码协议安全性验证方法中出现的状态空间爆炸问题，可从 2 个方面着手，一方面采用模型检测领域中的通用状态缩减技术；另一方面针对密码协议和攻击者的特点，设计能反映密码协议运行特征的状态表示方法和状态搜索算法。

在第一个方面，模型检测领域中有多种方法可以缓解状态空间爆炸问题，包括削减技术、符号技术和少量状态搜索技术等。

1) 削减技术。通过寻找程序行为过程中的等价关系，只处理每一等价类的一个成员，可削减大量状态，并且，这样的削减是完整的，主要有偏序削减、对称削减等技术。

收稿日期：2016-12-13；**修回日期**：2017-06-06
基金项目：国家自然科学基金资助项目（No.1309018）
Foundation Item: The National Natural Science Foundation of China (No.1309018)

2) 符号技术。符号化的模型检测技术是以状态集合为执行对象而非单个独立的某种状态。符号化的表示方式使系统状态表示更为简便，能更好地表示无限状态集合。

3) 少量状态搜索技术，通过限制搜索的深度或广度来缩小状态空间。

在第二个方面，Guttman[1]提出了串空间模型，它是一种结合定理证明和模型检测的混合方法，串空间模型准确利用了协议里消息收发节点之间的因果关系使证明更为简洁；文献[2]提出的 Athena 方法，采用一个状态来表示参与者个体交错执行的顺序，这种状态结构有效地避免了状态空间因为异步合成而产生的指数型增长。Scyther[3]在协议模型的设计上借鉴了串空间的思想，在搜索算法上也吸收了 Athena 算法的精髓，并依据协议运行的特点设计了多种启发式的搜索规则，状态空间数目较少。Scyther 工具是由 Cremers 团队开发的一款自动化协议分析器。在协议模型方面，其借鉴了串空间的思想，在搜索算法上也吸收了 Athena 算法的精髓，并设计了多种启发式的搜索规则，可以看作串空间和 Athena 的传承。该工具界面直观，操作简单，效率也极高，文献[4]比较了包括 Avispa 4 个终端工具[5~7]、Proveif 和 Scyther 等在内的 15 种协议自动化工具，Scyther 表现优秀。

本文设计的协议安全自动化验证机制基于模型检测方法，也将从这 2 个方面着手缩减状态空间，特别要针对密码协议和攻击者的特点，设计能反映密码协议运行特征的状态表示方法和状态搜索算法，主要解决以下 3 个问题。

1) 协议形式化模型

协议模型决定了分析方法的适用性。对协议进行统一描述，准确表达密码协议的消息结构、被分析协议的计算过程和通信过程等。协议形式化模型在第 3 节介绍。

2) 攻击者建模

攻击者的描述决定了分析结论的可靠性，攻击者模型能反映攻击者消息计算和网络控制的能力。攻击者建模在第 4 节介绍。

3) 状态搜索机制

状态搜索机制决定了分析方法的可用性。设计高效的状态搜索算法，主要内容包括状态定义、初始状态设定、状态转移规则设定、无效状态判断、终止条件判断等 5 个方面。状态搜索机制在第 5 节介绍。

2 相关工作

2.1 协议形式化模型

对协议进行安全性分析的前提是建立形式化模型来描述协议。目前，已提出了不少协议描述模型，包括模态逻辑模型、进程代数和多重集重写模型等。

模态逻辑模型首先根据密码协议消息交互规程的一般性质构造一组推理规则，再利用对密码协议运行环境的假设和加密算法的性质确立一组信任关系，从而最终实现对密码协议安全性的推理。

进程代数是指采用了代数方法研究通信并发系统的理论，常见的有 CSP 通信进程模型、CCS 通信系统演算模型和 π 演算等[8]。

多重集重写模型中，协议被视为角色的集合，这些角色通过交换消息来达到通信的目的。每个角色都由一个实际的主体（如参与的诚实主体、服务器、攻击者等）来扮演，当收到所期望的消息后，按照规则要求进行回复。

2.2 攻击者模型

Dolev-Yao 模型是一个广泛应用的攻击者模型，它有效地界定了攻击者的行为能力，即攻击者可以读取、修改网络中的消息，创建消息发送至网络中。它采用一阶逻辑的形式描述攻击者监听、插入、合成、分解、解密、加密等各方面能力。

当前，大部分工具的攻击者模型都基于 Dolev-Yao 模型，该模型下攻击者能通过运用加密、解密、链接及分拆等 4 种基本操作合成新的消息。其能产生的消息数目是无限的，如攻击者可以无限次对一个消息进行加密、链接。一些攻击者模型试图限制消息的规模，但是这些限制是针对某一特定协议而不是通用的。文献[9]认为没有必要记录所有的消息项，因为有些消息项的子项是共享的，将消息项分解成为子项来记录可有效减少攻击者知识的数目。Chevalier 等[10]采用了惰性的方法，不再试图构造出攻击者的消息空间，而是测试其能否生成消息以匹配目标消息项。

2.3 状态搜索机制

状态搜索主要是基于经典深度优先或广度优先算法而改进的。状态搜索机制中最重要的是对状态进行定义，一旦状态的定义确定，转移规则、终止条件等也随之确定。状态通常定义为 5 元组 $M=(Q,\Sigma,\partial,q,F)$，$Q$ 为状态集合，Σ 为输入集

合，通常为协议主体的接收发送消息的动作，∂ 为状态转移函数，依据输入数据进行状态转移，q 为初始状态，F 为可实现状态集合。

3　协议形式化模型

协议模型是对密码协议进行分析验证的基础。2.1 节中介绍的协议形式化模型在描述协议的各个角色时，选择的角度是不一致的，如描述某协议的发起者时，是在发起者的角度对其各个参数、收发动作及检验过程进行描述的；描述协议的响应者时，又是从响应者的角度描述上述信息的。从角色各自的角度对参数等信息进行定义并非不可，但是在此基础上设计的状态搜索算法难免会引起参数混淆、逻辑复杂等问题。状态搜索算法处理协议响应者角色的动作时，要在该角色的角度进行处理；处理协议响应者角色的动作时，又要切换为这一角色的角度。

基于上述考虑，建立一个攻击者角度下的协议模型，反映协议消息的字段组成和消息交互过程，主要从项、事件、角色这 3 个层次对协议进行了描述。

定义 1　消息项 $Term$。

$$Term ::= BasicTerm|(Term)_{Term}$$
$$(Term \,||\, Term)|Func(Term,Term,\cdots)$$

协议中的消息用消息项表示。消息项分为基本项和复合项，基本项包括密钥、主体名和随机数等消息项，复合项由基本项通过加密、链接和函数变化(签名、散列等操作)等操作形成。

定义 2　项类型 $Type$。

$$Type ::= BasicType|CompType$$
$$BasicType ::= agent \mid fresh \mid key$$
$$CompType ::= con \mid crypt \mid func$$

项类型分为基本项类型 $BasicType$ 和复合项类型 $CompType$，基本项类型包括主体名 $agent$、随机数 $fresh$ 和密钥 key。$sym(p_1,p_2)$ 表示主体 p_1 和 p_2 的共享密钥，$pubk(p)$、$prik(p)$ 分别表示主体 p 的公钥和私钥；复合项类型包括链接项 con、加密项 $crypt$ 以及函数项 $func$。

消息项中常会包含一些变量，协议验证过程中变量的用途有 2 种，一是用于存放主体未知的参数（通常由其他主体生成），二是依据符号化思想用于表示一类参数以减少状态空间。用变量集合 $V(t)$ 表示消息项 t 中所有的变量，$type(t)$ 表示消息项的类型。

定义 3　子项关系 $\subseteq$。若存在 $t_1,t_2 \in Term$，则有 $t_1,t_2 \subseteq t_1||t_2,t_1,t_2 \subseteq (t_1)_{t_2},t_1,t_2 \subseteq Func(t_1,t_2),t_1 \subseteq t_1$。

定义 4　项的分解树 $Tree(m)$。

依据项和子项的定义，项可以分层有序展开，形成项的分解树，树的每一个节点对应项的一个子项。以 $(na\#1\,||\,a)_{pubk(b)}$ 为例，其分析树如下：第 0 层为 $(na\#1\,||\,a)_{pubk(b)}$，第 1 层为 $pubk(b)$ 和 $na\#1\,||\,a$，第 2 层为 $na\#1$ 和 a，如下所示。

第0层　$(na\#1\,||\,a)_{pubk(b)}$

第1层　$(na\#1\,||\,a)$　$pubk(b)$

第2层　$na\#1$　a

本文假设攻击者能够完全控制网络环境，能阻断、转发、监听消息，主体发送的消息不论目的方是否为攻击者，都可视为攻击者接收，主体接收的消息都可视为攻击者所发送。因此，从攻击者的角度来看，网络中主体的行为有发送和接收消息这 2 种。

定义 5　事件 $Event$。

$$Event ::= send(Re,Term) \mid recv(In,Term)$$

$send(Re,Term)$ 表示主体 Re 发送消息项 $Term$；事件 $recv(In,Term)$ 表示主体 In 接收消息项 $Term$。

用 $Msg(e)$ 表示事件 e 发送或接收的消息项，$type(e)$ 表示事件的类型。事件之间存在前序关系 $\Rightarrow$，若 e_1 在 e_2 之前发生，则有 $e_1 \Rightarrow e_2$;设 $e_1 \cdot e_2 \cdot e_3$ 表示事件 e_1、e_2 和 e_3 按时间关系构成事件偏序序列。e_1 之前的事件集合记为 $before(e_1)$，之后的事件集合记为 $after(e_1)$。

定义 6　角色 $Role$。

$$Role = \{(\{m\},\{s\}) \mid m \in Term \wedge s \in Event\}$$。

角色由角色知识 $\{m\}$ 和事件序列 $\{s\}$ 组成，$\{s\}$ 定义了角色应该执行的事件及其顺序关系。$pro(agent)$ 表示 pro 协议中主体为 $agent$ 的角色，以 NS 协议为例，发起者 $Init$ 的角色（所有参数都为变量）定义为

$$NS(Init) = (\{Init,Resp,Na,pubk(Resp),$$
$$prik(Init),pubk(Init)\},$$
$$send(Init,(Na\,||\,Init)_{pubk(Resp)})\cdot$$
$$recv(Init,(Na\,||\,Fresh)_{pubk(Init)})\cdot$$
$$send(Init,(Fresh)_{pubk(Init)}))$$

定义 7 置换 $\partial=\{x/x'\}$。设 $t\in Term, x\subseteq t$，则 $t\circ\partial=t\circ[x/x']$ 表示将 t 中所有的变量子项 x 替换为 x'。

定义 8 实例因子 $Inst=[r,rid,\partial]$。

实例因子中 r 表示主体，rid 表示主体的运行轮次，∂ 为置换集合。随着验证过程的深入，实例因子中置换集合 ∂ 不断增大。攻击者实例因子表示为 $[I,0,\partial]$，I 为攻击者主体名。

实例因子可作用于事件，设 $Inst=[r,rid,\partial]$，当其作用于事件时，事件实例化为

$$Inst\circ send(Re,Term)=send(r^{\#rid},Term\circ\partial)$$

$$Inst\circ recv(In,Term)=recv(r^{\#rid},Term\circ\partial)$$

协议中最常见的关系便是主体之间消息的收发关系，本文称之为通信关系。

定义 9 通信关系 $\rightarrow$。

设 $\exists e_1,e_2,Inst_1=[r_1,rid_1,\partial_1],Inst_2=[r_2,rid_2,\partial_2]$，满足 $Msg(e_1)\circ\partial_1=Msg(e_2)\circ\partial_2$，且有 $type(e_1)\neq type(e_2)$，则 e_1 和 e_2 存在通信关系，记为 $e_1\rightarrow e_2$。寻找通信关系的过程称之为绑定。

定义 10 角色实例 $roleInst$。

角色实例由实例化的角色知识和事件组成。

$$roleInst=\{Role\circ Inst\}=$$

$$\{(\{m\circ Inst\},\{s\circ Inst\})\mid m\in Term\wedge$$

$$s\in Event\wedge V(m\circ Inst)=\varnothing\}$$

实例因子当其作用于角色时形成角色实例，表示主体的一次具体的运行。设角色实例 r 为主体 a 第 rid 次的运行，则 r 记为 $rInst(a^{\#rid})$，第 i 个事件表示为 $<r,i>$，该角色实例的事件集合可表示为 $Event(r)$。对于角色实例 $r=\{Role\circ Inst\}$，其中，事件 $<r,i>$ 受到实例化因子 $Inst_1$ 作用，则角色实例 r 更新为 $r=\{Role\circ Inst\circ Inst_1\}$。

定义 11 消息模板 $Template$。

$Template$ 是符合协议规定的消息格式和结构的消息项集合。协议交互过程中的所有消息可看成消息模板中消息的实例化，消息的实例化可分为两种情况：一种是角色实例的实例化，当新增一个角色实例时，角色的扮演者、密钥及其使用的随机数将会实例化；另一种是构建通信关系的实例化，将会对角色中的变量包括其他角色使用的主体名、随机数、密钥等进行实例化。

4 基于需求的攻击者模型（IMBN）

网络中的攻击者一方面可以接受主体发送的消息，对这些消息进行分析处理；另一方面，攻击者可以构造一些消息给合法主体，以达到攻击效果。据此，本节从攻击者接收和攻击者发送 2 个部分构建了攻击者模型。攻击者接收描述了攻击者对收到消息的分解能力、存储方法，攻击者发送描述了攻击者合成目标消息的能力。

4.1 攻击者接收

攻击者接收要解决的问题可描述如下：给定一个项集合 T（T 表示攻击者的知识），以及攻击者接收的消息项 s，如何计算出攻击者新增的知识集合 S。

攻击者知识由攻击者初始知识和增长的知识组成，是一个动态变化的消息项集合。攻击者知识是攻击者分解消息、合成消息的基础，在自动化验证算法中，许多过程都需遍历攻击者知识集合，这就要求集合的大小不能过大。实际上，知识集合是无限的，因为攻击者可以无限次运用链接、加密、签名、异或等函数操作生成新的知识。考虑到消息项中存在大量重复的子项，为精简攻击者知识集合，可将消息分解为子项进行存储，并去除重复的子项。

攻击者初始知识用符号 AtK_0 表示，一般包括协议参与者的主体名集合、公钥集合、攻击者的私钥及与其他主体的共享密钥等，该集合中消息项之间没有重复的子项。

定义 12 知识添加集合 $addTerm_{AtK}(m)$。表示知识集合为 AtK 时，处理消息项 m 得到的知识集合。

为精简攻击者知识集合，本文制定了攻击者知识添加规则 R1~R5，去除相同的不可分解子项。

$$R_1: m\notin Atk\wedge type(m)=BasicType$$

$$\rightarrow addTerm_{AtK}(m)=\{m\}$$

$$R_2: m\notin Atk\wedge m=t_1\,||\,t_2\rightarrow addTerm_{AtK}(m)$$

$$=addTerm_{AtK}(t_1)\bigcup addTerm_{AtK}(t_2)$$

$$R_3: m\notin Atk\wedge m=(t_1)_{t_2}\wedge t_2^{-1}\in Atk$$

$$\rightarrow addTerm_{AtK}(m)=addTerm_{AtK}(t_1)$$

$$R_4: m\notin Atk\wedge m=(t_1)_{t_2}\wedge t_2^{-1}\notin Atk$$

$$\rightarrow addTerm_{AtK}(m)=m$$

$$R_5: m\notin Atk\wedge m=Func(t_1,t_2)$$

$$\rightarrow addTerm_{AtK}(m)=m$$

注意到消息项 m 中可能包含变量，消息项的变量表示主体待接收的参数，而攻击者知识集合中所有的知识都应是确定的，这种情况下，$addTerm_{AtK}(m)$ 不能加入攻击者知识集合，Atk 表示攻击者知识。实际上，攻击者在处理消息项 m 之前，必存在通信关系

使该变量得以实例化，这将引起状态回溯，在第 5 节继续讨论。

定义 13　攻击者知识集合 AtK 。

当 AtK 中所有的消息项不包含任何变量，攻击者知识更新。当处理消息 m 时，更新后知识集合为 AtK 为

$$AtK = AtK \bigcup addTerm_{AtK}(m),\ V(m) = \varnothing$$

4.2　攻击者发送

攻击者发送模块主要解决 2 个问题：1)攻击者目标推理，判断攻击者能否基于攻击者推理规则和现有知识生成接收事件中的消息项；2)若攻击者不能合成接收事件中的消息项时，进一步判断攻击者能否通过某种途径获取特定新知识以合成目标项，称之为攻击者需求推理。

定义 14　符号 $\mapsto$ 。设消息集合为 IK ，m 为消息项，IK 中的元素若能通过加密、散列、链接等操作合成消息 m ，则记为 $IK \mapsto m$ 。

问题 1　设攻击者知识集合为 AtK ，m 为目标消息项，判断 $AtK ? \mapsto m$ 。

同经典的一阶逻辑推理系统一样，首先，攻击者将目标消息项 m 分解为树的结构，逐层判断是否为目标项，若某一层中，所有的节点都在攻击者知识集合中，则攻击者可合成消息项 m ，否则不能合成消息项 m 。以 $(na\#1 \| a)_{pubk(b)}$ 为例，攻击者知识为 AtK_0 ，$na\#1 \notin AtK_0$ 。第 0 层为 $(na\#1 \| a)_{pubk(b)}$ ，不在攻击者知识中，攻击者不能合成目标项；第 1 层为 $\underline{(na\#1 \| a)}$ 和 $pubk(b)$ ，由于 $\underline{(na\#1 \| a)} \notin AtK_0$ ，攻击者不能合成目标项；第 2 层为 $na\#1$ 和 a ，$na\#1 \notin AtK_0$ ，攻击者不能合成目标项，如下所示。

$$\text{第0层}\quad \underline{\qquad (na\#1 \| a)_{pubk(b)} \qquad}$$

$$\text{第1层}\quad \underline{(na\#1 \| a) \quad pubk(b) \in AtK_0}$$

$$\text{第2层}\quad \underline{na\#1 \quad a}$$

当 m 包含变量时，设该变量为 V_1 ，若 $type(V_1) = agent$ ，可认为 $V_1 \in AtK_0$ ，因为攻击者初始知识包含所有的主体名;若 $type(V_1) = fresh$ ，则认为 $V_1 \notin AtK_0$ 。

问题 2　攻击者不能合成目标项 m ，判断能否通过某种途径获取特定新知识以合成该目标项。该问题包括 2 个子问题：1)求解特定新知识；2)求解获取特定新知识的途径。

本文把特定的新知识集合称为需求，需求是攻击者未知的知识集合，对攻击者合成目标项至关重要。首先给出一个可读函数 $read$ 的定义。

定义 15　可读函数 $read$ 。对于消息项 $read(m)$ ，若 $m = t_1 \| \cdots \| t_n$ ，则 $read(m) = \{t_1, t_2, \cdots, t_n\}$ ，否则 $read(m) = m$ 。

定义 16　需求 $core(m)$ 。

设 $AtK \nmapsto m$ ，若 $core(m)$ 满足以下条件

1)　$Atk \bigcup core(m) \mapsto m$

2)　$Atk \not\mapsto m$

3)　$\forall t \in core(m), Atk \bigcap addTerm_{Atk}(t) = \varnothing$

则称 $core(m)$ 为攻击者关于消息 m 的需求。

在上述需求定义中，条件 1)说明攻击者获得 $core(m)$ 后，可以推导出消息 m ，条件 2)和条件 3)说明 $core(m)$ 缺少任何元素，攻击者都无法推导出消息 m 。

定义 17　运算 $\times$ 。

设集合 $\{n_1, n_2, \cdots, n_k\}$ 和 $\{m_1, m_2, \cdots, m_l\}$ ，规定运算 $\times$ 满足

$$\begin{aligned}&\{n_1, n_2, \cdots, n_k\} \times \{m_1, m_2, \cdots, m_l\}\\ &= \{\{n_1, m_1\}, \{n_1, m_2\}, \cdots, \{n_1, m_l\}, \{n_2, m_1\},\\ &\quad \{n_2, m_2\}, \cdots, \{n_2, m_l\}, \cdots, \{n_k, m_1\},\\ &\quad \{n_k, m_2\}, \cdots, \{n_k, m_l\}\}\\ &\quad \{n_1, n_2, \cdots, n_k\} \times \{\varnothing\} = \{n_1, n_2, \cdots, n_k\}\end{aligned}$$

基于运算 $\times$ ，可以给出消息项 n 需求的递归求解式。

若 n 为基本项，如果 $n \notin Atk$ ，则有 $\{core(n)\} = \{n\}$ ；否则 $\{core(n)\} = \varnothing$ 。

若 n 为复合项，设消息项 n 的下一层子项为 $n_1, n_2 \cdots$ ，如果 $Atk \bigcap addTerm_{Atk}(n) = \varnothing \wedge n \notin Atk$ ，则 $\{core(n)\} = \{n\} \bigcup \{core(n_1) \times core(n_2) \times \cdots\}$ ，否则 $\{core(n)\} = \{core(n_1) \times core(n_2) \times \cdots\}$ 。

定理 1　按上述递归求解需求集合 $\{core(n)\}$ 是的正确的。

证明　对于 $a \subseteq n \wedge a \notin AtK$ ，设 L 为 $Tree(n)$ 从顶点到叶子节点 a 的分支，深度为 m ，L_i 表示 L 第 i 层的节点，设集合 Q 满足 $Q \bigcup a \mapsto n$ ，$\{L_0, L_1, \cdots, L_m\}$ 通过运算 $\{L_0, L_1, \cdots, L_m\} \times Q$ 可先对需求集合完成了一次划分，但是这种划分是存在冗余的，如

$$L_m = na\#1, L_{m-1} = na\#1 \| a$$

依据需求定义的条件 3)，L_{m-1} 是冗余的。因此，需要判断 $Atk \bigcap addTerm_{Atk}(L_i) = \varnothing \wedge L_i \notin Atk$ 。

剔除 L 中不符合要求的分支节点后，该划分仍是完整的。因此，按照上述递归式求解需求，得到

的需求集合 $\{core(n)\}$ 是正确的，证毕。

据此递归关系，可求解出需求集合 $\{core(n)\}$，下面给出项 $(na\#1\|a)_{pub(b)}$ 需求的实例，其中，子项 a 和 $pub(b)$ 为主体名，属于攻击者知识， $na\#1$ 为随机数，不属于攻击者知识。

$$\begin{aligned}&\{core((na\#1\|a)_{pub(b)})\}=(na\#1\|a)_{pub(b)}\cup\\&(na\#1\|a)_{pub(b)}\cup\{core(na\#1)\times core(a)\}\\&=(na\#1\|a)_{pub(b)}\cup na\#1\end{aligned}$$

项 $(na\#1\|a)_{pub(b)}$ 的需求有 2 个，分别是项 $na\#1$ 和项 $(na\#1\|a)_{pub(b)}$。

攻击者获取新知识的渠道有作为合法主体参与协议获取知识，以及被动监听其他主体会话获取知识，可用事件 $recv(I,t)$ 来描述，它代表攻击者 I 接收了消息 t（不论目的方是否为攻击者），网络中协议消息数目很多，但格式有限，消息 t 可看作特定的实例化因子作用于消息模板的结果。

定义 18 解空间 S。

$$\begin{aligned}&S=\{recv(I,t_1)\,|\,t_1=t\circ Inst\wedge AddTerm(t_1)\cup\\&Atk\mapsto core(m),t\in Template\}\end{aligned}$$

消息模板是一个确定的有限集合，攻击者遍历消息模板中的消息，判断收到该消息后，能否得到集合 $core(m)$。求解过程中，需对消息模板中的部分变量进行实例化，而这其中密钥的实例化至关重要，依据攻击者添加知识的规则，当实例化为攻击者密钥（私钥和对称密钥），则能进一步对消息项进行分解，否则分解停止。当密钥被实例化为攻击者密钥时，攻击者作为合法主体参与协议获取知识，否则被动监听其他主体会话获取知识。

设攻击者已知的知识集合为 T，目的消息项为 g，需求为 U，攻击者推理即是这样一个问题，已知 T 和 g，求解 U，使 $T\cup U\mapsto g$，可分为以下 3 种情况。

1) 若 $U=\varnothing$，则 $T\mapsto g$，即攻击者基于已知的知识集合可合成消息 g。

2) 若 $U\neq\varnothing\wedge T\not\mapsto g$，即攻击者要合成消息 g，不仅要利用 T 中的知识，也要利用 U 中的知识。

3) 若 $U\neq\varnothing\wedge T\not\mapsto g\wedge U\mapsto g$，即攻击者完全基于 U 合成消息 g'。

据此可知，每一个 U 对应一种攻击者合成消息 g 的方式，每一种合成消息 g 的方式也对应一个 U。

5 状态搜索验证机制

5.1 总体思路

本文通过对前人工作总结，主要从以下 3 个方面着手优化状态空间。

1) 验证次序和网络时序

假设协议系统中协议事件运行的顺序为 $e_1\cdot e_2\cdot e_3\cdot e_n$，当本文对协议进行验证时，首先并不清楚协议的执行顺序，假定协议分析器处理事件的顺序是依据其验证机制的，因此，协议验证过程中可能就会出现 $e_3\cdot e_2\cdot e_1\cdot e_n$。如图 1 所示，实际网络时序消息 m_1 先于消息 m_2 出现，而协议验证次序先处理消息 m_2 再处理消息 m_1，处理消息 m_2 过程中，攻击者学习了新知识并用该新知识对消息 m_1 进行推理，但这违反了消息 m_1 先于消息 m_2 出现的时间逻辑，Brutus[11]、ProVeif[12]工具中存在错误的攻击路径，其原因正是如此。这种不遵循协议实际执行过程中时序约束的状态搜索算法，会浪费大量资源在这些无意义的扩展分支。

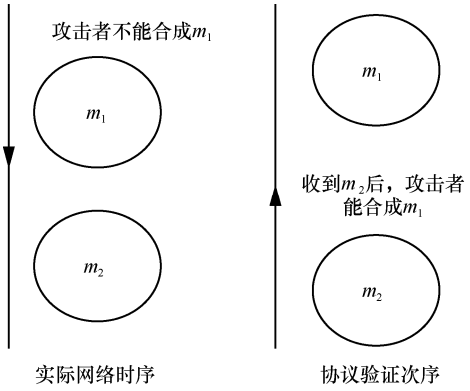

图 1 时序矛盾示例

2) 初始状态与最终状态集合的关系

在状态搜索算法的第一步是设定初始状态。在一些工具中，如 AVISPA 系列工具[13~15]，会先设定协议会话场景，如验证 NSPK 协议时设定为 3 个主体之间的平行会话，初始状态包含 3 个角色实例；另外一些工具不设定协议会话场景，如在 Athena 和 Scyther 工具中，初始状态仅包含一个角色实例，采用目标绑定或节点绑定的方法，增加角色实例，这种方法得到的最终状态即是一个协议会话场景。从不同的初始状态出发进行状态搜索，得到的最终状态集合是不同的，为确保状态空间的完备性，这些工具会设定多个初始状态。但是本文发现这些最

终状态集合之间是存在交集的，这意味着状态搜索过程中会有大量的中间状态和搜索分支是重复的，浪费了系统资源。

如果可以依据待验证协议的安全属性设定初始状态，使从该初始状态出发搜索得到的最终状态集合恰好可以验证协议的安全属性。那么，验证协议某个安全属性只需遍历某个特定状态空间集合，没有必要遍历整个状态空间，可有效缩减状态空间的规模。

3) 状态转移与攻击者的意图

假如在实际网络中，主体 a 与 b 进行一轮协议会话，但该轮会话中的所有消息都对攻击者的攻击毫无用处。从攻击者的角度来说，这轮会话是可有可无的。但是对于验证过程来说，验证状态空间随着协议会话数目增加呈指数级增长，该轮会话会增加验证状态空间的数目。本文在第 3 节和第 4 节建立了协议形式化模型和基于需求的攻击者模型，在此基础上可以统一角度，设计一种以攻击者为中心的状态搜索方式，即由攻击者依据其需求决定与哪些主体通信，主体接收哪些消息，截获哪些消息，更新状态。同 Athena 和 Scyther 一样，这种搜索方式从机理上避免了异步合成而产生的指数型增长，并且这种以攻击者为中心的状态搜索方式可有效减少一些无益于攻击者攻击的搜索分支。

5.2 状态表示

定义 19　状态 $State$。

$$State=\{RI,E,pair,tE,tpair,Atk\}$$

其中，RI 为角色实例集合；E 为待处理事件序列，由 RI 中角色实例的事件组成，且符合规范时序; $pair$ 为通信关系序列，其顺序可以反应协议执行轨迹；tE 为事件集合，用于存放与 E 中事件时序关系不确定的事件；$tpair$ 为通信关系集合，用于存放不能加入 $pair$ 中时序关系不定的通信关系；Atk 表示攻击者知识。

RI、E、$pair$ 和 Atk 这 4 个集合与其他学者定义的状态相似，反映了角色实例数目、攻击者知识变化和协议交互轨迹; tE 和 $tpair$ 这 2 个集合用于支持回溯机制。下面介绍状态中各个参数的变化条件。

1) RI 角色实例的添加

当攻击者的需求需从新的协议会话实例中 $send$ 事件获取时，新增角色实例到 RI 中。

2) E 中事件的处理

添加角色实例到 RI 中，E 也会添加该角色实例的事件，E 中的事件通过以下规则确定时序关系。

① 同一角色实例的事件顺序关系由协议规范确定。

② 具有通信关系的事件，$send$ 事件总是在 $recv$ 事件之前发生的。

③ 攻击者推理是一个逆向推理过程，它依据 $recv(In,Term)$ 逆向推理，得到解空间 S，假设 S 不为空，设 $recv(I,t)$ 事件为解空间 S 的解，则实际事件顺序为 $recv(I,t)\cdot send(I,Term)\cdot recv(In,Term)$，它表示攻击者收到消息 t 后合成消息 $Term$ 并发送，并为主体 In 所接收，故事件 $send(I,Term)$，$recv(I,t)$ 及其 $before(recv(I,t))$ 在事件 $recv(In,Term)$ 之前发生。

若不能通过确定时序关系，则加入 tE 中。当 $e=E.pop()$，事件 E 被选中后，删除该事件。

3) tE 中事件的处理

在状态搜索过程中，有些事件在某一状态下不能与 E 确定先后关系，但可能在另一状态下能确定时序，将暂时未能确定与 E 时序关系的事件加入 tE 中；而一旦满足上述 3 条规则，则必须将其从 tE 中能确定时序关系的事件剔除。

4) $pair$ 中通信关系的处理

当通信关系 $send(Re,Term)\rightarrow recv(In,Term)$ 形成时，若 $before(send(Re,Term))\wedge tE=\varnothing$，则可加入 $pair$ 中；否则，加入 $tpair$ 中。

5) $tpair$ 中通信关系的处理

攻击者推理过程中，形成了通信关系 $send(I,Term)\rightarrow recv(In,Term)$，由于事件 $send(I,Term)$ 仍有前驱事件，这一通信关系可能引起时序矛盾，故将 $send(I,Term)\rightarrow recv(In,Term)$ 加入 $tpair$ 中。

定义 20　可实现状态 $Ultstate$。

设 $s\in State, s.E=\varnothing$，则 $s\in Ultstate$。

当一个状态中 $E=\varnothing$，意味着没有驱动状态搜索的事件了，即该状态达到了终止状态，在可实现状态下验证协议的安全属性是否满足。

5.3 初始状态设定

可实现状态依据可包含 RI 的组成成分进行分类，设待验证的协议共有 n 个角色，分别为 $role_1,role_2,\cdots,role_n$，则可将可实现状态划分如下。

1) RI 只包含角色为 $role_1,role_2,\cdots,role_n$ 中的 1 种角色的角色实例，共 C_n^1 个划分。

2) RI 只包含角色为 $role_1,role_2,\cdots,role_n$ 中的 2 种角色的角色实例，共 C_n^2 个状态集合划分。

⋮

n) *RI* 只包含角色为 $role_1, role_2, \cdots, role_n$ 中的 *n* 种角色的角色实例，共 C_n^n 个状态集合划分。

以 NS 协议为例，该协议有 R_1 和 R_2 这 2 种角色，依据上述分类方法，可实现状态集合可分为 3 种。

1) *RI* 中只包含角色 R_1 的角色实例状态集合。

2) *RI* 中只包含角色 R_2 的角色实例状态集合。

3) *RI* 中包含角色 R_1 和角色 R_2 的角色实例状态集合。

若初始状态中 *RI* 仅包含角色 R_1 的一个角色实例，则经过状态扩展后，得到的可实现状态应存在于 1)和 3)之中；若初始状态中 *RI* 仅包含角色 R_2 的一个角色实例，则经过状态扩展后，得到的可实现状态应存在于 2)和 3)之中；若初始状态中 *RI* 包括角色 R_1 和 R_2 的各一个角色实例，则得到的可实现状态只存在于 3)中。

由此，可见初始状态的设定确实影响着可实现状态的结果，可依据待验证协议的安全属性设置初始状态，本文中安全属性主要考虑秘密性和认证性，定义如下。

定义 21 秘密性 $Secret(I,t)$。$Secret(I,t)$ 表示攻击者 *I* 不知道 *t*，即 *t* 满足秘密性。

定义 22 认证性 $Alive(p,p')$。$Alive(p,p')$ 表示主体 *p* 能够确信主体 p' 确实参与了同一协议同一轮次的运行。

定理 2 当协议的安全属性为秘密性 $Secret(I,t)$ 时，设秘密性 $Secret(I,t)$ 中待保密的参数 *t* 协议中的角色 R_1 生成，则初始状态中的 *RI* 可设定为包含 R_1 的一个角色实例。

证明 由于秘密性 $Secret(I,t)$ 的参数 *t* 由角色 R_1 生成，故可实现状态 *RI* 中必然包含角色 R_1 的一个角色实例，因此，初始状态设定为包含 R_1 的一个角色实例即可得到与秘密性相关的所有可实现状态。

定理 3 当协议的安全属性为单项认证性 $Alive(p,p')$ 时，设 *p* 为协议中角色 R_1 中的主动方，则初始状态中的 *RI* 可设定为包含 R_1 的一个角色实例。

证明 由于 $Alive(p,p')$ 表示主体 *p* 能够确信主体 p' 确实参与了同一协议同一轮次的运行，p' 为诚实主体。故无论 p' 是否参与了协议运行，主体 p' 必然参与了协议运行。即可实现状态 *RI* 中必然包含角色 R_1 的一个角色实例，因此，初始状态设定为包含 R_1 的一个角色实例即可得到与单向认证性 $Alive(p,p')$ 相关的所有可实现状态。

5.4 状态转移

定义 23 状态转移。

$$State \xrightarrow{e} State', e = E.pop()$$

在状态转移过程中，*RI* 会添加新的角色实例，*E* 会添加新的事件，*pair* 也会增加新的通信关系，状态转移过程中，应使序列 *E* 和 *pair* 满足规范时序，状态转移规则如下。

定义 24 状态转移规则 TRules。

TRule1：若 $e = send(s,t) \wedge \forall p \in tpair, e \neq p.send$，则

$$state' = \{RI, E/e, pair \cup (e \rightarrow recv(I,t)), tpair, tE, Atk \cup addTerm_{AtK}(t)\}$$

规则 TRule1 表示在无需回溯的情况下，攻击者接收消息，更新攻击者知识。

TRule2：若 $e = send(s,t) \wedge \forall p \in tpair, e = p.send$，则

$$State' = \{RI, E/e, pair \cup p, tpair/p, tE, Atk \cup addTerm_{AtK}(t)\}$$

规则 TRule2 表示处理的 *send* 事件时，*tpair* 集合中存在与之对应通信关系，由于该 *send* 事件的前驱事件已经处理完毕，应将该通信关系加入到 *pair* 中。

TRule3：若 $e = recv(s,t) \wedge s \neq I \wedge Atk \mapsto t$，则

$$State' = \{RI, E/e, pair \cup (send(I,t) \rightarrow e), tpair, sE, Atk\}$$

规则 TRule3 表示攻击者可以依据现有知识合成目标消息。

TRule4：若 $e = recv(s,t) \wedge s \neq I \wedge Atk \not\mapsto t$，设 *t* 的需求集合 $\{core(t)\}$，求解解空间 *S*，对于 $\forall e_1 \in S$，则

$$State' = \{RI, E \cup e_1, pair, tpair, sE, Atk\}$$

规则 TRule4 表示攻击者不能合成目标项，试图从外部获取需求以合成目标项。

TRule5：若 $e = recv(I,t) \wedge \exists e_1 \in sE$，且有 $Inst_1$ 和 $Inst_2$ 满足 $(e_1 \circ Inst_1 \rightarrow e \circ Inst_2) \wedge before(e_1) \wedge tE = \varnothing$，则形成状态为

$$State' = \{RI, E/e, pair \cup (e_1 \rightarrow e), tpair, sE/e_1, Atk \cup addTerm_{AtK}(t)\}$$

TRule6：若 $e = recv(I,t) \wedge \exists e_1 \in sE$，且有 $Inst_1$ 和 $Inst_2$ 满足 $e_1 \circ Inst_1 \rightarrow e \circ Inst_2 \wedge before(e_1) \wedge$

$tE \nsim \varnothing$，则形成的状态为

$$State' = \{RI, E \cup e_1 \cup before(e_1)/e, pair,$$
$$tpair \cup (e_1 \to e), sE/(e_1 \cup before(e_1)), Atk\}$$

规则 TRule5 和 TRule6 表示已有角色实例中存在事件可与目标事件形成通信关系。

TRule7：若 $e = recv(I,t) \wedge \exists e_1 \in Event(roleInst)$，且有 $Inst_1$ 和 $Inst_2$ 满足

$$e_1 \circ Inst_1 \to e \circ Inst_2 roleInst \notin RI$$
$$before(e_1) \bigcap tE \neq \varnothing$$

则形成状态为

$$State' = \{RI \cup roleInst, E \cup e_1 \cup before(e_1)/e,$$
$$pair, tpair \cup (e_1 \circ Inst_1 \to e \circ Inst_2),$$
$$sE \cup after(e_1), Atk\}$$

TRule8：若 $e = recv(I,t) \wedge \exists e_1 \in Event(roleInst)$ 且有 $Inst_1$ 和 $Inst_2$ 满足

$$e_1 \circ Inst_1 \to e \circ Inst_2 \wedge roleInst \notin RI$$
$$before(e_1) \bigcap tE = \varnothing$$

则形成状态为

$$State' = \{RI \cup roleInst, E/e, pair \cup$$
$$(e_1 \circ Inst_1 \to e \circ Inst_2), tpair, sE \cup after(e_1),$$
$$Atk \cup addTerm_{Atk}(t)\}$$

规则 TRule7 和 TRule8 表示已有角色实例中存在事件可与目标事件形成通信关系。

5.5　无效状态

在状态搜索过程中，若得到的中间状态不可能形成可实现状态，那么该状态为无效状态。SSMCI 机制中无效状态的产生主要在于其回溯机制。

定义 25　需求序列。

在状态转移过程中，当规则 TRule4 满足时，计算出需求 $core(m_i)$，记录需求形成需求序列 $core(m_1),\cdots, core(m_n)$，每一状态都有与之对应的需求序列。

定理 4　设 $s \in state$，若对于 $i, j \in [1,n]$，满足 $core(m_i) = core(m_j)$，这样的需求序列称为需求环，出现需求环的状态为无效状态。

证明　对于 $s \in state$，设 $core(m_i) = core(m_j)$，其分别对应事件 $recv(In, m_i)$ 和 $recv(In', m_j)$，先处理事件 $recv(In, m_i)$，若攻击者在之后的状态扩展过程中能够得到了需求集合，则处理事件 $recv(In', m_j)$ 时，由于 $core(m_i) = core(m_j)$，依据需求的定义，攻击者能够合成 m_j，即 $core(m_j) = \varnothing$，矛盾。故该状态是无效状态，证毕。

本文把上述无效状态称为需求环无效状态。该类状态出现的较为普遍，如不剔除，会出现不可终止的情况，这与 Athena 算法不可中止的情况类似。

假设某一状态下待处理事件为 $recv(In, Term)$，依据转移规则 TRule4，形成的下一状态 $State'$ 中 E 中存在形如 $recv(I,t) \cdot send(I, Term) \cdot recv(In, Term)$ 序列；继续对 $State'$ 进行扩展，当处理 $recv(I,t)$ 事件时，如满足规则 TRule6 或 TRule8，并假设 $recv(I,t)$ 其与事件 $send(In', t')$ 形成临时通信关系，若 $recv(In', t'')$ 是 $send(In', t')$ 的前驱事件，且 t'' 满足 $core(Term) = core(t'')$，则将会形成事件序列 $recv(I,t) \cdot send(I,t'') \cdot recv(In', t'')$，依据状态转移规则，上述过程将会一直循环下去，不可终止。

6　验证机制分析

本节主要讨论 SSMCI 机制对协议安全特性的验证是否合理，验证过程能否终止。

6.1　合理性

为了证明该方法的合理性，先给出如下 3 个定理。

定理 5　当初始状态只包含一个角色实例时，从该初始状态出发，至少存在一个可实现状态。

证明　考虑一轮包含该角色实例且无攻击者参与的正常协议会话，依据 Dolve-Yao 模型假设，此时攻击者的行为仅表现为转发消息，该行为对应 SSMCI 机制中 TRule4 形成的转发事件序列 $recv(I,t) \cdot send(I,t) \cdot recv(In,t)$，因为 t 总是 $\{core(t)\}$ 的成员，且是消息模板的元素实例化，因此，该序列式是存在的。因此，必然存在一个能反应该会话场景的可实现状态。

定理 6　当初始状态只包含一个角色实例时，得到的可实现状态是有限的。

证明　在协议的正常运行中，所有角色实例的事件严格遵守事件之间的通信关系和协议规定的顺序，这些事件和关系集合可构成一个可实现状态。在存在攻击者干预的情况下，对于某一协议来说，攻击者的攻击形式是有限的，因此对于包含攻击轨迹的可实现状态是有限的；另外，包含角色实例的协议运行迹也是有限的，因此，包含某一角色实例的可实现状态是有限的。

从定理 5 和定理 6 可知，从包含一个角色实例的初始状态出发，进行状态搜索，得到的可实现状态是有限且不为空。

定理 7 在 Dolve-Yao 假设下，当初始状态只包含一个角色实例时，从该初始状态出发，得到的可实现状态集合反映了该角色实例所有可能的协议会话场景。

证明 可实现场景是由初始状态依据状态转移规则一步一步地扩展的，欲证明该定理，只需证明状态转移规则的完备性。对于协议交互过程，存在以下 4 种情况。

1) 合法角色实例 R 发送消息，对应于规则 TRule1 和 TRule2 。

2) 合法角色实例 R 收到消息，即事件 $recv(R,t)$，攻击者必须合成消息 t 才能与该事件绑定，这只有 2 种情况：若在某一状态下，攻击者能够基于现有知识合成 t，这种情况对应于规则 TRule3；若攻击者需要从外部获取需求，才能合成消息 t，根据 4.2 节可知每一种需求对应一种合成消息 t 的可能，该情况对应规则 TRule4 。

3) 攻击者发送消息，对于规则 TRule4 。

4) 攻击者接收消息，对应规则 TRule5 ~ TRule8 。

因此，状态转移规则是完整的，即从该初始状态出发，得到的可实现状态集合能够反映了该角色实例所有可能的协议的运行轨迹。得到协议运行轨迹后，可进一步分析其安全属性是否被破坏，因此，SSMCI 机制是合理的。

6.2 终止性

SSMCI 机制中状态扩展过程进行状态回溯的过程中会不断增加新的角色实例，但其是可终止的。

定理 8 SSMCI 机制是可终止的。

证明 状态扩展过程是一个状态转移树，当其非叶子节点都有限且树的深度有限时，则状态转移树是可终止的。证明可分 2 步。

在状态扩展过程中，规则 TRule1~TRule3 只有一个分支；规则 TRule4，由于解空间也是有限的，因此，每个叶子节点的分支是有限的；规则 TRule5~TRule8，由于每个角色实例的事件是有限，因此能绑定的事件数目也是有限的，故其分支也是有限的。综上，每条规则的分支数目都是有限的。

下面证明其深度亦是有限的。注意到 SSMCI 机制的特点是有攻击者按需添加角色实例，添加角色实例的目的是为了获取需求集合，由于需求集合是有限的，因此，添加的角色实例数目是有限的，故状态中集合 E 是有限的，从而深度是有限的。

从另外一个角度来看，如果添加的角色实例数目是无限的，则说明其需求是无法满足的，考虑到需求集合的有限性，唯一的可能就是出现了需求环的情况，这属于无效状态。

综上，算法是可终止的。

7 实验结果与分析

7.1 NSPK 案例分析

本节以 NSPK 协议为例，通过状态转移得到可实现状态，并验证其安全属性是否满足，这里只验证 NS 协议中接收方的秘密性。验证目标：接收方 b 第一轮会话中产生的随机数 $nb\#1$ 的秘密性，即 $secret(b^{\#1}, nb\#1)$ 是否满足，步骤如下。

Step 1 设初始状态 $state_0$，验证目标为 $secret(b^{\#1}, nb\#1)$。根据定理 2，$state_0$ 角色实例集合只包括 $rInst(b^{\#1})$，假设其与 a 进行通信，攻击者知识为 AtK_0，初始状态为

$$\begin{aligned}&RI=\{rInst(b^{\#1})\},E=\{<rInst(b^{\#1}),1>,\\&<rInst(b^{\#1}),2>,<rInst(b^{\#1}),3>\},\\&pair=\varnothing,tpair=\varnothing,sE=\varnothing,Atk=AtK_0\}\end{aligned}$$

其中，

$$\begin{aligned}&rInst(b^{\#1})=NS(resp)\circ\\&[b,1,\{Nb/nb\#1,Init/a,resp/b\}]\end{aligned}$$

Step 2 取 $E.pop()=<rInst(b^{\#1}),1>$，即处理事件 $recv(b^{\#1},(W\|a)_{pubk(b)})$，$Atk \nvdash (W\|a)_{pubk(b)}$ 满足状态转移规则 TRule4，求解出需求集合

$$\{core((W\|a)_{pubk(b)})\}=\{\{W\},(W\|a)_{pubk(b)}\}$$

$$type(W)=fresh$$

NSPK 协议的消息模板为

$$\begin{aligned}&Template=\{(X\|agent_1)_{pubk(agent_2)},\\&(X\|Y)_{pubk(agent_1)},(Y)_{pubk(agent_2)}\}\end{aligned}$$

其中，$type(X,Y)=fresh$。如表 1 所示，对于 $t\in Template$，依据消息中密钥项中的主体名是否实例化为攻击者 I，可分为 2 类进行推导。

表 1　需求实例接收消息和对应的 $addTerm_{AtK}(t)$

消息项	实例化	$addTerm_{AtK}(t)$
$(X \| agent_1)_{pubk(agent_2)}$	$agent_2 / I$	X
$(X \| agent_1)_{pubk(agent_2)}$	$agent_2 \diagdown I$	$(X \| agent_1)_{pubk(agent_2)}$
$(X \| Y)_{pubk(agent_1)}$	$agent_1 / I$	$\{X, Y\}$
$(X \| Y)_{pubk(agent_1)}$	$agent_1 \diagdown I$	$(X \| Y)_{pubk(agent_1)}$
$(Y)_{pubk(agent_1)}$	$agent_1 / I$	Y
$(Y)_{pubk(agent_1)}$	$agent_1 \diagdown I$	$(Y)_{pubk(agent_1)}$

当需求为 $(W \| a)_{pubk(b)}$ 时，存在置换因子 $\sigma_0 = \{X / W, agent_1 / a, agent_2 / b\}$，使 $AddTerm_{AtK}((X \| agent_1)_{pubk(agent_2)}\sigma_0) = (W \| a)_{pubk(b)}$。

若需求为 W，则存在 $\sigma_1 = \{X / W, agent_2 / I\}$ 使 $AddTerm_{AtK}((X \| agent_1)_{pubk(agent_2)}\sigma_1) = W$。

置换因子 $\sigma_2 = \{Y / W, agent_1 / I\}$ 使 $AddTerm_{AtK} \cdot ((Y)_{pubk(agent_1)}\sigma_2) = W$。

置换因子 $\sigma_3 = \{Y / W, agent_1 / I\}$ 满足 $W \in AddTerm_{AtK}((X \| Y)_{pubk(agent_1)}\sigma_3)$。

置换因子满足

$$W \in AddTerm_{AtK}((X \| Y)_{pubk(agent_1)}\sigma_4)$$

因此，解空间包含的 $recv$ 事件为

$$S = \{recv(I, (W \| P)_{pubk(I)}),$$
$$recv(I, (W \| Y)_{pubk(I)}), recv(I, (X \| W)_{pubk(I)})$$
$$recv(I, (W)_{pubk(I)}), recv(I, (W \| a)_{pubk(b)})\}$$

即有 5 种方式可以使攻击者获得新知识合成消息 $(W \| a)_{pubk(b)}$，分别对应 5 种状态转移方式，由于篇幅有限，本节主要介绍破坏秘密性属性的状态转移方式，其对应解空间元素 $recv(I, (W \| P)_{pubk(I)})$，其中，$type(P) = agent$，按照状态转移规则 TRule4 形成 $state_1$ 为

$$RI = \{rInst(b^{\#1})\}, E = \{recv(I, (W \| P)_{pubk(I)}) \cdot$$
$$< rInst(b^{\#1}), 1 > \cdot < rInst(b^{\#1}), 2 > \cdot$$
$$< rInst(b^{\#1}), 3 >\}, pair = \varnothing, tpair = \varnothing,$$
$$sE = \varnothing, Atk = AtK_0\}$$

Step 3　$state_1.E.pop() = recv(I, (W \| P)_{pubk(I)})$，即下一处理事件为 $recv(I, (W \| P)_{pubk(I)})$。由于存在角色实例 $rInst(a^{\#1})$ 的事件为 $< rInst(a^{\#1}), 1 >= send(a^{\#1}, (na\#1 \| a)_{pubk(resp)})$，实例化因子 $inst_1 = [I, 0, \{W / na\#1, P / a\}]$，$inst_2 = [a, 1, resp / I]$

可形成通信关系为

$$send(a^{\#1}, (na\#1 \| a)_{pubk(resp)}) \circ inst_2 \rightarrow recv(I, (W \| P)_{pubk(I)}) \circ inst_1。$$

根据 TRule8，形成下一状态 $state_2$ 为

$$RI = \{rInst(b^{\#1}), rInst(a^{\#1})\}, E =$$
$$\{< rInst(b^{\#1}), 1 > \cdot < rInst(b^{\#1}), 2 > \cdot$$
$$< rInst(b^{\#1}), 3 >\}, pair = \{< rInst(a^{\#1}), 1 >$$
$$\rightarrow recv(I, (na\#1 \| a)_{pubk(I)})\}, tpair = \varnothing,$$
$$sE = \{< rInst(a^{\#1}), 2 >, < rInst(a^{\#1}), 3 >\},$$
$$AtK = AtK_0 \cup na\#1\}$$

此时，角色实例 $rInst(b^{\#1})$ 更新为

$$rInst(b^{\#1}) = NS(resp) \circ [b, 1, \{resp / b,$$
$$nb / nb\#1, W / na\#1, Init / a\}]$$

角色实例 $rInst(a^{\#1})$ 更新为

$$rInst(a^{\#1}) = NS(Init) \circ [b, 1, \{init / b^{\#1},$$
$$Na / na\#1, Resp / I\}]$$

Step 4　$state_2.E.pop() =< rInst(b^{\#1}), 1 >$，对消息 $(na\#1 \| a)_{pubk(b)}$ 进行知识分解，因为 $na\#1$、a 和 $pubk(b)$ 属于 AtK，即 $Atk \mapsto (na\#1 \| a)_{pubk(b)}$，依据规则 TRule3 形成状态 $state_3$ 为

$$RI = \{rInst(b^{\#1}), rInst(a^{\#1})\},$$
$$E = \{< rInst(b^{\#1}), 2 > \cdot < rInst(b^{\#1}), 3 >\},$$
$$pair = \{< rInst(a^{\#1}), 1 > \rightarrow recv(I,$$
$$(na\#1 \| a)_{pubk(I)}), send(I, (na\#1 \| a)_{pubk(b)})$$
$$\rightarrow < rInst(b^{\#1}), 1 >\}, tpair = \varnothing,$$
$$sE = \{< rInst(a^{\#1}), 2 >, < rInst(a^{\#1}), 3 >\},$$
$$AtK = AtK_0 \cup na\#1\}$$

Step 5　$state_3.E.pop() =< rInst(b^{\#1}), 2 >$，即处理事件 $send(a^{\#1}, (nb\#1 \| na\#1)_{pubk(a)})$，依据符合 TRule1，攻击者收到消息项 $(nb\#1 \| na\#1)_{pubk(a)}$，得到知识 $(nb\#1 \| na\#1)_{pubk(a)}$，形成状态 $state_4$ 为

$$RI = \{rInst(b^{\#1}), rInst(a^{\#1})\},$$
$$E = \{< rInst(b^{\#1}), 3 >\}, pair =$$
$$\{< rInst(a^{\#1}), 1 > \rightarrow recv(I, (na\#1 \| a)_{pubk(I)}),$$
$$send(I, (na\#1 \| a)_{pubk(b)}) \rightarrow < rInst(b^{\#1}), 1 >,$$
$$< rInst(b^{\#1}), 2 > \rightarrow recv(I,$$
$$(nb\#1 \| na\#1)_{pubk(a)}\}, tpair = \varnothing,$$
$$sE = \{< rInst(a^{\#1}), 2 >, < rInst(a^{\#1}), 3 >\},$$
$$AtK = AtK_0 \cup na\#1 \cup (nb\#1 \| na\#1)_{pubk(a)}\}$$

Step 6　$state_4.E.pop() =< rInst(b^{\#1}), 3 >$，即事件 $recv(b^{\#1}, (nb\#1)_{pubk(b)})$，又 $Atk \not\vdash (nb\#1)_{pubk(b)}$ 根据规

则 TRule4，求解需求集合

$$\{core((nb\#1)_{pubk(b)})\}=\{\{nb\#1\},\{(nb\#1)_{pubk(b)}\}\}$$

解空间为

$$S=\{(nb\#1\|Y)_{pubk(I)},(X\|nb\#1)_{pubk(I)},$$
$$(nb\#1)_{pubk(I)},(nb\#1\|P)_{pubk(I)},(X\|nb\#1)_{pubk(I)}\}$$

这里同样只介绍破坏秘密性的状态转移方式，对应解空间中的 $recv(I,(nb\#1)_{pubk(I)})$，根据 TRule4 形成状态 $state_5$ 为

$$RI=\{rInst(b^{\#1}),rInst(a^{\#1})\},$$
$$E=\{recv(I,(nb\#1)_{pubk(I)})\cdot<rInst(b^{\#1}),3>\},$$
$$pair=\{<rInst(a^{\#1}),1>\rightarrow recv(I,$$
$$(na\#1\|a)_{pubk(I)}),send(I,(na\#1\|a)_{pubk(b)})$$
$$\rightarrow<rInst(b^{\#1}),1>,<rInst(b^{\#1}),2>\rightarrow$$
$$recv(I,(nb\#1\|na\#1)_{pubk(a)}\},tpair=\varnothing,$$
$$sE=\{<rInst(a^{\#1}),2>,<rInst(a^{\#1}),3>\},$$
$$AtK=AtK_0\cup na\#1\cup(nb\#1\|na\#1)_{pubk(a)}\}$$

Step 7 下一处理事件 $state_5.E.pop()$，即事件 $recv(I,(nb\#1)_{pubk(I)})$，由于存在事件 $send(a^{\#1},(nb\#1)_{pubk(I)})\in sE$，满足

$$send(a^{\#1},(nb\#1)_{pubk(I)})\rightarrow recv(I,(nb\#1)_{pubk(I)})。$$
$$before(send(a^{\#1},(nb\#1)_{pubk(I)}))$$
$$=recv(a^{\#1},(V\mid na\#1)_{pubk(a)})$$

按照规则 TRule6，形成状态 $state_6$ 为

$$RI=\{rInst(b^{\#1}),rInst(a^{\#1})\},E=$$
$$\{<rInst(a^{\#1}),2>\cdot<rInst(a^{\#1}),3>\cdot$$
$$<rInst(b^{\#1}),3>\},pair=\{<rInst(a^{\#1}),1>$$
$$\rightarrow recv(I,(na\#1\|a)_{pubk(I)}),$$
$$send(I,(na\#1\|a)_{pubk(b)})\rightarrow<rInst(b^{\#1}),1>,$$
$$<rInst(b^{\#1}),2>\rightarrow recv(I,(nb\#1\|na\#1)_{pubk(a)}\},$$
$$tpair=\{<rInst(a^{\#1}),3>\rightarrow recv(I,(nb\#1)_{pubk(I)})\},$$
$$sE=\varnothing,AtK=AtK_0\cup na\#1\cup$$
$$(nb\#1\|na\#1)_{pubk(a)}\}$$

新增的角色实例其事件与 $recv(I,(nb\#1)_{pubk(I)})$ 都不存在通信关系，不满足规则 TRule7 与 TRule8，状态搜索在这条路径上终止。

Step 8 处理事件为 $recv(a^{\#1},(V\|na\#1)_{pubk(a)})$，由于 $\exists(nb\#1\|na\#1)_{pubk(a)}\in Atk,\partial=V/nb\#1$，使 $(nb\#1\|na\#1)_{pubk(a)}=(V\|na\#1)_{pubk(a)}\circ\partial$，故 $Atk\mapsto(nb\#1\|na\#1)_{pubk(a)}$，根据规则 TRule3，形成通信关系

$$send(I,(nb\#1\|na\#1)_{pubk(a)})$$
$$\rightarrow recv(a^{\#1},(V\|na\#1)_{pubk(a)})\circ[a,1,V/nb\#1]$$

此时，$rInst(a^{\#1})$ 更新为

$$rInst(a^{\#1})=NS(Init)\circ[a,1,\{Init/a^{\#1},$$
$$na/na\#1,resp/I,V/nb\#1\}]$$

形成状态 $state_7$ 为

$$RI=\{rInst(b^{\#1}),rInst(a^{\#1})\},E=$$
$$\{<rInst(a^{\#1}),3>\cdot<rInst(b^{\#1}),3>\},$$
$$pair=\{<rInst(a^{\#1}),1>\rightarrow$$
$$recv(I,(na\#1\|a)_{pubk(I)}),$$
$$send(I,(na\#1\|a)_{pubk(b)})\rightarrow<rInst(b^{\#1}),1>,$$
$$<rInst(b^{\#1}),2>\rightarrow$$
$$recv(I,(nb\#1\|na\#1)_{pubk(a)},$$
$$send(I,(nb\#1\|na\#1)_{pubk(a)})$$
$$\rightarrow<rInst(a^{\#1}),2>\},tpair=$$
$$\{<rInst(a^{\#1}),3>\rightarrow recv(I,(nb\#1)_{pubk(I)})\},$$
$$sE=\varnothing,AtK=AtK_0\cup na\#1$$
$$\cup(nb\#1\|na\#1)_{pubk(a)}\}$$

Step 9 $state_7.E.pop()=<rInst(a^{\#1}),3>$，即处理事件 $send(a^{\#1},(nb\#1)_{pubk(I)})$，由于 $tpair$ 包含通信关系 $<rInst(a^{\#1}),3>\rightarrow recv(I,nb\#1)_{pubk(I)})$，根据 TRule2，新增知识为 $nb\#1$，形成状态 $state_8$ 为

$$RI=\{rInst(b^{\#1}),rInst(a^{\#1})\},$$
$$E=\{<rInst(b^{\#1}),3>\},pair=$$
$$\{<rInst(a^{\#1}),1>\rightarrow recv(I,$$
$$(na\#1\|a)_{pubk(I)}),send(I,(na\#1\|a)_{pubk(b)})$$
$$\rightarrow<rInst(b^{\#1}),1>,<rInst(b^{\#1}),2>$$
$$\rightarrow recv(I,(nb\#1\|na\#1)_{pubk(a)},$$
$$send(I,(nb\#1\|na\#1)_{pubk(a)})\rightarrow$$
$$<rInst(a^{\#1}),2>,<rInst(a^{\#1}),3>$$
$$\rightarrow recv(I,(nb\#1)_{pubk(I)})\},tpair=\varnothing,sE=\varnothing,$$
$$AtK=AtK_0\cup na\#1$$
$$\cup(nb\#1\|na\#1)_{pubk(a)}\cup nb\#1\}$$

Step 10 下一处理事件 $recv(b^{\#1},(nb\#1)_{pubk(b)})$，对 $(nb\#1)_{pubk(b)}$ 分解，由于 $nb\#1$ 和 $pubk(b)$ 均在攻击者知识中，故攻击者可合成 $(nb\#1)_{pubk(b)}$，根据规则 TRule1 形成状态 $state_9$ 为

$$RI=\{rInst(b^{\#1}),rInst(a^{\#1})\},E=\varnothing,$$
$$pair=\{<rInst(a^{\#1}),1>\rightarrow recv(I,$$
$$(na\#1\|a)_{pubk(I)}),send(I,(na\#1\|a)_{pubk(b)})$$
$$\rightarrow<rInst(b^{\#1}),1>,<rInst(b^{\#1}),2>$$
$$\rightarrow recv(I,(nb\#1\|na\#1)_{pubk(a)},$$
$$send(I,(nb\#1\|na\#1)_{pubk(a)})\rightarrow$$
$$<rInst(a^{\#1}),2>,<rInst(a^{\#1}),3>$$
$$\rightarrow recv(I,(nb\#1)_{pubk(I)}),send(I,(nb\#1)_{pubk(b)})$$
$$\rightarrow<rInst(b^{\#1}),3>\},tpair=\varnothing,sE=\varnothing,AtK=$$
$$AtK_0\cup na\#1\cup(nb\#1\|na\#1)_{pubk(a)}\cup nb\#1\}$$

由于 $E=\varnothing$ 为空，故 $state_9\in Ultstate$，由于 $nb\#1\in state_9.Atk$，故其秘密性受到了破坏。其攻击路径如下。

1) 主体 a 产生随机数 $na\#1$，发送消息 $(na\#1\|a)_{pubk(I)}$ 给主体 I，I 用私钥 $prik(I)$ 解密消息 $(na\#1\|a)_{pubk(I)}$ 得到随机数 $na\#1$ 和主体名 a；I 冒充主体 a 与主体 b 会话，合成消息 $(na\#1\|a)_{pubk(b)}$ 发送给主体 b。

2) b 解密消息后得到 $na\#1$ 和 a，产生随机数 $nb\#1$，并向主体 a 发送消息 $(nb\#1\|na\#1)_{pubk(a)}$。

3) a 收到消息 $(nb\#1\|na\#1)_{pubk(a)}$，解密得到 $na\#1$ 和 $nb\#1$，核实 $na\#1$ 的一致性后，发送 $(nb\#1)_{pubk(I)}$ 给 I。

4) I 解密消息后 $(nb\#1)_{pubk(I)}$ 得到 $nb\#1$，并向主体 b 发送消息 $(nb\#1)_{pubk(b)}$。这样，攻击者成功冒充了主体 a 与主体 b 进行了一轮会话，并得到 $nb\#1$。

7.2 协议测试结果

本文基于 SSMCI 开发了原型系统并进行了测试，测试结果如表 2 所示。其中，NSPK 和 NSPK-FIX 协议来自 SPORE 协议库，Andrew、EKE、CHAPV2 和 Cham-MD5 协议选自 AVISPA 协议库，Helsiniki 协议选自国际标准 ISO/IECDIS11770-3 中提出的安全协议草案。结果表明，SSMCI 得到的结论与 Scyther、AVISPA 等工具一致。

SSMCI 的中间状态数目和可实现状态数目与初始状态有关。在表 2 中，测试 EKE 和 Andrew 协议时初始状态设定为仅包含一个发起者的角色实例，测试其他协议时初始状态为仅包含一个响应者的角色实例，Scyther 工具测试协议时，可从不同的角色出发，得到的可实现状态数目也不一样，表 2 数据中 EKE 和 Andrew 协议选择的是发起者的可实现状态数目，而其他协议选择的是响应者。由于 Scyther 工具不能得到验证过程中的中间状态的数目，仅能比较两者的终止状态数目。结果表明 SSMCI 略优于 Scyther，主要原因如下。

1) 避免了时序矛盾

在 Scyther 的状态扩展过程中，未考虑协议顺序和验证次序的矛盾关系，会出现一些因果矛盾的状态和状态分支，Scyther 方法中有关于这类状态的检测方法。SSMCI 引入回溯机制，在状态搜索过程中不会出现这类无效状态的扩展分支。对于模型检测方法来说，搜索过程可利用的信息越多，中间状态数目就越少，如 Athena 和 Scyther 在状态定义和所示方式上都考虑节点之间的因果关系，减少状态空间数目。SSMCI 继承这一思想并将时序关系考虑在内，可进一步减少了状态空间数目。

2) 减少了扩展分支

Scyther 方法在处理接收事件节点时，几乎每一状态都有 3 个大分支:攻击者合成绑定 Co、已有角色实例绑定 DeEx 和新增角色实例绑定 DeNew；SSMCI 在处理节点是根据攻击者需求来定的，当攻击者可以合成该消息时，无需增加新的会话实例，即满足 Co，则无需搜索分支 DeNew 和 DeEx。

表 2　协议测试结果

协议名称	结果	中间状态数	可实现状态数	Scyther 可实现状态数	验证时间/s
NSPK	攻击	175	2	4	0.87
NSPK-FIX	安全	101	1	1	0.55
Andrew	攻击	165	2	2	0.83
EKE	攻击	183	2	2	0.93
CHAPv2	安全	81	2	2	0.43
Helsiniki	攻击	168	2	4	0.83
Cham-MD5	安全	155	2	2	0.81

3) 依据安全属性设定初始状态

本文注意到 Scyther 的初始状态有多个，没有考虑安全属性、初始状态和可实现状态的关系。在对 Scyther 测试过程中，Scyther 方法中一些初始状态搜索得到的可实现状态集合存在重复的情况，形成一定的冗余。本文定理 2 和定理 3 初始状态的设定影响着可实现状态集合的结果，可根据待验证协议的安全属性设置初始状态，避免这种冗余。

8 结束语

由于攻击者行为的复杂性、协议会话数目过多和对称冗余，现有基于模型检测的安全协议验证方法都面临状态空间爆炸的问题。本文建立了一个以攻击者为中心的安全协议验证机制，它由攻击者依据需求添加会话实例，每一个会话实例都能为攻击者所利用，减少了许多无效协议会话。

一些工具如 Athena 和 Scyther 也采用类似的按需添加会话实例的状态搜索方法，和它们相比，本文进一步提炼了需求的概念，并在 SSMCI 搜索机制引入了回溯机制，能确保状态搜索中攻击者推理时无时序矛盾，减少了存在时序矛盾的状态分支。下一步工作拟刻画一个更加全面的攻击者模型，并在求解需求时体现攻击者的代数计算能力。

参考文献：

[1] GUTTMAN J D. Establishing and preserving protocol security goals[J]. Journal of Computer Security, 2014, 22(2):203-267.

[2] ARMANDO A, CARBONE R, COMPAGNA L. SATMC: a SAT-based model checker for security protocols, business processes, and security APIs[J]. International Journal on Software Tools for Technology Transfer, 2016, 18(2):187-204.

[3] CREMERS C, MANW S. Operational semantics and verification of security protocols[M]. Springer-Verlag Berlin Heidelberg, 2012: 78-103.

[4] CREMERS C, LAFOURCADE P, NADEAU P. Comparing state spaces in automatic security protocol analysis[M]. Formal to Practical Security. Springer Berlin Heidelberg, 2009: 70-94.

[5] DAIVD B, SEBASTIAN M, LUCA V. OFMC: a symbolic model checker for security protocols[J]. International Journal of Information Security, 2005, 4(3):181-208.

[6] ARMANDO A, CARBONE R, COMPAGNA L. SATMC: a SAT-based model checker for security protocols, business processes, and security APIs[J]. International Journal on Software Tools for Technology Transfer, 2016, 18(2):187-204.

[7] LUCA V. Automated security protocol analysis with the AVISPA tool[J]. Electronic Notes in Theoretical Computer Science, 2006, 155: 61-86.

[8] CHEVAL V, CORTIER V. Timing attacks in security protocols: symbolic framework and proof techniques[M]//Principles of Security and Trust, Springer Berlin Heidelberg, 2015:280-299.

[9] STYLIANOS B, PANAGIOTIS K, ANDREW P. An intruder model with message inspection for model checking security protocols[J]. Computers & Security, 2010, 29(1):16-34.

[10] BLANCHET B. Automatic verification of security protocols in the symbolic model: the verifier proverif[M]. Foundations of Security Analysis and Design VII. Springer International Publishing, 2014, 8604: 54-87.

[11] PINAZO S S. Advanced features in protocol verification: theory, properties, and efficiency in maude-NPA[J]. Escobar Román Santiago, 2015.

[12] CHOTHIA T, SMYTH B, STAITE C. Automatically checking commitment protocols in ProVerif without false attacks[C]//International Conference on Principles of Security and Trust. Springer Berlin Heidelberg. 2015:137-155.

[13] SIMON M. Advanced automatic security protocol verification[D]. ETH ZURICH, 2013:99-116.

[14] MEIER S, SCHMIDT B, CREMERS C. The TAMRIN prover for the symbolic analysis of security protocols[C]//The 25th International Conference on Computer Aided Verification (CAV'13). 2013:696-701.

[15] BASIN D, CREMERS C. Modeling and analyzing security in the presence of compromising adversaries[C]//The 15th European Conference on Research in Computer Security, Dimitris Gritzalis, 2010, 6345:340-356.

作者简介：

谷文（1992-），男，湖南耒阳人，解放军信息工程大学硕士生，主要研究方向为安全协议形式化分析与自动化验证。

韩继红（1966-），女，山西定襄人，博士，解放军信息工程大学教授、博士生导师，主要研究方向为网络与信息安全、安全协议形式化分析与自动化验证。

袁霖（1981-），男，河南商丘人，博士，解放军信息工程大学副教授，主要研究方向为安全协议形式化分析与自动化验证、软件可信性分析。