App 与嵌入式通讯协议 0904

🟦 PIN 配对流程（🚨 含新增提案）

配对流程子阶段一：设备正品认证流程(本流程已在嵌入式蓝牙中实现，用于正品防伪校验，确保设备来源可信)

步骤	方向	📩 数据内容	✍️ 用途	⚙️ 说明
1️⃣ (起点命令)	📱→🔐 App→Device	`"request"`	请求用户在场确认	utf-8→base64
1️⃣	🔐→📱 Device→App	`ID:<HEX>`	设备发起加密认证	连接后立即推送，HEX 字符串
2️⃣ (解码方法不会开源)	📱→🔐 App→Device	`ID:<APP解密后的HEX>`	回传解密 ID	utf-8→base64 编码
3️⃣	🔐→📱 Device→App	`"VALID"`	正品认证通过	明文
4️⃣	📱→🔐 App→Device	`"validation"`	响应认证通过	utf-8→base64 编码

配对流程子阶段二：设备解锁与 PIN 验证

（⚠️ 嵌入式设备必须在收到 `"validation"` 之后，才会生成并下发 `PIN:<随机验证码>,<Y/N>`）

步骤	方向	📩 数据内容	✍️ 用途	⚙️ 说明
5️⃣ （未解锁 → 必须先解锁）	🔐(锁屏状态)→📱	`PIN:<随机验证码>,<Y/N>` 例："PIN:6375,Y"	本地解锁 + 显示验证码	收到 `"request"` 后立即亮屏提示用户输入解锁 PIN（仅设备端处理，不经 BLE 传输）；解锁成功后生成并 **只在屏幕显示一次性验证码**（如 6 位数），并通过 BLE 下发 `"PIN:xxxx,Y/N"` ；⚠️ Y=已有钱包，N=未创建/导入钱包；超时 60 秒则作废。
6️⃣ （解锁状态）	🔐(解锁状态)→📱	`PIN:<随机验证码>,<Y/N>`	显示验证码	解锁状态下直接生成一次性验证码，并 **只在设备屏幕显示**，同时通过 BLE 下发 `"PIN:xxxx,Y/N"` ；⚠️ 必须由用户手动输入到 App，不能跳过。
7️⃣	🔐→📱 Device→App	`PIN_OK` / `PIN_FAIL`	验证结果（App 内验证）	成功进入授权会话；失败按失败策略处理

配对流程子阶段三：地址、公钥与 AccountId 提案

步骤	方向	📩 数据内容	✍️ 用途	⚙️ 说明
8️⃣	📱→🔐 App→Device	`address:<chainName>`	获取链地址	utf-8→base64，建议分批发送，指令间留短间隔；⚠️ 在 `PIN_OK` 后由 App 主动发起
9️⃣	🔐→📱 Device→App	`<prefix><address>`	返回链地址	明文，如 `ETH:0x...`
🔟	📱→🔐 App→Device	`pubkey:<chain>,<hdpath>\n`	获取链公钥	utf-8→base64，结尾 `\n` ；⚠️ 在 `PIN_OK` 后由 App 主动发起
1️⃣1️⃣	🔐→📱 Device→App	`pubkeyData:<chain>,<pubkey>`	返回链公钥	明文
🚨1️⃣2️⃣ [新增提案]	📱→🔐 App→Device	`"REQ_ACCOUNT_ID"`	请求设备返回 accountId	⚠️ 在 `PIN_OK` 后由 App 主动发起；保持请求-响应一致性
🚨1️⃣3️⃣ [新增提案]	🔐→📱 Device→App	`accountId:<基于Mac地址+序号生成的ID>`	返回设备内冷钱包的 accountId	⚠️ 新增功能，征求嵌入式团队确认。accountId 需独立性，避免设备冲突；建议初始化时生成并固定存储。App 在后续签名、NFT 发送、地址确认等操作必须携带此值。

> 📌 特别说明：第 5️⃣ 步与第 6️⃣ 步 PIN 验证码返回格式
>
> - 格式示例："PIN:<随机验证码>,<标志位>"
> - 例如："PIN:6375,Y" 或 "PIN:6375,N"
> - 标志位说明：
> - Y → 设备中已有有效钱包（已经完成 Create Wallet 或 Import Wallet 流程）。
> - N → 当前设备没有有效钱包（可能还没完成 Create Wallet 或 Import Wallet 流程）。

> 📌 特别说明：第 🔟 步与第 1️⃣1️⃣ 步 pubkey 的用途
>
> - pubkey 仅在 App 的 signTransaction.js 文件 中使用。
> - 它作为参数发送给 服务器，用于构造部分链（如 Cosmos、Ripple 等）的 预签名数据请求。
> - 不会再次下发给嵌入式设备，设备只负责接收 App 下发的 presign 数据 并进行签名。

> 📌 特别说明（请马总嵌入式团队确认）
>
> - 新增 `accountId` 的目的是让 App 可以唯一识别不同的冷钱包设备。
> - 该 ID 的生成方式提议为 **基于设备 MAC 地址 + account 序号**，确保唯一性和持久性。
> - 该方案需要嵌入式团队确认是否：
>   - **合理性** —— 是否满足唯一性与业务需求。
>   - **安全性** —— 是否存在隐私泄露或可预测风险，是否需增加随机因子或哈希处理。
>   - **实现复杂度** —— 在固件端是否易于实现，是否会增加额外存储或计算负担。
> - 目前这是 **提案阶段**，需要嵌入式团队确认是否采纳。

📌 特别说明（关于 accountId 的安全性）

> ⚠️ 是否能通过程序获取？
>
> - **可以获取**：
>   - 任何正常对接设备的 App，都能读取 `accountId`（因为这是协议的一部分）。
>   - 如果有人写了一个恶意 App，也可以通过和设备通信来拿到 `accountId`。
> - **但不能通过 `accountId` 攻破钱包**：
>   - `accountId` ≠ 私钥，它只是一个 **“身份证号码”**。
>   - 即使攻击者知道了 `accountId`，也无法伪造签名，因为签名仍需设备内部私钥完成。
>   - 最多能做的是 **“冒充设备 ID”**，但设备会在校验时拒绝外部伪造的 `accountId`。

签名（即交易发送数字货币）流程 📝🖋️

🟧 BLE 交易签名协议流程（🧯 含新增提案,并且已经明确标注了 结尾符号 \r\n）

步骤	方向	📨 数据内容	✍️ 用途	⚙️ 说明
🧯0️⃣ **[新增提案 · 征求嵌入式确认]**	📱→🔐 App→Device	`accountId:<ID值>\r\n` 例如：`accountId:5F60789A\r\n`	指定要确认的钱包账户	App 在发起地址确认前必须告诉设备，此次操作属于哪个 `accountId`（设备初始化时随机生成并固定存储）。
🧯0.1️⃣ **[新增提案 · 征求嵌入式确认]**	🔐→📱 Device→App	`PIN_SIGN_READY` / `PIN_SIGN_FAIL` / `PIN_SIGN_CANCEL`	用户解锁结果	⚠️ **仅在设备锁屏状态下触发**。• 锁屏 → 设备无法查询自身的 accountId → 必须先提示用户在设备端输入 PIN 解锁。• 解锁成功 → 返回 `PIN_SIGN_READY`，可继续流程。• 解锁失败/取消 返回 `PIN_SIGN_FAIL` 或 `PIN_SIGN_CANCEL`，流程终止。
🧯0.2️⃣ **[新增提案 · 征求嵌入式确认]**	🔐→📱 Device→App	`ACCOUNT_ID_OK` / `ACCOUNT_ID_FAIL`	返回 accountId 校验结果	⚠️ **解锁状态下设备可以读取自身的 accountId**。• 若与 App 的 accountId 一致 → 返回 `ACCOUNT_ID_OK` 并继续操作。• 一致 → 返回 `ACCOUNT_ID_FAIL` 并拒绝操作。
1️⃣	📱→🔐 App→Device	`"destinationAddress:<付款地址>,<收款地址>,<手续费>,<链标识>\r\n"`	下发交易主要参数（第一步）	App 仅在收到 `ACCOUNT_ID_OK` 后才发送。例如：`"destinationAddress:0x123abc...,0x456def...,100000,ethereu` utf-8 编码后 base64 发送
4️⃣	🔐→📱 Device→App	`"PIN_SIGN_READY"` / `"PIN_SIGN_FAIL"` / `"PIN_SIGN_CANCEL"`	用户密码验证结果	明文，表示用户在设备端 PIN 校验的结果
5️⃣	🔐→📱 Device→App	`"Signed_OK"` / `"Signed_REJECT"`	设备确认交易参数	明文：`"Signed_OK"` 表示设备同意处理交易；`"Signed_REJEC` 表示用户拒绝，App 应立即终止流程
6️⃣	📱→🌐 App→Server	POST `{chain, from, to, txAmount, ...}`	获取 nonce、gasPrice 等预签名参数	App 向后端 API 发 POST 请求，获取当前链的参数

步骤	方向	📨 数据内容	✍️ 用途	⚙️ 说明
7️⃣	📱→🌐 App→Server	POST encode 接口请求体	获取 presign (预签名数据)数据（hex/json）	根据链类型调用 encode 接口，返回对应预签名数据
8️⃣	📱→🔐 App→Device	`"sign:<链标识>,<BIP44路径>,<presign数据>\r\n"`	下发预签名数据	utf-8 编码后 base64 发送；需分包，每包 ≤200 字节，最后加 `\r\n`，分包之间建议 ≥5ms 间隔
9️⃣	🔐→📱 Device→App	`"signResult:<签名数据>"` 或 `"signResult:ERROR"`	返回最终签名结果或错误	明文，如 `"signResult:0xf86b..."` 为签名数据，如失败返回 `"signResult:ERROR"`
🔟	❓→🔐 App→Device	`"BCAST_OK,<链标识>,<txHash>"` 或 `"BCAST_FAIL,<链标识>,<错误码>\r\n"`	通知设备广播结果	utf-8→base64 编码发送，`"BCAST_OK"` 表示广播成功并附带易哈希；`"BCAST_FAIL"` 表示广播失败并附带原因码

> 📌 特别说明（请马总嵌入式团队确认）
>
> - 新增了两个步骤（2️⃣、3️⃣）：
>   - 2️⃣ **App 下发** `accountId` → 指定签名请求属于哪个账户。
>   - 3️⃣ **设备返回校验结果** → 如果匹配返回 `"ACCOUNT_ID_OK"`，如果不匹配返回 `"ACCOUNT_ID_FAIL"` 并拒绝操作。
> - 目的：防止 App 向错误的钱包账户发起签名，确保多账户场景下安全性与准确性。

从 App 发送 NFT 到嵌入式流程 🖼️🔗（🚨 含新增提案）

步骤	方向	📨 数据内容	✍️ 用途	⚙️ 说明
🚨0️⃣ **[新增提案 · 征求嵌入式确认]**	📱→🔐 App→Device	`accountId:<ID值>\r\n` 例如：`accountId:5F60789A\r\n`	指定要确认的钱包账户	App 在发起地址确认前必须告诉设备，此次操作属于哪个 `accountId`（设备初始化时随机生成并固定存储）。
🚨0.1️⃣ **[新增提案 · 征求嵌入式确认]**	🔐→📱 Device→App	`PIN_SIGN_READY` / `PIN_SIGN_FAIL` / `PIN_SIGN_CANCEL`	用户解锁结果	⚠️ **仅在设备锁屏状态下触发**。• 锁屏 → 设备无法查询自身的 accountId → 必须先提示用户在设备端输入 PIN 解锁。• 解锁成功 → 返回 `PIN_SIGN_READY`，可继续流程。• 解锁失败/取消 → 返回 `PIN_SIGN_FAIL` 或 `PIN_SIGN_CANCEL`，流程终止。
🚨0.2️⃣ **[新增提案 · 征求嵌入式确认]**	🔐→📱 Device→App	`ACCOUNT_ID_OK` / `ACCOUNT_ID_FAIL`	返回 accountId 校验结果	⚠️ **解锁状态下设备可以读取自身的 accountId**。• 若与 App 传入的 accountId 一致 → 返回 `ACCOUNT_ID_OK` 并继续操作。• 若不一致 → 返回 `ACCOUNT_ID_FAIL` 并拒绝操作。
1️⃣	📱→🔐 App→Device	`"DATA_NFT_TEXT_{字节数}SIZE"`	NFT 名称传输头	App 仅在收到 `ACCOUNT_ID_OK` 后才发送。在发送 NFT 名称正文之前，App 必须先告诉设备名称长度。`{字节数}` = NFT 名称的 **utf-8 字节长度**。整个字符串需再 base64 发送。例如：NFT 名称 `"CryptoPunk#1"` 长度 = 12 → 明文头部 = `DATA_NFT_TEXT_12SIZE` → base64 = `REFUQV9ORlRfVEVYVDEyU0laRQ==`。
2️⃣	🔐→📱 Device→App	`"GET1"`, `"GET2"`, …	请求下一个 NFT 名称分包	设备每次请求一包（最多 200 字节）
3️⃣	📱→🔐 App→Device	NFT 名称的各个包（base64）	分包发送 NFT 名称正文	把 NFT 名称按 ≤200 字节切片，每一片做 **utf-8 → base64** 后发送。最后一包必须加 `\r\n`，分包之间建议 ≥5ms 间隔。
示例：NFT 名称 `"CryptoPunk#1"` → utf-8 长度 = 12 → base64 = `Q3J5cHRvUHVua0Mx\r\n`				
4️⃣	🔐→📱 Device→App	`"FINISH"`	NFT 名称传输结束	明文
5️⃣	📱→🔐 App→Device	`"DATA_NFT_IMG_{字节数}SIZE"`	NFT 图片传输头	{字节数} 为图片 base64 字节数。utf-8 转 base64 发送

步骤	方向	📨 数据内容	✍️ 用途	⚙️ 说明
6️⃣	🔐→📱 Device→App	`"GET1"`，`"GET2"`，…	请求下一个 NFT 图片分包	每次最多 200 字节
7️⃣	📱→🔐 App→Device	NFT 图片的各个包（base64）	分包发送 NFT 图片	把 NFT 图片的 **原始 base64 数据** 按 ≤200 字节切片，每片再发送。最后一包必须加 `\r\n`，分包之间建议 ≥5ms 间隔。

示例：如果图片原始 base64 =

`/9j/4AAQSkZJRgABAQEASABIAAD...` (共 520 字节)

→ 分 3 包：

- 包 1 (0~199)：`/9j/4AAQSkZJRgABAQEASABIAAD...`
- 包 2 (200~399)：`...继续...`
- 包 3 (400~519)：`...结尾...\r\n` |

| 8️⃣ | 🔐→📱 Device→App | `"FINISH"` | NFT 图片传输结束 | 明文 |

> 📌 特别说明（请马总嵌入式团队确认）
>
> - 新增了 0️⃣、0.1️⃣ 步骤：
>   - 0️⃣ App 下发 `accountId` → 指定收藏 NFT 属于哪个钱包。
>   - 0.1️⃣ 设备校验并返回结果 → 一致返回 `"ACCOUNT_ID_OK"`，不一致返回 `"ACCOUNT_ID_FAIL"`。
> - 目的：避免 NFT 被错误存储到其他账户。
> - 所有分包传输均需注意：**最后一包追加 `\r\n`，前面分包不加**；分包之间建议 ≥5ms 间隔。
> - 征求嵌入式确认：是否同意此机制？是否需要定义更详细的错误码（例如 `"ACCOUNT_ID_MISSING"`、`"ACCOUNT_ID_INVALID"`）？

从 App 发送“确认地址”命令到嵌入式流程 🏦🔎

步骤	方向	📨 数据内容	✍️ 用途	⚙️ 说明
🧯0️⃣ **[新增提案·征求嵌入式确认]**	📱→🔐 App→Device	`accountId:<ID值>\r\n` 例如：`accountId:5F60789A\r\n`	指定要确认的钱包账户	App 在发起地址确认前必须告诉设备，此次操作属于哪个 `accountId`（设备初始化时随机生成并固定存储）。
🧯0.1️⃣ **[新增提案·征求嵌入式确认]**	🔐→📱 Device→App	`PIN_SIGN_READY` / `PIN_SIGN_FAIL` / `PIN_SIGN_CANCEL`	用户解锁结果	⚠️ **仅在设备锁屏状态下触发**。• 锁屏 → 设备无法查询自身的 accountId → 必须先提示用户在设备端输入 PIN 解锁。• 解锁成功 → 返回 `PIN_SIGN_READY`，可继续流程。• 解锁失败/取消 → 返回 `PIN_SIGN_FAIL` 或 `PIN_SIGN_CANCEL`，流程终止。
🧯0.2️⃣ **[新增提案·征求嵌入式确认]**	🔐→📱 Device→App	`ACCOUNT_ID_OK` / `ACCOUNT_ID_FAIL`	返回 accountId 校验结果	⚠️ **解锁状态下设备可以读取自身的 accountId**。• 若与 App 传入的 accountId 一致 → 返回 `ACCOUNT_ID_OK` 并继续操作。• 若不一致 → 返回 `ACCOUNT_ID_FAIL` 并拒绝操作。
1️⃣	📱→🔐 App→Device	`verify:<chainName>\r\n`	地址显示请求命令	App 仅在收到 `ACCOUNT_ID_OK` 后才发送。例如 `verify:bitcoin\r\n`；需以 `\r\n` 结尾，utf-8→base64 发送。
2️⃣	🔐→📱 Device→App	`Address_OK`	地址显示完成反馈	用户在设备端确认后返回。明文，不带 `\r\n`。
3️⃣	🔐→📱 Device→App	`Address_FAIL`	地址确认失败或异常	如用户拒绝、超时或设备异常，统一返回 `Address_FAIL`。

📌 特别说明（请马总嵌入式团队确认）

> 新增了 0️⃣、0.1️⃣ 步骤，即 App 必须下发 accountId，设备需返回校验结果。目的：确保展示的地址属于正确的钱包账户，避免多账户场景下误显示。建议错误码："ACCOUNT_ID_OK"：验证通过"ACCOUNT_ID_FAIL"：验证失败，拒绝继续请确认：是否采纳此机制？是否需要更细粒度的错误码（如 "ACCOUNT_ID_MISSING"、"ACCOUNT_ID_INVALID"）？

命令与链类型映射举例

币种	显示命令
BTC 比特币	`verify:bitcoin`
ETH 以太坊	`verify:ethereum`
TRX 波场	`verify:tron`
SOL 索拉纳	`verify:solana`
COSMOS	`verify:cosmos`
...	其他见 assetRouteDefs

OTA 升级流程 ⚙️⬆️

步骤	方向	📩 数据内容	✍️ 用途	⚙️ 说明
1️⃣	📱→🔐 App→Device	`"DATA_OTA_<文件字节数>SIZE\r\n"`	固件头部	例：`"DATA_OTA_163840SIZE\r\n"`，utf-8→base64 发送，必须以 `\r\n` 结束，通知设备准备接收
2️⃣	📱→🔐 App→Device	固件内容分包，每包 200 字节，HEX 字符串	固件分包数据	每包 HEX 字符串（200 字节），utf-8→base64 发送；**只有最后一包需追加 `\r\n`**；分包之间建议 ≥5ms 间隔
3️⃣	🔐→📱 Device→App	`"OTA_OK"` / `"OTA_FAIL"`	设备确认接收或异常	明文返回，不需要 `\r\n`；表示升级完成或异常

📌 特别说明

> 所有 App→Device 的命令必须以 \r\n 作为结束符。分包传输时：前面的分包不加 \r\n，仅最后一包追加 \r\n。

🗂️ 字段示例与流程说明

- **固件头部示例**：
  `DATA_OTA_12032SIZE`
  - 发送内容为 `DATA_OTA_12032SIZE` → utf-8 转 base64
- **固件分包示例**：
  - 第 1 包 HEX 为 `aabbcc...`（200 字节，HEX 字符串长度 = 400）
  - 发送内容为 `aabbcc...` → utf-8 转 base64
- **分包机制**：
  - 固件全部数据，循环 offset 每 200 字节，分多包下发
  - App 无需等设备 GET 或确认，可 **顺序连续** 写入

```
+--------------------------------+

| 正在升级...                    |
| ------------------------------ |
| 已发送: 3/48 包 ( 7% )         |
| \[████▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒]    |
| 请勿断开设备或关闭 App         |
| +------------------------------+ |
```