Application Note

TI 蓝牙芯片 CC2642R-Q1 使用中的配对问题讨论

Simon Huo *Wireless Connectivity Solution*

摘要

在基于 CC2642R-Q1 的蓝牙低功耗（BLE）系统开发中，配对与绑定问题因其多因素耦合特性，成为困扰开发者的典型技术瓶颈。这类问题集中爆发于连接建立的中后期阶段：向上关联蓝牙基础参数配置与连接事件管理机制，向下直接影响加密信道中的用户业务逻辑执行。其成因常涉及连接事件调度冲突、服务发现机制异常、内存资源分配不足等多维度因素，且核心处理逻辑高度依赖于供应商协议栈的封装实现。

当前汽车电子领域更面临双重挑战：一方面需满足 ISO 21434、WP.29 R155 等严苛的安全合规要求，另一方面又受限于嵌入式平台的计算资源，需通过轻量化软件架构实现多标准兼容。这使得故障根因定位与解决方案设计面临严峻挑战——开发者往往需要在协议栈黑盒、实时性约束、安全规范三重限制下开展问题溯源。

本文将以 CC2642R-Q1 芯片的配对绑定实现机制为核心，结合蓝牙核心规范（Bluetooth Core Specification）的技术要求，系统梳理典型异常场景的工程调试经验，重点阐述：

1.差异化配对模式的安全特性与实现要点

2. MIC failure/DHKey check fail/PIN or Key missing 等高频问题的诊断路径

3.提出一种应对加密资源冲突的轻量化解决方案

内容

插图清单

表格清单

1 蓝牙配对技术原理剖析

蓝牙低功耗技术中的设备配对是构建安全通信通道的核心机制，其本质在于通过加密链路建立防止无线信号被非法窃听的防护体系。在开放性的无线通信环境中，蓝牙设备传输的数据包存在被邻近未授权设备截获的固有风险，虽然物理层面的信号拦截难以彻底杜绝，但通过应用加密算法可有效保障数据内容的机密性。该安全机制的实施基础依赖于通信双方在初始交互阶段完成密钥材料的协商与交换，这正是配对流程的核心价值所在——它不仅需要协调生成用于数据加密的会话密钥，还需共同确定加密算法、密钥长度等安全参数。

1.1 LE Legacy 配对与 LE Secure Connections 配对

当前蓝牙协议栈中主要存在两种技术实现路径：LE Legacy 配对作为在《Bluetooth Core Specification》v4.0 中引入的基础安全方案，通过临时密钥（Short Term Key, STK）生成机制实现数据保护；而 LE Secure Connections 作为在《Bluetooth Core Specification》v4.2 中引入的安全方案，则基于更先进的椭圆曲线加密算法（ECDH），在密钥协商过程中提供更强的抗中间人攻击能力，标志着蓝牙安全体系从传统对称加密向非对称加密架构的重要演进。更多地，请参考阅读《Bluetooth Core Specification》和《TI BLE5-Stack user guide》（GAP Bond Manager 和 LE Secure Connections 部分）。

1.2 关联模式安全特性分析

如上文所提，LE legacy 模式下设备支持 3 种关联模式，LE secure connection 下支持 4 种关联模式。两种协议版本的关键差异在于密钥生成机制的安全基座：LE Legacy 依赖易被破解的短期密钥交换，而 LE Secure Connections 通过密码学强度更高的 ECDH 算法实现永久级前向安全。开发者可强制启用安全连接模式，从而在协议层规避 LE Legacy 的已知漏洞。进行了蓝牙连接的两个设备会在连接建立的初期协商双方采取的关联模式。具体遵循的规则请见 GAP Bond Manager and LE Secure Connections — SimpleLink™ CC13XX/CC26XX SDK BLE5-Stack User's Guide 2.2.11.00 documentation。BLE 设备支持的关联模式共分为四种，他们的特性和区别如下表所示：

表 1-1. 四种关联模式对比

关联模式	Just Works	Passkey Entry	Numeric Comparison	Out of Band
核心机制	无用户验证，配对自动完成。	用户在一个设备输入另一个设备显示的 6 位数字密码。	双方设备显示 6 位数字，用户确认两者是否相同。	通过非 BLE 方式（如 NFC、QR 码）预先安全交换配对信息。
用户交互	无	用户必须输入密码。	用户只需**确认**显示的数值是否匹配。	通常只需让设备靠近或扫描一次。
安全级别	最低	中到高	高	最高
抵御 MITM 攻击	无	要求攻击者实时知道并输入正确的密码才能成功实施 MITM。	要求攻击者同时篡改两个设备显示给用户的数值且不被发现。	依赖 OOB 通道的安全性（通常物理安全）。
所属连接方式	Legacy Pairing / Secure Connections	Legacy Pairing / Secure Connections	Secure Connections	Legacy Pairing / Secure Connections
密钥生成基础	STK/LTK	STK/LTK	LTK	LTK
适用场景	对安全性要求不高的设备；或无输入/输出能力的设备。	一个有显示能力（显示密码），另一个有输入能力（键盘输入密码）的设备。	双方都有显示能力的设备	需要最高安全性或追求极致便利性的场景；双方支持相同 OOB 技术的设备。

在关联模式的安全性排名中，Just Works 的安全性最低，而 OOB 方式的安全性最高。系统将根据双方设备支持的能力，自动选择最安全的关联模式进行配对。有关完整的安全关联模式选择流程及详细规范，请参阅蓝牙核心规范版本 v5.2 [Vol 3] Part H, Section 2.3.5.1 的表 2.7 和表 2.8。

1.3 隐私保护与地址管理

蓝牙设备通过其蓝牙设备地址（Bluetooth Device Address, BDA）进行标识。然而，为了防止被追踪，BLE 设备可以启用隐私功能，通过定期生成新的地址来替代固定的标识地址，从而有效隐藏设备的身份地址。具体而言，蓝牙设备的地址类型主要包括以下几种：

- 公开地址（Public Address）：使用设备的永久身份地址，该地址是固定不变的。
- 随机静态地址（Random Static Address）：设备在每次上电时生成一个随机的地址，此地址在设备工作期间不会发生变化。
- 可解析私有地址（Resolvable Private Address, RPA）：设备定期生成一个随机地址，需要通过设备的身份解析密钥（IRK）来解析为其真实身份地址，增强隐私性。
- 不可解析私有地址（Non-resolvable Private Address）：设备定期生成一个完全随机的地址，且无法解析到设备的真实身份地址，用于某些不需要身份验证的场景。

有关私有地址生成及解析的详细流程，请参阅《Bluetooth Core Specification》和《TI BLE5-Stack user guide》（GAP Bond Manager 和 LE Secure Connections 部分）以获得更深入的信息。

2 配对加密的常见问题

在上述配对场景中的配对问题，既有显见和易于排查的问题，也有较为复杂，根本原因不显然的问题。了解蓝牙配对问题以及其成因，能够帮助开发者更高效地定位故障，并提高设备的连接稳定性和交互可靠性。以下，我们将梳理一些在日常应用与调试中，经常遇到的蓝牙配对典型疑难问题。

2.1 完整性校验故障——MIC 校验失败(MIC failure)

MIC (Message Integrity Check) 是用来验证蓝牙通信数据完整性的重要工具。通常为 4 字节，在链路加密的情况下作为消息完整性的校验使用。这类错误可能的原因包括：

- 密钥不匹配：通信双方可能使用了不同的加密密钥（比如长期密钥或短期密钥 - LTK/STK）。
- 数据包被篡改：传输过程中，通信数据包可能因为噪声、干扰或恶意篡改而变得不完整或不一致。
- 协议栈实现问题：底层实现存在 BUG，导致在加密和解密处理中产生不一致的数据。
- 超时或通信冲突：通信双方处于不同的会话状态，导致 MIC 验证失败。
- 加密资源冲突问题：对于 CC2642 而言，rpa 解析和配对/加密流程均需要占用 crypto engine，该 crypto engine 如果在配对请求处理时被其他请求占用，可能导致加密资源冲突，导致 MIC 验证资源错误。

可以看到，MIC error 虽然表面上是 4 字节的校验问题，但是问题的发生点可能存在于链路的从空中数据包交互到芯片资源管理的各个流程。

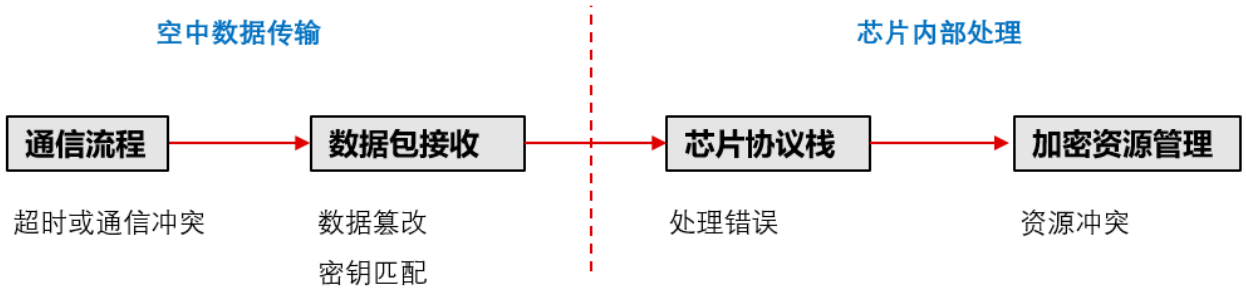

图 2-1. MIC failure 可能的问题场景

对于这类问题，建议采用分层递进的诊断策略，从物理层到协议层逐步深入：

- 测试射频环境：在干扰较小的环境（屏蔽房或安静测试环境）中进行复现实验，排除信号干扰。同时对问题发生的环境进行射频检查，如周边 WiFi 或其他 2.4 GHz 设备的冲突，是否有特殊蓝牙数据包等等。

- 检查通信参数：捕获通信过程中双方设备协议事件（使用抓包工具如 Ellisys、nRF Sniffer）验证通信参数是否匹配。蓝牙抓包工具的使用可以结合对芯片射频 TX/RX 的开启状态检查，帮助确认问题发生时通信的双方设备是否处于射频无响应的状态，以及是否在接收端接收到了被篡改或错误的数据包。
- 检查密钥生成：跟踪配对流程中的密钥协商过程，验证 LTK/STK 生成算法一致性，调试检查密钥交换阶段随机数生成器输出值，交叉验证加密计数器（IV）同步状态。
- 同步双方加密状态：解析空中数据包中的配对特征交换（Pairing Feature Exchange）字段

对照蓝牙协议规范验证加密模式协商流程。

案例一：

图 2-2. 案例一空中包 sniffer log

在本例的实际空中包中，可以清晰地看到在 pairing feature exchange 阶段，central 节点与 peripheral 节点协商使用 passkey entry 方式进行配对。而在 SMP Authentication stage 1 中，peripheral 节点对来自 central 的第一包 pairing random 数据反馈了拒绝继续执行加密指令。这一 rejection 及其错误码反映了 peripheral 认为来自 central 的 paring random 数据是错误的。我们在实践中，就可以以检查验算 paring random 包中所包含的 Na 数据是否正确为切入点，如果该值与前文 Ca, Cb 符合计算要求，则 peripheral 节点这里做了一个错误的判断，后续 debug 将重点调查 peripheral 节点为何计算错误；若不符合计算要求，则 central 端在对 Ca, Cb 的计算中出现了问题。

- 检查加密资源占用情况：对 CC2642，它包含一个 AES security module 可以以硬件资源加速计算加解密流程。详见《TI BLE5-Stack user guide》。在常见的蓝牙场景中，AES security module 的应用可以发生在连接前对对端设备地址（RPA 类型地址）的解析，连接中配对及加密流程，以及可能的应用层的 hash 计算。

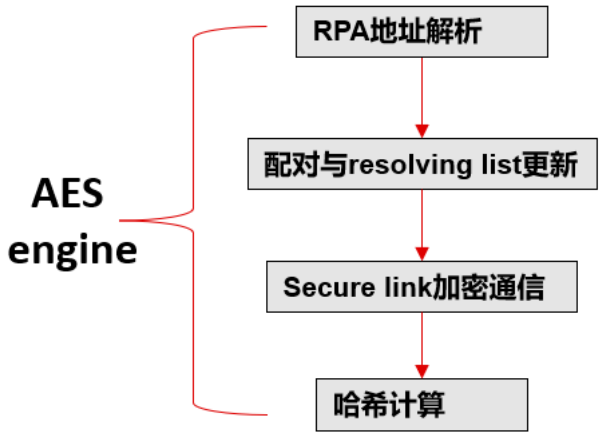

图 2-3. AES 硬件资源覆盖的流程

因此对此类原因的排查应考虑到当前蓝牙应用所涉及的所有场景，若环境复杂无法简单分析，则应在代码中 AES engine 调用处添加 log 或 IO 翻转，用实验的方式检查当前应用是否有预期之外的加密资源使用。并结合 log 出现的时间和位置判断问题原因。添加 IO 翻转的示例方法如下：

在 AESCCMCC26xx.c 中的 AESCCM_startOperation 函数中进行修改，（其中 IO8 和 9 仅为示例应用的 IOID）

```
#include <inc/hw_gpio.h>
#include <driverlib/ioc.h>

static int_fast16_t AESCCM_startOperation(AESCCM_Handle handle,
                                          AESCCM_Operation *operation,
                                          AESCCM_OperationType operationType)
{
IOCPinTypeGpioOutput( IOID_8);
IOCPinTypeGpioOutput( IOID_9);

GPIO_writeDio(IOID_8, 1);

  DebugP_assert(handle);
 …
 …
  /* Load the key from RAM or flash into the key store at a hardcoded and reserved
location */
  if (AESWriteToKeyStore(keyingMaterial, keyLength, AES_KEY_AREA_6) != AES_SUCCESS)
{
  GPIO_writeDio(IOID_9, 1);

  /* Release the CRYPTO mutex */
  SemaphoreP_post(&CryptoResourceCC26XX_accessSemaphore);
 GPIO_writeDio(IOID_9, 0);

  return AESCCM_STATUS_ERROR;
  }
  …
…
  AESStartDMAOperation(operation->input, operation->inputLength, operation->output,
operation->inputLength);
GPIO_writeDio(IOID_8, 0);
return AESCCM_waitForResult(handle);

}
```

案例二：

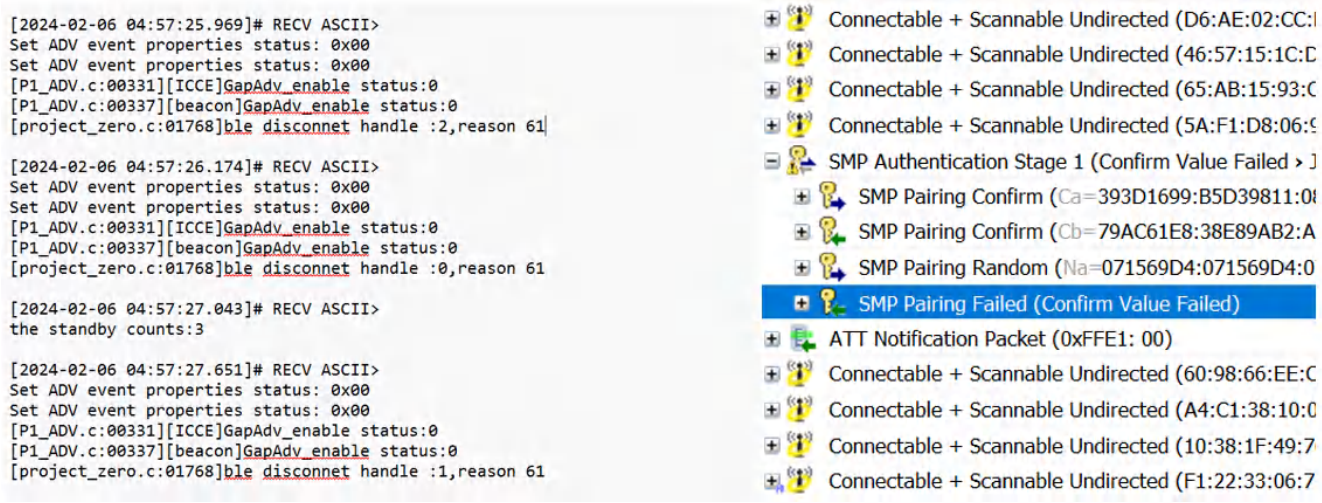

图 2-4. 案例二 UART log 与空中包 sniffer log

在本例的 log 中，加密连接连续出现 0x3D 的断连 error code，即为 0x3D MIC failure。重启后虽然可以立刻恢复，但很快又会发生，客户反馈问题发生时 cc2642 仅连接了一个测试设备，且测试设备运行基本示例程序，没有额外哈希运算或重复加密。我们将空中包 sniffer log，逻辑分析仪抓到的 aes engine 启动 log，和 uart 输出的调试 log 在同一时间进行比对，发现问题发生处（一个加油站）拥有非常大量的蓝牙 rpa 地址在进行扫描，一旦扫描到了 CC2642，就会尝试建立连接，在问题发生处 AES engine 大量的调用时刻也和被扫描的时刻基本吻合。最终问题定位为调用解析 rpa 地址的操作过于频繁，导致了加密操作分配不到加密资源。问题也以调整了被扫描时连接

的逻辑而得到了解决。这对于我们系统设计也起到了非常重要的启发作用，正如 debug 要对外界环境造成的影响十分警惕，设计一个健壮的蓝牙系统也应充分考虑外界的特殊环境兼容性。

2.2 密钥协商异常—— DHKey 校验失败(DHKey check fail)

BLE 配对过程中的 DH Key（Diffie-Hellman 密钥）验证失败，本质是安全连接建立的核心环节——ECDH（椭圆曲线 Diffie-Hellman）密钥协商流程异常。此类故障虽常表现为密钥不匹配，但在实际应用中，我们往往面对的并非狭义上的 DH Key 验证流程本身出现了密钥对不匹配问题。更多时候，问题发生于密钥解析时芯片的其他行为对密钥验证造成的干扰，我将主要的参考排查方向按照应用侧影响和协议栈侧影响分为两类，列举如下：

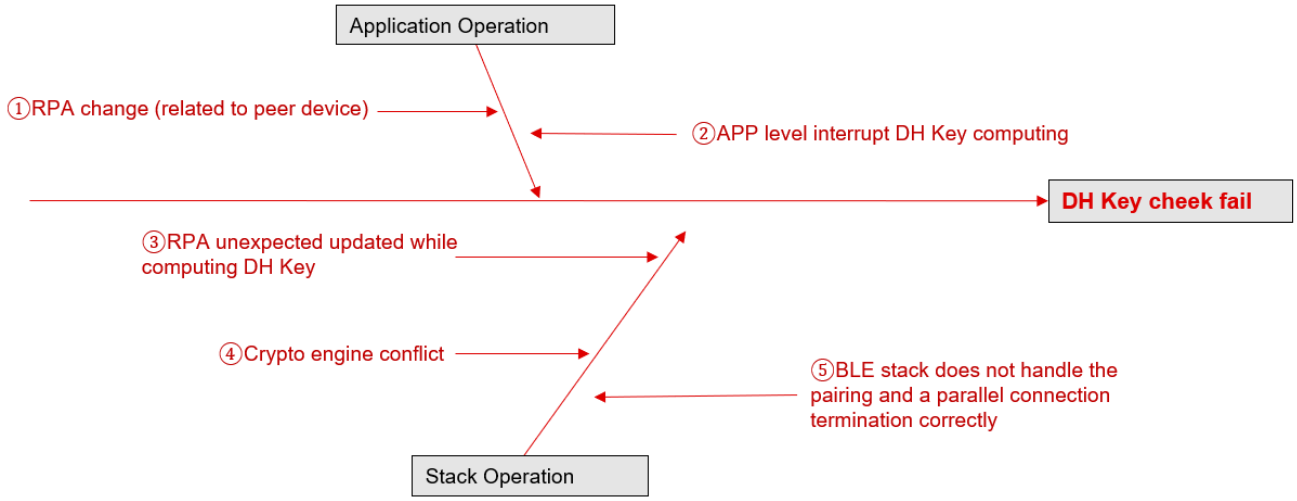

图 2-5. DH Key check fail 根因分析

同样地，在调试时应从外向内对应问题发生位置的逻辑关系进行排查：

- 测试射频环境：在干扰较小的环境（屏蔽房或安静测试环境）中进行复现实验，排除信号干扰。同时对问题发生的环境进行射频检查，如周边 WiFi 或其他 2.4 GHz 设备的冲突，是否有特殊蓝牙数据包等等。
- 验证安全连接支持：通过 Ellisys 嗅探器捕获全链路交互数据，重点关注：Pairing Public Key 交换阶段的公钥数据完整性; 双方是否同步启用 LE Secure Connections 模式;ECDH 算法标识符（如 P-256 曲线）的协商一致性.对比双方协议日志，验证公钥字节序处理方式是否符合规范（大端/小端）,密钥派生计数器（Counter）的同步状态
- 检查问题发生时芯片的异常行为：如是否发生了断连（⑤），是否发生了地址变化（①③），是否有复位或高优先级中断（②）。

案例三（Solved issue of TI BLE stack）：

问题复现：

设备 A 作为主节点与 CC2642 连接并配对，手机 B 作为主节点与 CC2642 反复连接和断连，偶现 A 设备的配对发生 DH Key check fail.

问题根本原因：

在连接断开的时候，蓝牙协议栈未检查断连的 connection handle 是否与正在进行配对的 connection handle 相同，进而导致协议栈错误删除了正在进行配对的设备的信息，从而在 DH key 匹配运算时找不到对应的公钥，进而发生 DH Key check fail.

对该问题的分析始于发现配对记录发生了丢失，进而通过添加 AES engine 的 log 来检查问题发生时的删除配对操作发生源与发生信息，并定位到时间上与断连行为恰好同时。该问题已修复于 SimpleLink CC13xx CC26xx SDK v7.10.02.23。并以修复 patch 的方式横展到所有旧版本 SDK。

案例四（ Solved issue of APP code）：

问题复现：

配对过程进行中时，CC2642 设备（或对端）的 rpa 发生变化。即发生 DH key check fail.

问题根本原因：

在进行 DH key chek 时以新的 rpa 进行运算，而非连接建立时的 rpa，这看上去是符合蓝牙逻辑的，但是实际上违反了蓝牙官方文档对配对时地址的定义。事实上，在 Bluetooth Core Specification Vol 3 Part H Section 2.2.7 中有明确规定："*BD_ADDR_C is the device address of the Central and BD_ADDR_P is the device address of the Peripheral. The device addresses are the values used during connection setup. The least significant bit in the most significant octet in both BD_ADDR_C and BD_ADDR_P.*"因此，问题的解决方法即是添加一个专门的单元存储连接建立时的地址。

- 检查加密资源占用情况：与第二节的讨论相同, 检查加密引擎抢占与 RPA 解析、扫描响应等操作的硬件资源竞争（如 CC2642 AES 模块）

2.3 密钥管理缺陷——PIN/Key 丢失(PIN or Key missing)

此错误既可能出现在配对失败时，通常表示设备无法找到配对所需的 PIN 码或密钥。也可能出现在已配对的两个设备进行加密时，通常表示设备无法找对所需的 LTK。

2.3.1 配对阶段密钥缺失

对配对中出现的 PIN or Key missing 问题，应当从配对码的协商->生成->交互->处理的逻辑，追踪配对码的芯片->空中->对端芯片路径，结合空中包与设备打印信息，重点关注：

- 设备间配对模式不匹配：双方配对模式不一致（如 Passkey Entry vs Just Works）安全参数请求（Security Request）与配对能力（Pairing Capabilities）不兼容。
- 配对初始化错误：设备的安全需求初始化未成功完成，导致未生成或未保存 PIN/密钥。无法与对端设备正常配对。
- 数据包被篡改：传输过程中，通信数据包可能因为噪声、干扰或恶意篡改而变得不完整或不一致。
- 配对流程错误：如对端设备与本设备有配对记录，但已在本设备删除且未与对端设备重启配对流程，导致因本地没有可用的密钥而配对失败。可以测试多种配对模式（如 Just Works、Passkey Entry、OOB Pairing）。

2.3.2 加密阶段 LTK 丢失

而已配对的加密过程中出现的 PIN or Key missing 问题，一般呈现出如下特点：

- 非系统级设计问题，较难从简单的工程逻辑梳理中找到漏洞并弥补。
- 偶发性强：实验室复现率通常低于 1%。
- 环境相关性：与客户端应用逻辑（如快速地址刷新）存在潜在耦合。
- 自愈机制干扰：一般情况下，对于配对失败或加密失败类问题，问题发生后设备会以删除配对记录并在下次连接时重新配对的方式进行自修复。因此问题发生时的核心现场情况可能已被破坏，造成信息获取困难。

因此在处理这类问题时，应把握 PIN or Key missing 错误码的核心逻辑。对于 TI CC2642 而言，唯一的在加密过程中上报 PIN or Key missing 错误的理由就是找不到 LTK。因此依然可以从引用侧代码影响和协议栈侧影响两个维度针对 LTK 丢失的可能性给出排查方向：

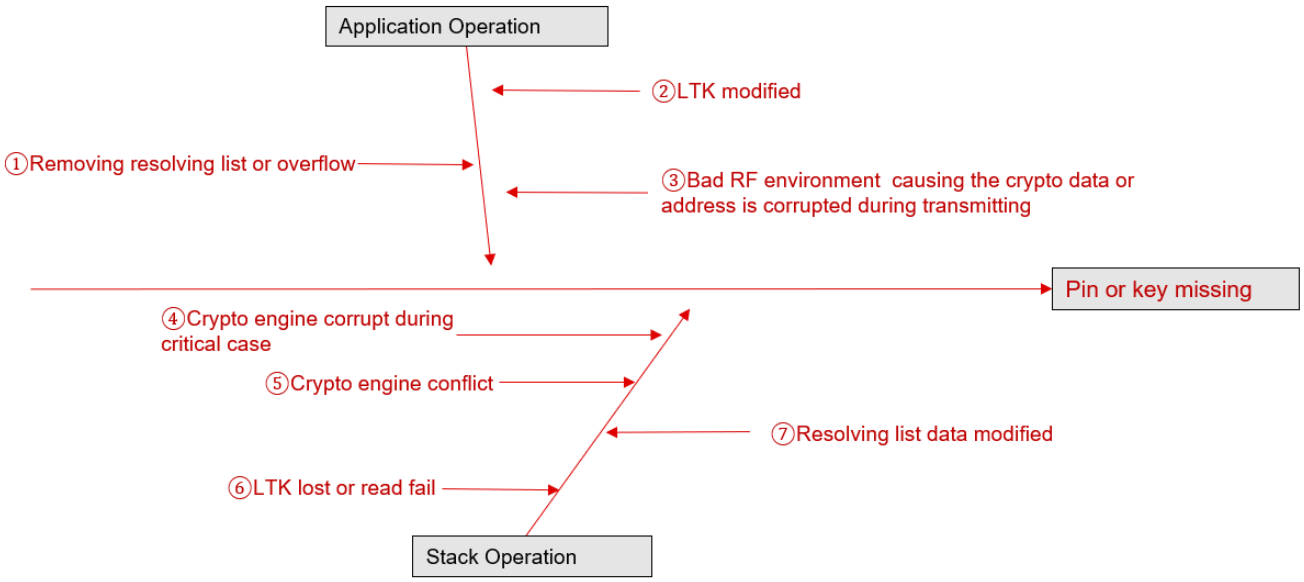

图 2-6. PIN or Key missing 根因分析

- 检查问题是否来自物理意义上的 LTK 丢失：检查问题发生前后的 flash 存储区域，读取 flash 中 LTK 的存储位置检查是否发生了错误、篡改。并相应检查操作源。（②⑥）
- 测试射频环境：在干扰较小的环境（屏蔽房或安静测试环境）中进行复现实验，排除信号干扰。同时对问题发生的环境进行射频检查，如周边 WiFi 或其他 2.4 GHz 设备的冲突，是否有特殊蓝牙数据包等等。（③）
- 测试问题发生时 RAM 中用于解析对端地址的 resolving list 是否发生了错误、篡改：由于配对过程中未交互的地址变化会导致配对时找不到当前进行配对操作的对端地址，因此要检查配对操作的合法性。（①⑦）

案例五：

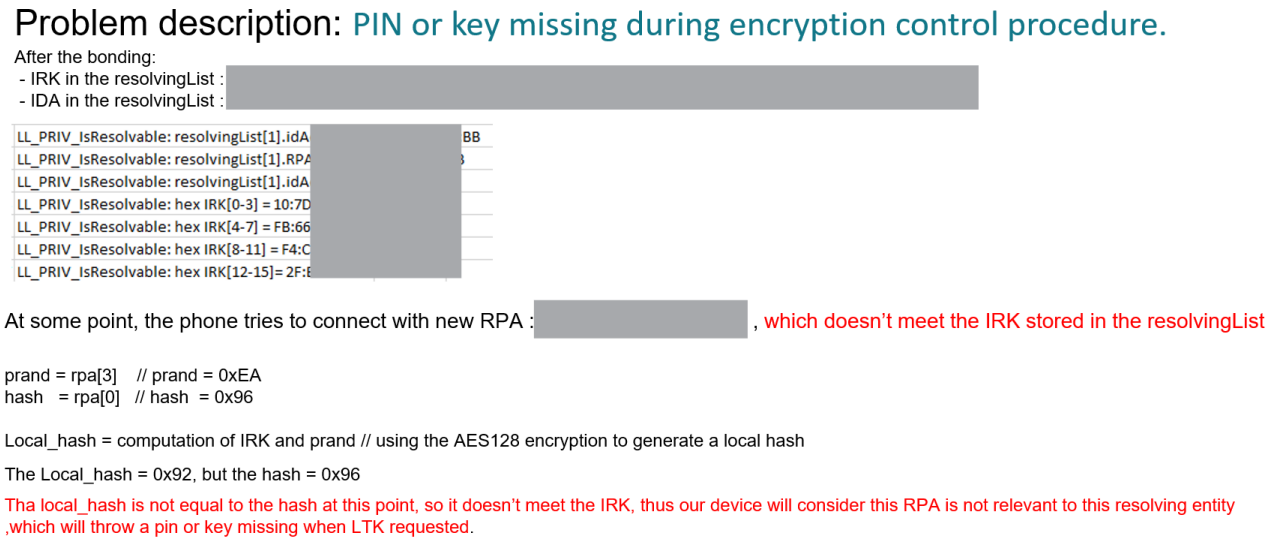

图 2-7. 案例五 sniffer log 地址分析

在上图的例子中，手机作为主节点与 CC2642 进行连接、配对和绑定，但在反复的断线重连中偶现 PIN or Key missing 加密失败。通过检查问题发生时的异常情况，我们发现手机作为主节点不仅更新了自己的 RPA，且更换了 RPA 的生成 hash 值。这一变动使得 CC2642 所记录的手机 IRK 无法再将当前的手机识别为一个已经配对的系统，因而产生了 PIN or key missing 问题。

- 检查加密资源占用情况：与第二节的讨论相同（⑤）。

- 其他关联应用：在一个实际工程中，对对端设备的识别有时候不仅与蓝牙配对绑定相关，还涉及客户所需要通过的其他规范或验证来识别。如果其中有在加密事件中进行的用户校验流程，也有可能由该校验错误导致 PIN or Key missing 问题。

案例六：

客户在使用 CC2642 时，设计了如下的两步验证系统：

问题复现：

1. 配对 CC2642 与外部 central 节点->
2. Central 节点下发自身所带的鉴权信息->
3. CC2642 将配对绑定信息（下称 R1）与鉴权信息（下称 R2）共同存储在对应此 central 节点的 flash 中
4. 在反复的断连回连中偶现 PIN or key missing

问题根本原因：

对该问题的解析一开始是局限于配对绑定信息是否丢失以及是否有读取错误的，但是在反复地验证存储和读取对于成熟的 CC2642 芯片而言不太会发生的情况下，最终我们将视线聚焦到了 R2 和 R1 的不同上。CC2642 的 flash 驱动在删除某存储信息的时候分为两步

1. 将该信息的原存储位置有效位置为无效
2. 重新以该信息的 header 写入一次该信息，但内容为全 0

而对用户自己操作的 R2 信息，未进行重写全 0 的操作，这会导致当发生了配对记录删除的情况下，R1 仍是有效的，只是值为全 0，但 R2 是无效的。此时若发生了一次新的配对，其新 R1 和 R2 所对应的 handle 会无法匹配。导致客户的第二步校验失败，进而导致 PIN or key missing.

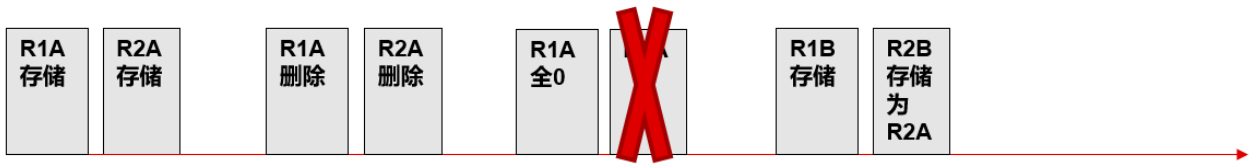

图 2-8. 案例五根本原因流程分析

3 一种加密资源冲突解决方案

基于前述各类配对异常的分析可见，CC2642 的加密资源竞争问题已成为系统性安全问题的核心之一。该芯片的硬件加密模块采用独占式架构设计，需同时承载 RPA 地址解析、配对密钥协商、链路数据加密等关键任务，而应用层既无法实时监控其负载状态，也难以实施优先级调度策略。针对这一硬件限制，本文提出软件化 AES 加密协同方案——通过构建可观测、可调度的软件加密通道，与硬件模块形成互补架构。该方案充分利用 CC2642 的 Cortex-M4 内核运算能力，在应用层实现以下创新设计：

- 加密任务分级机制：可将 RPA 解析等实时性要求高的操作保留在硬件模块执行，而配对流程中的 ECDH 计算、LTK 派生等非实时任务迁移至软件 AES 实现。
- 动态资源仲裁器：基于 FreeRTOS 的优先级继承特性，提供了检测到硬件模块过载时可选软件通道分流的可能。
- 混合加密流水线：对 AES-CCM 加密链路进行指令级优化，该混合架构不仅规避了硬件资源冲突风险，更通过软件层的状态可视化接口，为开发者提供加密任务时序分析、冲突事件追溯等高级调试能力，显著提升复杂射频环境下蓝牙连接的安全性和鲁棒性。

3.1 环境选择与配置

为了在 FreeRTOS 上实现 AES-256 加密，需要完成以下工作：

- 选择加密库，在 FreeRTOS 环境中，可以选择轻量级加密库，如 mbedTLS、WolfSSL 或 TinyAES。本文使用 TinyAES，因为它轻量、易用，并且特别适合嵌入式系统。
- 从 TinyAES GitHub 项目 中下载源码。TinyAES 支持 ECB（电子密码本）、CBC（密码分组链接）及 CTR（计数器模式）等模式，本文示例使用 CBC 模式。
- 在 FreeRTOS 工程目录中引入以下文件：aes.h/aes.c
- 修改 FreeRTOS 的配置：确保 FreeRTOS 的配置文件（FreeRTOSConfig.h）中堆栈和任务优先级设置足够支持加密操作，例如可根据系统需求调整堆栈大小，可提供足够的总内存大小.

3.2 使用 TinyAES 库实现 AES-256 加密和解密

- 添加 AES 加密任务的实现

```
 encrypt_task.c
#include "FreeRTOS.h"
#include "task.h"
#include "aes.h"
#include <stdio.h>
#include <string.h>

#define AES_BLOCK_SIZE 16

// AES-256 密钥（必须为 32 字节）
static uint8_t key[32] = {xx};

// 初始化向量（IV），必须为 16 字节
static uint8_t iv[AES_BLOCK_SIZE] = {yy};

void EncryptTask(void *pvParameters) {
  uint8_t plaintext_block[AES_BLOCK_SIZE] = {0};
  memcpy(plaintext_block, plaintext, strlen(plaintext));

  uint8_t ciphertext_block[AES_BLOCK_SIZE] = {0};
  uint8_t decrypted_block[AES_BLOCK_SIZE] = {0};

  // AES CBC 加密上下文
  struct AES_ctx ctx;
  AES_init_ctx_iv(&ctx, key, iv);

  // 加密
  AES_CBC_encrypt_buffer(&ctx, plaintext_block, AES_BLOCK_SIZE);
  memcpy(ciphertext_block, plaintext_block, AES_BLOCK_SIZE);

  // 解密（重新初始化上下文以确保 IV 被重置）
  AES_init_ctx_iv(&ctx, key, iv);
  AES_CBC_decrypt_buffer(&ctx, ciphertext_block, AES_BLOCK_SIZE);
  memcpy(decrypted_block, ciphertext_block, AES_BLOCK_SIZE);

  vTaskDelay(pdMS_TO_TICKS(1000)); // 任务延时
```

- 在主函数中创建任务

```
 main.c
#include "FreeRTOS.h"
#include "task.h"
#include "encrypt_task.h"

int main(void) {
  // 初始化硬件
  // 创建加密任务
  xTaskCreate(EncryptTask,      // 任务函数
              "Encrypt Task",   // 任务名称
              512,              // 堆栈大小（字节）
              NULL,             // 任务参数
              1,                // 优先级（任务越重要越高）
              NULL);       // 任务句柄
  // 启动 RTOS 调度器
  vTaskStartScheduler();
}
```

该方法的优势在于：

- 轻量级的软件库提供了几乎不损耗 CPU 运算和存储能力的加密选择。事实上，TinyAES 库占用代码非常轻量，在不含浮点操作的情况下大约需要 ~2-3 KB 的代码空间（包括 AES 实现和初始化向量支持），这同样取决于选定的加密模式（CBC 模式比 CTR 模式稍多代码）。应用代码若包含关键的打印语句（如 printf）和本地数据，大约会占用 ~2-4 KB 的额外 Flash 和~2.5KB 的额外 RAM。
- 软件方法允许开发者根据具体需求优化算法。例如，通过查表法优化混淆、引入混沌映射增强密钥安全性，或动态调整密钥生成策略。硬件实现通常固化算法逻辑，难以修改。采用不同的软件库可以支持更多的加密模式（如 ECB、CBC、CTR 等），开发者可根据需求灵活选择并扩展 AES engine 所不支持的加密算法。
- 软件实现的加密逻辑可通过固件升级（如 OTA）快速修复漏洞或更新算法，而硬件加速器需如前文介绍，调整整个应用的设计逻辑以寻求规避。成本更大。

3.3 系统级优化探索

选择使用软件方法进行加解密工作，相比起简单地调用 TI 实例代码开放的资源，需要讨论一些对于应用开发增加的系统负担和优化效果，包括：

3.3.1 内存分配检查

- 堆栈大小和任务优先级：确保分配足够的堆栈空间给加密任务，避免任务在执行加密算法时崩溃，并根据任务的重要性设置合适的优先级。
- 在资源受限的系统上，尽量复用内存区域存储加密数据，避免过多动态内存分配。可考虑根据所需的算法使用动态分配的缓冲区处理更大长度的明文和密文。考虑到内存分析在设计软件中的重要地位，以示例代码为例提供 RAM 资源估算的方法如下：

1. FreeRTOS 内核的内存占用

   任务控制块（TCB）：每个任务需要的 TCB 结构，包括任务栈指针、状态信息和任务优先级等。TCB 的大小通常约为 60-100 字节。

   任务堆栈：每个任务的栈空间。需要注意的是，栈的大小应设置为足够运行任务所需指令，包括中断服务、函数调用的局部变量以及中间操作等。示例代码中，为 EncryptTask 分配了 512 字 的栈空间（xTaskCreate() 中指定为 512 x 4 字节 = 2 KB）。

   全局堆空间（如果使用 pvPortMalloc() 动态分配模式）：FreeRTOS 内核需要动态内存分配实现，例如队列、信号量或其他功能所需的内存。这部分需求取决于具体的系统实现，但在本示例中不涉及额外的动态分配。
2. TinyAES 库的 RAM 占用

   AES 上下文结构（struct AES_ctx）：包含密钥和初始化向量，用于执行加密和解密操作。约 60 字节。

   AES 分组密钥扩展（密钥调度算法使用的内部变量）：对于 AES-256，共需存储 240 字节的密钥扩展数据。
3. 应用数据

应用代码中有全局数据：AES 密钥（32 字节）、初始化向量 IV（16 字节）、明文、密文和解密缓冲区每个分配了 16 字节（例子中是对齐的 AES 块大小）。总计 96 字节。

3.3.2 填充处理与线程安全

如果明文数据不是 AES 块大小的倍数（16 字节），需要手动添加填充逻辑，例如 PKCS7 填充。如果堆栈实际需求小于 512 字，可以缩小任务堆栈大小。例如，通过堆栈分析工具 (uxTaskGetStackHighWaterMark) 评估任务运行时的栈使用情况，逐渐调整到合适大小（典型 AES 任务可能只需 300 字左右堆栈）。另外，TinyAES 库本身不是线程安全的。如果多个任务同时调用加密算法，可以使用任务间同步机制（如 Mutex）。

4 总结

本文简明扼要地剖析了蓝牙低功耗（BLE）配对安全机制的核心原理与典型故障模式，针对一些常见疑难问题系统性地构建了从协议逻辑到硬件资源管理的全链路诊断框架，并结合实例给出了系统设计开发者的设计建议。同时针对 TI CC2642R-Q1 芯片常见的几种疑难配对加密问题中共性的加密资源冲突难题，提出了一种软件替代的高自由度解决方案实现资源竞争规避。为系统设计开发者提供了以软硬协同的加密配对方案应对系统设计需求的可行性。

5 参考文档

1. Bluetooth Core Specification v4.2 tiny-AES-c
2. CC13x2, CC26x2 SimpleLink Wireless MCU Technical Reference Manual (Rev. G)

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月