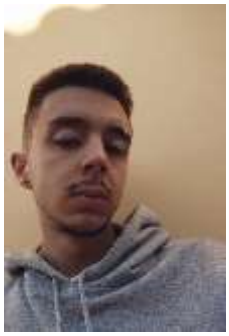
Universidade do Minho

MEI - Mestrado em Engenharia Informática

Engenharia de Segurança

1º ano / 2º semestre

Grupo 11 – Projeto de desenvolvimento 1

 pg47848 - Hugo Marques	 Pg46538 - José Florindo
 pg44420 - Nuno Mata	 a84288 - José Santos

02 de maio de 2022

Conteúdo

1 - Introdução

O presente relatório tem o objetivo descrever o desenvolvimento e decisões tomadas ao longo da elaboração do Projeto de Desenvolvimento 1 (PD1), no âmbito da unidade curricular Engenharia de Segurança, integrada no perfil de Criptografia e Segurança da Informação. O PD1 resume-se ao desenvolvimento de um cofre digital que permita depositar e fornecer qualquer tipo de documentos de forma segura, recorrendo a técnicas de *hashing* e a chaves públicas e privadas que vão entrar no processo de cifra e decifra do documento para que este seja apenas fornecido a utilizadores permitidos.

As restrições para o desenvolvimento do cofre digital impostas ao nosso grupo (Grupo 11), foram o desenvolvimento em *Java* e a utilização do Provider *BouncyCastle* (uma API de criptografia para Java).

2 - Contextualização

O problema exposto no enunciado é o desenvolvimento de um cofre digital e de forma a podermos entender melhor o funcionamento de um cofre digital vamos expor algumas ideias e conceitos que definem um cofre digital forte (ou seguro).

No cofre deve poder ser guardado qualquer tipo de documento, sendo que para qualquer documento guardado são gerados 2 artefactos: **(1)** o hash do documento; **(2)** a chave de decifra do documento cifrada. Estes artefactos são então fornecidos ao depositante, e já que foi usada a sua chave pública para cifrar a chave de decifra do documento, apenas alguém com informação da chave privada do depositante consegue ter acesso a estes artefactos, pois esta é necessária para decifrar o artefacto (2).

Adicionalmente, o depositante pode ter a opção de apenas permitir que o documento seja acedido apenas quando são mais do que um utilizador em simultâneo, de um grupo de utilizadores com permissão, está a tentar aceder ao documento. Para isso utiliza-se o esquema de *Shamir* que divide a chave de decifra em múltiplas partes, onde cada uma das partes divididas é entregue a um membro do grupo e cifrada com a sua chave pública para que depois as partes em conjunto possam reconstruir a chave original.

Por fim deve poder ser realizada a operação de fornecer o documento. Para isso é necessário que seja devolvida a chave de decifra do documento devidamente decifrada, recorrendo à chave privada do depositante. No caso da chave de decifra ser dividida por um grupo através do esquema *Shamir*, o processo de fornecimento do documento deve ser realizado sem que a parte da chave de cada participante seja exposta.

3 – Bibliotecas e Tecnologias

Com base na análise feita optamos por uma arquitetura **cliente-servidor**. O cliente tem a opção de cifrar, decifrar, juntar-se ao segredo de alguém (para cifrar ou decifrar) e cifrar de forma a partilhar o segredo com outros clientes. Como pedido, e por sermos o grupo 11, usamos o Provider da BouncyCastle e também uma biblioteca de *secret share*. Recorremos ao *maven* para instalar as mesmas e garantir uma simples instalação do projeto.

4 – Implementação

De forma que o cliente possa fazer o pedido, este envia uma mensagem, como é possível verificar no ficheiro DataClassMessage, com os parâmetros necessários para o servidor saber o pedido do cliente. Neste é enviada a chave pública do cliente, se o segredo vai ser partido, se é para decifrar, cifrar e no caso de cifrar o tamanho do ficheiro.

No caso de cifrar, o servidor recebe o pedido e prepara-se para receber o ficheiro de seguida. Gera uma chave aleatória e um **IV** e cifra o ficheiro usando ficheiros temporários para o efeito (de forma a permitir ficheiros de tamanho superior ao buffer). O ficheiro é então cifrado com *AES* através do Provider BouncyCastle.

De seguida envia um DataReportMessage com o *status*, o *hash* e a *key*. Esta key é cifrada com a chave pública do cliente. O cliente recebe o *report* e decifra a key com a sua chave privada.

Para decifrar o cliente envia o hash e a key decifrada e recebe o ficheiro decifrado. Para o servidor saber de qual ficheiro se trata através do *hash*, tem uma estrutura com o hash como key e um DataClass como value que retêm a informação necessária para o servidor.

Para a partilha de segredos, o cliente faz um pedido e insere o **n** (número de partes criadas) e o **k** (número de partes necessárias). O servidor recebe e divide o segredo, adicionado os segredos disponíveis a uma *class* SecretManager. Esta tem toda a informação necessária para trabalhar com os segredos. Implementa uma *queue* com as chaves disponíveis para os clientes. Estes depois fazem um pedido, enviando apenas o hash do ficheiro, o servidor envia a chave, sendo esta encriptada também com a sua chave pública. Para decifrar, o primeiro cliente a pedir para decifrar um ficheiro que tenha sido cifrado com a partilha de segredos fica à espera de que os outros clientes insiram a sua chave. Estes enviam um pedido com a sua chave e a *class* SecretManager adiciona a um mapa, as chaves inseridas pelos clientes. Quando as chaves necessárias forem inseridas nesse mapa, o servidor agrupa

as chaves e obtém a chave usada para cifrar o ficheiro. Este decifra o ficheiro e envia-o ao cliente que fez o primeiro pedido para receber o ficheiro decifrado.

Ao agrupar as chaves é preciso um **par (x, y)**. Este par é obtido através da geração de um *hash* de y, que permite ao servidor guardar o x tendo em conta o y , sem saber o y.

3 – Manual de utilização

De forma a simplificar a instalação da aplicação, é necessário recorrer ao *maven* para instalar as dependências necessárias. De seguida, compilando o projeto, basta correr uma instância do servidor e as instâncias que quisermos dos clientes. O cliente tem um menu bastante intuitivo e de fácil utilização. Como já referido, temos opções para cifrar, decifrar, cifrar com partilha de segredo, fazer um pedido para ser um participante na partilha de segredo e fazer um pedido para enviar a chave de um segredo de forma a alguém decifrar o ficheiro. Para decifrar um ficheiro com partilha de segredo basta usar a **opção 2** e enviar o hash e a chave.

4 – Melhorias futuras

De futuro poderíamos implementar algumas funcionalidades interessantes como por exemplo permitir que a chave pública e privada venha de um certificado (uma assinatura digital ou assim).

5 - Referências

1. https://github.com/codahale/shamir - Biblioteca Shamir para Java