UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS, TECNOLOGIAS E SAÚDE
DEPARTAMENTO DE COMPUTAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Relatório: Flutter RSA App

Bernardo Pandolfi Costa

Araranguá
22 de fevereiro de 2023

Conteúdo

1 Introdução

Este documento tem como objetivo apresentar os principais aspectos do desenvolvimento do aplicativo, descrevendo etapas percorridas, metodologias e problemas encontrados no desafio proposto pelo processo seletivo do LabSEC 2023 - *Aplicativo Mobile em Flutter*. O relatório apresentará imagens do aplicativo em funcionamento para uma melhor visualização.

2 Aprendizado Teórico

Antes de iniciar o desenvolvimento da aplicação, precisei estudar sobre os assuntos tratados no desafio que não conhecia ou era inexperiente. Investi o período inicial do desafio apenas estudando o funcionamento e a teoria por trás do sistema de criptografia RSA: como foi criado, como é usado, porquê funciona, etc. Através de alguns artigos e vídeos, fui capaz de entender o suficiente para prosseguir ao próximo passo. Na sequência, estudei sobre a relação do sistema RSA com as assinaturas digitais, ou seja, como a criptografia RSA é aplicada para assinatura de documentos online. Com isso, o primeiro dia do período do desafio foi inteiramente dedicado ao aprendizado teórico.

3 Desenvolvimento

A verdadeira dificuldade do desafio começa aqui. Como eu não possuia experiência prévia com a tecnologia Flutter, precisei investir mais tempo para aprender o básico, de forma que o desenvolvimento do aplicativo final começou dias depois da data em que o desafio foi proposto.

3.1 Tela 1: *Home*

Ainda não muito acostumado com flutter, decidi começar a fazer as telas do aplicativo, aprimorando meu conhecimento na prática. Comecei com a *Tela 1*, responsável pela navegação principal no aplicativo. Esta é a tela que mostra os botões para que o usuário possa navegar pelas outras interfaces.
Para desenvolvê-la, apenas elementos básicos de Flutter foram utilizados, como *ElevatedButtons*, *Containers* e a *AppBar*, junto com suas respectivas estilizações.

3.2 Tela 2: *BLE Devices*

Esta é a tela em que a busca por dispositivos BLE acontece. Para o escaneamento, utilizei a biblioteca flutter_blue, recomendada no próprio documento do desafio. Executar a busca em si foi fácil, porém estilizar a lista encontrada foi um pouco mais complexo, sendo necessária a criação de alguns templates, *ListViews*, *Cards*, dentre outros *widgets*. Sempre que uma nova busca é feita, a lista de dispositivos é atualizada no programa e é salva. Um escaneamento novo substituirá a lista antiga.
Para marcar o horário e data da busca mais recente, utilizei o pacote *intl*, específico para lidar com o tipo de dado *DateTime*. Assim, sempre que o *floatingActionButton* for acionado, a função *scan()* é chamada, a informação de data é salva em uma variável com o horário da busca, através da função *now()*.
Enquanto uma busca é feita, uma tela de carregamento toma lugar do campo de resultado, até que a busca seja concluída. Caso o usuário queira cancelar a busca, pode interrompê-lo clicando novamente no botão flutuante no canto inferior direito. Isso não salva uma lista nula no aplicativo, mas sim uma lista com os dispositivos encontrados até o momento da interrupção.

3.3 Tela 3: *Generate RSA Key Pair*

Nesta tela, o usuário pode gerar o par de chaves RSA, usado futuramente na assinatura. Para gerar o par de chaves RSA, utilizei uma biblioteca chamada Pointy Castle, específica para algoritmos de criptografia. Tal pacote possui funcionalidades para a geração de chaves, assinatura e verificação.
Quando o botão *Generate New Pair* é pressionado, o usuário é confrontado por uma tela de *input*, onde ele indica o tamanho de bits da chave. Os tamanhos aceitos estão entre 512 e 4096.
É importante destacar que o tempo para gerar o par de chaves varia muito conforme a capacidade do dispositivo, bem como do tamanho de chave selecionado por ele.

3.3.1 Geração de Chaves

Para gerar o par de chaves, é preciso obter um um gerador de números aleatórios. Usei a função *FortunaRandom()*, disponibilizada pela biblioteca Pointy Castle, para obtê-lo. Em seguida, deve-se instanciar um objeto do tipo *RSAKeyGenerator* e inicializá-lo. Enfim, podemos invocar o método *generateKeyPair*.

3.4 Tela 4: *Sign List*

Nesta página, o usuário pode assinar a lista de dispositivos BLE detectados usando o par de chaves gerado anteriormente.

3.4.1 Assinatura

A fim de assinar a lista, o programa começa extraindo os IDs, objetos do tipo *DeviceIdentifier*, dos dispositivos e transformando-os em *Strings*, formando uma segunda lista apenas com os IDs. Em seguida, esta é convertida do tipo *List* para *String* através de um método *json.encode()*, da biblioteca convert.
Exemplo de funcionamento do método *json.encode()*:

```
1 var encoded = json.encode([1, 2, { "a": null }]);
2 var decoded = json.decode('["foo",␣{␣"bar":␣499␣}]');
```

Com essa conversão, é possível tratar a nossa lista de dispositivos como uma *String*. O próximo passo é computar o hash dos dados. Isso é feito convertendo nossos dados com o método *sha256.convert()*. Em seguida, é preciso inicializar o *RSASigner*, indicando a chave privada e o algoritmo de *Digest* que será usado. Por fim, o método *generateSignature*, da biblioteca Pointy Castle, é invocado no objeto *RSASigner*.
O usuário pode visualizar uma representação na base 64 da assinatura pressionando o botão ”See Signature” da página.

3.5 Tela 5: *Verify Signature*

Na última tela, o usuário poderá verificar se a assinatura é, ou não, válida. Para fazer o teste de verificação, a aplicação usa a última lista de dispositivos escaneados, o último par de chaves gerados e a última assinatura feita.

3.5.1 Verificação

Os passos para verificação são bem parecidos com os feitos para a assinatura. É preciso converter os dados da mesma forma que antes, porém, ao invés de inicializarmos o *RSASigner* com a chave privada, inicializamos com a chave pública. No fim, chama-se o método *verifySignature*, no lugar do *GenerateSignature*. Esse método retornará um *booleano*: se a assinatura é válida, retorna *true*, caso contrário, *false*.

3.6 Temas

Terminadas as funções principais, decidi tentar adicionar uma função de temas, escuro e claro, no tempo restante. Pude implementá-lo utilizando o pacote Provider para carregar as informações do tema com o *ChangeNotifier*.

4 Como Testar o Aplicativo

1. Baixe o código fonte .zip disponibilizado
2. Extraia o conteúdo do arquivo em um diretório vazio
3. Abra o terminal no diretório 'myapp'
4. Execute o comando `flutter run` no terminal