App与嵌入式通讯协议0903

App 与嵌入式通讯协议 0903

从 App 接到蓝牙配对命令 🚀

🟧 设备正品认证流程

步骤	方向	📩 数据内容	✍️ 用途	⚙️ 说明
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	📱→🔐 App→Device	`ID:<解密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

🟦 PIN 配对及钱包操作命令（🧯 含新增提案）

步骤	方向	📩 数据内容	✍️ 用途	⚙️ 说明
5️⃣	📱→🔐 App→Device	`"request"`	请求用户在场确认	utf-8→base64
5.1	（Device 设备锁屏状态下 🔐）	嵌入式内部通讯	本地解锁 + 显示验证码	收到 "request" 后立即亮屏提示用户在设备上输入解锁 PIN（仅设备端处理，不经 BLE 传输）；解锁成功后生成并显示一次性验证码（如 6 位数字）；若 60 秒内未完成解锁，则不显示验证码并结束本次操作（App 端可提示超时）
5.2	（Device 设备解锁状态下）	嵌入式内部通讯	显示验证码	收到 "request" 后直接生成一次性验证码（如 6 位数字）并显示在设备屏幕上，等待用户输入到 App
7️⃣	📱→🔐 App→Device	`codeValue:<用户输入>`	回传用户输入的验证码	utf-8→base64
8️⃣	🔐→📱 Device→App	`PIN_OK` / `PIN_FAIL:<reason>`	验证结果	通过进入授权会话；失败按失败策略处理
9️⃣	📱→🔐 App→Device	`address:<chainName>`	获取各链地址	utf-8→base64，建议分批发送、指令间保留短间隔
🔟	🔐→📱 Device→App	`<prefix><address>`	返回链地址	明文，如 `ETH:0x...`
1️⃣1️⃣	📱→🔐 App→Device	`pubkey:<chain>,<hdpath>\n`	获取部分链公钥	utf-8→base64，结尾 `\n`
1️⃣2️⃣	🔐→📱 Device→App	`pubkeyData:<chain>,<pubkey>`	返回链公钥	明文
🧯1️⃣3️⃣ **[新增提案 · 征求嵌入式确认]**	🔐→📱 Device→App	`accountId:<随机生成并固定的ID>`	**返回设备内冷钱包的 account id**	⚠️ **此功能为新增提案，特此征求马总嵌入式团队的意见。**要求：accountId 需具备随机性，避免不同设备产生相同值；建议在设备初始化时生成并存储，后续固定返回。请马总确认此方案是否合理，以及是否需要调整实现方式。

ℹ️ 补充说明

设备生成并固定存储 `accountId`，并在 PIN 配对阶段首次返回给 App。之后 App 必须在所有关键操作请求（如交易签名、NFT 收藏、地址确认）里带上该值，设备收到后进行校验。

> 📌 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 需要具有随机性，避免设备之间重复。
> - 目前这是 **提案阶段**，需要嵌入式确认是否合理、是否符合安全要求、以及实现复杂度是否可接受。

📌 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`。

从 App 接到确认签名对命令 📝🖋️

🟧 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
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）	例如：`"destinationAddress:0x123abc...,0x456def...,100000,ethereum"` utf-8 编码后 base64 发送
🧯2️⃣ **[新增提案 · 征求嵌入式确认]**	📱→🔐 App→Device	`"accountId:<固定随机ID>\r\n"`	**指定签名请求对应的钱包账户**	App 必须告诉设备：此次签名属于哪个 `accountId`，以便设备在多账户场景下准确定位钱包。`accountId` 为设备初始化时随机生成并固定存储。
🧯3️⃣ **[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**	⚠️ 如果 `accountId` 一致 → 返回 `"ACCOUNT_ID_OK"` 并继续流程；如果不一致 → 返回 `"ACCOUNT_ID_FAIL"`，拒绝签名。请马总嵌入式团队确认此方案是否合理，以及错误码格式是否需要进一步标准化。
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_REJECT"` 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
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，最后一包加 `\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交易哈希；`"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 接到收藏 NFT 命令 🖼️🔗（🧯 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:<固定随机ID>\r\n"`	**指定收藏 NFT 对应的钱包账户**	App 必须在发起 NFT 收藏前告诉设备：此次操作属于哪个 `accountId`。该值在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	`"ACCOUNT_ID_OK"` / `"ACCOUNT_ID_FAIL"`	**返回 accountId 校验结果**	⚠️ 如果 `accountId` 一致 → 返回 `"ACCOUNT_ID_OK"` 并继续流程；如果不一致 → 返回 `"ACCOUNT_ID_FAIL"`，拒绝操作。请嵌入式团队确认是否采用此机制，以及错误码是否需要标准化。
1️⃣	📱→🔐 App→Device	`"DATA_NFT_TEXT_{字节数}SIZE"`	NFT 名称传输头	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
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 接到“确认地址”命令 🏦🔎

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:<固定随机ID>\r\n"`	**指定确认地址对应的钱包账户**	App 必须在发起地址确认前告诉设备：此次操作属于哪个 `accountId`。该值在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	`"ACCOUNT_ID_OK"` / `"ACCOUNT_ID_FAIL"`	**返回 accountId 校验结果**	⚠️ 如果一致 → 返回 `"ACCOUNT_ID_OK"` 并继续流程；不一致 → 返回 `"ACCOUNT_ID_FAIL"` 并拒绝操作。需嵌入式确认是否采用此机制，以及错误码是否需要更细化。
1️⃣	📱→🔐 App→Device	`"verify:<chainName>\r\n"`	显示地址请求命令	例如：`"verify:bitcoin\r\n"`，utf-8→base64 发送；必须以 `\r\n` 作为结束符
2️⃣	🔐→📱 Device→App	`"Address_OK"`	地址显示完成反馈	明文返回，不需要 `\r\n`
3️⃣	🔐→📱 Device→App	其他提示/错误码	异常/扩展命令	如遇异常可扩展 `"Address_FAIL"` 等命令，明文返回

📌 **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

从 App 接到 OTA 固件升级命令 ⚙️⬆️

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          |
| +---------------------------+ |
```