几个初始节点等, 使得集群内每个设备都不会因为初始节点的问题而连接不上. 而管理节点除了发送自身的心跳信息之外还要接受邻居节点的心跳信息. 如果收到邻居节点心跳信息后发现这个心跳信息自己没有记录, 则将这个信息记录在日志中, 然后将该心跳信息再转发给所有邻居. 直到管理节点收到所有设备的心跳信息或者超时, 最后将自身的日志信息发送给 VRF.

(2) 收集证明阶段

收集证明阶段的目标分为两方面: 一方面是管理节点对同构设备子节点的验证, 如图 1 中的 Subatt, 管理节点比较子节点发来的校验和, 不同于大多数的为被篡改节点, 否则为可信节点; 另一方面是网络拓扑结构中, 对管理节点组成的集群的证明, 如图 1 中的 AttVerify.

实验涉及到的主要消息类型有: ① req, 表示消息类型是请求类型. ② rep, 表示消息类型是响应类型. ③ subreq, 表示管理节点对子设备节点的请求信息. ④ subrep, 表示子设备节点对管理节点的回复信息. 对于待发送和接收的数据包, 我们分别用 MsgS(MsgF) 和 MsgR 表示.

1) 对分组内部同构节点间的验证.

由于同构节点的内存内容是相同的, 因此内存的校验和也是相同的.所以可以让大量的同构节点同时进行内存校验, 校验和不同的节点将被认为是内存被篡改节点. 具体流程见算法 Subatt_MgrNode, Subatt_SubNode, 分别从管理节点和子设备节点的角度来描述协议 Subatt.

a) 管理节点: 管理节点的验证分为四步, 如算法 1.

① 发送请求: 管理节点每隔一个固定的时间 (这个时间小于远程验证间隔) 就发送加密的验证请求 subreq 检测子设备节点的状态.验证请求中包含一个随机数据 Randata 和一个递增的序列号 CurSeq 以及设备 ID. 然后设置定时器进入等待阶段. 每次等待的时间固定为 T_{Att}.

② 等待: 管理节点等待子设备节点传回来的验证信息 subrep.如果收到信息时未超过定时器设置的时间则进入记录阶段.

③ 记录: 如果收到的消息属于 subrep 类型, 管理节点检查收到的信息的签名并解密. 如果信息中的序列号跟管理节点发送的序列号一致则记录该信息中的校验和并将信息中包含的节点 ID 放入已回复子节点集合 RecSubNode 中. 进入等待阶段.

④ 校验: 验证时间超过定时器设置的时间之后进入该阶段. 管理节点比对所有于记录阶段记录的子设备节点的校验和, 和大多数子设备节点不同的校验和的节点加入验证失败子设备集合 FailSubDevs 中, 其余的加入验证成功子设备集合 SucSubDevs. 剩余未发来验证信息的子设备节点加入未回复子设备集合 NorepSubDevs 中.

算法 1. Subatt_MgrNode (从管理节点一方)

```
1. while True do
2. Wait(const time);
3. CurSeq = getSeq();
4. RanData = getRandomData();
5. MsgS = SignEncrypt("subreq" ||DevID ||CurSeq||RanData);
6. MULTICAST(MsgS);
7. T = getTimer();
8. while (getTimer() - T) < T_Att) do
9. MsgR, DevID = receive();
10. if (Type(MsgR) == "subrep") then
11. CheckSignature(MsgR);
12. Seq, SubChecksum← Decryption(MsgR);
13. if (Seq == CurSeq) then
14. SubCheckList← DevID||SubChecksum;
15. Append(RecSubNode.DevID);
16. end if
17. end if
18. end while
19. SucSubDevs, FailSubDevs← DifferChk(SubCheckList);
20. NorepSubDevs← NotInRecSubNode(SubNodeSet, RecSubNode);
21. SubStateList← SucSubDevs, FailSubDevs, NorepSubDevs;
22. end while
```

b) 子设备节点: 子设备节点的验证分为三步, 如算法 2 所示.

① 等待: 等待管理节点发来的验证请求 subreq. 收到请求则进入验证阶段.

② 验证: 如果收到的消息属于 subreq 类型, 检查管理节点发来的验证请求的签名并解密获取序列号及随机数据. 再验证该请求的序列号, 必须大于子设备节点存储的上一次验证阶段使用的序列号. 通过验证之后, 子设备节点用随机数填充空白内存, 然后计算内存和收到的随机数据的校验和. 进入回应阶段.

③ 回应: 子设备节点将节点 ID, 接收到的序列号和上一步产生的校验和打包加密以 subrep 类型的消息发回管理节点 (父节点 par). 记录当次校验的序列号并恢复内存.

算法 2. Subatt_SubNode (从子设备节点一方)

```
1. while True do
2. Wait(const time);
3. MsgR = receive();
4. if (Type(MsgR) == “subreq”) then
5. CheckSignature(MsgR);
6. Seq, RanData← Decryption(MsgR);
7. if (Seq > PrevSeq) then
8. fillMemory(RanData);
9. Checksum=ComputeChecksum(RanData);
10. MsgS = SignEncrypt(“subrep”|| DevID||Seq||Checksum);
11. UNICAST(MsgS)→Par;
12. PrevSeq = Seq;
13. recoverMemory();
14. end if
15. end if
16. end while
```

2) 对全局集群环境内所有管理节点的验证.

对管理节点的验证比较简单, 由于 VRF 中已有管理节点的内存信息, 并且管理节点不同构, 因此可以针对每个管理节点的内存校验和进行单独比对.具体流程见算法 AttVerify_VRF, AttVerify_MgrNode. 分别从验证者 VRF 和管理节点的角度来描述协议 AttVerify.

a) 验证者 VRF: VRF 的验证分为四步, 如算法 3.

① 发送请求: VRF 根据需求选择一个管理节点 InitNode 向其发送加密的验证请求来检测所有设备节点的状态. InitNode 优先选择更安全的管理节点或者距离更近的管理节点. 验证请求中包含 InitNode 要计算校验和的随机数据 RanData 和一个递增的序列号 CurSeq. 然后设置定时器进入等待阶段. 设定接收验证和最长需要时间 T_{Rep}.

② 等待: VRF 等待管理节点传回来的验证信息 rep. 如果收到信息时未超过定时器设置的时间则进入校验阶段.

③ 校验: 如果收到的消息属于 rep 类型, VRF 检查收到的信息的签名并解密. 如果信息中的序列号 Seq 跟管理节点发送的序列号 CurSeq 一致则根据收到的随机数据 RanData 计算对应管理节点的校验和并与该信息中的校验和比对. 如果比对结果一致则将该管理节点归于 SucSubDevs, 并根据信息中的子设备状态 SubStateList 将子设备归于 SucSubDevs, FailSubDevs 或者 NorepSubDevs. 再将信息中包含的节点 ID 放入已回复子节点集合 RecSubNode 中进入等待阶段.

④ 结束: 验证时间超过定时器设置的时间之后进入该阶段. 失去联络的管理节点归为 NorepSubDevs. 将这三组集合跟校验阶段收集的子节点验证记录一起输出给用户.

算法 3. AttVerify_VRF 验证协议 (从 VRF 一方)

```
1. while True do
2. WaitForVerifyCommand();
3. CurSeq = getSeq();
4. RanData = getRandomData();
5. MsgS = SignEncrypt(“req”||CurSeq||RanData);
6. UNICAST(MsgS)àInitNode;
7. T = getTimer();
8. while (getTimer() - T) < T_Rep do
9. MsgR, DevID = receive();
10. if (Type(MsgR) == “rep”) then
11. CheckSignature(MsgR);
12. DevID, RanData, Seq, SubStateList, SubChecksum←
Decryption(MsgR);
13. if (Seq == CurSeq) then
14. if (SubChecksum== ComputeChecksum(RanData)) then
15. SucSubDevs← DevID;
16. SucSubDevs, FailSubDevs,
17. NorepSubdevs← SubStateList;
18. else
19. FailsubDevs← DevID, SubStateList;
20. end if
21. Append(RecSubNode, DevID);
22. end if
23. end if
24. end while
25. NorepSubDevs← NotInRecSubNode( SubNodeSet, RecSubNode);
26. end while
```

b) 管理节点: 管理节点的验证分为三步, 如算法 4.

① 等待: 等待本次验证相关信息. 如果收到 VRF 发来的验证请求 req 则进入验证阶段; 如果收到其余管理节点的验证信息 rep 则进入转发阶段.

② 验证: 判断接收到的信息的签名是否正确.正确则解密信息从信息中获得序列号 Seq 和随机数据 RanData. 再验证该请求信息的序列号, 必须大于管理节点存储的上一次验证阶段使用的序列号 PrevSeq. 通过验证之后先产生一个新的随机数据 NewRanData 并将其与刚才获取的序列号 Seq 一起加密之后以 req 信息的形式群发给邻居节点. 接着将内存中的空白部分用随机数进行填充, 然后计算内存和随机数据 RanData 的校验和. 再将本节点 ID, 校验和 Checksum,

序列号 Seq, 随机数据 RanData 连同子设备的可信状态一起传送给父节点. 记录当次校验的序列号并恢复内存, 进入等待阶段.

③ 转发: 将收到的 rep 类型的消息转发给父节点. 进入等待阶段.

算法 4. AttVerify_MgrNode 验证协议 (从管理节点一方)

```
1. while True do
2. MsgR = receive();
3. if (Type(MsgR) == “req”) then
4. CheckSignature(MsgR);
5. Seq, RanData ← Decryption(MsgR);
6. if (Seq > PrevSeq) then
7. NewRanData = getRandomData();
8. MsgF = SignEncrypt(“req”|| CurSeq ||NewRanData);
9. MULTICAST(MsgF);
10. fillMemory(RanData);
11. Checksum =
12. ComputeChecksum(RanData);
13. MsgS = SignEncrypt(“rep”||Randata||DevID||Seq||
SubStateList||Checksum);
14. UNICAST(MsgS)→Par;
15. PrevSeq = Seq;
16. recoverMemory();
17. end if
18. else if (Type(MsgR) == “rep” )then
19. UNICAST(MsgR)→Par;
20. end if
21. end while
```

3 安全性分析

为了保障验证结果的安全性, 我们依据远程证明安全威胁模型针对常见攻击方法进行如下安全分析.

(1) 伪造攻击: 指在攻击者不知道密钥的情况下, 构造一个新的消息及其签名值. 本系统采用加密签名算法, 由于假设在离线阶段的初始密钥交换阶段, 通信信道是安全的以及密码函数是安全的, 所以当敌手在通信信道上篡改或者伪造消息时, 会导致签名验证失败, 从而达到防伪造效果. 而且当管理节点与子设备节点同构时, 管理节点也不能利用子节点传来的校验信息, 因为随机数产生函数存储在被保护内存区域 R 中, 外部函数无法访问到它, 这样随机数产生函数只能由认证代码控制, 而每次产生的随机数不同, 即管理节点计算校验和的随机数和子设备节点不同, 所以无法伪造; 而且即使管理节点可以给子设备节点传自己的随机数也会带来时间的延迟, 所以基本上无法直接利用子节点信息进行伪造.

(2) 重放攻击: 指通过将以前认证的数据传送给 VRF, 本系统通过将随机数加入验证过程来防止预计算, 从而避免重放攻击.

(3) 内存复制或内存替换攻击: 指攻击者将内存复制到空白区域, 当验证内存时, 通过修改程序计数器 PC 和数据指针 DP 来实现攻击, 本系统通过给空白内存填充随机数使得没有空间存储攻击代码达到防范的目的.

(4) 代理攻击: 指使用更快的远程设备代理 checksum 计算.但是远程设备无法获得真正的密钥, 所以无法正确加密和签名, 可以防御.

(5) 内存压缩攻击: 压缩获得可以利用的空闲内存, 证明时实时解压. 本系统通过设定认证时间以及在空白内存中填充随机数能一定程度上防范这种攻击.

(6) 合谋攻击: 指两个设备串联攻击.本系统中当合谋攻击发生在子设备节点下的时候, 因为发送的都是加密签名过的结果, 即使发送给合谋节点, 因为合谋节点没有对应的密钥, 无法解密, 所以合谋攻击无效; 当发生在管理节点之间的时候, 场景是邻居节点将随机数预先传给下一节点, 导致下一节点可以预先计算. 本系统通过将产生随机数的函数放于被保护内存区域 R 中, 只有认证代码可以调用它, 使得随机数不能事先产生, 防止了合谋攻击.

除此之外, 本系统采用一些方式来加强系统安全性.

(7) 机密性保障: 本系统采用信息加密的方式, 并且假设使用的密码学函数是安全的, 所以敌手即使截获信息, 也无法获取消息内容, 达到防窃听的目的; 另一方面, 本系统中除了 VRF, 任何两个节点之间不知道对方的软硬件配置, 一定程度上可以保护隐私, 而且当一台设备被攻击的时候, 不会获取其他设备的数据, 使设备更安全.

(8) 可用性保障: 本系统通过在缺失探测节点之后, 重新选取管理节点, 使得缺失节点不影响其他节点的传输来保障协议的进一步运行. 除此之外, 本方案先用 OP 来验证 VRF, 可以在一定程度上防范 DOS 攻击. 而且, 本方案因为可以发现伪造攻击, 所以当敌手伪造一个请求的时候, 接下来的设备要进行身份验证, 验证没通过时会停止转发与计算校验和等, 所以只有身份验证这一步会产生 DOS 攻击, 一定程度上也可以减弱.

同时, 针对本文不考虑的攻击:运行时攻击和无法

通过缺失探测检测的物理攻击, 进行一些相关讨论.针对运行时攻击如 ROP 攻击 (Return-oriented Programming), 对于该攻击, 现在已有一些运行时证明方案如 C-FLAT[21], ATRIUM[22]等, 可以设计方案对执行指令以及控制流等进行认证; 对于不能通过缺失探测检测出的物理攻击, 如侧信道攻击等, 需要对设备的安全架构进行改进, 现在已有相关的架构, 如文献[23]中所示架构, 针对 cache 侧信道攻击, 利用硬件事务内存使得敏感代码和数据在程序执行过程中常驻 cache 来防止信息泄露.

4 实验评估

我们先在一个普通的 ARM SOC 开发板 Real210 上[24]对加密等关键操作进行测试和评估, 然后针对评估结果, 用 Python 语言在 Common Open Research Emulator (CORE)[25]软件上对本文提出的高效集群证明方案进行了模拟. CORE 是一个运行于 Linux 操作系统上的虚拟网络仿真工具. 该工具利用了 Linux 的网络堆栈从而使得该工具的仿真性能与真实网络非常相似.

4.1 实验设置

在本次模拟实验中, 我们使用了两种集群证明方案. 第一种采用最简单的集群证明方案, VRF 发一个验证请求给星型拓扑中的初始节点, 然后初始节点将验证请求转发给邻居节点, 以此类推, 一层层转发. 每个节点计算完校验和后将结果经过一层层转发, 传给 VRF. VRF 收到回应后, 一个个进行证明. 所有新加入的节点都直接加入星型拓扑中. 第二种则是本文提出的高效集群证明方案, VRF 跟管理节点处于同一个星型拓扑中.

两种方案中的所有节点都采用 802.11 作为数据链路层的介质访问控制协议. 同时采用 Optimized Link State Routing (OLSR) 协议作为网络层协议. 在密码方面选择 128 位的 SM4 进行加解密以及采用 256 位的 SM2 进行签名及签名验证操作. 内存验证则采用 SM3 算法.

本文采用的实验机器分别是 Lenovo ThinkCentre M4000t, 与 Real210 开发板, 它们通过 USB host 接口连接; Real210 开发板采用 ARM Cortex-A8, 频率为 1 GHz, 有 512 MB 内存, 256 MB NAND Flash. 数据采集 SM4 加解密和 SM2 签名验证分别是在 Real210 开发板上[24]测试出来的. 由于 Verifier 也要进行节点内存验证, 所以节点内存验证速度是在 Windows 机器上模拟出来的, 具体结果如表 2 所示.

表 2 采用的密码操作耗时

密码操作	耗时 (ms)
SM4 加密	7.0
SM4 解密	7.1
SM2 签名	176.0
SM2 验证	326.0
SM3 节点内存验证速度	43.445 MB/s

4.2 实验结果

本次模拟实验分成两部分. 第一部分测试两种集群证明方案的运行时间. 两种集群都有 50 个节点并且节点只需要验证 1 MB 的内存. 第二部分测试高效集群证明方案的性能.

第一部分测试的结果如图 3 所示.对于简单集群验证, x 轴表示整个集群节点一共有 50 个节点. 对于高效集群验证, x 轴表示管理节点的数量, 从 10 个到 50 个.由图可知, 集群中全部都是管理节点时, 高效集群验证跟简单集群验证效率一样. 但是随着同构程度的加深, 管理节点开始变少, 而管理节点的子节点开始增多, 高效集群验证的效率越来越高. 当只有 10 个管理节点时, 高效集群验证使用的时间大约只有简单集群验证的 1/4.

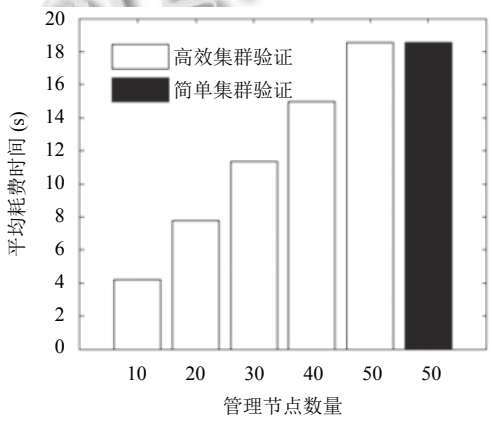
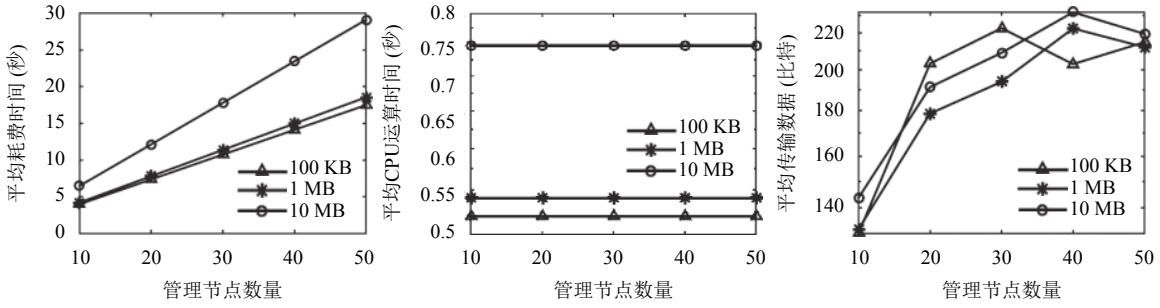


图 3 不同管理节点数量下两种方案运行时间

第二部分测试的结果如图 4 所示. 高效集群证明所耗费的时间随着需要验证的内存增大而增大. 这主要是因为 VRF 计算管理节点的校验和的时间也随着内存大小的增加而增加. 同时 CPU 运行时间随着需要

验证的内存大小的增加而增加. 节点的平均传输数据与节点数量相关性比较大, 与内存大小相关性比较小.



(a) 不同大小内存下完成高校集群证明所需时间　(b) 不同大小内存下节点的平均CPU运行时间　(c) 不同大小内存下节点的平均传输数据

图 4　高效集群证明性能测试

5　比较与讨论

“一对一”验证指 Verifier 一个个验证设备节点. 我们这里指 Verifier 同时发送 n 个验证请求, 然后设备节点进行计算并将验证结果返回给 Verifier, 然后 Verifier 一个个进行相关计算与比对来验证设备的可信性.

我们与 SEDA[19]以及“一对一”验证进行了比较, 如表 3 所示. 我们假设总设备个数为 n, 管理节点个数为 m, 时间耗费主要考虑从 Verifier 发送校验请求到 Verifier 完成整个校验总共花费的计算校验和的时间. 本文方案, 管理节点可以预先得知子节点信息, 所以 Verifier 只要验证管理节点信息即可, 而且管理节点和管理节点之间传完随机数后, 计算校验和以及传输的过程都是并行的, 比较省时. 如表 3 所示, 本文提出的高效集群证明机制可以识别具体损坏设备、时间消耗也比较低, 同时可以一定程度上抵御物理攻击和 DOS 攻击. 当 SEDA 以及“一对一”验证和本文的假设条件一致时, 本文抵御的软件攻击更多. 在可用性上, 如第 3 节安全性分析那章所示, 因为采用了缺失探测提前排查设备等, 提高了可用性; 从通用性的角度来说, 因为本实验方案在同构设备比较多的情况下有效, 如果没有同构设备, 其时间性能也是 O(n). 从 Verifier 是否容易成为性能瓶颈来说, 因为“一对一”验证一般要收发 n 个包并对这 n 个设备进行校验, 比较繁琐而且容易成为性能瓶颈, 而 SEDA 和本文方案可以减轻这一点, 只需要发一个包就可以得到验证结果, 而且 Verifier 要验证的设备大幅度减少, 降低了 Verifier 的计算量, 使其相对不容易成为性能瓶颈.

表 3　高效集群证明机制和 SEDA[19]以及单一 prover 验证的比较

	本方案	SEDA[19]	“一对一”验证
是否可以识别具体损坏设备	可以	否	可以
时间消耗	O(m)	O(n)	O(n)
是否考虑物理攻击	考虑	不考虑	不考虑
是否考虑 DOS 攻击	考虑, 但是只能减轻一部分	不考虑	不考虑
可以减弱或抵御的软件攻击	伪造攻击、重放攻击、内存复制或内存替换攻击、代理攻击、内存压缩攻击、合谋攻击	伪造攻击、重放攻击、代理攻击	伪造攻击、重放攻击, 不存在合谋攻击
可用性	高	较高	高
通用性	适用于同构设备比较多的情况下, 否则效率和 SEDA 一样	高	高
Verifier 是否容易成为性能瓶颈	否	否	是

虽然本文在 SEDA 上进行了改进, 但是本文因为采用管理节点验证子节点的想法, 所以当管理节点被验证失败的时候, 其负责的子节点就要重新被验证.

将本方案扩展成通用模型, 使其可以应用到其他场景时, 就有必要对本实验方案做进一步讨论.

(1) 本文管理节点验证子设备节点用的是多数表决机制, 但是如果管理节点和子设备节点采取的是同构设备, 也可以采用一一验证. 因为管理节点和子设备

节点的内存一样, 但是因为填充内存以及计算内存校验和的随机数不同, 所以管理节点还需要重新计算子设备节点的校验和, 增加了管理节点负载.

(2) 本文假设同构设备中大多数是可信的. 但是在通用场景中, 存在着大多数设备被攻击的可能性. 在这种情况下, 可以通过在管理节点中存储同构设备子节点中的一个标准值, 当收到子设备节点的校验和时, 用这个标准值和随机数进行计算, 然后分别和每个子设备节点传来的校验信息进行比较, 不同的即为被篡改设备来进行改进. 或者也可以通过预先验证一轮, 使其同构设备中大多数可信.

(3) 本文 Verifier 因为要验证所有的管理节点, 所以在设备类型比较多的情况下, 可能成为性能瓶颈, 而且网络传输量也比较大. 可以通过让管理节点之间互相验证, 最终由初始节点统一上传验证结果, 但是这样的话, 也会带来问题, 比如会增加隐私泄露, 特别是对医疗设备来说, 攻击一个设备就可以获取到其他设备的信息, 造成更大的破坏. 同时需要管理节点有足够的内存去存储集群设备内其他节点的信息, 因为每种设备的配置不一致, 所以有些设备可能没有足够的能力去存储别的设备的内存信息. 而且管理节点之间要相互等待, 是串行的, 耗时也会相应增加.

(4) 本文的实验场景是在同组内子设备不更新或者同时更新的情况下 (比如智能医疗设备), 但对于一些经常需要更新的场景 (比如安装不同的软件) 则不适用. 如果管理节点和子设备节点是同构设备的情况下, 可能需要将设备操作传输给管理节点, 因为是同构设备, 所以管理节点可以根据收到的操作进行相同操作, 然后进行验证.

6 结语

本文针对物联网集群设备远程证明和验证的问题, 提出了一种高效集群设备可信性证明安全方案, 该方案通过采取安全验证策略, 可以检测设备存在的软件攻击. 该方案通过将设备分组, 然后基于管理节点再进行验证, 提高了集群认证的效率. 方案的安全性评估结果表明, 它能够防御集群证明场景下的常用安全攻击. 但是该方案有些方面有待提高, 比如物联网设备无法抵御运行时攻击如 ROP 攻击以及不能抵御通过缺失探测检测不出的物理攻击, 如侧信道攻击等.除此之外, 本方案假设密钥交换时存在可信信道等以及扩展到通用模型中各种场景下的问题, 未来将重点研究这些方面的安全问题.

参考文献

1 广州华天正科技. Real210 LINUX 用户手册. https://wenku.baidu.com/view/19d6b7955ef7ba0d4a733bba.html.

2 Koscher K, Czeskis A, Roesner F, *et al*. Experimental security analysis of a modern automobile. Proceedings of 2010 IEEE Symposium on Security and Privacy. Berkeley/Oakland, CA, USA. 2010. 447–462. [doi: 10.1109/SP.2010.34]

3 Hernandez G, Arias O, Buentello D, *et al*. Smart nest thermostat: A smart spy in your home. Black Hat. Las Vegas, NV, USA, 2014.

4 Illera AG, Vidal JV. Lights off! The darkness of the smart meters. Black Hat Europ. Amsterdam, Netherlands. 2014.

5 Zonouz S, Rrushi J, McLaughlin S. Detecting industrial control malware using automated PLC code analytics. IEEE Security & Privacy, 2014, 12(6): 40–47. [doi: 10.1109/MSP.2014.113]

6 Trusted Computing Group. TPM Main: Part 1 Design Principles, version 1.2, revision 116. Beaverton, OR, USA: Trusted Computing Group, 2011.

7 国家密码管理局. 可信计算密码支撑平台功能和接口规范. 国家密码管理局公告 (第 13 号). 2007.

8 Anati I, Gueron S, Johnson S, *et al*. Innovative technology for CPU based attestation and sealing. Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy. ACM. New York, NY, USA. 2013.

9 Seshadri A, Luk M, Shi E, *et al*. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. ACM SIGOPS Operating Systems Review, 2005, 39(5): 1–16. [doi: 10.1145/1095809]

10 Seshadri A, Perrig A, Van Doorn L, *et al*. Swatt: Software-based attestation for embedded devices. Proceedings of 2004 IEEE Symposium on Security and Privacy. IEEE. 2004. 272–282. [doi: 10.1109/SECPRI.2004.1301329]

11 Li Y, McCune J M, Perrig A. SBAP: Software-based attestation for peripherals. In: Acquisti A, Smith SW, Sadeghi AR, eds. Trust and Trustworthy Computing. Trust 2010. Lecture Notes in Computer Science, vol 6101. Springer, Berlin, Heidelberg. 2010. 16–29. [doi: 10.1007/978-3-642-13869-0_2]

12 Seshadri A, Luk M, Perrig A, *et al*. SCUBA: Secure code update by attestation in sensor networks. Proceedings of the

5th ACM Workshop on Wireless Security. ACM, 2006. 85–94. [doi: 10.1145/1161289.1161306]

13 Castelluccia C, Francillon A, Perito D, *et al*. On the difficulty of software-based attestation of embedded devices. Proceedings of the 16th ACM Conference on Computer and Communications Security. ACM, 2009. 400–409. [doi: 10.1145/1653662.1653711]

14 Eldefrawy K, Tsudik G, Francillon A, *et al*. SMART: Secure and minimal architecture for (establishing dynamic) root of trust. Proceedings of the 19th Annual Network and Distributed System Security Symposium. San Francisco, CA, USA. 2012. 1–15.

15 Koeberl P, Schulz S, Sadeghi AR, *et al*. TrustLite: A security architecture for tiny embedded devices. Proceedings of the Ninth European Conference on Computer Systems. ACM, 2014. 10. [doi: 10.1145/2592798.2592824]

16 Brasser F, El Mahjoub B, Sadeghi A R, *et al*. TyTAN: Tiny trust anchor for tiny devices. 2015 52nd Design Automation Conference (DAC). ACM/EDAC/IEEE. San Francisco, CA, USA. 2015. 1–6. [doi: 10.1145/2744769.2744922]

17 TrustZone. CoreLink System Design Kit webinar: How to implement a secure IoT system on Armv8-M. https://developer.arm.com/technologies/trustzone/webinar-how-to-implement-a-secure-iot-system-on-armv8-m.

18 Schulz S, Sadeghi AR, Wachsmann C. Short paper: Lightweight remote attestation using physical functions. Proceedings of the Fourth ACM Conference on Wireless Network Security. ACM. Hamburg, Germany. 2011. 109–114.

19 Kong J, Koushanfar F, Pendyala PK, *et al*. PUFatt: Embedded platform attestation based on novel processor-based PUFs. Proceedings of the 51st Annual Design Automation Conference. ACM. San Francisco, CA, USA. 2014. 1–6. [doi: 10.1145/2593069.2593192]

20 Asokan N, Brasser F, Ibrahim A, *et al*. Seda: Scalable embedded device attestation. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM. Denver, CO, USA. 2015. 964–975. [doi: 10.1145/2810103.2813670]

21 Abera T, Asokan N, Davi L, *et al*. C-FLAT: Control-flow attestation for embedded systems software. Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. ACM. Vienna, Austria. 2016. 743–754. [doi: 10.1145/2976749.2978358]

22 Zeitouni S, Dessouky G, Arias O, *et al*. Atrium: Runtime attestation resilient under memory attacks. Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design. Irvine, CA, USA. 2017. 384–391.

23 Gruss D, Lettner J, Schuster F, *et al*. Strong and efficient cache side-channel protection using hardware transactional memory. Proceedings of the 26th USENIX Security Symposium. Vancouver, BC, Canada. 2017.

24 Ahrenholz J. Comparison of CORE network emulation platforms. Military Communications Conference, 2010-MILCOM 2010. IEEE. 2010. 166–171. [doi: 10.1109/MILCOM.2010.5680218]

25 Ibrahim A, Sadeghi A R, Tsudik G, *et al*. DARPA: Device attestation resilient to physical attacks. Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks. ACM. Darmstadt, Germany. 2016. 171–182. [doi: 10.1145/2939918.2939938]