北京大学学报(自然科学版) 第 58 卷 第 6 期 2022 年 11 月
Acta Scientiarum Naturalium Universitatis Pekinensis, Vol. 58, No. 6 (Nov. 2022)
doi: 10.13209/j.0479-8023.2022.095

基于iRAM的抗物理内存泄露攻击密码算法轻量化实现

李彦初[1] 荆继武[2,3] 雷灵光[4,5,†] 王跃武[4,5] 王平建[4,5]

1. 中国科学院大学计算机科学与技术学院, 北京 100049; 2. 中国科学院大学密码学院, 北京 100049; 3. 北京大学软件与微电子学院, 北京 100871; 4. 中国科学院信息工程研究所信息安全国家重点实验室, 北京 100093; 5. 中国科学院大学网络空间安全学院, 北京 100049; † 通信作者, E-mail: leilingguang@iie.ac.cn

摘要 提出一种基于iRAM的轻量安全密码算法实现方案, 可以在不影响系统中需要iRAM辅助的正常功能情况下, 实现多个密码算法的并发执行。该方案将密码算法实现中的敏感数据限制在单个可加载段中, 同时分离该段中的非敏感数据, 并通过修改可信应用的加载方式、仅将包含敏感数据的段分配到iRAM空间等方法, 尽量减少密码运算对iRAM的占用。在真实设备上实现国内外具有代表性的一系列密码算法, 实验结果表明, 所有算法的性能开销均小于4.3%, iRAM使用量皆少于4.5 KB, 比现有方案节省78%以上, 能够支持方案在所有主流平台上部署。

关键词 密码算法; TrustZone; iRAM; 物理内存泄露攻击

An iRAM-based Light-Weight Cryptographic Algorithm Implementation Scheme against Physical Memory Disclosure Attacks

LI Yanchu[1], JING Jiwu[2,3], LEI Lingguang[4,5,†], WANG Yuewu[4,5], WANG Pingjian[4,5]

1. School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 100049; 2. School of Cryptography, University of Chinese Academy of Science, Beijing 100049; 3. School of Software and Microelectronics, Peking University, Beijing 100871; 4. State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing 100093; 5. School of Cyber Security, University of Chinese Academy of Science, Beijing 100049; † Corresponding author, E-mail: leilingguang@iie.ac.cn

Abstract An iRAM-based light-weight secure cryptographic algorithm implementation scheme is proposed, which can execute multiple cryptographic algorithms concurrently without affecting the iRAM-assisted functions of the system. The scheme restricts the sensitive data in the cryptographic algorithm implementation to a single loadable segment, separates the non-sensitive data from this segment, and modifies the loading procedure of the trusted applications to allocate only the segment containing sensitive data to the iRAM space. It can minimize the occupation of iRAM by cryptographic operations. A series of representative cryptographic algorithms are implemented on the real device. The experimental results show that the performance overhead of all cryptographic algorithms is less than 4.3%, and each algorithm's demand for iRAM is less than 4.5 KB, saving more than 78% compared with existing schemes, which supports the deployment of the scheme on all mainstream platforms.

Key words cryptographic algorithm; TrustZone; iRAM; physical memory disclosure attacks

ARM平台广泛应用于个人移动终端、物联网和工业控制系统等场景[1]。相对于桌面系统或服务器, 移动终端设备更容易丢失或被窃取, 而物联网和工业控制系统经常需要在无人值守的非受控场景下工作, 因此这类平台的安全机制更容易遭受物理内存泄露攻击。因此, 如何构建ARM平台抗物理

国家自然科学基金(61802398)资助
收稿日期: 2022−01−10; 修回日期: 2022−03−22

内存泄露攻击的密码机制成为目前系统安全研究的热点。

物理内存泄露攻击的根源在于普通的DRAM (dynamic random access memory)内存芯片独立于ARM的SoC(system on a chip)芯片, 需要通过内存总线与SoC芯片进行双向数据传输。当数据存储在内存中时, 攻击者可通过如下3种方式发起物理内存泄露攻击。1) 冷启动攻击[2–3]: 利用内存的剩磁效应, 将正在运行的设备断电, 然后将内存接入攻击者控制的系统中, 读取其中残留的数据。2) 总线监听攻击[4–5]: 当内存芯片与SoC芯片通过内存总线传输数据时, 攻击者通过监听该总线, 窃取内存中的敏感数据, 相关的攻击工具[6–7]可以很容易地获取。3) DMA (direct memory access)攻击[8–10]: 攻击者控制特定I/O设备并发起DMA请求, 在处理器不参与的情况下, 直接从内存中拷贝敏感数据。因此, 避免直接在内存中存储敏感计算输入、输出和中间结果数据成为当前抗物理内存泄露攻击的主要技术途径。

现阶段应对物理内存泄露攻击的技术途径主要包括以Cache(高速缓存)、寄存器和iRAM为基础的3种方案。

Cache方案[11–13] 基于Cache的敏感计算抗物理攻击方法存在几个问题: 1) 对系统性能影响比较大, 比如, 60%的Cache被占用后, 系统性能急剧降低[13]; 2) 难以进行软件编程调度, 操控性差; 3) 无法与I/O设备直接进行数据交互。

寄存器方案[14–15] 通过引入安全机制, 确保敏感数据在计算过程中只存储在SoC芯片内的寄存器中, 不出现在内存中, 但因空间有限面临性能挑战, 此外, 不同SoC平台上的可用寄存器类型存在较大的差异, 方案的可移植性较差。

iRAM方案[16–19] 将敏感计算结果及中间值存储在iRAM中, 实现抗物理泄露内存攻击。不同于SoC外部的DRAM内存, iRAM剩磁效应小, 并且在SoC芯片内部, 与CPU单元通过片内总线连接, 具有良好的抗物理内存泄露攻击特性。此外, iRAM寻址方式与DRAM一致, 更易操控, 并能够直接与I/O设备交互。iRAM方案工程实现优势明显。

轻量化实现是目前iRAM方案面临的主要技术挑战。移动终端上的大部分iRAM资源会被操作系统使用, 只有少量空闲的iRAM。比如, FreeScale i.MX6Quad系列SoC的空闲iRAM只有32 KB。然而, 当前方案[16–19]使用的iRAM空间都超过了系统空闲iRAM的大小, 从而会影响普通世界操作系统的正常功能。因此, 亟需一个轻量高效的密码算法来实现方案, 以便降低iRAM资源消耗。

本文通过对密码运算过程中敏感数据的细粒度划分, 尽量缩小保护数据规模, 实现iRAM资源消耗的减少。首先, 通过修改系统应用加载器, 将程序栈段加载到iRAM; 然后, 通过密码算法计算过程分析, 标识出密码算法敏感变量的最小集合; 随后, 重新设计密码算法实现程序, 将敏感数据向程序栈段集中, 以便简化iRAM调度; 最后, 必要时通过辅助可信应用引入, 将栈段中的非敏感数据移除, 进一步降低iRAM消耗。此外, 该方案通过ARM TrustZone实现iRAM的抗软件攻击保护。本研究在FreeScale的i.MX6Quad开发板上对SM2/3/4以及RSA, AES, SHA3等主流的密码算法进行原型系统实现, 并开展方案验证评估。

1 相关工作

研究人员提出一些针对物理内存泄露攻击的防御方案, 主要针对冷启动攻击[2–3]以及总线监听攻击[4–5]。这些方案通过构建以SoC为边界的执行环境, 将敏感数据保存在SoC内部, 有效地阻止攻击。例如, TRESOR[14]将密钥存储在x86调试寄存器中, 并利用Intel的AES-NI扩展指令, 在微处理器中运行AES算法, 避免DRAM的使用, 从而阻止物理攻击。PRIME[15]将RSA算法中的私钥操作限制在Intel多媒体寄存器(共16个256 bit的寄存器)中, 保护了RSA-2048算法。Copker[11]将密码运算的密钥和所有中间结果保存在Cache中, 确保敏感信息不会进入DRAM, 实现抗物理攻击的AES算法和RSA算法。Sentry[16]和CryptMe[17]借助iRAM来保护AES算法中的敏感数据。

另外一种物理内存泄露攻击是DMA攻击[8–10], 可利用系统的安全机制(如TrustZone的DMA隔离)来阻止。由于安全机制的引入也可同时阻止软件攻击, 所以综合考虑上述3种物理内存泄漏攻击和软件攻击, 研究人员提出了相关的敏感数据保护方案。CaSE[13]利用Cache和ARM TrustZone技术[20]构建以SoC为边界的可信执行环境。Oath机制[19]、OP-TEE的pager机制[21]和m-TEE[18]都是利用iRAM和TrustZone机制保护敏感数据, pager机制和m-TEE方案分别需要180 KB和100 KB的iRAM空间;

Oath 通过分离敏感数据和非敏感数据，减少了对 iRAM 的占用，但所测试的 12 个可信应用仍需使用平均 50.52 KB 的 iRAM 空间。本研究支持在系统空闲的 iRAM 上同时运行多个密码算法，其中单个算法对 iRAM 的使用量不超过 4.5 KB。

内存受限场景下的密码算法实现需要同时考虑密码运算效率以及对内存的使用量。研究人员提出许多减少内存消耗且高效的密码运算或密码算法的实现方案，包括对称密码算法[22]、基于椭圆曲线的非对称密码算法[23−25]和格密码算法[26]等。本研究在 iRAM 内存受限的条件下，实现多种密码算法。首先，本文使用受限 iRAM 内存实现抗物理内存泄露攻击的密码算法，而上述工作聚焦于密码算法在减少内存使用同时的高效实现。其次，在相同的实现方式下，减少内存占用会导致性能的下降，所以上述工作通过特殊处理器指令和硬件扩展等方式同时实现高效和轻量化两个目标，本文仅依赖 ARM 平台上广泛部署的 TrustZone 安全扩展和通用组件 iRAM 来保护敏感数据，方案可移植性更好。此外，本文主要降低 iRAM 内存的使用量，而算法实际的内存使用量(iRAM+DRAM)并未减少，所以与未修改的实现相比，密码算法性能没有明显降低。

2 背景知识与相关工作

2.1 内部随机存取存储器(iRAM)

内部随机存取存储器(internal-RAM，iRAM)是一种片上随机存取存储器(on-chip random access memory，OCRAM)，是通用计算平台(例如 ARM 处理器)中广泛存在的组件，比普通的 DRAM 读写速度更快。由于价格限制，iRAM 在终端上的容量一般在几十 KB 到几百 KB 之间，主要用于存储系统中的特定信息(比如，暂停或恢复系统的相关代码)以及加速视频播放等多媒体处理过程。

2.2 OP-TEE 架构

OP-TEE(open portable trusted execution environment)[27]是一个开源 TEE OS，基于 TrustZone[20]运行。TrustZone 是 ARM 公司推出的安全扩展，将 SoC 上的中央处理器、内存和外设等资源划分到普通世界或安全世界，并基于硬件逻辑的访问控制实现普通世界与安全世界的隔离。OP-TEE 架构如图 1 所示，安全世界中运行用户态的若干可信应用(trusted application, TA)以及内核态的安全操作系统(OP-TEE OS)；普通世界中运行非敏感代码，包括

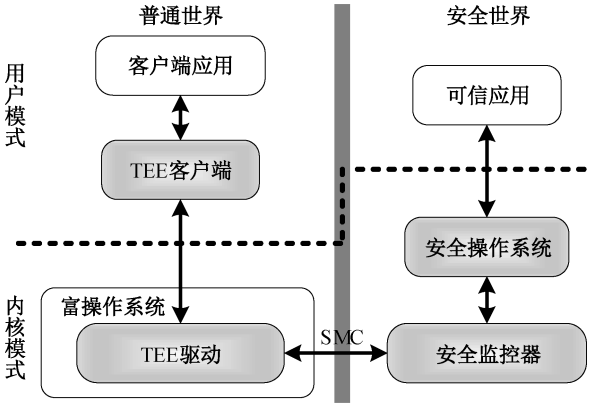

图 1　OP-TEE 系统架构

Fig. 1　Architecture of OP-TEE

用户态的客户端应用(client application, CA)以及内核态的富操作系统(rich OS)。客户端应用通过用户态的 TEE 客户端(TEE client)来调用可信应用中的敏感功能。调用过程中，富操作系统中的 TEE 驱动(TEE Driver)通过 SMC(secure monitor call)指令跳转到安全世界中的安全监控器模块(Secure Monitor)，从而与安全世界进行交互。

2.3 可信应用(TA)的运行时数据段分配

OP-TEE 架构中，安全世界的可信应用以 ELF 格式的形式存储于文件系统，其代码及代码中的各类变量存放在 ELF 文件的不同可加载段。当 OP-TEE OS 加载可信应用时，会给每个可加载段分配一段内存空间，并将相应数据从可执行文件拷贝到对应的内存空间中。之后，可信应用运行时，各类变量的值(包括敏感数据及非敏感数据)就存储在变量所在的可加载段的对应内存空间中。从总体上看，可信应用的可加载段包括.ta_head 段、.text 段、.rodata 段、.data 段、.bss 段(可信应用从.bss 段划分堆空间，所以堆变量也存放在.bss 段)和栈段等。其中，.ta_head 段存放可信应用的基本信息(可信应用的唯一标识符、栈段大小和入口函数地址等)，.text 段存放可信应用的代码，.rodata 段存放常量等信息，.data 段存放已初始化的全局变量和局部静态变量，.bss 段存放未初始化或初始化为 0 的全局变量、局部静态变量和由程序员分配/释放的堆变量，栈段存放由编译器自动分配/释放的栈变量。本文通过将敏感数据集中在一个段中，并为该段分配受 TrustZone 保护的 iRAM 来实现敏感数据泄露保护。

算法 1 的代码示例展示不同变量的定义方式及其对应的数据段。

算法 1 一个可信应用代码中不同变量的定义及其对应的数据段

```
1   #define SM2_SIGN_RESULT_LENGTH   64                                  /* 位于.rodata 段 */
2   int var_global_bss = 0;                                              /* 位于.bss 段 */
3   unsigned char var_global_data [3] = {0x47, 0x4C, 0x4F}               /* 位于.data 段 */
4   void func(TEE_Param params [4]){ /* 可信应用参数位于栈段; 参数指向的缓冲区位于非安全 DRAM 的共享内存 */
5   char var_stack [32];                                                 /* 位于栈段 */
6       static int var_static_bss = 0;                                   /* 位于.bss 段 */
7       static unsigned char var_static_data [3] = {0x73, 0x74, 0x61}    /* 位于.data 段 */
8       char *var_heap = (char *) malloc (32 * sizeof(char) );           /* 申请位于.bss 段的堆空间 */
9       ...;                                                             /* 函数的具体功能代码 */
10      Free (var_heap);                                                 /* 释放位于.bss 段的堆空间 */
11      ...;}
```

其中, 第1行是存放在.rodata段的常量; 第2行和第6行是.bss段的全局变量和局部静态变量; 第3行和第7行是.data段中的全局变量和局部静态变量; 第4行的函数参数以及第5行的局部变量存放在栈段, 但客户端应用传递过来的缓冲区形式的参数指向位于非安全DRAM的共享内存中的一块缓冲区; 第8行动态地申请一块堆空间, 并令一个存放在栈中的局部变量(指针)指向这块空间, 堆空间中的数据存放在.bss段, 第10行手动释放了堆空间。

3 威胁模型与安全假设

假设敌手可以发起软件攻击和物理内存泄露攻击。1) 软件攻击: 利用各种系统漏洞攻击控制普通世界的操作系统, 进而无限制地访问普通世界的资源, 包括寄存器、DRAM、iRAM、Cache以及其他设备, 但软件攻击无法破坏由TrustZone硬件隔离保护的安全资源。2) 物理内存泄露攻击: 可以物理访问被保护设备, 并通过冷启动攻击[2–3]、监听SoC外部总线的总线监听攻击[4–5]以及DMA攻击[8–10], 窃取DRAM中的敏感数据。

本文假定物理内存泄漏攻击无法攻击由TrustZone保护的安全iRAM。1) 系统上电后, 引导时执行的第一段代码会将iRAM中的内容清除, 而该代码不能被绕过[16]; 并且iRAM及其使用的高级可扩展接口(advanced extensible interface, AXI)总线都在SoC内部, 无法被取出, 所以冷启动攻击和总线监听攻击无法窃取iRAM中的敏感数据。2) TrustZone默认普通世界向安全世界发起的DMA请求被拒绝, 很多广泛使用的SoC中的DMA控制器[28–29]还可以拒绝特定DMA通道以及DMA设备发起的从安全世界传输数据的请求。所以DMA攻击无法窃取被TrustZone保护的安全iRAM中的敏感数据。进行密码运算的可信应用运行在安全世界的TEE OS之上, 侧信道攻击特征不明显。

本研究旨在保护密码算法运行时敏感数据的机密性。我们假设目标平台支持并正确实现了TrustZone技术; 安全世界的操作系统和可信应用不是恶意的, 没有敌手可利用的漏洞, 并且被平台可信启动技术安全加载。

4 方案设计

4.1 方案架构与内存布局

本文提出的方案利用iRAM存储密码运算过程中的敏感数据, 以便抵御物理内存泄漏攻击, 并尽量减少对iRAM的占用, 保证普通世界的功能不受影响。将密码算法中的敏感数据限制在栈段中, 通过修改可信应用的加载方式, 仅给栈段分配安全iRAM, 给其他可加载段分配安全DRAM, 从而减少iRAM的使用量。此外, 方案还将栈段中的非敏感数据分离到其他可加载段, 进一步减少iRAM的使用量。方案的基本架构和内存布局见图2。

方案涉及的内存包括iRAM和DRAM, 利用ARM TrustZone技术分别将其划分为安全模式和非安全模式。密码运算由在安全世界的可信应用完成。可信应用从非安全DRAM的共享内存获得运行在普通世界的客户端应用的输入参数, 执行密码运算, 并返回计算结果。方案通过修订密码算法实现, 将敏感数据全部集中在栈段。随后, 进一步修改可信应用加载程序, 将可信应用中除栈段外的其他可加载段全部放在安全DRAM中, 同时将栈段分配在安全iRAM中。此外, 栈段还包含密码算法实

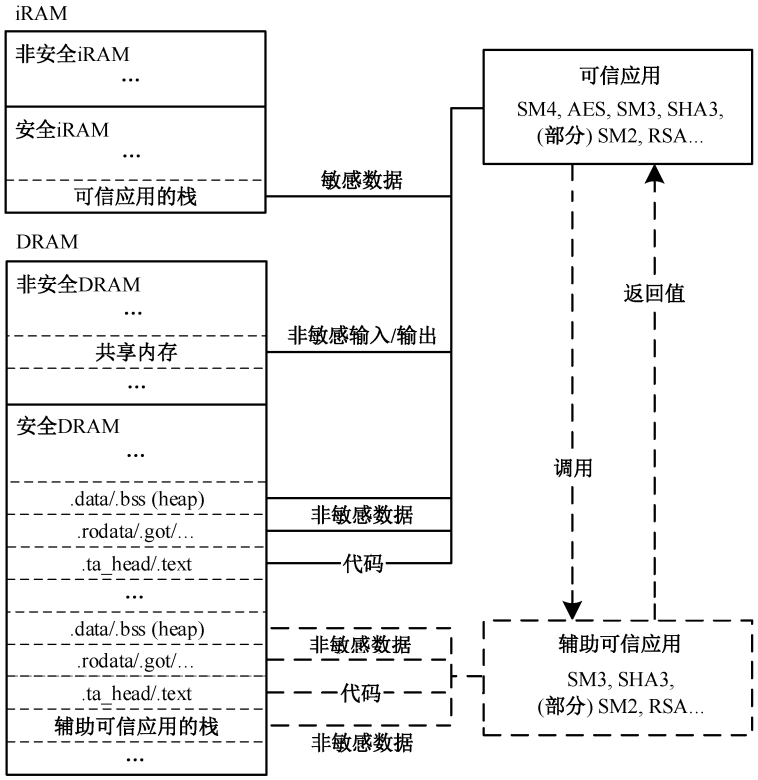

图 2　实现密码算法的可信应用及其辅助可信应用的代码和数据在内存中的位置

Fig. 2　Memory location of the TA and its assisted TA's code and data

现中的非敏感数据，为了进一步压缩 iRAM 使用量，必要时引入辅助可信应用，将栈段的非敏感数据转移到其他可加载段和辅助可信应用的可加载段。辅助可信应用的所有可加载段分配在安全 DRAM 中。这样，敏感数据仅存在于安全 iRAM 中，能够同时抵御物理内存泄露攻击和软件攻击。密码算法的其他数据分配至安全 DRAM 中，由 ARM TrustZone 隔离机制构建与普通世界的边界，抵御软件攻击。

4.2 集中敏感数据到栈段

数字签名算法的敏感数据包括私钥，非对称加密算法的敏感数据有私钥和明文，密码杂凑算法的输入可能为敏感数据，对称加密算法的敏感数据包括对称密钥和明文。此外，算法计算过程中的一些数据可以用于恢复上述敏感数据或增加恢复的可能性，也应当作为敏感数据加以保护。具体地，SM2 数字签名算法的签名结果为(r, s)，计算s的方式为$s=(1+d_A)^{-1}(k-r\times d_A) \bmod n$，其中$d_A$为私钥，$k$为用于本次签名运算产生的随机数，$n$为椭圆曲线基点$G$的阶。由于$s$，$r$和$n$公开，因而能获取$k$的攻击者可以通过上述公式来求解出私钥$d_A$，所以$k$也应该属于敏感数据。当密码杂凑算法的输入为敏感数据时，中间计算结果可用于降低恢复敏感输入的难度，所以也属于敏感数据。对称加密算法中扩展出的轮密钥直接用来加密明文，每轮中间结果的泄露相当于计算轮数的降低，所以二者都属于敏感数据。最后，在密码算法的实现中，与敏感数据有关的中间计算结果也可能属于敏感数据，比如 SM2 数字签名算法中的$(1+d_A)^{-1}$和$(k-r\times d_A) \bmod n$等，因为他们的泄露会极大地降低获取算法中敏感数据的难度。

如 2.3 节所述，密码算法实现代码可能包含多个数据段，上述敏感数据可能存在于各种不同的段(包括存放未初始化的全局变量/静态变量的.bss 段、存放栈变量的栈段以及存放堆变量的堆段)中。为了抵御物理内存泄露攻击，需要将敏感数据存储在 iRAM 中。如果不对算法实现做修订，则需要将上述数据段都加载到安全 iRAM 中。但这几类数据段同时还存放大量非敏感数据，比如未初始化的全局变量/静态变量中可能会存放一些密码算法中公开且固定的参数。这样，大量非敏感数据会造成不必要的 iRAM 消耗。栈段是密码运算过程中间

结果存储的不可或缺的数据段。因此，本文通过密码算法实现代码修订，将所有敏感参数集中到栈段。这样，就可以只给栈段分配安全iRAM空间，其他数据段依然使用安全DRAM，从而减少iRAM消耗。

4.3 分离栈段非敏感数据到安全 DRAM

敏感数据和非敏感数据处理都需要用到栈段，因此，除敏感数据外，栈段中还存在大量的非敏感数据。本文方案通过分离栈中的非敏感数据，进一步减少安全iRAM的使用量，具体包括以下两项内容。

4.3.1 变量分离

密码算法中的非敏感数据可分为如下3类：1)取值固定且公开的系统参数，如SM2算法中的椭圆曲线参数和AES算法中的S盒等；2)非敏感输入/输出参数，包括签名算法中的待签名消息、签名值、公钥以及其他公开信息(比如SM2数字签名算法中的用户可辨别标识)，密码杂凑算法中的杂凑值以及非敏感的输入，加密算法中的密文等；3)密码算法的实现中，存在一些与敏感数据无关的临时变量，这些变量无法用于恢复敏感数据或增加恢复的可能性，将这类临时变量中的数据也视为非敏感数据。

如图2所示，我们将第1类非敏感数据存储在全局变量和局部静态变量(位于可信应用ELF文件中的.bss段或.data段)中，将第2类非敏感数据存储在客户端应用传递过来的共享内存缓冲区(位于非安全DRAM)，将第3类非敏感数据存储在堆中的临时变量(位于.bss段)中。.data段和.bss段被分配的是安全DRAM空间。

4.3.2 运算步骤分离

密码算法中可能存在一些全程没有敏感数据参与的步骤，比如数字签名算法中计算待签名消息的杂凑值过程、数字签名验证的全过程以及当输入为非敏感数据时的密码杂凑算法的全过程等。针对这些运算步骤中的非敏感数据，一种分离方法是按照上述变量分离的方法修改密码算法的实现代码。但是，密码运算过程中涉及很多临时变量(比如SM2算法中椭圆曲线上点的运算)，手动给这些变量分配/释放堆空间对算法实现的改动很多，工作量大且容易出错。此外，密码算法实现中涉及一些生命周期很短、被频繁定义的临时变量(比如SM3算法中被频繁调用的压缩函数中定义的临时变量)，由于堆空间的申请和释放是程序内的代码通过系统调用来实现，所以频繁的申请/释放操作会引入过多的用户模式–内核模式切换，导致性能开销过大。因此，我们采用图2所示的计算分离方法：在一个辅助可信应用中实现这些步骤，通过OP-TEE提供的进程间通信(inter-process communi-cation, IPC)接口与可信应用进行通信，该辅助可信应用的所有数据都存储在安全DRAM。

考虑到很多密码算法的实现都将第1类非敏感数据存储在全局变量中，而这部分数据也占用一定的空间，第2类非敏感数据的分离也符合可信应用的编程习惯，为兼顾方案的实用性和通用性，本文方案中对存放第1类和第2类非敏感数据的变量分离是必选项；对存放第3类非敏感数据的变量分离以及部分运算步骤的分离这两部分内容，对原始密码算法实现有较明显的改动，将其作为可选项。

5 方案实现

我们基于FreeScale i.MX6Quad SABRE开发板实现方案原型系统。安全世界和普通世界中的操作系统分别为OP-TEE 2.2.0和Android 6.0.1，Android系统基于Linux 4.1.15内核。由于方案仅涉及对可信应用加载过程以及对密码算法实现方式的修改，与操作系统的版本无关，因此具有很好的系统兼容性。虽然选择在i.MX6Quad开发板实现原型系统，但所依赖的TrustZone机制和iRAM 都是在终端平台较为普遍部署的硬件，因此可以方便地移植到其他ARM硬件平台。

5.1 构建iRAM辅助的可信应用执行环境

为可信应用的栈段分配安全iRAM空间。OP-TEE OS加载可信应用时，在其上下文结构体中记录了4块物理内存空间的信息(分别存储可信应用的栈段、.ta_head/.text段、.rodata/.got/...段以及.data/.bss段中的数据)，包括物理地址、虚拟地址、段大小和段标识等。OP-TEE OS首先分配4块物理内存空间并填充结构体中的信息，然后依次建立地址映射关系，填写虚拟地址。对于实现密码算法的可信应用，我们修改分配物理内存空间的过程，从安全iRAM给栈段分配物理内存空间，从安全DRAM给其他段分配物理内存空间。

管理安全iRAM：我们通过维护一个链表来管理固定范围的安全iRAM。链表中的结点代表已经给某个可信应用分配的安全iRAM空间；在链表中插入/删除结点的操作对应分配/释放安全iRAM空

间; 在分配时, 使用首次适应算法搜索空闲的安全iRAM空间。我们在安全世界操作系统中存储一个实现密码算法的可信应用列表, 内容是所有可信应用的通用唯一识别码(universally unique identifier, UUID)。以此为过滤条件, 仅给列表中的可信应用分配安全iRAM空间, 方案中的辅助可信应用以及系统中的其他可信应用仍使用安全DRAM。

5.2 密码算法实现

我们实现了符合国家/国际标准的具有代表性的几种密码算法, 包括SM2数字签名算法[30]、SM2公钥加密算法[31]、SM3密码杂凑算法[32]、SM4分组密码算法[33]、RSASSA-PSS数字签名算法[34]、RSAES-OAEP公钥加密算法[35]、SHA3-256算法[36]和AES-128算法[37]。其中SM4算法及AES算法的工作模式为ECB模式。

集中敏感数据到栈段: 按照4.2节提出的方案, 实现密码算法时, 仅使用栈中的临时变量存放敏感数据。

分离栈段非敏感数据到安全DRAM: 这部分实现包括必选的变量分离以及可选的变量分离与运算步骤分离。

变量分离(必选项): 按照4.3节所述, 对存放第I类非敏感数据(代码中的全局变量)和第II类非敏感数据(非敏感的输入/输出参数)的变量进行分离。我们实现的密码算法中全局变量信息见表1。已初始化及未初始化的全局变量分别位于可信应用 ELF文件的.data段和.bss段, 被加载到安全DRAM中, 不占用安全iRAM的空间。对于非敏感的输入/输出参数, 在客户端应用调用可信应用之前, 我们在客户端应用中申请共享内存缓冲区, 写入输入参数并传递给可信应用。可信应用执行密码运算后, 将输出参数写入缓冲区, 返回给客户端应用。由于共享内存位于非安全DRAM中, 所以此类数据也不会使用安全 iRAM空间。

变量分离与运算步骤分离(可选项): 对于与敏感数据无关的临时变量的分离以及没有敏感数据参与的运算步骤的分离是可选项。对于SM3, SM4, AES和SHA3-256算法, 其实现非常简洁, 与敏感数据无关的临时变量很少(比如SM4/AES算法中只有一个用于循环计数的int类型变量符合条件), 且全部步骤都有敏感数据参与。所以, 没有对这些算法做可选分离。对于SM2算法和RSA算法, 进行如下的可选分离: 对SM2公钥加密算法和RSAES-OAEP公钥加密算法的实现进行变量分离; 对SM2数字签名算法和RSASSA-PSS数字签名算法的实现进行变量分离及运算步骤分离。

表 1　密码算法实现中的全局变量

Table 1　Global variables in the implementation of cryptographic algorithms

算法	变量名	内容	类型	是否初始化
SM2	g_paramFieldP	确定有限域范围的素数 p	BigNumber	×
	g_paramA, g_paramB	椭圆曲线的参数	BigNumber	×
	g_PntGx, g_PntGy	椭圆曲线基点 G 的 X 坐标和 Y 坐标	BigNumber	×
	g_paramN	椭圆曲线基点 G 的阶	BigNumber	×
	g_Inv2	2 模 p 的逆元	BigNumber	×
	g_tlbG	基点 G 的 2 的 1~256 次方倍点的坐标	BigNumber [256][2]	×
SM3	T1	用于压缩函数 CF 的算法常量	unsigned long	√
	T2		unsigned long	√
SM4	sm4_sbox	S 盒	unsigned char [256]	√
	sm4_ck	固定参数 CK	unsigned int [32]	√
	sm4_fk	系统参数 FK	unsigned int [4]	√
AES	do_init	S 盒及转换表的生成标识	int	√
	KT_init	解密轮密钥的生成标识	int	√
	FSb, FT0, … , FT4	正向 S 盒及转换表	unsigned long [256]	×
	RSb, RT0, … , RT4	反向 S 盒及转换表	unsigned long [256]	×
	RCON	轮常数	unsigned long [10]	×

说明: 全局变量的值是非敏感、固定且公开、系统参数, RSA算法和SHA3-256算法的实现中没有全局变量; BigNumber是用来存储最多256bit大数的数据结构; g_Inv2用于加速椭圆曲线点的加法运算; g_tlbG用于加速椭圆曲线点的乘法运算。

具体来说，变量分离指算法实现中，一些变量存储了4.3节中第3类非敏感数据(表2)。我们将这些变量存储到堆中，由于OP-TEE OS从.bss段中给可信应用划分堆空间，.bss段被加载到安全DRAM中，所以这些变量不占用安全iRAM的空间。

运算步骤分离是指，将SM2数字签名算法以及RSASSA-PSS数字签名算法的部分步骤放在辅助可信应用中实现，该可信应用仅使用安全DRAM而不使用安全iRAM。图3和4分别展示实现SM2数字签名算法以及RSASSA-PSS数字签名算法的客户端应用、可信应用及其辅助可信应用的调用关系。在SM2数字签名生成算法中，客户端应用首先调用可信应用来产生签名，并将待签名的消息(Msg)传递给可信应用；可信应用调用辅助可信应用，参数包括Msg、签名者的公钥(PbKey)及用户可辨别标识符(UserName)；辅助可信应用执行算法的步骤①~③并返回结果；可信应用执行步骤④~⑧，产生数字签名(Signature)，最终将签名返回给客户端应

表 2 密码算法实现中存放第 3 类非敏感数据的变量

Table 2 Variables that store Type 3 non-sensitive data in the implementation of cryptographic algorithms

算法	变量名	内容	类型
SM2 算法	pbkey	用户公钥	PublicKey
	LenofReturn{5}	输出的长度	int
SM2 公钥加密算法	digest	加密结果的 C_3 部分	unsigned char [32]
SM2 数字签名生成算法	LenofMsg	待签名消息长度	int
	UserName	用户可辨别标识	char*
	LenofUserName	用户可辨别标识的长度	int
	TEE_params	传递给辅助可信应用的参数	TEE_Param [4]
	e	密码杂凑函数作用于待签名消息的输出值 e	unsigned char [32]
	temp_x	椭圆曲线点$(x_1, y_1)=[k]G$ 的横坐标 x_1	BigNumber
	temp_e	密码杂凑函数作用于待签名消息的输出值 e	BigNumber
	temp_add	temp_x + temp_e 的结果	BigNumber [2]
RSA 算法	ctx_pub{5}	公开参数的结构	rsa_context_pub
	klen{5}	char 格式的 N 的长度	int
	hlen{3}	消息的密码杂凑值的长度	unsigned int
	ret{5}	函数返回值	int
RSAES-OAEP 算法	pHash	编码参数的密码杂凑值(P)	unsigned char [64]
RSASSA-PSS 算法	slen	盐值的长度	unsigned int

说明：{x}代表在代码中共有 x 个具有相同功能的变量；“公开参数的结构”包括算法实现的版本号、模数 N、公钥 e、char 格式的 N 的长度、填充标识符、EME-OAEP 和 EMSA-PSS 的密码杂凑标识符。

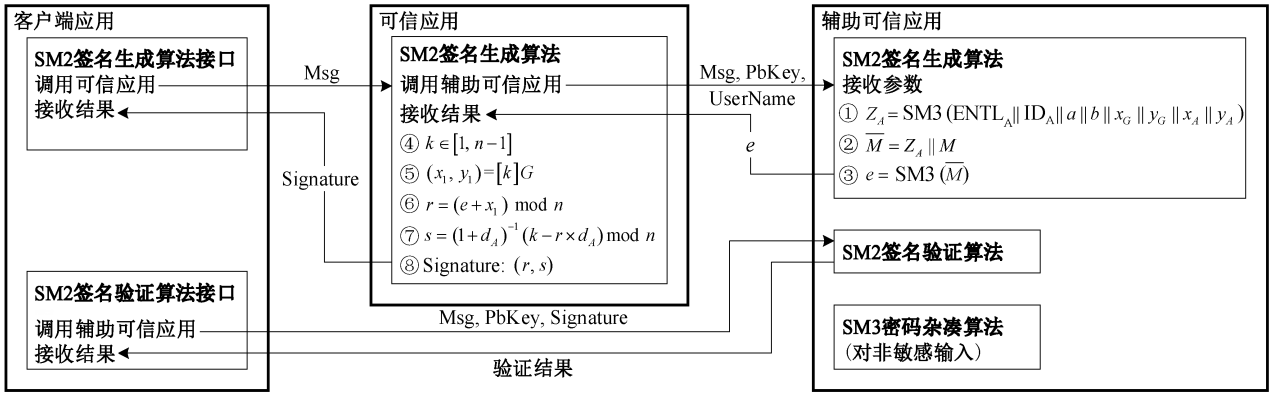

图 3 实现 SM2 数字签名算法的客户端应用、可信应用及其辅助可信应用之间的调用关系

Fig. 3 Invocation between CA,TA and assisted TA which implement SM2 digital signature algorithm

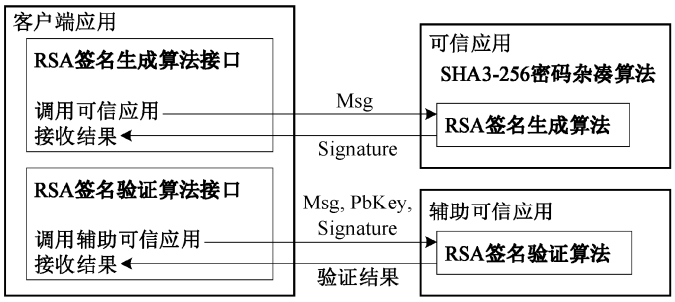

图 4　实现 RSASSA-PSS 数字签名算法的客户端应用、可信应用及其辅助可信应用之间的调用关系
Fig. 4　Invocation between CA,TA and assisted TA which implement RSASSA-PSS digital signature algorithm

用。在SM2数字签名验证算法中，普通世界的客户端应用直接调用辅助可信应用进行签名验证；辅助可信应用进行验证后返回验证结果。

RSASSA-PSS数字签名生成算法包括以下3步：1) 计算待签名消息的杂凑值；2) 对杂凑值进行填充；3) 对填充结果进行签名运算。我们没有使用辅助可信应用执行步骤1和2，主要出于两个原因：一方面，步骤2中填充数据的泄露会导致RSA算法的安全性降低[38]，所以步骤2涉及敏感数据，不能在辅助可信应用中执行；另一方面，步骤2的填充过程同样包括杂凑计算，所以仅将步骤1放在辅助可信应用中不会明显减少安全iRAM的使用量，故未将步骤1放在辅助可信应用中。在RSASSA-PSS数字签名算法中，普通世界的客户端应用调用可信应用进行签名，调用辅助可信应用进行签名验证，两个可信应用执行密码运算后返回结果。

我们将多个密码算法实现为安全世界的不同可信应用，对应普通世界中的多个客户端应用。由于普通世界和安全世界的操作系统Android(Linux内核)和OP-TEE OS都支持多线程调度，且方案对操作系统的调度功能是透明的，所以在多个客户端应用同时发起密码运算请求时，多个可信应用(即多个密码算法)可以自动实现并发执行。

6 验证评估

从iRAM使用量及性能两个方面评估本文提出的方案。实验平台为FreeScale的i.MX6Quad开发板，该开发板配备主频1.2 GHz的四核Cortex-A9处理器，系统DRAM容量为1 GB，iRAM容量为256 KB，其中空闲的iRAM容量为32 KB (扣除普通世界操作系统占用的iRAM空间)。

6.1 iRAM 使用量评估

对于每个可信应用，分别测试其栈段、堆段、.data段和.bss段(除堆以外的部分)的大小。经过测试，当输入为4，64和256 KB时，各算法对iRAM的使用量相同。SM2算法中用到的用户可辨别标识为默认值“1234567812345678”；RSAES-OAEP算法中的标签(Label)为空字符串。测量可信应用中各段实际大小的方法如下。

栈：在可信应用中将栈的大小统一设置为5 KB。通过在加载可信应用的过程中添加打印语句，了解到OP-TEE OS实际为可信应用分配了8 KB的栈空间。为测试可信应用实际使用的栈大小，在可信应用调用密码算法的功能前，在全部栈空间中写入特定的值(0x12345678)；在可信应用结束调用，关闭会话前，找到栈空间中值不等于0x12345678的最低地址，进而计算出实际使用的栈大小。

.data/.bss段：可信应用在编译后会生成后缀名为map的文件，记录ELF文件中各段的内容、大小及偏移，从该文件中获取.data段和.bss段的大小。

堆：在可信应用中，将堆的大小统一设置为20 KB。在map文件中，可以获取堆的起始地址在.data/.bss段中的偏移；加载可信应用时，可以从OP-TEE OS中获取.data/.bss段的起始地址。使用与栈段相同的方法，提前写入特定值，在可信应用结束运行后，计算实际使用的堆大小。

我们分别对SM2数字签名算法(SM2_SIG)、SM2公钥加密算法(SM2_ENC)、SM3密码杂凑算法(SM3)、SM4分组密码算法(SM4)、AES算法(AES-128)、RSA数字签名算法(RSA_SIG)、RSA公钥加密算法(RSA_ENC)和SHA3算法(SHA3-256)进行测试，结果如表3所示(其中SM2算法和RSA

表 3 密码算法实现的内存使用量(B)

Table 3 Memory usage of the implementation of cryptographic algorithms (B)

评估对象		SM2_SIG	SM2_ENC	SM3	SM4	AES-128	RSA_SIG	RSA_ENC	SHA3-256
可加载段的内存使用量	栈	4444	3852	2232	1864	2376	4312	4296	2060
	堆	104	135	200	232	200	14812	15972	200
	.data	2044	2044	8	400	8	0	0	0
	.bss	24952	24952	0	0	14376	0	0	0
安全iRAM的使用量	Oath[19]	31544	30983	2440	2496	16960	19124	19628	2260
	本文方案	4444	3852	2232	1864	2376	4312	4296	2060
	减少率/%	85.91	87.75	8.52	25.32	85.99	77.45	78.11	8.85

说明: “.bss”指.bss 段除堆以外的部分的大小。

算法未实现可选的非敏感数据分离)。Oath方案[19]将可信应用的栈段和.data/.bss段存储在安全iRAM中, 所以对iRAM的使用量为所有可加载段的内存使用量的和; 本方案对iRAM的使用量为栈的大小。表3最后一行数据展示本方案比Oath对iRAM使用量的减少率, 最大值为SM2公钥加密算法的87.75%。

我们还测试了实现可选的变量分离与运算步骤分离过后, SM2算法和RSA算法对iRAM的使用量, 结果如表4所示。由于SM2和RSA的数字签名验证算法在辅助可信应用中实现, 所以iRAM的使用量都为0。可以看到, 可选的变量分离与运算步骤分离对减小iRAM的使用量有一定的作用。

6.2 性能评估

本文方案对性能的影响包括两方面: 第一是可信应用的加载时间, 这是由于本文方案修改了可信应用的加载方式; 第二是算法本身的性能。我们使用 Cortex-A9处理器中的性能监控单元(performance monitoring unit, PMU)对这两部分影响进行评估, 对于每项测试重复运行500次并取平均值。

1) 可信应用加载时间评估。可信应用的加载时间指普通世界首次调用可信应用时, 将可信应用的 ELF文件从文件系统加载到内存所用的时间。经过测试, 当可信应用的全部数据存储在安全DRAM中时, 平均加载时间为82.22 ms, 基于本文方案实现的可信应用平均加载时间为85.23 ms, 增加3.66%。作为对比, Oath方案[19]对10个不同种类的可信应用的加载时间增量为1.37%。

2) 密码算法性能评估。我们分别评估原始的密码算法实现(基于安全DRAM, 未经本文方案的敏感数据集中非敏感数据及运算步骤分离)、基于Oath及本文方案保护的密码算法实现性能。对于SM2算法及RSA算法, 还评估包含可选分离时的性能。以原始的密码算法实现性能为基准, 统计后3种情况下各算法性能的相对值, 每个统计结果为不同大小输入(16 B~1 MB)条件下相对值的平均数。统计结果如表5所示。

由表5的性能评估数据可以得出如下结论。

1) 与原始基于安全DRAM的密码算法实现相比, 本文方案在不实现可选数据分离时引入的性能开销很小, 各算法的平均性能开销在–0.19%~2.25%之间(平均最小值和最大值分别为RSA签名算法和SM3算法)。在很多情况下, 本文方案的性能比原始情况更好, 主要是由于iRAM的读写速度大于DRAM。

2) 本文方案在不实现可选数据分离时, 相比Oath在性能方面的差异几乎可以忽略不计。

3) 可选数据分离对密码算法的性能影响是可以接受的。对于SM2公钥加密算法, 加密算法的平均性能开销为1.37%, 解密算法为1.11%, 最大性能

表 4 非敏感数据的可选分离前后 SM2 算法和 RSA 算法的 iRAM 使用量对比(B)

Table 4 Comparison of iRAM usage of SM2 and RSA algorithm before and after the optional separation of non-sensitive data (B)

评估对象	SM2			RSA			SM2 验签	RSA 验签
	签名	加密	解密	签名	加密	解密		
可选分离前	4444	3788	3852	3840	3656	4296	4428	4312
可选分离后	3604	3548	3588	3728	3520	4152	0	0
减少率/%	18.90	6.34	6.85	2.92	3.72	3.35	100	100

表 5　不同保护方案下各密码算法的性能与无保护时性能的比值(%)

Table 5　Ratios of the cryptographic algorithms' performance with different protection schemes to which without protection (%)

算法	SM2				RSA				SM3	SM4	AES-128	SHA3-256
	签名	验签	加密	解密	签名	验签	加密	解密				
Oath[19]	100.29	99.77	98.63	98.89	99.90	100.06	99.81	99.98	98.59	99.80	98.81	99.87
本文方案(无可选分离)	99.98	99.96	99.98	99.98	100.19	100.06	100.12	100.08	97.75	99.99	99.38	99.16
本文方案(含可选分离)	97.49	98.92	99.45	97.59	99.86	100.60	100.04	99.20	–	–	–	–

开销为2.75%, 出现在解密900 KB的消息时。对于SM2数字签名算法, 签名算法的平均性能开销为2.51%, 验签算法为1.08%, 在对4 KB的消息进行签名时产生最大开销, 为4.29%; 对于不同大小的输入, RSA算法的性能开销均未超过1.2%。

7 结论

本文提出一种基于iRAM的ARM平台密码算法实现方案, 利用iRAM的物理特性实现密码算法的抗物理内存泄漏攻击特性, 同时利用ARM Trust Zone技术实现抗软件攻击特性。特别地, 本文通过对密码运算过程中敏感数据细粒度的准确划分以及执行密码运算可信应用数据段的控制调度, 降低了iRAM的消耗, 使得在iRAM的消耗降低78%以上的情况下, 仍能达到与未修改的实现相近的性能。iRAM资源消耗的降低使得该方案能够在不影响系统正常功能的条件下, 在当前主流ARM平台部署。

参考文献

[1] ARM. The architecture for the digital world [EB/OL]. (2022)[2022–02–13]. https://www.arm.com

[2] Halderman J A, Schoen S D, Heninger N, et al. Lest we remember: cold-boot attacks on encryption keys. Communications of the ACM, 2009, 52(5): 91–98

[3] Müller T, Spreitzenbarth M. FROST // Applied Cryptography and Network Security. Berlin: Springer, 2013: 373–388

[4] Moon H, Lee H, Lee J, et al. Vigilare: toward snoop-based kernel integrity monitor // Proceedings of the 2012 ACM Conference on Computer and Communications Security. Raleigh, 2012: 28–37

[5] Lee D, Jung D, Fang I T, et al. An off-chip attack on hardware enclaves via the memory bus // 29th USENIX Security Symposium (USENIX Security 20). Boston, 2020: 487–504

[6] FuturePlus. FuturePlus systems [EB/OL]. (2022) [2022–02–13]. https://www.futureplus.com

[7] Nexus Technology. MA4100 series memory analyzer [EB/OL]. (2022) [2022–02–13]. https://www.nexustechnology.com/products/memory-analyzers/ma4100-series-memory-analyzer/

[8] Stewin P, Bystrov I. Understanding DMA malware // International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Berlin: Springer, 2013: 21–41

[9] Ufrisk. PCILeech [EB/OL]. (2022) [2022–02–13]. https://github.com/ufrisk/pcileech

[10] Markettos A T, Rothwell C, Gutstein B F, et al. Thunderclap: exploring vulnerabilities in operating system IOMMU protection via DMA from untrustworthy peripherals // Proceedings 2019 Network and Distributed System Security Symposium. San Diego, 2019: 23194

[11] Guan Le, Lin Jingqiang, Luo Bo, et al. Copker: computing with private keys without RAM // Proceedings 2014 Network and Distributed System Security Symposium. San Diego, 2014: 23125

[12] Guan Le, Lin Jingqiang, Luo Bo, et al. Protecting private keys against memory disclosure attacks using hardware transactional memory // 2015 IEEE Symposium on Security and Privacy. San Jose, 2015: 3–19

[13] Zhang Ning, Sun Kun, Lou Wenjing, et al. CaSE: cache-assisted secure execution on arm processors // 2016 IEEE Symposium on Security and Privacy (SP). San Jose, 2016: 72–90

[14] Müller T, Freiling F C, Dewald A. TRESOR runs encryption securely outside RAM // 20th USENIX Security Symposium (USENIX Security 11). San Francisco, 2011: no. 103

[15] Garmany B, Müller T. PRIME: private RSA infrastructure for memory-less encryption // Proceedings of the 29th Annual Computer Security Applications Conference. New Orleans, 2013: 149–158

[16] Colp P, Zhang Jiawen, Gleeson J, et al. Protecting data on smartphones and tablets from memory attacks // Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems. Istanbul Turkey, 2015: 177–189

[17] Cao Chen, Guan Le, Zhang Ning, et al. CryptMe: data leakage prevention for unmodified programs on ARM devices // Research in Attacks, Intrusions, and Defenses. Cham: Springer International Publishing, 2018: 380–400

[18] Zhao Shijun, Zhang Qianying, Qin Yu, et al. Minimal kernel: an operating system architecture for TEE to resist board level physical attacks // 22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019). Beijing, 2019: 105–120

[19] Chu Dawei, Wang Yuewu, Lei Lingguang, et al. OCRAM-assisted sensitive data protection on ARM-based platform // Computer Security — ESORICS 2019. Cham: Springer International Publishing, 2019: 412–438

[20] ARM. Building a secure system using TrustZone® technology [EB/OL]. (2009) [2021–12–19]. https://documentation-service.arm.com/static/5f212796500e883ab8e74531

[21] OP-TEE. Pager [EB/OL]. (2022) [2022–02–13]. https://optee.readthedocs.io/en/latest/architecture/core.html#pager

[22] Tillich S, Großschädl J, Szekely A. An instruction set extension for fast and memory-efficient AES implementation // Communications and Multimedia Security. Berlin: Springer, 2005: 11–21

[23] Seo H. Memory efficient implementation of modular multiplication for 32-bit ARM Cortex-M4. Applied Sciences, 2020, 10(4): 1539

[24] Liu Zhe, Seo H, Castiglione A, et al. Memory-efficient implementation of elliptic curve cryptography for the internet-of-things. IEEE Transactions on Dependable and Secure Computing, 2019, 16(3): 521–529

[25] Seo H. Compact software implementation of public-key cryptography on MSP430X. ACM Transactions on Embedded Computing Systems, 2018, 17(3): 1–12

[26] Mera J M B, Karmakar A, Verbauwhede I. Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020, 2: 222–244

[27] LINARO. Open portable trusted execution environment [EB/OL]. (2022) [2022–02–13]. https://www.op-tee.org/

[28] ARM. PrimeCell DMA controller (PL330) technical reference manual [EB/OL]. (2022) [2022–02–13]. https://developer.arm.com/documentation/ddi0424/a/

[29] 96Boards. HiKey960 SoC reference manual [EB/OL]. (2016–12–22) [2021–12–14]. https://github.com/96boards/documentation/raw/master/consumer/hikey/hikey960/hardware-docs/HiKey960_SoC_Reference_Manual.pdf

[30] 国家密码管理局. GMT 0003.2—2012 SM2 椭圆曲线公钥密码算法第 2 部分: 数字签名算法. 北京: 中国标准出版社, 2012: 1–13

[31] 国家密码管理局. GMT 0003.4—2012 SM2 椭圆曲线公钥密码算法第 4 部分: 公钥加密算法. 北京: 中国标准出版社, 2012: 1–16

[32] 国家密码管理局. GMT 0004—2012 SM3 密码杂凑算法. 北京: 中国标准出版社, 2012: 1–16

[33] 国家密码管理局. GMT 0002—2012 SM4 分组密码算法. 北京: 中国标准出版社, 2012: 1–9

[34] RFC4056, use of the RSASSA-PSS signature algorithm in cryptographic message syntax (CMS) [S]. Reston: The Internet Society, 2005

[35] RFC3447, Public-key cryptography standards (PKCS) #1: RSA cryptography specifications Version 2.1 [S]. Reston: The Internet Society, 2003

[36] NIST FIPS 202, SHA-3 standard: permutation-based Hash and extendable-output functions [S]. Gaithersburg: National Institute of Standards and Technology, 2015

[37] NIST FIPS 197, advanced encryption standard (AES) [S]. Gaithersburg: National Institute of Standards and Technology, 2001

[38] Bleichenbacher D. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1 // Advances in Cryptology — CRYPTO'98. Berlin: Springer, 1998: 1–12