文章编号:1671-4598(2024)04-0287-06 DOI:10.16526/j.cnki.11-4762/tp.2024.04.040 中图分类号:TP309.1 文献标识码:A

基于标识密码的双向认证的安全启动协议

冯云龙，张宏科，刘林海
（中国电子科技集团公司 第54研究所，石家庄 050081）

摘要：传统安全启动方案的认证环节是基于PKI体制实现，在设备数量剧增的情况下，证书的管理会增加系统复杂性，认证过程仅实现了单向认证，安全性不足；此外，由于选择了链式信任链，导致了在启动过程中的信任传递损失较大；针对上述问题，文章提出了一种基于IBC的Secure boot方案，即IBCEB方案；该方案使用了IBC体制的国家标准SM9算法作为实现方法，实现了无证书的双向认证协议，并对信任链模型进行了优化，降低了信任传递的损失；在ZC706评估板上进行了测试，测试结果表明，设备在双向认证后成功启动，提高了系统的安全性。

关键词：Secure boot；IBC；SM9；无证书；双向认证

A Secure Boot Protocol for Bidirectional Authentication Based on IBC

FENG Yunlong，ZHANG Hongke，LIU Linhai
（No.54 Research Institute，China Electronics Technology Group Corporation，Shijiazhuang 050081，China）

Abstract：The authentication process of traditional secure boot schemes is based on the Public Key Infrastructure (PKI) system. With the sharp increase in the number of devices，certificate management will increase system complexity，and the authentication process only achieves one-way authentication，resulting in insufficient security. In addition，because of the selection of a chain-based trust chain，there is a significant loss of trust transmission during the startup process. In response to the above issues，a secure boot scheme based on the identity-based encryption (IBC) system is proposed，namely the IBCEB scheme. The scheme uses the national standard SM9 algorithm of IBC system as an implementation method，implements the uncertified bidirectional authentication protocol，optimizes the model of trust chain，and reduces the loss of trust transmission. Test on the ZC706 evaluation board，the test results show that the device successfully starts after the bidirectional authentication，improving the security of the system.

Keywords：Secure boot；IBC；SM9；no certification；bidirectional authentication

0 引言

由于在设备生命周期的早期阶段，防火墙和反病毒软件等传统对策尚未启用[1]，设备在启动过程中遭到的攻击很难被检测到，导致系统安全受到影响。近年来，随着密码学的发展，安全启动（Secure Boot）方案较好地解决了这一问题。它可以防止恶意软件和其他恶意操作对启动过程进行攻击[2]，强制每个引导阶段对后续阶段进行身份验证，只有经过授权实体签名的固件才能被加载，从而在整个引导过程中建立信任链，保证系统的安全。在此过程中，设备通常需要借助efuse、BootROM等相关硬件作为信任根（RoT，root-of-trust）来引导信任链[3]。可见，硬件对系统的安全性非常重要。

然而，日益复杂的全球化硬件供应链正在威胁着核心硬件的可信度。安全引导中的RoT不可避免地受到了供应链的攻击[4]。具体来说，现在许多原始设备制造商将其硬件或固件开发外包给第三方供应商，而没有对其安全状况进行全面检查[5]，在这种情况下，设备可能会在多次中转中被拦截并被植入破坏组件[6]。上述情况表明，系统启动时硬件必须真实可信。例如，攻击者可以拦截客户的嵌入式设备并用恶意的处理器替换原处理器，或者植入一个额外的芯片破坏处理器间的通信，这些攻击被称为中间人攻击。这些攻击可能会导致恶意映像的加载，从而在启动阶段就控制了系统。上述情况表明，系统启动时硬件必须真实可信。

随着科学技术的发展，很多研究人员对硬件在保护系统安全方面做了深入研究。Jiang等人[7]提出了一种基于ARM TrustZone技术的安全引导方案，在传统Secure boot的基础上，借助开源OP-TEE和ARM硬件的支持，该方案在ZC702上实现了同时运行OP-TEE可信操作系统和Linux系统，敏感操作在OP-TEE中执行，结果通过驱动接口返回给Liunx，较好地保护了用户的信息安全。Kumar等人[8]提出了一种后量子安全引导（PQSB）方法，该方案完全由硬件实现，虽然其安全性高，速度快，但是不便于部署在已有设备上。Ehret等人[9]设计了一个以安全为重点的低功耗SoC架构，具有硬件信任根，用于边缘设备。该体系结构被命名为最优资源分配的可重构边缘计算。Pocklassery等人[10]基于物理不可克隆技术提出了一种针对FPGA

收稿日期:2023-09-08； **修回日期**:2023-10-25。
基金项目:中国电子科技集团公司第五十四研究所项目研究发展基金(SXX22107X042)。
作者简介:冯云龙(1998-),男,硕士。
引用格式:冯云龙,张宏科,刘林海.基于标识密码的双向认证的安全启动协议[J].计算机测量与控制,2024,32(4):287-292,307.

的自认证安全引导方法。上述这些方案都默认了设备本身的真实可信，仅实现了设备对用户身份的单向认证，忽略了设备本身也可能成为攻击者的事实。

此外，传统 secure boot 方案存在如下问题：首先，认证方式都是基于证书的公钥基础设施（PKI，public key infrastructure）体制，在设备增多时，证书的管理会变得复杂。其次，采用的链式信任链过长，导致信任值损失较多。而基于标识的密码体系（IBC，identity-based encryption）是一种密码学体制，旨在解决传统 PKI 中的密钥分发与管理问题。在传统的 PKI 中，用户需要通过证书颁发机构来获取公钥证书。而在 IBC 中，用户可以直接使用自己的身份信息作为公钥，无需经过第三方机构的证书颁发。考虑到上述优点，提出了一种基于 IBC 体制的 Secure boot 方案，简称为 IBCEB 方案。以 Xilinx 的 Zynq7000 系列 SoC 为例介绍传统 Secure boot 流程并进行分析，详细介绍 IBCEB 方案并对其进行安全性分析，介绍双向认证在 ZC706 评估板上的实现过程，总结并展望未来工作。IBCEB 方案实现了在启动过程中的无证书双向认证，并且优化了信任链的传递，降低了信任传递的损失。

1 传统 Secure boot

Xilinx 推出的 Zynq-7000 系列 SoC，已广泛应用于各行各业。ZYNQ-7000 系列所支持的 Secure boot 非常具有代表意义。本节将以 Zynq 系列为例介绍并分析传统 Secure boot 的过程。

1.1 安全启动镜像

启动镜像文件又称为 BOOT.BIN 文件，是 Zynq 启动的必备文件。它由 FSBL、bit 流、U-boot、Linux 内核和 BootROM 组成。

BootROM 程序由 Xilinx 编写，出厂后无法更改。BootROM 程序为设备启动后第一段执行的代码，其主要作用有根据输入信号初始化 CPU、初始化基本系统功能和判断启动方式等。在其执行完毕后会将 CPU 控制权移交给 FSBL。另外，如果启用 Secure boot，那么 BootROM 还会负责完成对 FSBL 的验签以及解密工作。

FSBL 作为一段 Secure boot 中的核心程序，它的作用有进一步初始化 PS、使用 bit 流文件对 PL 侧编程、装载裸机程序或者 U-boot、执行用户定义的代码，并移交 CPU 控制权。如果启用了 Secure boot 功能，FSBL 还会对接下来的程序进行 RSA 的验签认证以及 AES 解密工作。

在实际操作时，BOOT.BIN 由 Xilinx 提供的 BootGen 工具生成。BootGen 可以整合从 FSBL 到 APP 的所有代码，并且支持选择每一块区域是否需加密或者验签。在整合的过程中，会先添加一段引导头信息，每一段程序都会变为相应的分区（Partition），每个分区的数据都可被 AES 加密、RSA 签名，以确保镜像无法被随意读取，并且如果启用 RSA 认证，那么在每个分区后会追加数字证书，以确保分区数据的可信性。BOOT.BIN 的结构如图 1 所示。

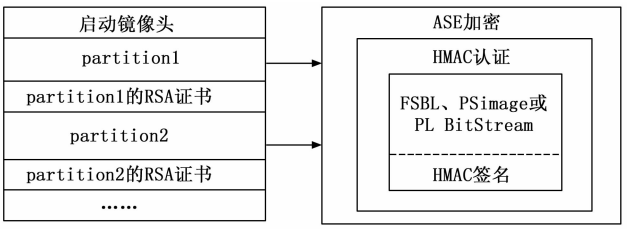

图 1 BOOT.BIN 格式

RSA 认证模式为传统的 PKI 体系。在该体系下，会建设一个证书权威中心 CA（Certificate Authority）。传统 Secure Boot 中关于认证涉及 4 种秘钥：主公钥 PPK、主私钥 PSK、次公钥 SPK、次私钥 SSK。其中 PPK 和 PSK 作为 CA 用于发放数字证书的秘钥对。RSA 秘钥的详细信息如表 1 所示。

表 1 认证相关秘钥

秘钥名称	所有者	功能
Primary public key(PPK)	CA	用于签发数字证书
Primary secret key(PSK)	CA	
Secondary public key(SPK)	USER	用于系统镜像签名
Secondary secret key(SSK)	USER	

每一个 Partition 后的证书含有 RSA 认证的参数 Modulus（n）、PPK、SPK、PSK 对 SPK 的签名值和 SSK 对 Partition 内容的签名值。

1.2 Secure Boot 流程

传统 Secure Boot 方案一般分为 3 个阶段：准备阶段、认证阶段和解密阶段。在启动过程中，需要每一分区对下一分区进行认证和解密。为了存储信任根，需要使用到 eFuse（Electronic Fuse）。它是一次性可编程存储器，在向其烧写内容后用户层面是无法读取的。在 Zynq 平台上，PS 侧和 PL 侧均有一个 eFuse。PS 侧的 eFuse 用于验证 CA 是否合法。PL 侧的 eFuse 存放 AES 加密的秘钥，用于给每个分区解密。为了简化说明将 PS 侧的 eFuse 称为 eFuse1，PL 侧的 eFuse 称为 eFuse2。下面介绍各阶段完成的工作，其中，Secure boot 中各符号含义如表 2 所示。

表 2 Secure boot 中各符号含义

符号	含义
p	Partition 内容
$H(m)$	m 的哈希值
$E_{AES}(m)$	AES 秘钥加密后的 m
$D_{AES}(m)$	AES 秘钥解密后的 m
$S_{PSK}(m)$	使用 PSK 对 m 签名
$V_{PPK}(m)$	使用 PPK 对 m 进行验签

传统 Secure boot 流程如图 2 所示。

准备阶段：用户需要生成 AES 密钥和 RSA 密钥（SPK，SSK），并向指定 CA 注册数字证书，即可获取 S_{PSK}（SPK）。然后向 eFuse1 中烧写 CA 的公钥的哈希值 H（PPK），最后为了实现 AES 的加密，需要向 eFuse2 中烧写

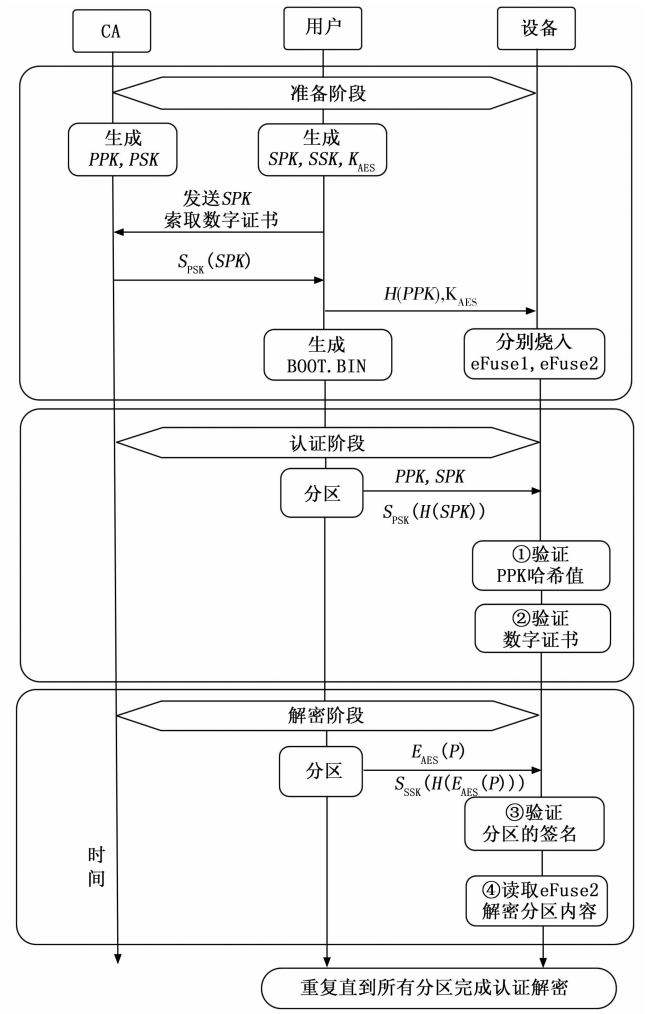

图 2　传统 Secure boot 流程

AES 密钥。准备完毕后按照上一节的启动镜像格式生成 BOOT. BIN 文件。

认证阶段：1）为了确定当前分区中的签名是否来自合法的 CA，设备将比较计算式（1），如果一致则说明该镜像来自可信的 CA 授权的用户；2）为了确认用户的身份是否真实，设备将读取分区中的 SPK 值和证书，计算式（2）进行比较。如果相等则说明用户身份也真实。

$$H(PPK) = eFuse1 \tag{1}$$

$$V_{PPK}(S_{PSK}(H(SPK))) = H(SPK) \tag{2}$$

解密阶段：3）解密之前需要计算式（3），确保分区的完整性，以防止恶意镜像对 AES 引擎的攻击。4）读取 eFuse2 中的 AES 密钥，然后将该分区送入 AES 引擎解密，即计算式（4）。之后将 CPU 控制权移交给下一分区。随后下一分区重复上述步骤，直到最后一块分区完成，设备的信任链构建成功，则视为安全启动成功。

$$V_{SPK}(S_{SSK}(H(E_{AES}(p)))) = H(E_{AES}(p)) \tag{3}$$

$$p = D_{eFuse2}(E_{AES}(p)) \tag{4}$$

1.3　传统 Secure boot 方案分析

传统 Secure boot 的认证方案中采用的 PKI 技术综合使用了数字摘要技术、数字签名等密码技术以及一套完整的证书管理机制来提供安全服务[11]。系统建设有公信力的认证中心（CA，certification authority）鉴定用户身份，然后为用户签发数字证书。数字证书安全地将用户身份和用户密钥绑定在一起。用户在业务系统中先交换证书，然后使用公私钥完成用户的身份认证、访问控制和信息安全传递等操作。它的特点是简单易懂，但是其中存在着证书管理复杂，链式信任链信任值损失多，需要消耗的计算资源高等缺点，可能并不适合部署在计算资源紧缺的嵌入式设备上。

传统 Secure Boot 的安全模型是建立在使用者是攻击者，设备是被攻击者的基础上，只实现了设备对用户的认证。如今，芯片供应链在设计、制造和分销等方面都已全球化[12]。PCB 设计人员仅在内部设计和生产一小部分组件，而依赖于合同制造商、分销商和 EDA 工具等各种可能含有恶意的第三方组件，从而使伪设备攻击供应链。这种情况下设备成为攻击者，仅仅实现单向认证是不安全的。

2　IBCEB 方案

IBCEB 方案采用了基于标识密码体制的 SM9 算法，可以直接使用身份信息进行密码运算，签名私钥则通过可信第三方密钥生成中心（KGC，key generation center）生成。此外，为了检测到设备是否被篡改以及实现用户与设备的双向身份认证，需要使用物理不可克隆函数（PUF，physical unclable function）技术[13-15]。PUF 会根据生产电路板时微小的差异产生唯一设备 ID。此 ID 结合 SM9 即可实现用户和设备的双向认证。

2.1　方案模型

本方案系统模型如图 3 所示。

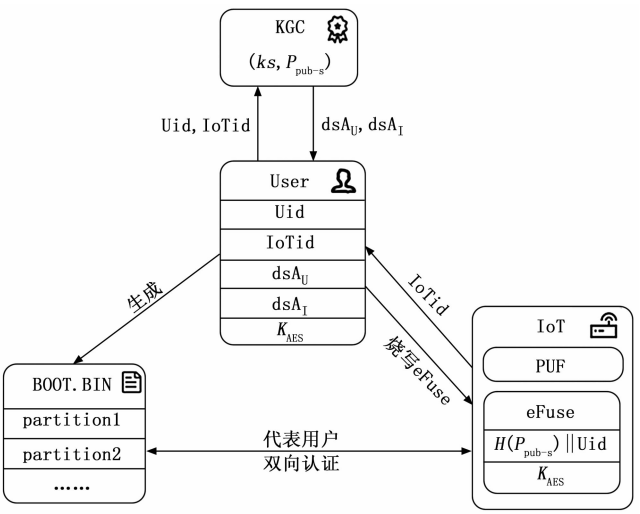

图 3　系统模型

涉及 4 个实体和 3 个阶段。下面对这 4 个实体、3 个阶段以及相关安全假设做简要介绍。其中，IBCEB 中的符号如表 3 所示。

表 3 IBCEB 中的符号

符号	含义
Ks	KGC 主私钥
$P_{pub\text{-}s}$	KGC 主公钥
Uid	用户的唯一标识
IoTid	通过 PUF 生成的设备标识
dsA_U	User 签名私钥
dsA_I	设备签名私钥
K_{AES}	AES 加密密钥

KGC：秘钥生成中心，方案中的核心机构，负责用户和设备的签名私钥生成和私钥分发。假设 KGC 完全可信，其私钥 ks 永远不会泄露。

User：IoT 设备的所有者和使用者。用户 ID 可以是其手机号、邮箱等序列。User 会保存向 KGC 申请的用户签名私钥 dsA_U，以及向 KGC 申请的 IoT 设备的签名私钥 dsA_I。假设用户存放的 dsA_U dsA_I 是安全的不会泄露。

IoT 设备：指用户使用的产品，在某一台设备到达用户手中后，用户会通过 PUF 技术为设备生成唯一的序列号 IoTid，并发送给 KGC 注册得到 dsA_I 签名私钥。随后用户会向其 eFuse1 中烧写 H（$P_{pub\text{-}s}$ || Uid）。假设 eFuse 中数据除了设备本身，无法被他人读取。

BOOT.BIN：IoT 设备的启动镜像文件，该文件由用户使用各个分区文件经过 AES 加密和 SM9 签名后生成。BOOT.BIN 在生成并存入 IoT 设备之后，就代表着用户的身份，与设备进行认证。

2.2 方案设计

IBCEB 方案主要包括系统初始化、启动镜像生成和设备启动 3 个步骤。其中涉及椭圆曲线参数、签名私钥生成、签名算法和验签算法，以上均按照 SM9 国家标准进行选取和实现。

2.2.1 初始化阶段

1）KGC 按国标 SM9 选取参数，并生成（ks，$P_{pub\text{-}s}$）；

2）用户对 IoT 设备运行 PUF 模块，得到 IoTid；

3）用户与 KGC 建立安全的连接；

4）用户将自身 Uid 和设备 IoTID 发送给 KGC；

5）KGC 分别为 Uid 和 IoTid 生成 dsA_U 和 dsA_I，并发送给用户；

6）用户生成加密密钥 K_{AES}；

7）用户在 eFuse1 中烧写 H（$P_{pub\text{-}s}$ || Uid），eFuse2 中烧写 AESkey。

2.2.2 镜像生成阶段

1）User 准备好正常启动所需的 FSBL、U-boot、Kernel、APP 等相关文件。

2）对于以上每一个 partition 执行如下操作：

（1）对 partition 使用 AES 加密得到 E_{AES}（p）；

（2）使用 dsA_U 对 E_{AES}（p）进行签名得到 S_{dsAU}（E_{AES}（p））；

（3）使用 dsA_I 对 $P_{pub\text{-}s}$ 进行签名得到 S_{dsAI}（$P_{pub\text{-}s}$）；

（4）最后得到加密后的 partition 结构如图 4 所示。

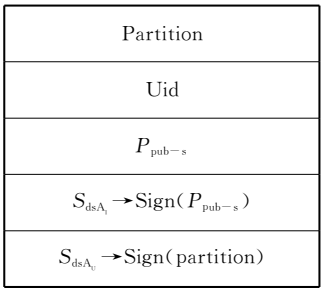

图 4 加密后 partition 结构

3）添加相关控制信息后，将上述合并至一个文件 BOOT.BIN 中。

2.2.3 启动阶段

1）BootRom 阶段执行如下操作：

（1）读取 Partition1，即 FSBL 分区。

（2）读取分区中 $P_{pub\text{-}s}$，Uid，并计算验证 H（$P_{pub\text{-}s}$ || Uid）是否等于 efuse1 中的值，相等则继续，不相等则终止启动。

（3）调用 PUF 模块获得 IoTid，并使用 IoTid 计算 V_{IoTid}（S_{dsAI}（$P_{pub\text{-}s}$））若验签通过则继续，不通过则终止启动。

（4）使用分区中提供的 Uid 计算 V_{Uid}（S_{dsAU}（E_{AES}（p））），若验证通过则继续启动，不通过则终止启动。

（5）使用 eFuse2 中的 K_{AES} 解密当前 Partition。

（6）转移 CPU 控制权给 FSBL。

2）FSBL 阶段：

（1）读取 Partition2，即 bitstream。

（2）重复 1）中（2）～（6）。

（3）读取 partition3，即 U-boot 和 kernel。

（4）重复 1）中（2）～（6）。

（5）转移 CPU 控制权给 Linux Kernel。

3 安全性分析

IBCEB 方案实现了用户和设备的双向认证，能够抵御伪造设备攻击。IBCEB 方案的信任链采用了星型和链型相结合的方式。

3.1 信任链模型

Demper-Shafer 理论适用于分析信任链的传递过程，其中的信任衰减法则定义如下：A 对 B 的信任值称为 T（A，B），B 对 C 的信任值称为 T（B，C），A 经过 B 对 C 的信任值称为 T_B（A，C），则有：

$$T_B(A,C) \leqslant min(T(A,B),T(B,C)) \tag{5}$$

通过公式（5）可知，链式模型在信任传递过程中，信任传递次数多，导致了信任值损失多，此外链中任意节点出现问题都会摧毁整条信任链，而星型信任链没有多级信任传递，信任值损失小[16-18]。IBCEB 方案采用了链式与星型信任链结合的方式，缩短了信任的传递路程，减少了信任值损失。信任链模型如图 5 所示。

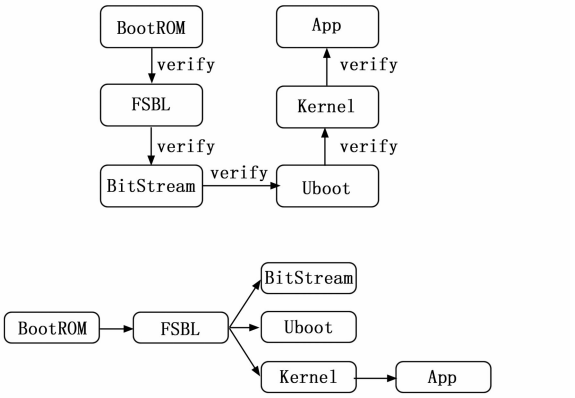

图 5　信任链模型

3.2　安全性分析

假设敌手试图在 BootROM 执行期间读取芯片内部的信息。Secure boot 是从 BootROM 开始执行，如果 BootROM 检测到启用 Secure boot，那么就会禁用 PS 和 PL 的 debug 端口。因此想要通过 JTAG 接口在启动阶段访问内部寄存器或内存数据是不可能的[19-21]。

假设敌手试图读取在外部存储器中的 FSBL 和操作系统镜像。镜像文件是先经过 AES 加密[22-24]，后经 SM9 签名的。AES 秘钥由用户自己生成并烧写到 PL 的 eFuse 中，efuse 中的秘钥是安全的，因为 eFuse 不提供读取接口，因此敌手无法在没有 AES 秘钥的情况下解密镜像数据。

假设敌手试图修改镜像文件，试图使用恶意代码对 AES 解密引擎进行攻击。这种攻击在本方案下是无效的。因为所有的 partition 都经过了 dsA_U 的签名，且 BootROM 在解密前会先使用 Uid 进行验签，验签不通过则无法进入 AES 解密阶段。由于 dsA_U 是 KGC 由用户标识 Uid 生成，在验签通过的同时还可实现设备对用户身份的认证。

假设敌手使用恶意设备替换原设备来试图窃取用户信息。这种伪造设备攻击在本方案下也是无效的。因为 IBCEB 不仅使用 dsA_U 对 partition 进行签名，还用 dsA_I 对 P_{pubs}进行签名。在启动过程中，由用户代码读取设备 IoTid 并对 $P_{pub\text{-}s}$进行验签，以实现用户和设备的双向认证。伪造设备计算得到的标识 IoTid 必定与原设备不同，这就保证了伪造设备是无法通过认证的。

4　实验分析

为了验证基于标识密码的双向认证的安全启动协议方案的可行性，选择在 Xilinx ZC706 测试评估板上进行实验分析。ZC706 搭载的芯片为 XC7Z045，属于 Zynq7000 系列。

4.1　实验过程

SM9 标识加密算法是基于标识的非对称加密算法，256 bit 的 SM9 算法加密强度等同于 2 048 bit 密钥的 RSA 加密算法，并且运算速度优于 RSA。因此，基于标识密码的双向认证的安全启动协议方案中认证算法选择了国标 SM9 算法，IBC 体系的 SM9 算法在 2020 年纳入国家标准（GB/T 38635.2-2020）。SM9 国标中选取了高效率的配对友好曲线，并且选择了运算速度快的 R-ate 双线性对运算，以减少 Miller 循环次数，提高计算效率。

SM9 算法需要在有限域上进行双线性对的计算，由于其计算过程复杂，完全重新实现并不现实。因此在实际实现中，选择了由北京大学开发并维护的开源国密算法库 GmSSL。GmSSL 中实现了对所有国密算法、标准和安全通信协议的全面功能覆盖。其中，在 SM9 算法的实现上，GmSSL 实现了定义在上椭圆曲线的各种运算和各类数据转换，以及双线性对的计算，并且在其中加入了常见的优化方法。

传统 Secure boot 的 RSA 认证是在 FSBL 阶段调用 RSA 库形式实现，因此为了方便对比性能差异，需要将 FSBL 中的 RSA 认证代码删除，添加基于 GmSSL 实现的 SM9 验签部分的代码。并在 FSBL 初始化完成之后，加载 bit 流之前，即在 Fsbl Hook Before Bit stream Dload（）的 hook 函数中实现。根据启动介质的不同，对后续分区实现相应的验签操作。

实际上由于 FSBL 是由 Bootrom 搬运至 OCM 执行，而 OCM 的大小仅有 256 kB，其中还有 64 kB 地址不连续，因此，需要精简 GmSSL 库，只保留验签所需代码，且选择 Release 模式编译 FSBL，但在精简至最少代码时仍然无法通过编译，此时还需要修改 lscript.ld 文件，将 .heap 段映射至 ps7_ram_1_S_AXI_BASEADDR 中，最终才能实现将 SM9 移植到 FSBL 中。

在实现了基于 SM9 的认证功能后，还需考虑对镜像文件的加密。由于 zynq 本身硬件支持的解密效率高达 100 MB/s，因此仍选择沿用 Zynq7000 自身硬件支持的 AES 算法。最后在实际生成镜像文件时，使用 BootGen 工具，对镜像进行加密以及整合操作。生成启动镜像如图 6 所示。

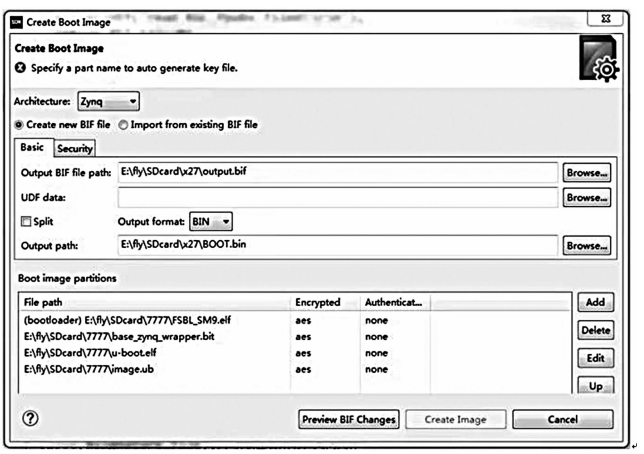

图 6　生成启动镜像

在启动镜像生成完毕后，用户需使用 SM9 算法对 BOOT.BIN 中各分区添加分区的签名信息和 $P_{pub\text{-}s}$的签名信息。对 BOOT.BIN 签名过程如图 7 所示。

eFuse 的烧写需要使用 Xilinx 提供的 eFuse 驱动。在计

```
[fyl@localhost SM9_user_sign]$ ./SM9sign
************begining sign*************
Reading dsA                OK
SM3 init                   OK
Reading source file        OK
signing file               OK
generating sign file       OK
***************sign done**************
[fyl@localhost SM9_user_sign]$
```

图 7 对 BOOT.BIN 签名过程

算好 H（$P_{\text{pub-s}}$ || Uid）后，将值写入驱动的配置文件中。在编译完成后，选择 JTAG 引导模式，使用 XSCT 命令行对 eFuse 进行烧写。基于 AES 密钥的存储有两种方式，一种为存储至 eFuse 中，虽然 eFuse 中更加安全，不会泄露密钥，但是之后都无法更改；另一种可以选择存放在 BBRAM 中，BBRAM 是由电池供电的一块 RAM，在开发板断电后其中的数据不会消失，并且密钥可以多次修改。为了方便测试选择了 BBRAM 形式存储 AES 密钥，BBRAM 在开发板断电后数据不会消失且密钥可以修改。烧写 AES 密钥如图 8 所示。

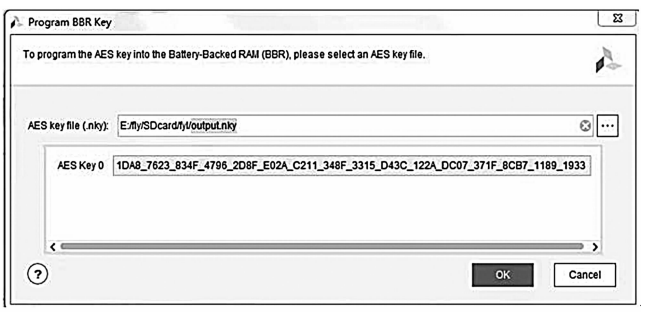

图 8 烧写 AES 密钥

4.2 实验结果与分析

嵌入式设备的启动所需的时间一般主要由 3 部分组成：BootROM 将 FSBL 从存储介质搬运至 OCM 的时间 t_1，FSBL 将后续 U-boot、Kernel 从存储介质搬运至 DDR 时间 t_2，以及 U-Boot、Kernel 程序自身的初始化时间 t_3。

对于 t_1，由于 BootROM 的不可修改性和不可访问性，导致无法测量 t_1 的准确数值，但是一般而言由于 FSBL 的大小仅为 100 kB，相比于 bit 流、U-Boot、kernel 一般 20 MB的大小，t_1 可以忽略不计；对于 t_3，在选择常用版本的 Linux，U-Boot 的情况下，测量得到 U-Boot 启动耗时 1 287 ms，Linux 内核启动耗时 4 065 ms。这里并不包括搬运时间，而是指搬运至 DDR 后各自的执行时间，此时间和存储介质的传输速度无关，即在不更换 CPU 和 DDR 的情况下可以看作固定值；对于 t_2，在启用 Secure boot 时，FSBL 在搬运之前会先读取分区信息进行 RSA 认证，通过后再将数据通过 PCAP 总线传送至 AES 解密引擎解密，最后再送入 DDR。FSBL 的代码可以添加用户自定义的部分，因此可以便捷地实现对上述功能计时。综上所述，t_1 无法测量且占比很小，t_3 为固定值，所以 t_2 是系统启动时间中有意义的统计因素。

此次实验采用了常见的含有 Bit 流、U-boot、Linux 的 23 M 的镜像从 SD 卡来启动设备，设备成功启动如图 9 所示。

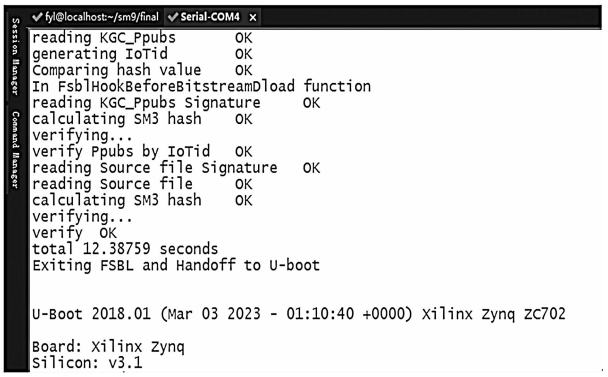

图 9 设备成功启动

由此测得 t_2 值为 12.39 s。由于从 SD 卡启动会涉及文件的读写，双线性对的运算复杂，且 OCM 仅有 256 kB，因此面对较大的 bit 流、Linux 内核文件时，会导致效率的下降。虽然 IBCEB 方案牺牲了部分效率，但是 IBCEB 方案实现了设备与用户的双向认证，保证了系统的安全。

在安全性方面，为了测试 IBCEB 能否抵御篡改镜像攻击，首先选择在 ZC706 评估板上选择常用的 U-Boot、Linux 版本实现了正常启动。然后通过更换 APP 中的内容来模拟对系统镜像的篡改攻击。在启动时，FSBL 计算验签的结果不通过，启动终止。启动验证失败测试结果如图 10 所示。

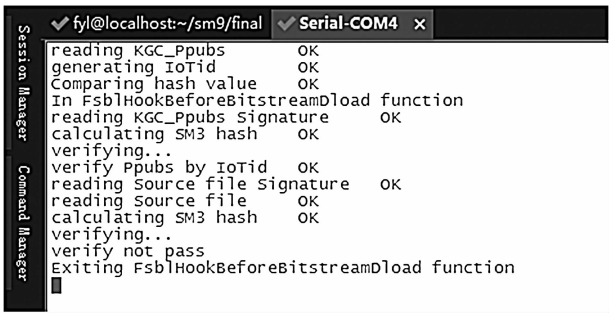

图 10 启动验证失败

5 结束语

文章提出了基于标识密码的双向认证的安全启动协议的方案，简称 IBCEB 方案。文章对该方案进行了详细的理论分析，同时，在 ZC706 评估板上进行了相关实验分析。实验结果显示，实现了用户和设备之间无证书的双向认证协议，提高了系统的安全性。此外，还对传统信任链模型进行了优化，采用了星型和链式相结合的信任链，降低了信任传递的损失。此方案适用于公共安防、智慧家居等诸多领域，使用前景广阔。尽管 IBCEB 方案存在优势，但也存在一些不足。首先，一旦 KGC 主密钥被泄露，则任何一个用户的私钥都可以被计算出来，就会严重威胁到系统的安全。其次，虽然理论上 SM9 的性能优于 RSA，但是 SM9

（下转第 307 页）