软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(11):3400−3411 [doi: 10.13328/j.cnki.jos.005622]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 Event-B 方法的安全协议设计、建模与验证*

李梦君[1,2], 潘国腾[1], 欧国东[1]

[1](国防科技大学 计算机学院,湖南 长沙 410073)

[2](信息安全国家重点实验室(中国科学院 信息工程研究所),北京 100093)

通讯作者: 李梦君, E-mail: mengjun.li@163.com

摘 要: 随着软件精化验证方法以及 Isabella/HOL、VCC 等验证工具不断取得进展,研究者们开始采用精化方法和验证工具设计、建模安全协议和验证安全协议源程序的正确性.在介绍 Event-B 方法和验证工具 Isabella/HOL、VCC 的基础上,综述了基于 Event-B 方法的安全协议形式化设计、建模与源程序验证的典型研究工作,主要包括从需求规范到消息传递形式协议的安全协议精化设计、基于 TPM(trusted platform module)的安全协议应用的精化建模以及从消息传递形式协议到代码的源程序精化验证.

关键词: 安全协议设计;安全协议建模与验证;精化; Event-B 方法

中图法分类号: TP311

中文引用格式: 李梦君,潘国腾,欧国东.基于 Event-B 方法的安全协议设计、建模与验证.软件学报,2018,29(11):3400−3411. http://www.jos.org.cn/1000-9825/5622.htm

英文引用格式: Li MJ, Pan GT, Ou GD. Design, modeling and verification of security protocols based on event-B method. Ruan Jian Xue Bao/Journal of Software, 2018,29(11):3400−3411 (in Chinese). http://www.jos.org.cn/1000-9825/5622.htm

Design, Modeling and Verification of Security Protocols Based on Event-B Method

LI Meng-Jun[1,2], PAN Guo-Teng[1], OU Guo-Dong[1]

[1](College of Computer, National University of Defense Technology, Changsha 410073, China)

[2](State Key Laboratory of Information Security (Institute of Information Engineering, The Chinese Academy of Sciences), Beijing 100093, China)

Abstract: With the progress in software refinement verification methods and theorem provers such as Isabella/HOL and VCC, researchers begin to study the design and modeling of security protocols and verify the correctness of their source codes based on the refinement technique and theorem provers. In this paper, the event-B method and verification tools Isabelle/HOL and VCC are introduced, and the typical work on the design and modeling of security protocols and verification of the correctness of their source codes is surveyed. These work include: the refinement design method of security protocols, the refinement modeling method of TPM-based protocol applications, and the refinement verification method of source code.

Key words: security protocol design; security protocol modeling and verification; refinement; event-B method

安全协议是建立在密码体制基础上的一种通信协议,它运行在计算机网络或分布式系统中,借助于密码算法为在网络环境中实现密钥分配、身份认证、电子商务交易等任务的各方约定任务执行步骤和执行规则.安全协议的正确性对网络应用的安全至关重要,安全协议多个会话的并发交叠运行和攻击者的破坏,使得安全协议运行时往往难以实现其设计目标.形式化方法已被证明是分析和验证安全协议的有效手段,安全协议形式化方

* 基金项目: 国家自然科学基金(61672525); 中国科学院信息工程研究所信息安全国家重点实验室开放课题(2016-MS-21)

Foundation item: National Natural Science Foundation of China (61672525); Open Projects of the State Key Laboratory of Information Security (Institute of Information Engineering, The Chinese Academy of Sciences) (2016-MS-21)

收稿时间: 2018-01-08; 修改时间: 2018-04-19, 2018-05-29; 采用时间: 2018-06-29

法研究一直是网络安全领域的研究热点之一.

安全协议形式化方法研究主要包括协议设计、建模、安全性质验证以及形式化工具开发[1].其中,协议建模的研究内容包括:攻击者建模、安全性质刻画和协议内容建模.一般使用逻辑系统描述攻击者的 Dolev-Yao 模型,基于 CCS、π演算等进程代数语言、Strand 空间、状态迁移系统、逻辑系统等多种形式系统建模安全协议内容,基于时序逻辑、互模拟等价、谓词描述保密性、认证性、完整性、非否认性等安全性质.

在安全协议形式化验证方法中,一般使用模型检验方法遍历安全协议状态迁移系统模型所有状态空间、计算安全协议逻辑系统模型的最小不动点、或者基于结构归纳法和定理证明器等方法验证安全协议逻辑系统模型满足安全性质.已开发的安全协议形式化工具有 Proverif、Scythe、Athena、Avispa、Casper/FDR 等.应用形式化自动工具,研究者对多个投入实用的安全协议进行了建模和分析,实际发现了其中存在的安全漏洞,表明了安全协议形式化方法研究具有实际应用价值.

上述研究首先建立形式化模型,然后验证形式化模型是否满足安全性质.因为形式化模型只是安全协议实现程序的抽象,很多实现细节在形式化模型中被忽略,比如实现程序中加/解密算子、其他构造/析构算子具有的代数性质,实现程序中可能存在缓冲区溢出、空指针引用、数组越界等运行时错误.

安全协议攻击者可以利用实现程序中这些被忽略的细节及存在的漏洞进行攻击,导致安全协议实现程序不再满足安全性质.一个典型的例子是传输层协议 TLS,TLS 协议是互联网上部署最多的安全协议,用于实现保密性和认证性.1999 年,Paulson 基于踪迹为 TLS 协议的简化版本建立了抽象模型,抽象模型中忽略了记录格式、域宽度、密码算法、预警和出错消息、服务器秘钥交换消息等细节,然后基于结构归纳法和定理证明器 Isabelle/HOL 成功验证了抽象模型满足保密性和认证性[2].

但 2014 年 1 月~2015 年 1 月的 13 个月中,OpenSSL、GnuTLS 等 6 个被广泛应用的 TLS 协议实现系统共发布了 54 个安全报告,其中,OpenSSL 有 22 个,GnuTLS 有 6 个[3].引起 TLS 实现系统脆弱性的主要原因有[3]:

(1) 内存安全性违背:总计有 15 个,包括越界读/写操作、空指针引用等内存错误,这些漏洞可以导致 TLS 协议客户端进程崩溃、拒绝服务攻击以及敏感信息泄漏.

(2) 证书解析:为了解码 X.509 证书,TLS 实现系统需要解析 ASN.1.因为 ASN.1 是一个大型且复杂的标准,一些 TLS 实现系统选择已有的 ASN.1 解析器,这些解析器本身存在的脆弱性(例如,解析器解析时使用了没有初始化的内存、不完整的选择性解析)就成为 TLS 协议实现系统脆弱性的来源之一.

(3) 证书验证:X.509 证书具有嵌套数据结构,它有 3 个标准化版本,并提供可选扩展,这些特点导致 X.509 证书验证时在控制流逻辑和证书解释上容易出错.一个典型的例子就是 GnuTLS 中的控制流逻辑错误,当中间证书是 X.509 的版本 1 时,一个写错位置的 goto 语句将使证书验证被跳过.

(4) 状态机错误:这种类型的脆弱性错误一共有 10 个,一个典型例子是苹果公司的 goto fail,在负责验证 ServerKeyExchange 消息中的数字签名过程中,一个重复的 goto 语句使过程跳过剩下的语句并返回输出变量的值,这个输出变量的初始值被设置成 success,因而实际上没有验证数字签名.

(5) 协议错误:2014 年,TLS 协议协议本身的两个错误被指出:POODLE 是 SSL3 中的一个错误,没有指定 CBC 模式下填充字节的值;三方握手是一种中间人攻击,一个参与者能够使用相同的安全性参数和摘要进行会话协商.

(6) 时间侧信道泄漏:两个脆弱性与时间侧信道泄漏相关,密码操作的时间与密码秘钥相关.

(7) TLS 库的使用:在已经统计的错误中,有 10 个错误是客户端软件在 TLS 库提供的 API 的使用方式上出错.

Sewell 等人认为[3],TLS 协议实现系统出现脆弱性的直接原因是内存管理等多个方面的编程错误,但是根本原因在于 TLS 协议的 RFC 规范具有歧义,难以正确理解;编程使用的大型 API 接口和代码库本身具有复杂性;一些编程选择具有不安全性;需求规范和代码之间具有语义断层(semantic gap),无法对规范和代码直接进行一致性测试.

Paulson 对 TLS 协议简化版本的成功验证和 TLS 协议实现系统发现的脆弱性表明:基于形式化模型的安全

协议分析与验证能够保证形式化模型满足安全性质,但是并不保证安全协议实现程序一定满足安全性质,因而需要直接验证安全协议实现程序的正确性.另一方面,基于状态迁移系统和逻辑系统建模的验证方法中,验证安全性质时,模型检验方法要求状态迁移系统模型状态空间数目必须有穷,逻辑系统模型要求最小不动点计算必须终止,因而这些方法不适合验证大型复杂安全协议,也不适合验证安全协议实现程序.

随着符号执行、契约、精化等软件验证方法和 Isabelle/HOL,VCC 等验证工具取得进展,研究者们开始采用软件验证方法和验证工具验证安全协议实现程序的正确性[4].在基于符号执行的安全协议验证中,文献[5]基于符号执行、安全协议验证工具 Proverif 提出了安全协议 C 语言程序的验证方法,对 simple mac、RPC、NSL、Csur、minexplib 等 250 行~1 000 行的协议源程序进行了验证,发现了 CSur 不满足认证性、minexplib 不正确的指针引用攻击.文献[6]进一步将符号执行、安全协议验证工具 Cryptoverif、SMT 工具 Yices 应用于验证安全协议 C 语言程序,能够对 3 000 行左右安全协议源程序的计算语义安全性进行分析与验证.

在基于契约的安全协议源程序模块化验证中,文献[7]基于 SMT 带精化类型的类型检查器 F7,实现了 F#语言安全协议程序基于契约的模块验证方法.文献[8]基于类型检查器 F7,进一步提出了 ML 语言安全协议源程序计算语义基于契约的模块验证方法.在 F7 及 Fine 等相关工作基础上,文献[9]为安全分布式编程设计和实现的一个可靠的类型化语言 F*,F*被用于编程和验证 20 000 行左右的 F*程序,包括多方会话、Web 浏览器、零知识证明协议、TLS 标准以及 F*类型检查器自身.在 F*基础上,文献[10]进一步提出了一个高阶状态编程语言 RF*,并基于精化类型建立了验证系统,RF*方法被用于为一系列的密码构造和协议编程,并用于验证密码构造和协议实现的完整性、隐私性等安全性质.

安全协议需求规范和代码之间具有语义断层,与软件需求规范和软件实现代码之间的关系类比可知,安全协议需求规范用问题域概念描述,易于被理解和推理,但是不能被计算机执行,安全协议实现代码基于程序设计语言编写,能够在计算机上运行,但是难以被理解和推理,表明安全协议需求规范的语义与安全协议实现代码的语义是不同的,它们之间存在语义断层.安全协议需求规范经过分解、精化、一般实例化[10]并用程序设计语言编程构造出程序代码,经过上述复杂的语义变换后,从安全协议实现代码中难以直接重构出语义变换过程中所有子规范以及最初的需求规范,因而计算机很难直接验证安全协议实现程序是否满足安全协议需求规范.规范与实现代码之间的语义断层是导致安全协议实现代码验证困难的根本原因[11−13].

洞察到软件需求规范与软件实现代码之间的语义断层问题,Back、Morgan、Woodcodk 等人提出了指导软件形式化开发的精化演算[14−16].将规范也作为语句的程序称为抽象程序,精化是抽象程序之间的二元关系,精化保持抽象程序的部分正确性(partial correctness).精化演算提供了一组规则指导抽象程序的逐步精化,形式化规范通过迭代精化变换为可执行程序.保证精化规则可靠性的条件称为证明义务,为了保证精化规则应用的可靠性,精化过程使用验证工具检查证明义务的有效性.应用精化规则并证明证明义务有效性以及精化关系具有的传递性,将保证迭代精化构造的可执行程序一定满足形式化规范.

安全协议需求规范和实现代码之间具有语义断层,存在的语义断层使形式化验证方法不能直接验证安全协议需求规范和实现代码之间具有一致性.精化是消除需求规范与实现代码之间语义断层的有效方法,本文将聚焦于基于精化思想的 Event-B 方法,介绍 Event-B 方法、综述基于 Event-B 的安全协议形式化设计、安全协议建模与源程序验证的主要工作,并进一步展望未来的研究工作.

1 Event-B 方法

在 Z 方法、B 方法和 Action 系统的基础上,Jean-Raymond Abrial 进一步提出了 Event-B 方法[17,18].Event-B 方法支持并发系统、分布式系统等复杂离散系统的描述、形式化设计、推理直至生成实现代码.

Event-B 方法以集合论、谓词逻辑和广义代换语言作为建模语言,上下文(context)和抽象机(machine)是描述离散系统模型的基本构件.上下文包含了集合常数(carrier set)、数值常数、定理、公理等基本构件,用于描述模型的静态性质.抽象机包含了变量、不变式、定理、事件等基本构件,其中,变量、不变式刻画系统的状态,事件由卫式条件和动作组成,刻画系统的状态迁移.抽象机主要描述模型的动态行为.

上下文之间具有扩展(extend)关系,扩展上下文可以引用被扩展上下文中定义的集合常数、数值常数.抽象机与上下文之间具有看见(sees)关系,抽象机可以引用它看见的上下文中定义的集合常数、数值常数.抽象机之间具有精化关系,一个抽象机最多只能精化一个抽象机,通过引入新的变量、新的不变式和新的事件精化抽象机中的状态或状态迁移,精化使抽象机之间形成层次化关系.

在 Event-B 方法中,抽象机之间的精化关系由具体抽象机中的具体事件和被精化的抽象机中的抽象事件之间的精化关系刻画:具体事件的卫式条件增强抽象事件的卫式条件,具体事件的动作模拟抽象事件的动作.精化抽象机与被精化抽象机之间通过粘结不变式(gluing invariants)关联,所有粘结不变式可以表明模型如何满足需求规范.抽象机中所有事件都必须保持不变式.

Rodin 是 Event-B 方法的开放工具集[19],Rodin 的集成开发环境提供了一组工具,包括抽象机、上下文的模型编辑、抽象机、上下文中变量、不变式、定理、事件等构件的类型检查、模型检查器 Prob、证明义务自动生成、证明义务自动证明和交互证明工具等.

基于 Event-B 的复杂离散系统建模,首先构造出满足需求规范的抽象机和上下文,然后迭代地应用分解、精化和一般实例化的建模方法[10],构造需求规范不同精化层次模型,并且从需求规范到生成实现代码的每一步精化都有证明义务有效性的证明,精化关系具有传递性,将保证最终构造的模型或实现代码一定满足需求规范.构造复杂离散系统不同精化层次的多层模型,使相邻模型之间语义比较接近,从而保证模型正确的证明义务更易于成功验证.

构造系统不同精化层次模型,在抽象模型中还可以引入理想性假设,使需求规范或者抽象机中的不变式容易验证,然后,在后继精化步骤中通过引入新的变量、不变式和事件精化抽象模型,消除理想性假设,使构造的精化模型易于编程实现.如,为分布式系统建模时,通常在抽象模型中假设分布的多个进程之间能够相互直接读/写局部存储,然后在后继精化步骤中,再用接收/发送消息精化分布进程之间直接读/写局部存储[17].形式化设计或者建模安全协议时,通常在抽象模型中假设诚实参与者在具有保密性、认证性的安全信道上接收/发送消息,在后继精化步骤中,使用在不安全信道上的加密消息精化在安全信道上接收/发送的消息[20−22].

2 验证工具 Isabelle/HOL 和 VCC

安全协议基于 Event-B 方法的形式化设计、建模与验证研究中,验证工具用于自动或者交互式验证证明义务的有效性,Isabelle/HOL 和 VCC 是常用的两款验证工具.

Isabelle[23]是通用的定理证明辅助系统,定义了形式语言并提供了验证数学公式的验证工具,Isabelle 的主要应用是数学证明的形式化,包括计算机硬件和软件系统的正确性验证以及计算机语言和协议的性质验证.Isabelle/HOL 提供了高阶逻辑的定理证明环境,它是 Isabelle 应用最广泛的一个版本.Isabelle/HOL 定义了强大的描述语言,基于描述语言,可以方便地自定义集合、关系、函数、逻辑公式,支持集合归纳定义、带复杂模式匹配的递归函数以及集合、线性列表和结构体等数据类型.Isabelle/HOL 集成了一些验证工具以提高验证效率,例如,Isabelle 的经典推理器能够处理推理步骤的长链完成定理证明,工具 Simplifier 能够对等式进行推理,集成了多个代数判定过程使得线性算术公式在 Isabelle/HOL 中能够被自动证明,通过 sledgehammer 可以调用外部的一阶逻辑证明器.Isabelle/HOL 还定义了制导定理证明的结构化证明语言 Isar,Isar 使证明文本易于被用户和计算机理解.Isabelle/HOL 还支持将可执行的规范直接转换为 SML,Ocaml,Haskell 和 Scala 这 4 种编程语言的实现代码.

VCC[24]是针对 C 语言编程的低层次并发系统实现代码的一个具有工业强度的验证环境.对标注了函数契约、状态断言和类型不变式等规范的程序,VCC 验证这些标注规范的正确性,其中,程序规范包括内联断言、函数前置条件和后置条件以及与复杂 C 类型(结构体和联合)相关联的对象不变式.通过 Boogie 中间语言,VCC 根据程序、程序规范和 Dijkstra 谓词转换器生成验证条件,然后用 Z3 等 SMT 工具证明生成的验证条件的有效性.

3 基于 Event-B 方法的安全协议设计

基于 Event-B 和定理证明器 Isabelle/HOL,Sprenger 等人提出了安全协议形式化设计方法[20−22],从安全协议需求规范和环境假设出发,构造出消息传递形式的安全协议,并且保证构造的安全协议一定满足需求规范.安全协议需求规范包括服务器产生并分发新会话秘钥、秘钥具有保密性、服务器与协议发起者、协议接收者相互之间的认证等.

安全协议形式化设计方法中,模型是形如 $S=(T,O,obs)$的规范,其中,$T=(\Sigma,\Sigma_0,\rightarrow)$为状态迁移系统,$O$ 是观察对象集合,$obs:\Sigma\rightarrow O$ 是状态集合到观察对象集合的观察函数.状态迁移系统 T 中,Σ是状态集合,Σ_0 是起始状态集合,迁移关系$\rightarrow$是事件的有穷集合.事件是形如 $evt(x)=\{(s,s')|G(x,s)\wedge s'=s(|v:=f(x,s)|)\}$的参数化关系,其中,$G(x,s)$是卫式条件,卫式条件满足时事件才会发生,$s'=s(|v:=f(x,s)|)$表示状态 s'通过修改状态 s 中变量构成的向量 v 的值为 $f(x,s)$得到.状态迁移系统用 Isabelle/HOL 中的结构体 Record 建模,结构体中包含起始状态集合和迁移关系两个成员.

安全协议模型之间的精化关系用状态迁移系统之间标准的模拟关系刻画.对于具体模型 $S_c=(T_c,O_c,obs_c)$和抽象模型 $S_a=(T_a,O_a,obs_a)$,中间函数$\pi:O_c\rightarrow O_a$,如果存在模拟关系 $R\subseteq\Sigma_a\times\Sigma_c$ 满足以下 3 个条件.

(1) 具体模型和抽象模型的初始状态集合之间具有模拟关系 $R:\Sigma_{c,0}\subseteq R(\Sigma_{a,0})$.

(2) 具体模型的状态迁移和抽象模型的状态迁移之间保持模拟关系 R.

- 卫式条件加强:$G_c(w(x),t)\Rightarrow G_a(x,s)$.
- 抽象模型的状态迁移和具体模型的状态迁移保持模拟关系 $R:(s,t)\in R\Rightarrow(s(|u:=f_a(x,s)|),t(|v:=f_c(w(x),t)|))\in R$.

其中,$G_a(x,s)$和 $G_c(w(x),t)$分别是抽象事件 evt_a 和具体事件 evt_c 的卫式条件,卫式条件加强中,$G_c(w(x),t)$逻辑蕴涵 $G_a(x,s)$保证抽象事件如果发生,则具体事件一定发生.$w(x)$是见证(witnesses),它建立具体事件的参数 $w(x)$和抽象事件的参数 x 之间的关系,$(s,t)\in R$ 逻辑蕴涵$(s(|u:=f_a(x,s)|),t(|v:=f_c(w(x),t)|))\in R$ 保证:如果抽象状态 s 和具体状态 t 具有模拟关系 R,分别执行抽象模型中的抽象事件 $u:=f_a(x,s)$和具体模型中的具体事件 $v:=f_c(w(x),t)$后,到达的抽象状态 $s(|u:=f_a(x,s)|)$和具体状态 $t(|v:=f_c(w(x),t)|)$仍然具有模拟关系 R.

(3) 对于所有$(s,t)\in R,obs_a(s)=\pi(obs_c(t))$,中间函数将具体状态的观察对象集合映射为抽象状态的观察对象集合.

则称具体模型 S_c 基于中间函数π精化抽象模型 S_a.

通过中间函数π在两个状态空间不相同的模型之间建立了精化关系.如果具体模型 S_2 使用中间函数π精化抽象模型 S_1,则 S_1 的所有抽象不变式将是中间函数π作用在 S_2 的可达状态集合得到的状态集合的不变式.上述性质表明精化保持抽象不变式,具体模型中所有事件保持抽象模型中的不变式将不需要验证.如果具体模型 S_3 基于中间函数ρ精化抽象模型 S_2,模型 S_2 基于中间函数π精化抽象模型 S_1,则具体模型 S_3 基于中间函数$\pi\circ\rho$精化抽象模型 S_1,上述性质表明精化具有传递性.

基于逐步精化的安全协议形式化设计由 4 个精化层次模型组成.

(1) 第 0 层是抽象、与具体协议无关的安全性和认证性规范模型.

所有状态都是可观察的,为保密性建立保密性模型,为认证性建立非单射一致性模型或者单射一致性模型.

在保密性模型中,定义了两个状态变量 kn 和 az,分别表示参与者与数据之间的知道关系和授权关系,定义了两个事件 gen 和 $learn$,分别表示秘密产生事件、秘密获取事件,保密性用不变式 $secrecy=\{s|s.kn\subseteq s.az\}$描述,即:在每一个状态,参与者知道的数据都必须是被授权的刻画.通过为 gen 和 $learn$ 事件添加数据被授权的卫式条件,保密性在保密性模型中平凡满足.

认证性用 Gavin Lowe 提出的非单射一致性和单射一致性刻画.在非单射一致性模型和单射一致性模型中,定义了 $Running(h,d)$和 $Commit(h,d)$中两个信号,其中,h 是协议角色列表,d 是多态类型δ数据,多态类型δ可以实例化为具体类型.描述模型状态的结构体只有一个成员 $sigs$,它是 $Running(h,d)$和 $Commit(h,d)$这两种信号组成的

多集.非单射一致性被刻画为:如果协议角色列表 h 中的角色都是诚实参与者,则 *sigs* 中一个 *Commit*(h,d)信号总有一个与之匹配的 *Running*(h,d)信号,单射一致性被刻画为 *sigs* 中 *Commit*(h,d)信号的数量不多于 *Running*(h,d)信号的数量.在非单射一致性模型和单射一致性模型中,定义了 *running*(h,d)和 *commit*(h,d)这两个事件,它们把对应的信号加入到 *sigs* 中.在模型中非单射一致性和单射一致性都用不变式刻画,为 *running*(h,d)和 *commit*(h,d)事件添加 *Commit*(h,d)信号与 *Running*(h,d)信号匹配的卫式条件,它们在非单射一致性模型和单射一致性模型中都平凡地满足.使用恒等函数作为中间函数、恒等关系作为模拟关系,单射一致性模型显然是非单射一致性模型的精化.

在第 0 层的规范模型中都没有引入攻击者及相关事件.

(2) 第 1 层是没有消息传递的卫式协议模型.

在模型中引入了协议角色和运行,运行是参与者以某一协议角色执行的进程,每一个运行都有一个局部存储用于保存状态信息.在卫式协议模型中,参与者直接从其他参与者的局部存储读/写消息.描述模型状态的结构体只有一个成员 *runs*,*runs* 是运行标识符到运行局部存储的部分函数,因为运行标识符是新生成的值,它们可以作为协议中的新鲜值(nonces)或者秘钥.运行的局部存储包括执行角色、协议参与者序偶、参与者局部状态、记录角色特定信息的框架.例如在卫式协议模型中,Needham-Schroeder-Lowe 协议的前两个步骤如下建模.

- $a_1\_step_1(A,B,N_a)\equiv\{(s,s')|N_a\notin dom(s.runs)\wedge s'.runs:=s.runs(N_a\rightarrow(A,B,Init(\perp)))\}$;
- $a_2\_step_2(A,B,N_a,N_b)\equiv\{(s,s')|N_b\notin dom(s.runs)\wedge s'.runs:=s.runs(N_b\rightarrow(A,B,Resp(N_a)))\}$.

其中,状态 s' 是 s 的后继状态,$N_a\notin dom(s.runs)$和 $N_b\notin dom(s.runs)$表明 N_a 和 N_b 是新鲜值,*runs* 将 N_a 和 N_b 映射为对应的局部存储,其中,*Init*(⊥)描述发起者的新鲜值未知,*Resp*(N_a)表示新鲜值 N_b 是对新鲜值 N_a 的响应.

在第 1 层卫式协议模型中仍然没有引入攻击者及其动作.定义中间函数将卫式协议模型状态变量 *runs* 中的知识和授权知识映射为保密性模型中的知识和授权知识,卫式协议模型是保密性模型的精化.定义中间函数,将卫式协议模型状态变量 *runs* 中的协议发起者、协议接收者、新鲜值、新鲜会话秘钥等映射为认证性模型中的 *Running*(h,d)和 *Commit*(h,d)两种信号,卫式协议模型是认证性模型的精化.

(3) 第 2 层是在秘密信道、认证信道等安全信道上的信道协议模型.

通信信道被划分为不安全信道、提供保密性和认证性的静态安全信道以及提供认证性的动态认证信道等多种类型.攻击者可以窃听非保密信道、基于初始知识和窃听消息可以在非认证信道上伪造消息,攻击者的初始知识包括所有参与者、不安全的秘钥和自然数集合.

信道协议模型中,描述模型状态的结构体中新增加了一个表示信道消息集合的成员 *chan*,从而扩展了卫式协议模型的状态.接收消息 M 时增加了形如 $M\in S.chan$ 的卫式条件,消除了卫式协议模型中可以直接从其他运行的局部存储中读/写消息的理想假设,其中,S 是信道协议模型的状态变量;发送消息 M 则用形如 $S'.chan:=S.chan\cup\{M\}$的动作刻画,其中,S'是 S 的后继状态.在信道协议模型中,Needham-Schroeder-Lowe 协议的第 2 个步骤描述如下:

$$a_2\_step_2(A,B,N_a,N_b)\equiv\{(s,s')|N_b\notin dom(s.runs)\wedge Confid(A,B,M_1(N_a,A))\in s.chan\wedge s'.runs:=s.runs(N_b\rightarrow(A,B,Resp(N_a)))\wedge s'.chan:=s.chan\cup\{Confid(B,A,M_2(N_b,N_a,B))\}\},$$

其中,

- *Confid*(A,B,M_1(N_a,A))表示在 A 和 B 之间具有保密性的安全信道上发送消息 M_1(N_a,A);
- *Confid*(B,A,M_2(N_b,N_a,B)表示在 B 和 A 之间具有保密性的安全信道上发送消息 M_2(N_b,N_a,B).

信道协议模型中还包括一个攻击者事件,刻画了攻击者基于信道消息集合 *chan* 和新鲜值伪造消息的能力.信道协议模型是卫式协议模型的精化,两者之间的模拟关系是标准投影算子.

(4) 第 3 层是密码协议模型.

信道协议模型中在安全信道上发送和接收的消息将用不安全信道上的加密消息代替,标准的 Dolev-Yao 攻击者控制整个网络,使用 Paulson 的归纳法建模安全协议攻击者,安全性质和协议攻击者基于 *parts*,*analz* 和 *synth* 这 3 个闭包算子形式化,其中,*parts*(H)计算消息集合 H 中所有消息的子项的闭包,*analz*(H)计算 H 中所有消息关

于分解操作和基于 H 中秘钥解密操作构造的子项的闭包,$synth(H)$计算 H 中所有消息组合的闭包.

在密码协议模型中,使用表示消息集合的变量IK代替了信道协议模型中的变量$chan$,初始化事件中变量IK包含协议攻击者的初始知识.在密码协议模型的事件中,接收消息 M 时增加了形如 $M\in S.IK$ 的卫式条件,其中,S是密码协议模型的状态结构体变量;发送消息 M 则用形如 $S'.IK:= S.IK\cup\{M\}$的动作刻画,其中,S'是 S 的后继状态.在密码协议模型中,Needham-Schroeder-Lowe 协议的第 2 个步骤描述如下:

$$a_2\_step_2(A,B,N_a,N_b)\equiv\{(s,s')|N_b\notin dom(s.runs)\wedge\{|N_a,A|\}_{pub(B)}\in s.IK\wedge s'.runs:=$$
$$s.runs(N_b\rightarrow(A,B,Resp(N_a)))\wedge s'.IK:=s.IK\cup\{\{|N_b,N_a,B|\}_{pub(A)}\}\},$$

其中,$\{|N_a,A|\}_{pub(B)}$表示用 B 的非对称公钥加密消息$\{|N_a,A|\}$,$\{|N_b,N_a,B|\}_{pub(A)}$表示用 A 的非对称公钥加密消息$\{|N_b,N_a,B|\}$.

密码协议模型中还包括一个攻击者事件,攻击者能够发送集合 $synth(analyz(S.IK))$中的所有消息.密码协议模型是信道协议模型的精化,两者之间的模拟关系依赖于具体协议的消息抽象函数,模拟关系是 4 个关系R_{23}^{msgs},R_{23}^{key},R_{23}^{non},R_{23}^{pres}的交集,其中,关系R_{23}^{msgs}描述变量IK中具体消息的子项的抽象包含在变量$chan$中,关系R_{23}^{key}和R_{23}^{non}描述信道协议模型中协议攻击者至少获取了密码协议模型中协议攻击者获取的秘钥和新鲜值,关系R_{23}^{pres}描述密码协议模型和信道协议模型中变量$runs$都具有同样的值.

规范模型、卫式协议模型、信道协议模型、密码协议模型这 4 个精化层次模型,从刻画保密性和认证性的规范模型出发,依次引入了新的状态变量 $runs$,$chan$ 和 IK,逐步消除了直接读/写局部存储、具有安全性的通信信道的理想假设,构造出消息传递形式的安全协议,不同层次模型之间具有精化关系,从而保证构造的消息传递形式的安全协议一定满足保密性和认证性等安全性规范.

基于逐步精化方法,他们构造出了 ISO/IEC 9798-3 协议和 Needham-Schroeder-Lowe 协议的前两步,它们是基于数字签名和公钥加密的单向实体认证协议.他们同时构造出了多个使用对称密钥的基于服务器的密钥传输协议,包括 Otway-Rees 协议的一个变种、Needham-Schroeder 共享密钥协议、Yahalom 协议和 Kerberos IV 和 KerberosV 协议的内核.Sprenger 等人进一步将安全协议形式化设计方法应用于密钥建立协议[21],他们开发出了 Needham-Schroeder 共享密钥协议、Kerberos IV 和 Kerberos V 协议的核心、Denning Sacco 协议,并且包含了一些实现细节,如密钥确认、重放缓冲区、加密票据.通过研究,他们确信,基于 Event-B 的形式化设计方法能够开发具有相当规模和复杂度的安全协议.

Sprenger 等人将安全协议设计划分为规范模型、卫式协议模型、信道协议模型、密码协议模型这 4 个精化层次,给出了安全协议形式化设计的可行方法,相邻精化层次模型之间语义比较接近,保证模型正确的证明义务更易于成功验证,因而他们成功地形式化设计出了已有的多个安全协议.基于中间函数、模拟关系和观察函数定义了模型之间更一般化的精化关系,与 Event-B 方法中模型精化的定义不同,因而 Rodin 等 Event-B 建模工具不能用于自动生成证明义务和自动验证生成的证明义务,需要在 Isabelle/HOL 中手工生成大量的证明义务并手工验证这些证明义务的有效性.在文献[22]中,Sprenger 等人将自动生成证明义务和自动验证证明义务作为未来研究工作之一.

基于 Event-B 和精化建模工具 Rodin,Nazim Benaissa 和 Dominique MÅLery 研究了安全协议基于证明的开发[25],由 4 个精化层次构成.

- 初始模型:协议运行的不同步骤用抽象事务建模,抽象事务具有源和目的地等多个属性,这些属性用于描述认证性等安全性质.
- 第 1 次精化模型:在初始模型中没有建模的其他协议元素,将在第 1 次精化模型中被建模.
- 第 2 次精化模型:协议攻击者的 Dolev-Yao 模型被建模.
- 第 3 次精化模型:精化主要针对秘钥建立协议,抽象事务将被具体的新鲜值或会话秘钥代替.

他们开发了 Needham-Schroeder 公钥认证协议和 Blake-Wilson-Menezes 秘钥传输协议.在建模工具 Rodin 自动生成的 285 个证明义务中,绝大部分证明义务被 Rodin 的定理证明工具自动验证,只有 5%左右的证明义务需要交互式验证.

4 基于 Event-B 方法的安全协议建模与验证

在 Sprenger 等人的研究基础上,基于 Event-B 和精化建模工具 Rodin,Huang 等人为基于 TPM(trusted platform module)的安全协议应用提出了细粒度的精化框架[26].修改了协议攻击者的 Dolev-Yao 模型,未受 TPM 保护的协议参与者将被协议攻击者控制.精化框架由 7 个精化层次构成.

- 初始模型:在初始上下文中定义了表示参与者集合的 *AGENT* 和表示消息集合的 *MSG*,在初始抽象机中定义了描述消息传输信道的变量 *channel*,定义了描述发送消息的 *Send* 事件和描述接收消息的 *Receive* 事件,*Send* 事件和 *Receive* 事件通过修改 *channel* 的值定义.
- 第 1 次精化模型:在第 1 次扩展上下文中,集合 *AGENT* 被分解为平台 *PLATFORM* 和 *TPM* 两个不相交的子集.在第 1 次精化抽象机中,*channel* 被分解为表示平台之间公共信道的变量 $chan_p$ 以及表示 TPM 和平台之间私有信道的变量 $chan_s$,*Send* 事件和 *Receive* 事件分别被精化为 *SendPub*,*SendfromTPM*,*SendtoTPM* 这 3 个事件和 *ReceivePub*,*ReceivefromTPM*,*ReceivebyTPM* 这 3 个事件.
- 第 2 次精化模型:在第 2 次扩展上下文中,*MSG* 被分解为结构化消息集合 *STRUCT*、公钥集合 $AKEY_p$、私钥集合 $AKEY_s$、非对称密文集合 *AENCRYPT*、对称密文集合 *SENCRYPT*、数字签名集合 *SIGN* 等多个子集.在第 3 次扩展上下文中,公钥集合 $AKEY_p$、私钥集合 $AKEY_s$ 被分解为 TPM 协议中的多个具体秘钥子集,并定义了平台的可信(trust).在 TPM 规范中,可信反映了设备按照某种特殊方式实现特定目标的预期,当平台的 TPM 未被攻破并且该 TPM 的 PCR 属于 PCR 信任列表时,则平台是可信的.在第 2 次精化抽象机中,引入了保存 TPM 公有消息和私有消息的变量 t_p,t_s 和保存平台公有消息和私有消息的变量 s_p,s_s,引入了刻画攻击者拥有知识集合的变量 m_p,基于引入的这些变量,定义了描述保密性的不变式.
- 第 3 次精化模型:第 2 次精化抽象机中的抽象变量 t_p,t_s 和 s_p,s_s 被 *TPM* 和 *PLATFORM* 的一组具体变量精化并替换.
- 第 4 次精化模型:区分 TPM 是否被攻破,事件 *SendPub* 被精化为 *SendPubt* 和 *SendPubf* 这两个事件,*SendPubt* 描述可信平台发送消息,*SendPubt* 事件可以继续被精化,*SendPubf* 描述不可信平台发送消息,通过分析和组合攻击者知识集合中的消息,*SendPubf* 可以向任意平台发送消息,*SendPubf* 不会被继续精化.
- 第 5 次精化模型:DAAODV 是一个基于 TPM 的路由协议,对 DAAODV 协议的秘钥协商过程进行了细粒度的精化.DAAODV 协议的秘钥协商有一些共有的消息处理操作,包括产生消息、消息发送和消息验证,第 5 次精化模型中,对这些共有的消息处理操作进行了建模,*SendPubt* 事件被精化为 *SendPubt_HSIGN*,*SendPubt_DSIGN* 和 *SendPubt_PSIGN* 这 3 个事件,*ReceivePub* 事件被精化为 *ReceivePub_HVERIFY*,*ReceivePub_DVERIFY* 和 *ReceivePub_PVERIFY* 这 3 个事件.
- 第 6 次精化模型:DAAODV 秘钥协商协议的细粒度事件被建模,精化 *SendPubt* 的事件主要描述组合和发送消息,精化 *ReceivePub* 的事件主要描述验证数字签名和生成秘密,协议的其他保密性和认证性约束被刻画为抽象机的不变式并被验证.

对于 DAAODV 秘钥协商协议的精化建模与验证,建模工具 Rodin 自动生成所有证明义务,其中大部分需要交互式证明,只有一些被 Rodin 自动验证.与其他基于 Event-B 方法的研究不同,基于 TPM 的安全协议应用的细粒度精化框架并没有遵循 Event-B 方法首先描述功能规范再逐步精化的建模方法,在初始化模型中没有定义保密性和认证性等安全性规范,这些安全性规范在第 6 次精化模型中才被刻画,这是导致大部分证明义务不能被 Rodin 自动验证的原因之一.

Snook 等人基于抽象和精化,并应用规范和验证提出了一种安全协议构造和分析方法[27].将 Event-B 和 iUML-B 中的类图和状态图作为建模工具,类图描述概念系统实体之间的关系,抽象机描述协议执行和实体之间的交互,安全性质作为抽象机中的不变式,所有事件都保持不变式描述的安全性质;然后,基于定理证明和模型检验证明自动生成的证明义务的有效性.在此基础上,他们对 VLAN(virtual local area network)标签协议进行

了精化建模与分析,其中,局域网由一些网络设备组成,它们通过中间路由设备交换数据.VLAN 标签协议模型由 3 个精化层次组成.

- 初始化模型 M_0:在初始上下文中引入了数据报文集合、虚拟局域网集合、结点集合,初始抽象机 M_0 主要建模安全性质,安全性质用初始抽象机中的不变式刻画,即,每一个数据报文只能发送到被数据报文拥有者允许访问的虚拟局域网.在事件中增加了数据报文发送目的地属于数据报文拥有者允许访问的虚拟局域网的卫式条件,使得不变式描述的安全性质平凡地成立.
- 第 1 次精化模型 M_1:网络结点包括网络设备和网络开关,网络设备可以生成数据报文,网络开关不能生成数据报文;网络设备只能访问它所在的虚拟局域网,网络开关可以访问所有的虚拟局域网.在第 1 次精化模型 M_1 中,结点集合被划分为不相交的网络设备集合和网络开关集合,发送数据报文事件被精化为向网络设备发送和向网络开关发送这两个事件.
- 第 3 次精化模型 M_2:引入标签集合,每一个数据报文都将分配标签,标签中包含 VLAN 的身份标识.引入标签,消除了数据报文发送事件中目的地属于它的拥有者允许访问的虚拟局域网的卫式条件.在 M_2 中,还对嵌套分配标签进行了建模.验证第 2 次精化模型 M_2 的证明义务时,一个不变式保持证明义务无法成功验证,他们使用 Rodin 中的模型检验工具 Prob,发现了 VLAN 标签协议的双标签(double tagging)攻击.

5 基于 Event-B 方法的安全协议实现代码精化验证

基于 Event-B 和验证工具 VCC,Polikarpova 等人验证了 TPM 的密钥管理模块的 C 语言代码是其消息传递形式协议的精化实现[28].TPM 是现代大多数个人电脑中的安全密码处理器,TPM 设备提供秘钥生成、随机数产生、加密、数字签名等密码操作功能.TPM 用自然语言描述的规范有 1 000 页左右,C 语言实现代码大约有 30 000 行.TPM 规范包含 102 个命令,包括生成秘钥(create)、载入秘钥(load)以及解密密文等.在 TPM 中,秘钥被层次化地组织,子秘钥被父秘钥保护,保护其他秘钥的秘钥称为存储秘钥.一个存储秘钥包括公开和敏感两个部分,公开部分包括一个非对称公钥以及一些属性和参数,敏感部分包括一个非对称私钥和一个对称保护秘钥.

Polikarpova 等人直接用 VCC 为安全协议建模.基于状态迁移系统为协议建模,状态用 ghost 结构体表示,为了实现高层抽象,ghost 结构体中使用了无界整数和无穷映射方便建模.状态迁移用带有契约的 ghost 函数表示,*ghost* 函数可以访问和修改 ghost 结构体中的对象.在每一个精化层次,状态用带有粘结不变式的 ghost 结构体表示,具体事件用带有契约(描述具体状态)的 ghost 函数表示.

在精化验证中,安全协议模型被划分为 3 个精化层次.

- 第 1 层模型是高层抽象协议定义.

初始化模型用符号化密码描述,其中,消息集合用项代数建模,项集合包括字节字符串类型的文字、描述密码操作(加密和 Hash 操作)的项以及描述消息组合的 Sensitive,Object 和 Private 这 3 种类型的复合项,实现代码中,这 3 种类型的组合项的格式不同.

初始模型中,变量用于描述协议执行的状态,包括描述对象(主要是秘钥)层次和载入对象的内部 TPM 状态、诚实参与者生成的描述协议消息的所有项以及网络攻击者通过窃听或构造得到的所有项,这些项集合通过 objects,loaded,prococol,attacker 等刻画函数用无穷映射表示.

初始化模型的状态用 ghost 结构体类型 LOG 的变量 *log* 表示,事件用伴有契约的 ghost 函数表示,每一个事件建模协议的一个步骤,它们对应于诚实主体发送、接收消息和安全协议攻击者基于已有知识构造出新消息.消息发送事件和消息接收事件都用 ghost 函数建模,消息发送会增加一些项到 *protocol* 刻画的项集合中,其中一些项被添加到 *attacker* 刻画的项集合中,同时会修改协议内部状态,消息接收事件不会修改 *protocol* 刻画的项集合和 *attacker* 刻画的项集合,但是会修改协议内部状态,消息接收事件的前置条件还增加了接收消息是网络攻击者的部分知识的断言,实际上刻画了网络攻击者窃听所有网络通信.安全协议攻击者的 Dolev-Yao 模型通常用一组逻辑规则描述,用事件建模 Dolev-Yao 模型的逻辑规则时,规则的前提条件将作为事件的前置条件,规则

结论作为事件的后置条件,安全协议攻击者生成的项将全部增加到 *attacker* 刻画的项集合中.

安全协议的保密性用 *log* 的不变式刻画,不变式约束 *attacker* 的项集,所有对象的敏感域都不能被攻击者获知.通过验证所有事件都保持不变式,完成保密性验证.认证性总是与诚实参与者接收到消息相关联,当接收消息时,诚实参与者希望能够确认消息来自于某一个参与者,因而认证性可以用接收事件的后置条件刻画.通过验证 *log* 中的所有事件保持不变式并满足其后置条件验证保密性和认证性,验证时需要增加一些辅助不变式.

- 第 2 层模型是从项代数到字节字符串.

初始化模型用符号项表示协议消息,在协议实现程序中,用字节字符串表示协议消息.为关联这两种表示,需要匹配符号项与字符串两种表示.为了匹配符号项和字符串表示,开发了一个 VCC 库,定义了类型 *ByteString*,用于表示任意大小的字节有穷序列,还定义了一系列规范函数管理这些字节有穷序列,规范函数 string 基于项的构造定义项的字符串格式.对于加/解密操作和 Hash 函数生成的密码项,假设它们基于一个可信的密码库实现,并能计算出对应的字节字符串,因而将它们用未解释函数 lib_encrypt 和 lib_hash 建模.

在第 2 层模型中引入了字符串处理操作的符号化假设:构造新项时,字符串处理操作生成的与项匹配的字符串不能被用于表示另外一个项;如果操作作用在一个特定类型的项上,则该操作不能作用于与其他类型项匹配的字符串上.符号化假设将保证字符串处理操作生成的字符串集合和协议执行时使用的项集不会有交集,字符串与项之间存在一个映射,在 ghost 结构体 Table 中保存了该映射以及描述第 1 层模型中的项和第 2 层模型中的字符串的匹配关系的粘结不变式.除了协议消息用字节字符串表示之外,第 2 层模型中的事件与第 1 层模型中事件是一一对应的.

- 第 3 层是实现代码.

TPM 实现代码中保存了载入对象列表和 *log* 中的载入集合相关联.实现代码中,TPM 对象表示为 *OBJECT* 结构体的实例,其中包括 *PUBLIC*,*SENSITIVE* 和 *SYM_KEY* 实例,*PUBLIC* 对象用于存储公开的非对称秘钥,*SENSITIVE* 对象用于存储保密的非对称秘钥,*SYM_KEY* 对象用于存储对称秘钥,所有秘钥都存储在带有长度域的静态分配的缓冲区中.为了访问存储在缓冲区中的字节字符串,在所有包含缓冲区成员的 ghost 结构体中都增加了 *content* 成员.在实现代码中没有用数据结构表示网络攻击者的知识,而是假设 TPM 协议的输入和输出信息都将被攻击者获取,输入和输出消息被攻击者获取都被编码到与外部世界有信息交换的 TPM 函数的前置条件或后置条件中.对于实现代码中的密码操作,在密码库中插桩了与第 2 层模型中引入的未解释函数 *lib_encrypt* 和 *lib_hash* 相关的契约.

Polikarpova 等人验证了 *Create* 和 *Load* 两个命令实现程序的安全性,发现了实现程序的两个错误:一个是内存错误,另一个是安全性错误.Polikarpova 等人认为,Event-B 方法的逐步精化能够处理安全协议规范和实现代码验证的复杂性:首先,对安全协议的高层抽象模型的安全性规范进行分析;然后,将验证后的高层抽象模型逐步向实现代码精化,在逐步精化过程中,既考虑了安全协议的安全性规范,又验证了安全协议实现代码;同时,具体事件是抽象事件的精化,因而不再需要验证具体事件满足抽象不变式,只需要验证粘结不变式,因而逐步精化实现了证明分解.

与 Sprenger 等人基于中间函数和模拟关系定义模型精化不同,Polikarpova 等人通过粘结不变式定义了相邻精化层次模型之间的关系,并借力 VCC 提供的验证 C 语言程序的功能,验证了 TPM 中 *Create* 和 *Load* 两个命令实现程序.与基于模拟关系定义模型精化相比较,通过粘结不变式定义相邻精化层次模型之间的关系,基于已有的 Event-B 建模工具可以自动生成并自动验证证明义务.Polikarpova 等人验证了 TPM 中 *Create* 和 *Load* 两个命令实现程序,对 TPM 以及其他实用系统实现代码的完整验证还需要更进一步的研究.

6 结束语

TLS 协议实现系统的脆弱性,表明安全协议形式化方法研究需要考虑安全协议实现代码的安全性,需要考虑实现代码是否满足安全协议需求规范.国外学者已经开始了安全协议实现代码是否满足需求规范的研究工作,例如,Sewell 等人开展了对 TCP、UDP 和 TLS 等实用系统的安全性分析研究[3,29,30].

安全协议需求规范和实现代码之间具有语义断层,直接验证实现代码满足需求规范不可行.基于 Event-B 方法等精化方法和 Isabelle/HOL、VCC 等验证工具,为安全协议需求规范建立形式化模型,可以发现需求规范中的不一致性错误.在需求规范形式化建模基础上迭代地逐步精化,建立不同精化层次的形式化模型,当精化模型与安全协议实现代码语义比较接近时,基于精化模型对安全协议实现代码开展一致性测试与验证,可以实现安全协议需求规范与实现代码之间的一致性测试和验证.因而,Event-B 等精化方法在安全协议实现代码以及实用系统形式化分析与验证中将发挥出重要作用.

基于 Event-B 模型的代码自动生成技术的研究进展[31,32],为安全协议代码自动生成提供了技术途径.形式化设计、建模和验证安全协议模型、验证安全协议实现代码以及实用系统核心模块正确性、基于模型的安全协议代码自动生成,将成为安全协议形式化方法研究的重点和热点.

References:

[1] Blanchet B. Security protocol verification: Symbolic and computational models. In: Degano P, Guttman JD, eds. Proc. of the 1st Int'l Conf. on Principles of Security and Trust. Heidelberg: Springer-Verlag, 2012. 3–29.

[2] Paulson LC. Inductive analysis of the Internet protocol TLS. ACM Trans. on Information and System Security, 1999,2(3):332–351.

[3] Kaloper-Mersinjak D, Mehnert H, Madhavapeddy A, Sewell P. Not-Quite-So-Broken TLS: Lessons in re-engineering a security protocol specification and implementation. In: Jung J, Holz T, eds. Proc. of the 24th USENIX Security Symp. USENIX Association, 2015. 223–238.

[4] Avalle M, Pironti A, Sisto R. Formal verification of security protocol implementations: A survey. Formal Aspects of Computing, 2014,26(1): 99–123.

[5] Aizatulin M, Gordon AD, Jürjens J. Extracting and verifying cryptographic models from C protocol code by symbolic execution. In Chen Y, Danezis G, Shmatikov V, eds. Proc. of the 18th ACM Conf. on Computer and Communications Security. ACM Press, 2011. 331–340.

[6] Aizatulin M, Gordon AD, Jürjens J. Computational verification of C protocol implementations by symbolic execution. In: Yu T, Danezis G, Gligor VD, eds. Proc. of the 19th ACM Conf. on Computer and Communications Security. ACM Press, 2012. 712–723.

[7] Bhargavan K, Fournet C, Gordon AD. Modular verification of security protocol code by typing. In: Hermenegildo MV, Palsberg J, eds. Proc. of the 37th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. ACM Press, 2010. 445–456.

[8] Fournet C, Kohlweiss M, Strub PY. Modular code-based cryptographic verification. In: Chen Y, Danezis G, Shmatikov V, eds. Proc. of the 18th ACM Conf. on Computer and Communications Security. ACM Press, 2011. 341–350.

[9] Swamy N, Chen J, Fournet C, Strub PY, Bhargavan K, Yang J. Secure distributed programming with value-dependent types. In: Chakravarty MMT, Hu ZJ, Danvy O, eds. Proc. of the 16th ACM SIGPLAN Int'l Conf. on Functional Programming. ACM Press, 2011. 266–278.

[10] Abrial JR, Hallerstede S. Refinement, decomposition, and instantiation of discrete models: Application to event-B. Fundamenta Informaticae, 2007,77(1-2):1–28.

[11] Greenaway D, Andronick J, Klein G. Bridging the gap: Automatic verified abstraction of C. In: Beringer L, Felty AP, eds. Proc. of the 3rd Int'l Conf. on Interactive Theorem Proving. Heidelberg: Springer-Verlag, 2012. 99–115.

[12] Winwood S, Klein G, Sewell T, Andronick J, Cock D, Norrish M. Mind the gap: A verification framework for low-level C. In: Berghofer S, Nipkow T, Urban C, Wenzel M, eds. Proc. of the 22nd Int'l Conf. on Theorem Proving in Higher Order Logics. Heidelberg: Springer-Verlag, 2009. 500–515.

[13] France RB, Rumpe B. Model-Driven development of complex software: A research roadmap. In: Briand LC, Wolf AL, eds. Proc. of the Workshop on the Future of Software Engineering. IEEE Computer Society, 2007. 37–54.

[14] Back RJ. On the correctness of refinement steps in program development [Ph.D. Thesis]. Helsinki: University of Helsinki, 1978.

[15] Morgan C. Programming from Specifications. 2nd ed., Prentice-Hall, 1994. 1–250.

[16] Woodcock J, Davies J. Using Z, Specification, Refinement, and Proof. Prentice-Hall, 1996.

[17] Abrial JR. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2010.

[18] Abrial JR. From Z to B and then event-B: Assigning proofs to meaningful programs. In: Proc. of the IFM 2013. 2013. 1–15.

[19] Abrial JR, Butler MJ, Hallerstede S, Hoang TS, Mehta F, Rodin VL. An open toolset for modelling and reasoning in event-B. Int'l Journal on Software Tools for Technology Transfer, 2010,12(6):447−466.

[20] Sprenger C, Basin DA. Developing security protocols by refinement. In: AI-Shaer E, Keromytis AD, Shmatikov V, eds. Proc. of the 17th ACM Conf. on Computer and Communications Security. ACM Press, 2010. 361−374.

[21] Sprenger C, Basin DA. Refining key establishment. In: Chong S, ed. Proc. of the 25th IEEE Computer Security Foundations Symp. IEEE Computer Society, 2012. 230−246.

[22] Sprenger C, Basin DA. Refining security protocols. Journal of Computer Security, 2018,26(1):71−120.

[23] Paulson LC. Isabelle: The next 700 theorem provers. In: Odifreddi P, ed. Proc. of the Logic and Computer Science. Academic Press, 1990. 361−386.

[24] Cohen E, Dahlweid M, Hillebrand MA, Leinenbach D, Moskal M, Santen T, Schulte W, Tobies S. VCC: A practical system for verifying concurrent C. In: Berghofer S, Nipkow T, Urban C, Wenzel M, eds. Proc. of the 22nd Int'l Conf. on Theorem Proving in Higher Order Logics. Heidelberg: Springer-Verlag, 2009. 23−42.

[25] Benaïssa N, Méry D. Proof-Based design of security protocols. In: Ablayev FM, Mayr EW, eds. Proc. of the 5th Int'l Computer Science Symp. in Russia. Heidelberg: Springer-Verlag, 2010. 25−36.

[26] Huang WC, Xiong Y, Wang XF, *et al*. Fine-Grained refinement on TPM-based protocol applications. IEEE Trans. on Information Forensics and Security, 2013,8(6):1013−1026.

[27] Snook C, Hoang TS, Butler M. Analysing security protocols using refinement in iUML-B. In: Barrett C, Davies M, Kahsai T, eds. Proc. of the NASA Formal Methods 2017. Heidelberg: Springer-Verlag, 2017. 84−98.

[28] Polikarpova N, Moskal M. Verifying implementations of security protocols by refinement. In: Proc. of the VSTTE 2012. 2012. 50−65.

[29] Bishop S, Fairbairn M, Norrish M, Sewell P, Smith M, Wansbrough K. Engineering with logic: HOL specification and symbolic-evaluation testing for TCP implementations. In: Proc. of the POPL 2006. 2006. 55−66.

[30] Bishop S, Fairbairn M, Norrish M, Sewell P, Smith M, Wansbrough K. Rigorous specification and conformance testing techniques for network protocols, as applied to TCP, UDP, and sockets. In: Guerin R, Govindan R, Minshall G, eds. Proc. of the ACM SIGCOMM 2005 Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications. ACM Press, 2005. 265−276.

[31] Fürst A, Hoang TS, Basin DA, Desai K, Sato N, Miyazaki K. Code generation for event-B. In: Albert E, Sekerinski E, eds. Proc. of the 11th Int'l Conf. on Integrated Formal Methods. Heidelberg: Springer-Verlag, 2014. 323−338.

[32] Méry D, Monahan R. Transforming event B models into verified C# implementations. In: Lisitsa A, Nemytykh AP, eds. Proc. of the 1st Int'l Workshop on Verification and Program Transformation. In: Proc. of the EPiC Series in Computing. 2013. 57−73.

李梦君(1975－),男,湖北云梦人,博士,副教授,CCF 专业会员,主要研究领域为形式化方法与技术信息安全技术,CPU 验证.

欧国东(1977－),男,博士,助理研究员,主要研究领域为计算机体系结构微处理器设计,形式化验证.

潘国腾(1977－),男,博士,副研究员,CCF 专业会员,主要研究领域为计算机体系结构微处理器设计,形式化验证.