文章编号：0253-374X(2024)S1-0176-09 **DOI**：10.11908/j.issn.0253-374x.24721

抽样校验和编译优化的车载MCU安全启动方案

张　平[1]，余卓平[1]，张　佩[2]
(1. 同济大学 汽车学院，上海 201804；2. 上海创时汽车科技有限公司，上海 201206)

摘要：随着网联技术的发展，针对车辆的网络攻击事件频发，黑客可将恶意软件刷入车载微控制单元(MCU)，实现恶意控制车辆或窃取重要信息的目的。针对上述攻击场景，需在车载MCU中加入安全启动功能进行防护，以保证只有合法的软件能被运行。为此，本文对当前主流车载MCU安全启动方案进行研究，总结它们在安全性、稳定性、低耗时方面的性能表现和存在的不足，然后针对低耗时性能提出优化方案，最后通过搭建实物系统对优化方案的可行性和有效性进行验证。结果表明，采用抽样校验和编译优化的方案可以在满足低耗时要求的同时保护尽可能多的重要代码。

关键词：微控制单元；硬件安全模块；消息认证码；链接器
中图分类号：U461.91　　**文献标志码**：A

Research on the Secure Boot Scheme of Automotive MCU

ZHANG Ping[1], *YU Zhuoping*[1], *ZHANG Pei*[2]
(1. College of Automotive Studies, Tongji University, Shanghai 201804, China; 2. SAIC TTTECH Auto Technology Co., Ltd., Shanghai 201206, China)

Abstract: With the development of Internet technology, attacks against vehicles occur frequently. Hackers can flash malware into the automotive MCU to maliciously control the vehicle or steal important information. For the above attack scenarios, it is necessary to add secure boot to the automotive MCU. Functions are protected to ensure that only legitimate software can be run. This paper first conducts an in-depth study of the current mainstream secure boot schemes, summarizes their performance and shortcomings in terms of safety, stability, and low time-consuming, and then proposes an optimization scheme for low-time-consuming performance , and finally verified the feasibility and effectiveness of the optimization scheme through the built physical system. The results show that sampling verification and compilation optimization can protect as much important code as possible while satisfying the time-consuming verification.

Keywords: micro control unit (MCU); hardware security module (HSM); message authentication code (MAC); linker

随着汽车智能化、网联化技术的发展，当前车辆不再是一个封闭的系统，为了实现与外部的网络通讯，其增加了大量的对外通讯接口，同时也为黑客攻击提供了更多的攻击窗口，降低了网络攻击难度，进而导致当前针对车辆的网络攻击事件频发。由于汽车使用场景的特殊性，一旦发生网络攻击事件将会导致无法接受的严重后果，例如，车辆召回事件，危害驾驶员生命和财产安全等。因此，目前车辆的网络安全受到了国家和汽车行业的高度重视，随着UN/WP.29(联合国车辆法规协调论坛)发布了R155和R156法规，车辆网络安全也正式进入了法规时代。

R155法规适用于1958协议下的成员国，其要求车辆必须经过网络安全管理体系认证(CSMS)和车辆型式认证(VTA)才能发行销售。法规中针对产品研发过程明确提出了一个威胁场景列表(Annex5)，要求列表中列出的所有威胁场景在执行威胁分析和风险评估(TARA)时均被充分考虑并处理，其中就包含了恶意软件攻击场景[1]。

R156法规对OEM的软件开发管理体系以及软件升级功能提出了相应的要求，法规中明确要求在软件远程升级过程中要保证其真实性和完整性，要以合理的方式阻止软件包被破坏，并阻止无效的升级。OEM需要提供所采用保护机制的详细说明，确保在车辆上仅可执行经过身份认证和完整性校验的软件包[2]。

基于以上车辆网络安全法规要求，就需要开发

收稿日期：2023-09-28
基金项目：国家科学自然基金(52072268)
第一作者：张平(1979—)，男，博士研究生，高级工程师，主要研究方向为汽车整车及模块化架构开发和智能驾驶。
E-mail:zhangping9814@163.com

商在车载控制器开发早期执行TARA分析，识别潜在的威胁和安全漏洞，综合考虑攻击可行性、影响等级等因素，确定系统可能存在的风险及其风险等级，从而选择合适的风险处理方案[3]。而针对恶意软件攻击场景，通常采用安全启动方案发现并阻止恶意软件被执行，以达到缓解该项风险的目的。

另外，车载控制器应用场景对安全启动功能在安全性、稳定性和低耗时三个方面的性能提出了很高的要求。

安全性：由于车载控制器面对的攻击场景较为复杂，且攻击危害很严重，可能会危害驾驶员的生命安全，所以对安全启动方案的安全性要求很高。通常要借助硬件安全模块（HSM）实现高安全性能，有效防止安全启动方案被黑客破解。

稳定性：由于车载控制器对功能安全要求很高，所以对安全启动功能的稳定性也提出了很高的要求，避免由于安全功能失效引起的系统无法正常启动现象。

低耗时：由于车载控制器对启动时间要求很高，一般要求要小于150ms，而安全启动功能的加入会带来不可避免的毫秒级耗时，且校验代码量越大耗时越严重，对启动时间影响很大，所以对安全启动的低耗时性能也提出了很高的要求[4]。

本文主要从车载MCU安全启动功能的安全性、稳定性和低耗时三个性能方面入手，分析当前主流的安全启动方案，研究其在安全性、稳定性和低耗时三个性能方面存在的问题，并针对当前存在的问题提出解决方案，旨在设计出一个安全、可靠、低耗时的车载MCU安全启动方案。

1　相关依赖技术

1.1　CMAC算法

CMAC（cipher block chaining-message authentication code）算法，是一种基于密钥的消息认证码算法，可以用于保护数据的完整性和真实性。由于其具有执行速度快的优点，当前主流的安全启动方案均使用该算法。使用CMAC算法实现数据完整性校验的基本方法如图1所示。即先使用密钥将一个“消息块”生成为固定长度的消息认证码（MAC），一旦“消息块”的任意部分内容被篡改，其对应的消息认证码也会随之改变，因此通过对比前后两次用同一把密钥对“消息块”计算出的消息认证码是否相等，即可验证“消息块”是否被篡改过。CMAC算法一般使用AES对称密码学算法作为基础。算法耗时与“消息块”大小成近似正比关系[5]。

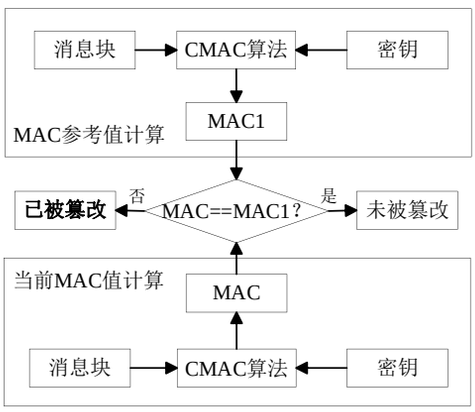

图1　CMAC算法使用方法

Fig.1　CMAC algorithm mechanism

1.2　硬件安全模块（HSM）

要实现车辆的网络安全仅靠软件无法实现，需要硬件和软件相结合，因为密码学算法的安全性完全由密钥的机密性决定，将密钥存放在软件中容易泄露，进而导致网络安全方案失效。因此，宝马和奥迪联合推出了SHE（security hardware extension）规范，对安全硬件模块提出了一系列要求，例如，支持密钥的安全存储、有独立的CPU，CMAC算法对128 kB数据进行校验，耗时要小于10 ms等[6]。

为了应对V2X场景下的车辆网络安全，欧盟组织了一个项目（Evita）基于SHE规范提出了HSM硬件规范，把HSM分为high、medium、light三个等级[7]。三个等级间的区别如图2所示。

	Full HSM	Medium HSM	Light HSM
RAM	✓	✓	optional
NVM	✓	✓	optional
Symmetric cryptographic engine	✓	✓	✓
Asymmetric cryptographic	✓		
Hash engine	✓		
Counters	✓	✓	optional
Random-number generator	✓	✓	optional
Secure CPU	✓	✓	
I/O component	✓	✓	✓

图2　Evita HSM等级分类

Fig.2　Evita HSM classes

HSM主要解决了如下问题：密钥和机密数据泄露；加密和解密运算消耗大量HOST资源，执行速度较慢，安全性低；加解密算法在HOST内存中运行，可能被恶意提取中间重要数据。

目前主流的车载MCU基本都支持HSM硬件规范，即一颗车载MCU中有两个微处理器：分别为HSOT和HSM，两者都具有各自独立的CPU、RAM

和ROM资源，HSM有权访问HOST的RAM和ROM资源，而HOST无权访问HSM的RAM和ROM资源，两者通过中断、寄存器和共享内存实现信息交互，保证了HSM内部重要信息的安全性[8]。HOST用于普通的任务调度处理，HSM则专门用于提供加解密算法和安全存储服务。对于安全启动场景，HSM可向HOST提供CMAC算法和密钥安全存储服务，提高整个安全启动方案的安全性。

2 安全启动方案研究分析

首先介绍车载MCU安全启动基础的原理性框架，然后分析基于该框架的几种具体实现方案，从安全性、稳定性和低耗时三个性能方面总结各自的优点和存在的问题。

2.1 安全启动基本框架

对于车载MCU来说，其软件(HEX)一般会分Bootloader和APP两部分，Bootloader是系统上电或复位启动后，运行的第一段程序，主要用于系统启动、硬件初始化和软件刷写。APP是应用层逻辑代码，主要是一些控制算法。正常MCU启动流程为系统上电或复位后，先执行Bootloader，完成相应的初始化工作后启动内核，然后直接跳转至APP，执行相应的控制算法。而安全启动就是在软件执行跳转前通过CMAC算法校验即将执行软件的完整性，完整性校验成功才会执行正常跳转，否则，启动失败，进入刷新模式，等待软件刷新，如图3所示。安全启动功能的应用，可有效发现恶意软件并阻止其在控制器中被运行的现象发生。

图3中的安全启动功能大致分为三个部分。MAC参考值计算模块：使用安全启动密钥先对APP进行CMAC算法计算，获得参考值MAC1。完整性校验模块：在执行跳转前，根据Bootloader的请求，使用安全启动密钥对当前APP再进行CMAC算法计算，获取当前值MAC，然后对比当前值MAC是否与参考值MAC1相等，相等则返回true，否则返回false。跳转决策模块：Bootloader基于完整性校验模块返回的结果做出响应，返回true正常启动，返回false则停止启动，进入刷新模式。

2.2 安全启动方案一

方案一将2.1中的MAC参考值计算模块单独做了一个脚本工具，在电脑端执行，如图4所示。MAC参考值生成流程简述：

(1) 将APP代码段的起始地址和长度信息放在

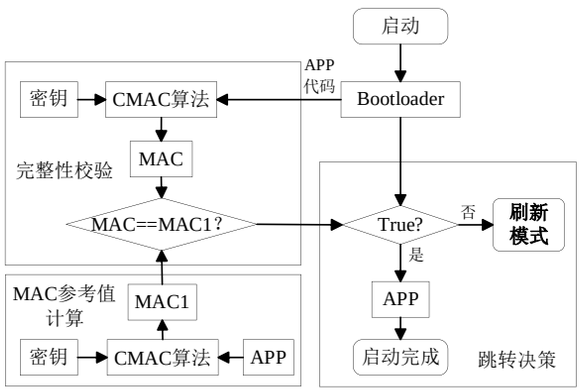

图3 安全启动基本框架

Fig.3 Secure boot basic framework

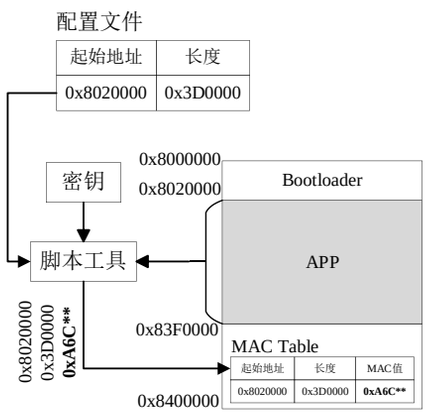

图4 MAC参考值生成流程

Fig.4 MAC reference value generation process

配置文件中。

(2) 使用脚本工具调用密钥对配置文件中定义的代码段进行CMAC计算，算出的MAC值作为参考值使用，然后将对应的起始地址、长度和MAC值以图中MAC Table的格式放到HEX的指定位置。

将2.1节中的完整性校验和跳转决策模块分别放在HSM和HOST端执行，如图5所示。MAC验证与跳转决策流程如下：

(1) 在启动过程中，当需执行跳转时，HOST从指定位置获取之前放入的MAC Table信息，提取其中的起始地址、长度和MAC参考值传至HSM，请求HSM进行完整性校验。

(2) HSM基于HOST端所提供的信息，调用对称加密算法引擎使用密钥执行MAC校验，并将校验结果反馈至HOST端。

(3) HOST同步等待HSM验证完成后，获取验证结果，基于验证结果决定是正常启动，还是停止启动，进入刷新模式。

该方案存在的问题如下：

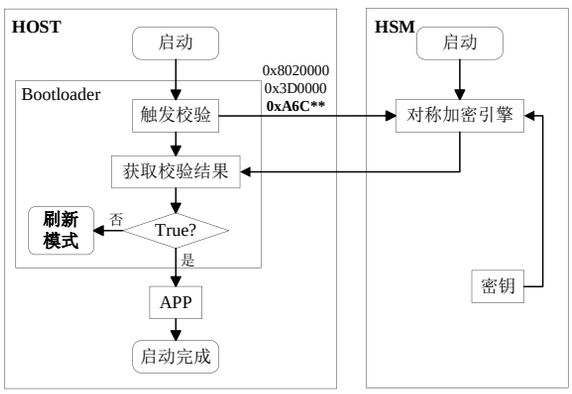

图5　MAC验证与跳转决策流程

Fig.5　MAC verification and jump process

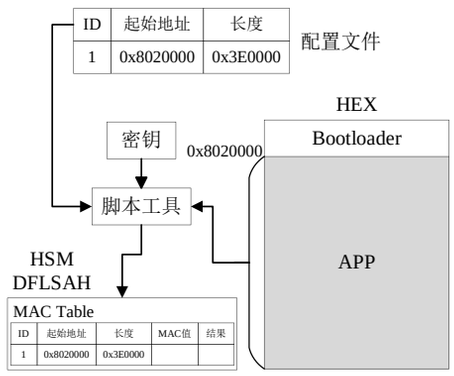

图6　代码段配置流程

Fig.6　Code segmentation configuration process

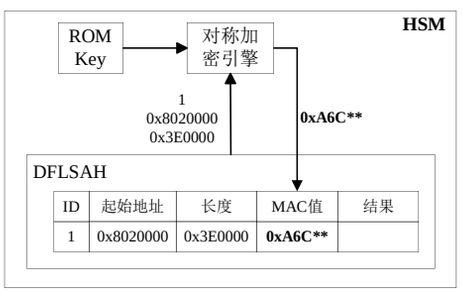

图7　MAC参考值生成流程

Fig.7　MAC reference value generation process

(1) MAC Table存放在HOST的PFLASH中,安全性相对较低,内容容易被篡改,进而导致软件无法正常启动,同时会占用部分HOST的PFLASH。

(2) 由于电脑端执行的脚本工具需要使用密钥执行CMAC算法获取MAC参考值,需将密钥Key存储在HSM的外部,增加了密钥丢失的风险。

(3) 硬件加速器存在一定的失效率,失效率一般要求在百万分之几,虽然概率很小,一旦由于硬件加速器失效导致的校验失败,也会导致系统启动失败,影响系统稳定性,也应该在软件中被合理解决。

(4) 控制器上电后从Boot ROM直接进入Bootloader,未验证Bootloader的完整性,Bootloader被恶意篡改无法被有效识别。本文不研究Boot ROM,因此图中未体现相关过程。

(5) 增加安全启动功能之后,完整性校验过程存在不可避免的耗时。以1.2中提到的SHE规范中要求128 kB代码校验耗时小于10 ms为例,图7中3968KB的APP代码校验耗时可高达310 ms,会导致控制器启动时间过长,无法满足要求。

2.3　安全启动方案二

针对方案一中存在的问题,提出了方案二。该方案将2.1节中提到的MAC参考值计算模块和完整性校验模块都放在HSM执行,跳转决策模块放在HOST执行。将整个安全启动功能的重要逻辑全部放在HSM端执行,HOST端仅需从HSM端获取校验结果,然后决定是否正常启动即可。MAC参考值计算模块分为两部分,如图6、图7所示。MAC参考值生成流程简述:

(1) 将APP代码段的起始地址和长度信息放在配置文件中。

(2) 在电脑端通过脚本将配置文件中代码段的起始地址和长度信息放在HSM DFLASH的MAC Table中。

(3) HSM基于控制器状态自动根据代码段的起始地址和长度信息计算其对应的MAC参考值,并将其对应存入MAC Table中作为参考值使用。

完整性校验和跳转决策分别在HSM和HOST端执行(见图8)。MAC验证与跳转决策流程简述:

(1) 在启动过程中,当需执行跳转时,HOST将需要进行完整性校验代码段对应的ID发送至HSM,请求HSM对其进行完整性校验。

(2) HSM基于HOST端所提供的ID信息,从MAC Table中查询对应的起始地址、长度和MAC参考值,然后调用硬件加速引擎使用内部密钥执行MAC校验,并将校验结果反馈至HOST端。

(3) HOST等待HSM验证完成后,获取验证结果,如果验证成功,则直接执行跳转,正常启动。如果验证失败,则请求HSM调用非对称加密算法执行二级认证。

(4) HOST等待HSM二级验证完成后,获取二级验证结果,如果验证成功,则直接执行跳转,正常启动。如果验证失败,停止启动,进入刷新模式。

上述二级认证是基于软件安全刷新使用的证书和签名机制实现,因此签名的计算和证书的配置已在软件安全刷新过程中实现,图6、图7中不再体现

该部分内容。

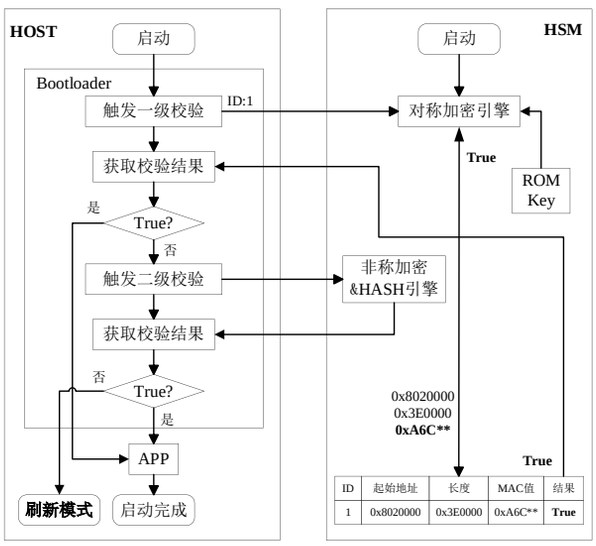

图8 MAC验证与跳转决策流程

Fig.8 MAC verification and jump process

方案二特点总结：

(1) 将MAC Table放在HSM的DFLASH中，增加了安全性，同时避免了对HOST PFLASH的占用。解决了方案一中的问题1。

(2) 密钥采用HSM中的ROM Key，该密钥在芯片下线时生成，且其值唯一，可实现“一机一密”，密钥仅能在HSM内部访问使用，HOST端无权获取，安全性极高。解决了方案一中的问题2。

(3) 采用了二级验证机制，有效避免了因HSM硬件加速器失效导致MCU无法启动的现象，增加了系统稳定性。解决了方案一中的问题3。

综上所述，方案二中仍未解决方案一中的问题4和问题5。针对问题4一般处理方式有两种：

(1) 在产品下线时，将Bootloader区设置为OTP模式，对应代码段不能被再次刷写。

(2) HSM先启动，HOST停留在Boot ROM阶段，HSM启动后校验HOST端的Bootloader，校验成功，则释放HOST跳转至Bootloader运行，否则，复位HOST，重复上述操作。

当前主流方案是方案一，方案二存在控制器“变砖”的风险，不太常见，本文不对此进行深入研究。

2.4 安全启动方案三

针对上述关于校验耗时，影响启动时间的问题，在方案二的基础上提出了HSM和HOST并行处理方案。1.2节中提到HSM具有独立的CPU，可以独立完成MAC校验工作，因此，在HSM执行MAC校验时，HOST端无需原地等待HSM执行MAC校验，直至校验完成。可将HOST端请求HSM执行MAC校验过程设计为异步操作，即HOST和HSM同时启动，待HSM启动完成后，HOST端则先向HSM发送MAC验证请求，请求发出后，HOST不做等待，继续执行后续操作，当要执行相应代码段时再向HSM请求校验结果。具体过程如图9所示。

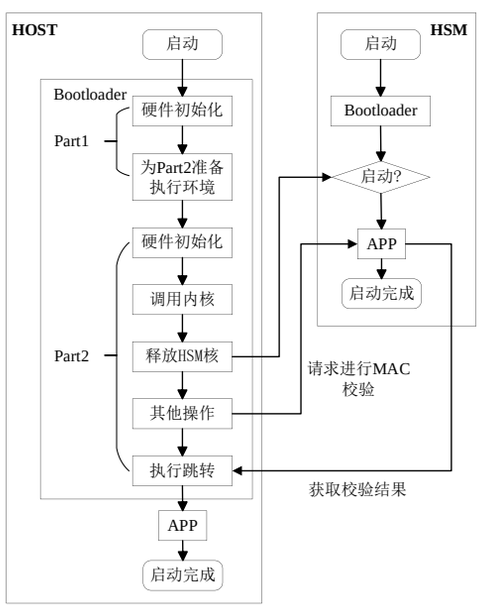

图9 并行启动流程

Fig.9 Parallel startup process

方案三描述：

(1) Bootloader分为两部分，Part1为汇编部分，Part2为C代码部分，Part1阶段主要实现硬件初始化，并为Part2阶段准备执行环境(RAM、堆栈)，环境准备完成后跳转至C语言入口。Part2阶段主要初始化该阶段使用的硬件，调用内核，启动操作系统，释放HSM核，判断执行刷新或启动流程，如果无刷新请求，则跳转至APP[9]。

(2) HOST和HSM同时启动，HSM启动过程较快，且由HOST释放，当HSM启动完成时，HOST还未执行到跳转步骤，因此，可在HSM启动完成后立即请求其执行MAC校验，当HOST执行到跳转步骤时，再去请求获取校验结果。将HSM执行MAC校验的时间和Bootloader启动流程中的部分过程耗时相抵消，一定程度上减小HOST的等待时间。该方案的优化性能取决于MCU软件启动流程中的跳数，跳数越多优化效果越明显。

方案三在一定程度上可以减小安全启动功能对控制器启动时间的影响，特别在多级启动案例中效果较为明显，但是针对文中介绍的两级启动案例，仅

采用并行处理机制其效果不太明显。

综上所述,以上三种主流的安全启动方案在安全性、稳定性和低耗时方面的性能表现对比如表1所示。方案二和三已经满足了安全性和稳定性的性能要求,但是在低耗时性能方面依然存在不足,需要进一步的合理优化。

表1　主流安全启动方案性能对比

Tab.1　Performance comparison of mainstream secure boot solutions

方案	安全性	稳定性	耗时性
一	低	低	低
二	高	高	低
三	高	高	中

3　安全启动方案优化与验证

3.1　安全启动方案优化

针对并行处理方案对两级启动案例提升效果不明显的问题,基于CMAC算法耗时与校验代码量成近似正比关系的特性,可以采用抽样校验的方式减小校验代码量来减小校验耗时。但为了保证安全启动功能的有效性,抽样比例又不能太小,一般抽样部分要占总大小的5%～10%[6],具体比例大小根据实际情况定义。MCU安全启动的主要目的就是保证重要的代码段未被恶意篡改,重要代码段主要包括核心的控制算法等。如何保证抽取的5%～10%的代码段包含所有重要代码成为方案优化的关键,基于编译原理,将重要代码编译到指定地址区域,然后在抽取校验代码段时覆盖该区域,即可保证在5%～10%抽取比例下,覆盖所有重要代码。

软件编译时由源文件到生成最终的可执行文件,需要经过一系列的操作[10],如图10所示。

预编译过程主要处理源代码文件中以“#”开始的预编译指令。编译器对预处理过的文件进行一系列的语法分析、词法分析、语义分析以及优化,生成汇编代码文件。汇编器将汇编代码转为目标文件,即机器可以执行的二进制指令。链接器则负责把多个目标文件,以及相关的库文件链接到一起,形成一个最终的可执行文件。可执行文件通常由代码段(.text),数据段(.data)和bss段构成。

链接的过程决定了数据和指令标签的地址,而且链接过程受链接脚本的控制,可以编写链接脚本在整个链接过程中提供显式的、全局的控制。因此,可通过修改链接脚本,实现将重要代码编译到指定区域的功能。具体方法如下:

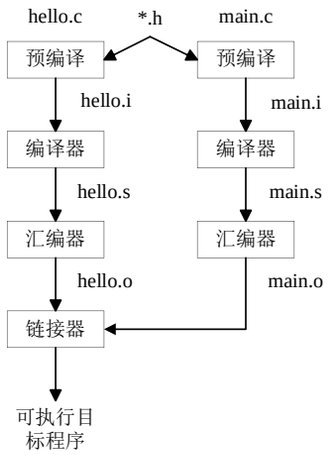

图10　编译过程

Fig.10　Compilation process

(1) 修改链接器脚本文件,以MCU的总PFlash 4M大小为例,对4M PFlash进行划分,128KB的Bootloader段,用于存放Bootloader代码。400KB的“important_code”段,用于存放重要代码,大小满足5%～10%的抽样比例要求。剩余部分划分为“other_code”段,用于存放其他非重要代码。如图11所示。

SECTIONS命令是链接脚本的主题,指定了各种输入段到输出段的变换。大括号中包含了SECTIONS的变换规则,以“important_code”段为例:.=0x8020000为赋值语句,将当前的地址设置为0x8020000, important_code:{*(.text)}为转换规则,将目标文件中.text的段合并到可执行文件的“important_code”段中。

(2) 定义宏。结合_attribute_在代码中定义指定段,#define IMPORTANTCODE_attribute_((section(“important_code”))),#define OTERHCODE _attribute_((section(“other_code”)))。

(2) 在代码开发阶段识别出重要算法函数,通过宏IMPORTANTCODE将其标记为重要代码,通过宏OTERHCODE将其标记为非重要代码,在执行链接操作时链接器将它们链接到各自指定区域中去,如图12所示。

(3) 将“important_code”段的起始地址和长度配置到安全启动方案中的CMAC Table中,在Bootloader跳转至APP时,对其进行消息验证码校验,校验成功则执行跳转,否则,停止正常启动,进入刷新模式。

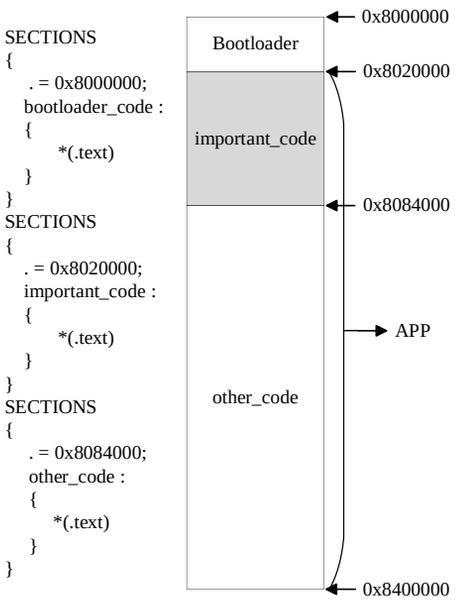

图11 链接脚本

Fig.11 Linker Script

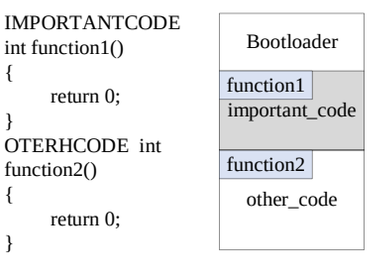

图12 编译执行结果

Fig.12 Compile result

3.2 优化方案实物验证

为了验证3.1节中优化方案的可行性,基于NXP LPC4078微控制器(ARM Cortex M4 with FPU, 120 MHz),借助GNU ARM工具链,通过修改对应的链接脚本、编写测试代码、执行编译,进行了简单的可行性测试。实物验证系统如图13所示。

(1)链接脚本修改,修改内容如图14所示。先将PFLASH划分为ROM1和ROM2两部分,然后通过SECTIONS命令定义“important_code”和“other_code”两个代码段,并分别将ROM1和ROM2两部分PFLASH区域分配给代码段“important_code”和“other_code”。

图13 实物验证系统

Fig.13 Physical verification system

```
/* Entrypoint */
ENTRY(_boot)
/* Memory Definitions */
MEMORY
{
 ROM1 (rx) : ORIGIN = 0x00002000, LENGTH = 0x00030000
 ROM2 (rx) : ORIGIN = 0x00030000, LENGTH = 0x0003C000
 RAM (rw) : ORIGIN = 0x40000000, LENGTH = 0x00004000
}
/* Section Definitions */
SECTIONS
{
 /* section is .important_code which is used for important code */
 .important_code :
 {
        *(.text)        /* remaining code */
 } > ROM1
 . = ALIGN(4);
}
SECTIONS
{
 /* section is .other_code which is used for other code */
 .other_code :
 {
        *(.text .text.*)        /* remaining code */
 } > ROM2
 . = ALIGN(4);
}
```

图14 链接脚本

Fig.14 Linker Script

(2)编写测试代码,如图15所示。依据3.1节中的描述定义相应的宏和函数,function1被指定为重要代码,function2被指定为非重要代码。

(3)执行编译,在map文件中查看function1和function2被编译的位置,如图16所示。

测试结果显示,function1被正确编译到了“important_code”代码段的地址空间,function2被正确编译到了“other_code”代码段的地址空间。

上述过程通过编译优化操作将重要的代码编译到了指定区域内,验证了优化方案的可行性。下面基于实物系统对优化方案的有效性进行了进一步验证。大致过程:依据实际情况在工程代码中定义重要代码并执行交叉编译,统计编译优化前后重要代码覆盖率;然后使用调试器将优化前后的代码依次刷入开发板,测试优化前后的安全启动耗时。性能对比测试流程如图17所示。

重要代码覆盖率基于编译生成的map文件手工

```
#define IMPORTANTCODE
__attribute__((section(".important_code")))
#define OTHERCODE __attribute__((section(".other_code")))
IMPORTANTCODE int function1()
{
        int result = 1;
        return result;
}
OTHERCODE int function2()
{
        int result = 1;
        return result;
}
int  main ( void)
{
        function1();
        function2();
        return 0;
}
```

图15　测试代码

Fig.15　Test code

```
.important_code
        0x00003580      0x8 ./obj/main.o
        0x00003580          function1

.other_code
        0x00030008      0x8 ./obj/main.o
        0x00030008          function2
```

图16　测试结果

Fig.16　Test Result

计算获取,MAC校验耗时则使用调试软件分别在执行MAC校验操作的前后打断点测量获取。最终的测试结果如表2所示。

表2　方案优化前后耗时性能对比

Tab.2　Comparison of time-consuming performance before and after solution optimization

方案	校验比例/%	重要代码覆盖率/%	校验耗时/ms
未优化	100	100	39.4
抽样校验	5	6.12	1.8
抽样校验+编译优化	5	100	1.8

测试结果分析:

(1) 加入抽样校验后,校验耗时约为未抽样前的1/21.9,主要原因是抽样校验将校验区域减小了20倍,而校验耗时与校验区域的大小成近似正比关系。

(2) 仅加入抽样校验后,重要代码覆盖率仅有6.1%,导致安全启动功能无法有效的保护重要代码。主要原因是编译时重要代码随机的散落在整个代码区域的不同位置,而抽样时仅抽取5.0%,从而

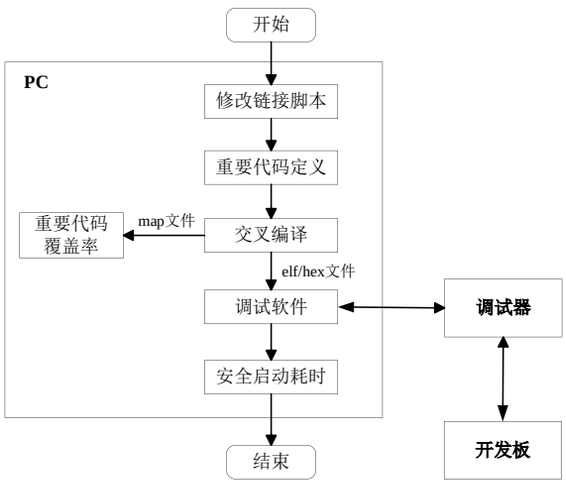

图17　性能测试过程

Fig.17　Performance Testing Process

导致抽取的5%代码中仅包含重要代码的6.1%,此值的大小较为随机,但一般和抽样比例大小近似。

(3) 加入抽样校验和编译优化后,校验耗时约为未抽样前的1/21.9,原因同上。而重要代码覆盖率则可达到100%,主要原因是通过编译优化提前将重要代码集中编译到要抽取的5.0%的代码区域中,从而保证了在满足安全启动耗时要求的前提下覆盖尽可能多的重要代码。

加入抽样校验和编译优化后的重要代码覆盖率取决于定义的重要代码量和抽样比例的关系,当定义的重要代码量比例(例如为4.0%)小于抽样比例(例如为5.0%)时,覆盖率可达100%。而当定义的重要代码量比例(例如为6.0%)大于抽样比例(例如为5.0%)时,覆盖率为抽样比例与重要代码比例之比(83.3%)。抽样比例和重要代码比例需基于实际场景进行合理设计。

4　结语

本文首先从车辆网络安全法规要求出发,分析了车载MCU实施安全启动功能来缓解恶意软件攻击的必要性。然后结合车载场景下对安全启动功能在安全性、稳定性和低延时三方面性能的严格要求,分析了当前主流的安全启动方案在这三个性能方面的表现。分析后发现当前主流方案在安全性方面通过使用HSM执行密码学算法和存储密钥得以保证;稳定性性能通过HSM硬件加速器的可靠性和二级验证策略保证;而低耗时性能方面虽然也存在HSM和HOST并行处理方案,但是该方案的性能提升依赖于启动跳数,启动跳数越多,优化效果越明显,但

对于常见的两级启动来说,仅存在一跳,优化效果不明显。由于CMAC算法的执行耗时与数据量成近似正比关系,而且对于MCU来说,仅需保证重要代码部分不被篡改即可,没必要全部校验,可以采用抽样校验的方案来优化低耗时性能。而如何保证在5%~10%抽样比例的情况下覆盖所有重要代码是一个亟需解决的问题,本文提出了基于编译原理将重要代码生成在提前划分好的代码区域的方案,并在LPC 4078微控制器上通过更改链接脚本验证了方案的可行性。

在实际使用过程中,重要代码段的制定尤为重要,其大小直接决定了重要代码覆盖率和启动耗时。重要代码段制定的大致步骤如下:① 测量未加入安全启动功能时的控制器启动时间,结合启动时间要求,计算出所允许的最大安全启动功能耗时。② 基于最大安全启动功能耗时、CMAC算法执行效率和代码总量推算出最大的抽样代码比例x%。③ 识别工程中所有需要保护的重要代码,计算重要代码占代码总量比例y%。④ 最后,基于上述结果制定重要代码段的大小,使$y \leqslant x$;当$y > x$时,需对重要代码进行分级,将更重要的代码放入校验区域,保证满足安全启动耗时要求的前提下覆盖尽可能多的重要代码。重要代码段的制定过程比较繁琐,且灵活度较高,取决于代码总量、CMAC算法执行效率和允许的最大安全启动功能耗时等因素。本文给出了一个通用的制定思路,在实际项目中还需结合具体情况进行相应的调整。

参考文献:

[1] United Nations Economic Commission for Europe (UNECE). Proposal for a new UN regulation on uniform provisions concerning the approval of vehicles with regards to automated lane keeping system: GRVA-05-07-Rev.3[S/OL].(2020-02-15)[2023-09-01]. https://unece.org/DAM/trans/doc/2020/wp29grva/GRVA-05-07r3e.pdf.

[2] UN Regulation No 156: Uniform provisions concerning the approval of vehicles with regards to software update and software updates management system [2021/388][S].(2021-03-09)[2023-09-01]. http://data.europa.eu/eli/reg/2021/388/oj.

[3] DOBAJ J, EKERT D, STOLFA J, *et al*. Cybersecurity threat analysis, risk assessment and design patterns for automotive networked embedded systems: a case study[J]. JUCS: Journal of Universal Computer Science, 2021, 27(8): 830.

[4] PROFENTZAS C, GÜNES M, NIKOLAKOPOULOS Y, *et al*. Performance of secure boot in embedded systems[C]// 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS). Santorini, Greece: IEEE, 2019: 198. doi: 10.1109/DCOSS.2019.00054.

[5] AES. Advanced Encryption Standard: FIPS-197[S]. NIST, 2001.

[6] ESCHERICH R, LEDENDECKER I, SCHMAL C, *et al*. SHE: Secure hardware extension-functional specification (version 1.1)[J]. Hersteller Initiative Software (HIS) AK Security, 2009.

[7] HENNIGER O. EVITA: E-safety vehicle intrusion protected applications[R]. Brussels: EVITA, 2011. http://www.evita-project.org/EVITA_factsheet.pdf.

[8] STUMPF C P F, POHL C. An analysis and comparison of hardware security modules for the automotive domain[C]// Proc Embedded Security in Cars (ESCAR) USA Conf. 2014.

[9] SHA C, LIN Z Y. Design optimization and implementation of bootloader in embedded system development [C]//2015 International Conference on Computer Science and Applications (CSA). IEEE, 2015: 151.

[10] 陈意云,张昱. 编译原理[M]. 北京: 高等教育出版社, 2003.
CHEN Yiyun, ZHANG Yu. Principles of compilation[M]. Beijing: Higher Education Press, 2003.