Proposta de Integração - Pix por Aproximação

Autor: Paulo Neves

Data: 03/07/2025

Objetivo

Este documento tem por objetivo apresentar uma proposta de integração de pagamentos via **Pix por aproximação**, utilizando uma aplicação mobile desenvolvida em **React Native**, com um módulo nativo escrito em **Kotlin**, capaz de se comunicar via **NFC** e utilizando os protocolos definidos no documento *"Especificações do Pix por Aproximação para Android"*, fornecido pelo **Banco Central do Brasil (Bacen)**.

Solução Proposta

1. Desenvolver um aplicativo capaz de:
2. Detectar a aproximação de um dispositivo com **NFC** apto a realizar pagamentos via **Pix por aproximação**.
3. Realizar a comunicação utilizando o protocolo **Application Protocol Data Unit (APDU)**.
4. Finalizar o processo com a confirmação do pagamento do Pix.

Considerações

Durante o desenvolvimento da solução, é importante compreender que há duas partes envolvidas:

1. **Terminal**: é a aplicação que estamos desenvolvendo. Ela é responsável por iniciar a comunicação, transmitir os dados e confirmar o recebimento do Pix.
2. **Recebedor**: é o lado que responde à solicitação. De forma simplificada, pode ser considerado o banco pagador.

Arquitetura da Solução

Componentes principais:

- **Aplicativo React Native**: Camada de interface e lógica principal da aplicação. É responsável por acionar o módulo NFC, capturar a interação do usuário e iniciar a jornada de pagamento.
- **Módulo Nativo Kotlin (Android)**: Camada responsável pela comunicação com o hardware NFC. É ele quem executa os comandos da comunicação entre o terminal e o dispositivo HCE utilizando o protocolo **APDU**.
- **Dispositivo HCE (ex: app bancário)**: Atua como um cartão virtual NFC, preparado para responder a comandos recebidos, como SELECT AID e UPDATE BINARY. No contexto do Pix, ele é quem recebe e interpreta os dados da transação.
- **Backend da aplicação**: Sistema responsável por gerar o conteúdo do Pix (em formato de código copia e cola), que será entregue ao módulo NFC para envio ao dispositivo HCE. Assume-se que esse backend já possui a lógica de geração dos dados Pix.

Descrição do fluxo:

1. O aplicativo React Native solicita ao backend o **código Pix copia e cola**, que será usado na transação.
2. Em seguida, ele aciona o módulo nativo Kotlin para iniciar a escuta de dispositivos NFC nas proximidades.
3. Quando um dispositivo HCE é detectado, o módulo envia um comando chamado **SELECT AID**. Esse comando serve para identificar e selecionar o serviço de pagamento que está emulando um cartão no dispositivo aproximado. Se o dispositivo responder corretamente, a comunicação está estabelecida.
4. Após o reconhecimento do AID, o módulo envia outro comando, chamado **UPDATE BINARY**, que basicamente transmite os dados do Pix (em formato URI codificado) ao dispositivo HCE. Esse dado é interpretado como se fosse a “mensagem” que o cartão está esperando receber.
5. Se todos os comandos forem aceitos (confirmados pelo código de resposta 9000), o processo é considerado concluído com sucesso, e o dispositivo HCE pode processar a transação.

Sobre os comandos APDU:

- **SELECT AID**: Um comando padrão usado para selecionar uma aplicação específica (neste caso, a aplicação Pix) no dispositivo NFC. Ele garante que o terminal está falando com o "cartão certo".
- **UPDATE BINARY**: Comando utilizado para enviar dados para o dispositivo NFC. Neste fluxo, ele carrega o código Pix codificado como uma URI NDEF.

Detalhamento dos Comandos APDU

SELECT AID (Application Identifier)

O primeiro passo para realizar uma comunicação bem-sucedida com um dispositivo HCE via NFC é o envio do comando **SELECT AID**. Esse comando é responsável por **selecionar a aplicação correta** no dispositivo emulado. No caso do Pix, a aplicação responsável por processar os dados da transação.

O SELECT AID **deve ser enviado imediatamente após a detecção da tag NFC** (ou seja, assim que o IsoDep é conectado). Esse foi um ponto **extremamente sensível e difícil de depurar durante o desenvolvimento**.

Se o comando for enviado com atraso ou fora da sequência esperada, o dispositivo HCE pode rejeitar a comunicação silenciosamente, sem fornecer um erro claro. Em nossos testes, observamos que mesmo atrasos mínimos (como algumas centenas de milissegundos) entre a conexão e o envio do SELECT AID **podem resultar em falha de comunicação**.

```
private val SELECT_AID_COMMAND = byteArrayOf(  1 Usage
    0x00, 0xA4.toByte(), 0x04, 0x00, 0x08,
    0xA0.toByte(), 0x00, 0x00, 0x09, 0x40, 0xBC.toByte(), 0xB0.toByte(), 0x00
)
```

Esse comando segue o padrão ISO 7816-4 para seleção por AID. O valor hexadecimal do AID utilizado (A00000000940BCB000) é o definido na especificação oficial do **Pix por Aproximação**, disponibilizada pelo Banco Central do Brasil.

Fluxo correto:

1. O módulo Kotlin detecta um dispositivo NFC compatível (via IsoDep).
2. A conexão é aberta imediatamente (iso.connect()).

3. O comando SELECT AID é enviado **antes de qualquer outro comando**.
4. A resposta do dispositivo é verificada para garantir que os bytes finais sejam 0x90 0x00, indicando sucesso.

Código de validação da resposta:

```
private fun ByteArray.isSuccessfulSelectAid(): Boolean =  1 Usage
    size >= 2 && this[size - 2] == 0x90.toByte() && this[size - 1] == 0x00.toByte()
```

Se a resposta não for bem-sucedida, a aplicação deve rejeitar a sessão NFC com a mensagem apropriada (AID_REJECTED), encerrando a conexão com segurança.

UPDATE BINARY

Após o SELECT AID ser aceito com sucesso pelo dispositivo HCE, o próximo passo é transmitir os **dados da transação Pix,** ou seja, o **código Pix copia e cola**.

Essa transmissão é feita por meio de uma sequência de comandos **UPDATE BINARY**, definidos na especificação NFC para escrita de dados. No nosso caso, o dado transmitido é uma **mensagem NDEF contendo a URI codificada com o Pix**, como por exemplo: "*pix://broker-url.com?qr=000201...6304XXXX*"

Estrutura do comando UPDATE BINARY

Cada comando UPDATE BINARY carrega um pedaço da URI, respeitando o limite de aproximadamente **240 bytes por bloco**. A estrutura básica do comando é:

[CLA, INS, P1, P2, LENGTH, ...DATA]

Onde:

- CLA: 0x00
- INS: 0xD6 (identifica o comando como UPDATE BINARY)
- P1 e P2: posição onde os dados serão escritos (offset)
- LENGTH: tamanho do bloco de dados
- DATA: conteúdo do chunk NDEF

A sequência completa de comandos é montada na função **buildNdefApdus**, que divide a URI em blocos e constrói os APDUs:

```
for (let offset :number  = 0; offset < ndefMessage.length; offset += maxChunkSize) {
  const chunk :Uint8Array  = ndefMessage.slice(offset, offset + maxChunkSize);
  const p1 :number  = (offset >> 8) & 0xFF;
  const p2 :number  = offset & 0xFF;

  apdus.push([0x00, 0xD6, p1, p2, chunk.length, ...chunk]);
}
```

Cada comando UPDATE BINARY deve indicar a posição da memória onde os dados serão escritos. Essa posição é representada por dois campos: **P1** (byte mais significativo) e **P2** (byte menos significativo), que juntos formam o deslocamento (offset) do bloco em relação ao início da memória.

No código, esse deslocamento é calculado da seguinte forma:

- offset representa a posição inicial do bloco dentro da mensagem completa.
- p1 é calculado com (offset >> 8) & 0xFF, o que equivale a pegar os 8 bits mais significativos do offset (divisão por 256).
- p2 é calculado com offset & 0xFF, isolando os 8 bits menos significativos (resto da divisão por 256).

Exemplo:
Se o offset for 512 (em hexadecimal, 0x0200):

- p1 será 2 (0x02)
- p2 será 0 (0x00)

Portanto, o comando UPDATE BINARY irá escrever esse bloco a partir da posição 0x0200 da memória do dispositivo NFC.

Validação da resposta

Assim como no SELECT AID, cada resposta recebida deve ser verificada:

```
function isSuccessResponse(resp: number[]): boolean {  Show usages
  const len :number  = resp.length;
  return len >= 2 && resp[len - 2] === 0x90 && resp[len - 1] === 0x00;
}
```

O par de bytes 0x90 0x00 indica que o dispositivo aceitou o bloco e está pronto para o próximo.

Observações importantes:

- A transmissão deve ser feita **em sequência**, sem longos atrasos entre os comandos.
- Se **qualquer bloco falhar**, o processo inteiro deve ser interrompido e a conexão finalizada com finish().
- Após o envio do último bloco, o terminal deve considerar o processo como concluído.

Extração do Hostname e Limitações de Uso

Para montar corretamente a URI que será enviada ao dispositivo HCE via comando UPDATE BINARY, é necessário conhecer o **hostname da instituição que gerou o código Pix**. Esse hostname é um elemento essencial da URL e deve apontar para o servidor da instituição responsável por processar a cobrança. O hostname pode ser **extraído diretamente do código Pix copia e cola**, mais especificamente do campo que contém a URL de redirecionamento ou consulta. Esse campo normalmente segue o formato: ...2563exemplo.banco.com.br/qr/...

Nesse caso, exemplo.banco.com.br é o hostname que deve ser utilizado para compor a URI no formato: pix://exemplo.banco.com.br?qr=<codigo_pix>

No entanto, é importante destacar que **nem todas as instituições de pagamento estão preparadas para aceitar códigos Pix por aproximação via o modelo copia e cola.** Além disso, o **Pix Saque** e outras modalidades especiais **não estão disponíveis** para pagamento via aproximação. Apenas **cobranças imediatas** (como as geradas em tempo real ou via QR dinâmico) foram aceitas nos testes realizados.

Links

Github

Documentação Banco Central