基于 PUF 的硬件辅助软件认证方法

李雪筠[1]，　叶　靖[2,3]，　黄正峰[1]，　李晓维[2,3]，　李华伟[2,3]

(1. 合肥工业大学 电子科学与应用物理学院　安徽 合肥 230601；2. 中国科学院计算技术研究所 计算机体系结构国家重点实验室　北京 100190；3. 中国科学院大学　北京 100190)

摘要：提出了一种基于物理不可克隆函数(PUF)的硬件辅助软件认证方法,包括软硬件认证协议、嵌入 PUF 的 RISC-V 设计、软件混淆技术三部分,通过插入独立式 PUF 认证点或嵌入式 PUF 认证点,将软件与硬件绑定在一起,软件仅在特定的授权设备上才能正确执行,并从硬件和软件层面解决了 PUF 可靠性的问题。实验结果表明,所提方法的额外硬件开销小于 2%,基准程序测试结果表明单个认证点的额外性能开销小于 0.5%。

关键词：软件保护；硬件辅助；物理不可克隆函数；RISC-V

中图分类号：TP309.2　　**文献标志码**：A　　**文章编号**：1671-6841(2021)01-0088-07

DOI：10.13705/j.issn.1671-6841.2020268

0　引言

近年来,随着物联网的飞速发展,互联嵌入式设备的数量持续增长,广泛应用于家居、医疗、工业等领域。然而由于大多数互联嵌入式设备的安全机制不足,针对其软件的盗版或篡改等攻击愈演愈烈,给制造商带来严重的经济损失。因此,保护软件的完整性和防篡改性尤为重要。现有软件保护策略主要分为软件增强和硬件辅助两类。

传统的软件增强策略通常分为混淆、防篡改以及水印三种,通过对软件代码自身进行修改,使攻击者无法篡改程序或篡改后的程序难以执行。混淆处理侧重于防止逆向工程,通过增加程序的复杂性,使软件代码难以理解[1-2],从而阻止攻击者对软件的更改。常见的代码混淆主要包括布局混淆、数据流混淆、控制流混淆和预防性混淆[3],其中控制流混淆包括使用不透明谓词、插入多余控制流等方式;防篡改通过添加防篡改代码等方式检测程序是否被篡改,一旦发现篡改将终止程序执行。代码自检方案[4-5]采用自修改和自解密代码段来防止软件篡改;数字水印有静态和动态两种,用以发现软件盗版时证明软件的所有权[6]。然而,由于大多数设备制造商使用相似的微控制器和微处理器,硬件执行环境的兼容使得针对特定软件的攻击应用于其他设备,而越来越多的嵌入式设备通过互联网连接在一起,也使得病毒类攻击的传播更广泛。

通过硬件辅助来保护软件方面,也有大量的研究和开发。一种常见的方案是使用加密狗,但其用户使用感不好,使用方法复杂且易受攻击。文献[7]提出了两种针对加密狗的攻击,文献[8]对加密狗的软件保护强度进行了评估,结果表明,加密狗提供的保护非常有限。文献[9]通过软硬件绑定来提高软件的混淆程度,其通过修改指令集以适用软件保护的方法,将性能开销转换为软件开发和硬件采用指令集架构(instruction set architecture, ISA)新指令的额外成本。文献[10]提出了一种基于物理不可克隆函数(physical unclonable function,PUF)的指令级认证方法,利用 PUF 对指令进行混淆,每个指令分为两部分存储在内存中:混淆操作码的指令和 PUF 的激励。PUF 响应生成实际操作码以解码指令流水中的指令。利用 PUF 的不可克隆性,可以提高安全级别。但是,对每一条指令都进行 PUF 认证非常耗时,性能会受到影响;其次,指令认证依赖于 PUF 的可靠性,在程序执行过程中,一旦某一条指令对应 PUF 的激励响应对(challenge-response pair, CRP)受到如温度、电压等的干扰而出错,整个程序也随之出错。文献[11]采用 DPUF 提高 PUF 的可靠性,

收稿日期:2020-08-19

基金项目:国家自然科学基金项目(61704174, 61532017, 61432017, 61521092)。

作者简介:李雪筠(1995—),女,硕士研究生,主要从事集成电路安全研究,E-mail:lixueyun95@163.com;通信作者:叶靖(1988—),男,副研究员,主要从事集成电路设计、测试与安全研究,E-mail:yejing@ict.ac.cn。

但无法保证百分百的可靠性,同样存在着性能影响大和 PUF 可靠性难以保证等问题。

为解决上述问题,本文提出了一种基于 PUF 的硬件辅助软件认证方法,在嵌入式处理器中添加 PUF,软件开发人员则在特定的位置插入 PUF 认证点,程序运行到相应的位置就会进行 PUF 的认证,将软件的执行与硬件绑定,即一个编译好的软件只能运行在特定的硬件之上,包括软硬件认证协议、嵌入 PUF 的 RISC-V 设计和软件混淆技术三部分。本文的主要贡献包括:

1) 运行在 RISC-V 上的软件通过调用 PUF,使得一个软件的执行可以被限制在特定的硬件上,以保护软件不会在未经授权的硬件平台上执行;

2) 所提方法可以从硬件和软件两个层面解决 PUF 可靠性的问题;

3) 所提方法能够从软件层面上防止一定程度的逆向攻击。

1　研究背景

1.1　物理不可克隆函数

物理不可克隆函数,即 PUF,是一种硬件安全模块,能够提取和放大集成电路制造过程中的工艺偏差,对于每一个输入的激励,都会输出一个独特的响应,形成激励响应对,即 CRP。因为生产制造过程中的工艺偏差是不可复制的,所以不同芯片的 CRP 也各不相同,在物理层面难以克隆,可用作芯片独有的标识使其具有防伪功能。

仲裁器 PUF[12]是一种典型的强 PUF,它通过比较两条路径传输同一跳变的时延来确定响应。这两条路径具有相同的设计时延,但实际时延由于工艺偏差并不相同。文献[13]提出可调仲裁器 PUF,包括传统仲裁器 PUF 模块和调整电路模块两部分。由于传统仲裁器 PUF 可在现场可编程门阵列(field programmable gate array, FPGA)上实现且硬件开销较适中,因此图 1 的可调仲裁器 PUF 选择传统仲裁器 PUF 而不是其他强 PUF。在实际应用中,仲裁器 PUF 会受到温度、湿度、电压等外界环境的干扰,当两条路径的时延非常接近时[14],存在着可靠性的问题。

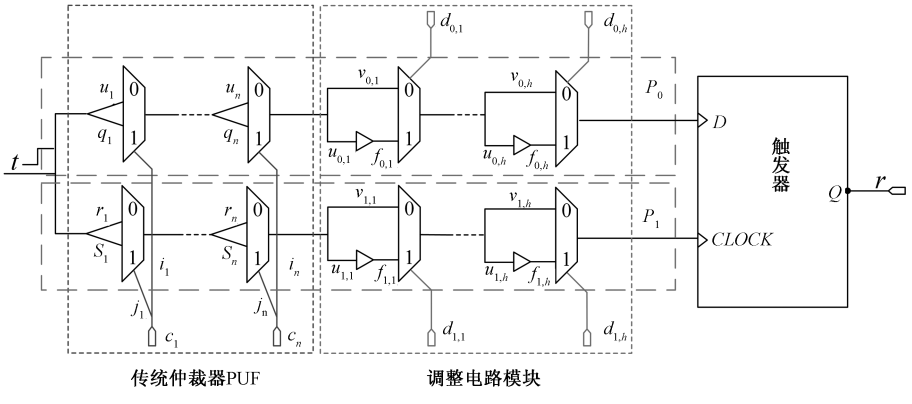

图 1　可调仲裁器 PUF

Figure 1　Adjustable arbiter PUF

1.2　RISC-V

RISC-V 架构是第五代精简指令级(reduced instruction set computer, RISC)架构,是一个开放且免费的架构,由基本指令集和可选指令集扩展组成,逐渐成为嵌入式系统日益流行的架构。可用于存储设备中的控制处理器[15]、USB 安全加密狗[16]、使用安全硬件区域构建可信执行环境[17]等。

VexRiscv(https://github.com/SpinalHDL)实现了一个由 SpinalHDL 编写的 32 位 RISC-V 开源处理器。SpinalHDL 是一种基于 Scala 的硬件描述语言,通过使用简单的单元(触发器、逻辑门等)建立新的抽象级别,帮助设计人员重用代码以避免重复编写。SpinalHDL 中几百行代码能够实现的功能在 Verilog 中可能需要上千行,但需通过专用的编译器生成 Verilog 来进行硬件实现。VexRiscv 支持 RV32IMCA 指令集,具有五级流水线,所有组件均作为插件实现的设计使其模块化和易集成,提供可选 Cache、IO 外设和总线等。目前,VexRiscv 已经支持在 Linux 操作系统上运行。

2 基于 PUF 的硬件辅助软件认证方法

本节具体介绍提出的基于 PUF 的硬件辅助软件认证方法,分为软硬件认证协议、嵌入 PUF 的 RISC-V 设计和软件混淆技术三部分。软硬件认证协议具体阐述了软件利用硬件 PUF 实现认证的完整过程,即如何将软件的执行与特定的嵌入 PUF 绑定在一起。嵌入 PUF 的 RISC-V 设计介绍 PUF 在 RISC-V 中的硬件实现。软件混淆技术则是介绍在软件中设计 PUF 认证点的具体实现方式。

2.1 威胁模型与软硬件认证协议

本节提出了基于 PUF 的软硬件认证协议,将软件的执行与特定的硬件设备绑定以保护软件。首先提出了参与的各方和威胁模型,然后为了抵抗该攻击,提出了相应的软硬件认证协议。

软硬件认证协议包括有硬件供应商、应用商店和用户三方。硬件供应商提供内嵌有 PUF 的处理器或包含该处理器的设备。应用商店提供具有特定 ID 的应用程序,不同 ID 的不同应用程序可以在同一硬件处理器上运行。用户则根据需要购买硬件和应用程序。参与的三方的威胁模型如下。假设硬件供应商和应用商店是值得信赖的,不会泄露 PUF 的 CRP,且它们之间的通信是安全的。但是用户可能会成为攻击者,应用商店与用户之间的通信也可能是不安全被窃听的。攻击者可以通过假扮用户购买或窃取具有特定 ID 的应用程序,然后将其复制分发给未经授权的用户。所提方法在将特定 ID 的应用程序分发到用户前,插入 PUF 的认证点对其进行混淆,用户从攻击者处克隆得来的应用程序则会因无法通过 PUF 的认证而失效,因为基于 PUF 的软硬件认证协议将软件的执行与特定的硬件设备绑定,从而能够有效地阻止此类克隆复制攻击。

为了抵抗该攻击,提出了图 2 所示的软硬件认证协议。首先,硬件供应商在处理器中设计内嵌的 PUF,收集 PUF 的 CRPs 并在处理器出售给用户之前将其存储在安全的数据库中,在该过程中,所提方法会对 CRP 的可靠性进行判别,具体细节将在后文中叙述。当用户请求购买硬件时,硬件供应商找到并销售内嵌有 PUF 的处理器或包含该处理器的设备 C 给用户。用户在使用时,为购买和使用某一特定的应用程序 A,发送 C 和 A 的序列号($ID(C)$和 $ID(A)$)给应用商店,应用商店发送 $ID(C)$给硬件供应商以获取 P,即 PUF 的 CRP,并利用 P 对程序 A 插入 PUF 的认证点进行混淆,得到混淆后的程序 A',然后将 A'发送给用户。最终,用户可以在 C 上运行程序 A'。

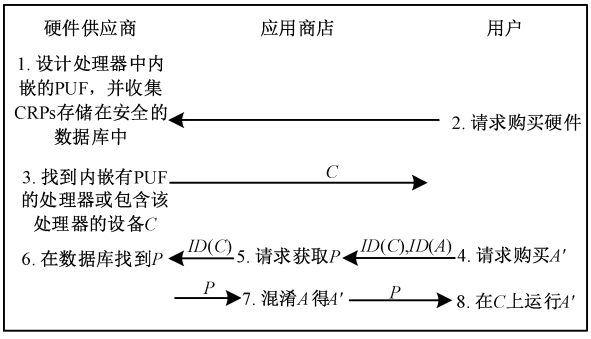

图 2 软硬件认证协议

Figure 2 Software and hardware authentication protocol

2.2 嵌入 PUF 的 RISC-V 设计

本节将从硬件和软件两个方面介绍在 VexRiscv 片上系统(system on chip, SoC)中嵌入 PUF 的设计方法。

首先在硬件层面,利用 APB 总线,以模块化方式嵌入 PUF 以扩展 VexRiscv SoC。所提嵌入 PUF 的 VexRiscv SoC 硬件设计如图 3 所示,包括 JTAG 接口、VexRiscv CPU、片上存储器、APB 总线及其挂载的多个外设——GPIO、UART 和嵌入 PUF 模块等。设计的 APB3PUF 模块可以将 PUF 挂载到 APB 总线上,此模块一侧连接到 APB 总线,另一侧连接到激励响应模块,包含数据寄存器 *challenge* 和 *response* 以及控制寄存器 *number*、*valid* 和 *done*。其中,*challenge* 为 32 位的激励,*response* 为产生的 1~32 位的响应,*number* 控制产生响应的位数,*valid* 表征激励响应模块是否已准备好被访问,*done* 表征是否已完成对其的访问。激励产生模块

由线性反馈移位寄存器(linear feedback shift register, LFSR)等构成,PUF 采用文献[13]中的可调仲裁器 PUF,每一个 32 位的 *challenge* 产生一个单比特位的 *response*。当 *valid* 为 1 时,激励扩展模块根据 *number* 对 *challenge* 进行扩展,产生多个不同的 *challenge*;然后,多个不同的 *challenge* 依次输入给 PUF,产生多个单比特位的 *response*;最后,响应收集模块根据 *number* 收集指定位数的 *response*,并写回到 *response* 寄存器中,同时将 *done* 置为 1,表示响应已产生。图 4 为设计流程图。首先,要为 PUF 模块分配相应的地址空间以实现软件的访问,需要修改并添加 scala 文件。其次,经过 scala 专有编译器编译以生成 Verilog 文件,并对其进行嵌入 PUF 模块的添加。最后,经过逻辑综合和布局布线生成嵌入 PUF 的 VexRiscv SoC 硬件电路。此设计是基于 FPGA 的可编程逻辑(progarmmable logic,PL)实现。

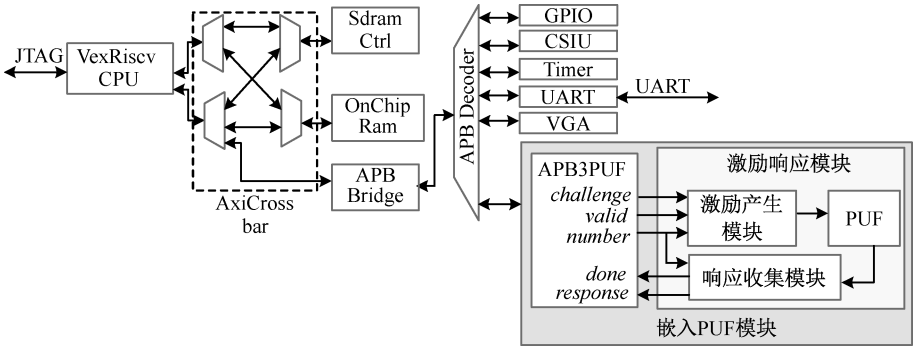

图 3　嵌入 PUF 的 VexRiscv SoC 硬件设计

Figure 3　VexRiscv SoC hardware design with PUF embedded

图 4　设计流程图

Figure 4　Design flow diagram

在软件层面,VexRiscv SoC 上运行的软件通过读写映射到地址空间的控制和数据寄存器,以访问 APB 总线上的不同外设。本文预定义了一个激励响应函数 *get_response*(*challenge*, *number*),通过调用此函数可以访问嵌入 PUF 模块。其中,输入数据 *challenge* 和 *number* 通过 APB 总线写入 APB3PUF 模块对应的寄存器中,然后将 *valid* 置为 1,之后该函数将持续监视 *done* 的值,当 *done* 为 1 时,从 *response* 寄存器中读出 PUF 的响应。为保障 PUF、CRPs 的安全,可以禁止一般用户调用该函数。

2.3　软件混淆技术

本节具体提出了两种软件混淆策略:独立式 PUF 认证点与嵌入式 PUF 认证点以混淆软件。

独立式 PUF 认证点与原程序无关,可以在程序中任意位置插入认证函数来保护原程序,当程序执行到该位置时,通过调用嵌入 PUF 模块来完成 PUF 认证,只有认证通过程序才能继续向下执行,否则程序将终止运行并报错。算法 1 给出了一个独立式 PUF 认证点的实现示例函数 *PUF_AUTH*()。其中 *auth_challenge* 和 *auth_response* 是硬件供应商收集的 CRPs 数据库中的一组随机的激励响应对,*PUF_AUTH*()利用激励响应函数 *get_response*(*auth_challenge*, 32)来调用嵌入 PUF 模块,并比较返回的 32 位响应 *this_response* 与程序中预先写入的 *auth_response*。如果它们相同的位数达到某一阈值 *threshold*,则认证通过。此方法通过调整阈值能够从软件层面解决 PUF 可靠性问题,不需要特意挑选可靠的 CRPs,但需要产生多比特位的响应。阈值的设置取决于 PUF 的可靠性,当阈值所要求的可靠位数超过产生的多比特响应中的最大可靠位数时,软件可能出现误判而导致执行失败。已知可调仲裁器 PUF 的可靠性为 94%,则 n bit 响应中至少有 m bit 可靠的概率 M 为:$M = \sum_{i=m}^{n} C_n^i (0.94)^i (0.06)^{n-i}$。当产生 32 位的响应时,由上式计算得最少有 24 位可靠的概率为 99.99%,因此阈值最大可设置为 75%,且软件不会误判。同时,为了保证软件无法在错误的硬件上正确执行,可以插入多个认证点。独立式 PUF 认证点因其独立于原程序之外的特性,易被逆向分析破解,一旦独立式 PUF 认证点的认证函数被攻击者识别并删除,程序在未经授权的硬件上也将能够正确执行。

嵌入式 PUF 认证点是通过修改原程序中的部分代码以实现控制流混淆,增加攻击者通过逆向分析攻击

程序的难度,能够在一定程度上逆向攻击。混淆后的程序只有在授权的硬件设备上执行时,才与原程序的执行结果一致,否则程序执行结果将会因与原程序不同而出错。算法2给出了一个嵌入式PUF认证点的实现示例代码。以条件语句为例,通过调用激励响应函数 *get_response*(*challenge*, 1),将 *challenge*1 和 *challenge*2 输入到APB总线上的嵌入PUF模块,并得到两个单比特位响应。当这两个响应相同时,混淆后的程序与原程序的控制流一致,能够通过PUF认证而正确地继续执行;否则,混淆后程序的控制流将出错,导致程序执行结果将出错。此方法仅需要产生单比特位的响应,但需要挑选可靠的CRPs,否则一旦这个响应不可靠,软件在正确的硬件上将不能正确执行。

算法1 插入独立式PUF认证点

```
Function PUF_AUTH( )
{
  SET this_response = get_response (auth_challenge, 32)
  SET num = different bits between this_response and auth_response
  IF (num>threshold)
    RETURN FALSE
  ELSE
    RETURN TRUE
}
```

算法2 插入嵌入式PUF认证点

原程序	混淆后的程序
	IF (*get_response* (*challenge*1, 1) = = *get_response* (*challenge*2, 1)
IF (*condition* is satisfied)	{
{	IF (*condition* is satisfied)
CONDUCT *A*	CONDUCT *A*
}	ELSE
ELSE	CONDUCT *B*
{	ELSE
CONDUCT *B*	IF (*condition* is satisfied)
}	CONDUCT *B*
	ELSE
	CONDUCT *A*
	}

为此,基于可调仲裁器PUF的设计,我们进一步提出了判断CRP是否可靠的方法,能够利用硬件电路设计快速检测单个CRP的可靠性。由于只有当两条路径的时延差较小时,响应才可能因不稳定而导致可靠性问题。因此,本文利用调整信号对两条路径的时延差大小进行分析,并选择合适的或较大的 δ,可以有效判定响应是否可靠。具体原理如下,图1中假设 P_0 和 P_1 分别为可调仲裁器PUF的上下两条路径,在某一个激励输入下,如果调整信号 $d_{0,1}\sim d_{0,h}=d_{1,1}\sim d_{1,h}=0$ 时,输出响应为1,说明 P_0 的时延小于 P_1 的时延。此时,如果调整信号变为 $d_{0,1}\sim d_{0,h}=1, d_{1,1}\sim d_{1,h}=0$,相当于在路径 P_0 上增加时延 δ,若输出响应变为0,则说明 P_1 的时延减去 P_0 的时延小于 δ;若响应仍为1,则说明 P_1 的时延减去 P_0 的时延大于 δ。δ 的大小可以通过设置调整信号值来改变。

3 实验结果及分析

3.1 实验设置

实验操作系统为Ubuntu14.04,软件为Vivado2018.1,处理器为时钟频率100 MHz的基于VexRiscv的SoC系统,开发板为Digilent公司推出的Xilinx Artix-7 35T FPGA的Arty开发板。

3.2 实验评估

本节分别从硬件开销、性能和可靠性三个方面对所提方法进行评估。

首先,评估嵌入PUF带来的额外硬件开销。表1列出了主要使用的资源触发器(flip flop, FF)和查找表(look up table,LUT)的额外开销,分别为1.98%和0.81%。与已有文献[9-10]需要6个PUF相比,本文只需插入单个PUF。

然后,评估设计对性能的影响。如表2所示,调用单个基于32位响应的独立式PUF认证点所需运行时间为12.74 μs,调用单个基于单比特响应的嵌入式PUF认证点所需运行时间为9.10 μs。图5给出了基准程序Dhrystone插入不同PUF认证点的额外性能开销。从图5可以看出,插入的PUF认证点越多,对性能造成的影响越大,在实验中,插入10个PUF认证点增加了4.39%的时间开销。软件设计者也可以根据实际软件运行过程和时间,调整插入认证点的位置和个数。

表 1　嵌入 PUF 的额外硬件开销

Table 1　Embedded PUF additional hardware overhead

类型	额外硬件开销/%
LUTs	1.98
FFs	0.81

表 2　单个 PUF 认证点运行时间

Table 2　Running time of a single PUF certification point

类型	运行时间/μs
独立式	12.74
嵌入式	9.10

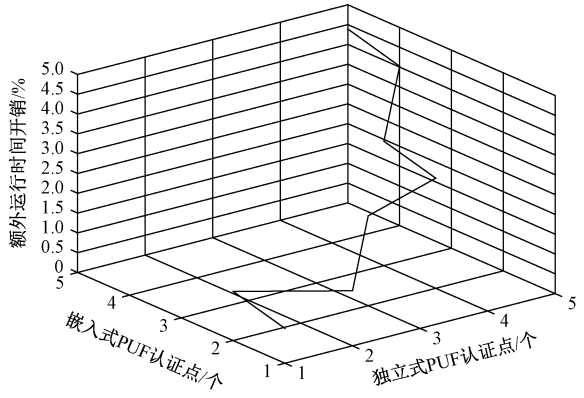

图 5　Dhrystone 的额外性能开销比较

Figure 5　Additional performance overhead of Dhrystone

最后,分析 PUF 的可靠性对设计的影响。文献[9-10]是基于指令级的 PUF 认证,要求每一条指令都经过 PUF 的认证,程序能否正确执行强烈依赖于 PUF 的数据是否可靠。然而,PUF 在实际应用中,会受到如电压、温度等的影响,使得 PUF 的 CRPs 可能变得不可靠。本文所提方法分别从硬件和软件两个层面解决 PUF 可靠性问题。对于独立式 PUF 认证点而言,在软件层面可以根据 PUF 的可靠性设置合适的认证阈值来避免误判的出现,由 2.3 小节的理论计算得出,阈值为 75%时软件不会误判,且在实验中多次的测试验证也证明了理论计算的正确性。此外,也可以通过使用同一激励多次测试的方法进一步提高可靠性。独立式 PUF 认证点的认证方式决定了其需要通过多比特位进行认证,若不在软件层面设置阈值,则需要选出较多的有相同响应且完全可靠的 CRP 对,搜索满足条件的 CRP 是非常耗时的,因此采用软件添加阈值的方式进行认证更加可行。对于嵌入式 PUF 认证点而言,通过调整信号对单个 CRP 单个响应比特是否可靠进行判别,在无须任何软件层面的辅助是可行的。实验证实,所有被判别为可靠的 CRP 在实际运行过程中确实产生了稳定不变的输出响应。因此,所提方法能够解决 PUF 可靠性难以保障的问题,不会出现因为可靠性的问题而使得软件程序无法运行或出错的情况。

4　总结

本文提出了一种基于 PUF 的硬件辅助软件认证方法,包括软硬件认证协议、嵌入 PUF 的 RISC-V 设计和软件混淆技术,能够保护软件不在未经授权的设备上执行。通过在基于 VexRiscv 的 SoC 嵌入式系统中实现,验证了此方法的可行性。实验结果表明,所提方法的额外硬件开销小于 2%,经过基准程序测试,单个认证点的额外性能开销小于 0.5%,并从硬件和软件层面解决了 PUF 可靠性的问题。

参考文献:

[1]　COHEN F B. Operating system protection through program evolution[J]. Computers & security, 1993, 12(6): 565-584.

[2]　OGISO T, SAKABE Y, SOSHI M, et al. Software obfuscation on a theoretical basis and its implementation[J]. IEICE transactions on fundamentals of electronics, communications and computer science, 2003, E86-A(1): 176-186.

[3]　SU Q, WANG Z Y, WU W M, et al. Technique of source code obfuscation based on data flow and control flow tansformations [C]//International Conference on Computer Science & Education. Melbourne, 2012: 1093-1097.

[4]　AUCSMITH D. Tamper resistant software: an implementation[C]//International Workshop on Information Hiding. Cambridge, 1996: 317-333.

[5] HORNE B, MATHESON L, SHEEHAN C, el al. Dynamic self-checking techniques for improved tamper resistance[C]//ACM Workshop on Digital Rights Management. Philadelphia, 2001:141-159.
[6] 罗昊, 谢晓尧, 彭长根. 基于直方图平移的加密域可逆水印算法[J]. 郑州大学学报(理学版), 2018, 50(2): 29-34.
LUO H, XIE X Y, PENG C G. Reversible watermarking algorithm in histogram shifting based on encrypted domain[J]. Journal of Zhengzhou university (natural science edition), 2018, 50(2): 29-34.
[7] MITCHELL W P R. Protecting secret keys in a compromised computational system[C]//International Workshop on Information Hiding. Dresden, 1999: 448-462.
[8] PIAZZALUNGA U, SALVANESCHI P, BALDUCCI F, et al. Security strength measurement for dongle-protected software[J]. IEEE security & privacy, 2007, 5(6): 32-40.
[9] SCHRITTWIESER S, KATZENBEISSER S, MERZDOVNIK G, et al. AES-SEC: improving software obfuscation through hardware-assistance[C]//The 9th International Conference on Availability, Reliability and Security. Fribourg, 2014: 184-191.
[10] ZHENG J X, DONGFANG L, POTKONJAK M. A secure and unclonable embedded system using instruction-level PUF authentication[C]//The 24th International Conference on Field Programmable Logic and Applications. Munich, 2014: 1-4.
[11] ZHENG J X, XU T, POTKONJAK M. Securing embedded systems and their IPs with digital reconfigurable PUFs[C]//The 26th International Workshop on Power and Timing Modeling, Optimization and Simulation. Bremen, 2016: 169-176.
[12] LIM D, LEE J W, GASSEND B, et al. Extracting secret keys from integrated circuits[J]. IEEE transactions on very large scale integration systems, 2005, 13(10): 1200-1205.
[13] YE J, LI X W, LI H W, et al. Adjustable arbiter physical unclonable function with flexible response distribution[C]//China Semiconductor Technology International Conference. Shanghai, 2019: 1-3.
[14] YE J, HU Y, LI X W. VPUF: voter based physical unclonable function with high reliability and modeling attack resistance[C]//IEEE 23rd International Symposium on On-line Testing and Robust System Design. Thessaloniki, 2017: 74-79.
[15] HIGGINBOTHAM S. The rise of RISC[J]. IEEE spectrum, 2018, 55(8): 18.
[16] MERRITT R. Microsoft and google planning silicon-level security[EB/OL]. [2018-08-22]. https://www.eetasia.com/18082202-microsoft-and-google-planning-silicon-level-security.
[17] LEE D, KOHLBRENNER D, SHINDE S, et al. Keystone: an open framework for architecting trusted execution environments[C]//Proceedings of the Fifteenth European Conference on Computer Systems. Heraklion, 2020: 1-16.

PUF-based Hardware-assisted Software Authentication Method

LI Xueyun[1], YE Jing[2,3], HUANG Zhengfeng[1], LI Xiaowei[2,3], LI Huawei[2,3]
(1. *School of Electronic Science and Applied Physics, Hefei University of Technology, Hefei* 230601, *China*;
2. *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190, *China*;
3. *University of Chinese Academy of Sciences, Beijing* 100190, *China*)

Abstract: A hardware-assisted software authentication method based on PUF was proposed. It included software and hardware authentication protocol, RISC-V design embedded in PUF, and software obfuscation technology. The software and hardware were bound together by inserting independent PUF authentication points or embedded PUF authentication points. The software could only be executed correctly on a specific authorized device, and the reliability of PUF was solved from the hardware and software level. Experiments showed that the additional hardware overhead of the proposed method was less than 2%; and the benchmark test showed that the additional performance overhead of a single authentication point was less than 0.5%.

Key words: software protection; hardware assist; PUF; RISC-V

(责任编辑:王浩毅 方惠敏)