计 算 机 研 究 与 发 展
Journal of Computer Research and Development

DOI: 10.7544/issn1000-1239.202440088
61(6): 1458−1475, 2024

# SeChain: 基于国密算法的 RISC-V 安全启动机制设计与实现

芮志清[1,2] 梅 瑶[1,4] 陈振哲[1] 吴敬征[1,3] 凌 祥[1] 罗天悦[1] 武延军[1,3]

[1](智能软件研究中心(中国科学院软件研究所) 北京 100190)

[2](中国科学院大学 北京 100190)

[3](计算机科学国家重点实验室(中国科学院软件研究所) 北京 100190)

[4](北京航空航天大学电子信息工程学院 北京 100191)

(zhiqing@iscas.ac.cn)

# SeChain: Design and Implementation of RISC-V Secure Boot Mechanism Based on Domestic Cryptographic Algorithms

Rui Zhiqing[1,2], Mei Yao[1,4], Chen Zhenzhe[1], Wu Jingzheng[1,3], Ling Xiang[1], Luo Tianyue[1], and Wu Yanjun[1,3]

[1](*Intelligent Software Research Center* (*Institute of Software*, *Chinese Academy of Sciences*), *Beijing* 100190)

[2](*University of Chinese Academy of Sciences*, *Beijing* 100190)

[3](*State Key Laboratory of Computer Science* (*Institute of Software*, *Chinese Academy of Sciences*), *Beijing* 100190)

[4](*School of Electronic Information Engineering*, *Beihang University*, *Beijing* 100190)

**Abstract** The open-source RISC-V instruction set architecture (ISA) provides a significant opportunity for China to establish an independent and controllable IoT ecosystem. However, IoT devices often lack hardware protections and are vulnerable to physical-level firmware tampering attacks. Therefore, ensuring firmware integrity to improve device security is crucial. Previous researches have focused on secure boot techniques, but several challenges remain: 1) The traditional software-level trusted root can be physically tampered, making it difficult to ensure the reliability of the trusted root; 2) Mainstream hardware-level secure boot technology is controlled by international chip manufacturers, the specific implementation method is not publicly available, and does not support domestic cryptographic algorithms, making it impossible to ensure security independence and controllability for China; 3) Existing research on secure boot mechanisms based on RISC-V CPUs can verify the credibility of executable code but lacks a mechanism for verifying the integrity of upper-level firmware. To solve these challenges, we design and implement SeChain, a RISC-V secure boot mechanism based on the Chinese domestic cryptographic algorithm SM9. Specifically: 1) A signature calculation unit (SCU) is added inside the RISC-V SoC to generate key pairs and signatures; 2) A key verification unit (KVU) is added to perform on-chip execution of verification algorithms and firmware integrity verification; 3) A multi-level secure boot process based on verification boot is designed and implemented, starting from an immutable hardware trust root, completing the integrity verification of the next-stage boot program step by step, and achieving secure boot. Based on this design, SeChain realizes the security and trustworthiness of the trust root in the startup phase and constructs a trusted secure boot chain based on the domestic cryptographic algorithm SM9, providing reliable protection for device secure boot and trusted execution. To verify the effectiveness, efficiency, and security against trust root attacks of SeChain, we complete SeChain simulation verification experiments based on VexRiscv

**收稿日期**: 2024-02-02; **修回日期**: 2024-03-13

**基金项目**: 中国科学院战略性先导科技专项资助(XDA0320401); 本论文得到了源图重大基础设施的支持

This work was supported by the Strategic Priority Research Program of the Chinese Academy of Sciences (XDA0320401) and the YuanTu Large Research Infrastructure.

**通信作者**: 吴敬征(jingzheng08@iscas.ac.cn)

CPU on FPGA hardware platform. The experimental results show that SeChain can resist various firmware tampering attacks, as well as resist trust root attacks, which has an average additional time overhead of no more than 6.47 seconds. SeChain is suitable for resource constrained IoT devices, while meeting secure and trusted boot requirements, it can provide strong protection for domestic RISC-V ecosystem security independence and controllability.

**Key words** trusted computing; IoT security; secure boot; SM9 cryptographic algorithm; RISC-V

**摘 要** 开源 RISC-V 指令集为我国建立自主可控物联网生态提供了重大机遇. 然而, 物联网设备通常缺乏硬件加固措施, 容易遭受物理级的固件篡改攻击, 因此保障固件完整性以提高设备安全性至关重要. 为此, 已有基于安全启动技术的初步探索, 但仍存在 3 个问题: 1) 传统软件信任根难以保证物理级可靠性; 2) 主流硬件级安全启动技术被国际芯片厂商掌握, 技术未公开且不支持国密算法, 无法保证安全自主可控; 3) 已有基于 RISC-V CPU 的安全启动研究缺乏对上层固件的校验机制. 为解决上述 3 个问题, 首次设计并实现基于国密 SM9 算法的 RISC-V 安全启动机制——SeChain. 具体而言: 1) 在 RISC-V SoC 内部增加了签名计算单元 (signature calculation unit, SCU), 实现密钥对生成与签名; 2) 增加了密钥验证单元 (key verification unit, KVU), 实现验证算法的片内执行及固件完整性验证; 3) 设计实现基于验证引导的多级安全启动机制, 从不可篡改的硬件信任根出发, 逐级完成引导程序的完整性校验. 基于上述设计, SeChain 实现了信任根的不可篡改和安全可信, 构造了一个可信的安全启动链, 基于国密 SM9 算法为设备的安全启动和可信执行提供可靠保障. 为了验证 SeChain 的有效性、高效性和可靠性, 基于 VexRiscv CPU 在 FPGA 硬件平台完成了 SeChain 仿真验证实验. 实验结果表明, SeChain 能够有效抵御各类固件篡改攻击, 并能对抗信任根攻击, 且平均额外时间开销不超过 6.47 s. SeChain 适用于资源受限的 IoT 设备, 在满足安全可信启动的同时, 能为国产 RISC-V 生态的安全自主可控提供有力保障.

**关键词** 可信计算; 物联网安全; 安全启动; SM9 国密算法; RISC-V

**中图法分类号** TP391

随着物联网 (Internet of things, IoT) 的高速发展, IoT 设备的数量也在迅速增加. 据全球移动通信系统协会 (GSMA) 统计, 2030 年全球联网物联网设备数将达 53 亿, 其中中国占比超过三分之二[1]. 在物联网设备中, 主流的 ARM, MIPS 指令集架构存在知识产权壁垒、高额的授权费用和潜在的“卡脖子”风险, 给我国指令集架构的自主可控带来严峻挑战. 与此同时, RISC-V 开源、开放的特性, 为我国建立自主可控物联网生态提供了重大机遇. 特别地, RISC-V 具有开源、可定制、轻量级和低成本等特点, 这使得它在追求低功耗、小尺寸和低成本的 IoT 领域优势显著[2], 进一步推动了 RISC-V 在物联网领域的应用推广. 例如, 在处理器层面, 阿里巴巴公司旗下的平头哥推出了基于 RISC-V 的玄铁系列处理器, 该系列处理器具有低功耗、高性能的特点, 广泛应用于智能家居、工业自动化等物联网场景[3]. 在操作系统层面, “傲来” RISC-V 原生操作系统面向人工智能物联网场景, 提供了 RISC-V 指令集软件生态的全栈优化[4].

尽管 RISC-V 的开源开放给 IoT 生态的繁荣提供了重大机遇, 但是 IoT 设备信息安全风险仍较为突出. 开源网络应用程序安全项目 OWASP 指出, 缺乏硬件加固措施是 IoT 领域十大威胁之一[5]. 物联网设备的硬件安全防护通常较弱, 且部署在无人值守的非可信物理环境中, 潜在攻击者能够从硬件级别修改设备运行逻辑、获取额外权限和敏感信息, 从而进行本地控制或远程攻击. 在 IoT 设备启动时, 通过硬件级别对上层软件进行篡改攻击是一种常见的攻击方式. 例如, 攻击者可在启动时篡改操作系统内核, 以达到攻击目的. 攻击者还可以对底层固件进行攻击, 这种攻击难以检测, 甚至可以绕过防病毒软件的安全防护.

为了抵御硬件篡改攻击, 为上层应用提供可信计算环境, 大量研究人员对可信物联网展开研究, 提出物联网安全启动方案. 例如, 杨宏等人[6] 对可信物联网的相关标准进行了研究, 提出了在计算平台 (如物联网终端) 中建立安全信任根的方法, 进一步通过密码学工具从信任根开始逐层验证硬件平台、操作系统、应用软件的安全性, 从而实现信任的递进扩展, 形成安全可信的计算环境. 安全启动是可信计算的重要组成部分, 用于帮助确保设备仅使用受原始设备制造商 (original entrusted manufacture, OEM) 信任的软件进行启动[7-8]. 当设备启动时将检查每个启动软

件片段的签名，包括设备固件和操作系统，其组成组件包括不易篡改的安全信任根和校验密码算法等. 设备安全启动按照环节可分为硬件启动阶段、固件引导启动阶段、操作系统启动阶段和应用软件启动阶段. 本文主要关注硬件启动阶段，即确保下一阶段固件的完整性和可信性，并将可信启动链传递至固件引导启动阶段.

尽管上述可信物联网安全启动方案在一定程度上实现安全可信的效果，但是目前针对硬件安全启动阶段的研究在保证固件完整性上仍有不足. 微软公司的安全启动(Secure Boot)机制是一种成熟的安全启动技术，其利用板载可信平台模块(trusted platform module, TPM)芯片实现硬件安全可信根，能够在固件引导启动和操作系统启动阶段实现对基本输入输出系统(basic input/output system, BIOS)固件的自我校验、可扩展固件接口(extensible firmware interface, EFI)驱动固件校验和操作系统校验，从而实现安全启动，阻止攻击者对引导程序、系统驱动和操作系统映像进行篡改[9-10]. 但 Secure Boot 安全校验模块位于 BIOS 固件中，而 BIOS 固件通常位于主板的可读写闪存中，这使得攻击者能够通过修改 BIOS 固件的方式，绕过 BIOS 固件完整性的自我校验机制. 英特尔(Intel)公司的 Boot Guard 技术提出了基于硬件的安全启动机制，将安全信任根和校验逻辑硬编码在 CPU 内部[11]. Boot Guard 提供了基于硬件的安全信任根，负责在 BIOS 执行之前验证 BIOS 映像，以保障 BIOS 的安全可信. 由于只有英特尔公司或被授权的制造商拥有用于签署固件包的密钥，因此 Boot Guard 的可定制性较低，无法在这些处理器上使用 coreboot、LinuxBoot 或其他等效固件，这与物联网场景下对于可定制性的需求相冲突. 在 RISC-V 领域，Jawad 等人[12] 提出了一种针对 RISC-V 片上系统(system on chip, SoC)的轻量级安全启动框架，通过在 SoC 中内嵌公钥和中心式的代码认证单元(code authentication unit, CAU)，基于椭圆曲线数字签名算法(elliptic curve digital signature algorithm, ECDSA)和第三代安全散列算法(SHA-3)实现对代码的可信认证. 但该框架要求任意程序代码均需包含签名才能被成功执行，无法保持对已有程序代码的兼容性. 另外，目前尚无支持国密算法的 RISC-V 安全启动技术，不利于国产安全可控生态的发展.

为了解决上述问题，本文提出了一种基于国密 SM9 算法的 RISC-V 安全启动机制——SeChain. 具体而言：1)SeChain 在 RISC-V SoC 内部增加了签名计算单元(signature calculation unit, SCU)，实现 SM9 算法的密钥对生成与签名. 在设备激活时，SCU 根据用户标识 ID 生成主密钥对和用户密钥对. 随后，SCU 使用用户私钥对各级引导程序进行签名，并将签名存入上一级引导程序帧头. 2)增加了密钥验证单元(key verification unit, KVU)，实现 SM9 验证算法的片内执行，通过嵌入的用户公钥与签名对固件进行完整性验证；3)设计实现基于验证引导的多级安全启动流程，将硬件信任根固化在不可篡改的片内 ROM 模块中，通过在零级引导加载程序(zero stage boot loader, ZSBL)、一级引导加载程序(first stage boot loader, FSBL)、二级引导加载程序(second stage boot loader, SSBL)、系统内核中逐级完成对下一阶段程序的完整性校验，实现安全启动. 上述设计通过充分固化硬件信任根的方式解决了信任根易被篡改的问题，保障了信任根的可信性. 利用 RISC-V 开源开放的特点，基于国密 SM9 算法重新设计 SoC，实现了国产自主可控. 基于上述设计，SeChain 实现了启动阶段的信任根安全可信，构造了一个可信的安全启动链，并基于国密 SM9 算法为设备的安全启动和可信执行提供可靠保障. 为了验证评估 SeChain 的实际效果，本文实现了 SeChain 原型系统，并完成了有效性、高效性和信任根攻击实验. SeChain 原型系统基于 VexRiscv 32 bit RISC-V CPU 与 Litex SoC 平台搭建[13]，并在 Artix-7 100T 现场可编程门阵列(field programmable gate array, FPGA)硬件平台进行仿真，搭载了 OpenSBI，Linux 软件负载. 在有效性验证实验中，本文对比了 SeChain 与原 VexRiscv CPU 的在正常启动和篡改攻击下的启动流程. 结果表明，相对于原 VexRiscv CPU，SeChain 在受到攻击后能阻断异常启动流程，验证了 SeChain 的有效性. 高效性验证实验分析了 SeChain 的硬件资源消耗和额外时间开销，验证 SeChain 在加密算法所需硬件资源消耗较高的情况下仍有足够的硬件余量，且平均额外时间开销不超过 6.47 s. 信任根攻击实验验证了 SeChain 能对抗密钥窃取与签名伪造攻击以及 SeChain 的可靠性.

本文的主要贡献有 3 点：

1)提出首个基于 SM9 国密算法的 RISC-V 安全启动机制 SeChain，将可信根嵌入 SoC 内部以实现不可篡改性，并设计了 SCU 和 KVU 模块，实现了基于验证引导的多级安全启动流程；

2)基于 VexRiscv CPU 搭建 RISC-V 仿真平台，并在 FPGA 硬件平台进行仿真，在应用 SeChain 的 RISC-V CPU 上高效可靠地实现了安全启动，并将相关源

代码发布至开源社区[14]；

3）对包含安全启动机制的 SeChain 原型系统进行有效性、高效性和信任根攻击实验，实验结果表明，该系统能够有效抵御各类固件篡改攻击，平均额外时间开销不超过 6.47 s，可靠地对抗信任根攻击，验证了其安全启动流程的有效性、高效性和可靠性.

## 1 背景知识与相关工作

### 1.1 密码学基础知识

本节首先介绍相关的密码学基础知识，包括哈希算法、非对称密码算法、数字签名和国密 SM9 算法.

1）哈希算法. 哈希算法是一类把任意长度的消息变换为固定长度的比特串的算法，通常用于确保数据的完整性，变换的结果被称为摘要或哈希值[15]. 哈希算法也被称为散列算法或杂凑算法，是密码学领域中的一类重要算法. 常见的哈希算法包括 MD5[16]、SHA-1[17]、SHA-3[18] 和国密 SM3 算法[19].

哈希算法可被表示为 $H:\{0,1\}^{*}\rightarrow\{0,1\}^{n}$，其中 $\{0,1\}^{*}$ 表示输入消息可以是任意长度的比特串，$\{0,1\}^{n}$ 表示输出是一个固定长度的比特串. 哈希算法应具备抗碰撞攻击和抗原像攻击的基本安全性原则. 抗碰撞攻击指对于 2 个不同的输入消息 $M$ 和 $M'$，从计算上找到 $H(M)=H(M')$ 是不可行的；抗原像攻击指给定哈希值 $h$ 从计算上找到消息 $M$，使得 $h=H(M)$ 是不可行的.

2005 年，我国密码学家 Wang 等人[20-21] 提出了对重要哈希算法 MD5 和 SHA-1 的碰撞攻击方法，引发了国际密码社区对传统哈希函数标准安全性的质疑. 美国国家标准技术研究所（NIST）于 2007 年开始了新一代哈希算法标准的征集. 最终于 2012 年，KECCAK 算法凭借其安全性、硬件效率、灵活性和整体表现被评为 SHA-3 标准[18]. 随着 SHA-3 征集活动的进行，我国也在设计自主可控的哈希算法标准. 2012 年，SM3 算法被定为我国密码行业标准. 2016 年，国家标准化委员会将 SM3 算法作为国家标准[19].

2）非对称密码算法. 非对称密码算法是指使用一对公钥和私钥进行加解密的算法，也被称为公钥加密算法，是密码学中的另一类重要算法[22]. 在非对称密码算法中，每个参与方均有一对公钥和私钥，其中公钥 $pk$ 是公开的，用于加密信息，而私钥 $sk$ 需秘密保存，用于解密信息. 当发送方给接收方发送秘密信息 $m$ 时，只需要利用接收方的公钥 $pk$ 加密消息，并将密文 $c$ 发送给接收方. 然后接收方便可以利用私钥 $sk$ 将密文 $c$ 解密为原始消息 $m$. 如果以 $E$ 表示加密函数，$D$ 表示解密函数，则上述加密过程可表述为 $c=E(pk,m)$，解密过程可表述为 $m=D(sk,c)$. 相对于对称密码算法，非对称密码算法不需要将密钥通过安全信道传输给接收方，避免了密钥管理的问题.

非对称密码的思想最早于 1976 年由 Diffie 等人[23] 提出. 此后，国内外的研究人员[24] 设计了许多优秀的非对称密码体制，如 RSA 体制、Rabin 体制和椭圆密码体制等. 非对称密码由于具有密钥管理方便的优势且安全性强，已被广泛应用于加密通信和数字签名领域. 但非对称密钥生成机制仍存在安全问题. 2007 年，斯诺登披露的文件中显示，NIST 将带有隐秘缺陷的双椭圆确定性随机数生成算法（Dual_EC_DRBG）定为 NIST 标准，并将其出售给 RSA 公司用于密钥生成，使得攻击者能够破解加密后的数据[25-26].

3）数字签名. 数字签名技术是一种确保信息传输完整性和验证消息发送者身份的密码学手段. 数字签名技术包括签名生成阶段和签名验证阶段. 在实践中，签名生成阶段通常基于哈希算法 $H$ 生成消息 $m$ 的摘要 $H(m)$，并利用发送者的私钥 $sk$ 对该摘要使用非对称密码算法进行加密. 在签名验证阶段，接收方使用发送者的公钥 $pk$ 解密摘要，然后将其与消息 $m$ 的摘要 $H(m)$ 比较，以验证消息的完整性和发送者的身份. 签名生成阶段可表示为 $\sigma=S(sk,H(m))$，签名验证阶段可表示为 $V(pk,H(m),\sigma)=\{\text{true},\text{false}\}$. 其中，$S$ 为签名生成函数，$V$ 为签名验证函数，签名为 $\sigma$.

数字签名技术的目的在于提供一种安全可靠的方式来验证消息的真实性和完整性，防止篡改和伪造. 数字签名技术依赖于非对称密码算法和哈希函数的结合，确保了数字签名的非伪造性、不可否认性和可验证性.

4）国密 SM9 算法. SM9 标识密码算法是我国自主研发的基于标识密码的非对称密码算法，其对于我国密码学领域有非常重要的意义[27]. 数字签名是 SM9 算法的重要功能之一. 与其他密码算法相比，SM9 算法不需要数字证书、证书库或密钥库，因此密钥管理环节能够得到简化[23,28]. 如现有的 IoT 硬件信任根设计方案中，基于公开密钥基础设施（public key infrastructure, PKI）的密码体系往往需要设置额外的证书模块，启动中需要与证书模块进行交互[8]. 而基于身份标识的密码（identity-based cryptography, IBC）体系的 SM9 算法在实现中无需考虑证书模块，减少了 IoT 设备管理中的复杂性与潜在的安全隐患. 此外，SM9 密码算法基于 256 b 的 Barreto-Naehrig（BN）椭圆

曲线[29-30]，加密强度相当于3 072 b密钥的RSA加密算法，在现有计算能力下无法被破解，安全性高[23]，且由于国密SM9算法密钥短，在加解密和验签效率方面性能更好，适用于身份认证、数据加密和通道加密等IoT场景，在泛用性等方面有良好表现[31]. 由此可见，SM9算法不仅能实现自主可控，而且在成本和易用性方面更具优势.

SM9是基于256 b的BN椭圆曲线的密码算法，以下将简单介绍其签名验签的数学原理[29-30].

首先定义3个阶均为素数$N$的循环群$(G_T,\times)(G_1,+)(G_2,+)$，线性对$e$是满足一定条件的$G_1\times G_2\to G_T$的映射[30]. 定义通信双方用户$A$和$B$的$ID$分别为$ID_{\rm A}$和$ID_{\rm B}$，签名方用户$A$使用SM9算法产生的2对密钥分别为主公钥$P_{\rm Pub}$和主私钥$s$；用户公钥$ID_{\rm A}$和用户私钥(签名私钥)$ds_{\rm A}$.

SM9算法的签名、验签主要有2个过程：

1) 签名主要过程

设待签名消息比特串为$M$，根据主公钥$P_{\rm Pub}$计算得出$G_T$中的元素$\omega$，令$h=H_2(M|\omega,N)$，根据随机数$r\in[1,N-1]$和$h$，计算整数$l=(r-h)\bmod N$，再计算群$G_1$中元素得到用户$A$对$M$的签名$(h,S)$.

2)验签主要过程

设待签名消息比特串为$M$，用户$B$收到消息$M'$及数字签名$(h',S')$，首先检验必要条件$h'\in[1,N-1]$和$S'\in G_1$，其中一项不成立则验签失败；若都成立，则根据主公钥$P_{\rm Pub}$及$ID_A$计算出整数$h_2$，若$h_2=h'$，则验签通过.

在SM9算法的计算过程中，需要计算双线性对，常用的双线性对算法是米勒算法：

设$\mathrm{F_q}k$上椭圆曲线$E\left(\mathrm{F_q}k\right)$的方程为$y^2=x^3+ax+b$，定义过椭圆曲线$E\left(\mathrm{F_q}k\right)$上点$U$和$V$的直线为$g_{U,V}$：$E\left(\mathrm{F_q}k\right)\to\mathrm{F_q}k$，若过$U$，$V$两点的直线方程为$\lambda x+\delta y+\tau=0$，则令函数$g_{U,V}(Q)=\lambda x_Q+\delta y_Q+\tau$，其中$Q=\left(x_Q,y_Q\right)$. 当$U$=$V$时，$g_{U,V}$定义为过点$U$的切线；若$U$和$V$中有一个点为无穷远点$O$，$g_{U,V}$就是过另一个点且垂直于$x$轴的直线. 一般用$g_U$作为$g_{U,U}$的简写. 记$U=(x_U,y_U)$，$V=(x_V,y_V)$，$Q=\left(x_Q,y_Q\right)$，$\lambda_1=\left(3x_V^2+a\right)/(2y_V)$，$\lambda_2=(y_U-y_V)/(x_U-x_V)$. 则有4种性质：

① $g_{U,V}(O)=g_{U,O}(Q)=g_{O,V}(Q)=1$；

② $g_{V,V}(Q)=\lambda_1\left(x_Q-x_V\right)-y_Q+y_V,Q\neq O$；

③ $g_{U,V}(Q)=\lambda_2\left(x_Q-x_V\right)-y_Q+y_V,Q\neq O,U\neq\pm V$；

④ $g_{V,V}(Q)=x_Q-x_V,Q\neq O$.

因此，双线性对的计算过程为：

输入：曲线$E$上2点$P$和$Q$，整数$c$；

输出：$f_{P,c}(Q)$，一般称$f_{P,c}(Q)$为Miller函数.

① 设$c$的二进制表示为$c_j\cdots c_1c_0$,最高位$c_j$为1；

② 置$f=1,V=P$；

③ 对$i$从$j-1$降至0，执行：计算$f=f^2\cdot g_{V,V}(Q)/g_{2V}(Q),V=[2]V$，若$c_i=1$，令$f=f\cdot g_{V,P}(Q)/g_{V+P}(Q),V=V+P$；

④ 输出$f$.

目前，传统SM9国密算法一大应用难题就是其双线性对的计算复杂度较高，但近年有学者提出了对于SM9算法的优化与相应的硬件实现，大大提高了其运行速度，如杨国强等人[32]在2020年提出的一种基于FPGA的SM9快速实现方法可将SM9算法的性能提升2.3倍. 随着研究的推进，智能密码钥匙、标识密码机等采用国密SM9算法的物联网设备已经面世，SM9算法正在物联网可信计算领域得到发展应用.

### 1.2 可信计算

目前，可信计算尚无明确定义，从不同角度出发，不同专家和组织给出了以下定义. IEEE可信计算与容错技术委员会(IEEE Computer Society Technical Committee on Dependable Computing Fault Tolerance, TCFT)认为可信是指计算机系统提供的服务可以被论证是可信赖的(dependable)，主要针对系统内部的不稳定性，通过故障诊断、冗余备份、容错等技术，确保系统预期行为不被局部故障影响[33]. 其中，可信赖主要指系统的可靠性和可用性. 可信计算组织(Trusted Computing Group, TCG)将可信计算定义为：如果一个实体的行为总是以预期的方式朝着预定目标进行，则该实体是可信计算的(trusted computing)[34]. 基于此，TCG提出了一种链式的信任度量模型，简称信任链. 信任链构建了一条从BIOS启动区块到BIOS、操作系统加载器、操作系统，最终到应用的串型链路，其中BIOS启动区块作为可信测量根. 信任链采用迭代计算哈希值的方式，实现从信任根开始到硬件平台、操作系统再到应用，将信任扩散到整个系统，保障整个系统的可信性. 沈昌祥等人[35]对上述2种定义进行综合和扩展，提出可信计算是能够提供系统的可靠性、可用性、信息和行为安全性的计算机系统. 其中，系统的安全性和可靠性是可信计算最主要的2个方面. 微软公司将可信计算定义为计算机系统需提供安全、可用、可靠的服务，还能够提供该系统能满足这些属性的证据[36]. 虽然不同领域研究人员对于“可信计算”(trustworthy computer)有着不同的理解和定义，但它们的基础目标是共通的，即系统能够

确保功能和属性以预期的方式得以实现和满足.

### 1.3 安全启动

安全启动是一种确保设备仅使用受原始设备制造商信任的软件进行启动的机制，是实现可信计算的重要手段[8,37]. 在 SoC 平台中，安全启动的关键因素包括安全信任根、可信执行环境（trusted execution environment, TEE）和运行系统. 如图 1 所示，安全启动信任链是从可信零级引导加载程序运行到可信执行环境的所有组件生成的信任链，该信任链是从不易篡改的信任根上建立的. 安全信任根是系统信任链的基础，由一组具有不同安全功能的模块组成，如监控器、加密引擎、身份验证器等，该模块可执行安全算法和函数保护系统中的组件、数据和资源[38]. 基于可编程硬件的信任根设计可以提供最高效的安全系统. 可信应用执行环境是一种隔离的运行环境，可以保护敏感数据和代码不受外部干扰和攻击. 运行系统是指在 TEE 中执行的软件，如操作系统、应用程序等，它们需要通过安全启动的验证才能加载和运行.

目前，安全启动的实现方法主要有 2 种：测量引导和验证引导. 测量引导是指在每个启动阶段都会对后续阶段的软件进行完整性测量，并将测量结果存储在 TPM 中，以便后续进行验证或远程证明. 验证引导是指在每个启动阶段都会对后续阶段的软件进行完整性验证，并只有在验证通过后才会执行. 测量引导和验证引导都需要建立一个信任链，即从一个固定的信任根开始，逐步验证每个组件的签名或哈希值，直到最终的应用程序[8,38-39]. 相对于测量引导，验证引导模式无需如 TPM 等的外部安全元件，更贴合 IoT 应用场景对设备复杂度、成本的严格要求.

目前，许多商业化的平台都采用了安全启动的机制，如英特尔、微软、谷歌等公司. 2001 年，由英特尔、IBM、惠普、微软、康柏等巨头企业发起的 TCPA 组织提出了第一个 TPM 安全规范[10]. TPM 安全芯片提供了硬件级别的密钥管理，是应用最成熟的安全启动防护机制. 自 2005 年前后以来，TPM 在个人电脑与服务器中得到了大规模商用，构建形成了 BitLocker, Trusted Sensors 等常见安全方案. 英特尔公司使用 TPM 来实现测量引导，并提供了一套基于统一可扩展固件接口（unified extensible firmware interface, UEFI）规范的安全启动解决方案. 然而，由于空间、成本和功率的限制，TPM 并不适用于小型嵌入式设备或 IoT 设备中. TEE 是基于硬件修改以划分不同保护域的安全启动方案，如英特尔公司的软件保护扩展（software guard extensions, SGX）通过特殊指令对一段受保护空间实行严格的访问控制和加密操作；ARM 公司提出的 TrustZone[40] 技术通过修改原有硬件架构实现安全隔离；微软公司的 Raj 等人[41] 面向小型嵌入式设备，改进了基于 ARM Trustzone 技术的固件 TPM（FTPM）来实现验证引导. 谷歌公司使用基于 UEFI 规范的验证引导来保护其 Chrome 操作系统和 Android 操作系统，并提供了验证访问（verified access）等远程安全验证功能[9].

然而，现有的安全启动方案也存在一些缺陷和挑战. 基于 TPM 的实现不适用于小型嵌入式设备及 IoT 设备，同时由于其复杂性和封闭性，容易产生未知的漏洞和风险. 安全公司 Quarkslab 的研究人员于 2022 年 11 月发现了 2 个影响 TPM 芯片的漏洞 CVE-2023-1017 和 CVE-2023-1018，这些漏洞可以利用越界读写漏洞造成加密密钥泄露或内存破坏[10]. 基于 TEE 的安全启动实现未形成统一规范，导致不同厂商设计方案的碎片化，产生了额外的设计成本、复杂度和授权问题. 其他基于复杂的 UEFI 规范的安全启动实现也存在一些不足之处，如缺乏统一的接口标准、过于庞大和复杂、缺乏开源实现等[11]. 同时，在密码学算法的选用方面，现有的安全启动实现缺乏对于国密算法的兼容与支持，如 UEFI 中，哈希算法仅支持 SHA-2-256, SHA-2-384, SHA-2-512 等，数字签名校验算法仅支持 EMSA-PKCS1-v1_5. 同时仅支持 RFC 标准 PKCS7 Signed Data 格式，不兼容国密算法定义的签名格式. 即使在具有兼容国密算法的安全机制中，基于 PKI 体系由于其强依赖于外部营运的 UEFI CA，也将造成验证的不可靠性. 例如白名单中的所有证书处于相同等级，未支持国密证书强制验证或优先验证. 当国密证书和其他 CA 证书共存时，

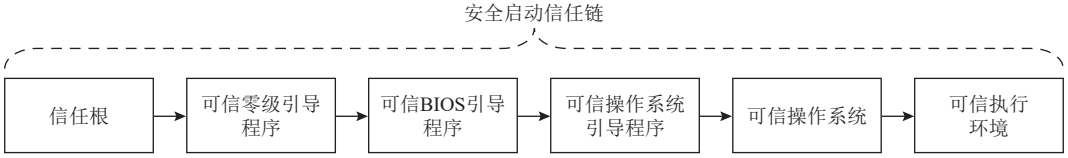



Fig. 1 The illustration of trust chain during secure boot

图 1 安全启动信任链示意图

UEFI 安全启动可能无法构建一条完整的、基于国密验证的安全链条[9,10,42].

### 1.4 新型体系架构下的安全启动

为了解决安全启动问题，一些学者和研究机构开始探索基于 RISC-V 架构的安全启动方案. RISC-V 是一种开源的指令集架构(ISA)，具有简洁、灵活、可扩展等优点，可以支持多种硬件实现和软件定制[43]. 基于 RISC-V 架构的安全启动方案可以分为软件实现和硬件实现 2 种类型. 软件实现主要依赖于 RISC-V ISA 中提供的特权级别和异常处理机制来实现安全启动过程中的软件验证和隔离. 例如，Shakti-T[44] 利用基址寄存器和界限寄存器来确保指针只能访问有效的内存区域；SMARTS[45] 所实现的集成在 RISC-V SoC 中的内存保护单元(MPU)提供了外部内存(DRAM)的完整性和机密性保护. 基于硬件划分可信域的 Keystone[46] 启动了一个开源项目，旨在构建基于 RISC-V 体系结构的 TEE；Sanctum[47] 引入了基于软件的安全启动和远程证明过程；Vikiema[48] 针对边缘计算设备的 RISC-V 双核锁步实现机制，通过额外的时钟周期提供瞬态错误的检测和纠正，以实现后量子时代的安全启动；张雨昕等人[49] 基于协处理器设计了针对内存缓冲溢出攻击的防御机制，保障了执行环境的可信性. 然而，文献 [45-49] 方案仍然存在可篡改性、性能损失、兼容性问题等挑战.

硬件实现主要依赖于可编程硬件技术(如 FPGA)来实现安全启动过程中的硬件验证和保护. 例如，Care[39] 提出了一个位于第一阶段引导固件和第二阶段引导闪存之间的 CARE 模块，该模块在安全启动期间监控代码完整性和真实性，并使用来自 ROM 中不可篡改数据恢复损坏闪存区域.

## 2 问题定义

### 2.1 研究挑战

基于上述分析，目前 RISC-V 领域安全启动研究主要存在 3 个挑战：

1)信任根的完全可靠性难以保证. 保证信任根的可信是安全启动问题的基础. 英特尔公司提出的 Boot Guard 为其提供硬件强制引导策略控制，以授权允许在该平台上运行哪些 BIOS 代码. Boot Guard 提供了基于硬件的信任根，负责在 BIOS 执行之前验证 BIOS 映像，以验证 BIOS 的安全性，保证 BIOS 安全可信. 而现有的 RISC-V 系统中缺乏从硬件上保证信任根可信的设计.

2)缺乏自主设计、自主产权的安全启动策略. Boot Guard 的知识产权和核心技术被国外企业掌握，对于用户来说，无法使用 coreboot, LinuxBoot 或其他等效开源固件，限制了用户对于固件的自主操作. 在国家层面，尽管英特尔等公司宣称其安全启动数据对公司保密，但被各企业掌控的密钥与安全启动技术核心，以及非自主产权的通用密码算法使我国的自主信息安全存在巨大风险.

3)RISC-V 生态缺乏验证上层固件完整性校验的安全启动机制. 已有的针对 RISC-V 安全启动的初步探索可实现对执行代码可信性的校验，但对于固件完整性校验尚无完整可用的解决方案. 另外，现有的 RISC-V 安全启动方案存在依赖公共 CA 服务提供签名证书、加密算法选用受限等缺陷.

### 2.2 威胁模型

针对 2.1 节中 3 个挑战，本文重新定义了安全启动过程中的威胁模型，探讨潜在攻击者、攻击手段以及保护目标，确保提出的安全机制能够有效抵御这些威胁，保障嵌入式设备在执行时固件文件的完整性与可信性.

本文研究聚焦于 IoT 设备，这些设备被广泛应用于智能家居、工业控制、医疗监护等领域，对安全性的要求极高. 目标用户群体主要是设备的制造商，负责设备的设计、生产和部署. 设备制造商需要保证设备的可信性，保护设备免受未经授权的访问和篡改.

本文研究考虑了 2 种类型的攻击者：

1)设备芯片上游供应商. 此类攻击者可在通过供应链的供应关系，在芯片设计、生产制造过程中植入有缺陷或恶意的后门机制. 通过这些手段，攻击者可能实现对设备启动过程的控制，进而达成隐私窃取、远程控制等攻击操作. 此类后门已有先例，2021 年，镁光公司被披露其生产的企业级固态硬盘产品包含漏洞，攻击者可发出特殊控制命令对硬盘进行攻击[50]. 而对于 x86、ARM 指令集架构的 CPU 芯片，其内部实现通常极为复杂且严格保密，导致安全人员对其进行后门安全审计的难度大大增加.

2)供应链下游及终端用户. 供应链下游及终端用户可直接接触到设备成品，便可通过物理手段攻击设备. 例如，攻击者可绕过厂商提供的使用接口，通过电路连接外部存储芯片(如 flash 芯片)，直接访问外部存储芯片，进而读取敏感信息、篡改设备固件. 另外，攻击者还可通过其他硬件攻击手段(如故障注入)向固件中写入恶意后门，破坏设备安全.

鉴于上述威胁，本文研究的目标是确保设备在

执行时所使用的固件文件是可信的，未经任何形式的外部篡改.

## 3 SeChain 设计与实现

针对上述挑战和威胁模型，本文提出了一种基于国密算法的 RISC-V 安全启动机制——SeChain.

如图 2 所示，SeChain 设计了一种全新的 SoC 方案，基于国密算法在 SoC 内部实现安全校验功能，并基于验证引导模式实现对外部固件的安全校验. 具体包括 3 方面：

1）设计了 SCU 模块，实现 SM9 算法密钥对的生成与签名；

2）增加 KVU 模块，实现国密 SM9 验证算法的片内执行，负责利用公钥与签名验证固件完整性；

3）设计实现基于验证引导的多级安全启动机制，从不可篡改的硬件信任根出发，逐级完成对下阶段引导程序的完整性校验.

传统的安全启动方案 Secure Boot 和 Boot Guard 主要使用 RSA 等国际通用密码算法，无法满足国产“安全、自主、可控”的核心需求. 因此本文选用了国家密码管理局 GM/T0044-2016 标准的 SM9 算法. SM9 算法具备无需数字证书、加解密性能好的特点，不仅能够满足“安全、自主、可控”需求，且在成本和易用性上更有优势.

SM9 密码算法规定的基于标识的数字签名算法中涉及到 2 类辅助函数：密码杂凑函数和随机数发生器. 为确保密码算法的自主化，本文将密码杂凑函数设定为国家密码管理局 GB/T 32 905—2016 标准规定的 SM3 杂凑算法，将随机数发生器设定为国家密码管理局 GM/T 0105—2021 标准的随机数发生器. 同时，国密 SM9 算法和 SM3 算法也已被纳入 ISO/IEC 国际标准，在国际上也具有广阔的应用前景.

本文将 SM9 所需的公钥烧录在 SoC 内中，实现验证引导模式，不涉及如 TPM 等的外部安全元件，更贴近 IoT 场景下的使用模式. 芯片设计厂商可在芯片制造时将公钥烧录在 SoC 内，并通过 SCU 生成私钥提供签名服务. 在出厂前使用签名密钥对每个固件进行签名，启动时以硬件不可篡改的公钥为信任根，逐级对下一级固件进行完整性校验，下一级只有经过校验后才能获得控制权，以确保下一级执行的程序可信. 基于上述设计，SeChain 可实现硬件启动阶段的安全可信，为下一阶段的安全启动和可信执行提供可靠保障. 下面将具体阐述 SeChain 系统设计.

### 3.1 系统整体设计

由于现代 SoC 芯片集成度极高，芯片安全设计已较为成熟，针对 SoC 芯片的篡改攻击成本高昂，几乎难以实现，因此选择将安全可信根及安全校验机制通过硬件电路编码的方式置于 SoC，能够保证安全可信根及校验机制的不可篡改，保障安全启动的可信性[38]. 因此，本文基于 SoC 设计了整体的技术结构. 如图 2 所示为 SeChain 的系统整体结构，SeChain 的主要功能模块位于 SoC 中. 整个片上系统的主要部件可以分为 7 个模块：

1）RISC-V 核. 用于执行核心的计算任务，可采用既有的 RISC-V 核.

2）系统总线（System Bus）. 负责连接处理器及片上其他功能单元.

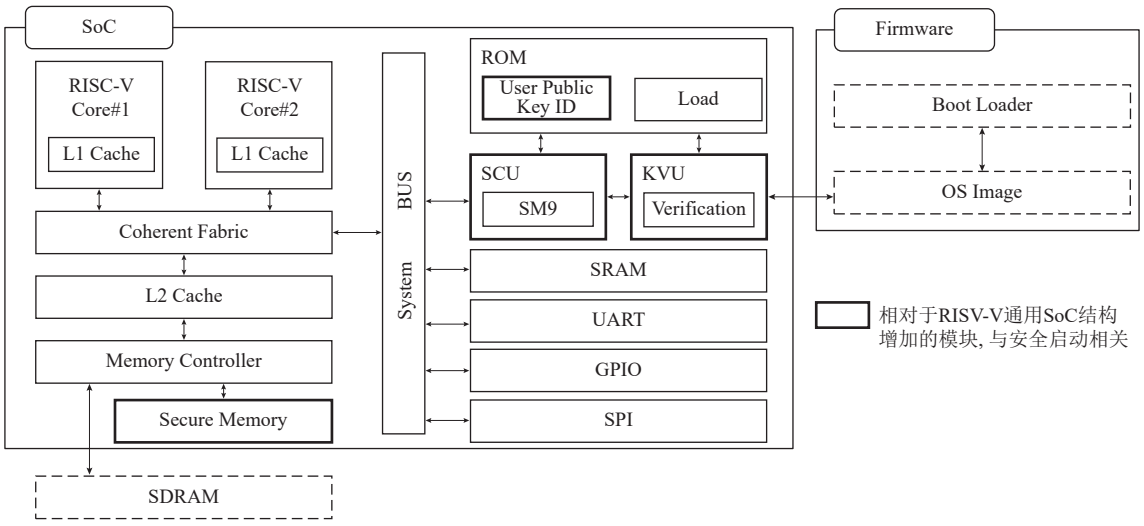



Fig. 2 The SoC structure of SeChain

图 2 SeChain 上 SoC 结构

3)存储. 包括 RISC-V 核内的一级缓存、CPU 外的二级缓存以及一块受验证的安全存储单元(Secure Memory).

4)SCU. 负责实现用户隐私和主公钥的生成以及基于私钥的签名,并实现固件签名验证逻辑.

5)KVU. 实现国密 SM9 算法的片内执行,负责完成公钥验证.

6)内部 ROM. 包含加载模块,用于存放零级引导加载程序(zero stage boot loader, ZSBI),同时设备标识 ID 内嵌在 CPU 中的 ROM 中,作为用户公钥,出厂后具有不可更改性.

7)其他片上外设. 如通用异步收发传输器(universal asynchronous receiver/transmitter, UART)、通用输入输出(general purpose input output, GPIO)等,将在系统上电的第 0 阶段由 ZSBL 进行初始化.

除 SoC 外,SeChain 系统中包括虚线框中的片外设备 SDRAM 及包含操作系统映像(OS Image)等部分的固件(Firmware)将在系统上电的第一引导加载阶段进行初始化.

### 3.2 安全启动模块设计

#### 3.2.1 签名计算单元(SCU)

SCU 在整个认证流程中起到密钥生成与签名功能作用,仅在设备激活时被调用 1 次,之后便被禁用. SCU 根据用户标识 $ID$ 生成 2 对密钥,使用用户私钥 $ds_A$ 对固件进行签名,成为上一级固件帧头的签名. 在签名与验证签名之前,都需用 SM3 杂凑算法对固件进行摘要计算,得到固件哈希值 $M$. 由于本文中 SM3 算法和 SM9 算法有较高的耦合度,为了使得模块布局更加简化,SM3 算法和 SM9 算法均在 SCU 模块中实现.

1)SCU 中的密钥生成流程为:SCU 读取 ROM 中的用户标识 $ID$ 作为用户公钥,调用 SM9 算法生成随机数 $s$ 作为主私钥,根据存储的一组预定义好的系统参数 $param$ 及随机数 $s$ 计算主公钥 $P_{\text{Pub}}$,同时根据 $ID$、$param$, $s$ 生成用户私钥 $ds_A$. 因此,共存在 2 对密钥,分别为:主密钥对 $(s, P_{\text{Pub}})$,用户密钥对 $(ds_A, ID)$. 主公钥 $P_{\text{Pub}}$ 将被存入 KVU 中,主私钥 $s$ 及用户私钥 $ds_A$ 在签名后废弃.

2)SCU 中的签名流程如图 3 所示,设待签名固件经过 SM3 杂凑函数得到的哈希值为 256 的比特串 $M$,根据主公钥 $P_{\text{Pub}}$ 计算得出 $G_T$ 中元素 $\omega$,令 $h = H_2(M|\omega, N)$,根据随机数 $r \in [1, N-1]$ 和 $h$,计算整数 $L = (r-h) \bmod N$,再计算群 $G_1$ 中元素 $S = [L]ds_A$,得到 $M$ 的签名 $(h, S)$,该签名 $(h, S)$ 被转换为字符串后,将

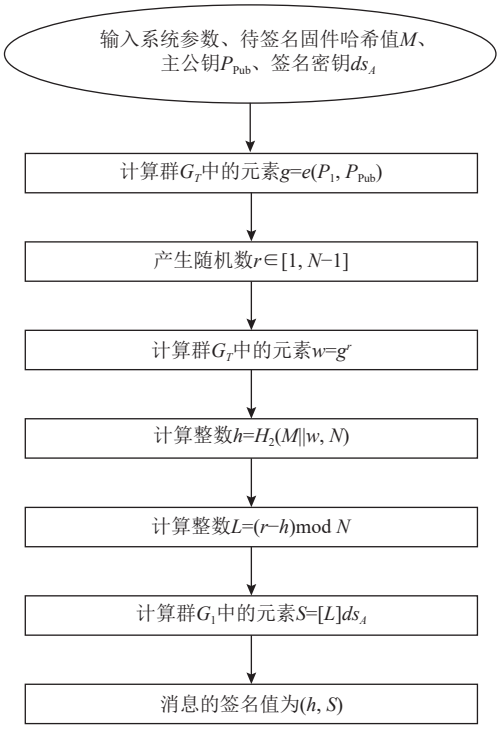



Fig. 3 The signature process of SM9 algorithm

图 3 SM9 算法签名过程

被写入上一级固件的头部,在启动验证流程中使用.

#### 3.2.2 密钥验证单元(KVU)

KVU 的目的是使用合适的加密算法和密钥验证外部输入的完整性,在 SeChain 中用以保障外部固件的安全性,其流程是使用内嵌的用户公钥进行签名验证,通过则信任外部固件,不通过则停止启动并报错.

KVU 验签主要过程如图 4 所示,待验证固件经过 SM3 函数得到哈希值为 256 的比特串 $M'$,对于哈希值 $M'$ 及数字签名 $(h', S')$,根据主公钥 $P_{\text{Pub}}$ 及 $ID$ 计算出整数 $h_2$,若 $h_2 = h'$,则验签通过,信任并执行此固件后,继续下一个信任链节点的验证.

### 3.3 安全启动流程

#### 3.3.1 签名生成流程

签名生成流程适用于 SoC 生产出厂前,供设备制造商对 SoC 进行初始化与激活,此过程包括密钥生成与签名. 激活流程如图 5 所示,仅进行密钥生成与签名操作,调用 SM9 算法生成签名的步骤为:SCU 使用设备 $ID$ 作为用户公钥,根据用户标识 $ID$ 生成 2 对密钥,使用生成的用户私钥 $ds_A$ 对固件进行签名,写入上一级固件帧头作为签名部分.

具体签名生成步骤为:

1)密钥生成. 上电后,在无安全启动的情况下完成设备裸机启动. SCU 读取 ROM 中所存储的用户标识 $ID$ 作为用户公钥,调用 SM9 算法生成主密钥对

$(s, P_{Pub})$及用户密钥对$(ds_A, ID)$. 主公钥$P_{Pub}$同时存入KVU中.

2)生成签名. SCU使用SM3杂凑函数得到待签名固件的哈希值$M$, 再根据主公钥$P_{Pub}$、用户私钥$ds_A$及系统参数$param$生成签名$(h,S)$并转换为字符串.

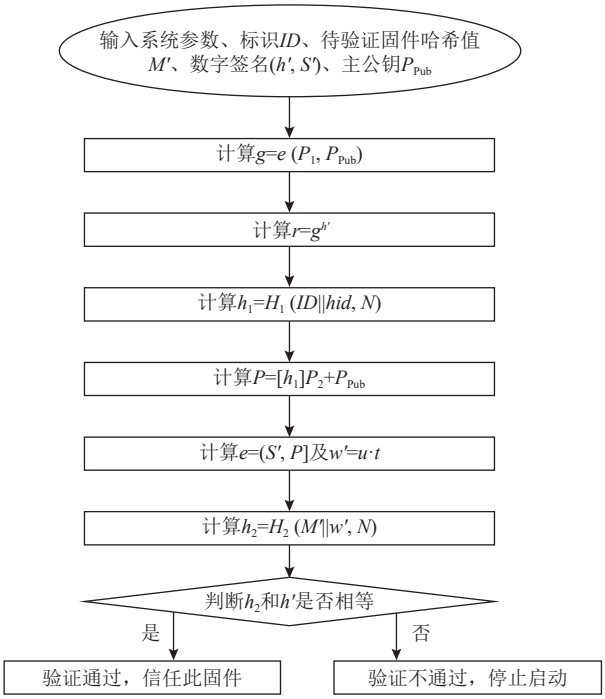



Fig. 4 The verification signature process of SM9 algorithm

图4 SM9算法验证签名过程

3)写入签名. 将$(h,S)$转换成的字符串写入上一级固件帧头签名部分.

4)SCU禁用. SCU从OS Loader开始, 逐级向前进行生成签名与写入签名步骤, 直到将第一阶段的FSBL签名写入ROM为止, 完成上述功能后将被禁用, 以实现一次性读写, 主私钥$s$及用户私钥$ds_A$也被相应废弃.

### 3.3.2 签名验证流程

签名验证流程适用于终端用户日常开机启动时所进行的自动安全校验操作. 在用户使用场景的启动流程中, 仅进行签名验证操作, 无需调用SCU, 仅调用KVU通过CPU内置公钥进行逐层信任验证, 签名验证流程如图6所示. 具体签名验证步骤为:

在上电后, 首先读取板载ROM中的ZSBL, 由ZSBL完成包括SPI等片上设备的初始化. 之后, ZSBL通过SPI与密钥验证单元KVU交互, KVU读取缓存中的FSBL程序, 使用SM3算法计算其哈希值后, 结合ROM中内嵌的用户公钥、ROM中预存的FSBL签名值与KVU中存储的主公钥对第一阶段加载程序FSBL进行验证. 若验证通过, 则将控制权移交给FSBL, FSBL进行片外设备的初始化后, 采用相同的方式验证下一级Boot Loader, 并将控制权移交. SCU与KVU存在于片上的Secure Memory中, 以保证其无法被外部程序篡改. SeChain安全启动流程采用信

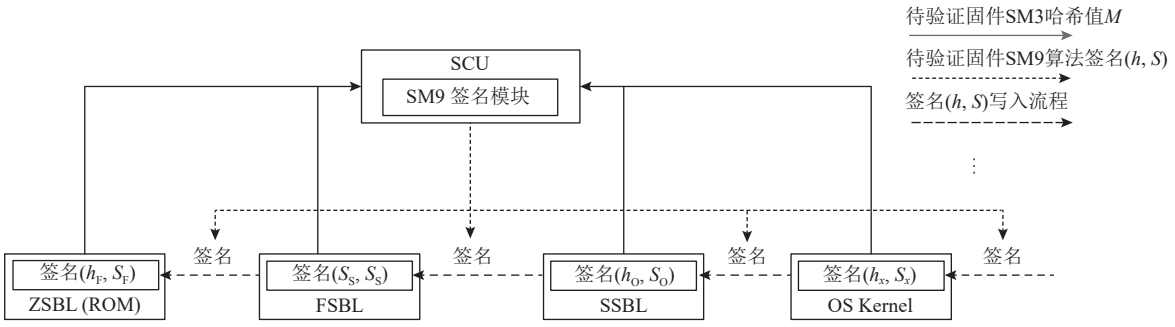



Fig. 5 The signature generation process in SeChain

图5 SeChain中的签名生成流程

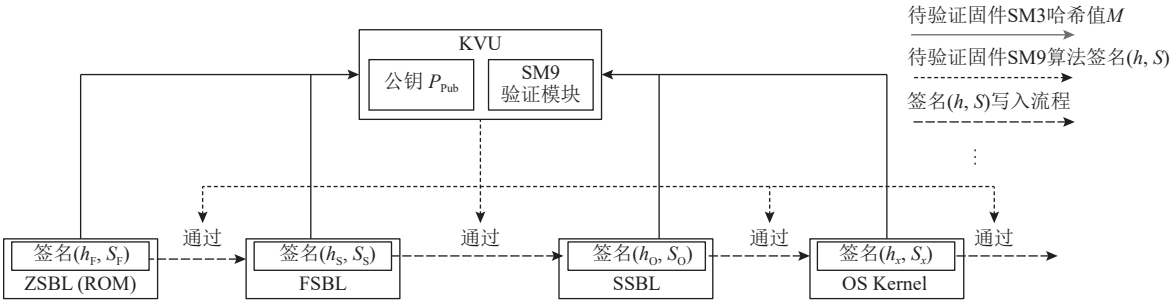



Fig. 6 The signature verification process in SeChain

图6 SeChain中的签名验证流程

任链的方式,逐级进行SM9校验与控制权传递,最终实现系统安全启动.若任何一个环节验证失败,则拦截启动并报相应错误.SeChain将在系统上电的第0阶段进行初始化并开始进行校验.整个启动流程主要可以分为4个阶段.

1)零级引导加载阶段.进入片上的ROM中读取地址为0x10000的位置,此时处于M-mode, ZSBL负责初始化片上设备,如初始化SPI和闪存控制器等;将控制权移交给KVU, KVU读取ROM帧头的签名$(h'_{\mathrm{F}},S'_{\mathrm{F}})$和用户公钥$ID$以及KVU中存储的主公钥$P_{\mathrm{Pub}}$,并计算下一固件FSBL的哈希值$M_{\mathrm{F}}$,验证$(M_{\mathrm{F}},ID,P_{\mathrm{Pub}},(h'_{\mathrm{F}},S'_{\mathrm{F}}))$是否匹配,若不匹配则阻截启动;若通过则进入下一单元.

2)第一引导加载阶段.将FSBL安装在缓存中,可直接读取并执行.通过验证的FSBL负责初始化片外设备、设置核心频率,配置DDR PLL、PHY和DDR控制器等;完成后将控制权移交给KVU, KVU读取FSBL帧头的签名$(h'_{\mathrm{S}},S'_{\mathrm{S}})$,计算下一阶段引导加载程序,即二级引导加载(second stage boot loader, SSBL)程序的哈希值$M_{\mathrm{S}}$,验证$(M_{\mathrm{F}},ID,P_{\mathrm{Pub}},(h'_{\mathrm{S}},S'_{\mathrm{S}}))$是否匹配,若不匹配则阻截启动;若通过则进入下一单元.

3)第二引导加载阶段.SSBL用于完成内核启动之前所必须的硬件和底层系统服务初始化.初始化成功之后,SSBL将从Flash中读出系统内核(OS kernel),随后加载在内存中.完成后将控制权移交给KVU, KVU读取SSBL帧头的签名$(h'_{\mathrm{O}},S'_{\mathrm{O}})$,计算下一阶段所需OS Kernel的哈希值$M_{\mathrm{O}}$,验证$(M_{\mathrm{F}},ID,P_{\mathrm{Pub}},(h'_{\mathrm{O}},S'_{\mathrm{O}}))$是否匹配,若不匹配则阻截启动;若通过则将控制权交给OS Kernel进入下一单元.

4)系统内核运行阶段.通过验证的OS kernel中包含了下一阶段应用程序的安全签名$(h'_x,S'_x)$,可进一步调用KVU校验应用程序的完整性.该过程与上述固件校验操作类似,不再赘述.

## 4 实验分析

为验证Sechain的有效性和性能表现,本节设计实现了具体的SeChain原型系统,基于FPGA进行了小型IoT设备的启动模拟,并设计了有效性验证实验、高效性验证实验和信任根攻击实验.

### 4.1 实验环境

#### 4.1.1 软核片上系统

SeChain原型系统基于VexRiscv 32 b RISC-V CPU与Litex SoC平台搭建.VexRiscv是一款流行的RISC-V核,使用SpinalHDL实现.相比于传统的RTL语言如Verilog、VHDL等,SpinalHDL更加灵活且高效.VexRiscv有效利用了SpinalHDL的特性,具有高性能和低资源消耗的特点,以及更加多样化的配置选项.这些特点有助于SeChain原型系统的快速高效实现.Litex SoC是一个开源的片上系统项目,其提供了对VexRiscv CPU的支持,并提供了如RAM、ROM、Timer、DRAM、以太网的外设支持,这也极大简化了SeChain的原型系统的开发流程.

本文在上述VexRiscv和Litex的基础上扩展开发了SCU, KVU模块,并修改总线配置使得这2种模块能正常工作.增加了安全内存模块Secure Memory,用于在运行时存放SCU, KVU模块的运行代码和计算结果.为了存储签名、公钥信息,本文还修改了板载ROM模块.通过修改实现了SeChain原型系统的核心功能用于后续的实验验证.

#### 4.1.2 硬件环境

本文将软核片上系统烧录在目标FPGA板上,实现了硬件级别的实验验证.目标实验平台采用Xilinx公司Artix-7 100T FPGA开发板,与上位机系统中的Xilinx Vivado设计软件通过USB高速串行总线接口连接,在USB接口上层模拟实现了UART协议接口.上位机系统使用UART接口将片上系统传入FPGA开发板,并与片上软件负载通信,使用开发板自带的JTAG进行调试.

#### 4.1.3 软件负载

原型系统的软件负载包括FSBL, SSBL和OS Kernel.其中,FSBL使用了Litex SoC默认的BIOS模块,支持在Litex SoC环境中初始化片外设备.SSBL使用了OpenSBI. OpenSBI是适用于RISC-V的底层系统接口,不仅起到了引导启动的作用,还提供了一些底层系统服务,以实现并提高内核跨硬件细节的能力.本文使用的OpenSBI版本为0.8,并针对Litex SoC增加了兼容性定制.为了更好地验证SeChain在真实环境下的使用效果,OS Kernel选取了流行的Linux操作系统内核作为上层软件负载,内核版本为5.0.9.

### 4.2 有效性验证实验

为了验证上述原型系统能够防御固件篡改攻击,本节设计了有效性验证实验.在IoT设备的启动流程中,攻击者可通过篡改并刷写固件的方式劫持正常启动流程,实现固件篡改攻击.具体攻击方式包括在引导程序中修改跳转地址、修改关键函数、增加额

外模块等, 实现对后续一系列的启动过程的干预, 以完成植入恶意代码、更换操作系统内核、修改内核配置等操作, 最终达到本地劫持或远程控制的目的. 为验证 SeChain 能够防御各类固件篡改攻击, 本节设置有效性验证对照实验. 实验组分别为对 SeChain 原型系统和原 VexRiscv 系统实施相同类型的固件篡改攻击.

为了更加全面反映启动流程中各阶段受攻击的启动状态, 固件篡改攻击方式包括对第 0 级引导程序中的主要 4 个模块 boot.c, main.c, readline.c, cmd_bios.c 进行代码增加、删除、修改操作. 本节实验共使用了 20 种攻击方式对固件进行攻击, 具体可见表 1. 固件被篡改后均可正常通过编译, 不影响其主要功能. 将固件重新编译并刷写到 ROM 中, 上电并观察启动过程的输出结果, 判断能否阻断异常启动过程. 如果能够正常阻断, 则表明具备防御固件篡改攻击能力.另外, 还设置了对照组, 在未受攻击的状态下正常启动 SeChain 原型系统和原 VexRiscv 系统, 并记录启动时间.

实验结果如表 1 所示, 在对固件实施任意类型的篡改攻击后, SeChain 均能防御固件篡改攻击, 中止异常启动过程. 由于 SeChain 原型系统记录了正常固件的签名, 而在启动过程中 KVU 模块将读取并计算当前加载的固件签名, 如与已记录的签名不符, 则并报告启动异常, 中止启动过程. 而原系统未能识别到攻击, 将使用被篡改的固件继续启动. 基于国密 SM9 算法的安全启动验证实验中, 在应用 SeChain 安全启动后的 VexRiscv CPU 上更改任意代码后, 都将引起 KVU 单元签名验证的失败并最终拦截启动, 由此充分说明 SeChain 安全启动的有效性. 由于在攻击测试中删改了代码数量, 编译后的系统固件大小发生变化. 因此, 输入 KVU 单元进行签名计算的数据量大小不一样, 不同攻击方式所对应的开机-阻截时间长短相差较大. 本节实验也统计了从设备上电开机至完成校验或被阻断的时间, 通过校验后的操作系统启动时间未计入统计. 应用 SeChain 后的开机-阻截时间平均值为 3.52 s. 而在不进行攻击的情况下, 设备上电至通过一次验证的时间平均值为 3.91 s, 由此说明 KVU 单元对于通过验证与不通过验证的签名都能高效可靠地给出判断结果. 本节实验证明, 采用 SeChain 安全启动后, 系统能够有效抵挡篡改攻击, 实现信任链的根节点可信.

**Table 1 Experimental Results of SeChain Secure Boot Verification Based on SM9 Algorithm**

**表 1 基于 SM9 算法的 SeChain 安全启动验证实验结果**

| 攻击位置 | 攻击方式 | 攻击检测结果 | 攻击阻断时间/s | 无攻击时验证成功时间/s |
|---|---|---|---|---|
| boot.c | 修改调用 boot helper 函数时传入的地址 | 验证失败，无法启动 | 3.1 | 3.9 |
| | 将 boot helper 传入地址硬编码为 0x00000 | 验证失败，无法启动 | 3.2 | 3.9 |
| | 修改 boot helper 中所有地址为固定值 | 验证失败，无法启动 | 3.2 | 4.0 |
| | 修改 seiral boot 启动时的 printf 函数 | 验证失败，无法启动 | 4.1 | 3.8 |
| | 在任意位置插入新的 printf 调用 | 验证失败，无法启动 | 4.4 | 3.9 |
| | 删除 serial boot 的 printf 函数 | 验证失败，无法启动 | 2.9 | 3.7 |
| main.c | 修改头部关于 litex logo 的 printf 语句 | 验证失败，无法启动 | 3.0 | 3.8 |
| | 大面积删除不影响编译完整性的 printf | 验证失败，无法启动 | 3.3 | 3.9 |
| | 修改调用 readline 函数时传入的参数 | 验证失败，无法启动 | 3.3 | 4.0 |
| | 删除对 readline 函数的调用 | 验证失败，无法启动 | 3.8 | 3.8 |
| | 删除除 serial boot 外的 boot 方式 | 验证失败，无法启动 | 3.6 | 3.9 |
| | 删除将是否进入 console 的判断语句 | 验证失败，无法启动 | 3.0 | 3.7 |
| | 删除 boot_sequence 函数的调用 | 验证失败，无法启动 | 3.8 | 3.8 |
| readline.c | 修改输入对照哈希表 | 验证失败，无法启动 | 4.3 | 3.9 |
| | 修改 readline 中对 command 的处理方式 | 验证失败，无法启动 | 3.5 | 3.9 |
| | 增加一个新的 command 处理结果 | 验证失败，无法启动 | 3.9 | 3.8 |
| cmd_bios.c | 任意位置增加一行 for 空循环语句 | 验证失败，无法启动 | 4.2 | 3.9 |
| | 添加 printf 函数调用, 打印某不被支持的 cmd | 验证失败，无法启动 | 3.3 | 3.7 |
| | 删除所有 printf 函数使无法展示可用 cmd | 验证失败，无法启动 | 2.9 | 3.8 |
| | 删除获得 help cmd 后的 printf 函数 | 验证失败，无法启动 | 3.6 | 3.9 |

### 4.3 高效性验证实验

加密算法通常消耗的硬件资源较多，可能造成较高的硬件负载占用，导致设备启动效率低下甚至无法启动. 为了分析 SeChain 的硬件资源消耗和时间效率，本节设计实现了 SeChain 的高效性验证实验.

首先统计分析了重要硬件组件的占用率，包括 LUT(look up table)，RAM，BRAM(block RAM)，DSP. 分别在原环境下和 SeChain 环境下重复执行 20 次启动引导测试，并统计该过程中的最高资源占用情况. 将原环境下的最高资源占用情况作为正常开机硬件占用，将 SeChain 环境下的最高资源占用情况与原环境的最高资源占用情况的差值作为签名验证算法的额外占用，剩余值作为硬件余量.

硬件使用率的评估结果如图 7 所示，原 VexRiscv CPU 在启动过程中未使用任何安全启动方案，硬件使用率较低. 在使用了 SeChain 安全启动后，启动过程中的 LUT，RAM，BRAM，DSP 硬件使用率分别增长了 47.10 个百分点、10 个百分点、44.8 个百分点、30 个百分点，仍有 44%、39%、48%、70% 的硬件余量，为其他可能的启动程序预留了充分的硬件空间. 且这些值为最高硬件占用，所需的平均硬件资源更低.

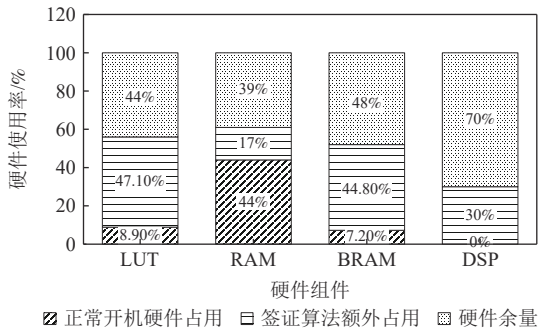



Fig. 7 Hardware overhead diagram of SeChain

图 7 SeChain 硬件开销图

此外，本节对启动时间进行了评估. 系统对原 VexRiscv CPU 与应用 SeChain 安全启动后的 VexRiscv CPU 进行了 20 次不同的完整启动测试，启动时间如图 8 所示. 相对于原 VexRiscv CPU，SeChain VexPiscv CPU 的启动时间有所增长，时间增长差值平均为 6.47 s. 原 VexRiscv CPU 在未启用任何安全机制的情况下，平均需要 20.22 s 的时间开销. 安全启动过程中需要加载相关代码，进行大量的加密运算，在该种情况下仍需 6.47 s 的时间开销，性能表现优异. 而本节实现的原型系统为了能够实现更好的通用性，并未对算法进行深度调优. 通过在具体实际场景下调试优化、精简冗余代码可以达到更好的时间开销效果，满足实际应用要求.

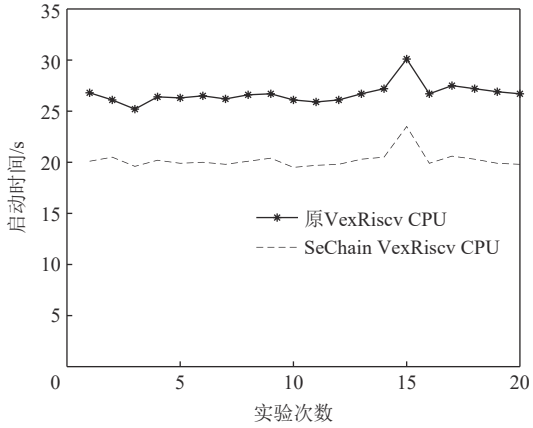



Fig. 8 Comparison of starting time growth diagram

图 8 启动时间增长图对比

### 4.4 信任根攻击实验

为了对比软件信任根和硬件信任根的防御效果，以及验证自主可控的安全启动机制的重要性，本节设计了实验信任根攻击实验.

软件信任根指将用户公钥写入可重复读写的闪存中，是目前 IoT 领域的常见保护方法，但也易受攻击者的篡改. IoT 设备制造商通常通过在固件中放置公钥、签名和解密算法，实现对固件升级过程的安全校验. 基于软件信任根的保护措施能在一定程度上提高破解门槛，但无法防御物理级的固件篡改攻击. 为验证硬件信任根相对于软件信任根的高安全性，本节设置实验 4b，模拟攻击者对软件信任根的篡改攻击. 在实验 4b 中，攻击者窃取设备 *ID*，通过修改 boot.c 中的 printf 函数实现固件篡改攻击. 随后攻击者使用私自生成的私钥对篡改后的固件重新签名，并将伪造后的签名和攻击者公钥写入闪存中，欺骗安全启动机制对固件完整性的校验过程，实施信任根攻击.

为了验证自主可控的安全机制的重要性，本节还设置了实验 4d，模拟非自主可控安全启动机制的潜在安全风险. 在非自主可控安全启动机制中，由于机制的实际实现方式未公开，其使用的加密算法本身及实现方式存在潜在的安全后门风险. 另外，密钥计算和签名过程也掌握在厂商手中. 因此，用户私钥存在潜在的泄露风险. 实验 4d 则模拟了攻击者知晓用户私钥的情况下使用用户私钥对篡改后的固件重新签名，实施信任根攻击.

实验 4b 和实验 4d 模拟了软件可信根和启动机制不安全的情况下的实验，并设定为实验组. 为了对比实验组的结果，本节还设置了对照组实验 4a 和对照组实验 4c. 4a 模拟了未使用安全启动的情况下能

否实施攻击. 4b 则模拟了 SeChain 的标准模式下使用不可篡改的硬件信任根存储用户公钥. 而攻击者尝试使用伪造的私钥计算受篡改固件的签名, 实施信任根攻击.

实验结果如表 2 所示, 4a、4b、4d 均可攻击成功, 而 4c 可以防御信任根攻击. 在 4a 中, 由于未采取任何安全启动机制, 没有对固件进行完整性的校验, 攻击可以被轻松实施. 在 4b 中, 虽然采取了安全启动机制, 但是攻击者可以随意篡改固件, 使用私自伪造的公私钥对固件进行签名计算, 并将签名和公钥替换到启动流程中. 得到的结果是在设备启动时 KVU 会使用被伪造的公钥以验证被伪造的固件签名, 如验证通过, 无法检测出攻击, 设备将执行攻击者篡改后的固件, 攻击成功. 在 4d 中, 由于私钥泄露, 攻击者已经知晓了用户私钥, 在固件被篡改后, 攻击者可使用用户私钥对固件重新签名. 上电启动后, KVU 将使用用户公钥计算固件签名的正确性, 检查如通过, 攻击成功. 而 4c 使用了硬件信任根, 使得攻击者无法通过篡改信任根的方式加入私自生成的公钥. 而且自主可控的安全启动机制能保证私钥的生成和签名计算过程中私钥不被泄露, 攻击者无法窃取用户私钥, 实现对篡改后的固件重新签名, 保障了安全性.

本节实验验证了硬件信任根和自主可控的安全启动机制能够对抗信任根攻击, 表明硬件信任根具备良好的防御效果, 而且自主可控的安全启动机制能进一步增强设备安全性, 达到了预设的实验目的.

## 5 讨　　论

本文所述的有效性验证、高效性验证和信任根攻击实验均达到了相应的实验目的, 完成了对 SeChain 的验证, 表明 SeChain 能实现对固件完整性的有效验证, 能够满足硬件资源占用和时间开销的性能要求, 且能对抗信任根攻击. 具体表现有 5 点:

1) SeChain 能够防御固件篡改攻击. 在有效性验证实验中, 任意代码的更改都会导致 KVU 验证错误, 从而拦截启动. 而对于安全未经篡改的固件, SeChain 能够高效地完成验签操作并将控制权交由可信的固件继续安全启动进程, 证明了本文系统的有效性. 本文实验中未涉及对下一阶段操作系统可信性的验证, 这是由于目前市面上的安全启动机制中验证操作系统可信性的技术已较为成熟, 而本文目标是从硬件可信根验证下一阶段启动固件的有效性, 下阶段操作系统可信性可由可信固件进一步校验, 不在本文考虑范围内.

2) SeChain 的硬件资源占用较低. 在高效性验证实验中, 通过在启动过程中对重要硬件组件 LUT, RAM, BRAM, DSP 的最高资源占用率进行统计, 发现上述组件在极值时仍有较多的硬件余量, 可满足资源占用要求. 但实验中也出现 DSP 计算单元占用增长较多的情况, 这是由于选用的轻量 Vexriscv RISC-V CPU 与片上系统 Litex SoC 默认不涉及加解密等复杂数学操作, DSP 资源占用极小. 而 SeChain 在加解密计算中需要进行大量的双线性对运算等复杂数学计算, 对 DSP 资源的占用较大. 另外, 本文实验出于通用性考量, 仅将 gmssl 库中的 SM9 计算相关的代码移植到 VexRiscv 中, 修复了部分兼容性问题, 且为方便后续移植到其他 RISC-V 平台, 并未对其进行深入的算法优化. 在实际应用中, 可针对具体硬件方案做深入的性能优化以降低资源占用.

3) SeChain 未造成较大额外时间开销. 在高效性验证实验中, 验证过程平均仅需 6.47 s 的额外时间开销, 可满足时间开销要求. SM9 算法中大量的指数运算和配对运算较为复杂, 在额外时间开销中占比较大. 近年来, 提高 SM9 速率的快速算法和优化硬件资源使用的双线性对优化运算已经相继被提出, 如杨国强等人[32] 在 2020 年提出的一种基于 FPGA 的 SM9 快速实现方法可将 SM9 算法的性能提升 2.3 倍. 朱留富等人[51] 在 2023 年提出了离线进行指数运算和配对运算的在线/离线验证方案, 能够解决物联网环境

**Table 2　Experimental Results of Trust Root Attack**

**表 2　信任根攻击实验结果**

| 攻击实验 | 信任链 | 信任根位置 | 攻击方式 | 攻击检测结果 | 平均时间/s |
|---|---|---|---|---|---|
| 4a | | | 篡改 boot.c 中的 printf 函数 | 无验证，攻击成功 | 3.3 |
| 4b | 软件 | 闪存 | 攻击者窃取设备 *ID*，篡改 boot.c 中的 printf 函数，使用攻击者私钥伪造签名，并修改闪存中的公钥为对应的攻击者公钥 | 验证通过，设备启动异常，攻击成功 | 7.3 |
| 4c | 硬件 | 写入 ROM | 攻击者窃取设备 *ID*，篡改 boot.c 中的 printf 函数，使用攻击者私钥伪造签名，并修改闪存中的公钥为对应的攻击者公钥 | 验证失败，停止启动，攻击失败 | 4.3 |
| 4d | 硬件 | 写入 ROM | 攻击者预先知晓用户私钥与系统参数，篡改 boot.c 中的 printf 函数后，使用用户私钥伪造签名 | 验证通过，设备启动异常，攻击成功 | 6.8 |

中轻量级设备在线计算资源受限问题. 另外, 还可以通过多种方案进一步优化 SeChain 的时间开销, 如替换实验平台 Vexriscv RISC-V CPU 及 Litex SoC. 本文的实验平台 Vexriscv RISC-V CPU 及 Litex SoC 具有简洁易用等特点, 在具体的实际应用场景中, 可针对不同类型的 IoT 设备, 选择对应功能特性的 RISC-V CPU, 如在追求更快的启动速度与高性能的情况下, 可应用 SeChain 于国产“香山”系列等高性能 RISC-V CPU, 并进行相应的适配和优化, 以达到加快启动速度的目的.

4)SeChain 的自主可控能有效提升设备安全性. 流行的安全启动机制, 如 Boot Guard, Secure Boot 尚未公开其源代码, 密钥生成仅可由授权的设备制造商完成, 其使用的加密算法本身及实现方式可能存在潜在的安全后门风险. 另外, 密钥计算、签名过程和私钥存储也掌握在设备制造商手中. 因此, 私钥存在潜在的泄露风险. 如 2023 年 5 月, 著名设备制造商 MSI 数据泄露事件造成 166 种产品受 Boot Guard 私钥泄露的影响, 甚至可被用于攻击其他设备制造商的硬件, 造成的危害非常深远①. SeChain 安全启动机制的实现方式公开透明, 且使用自主的国密 SM9 算法, 避免了传统安全启动机制的黑盒化. 另外, SeChain 支持终端用户初次使用激活时再生成密钥对和签名, 从源头避免了上游设备制造商信息泄露导致的安全风险. 特别是对于国家关键基础设施领域和国家重要机关部门, 使用自主可控的安全启动机制及自主产权的密码算法更具实际意义. 另外, 可信根攻击实验也实际证明了自主可控的安全启动机制的安全优势.

5)SeChain 的可定制性和硬件可信根更加适用于 IoT 领域. IoT 领域对设备成本和功耗有较高要求, 通过基于 RISC-V 的开源 SoC 方案, 开发者可更加灵活地选择目标硬件平台. 而且相对于广泛应用的 TPM 模块, SeChain 更加轻量级, 设备实现的复杂度更低. 基于硬件的可信根也保证了在无人值守的环境中能防御固件篡改攻击. 这些特性使得 SeChain 在保证 IoT 设备安全性的同时又能有效控制成本和功耗.

## 6 结论与展望

针对当前基于安全启动技术保障 IoT 固件完整性仍不完善的问题, 本文设计并实现了基于国密 SM9 算法的 RISC-V 安全启动机制 SeChain. SeChain 利用 RISC-V 架构的开源、轻量、可定制的特点, 设计了一个简洁高效的安全启动机制. 通过在 SoC 内部增加 SCU, KVU 模块, SeChain 实现了基于 SM9 算法的密钥生成、签名计算和签名校验, 并进一步通过基于验证引导的多级安全启动流程, 从不可篡改的硬件信任根出发, 逐级完成对下一阶段引导程序的完整性校验, 实现安全启动机制. 本文基于 VexRiscv CPU 在 FPGA 硬件平台仿真完成了 SeChain 原型系统, 并通过实验验证了 SeChain 的有效性、高效性和可靠性. 实验结果表明, SeChain 能够防御固件篡改攻击, 具备较低的硬件资源占用率, 且未造成较大额外时间开销. 通过自主可控的设计, SeChain 可有效提升设备安全性. 另外, SeChain 的可定制性和硬件可信根也使其更加适用于 IoT 领域. SeChain 的设计思路在满足安全可信启动的同时, 更能为国产 RISC-V 生态的安全自主可控提供有力保障. 另外, SeChain 相关源代码已发布至开源社区平台[14].

目前研究仍有 2 点不足之处和改进空间.

1)SeChain 目前尚缺乏良好的固件更新机制. 由于设备 ROM 仅可在激活时一次性写入, 无法重复读写, 而多级验证引导机制在保证高安全性的同时也损失了一定的灵活度. 后续研究中将在 SeChain 的基础上提出更优化的安全机制, 例如固件修改后信任链条可手动更新、IoT 设备易主后的设备重新激活支持等, 实现灵活度和高安全性的更好融合.

2)SeChain 目前仅在支持 RISC-V 架构的 FPGA 硬件平台上进行了仿真验证实验, 未使用真实 RISC-V 芯片进行测试和评估. 这主要受限于 SoC 设计、验证和流片的时间周期较长、成本高昂, 因此本文研究中并未使用真实的 RISC-V 芯片测试. 后续将考虑在真实的 RISC-V 芯片的基础上设计采用 SeChain 的 SoC, 同时应用于小型 IoT 设备上, 将其在实际场景中具体应用.

**作者贡献声明:** 芮志清提出了文章整体方法思路、实验方案, 并撰写和修改论文; 梅瑶负责完成实验、分析实验结果和撰写部分章节; 陈振哲参与论文实验验证, 负责解决实验过程中遇到的相关技术问题; 吴敬征提供指导意见, 并参与论文修订; 凌祥审阅论文, 提供修改意见; 罗天悦参与思路讨论和论文的修订; 武延军提出指导性意见, 并参与论文修订.

① MSI leak: tips for users, organizations, and developers. https://www.kaspersky.com/blog/msi-firmware-keys-leak/48300/

## 参 考 文 献


[1] GSMA. The Mobile Economy [R/OL]. London: GSMA Head Office. 2023 [2023-09-30]. https://www.gsma.com/solutions-and-impact/connectivity-for-good/mobile-economy/wp-content/uploads/2023/03/270223-The-Mobile-Economy-2023.pdf

[2] He Xiaoqing. RISC-V processor embedded development overview[J]. Microcontrollers & Embedded Systems, 2020, 20(11): 1−6 (in Chinese)
(何小庆. RISC-V 处理器嵌入式开发概述[J]. 单片机与嵌入式系统应用, 2020, 20(11): 1−6)

[3] T-Head. T-Head Semiconductor Co. , Ltd[EB/OL]. 2023[2023-08-31]. https://github.com/T-head-Semi.html

[4] EulixOS. EulixOS[EB/OL]. (2022-12-28) [2023-09-15]. https://eulixos.com

[5] OWASP Foundation. OWASP Internet of things[EB/OL]. 2023 [2023-08-11]. https://owasp.org/www-project-internet-of-things/

[6] Yang Hong, Wang Xiaochun, Lin Guanchen, et al. Research on trusted Internet of things technology and standardization[J]. Information Technology & Standardization, 2023, 65(5): 10−15, 20 (in Chinese)
(杨宏, 王晓春, 林冠辰, 等. 可信物联网技术与标准化研究[J]. 信息技术与标准化, 2023, 65(5): 10−15, 20)

[7] Zhao Bo, Fei Yongkang, Xiang Shuang, et al. Research and implementation of secure boot mechanism for embedded systems[J]. Computer Engineering and Applications, 2014, 50(10): 72−77 (in Chinese)
(赵波, 费永康, 向骕, 等. 嵌入式系统的安全启动机制研究与实现[J]. 计算机工程与应用, 2014, 50(10): 72−77)

[8] Ma Hongfeng. Secure boot design of IoT devices based on hardware root of trust[J]. Microcontrollers & Embedded Systems, 2021, 21(10): 16−17, 21 (in Chinese)
(马洪峰. 基于硬件信任根的物联网设备安全启动设计[J]. 单片机与嵌入式系统应用, 2021, 21(10): 16−17, 21)

[9] UEFI Forum, inc. Unified extensible firmware interface specification[EB/OL]. (2022-08-29)[2023-08-10]. https://uefi.org/sites/default/files/resources/UEFI_Spec_2_10_Aug29.pdf

[10] Naked Security. Serious Security: TPM 2.0 vulns–is your super-secure data at risk[EB/OL]. (2023-03-07)[2023-9-10]. https://nakedsecurity.sophos.com/2023/03/07/serious-security-tpm-2-0-vulns-is-your-super-secure-data-at-risk

[11] Intel. Intel Hardware Shield Below-the-OS Security: Information technology cybersecurity white paper[EB/OL]. 2022[2023-04-15]. https://www.intel.com/content/dam/www/central-libraries/us/en/documents/below-the-os-security-white-paper.pdf

[12] Jawad H Y, Wong M M, Pudi V, et al. Lightweight secure-boot architecture for RISC-V system-on-chip[C]//Proc of the 20th Int Symp on Quality Electronic Design (ISQED). Piscataway, NJ: IEEE, 2019: 216−223

[13] SpinalHDL. VexRiscv[EB/OL]. (2021-11-16)[2024-03-15]. https://github.com/SpinalHDL/VexRiscv

[14] Rui Zhiqing, Mei Yao. SeChain[EB/OL]. (2024-03-10)[2024-03-15]. https://github.com/m2kar/SeChain

[15] Chen Shiwei. Research on the attack methods of the Hash functions[D]. Zhengzhou: Information Engineering University of the People's Liberation Army, 2012
(陈士伟. 杂凑函数的攻击方法研究[D]. 郑州: 解放军信息工程大学, 2012)

[16] Rivest R. The MD5 message-digest algorithm: RFC1321[R/OL]. 1992[2024-03-15]. https://www.rfc-editor.org/info/rfc1321

[17] Stevens M, Bursztein E, Karpman P, et al. The first collision for full SHA-1[C]//Proc of the 37th Annual Int Cryptology Conf. Berlin: Springer, 2017: 570−596

[18] Dworkin M J. SHA-3 standard: Permutation-based Hash and extendable-output functions: NIST FIPS 202[R/OL]. Gaithersburg, Maryland: National Institute of Standardsand Technology, 2015[2024-03-15]. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf

[19] Wang Xiaoyun, Yu Hongbo. SM3 cryptographic Hash algorithm[J]. Journal of Information Security Research, 2016, 2(11): 983−994 (in Chinese)
(王小云, 于红波. SM3 密码杂凑算法[J]. 信息安全研究, 2016, 2(11): 983−994)

[20] Wang Xiaoyun, Yin Y L, Yu Hongbo. Finding collisions in the full SHA-1[C] //Proc of the 25th Annual Int Cryptology Conf. Berlin: Springer, 2005: 17−36

[21] Wang Xiaoyun, Yu Hongbo. How to break MD5 and other Hash functions[C]//Proc of Annual Int conf on the Theory and Applications of Cryptographic Techniques. Berlin: Springer, 2005: 19−35

[22] Wikipedia. Public-key cryptography[EB/OL]. (2024-02-22)[2024-03-15].https://en.wikipedia.org/w/index.php?title=Public-key_cryptography&oldid=1209561246#Algorithms

[23] Diffie W, Hellman M E. New directions in cryptography[J]. IEEE Transactions on Information Theory, 1976, 22(6): 644−654

[24] Shen Changxiang, Zhang Huanguo, Feng Dengguo, et al. Summary of information security[J]. SCIENTIA SINICA Informationis, 2007, 37(2): 129−150 (in Chinese)
(沈昌祥, 张焕国, 冯登国, 等. 信息安全综述[J]. 中国科学: 信息科学, 2007, 37(2): 129−150)

[25] Hales T C. The NSA back door to NIST[J]. Notices of the American Mathematical Society, 2013, 61(2): 190−192

[26] Schwennesen B, Bray H. Elliptic curve cryptography and government backdoors[EB/OL]. (2016-04-24)[2024-03-15]. https://blkcipher.pl/assets/pdfs/BAS_Paper3_EllipticCurveCryptography.pdf

[27] Dong Yixiao, Quan Jianbin, Wang Mingru, et al. Research on the application of SM9 algorithm in the field of Internet of things security[J]. Telecom Engineering Technics and Standardization, 2022, 35(9): 22−27, 32 (in Chinese)
(董一潇, 全建斌, 王明儒, 等. 国密 SM9 算法在物联网安全领域的应用研究[J]. 电信工程技术与标准化, 2022, 35(9): 22−27, 32)

[28] Song Weiping. Digital signatures based on public-key cryptography[J]. Chinese Journal of ICT in Education. 2007, 162(15): 59−60 (in Chinese)
(宋维平. 基于公开密钥密码体制的数字签名[J]. 中国教育信息化: 高教职教, 2007, 162(15): 59−60)

[29] Yin Ming. A review of identity-based cryptographic algorithms

SM9[J]. Information Technology and Informatization, 2020, 242(5): 88−93 (in Chinese)
(殷明. 基于标识的密码算法 SM9 研究综述[J]. 信息技术与信息化, 2020, 242(5): 88−93)

[30] Yuan Feng, Cheng Zhaohui. Overview of SM9 identity cryptography algorithms[J]. Journal of Information Security Research, 2016, 2(11): 1008−1027 (in Chinese)
(袁峰, 程朝辉. SM9 标识密码算法综述[J]. 信息安全研究, 2016, 2(11): 1008−1027)

[31] Bian Xingbin, Ma Junming, Hu Zhiyong, et al. Application research of SM9 in ubiquitous Internet of things[J]. Information Technology & Network Security, 2022, 41(2): 27−32 (in Chinese)
(边杏宾, 马俊明, 胡志勇, 等. SM9 在泛在物联网中的应用研究[J]. 信息技术与网络安全, 2022, 41(2): 27−32)

[32] Yang Guoqiang, Kong Fanyu, Xu Qiuliang. A fast implementation method for SM9 based on FPGA[J]. Journal of Shandong University (Natural Science), 2020, 55(9): 54−61 (in Chinese)
(杨国强, 孔凡玉, 徐秋亮. 一种基于 FPGA 的 SM9 快速实现方法[J]. 山东大学学报 (理学版), 2020, 55(9): 54−61)

[33] Avizienis A, Laprie J C, Randell B, et al. Basic concepts and taxonomy of dependable and secure computing[J]. IEEE Transactions on Dependable and Secure Computing, 2004, 1(1): 11−33

[34] Tomlinson A. Smart Cards, Tokens, Security and Applications, Introduction to the TPM[M]. Berlin: Springer, 2017

[35] Shen Changxiang, Zhang Huanguo, Wang Huaimin, et al. Research and development in trustworthy computing[J]. SCIENTIA SINICA Informationis, 2010, 40((2): ): 139−166 (in Chinese)
(沈昌祥, 张焕国, 王怀民, 等. 可信计算的研究与发展[J]. 中国科学: 信息科学, 2010, 40(2): 139−166)

[36] Lee R B. Processor architecture for trustworthy computers[C] //Proc of Asia-Pacific Conf on Advances in Computer Systems Architecture. Berlin: Springer, 2005: 1−2

[37] Parno B, Mccune J M, Perrig A. Bootstrapping trust in commodity-computers[C] //Proc of IEEE Symp on Security and Privacy. Piscataway, NJ: IEEE, 2010: 414−429

[38] Tan Liang, Xu Zhiwei. Research progress on trust chain transmission based on trusted computing platform[J]. Computer Science, 2008, 35(10): 15−18 (in Chinese)
(谭良, 徐志伟. 基于可信计算平台的信任链传递研究进展[J]. 计算机科学, 2008, 35(10): 15−18)

[39] Dave A, Banerjee N, Patel C. Care: Lightweight attack resilient secure boot architecture with onboard recovery for RISC-V based SoC[C] //Proc of the 22nd Int Symp on Quality Electronic Design (ISQED). Piscataway, NJ: IEEE, 2021: 516−521

[40] Ling Zhen, Yan Huaiyu, Shao Xinhui, et al. Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT nodes[J/OL]. Journal of Systems Architecture, 2021[2024-03-15]. https://www.sciencedirect.com/science/article/abs/pii/S1383762121001661

[41] Raj H, Saroiu S, Wolman A, et al. fTPM: A software-only implementation of a TPM chip[C/OL] //Proc of the 25th USENIX Security Symp. Berkeley, CA: USENIX Association , 2016: 841−856

[42] OpenAnolis. Commercial Cryptography Best Practices White Paper [EB/OL]. 2022[2023-08-30]. https://openanolis.github.io/whitebook-shangmi/secure_boot.html (in Chinese)
(OpenAnolis. 商用密码技术最佳实践白皮书 [EB/OL]. 2022[2023-08-30].https://openanolis.github.io/whitebook-shangmi/secure_boot.html)

[43] Liu Chang, Wu Yanjun, Wu Jingzheng, et al. Survey on RISC-V system architecture research[J]. Journal of Software, 2021, 32(12): 3992−4024 (in Chinese)
(刘畅, 武延军, 吴敬征, 等. RISC-V 指令集架构研究综述[J]. 软件学报, 2021, 32(12): 3992−4024)

[44] Menon A, Murugan S, Rebeiro C, et al. Shakti-T: A RISC-V processor with light weight security extensions[C]//Proc of the Hardware and Architectural Support for Security and Privacy. 2017[2024-03-15]. https://dl.acm.org/doi/10.1145/3092627.3092629

[45] Wong M M, Haj-Yahya J, Chattopadhyay A. SMARTS: Secure memory assurance of RISC-V trusted SoC[C] //Proc of the 7th Int Workshop on Hardware and Architectural Support for Security and Privacy. Los Angeles, CA: HASP, 2018: 1−8

[46] Lee D, Kohlbrenner D, Shinde S, et al. Keystone: An open framework for architecting TEEs [EB/OL]. (2022-04-02) [2024-03-15]. http://docs.keystone-enclave.org/en/latest/

[47] Lebedev I, Hogan K, Devadas S. Secure boot and remote attestation in the Sanctum processor[C] //Proc of the 31st Computer Security Foundations Symp(CSF). Piscataway, NJ: IEEE, 2018: 46−60

[48] Nikiema P R, Palumbo A, Aasma A, et al. Towards dependable RISC-V cores for edge computing devices[C]//Proc of the 29th IEEE Int Symp on On-Line Testing and Robust System Design (IOLTS). Piscataway, NJ: IEEE, 2023: 1−7

[49] Zhang Yuxin, Rui Zhiqing, Li Weiwei, et al. PEC-V: Memory overflow defense mechanism based on RISC-V coprocessor[J]. Computer Systems & Applications, 2021, 30(11): 11−19 (in Chinese)
(张雨昕, 芮志清, 李威威, 等. 基于 RISC-V 协处理器的内存溢出防御机制[J]. 计算机系统应用, 2021, 30(11): 11−19)

[50] Ahn N Y, Lee D H. Forensic issues and techniques to improve security in SSD with flex capacity feature[J]. IEEE Access, 2021, 9: 167067−167075

[51] Zhu Liufu, Li Jiguo, Lai Jianchang, et al. Attribute-based online/offline signature scheme based on SM9[J]. Journal of Computer Research and Development, 2023, 60(2): 362−370 (in Chinese)
(朱留富, 李继国, 赖建昌, 等. 基于商密 SM9 的属性基在线/离线签名方案[J]. 计算机研究与发展, 2023, 60(2): 362−370)



**Rui Zhiqing**, born in 1997. PhD candidate. His main research interests include system security. Internet of things security, and software supply chain security.

**芮志清**, 1997 年生. 博士研究生. 主要研究方向为系统安全、物联网安全、软件供应链安全.



**Mei Yao**, born in 2000. Master. Her main research interests include channel codes and software security.

**梅　瑶**, 2000 年生. 硕士. 主要研究方向为信道编码、软件安全.



**Chen Zhenzhe**, born in 1998. PhD candidate. His main research interests include physical unclonable functions and network security.

**陈振哲**, 1998 年生. 博士研究生. 主要研究方向为物理不可克隆函数、网络安全.



**Wu Jingzheng**, born in 1982. PhD, professor, professor supervisor. Senior member of CCF. His main research interests include system security, vulnerability mining, and operating system security.

**吴敬征**, 1982 年生. 博士, 研究员, 博士生导师. CCF 高级会员. 主要研究方向为系统安全、漏洞挖掘、操作系统安全.



**Ling Xiang**, born in 1992. PhD. Member of CCF. His main research interests include software security and AI security.

**凌　祥**, 1992 年生. 博士. CCF 会员. 主要研究方向为软件安全、人工智能安全.



**Luo Tianyue**, born in 1990. Engineer. His main research interests include operating system security analysis, code vulnerability mining, and AI security.

**罗天悦**, 1990 年生. 工程师. 主要研究方向为操作系统安全分析、代码漏洞挖掘、人工智能安全.



**Wu Yanjun**, born in 1979. PhD, professor, PhD supervisor. Distinguished member of CCF. His main research interests include operating systems, machine learning system software, and system security.

**武延军**, 1979 年生. 博士, 研究员, 博士生导师. CCF 杰出会员. 主要研究方向为操作系统、机器学习系统软件、系统安全.