



UNIVERSIDADE
TÉCNICA DO
ATLÂNTICO



CAMPUS
DO MAR

Trabalho Prático Criptografia Avançada

Licenciatura em Engenharia Informática e Telecomunicações

Docente:

Estanislau Lima

Discentes:

Fredy Brito

Gabriel da Luz

Introdução

Este trabalho prático consiste em desenvolver um sistema seguro de comunicação ponto-a-ponto, utilizando técnicas modernas de criptografia para garantir confidencialidade, integridade e autenticidade das mensagens trocadas entre dois utilizadores. Ela conta com a combinação de técnicas avançadas de criptografia, incluindo criptografia simétrica (AES), assinaturas digitais (RSA), troca segura de chaves (Diffie Hellman) e mecanismos de integridade (HMAC), além de armazenamento seguro de credenciais com algoritmos modernos de hash.

Todos os experimentos realizados neste projeto foram desenvolvidos no editor de texto *Visual Studio Code (VSCode)* e executados localmente. A linguagem de programação utilizada foi Java, escolhida pela sua robustez e pela ampla disponibilidade de bibliotecas nativas, como o pacote *java.security*, que facilita a implementação de funcionalidades criptográficas, evitando a necessidade de desenvolvê-las do zero.

Objetivos

- Aplicar criptografia simétrica (AES) com modos seguros de operação.
- Integrar HMAC para garantir integridade e autenticidade.
- Utilizar Diffie-Hellman para troca segura de chaves.
- Implementar criptografia assimétrica (RSA) e assinatura digital.
- Armazenar senhas com algoritmos seguros de hash (bcrypt ou Argon2).
- Avaliar segurança de PRNGs utilizados na geração de chaves e IVs.

Desenvolvimento

Começamos por preparar o ambiente para realizar as experiências, nomeadamente configuração do **VSCode**, e algumas bibliotecas que teriam de ser usadas, a principal usada foi o pacote **java.security**, é uma parte central da arquitetura de segurança, com ele é possível implementar e controlar questões de segurança, incluindo criptografia, autenticação, comunicação segura, e controlo de acesso e tudo isso na forma de funções pré definidas. Outras ferramentas usadas neste projeto, foram o **Typescript**, usada para não apenas apresentar os componentes da interface gráfica, mas não para logo implementar algumas funções importantes para que o ecossistema de criptografia funcione. Figura 1 abaixo mostra como ficou a interface gráfica desenhada e a *Figura 1* mostra a mesma interface porém supostamente de outra pessoa que iniciou uma comunicação.

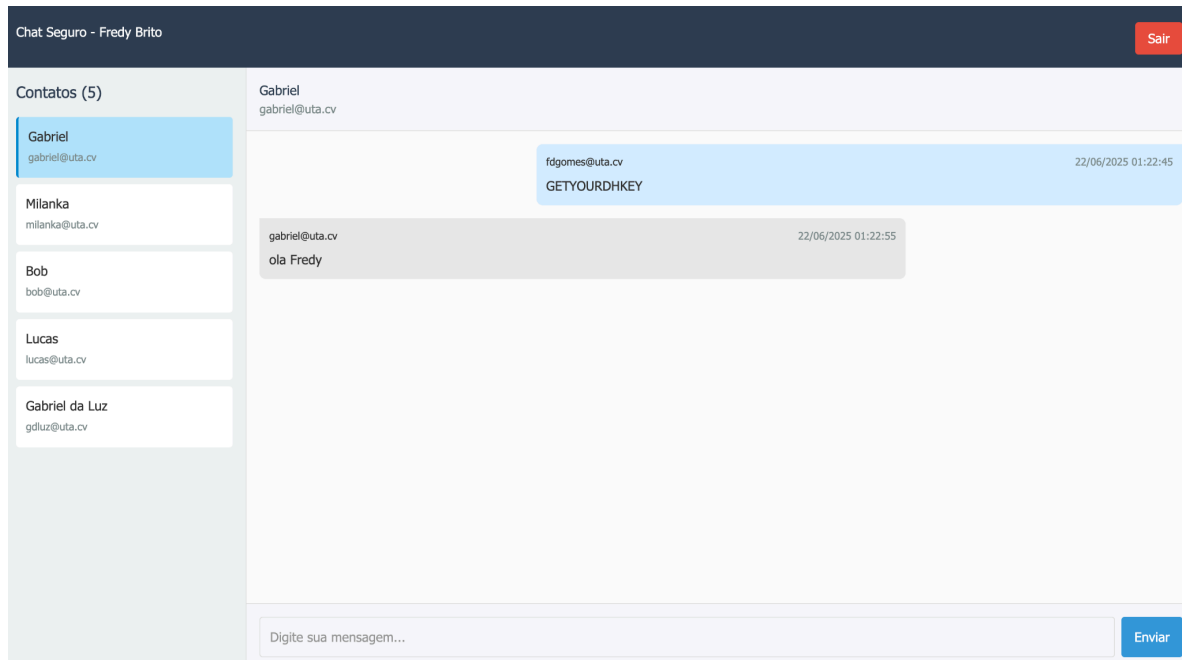


Fig. 1

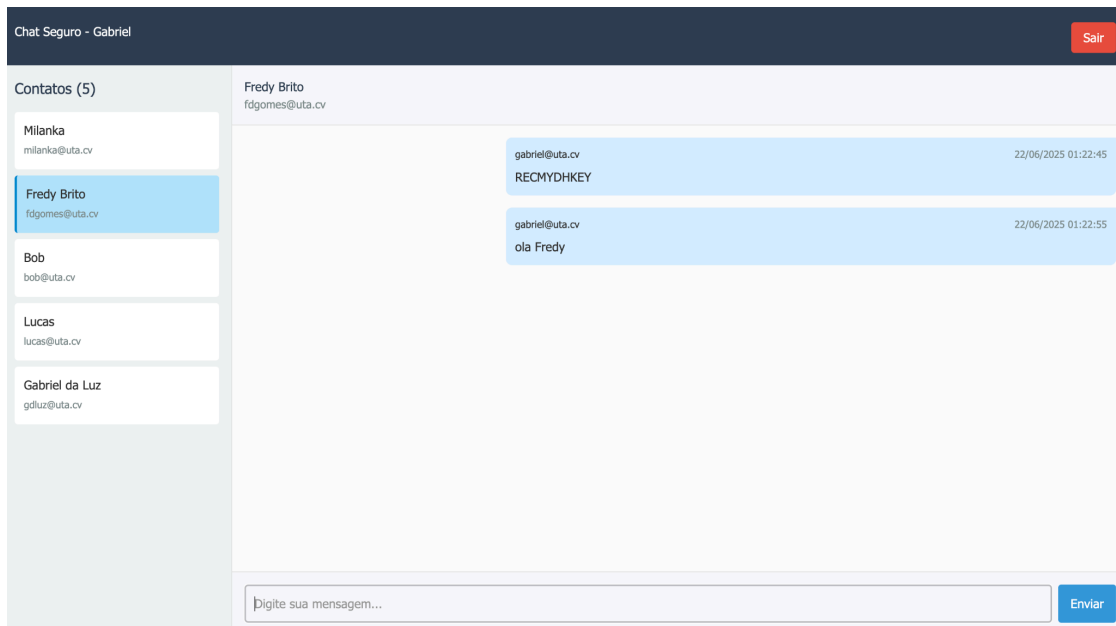


Fig. 2

Nele encontra-se a função apelidado de `cryptoService`, em que se pode encontrar os métodos *encrypt* e *decrypt* em que os nomes já falam por si, métodos para que o cliente consiga encriptar desencriptar a mensagem, elas provêm de uma biblioteca do **Javascript**, e *Figura 3* abaixo mostra um trecho do código que implementa essas funções.

```
export const encryptRSA = (message: string): string | false => {
  const encryptor = new JSEncrypt();
  encryptor.setPublicKey(publicKey);
  return encryptor.encrypt(message);
};

export const decryptRSA = (encryptedMessage: string): string | false => {
  const decryptor = new JSEncrypt();
  decryptor.setPrivateKey(privateKey);
  return decryptor.decrypt(encryptedMessage);
};
```

Fig. 3

A primeira escolha a ser feita foi qual algoritmo usar, sendo que poderia se escolher ou a criptografia simétrica AES (Advanced Encryption Standard), **algoritmo de criptografia simétrica** ou RSA **algoritmo de criptografia assimétrica** também usada para garantir segurança na comunicação digital, em que a sua ideologia é usar um par de chaves (uma pública, e uma privada). Cada um deles poderia permitir que as mensagens fossem enviadas com segurança, ou seja usados mesmo em canais inseguros, o que nos levou a escolher portanto o RSA, que garante a autenticidade e integridade das mensagens enviadas, através da assinatura digital ou mesmo pelo uso de não, mas duas chave. Usamos também a linguagem **Java**, ferramenta que permitiu criar o canal de comunicação entre estes dois ou mais clientes. Coordenar estas duas ferramentas é um desafio, porque não se consegue conectar o frontend (feito em typescript) com o backend (feito em java) apenas executando cada um separando, o que se quer dizer é que, a conexão entre estes dois não começa a partir do momento que o próprio sistema verificar que consegue enviar as chaves, para que esteja disposto a acontecer as trocas de chaves. Falando em troca de chaves, a criptografia não se resume apenas a chaves, porém a outras ferramentas que ajudam. Uma delas é a **SHA256** função hash da criptografia, que ao receber um input e irá produzir um valor com valor fixo de *256 bits (32 byte)*, que no nosso caso foi usada na segurança de hashing de password. Combinando com o SHA256, também aproveitamos em usar o **PKCS#7** (Public Key Cryptography Standards #7), que é combinado com o algoritmo de *SHA256*, opera exatamente sobre blocos de tamanho fixo, RSA neste caso mesmo não tendo um tamanho fixo porém como depende do tamanho da chave então consegue-se implementar SHA256. PKCS#7 atua quando a mensagem não tem o tamanho exato múltiplo do bloco, é necessário preencher o final da mensagem com bytes extras.

A comunicação só pode ser feita, porque é criado um WebSocket (outra opção, criar um socket com TCP/IP puro), que ao ser criado têm métodos, que justamente por estar a usar a linguagem Java, permitiu manipulá-los e por conseguinte permitiu injectar o **DH** (Diffie Helman), método da criptografia que permite duas partes estabeleça uma chave secreta compartilhada permitindo que haja uma conversação num canal de comunicação inseguro. *Figura 4* ilustra como é o funcionamento prático do DH.

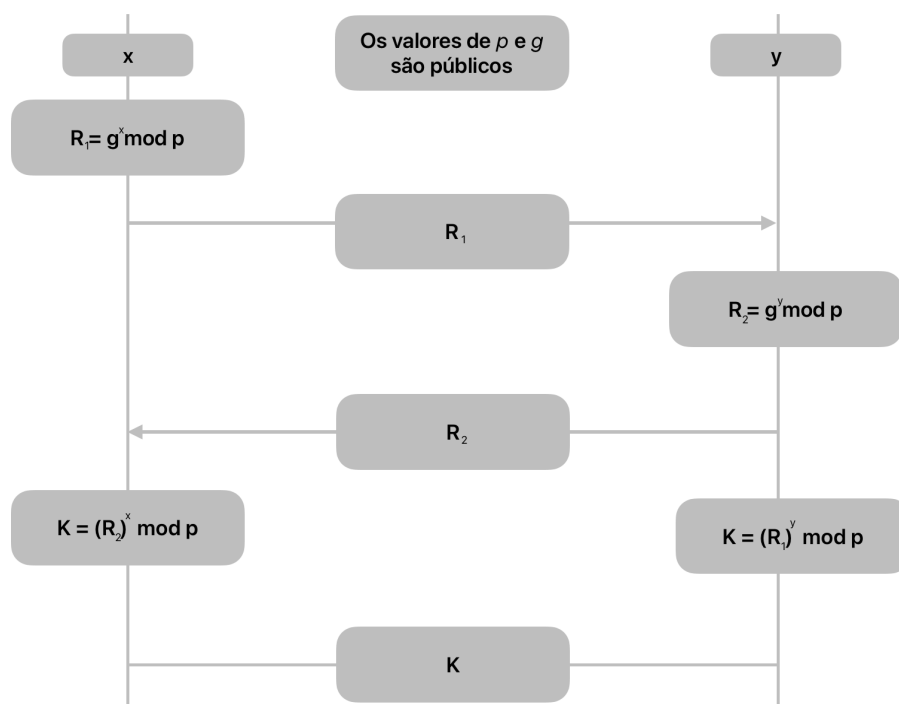


Fig. 4

DH normalmente gera as chaves para cada lado, porém para testes o nosso servidor WebSocket tem um chave fixa, e para a parte do cliente, é gerado uma chave privada aleatória segura entre 2 e 7917. Chaves públicas e privadas, têm de passar por um processo de encriptação excelente, portanto o nosso sistema teve que ser o mais seguro possível. A maneira prática de explicar o nosso sistema é explicando, se alguém deseja enviar uma informação confidencial, encripta o conteúdo com a chave pública do destinatário. Apenas o destinatário, com sua chave privada, poderá descriptar a mensagem. E além disso, RSA também pode ser usado para criar **assinaturas digitais**, o que no nosso sistema também prevalece, e portanto as chaves públicas se em qualquer caso não forem encriptadas corretamente, mas também são assinadas digitalmente, assegurando que não foram alteradas durante a transmissão. Explicando melhor o remetente assina a mensagem com sua chave privada correspondente, garantindo a autenticidade da mensagem. *Figura 5* mostra justamente

como é dado o início do processo de todo o sistema, pois o utilizador só terá a criação da sua chave ao registar ou fazer o seu login, e aproveitando para consolidar logo tudo, o botão entrar que vai excitar todo o sistema, para que ele começa a criar as chaves públicas e privadas.

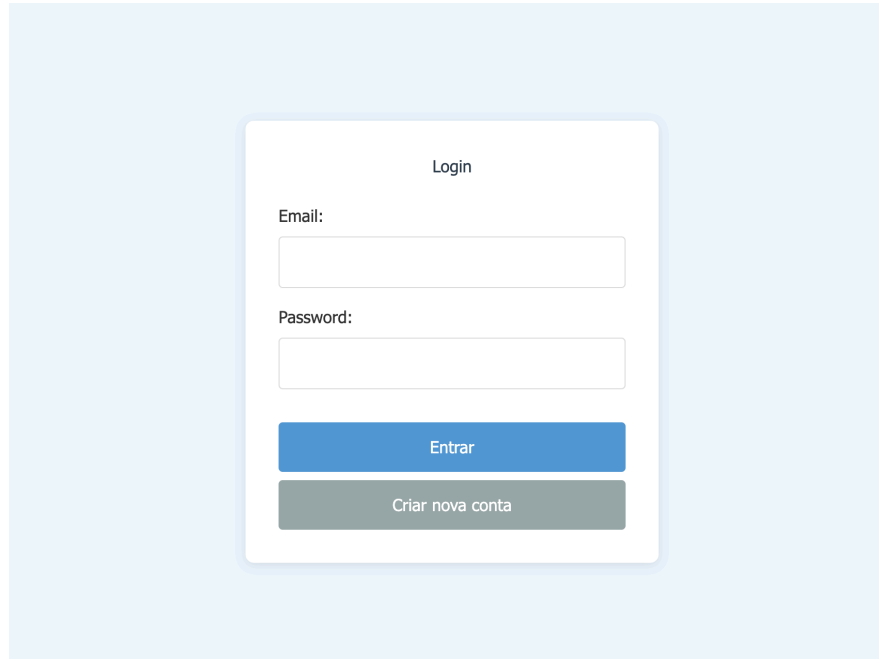
A login form titled "Login" is centered on a light blue background. The form is white with rounded corners and contains two input fields: "Email:" and "Password:". Below the "Password:" field are two buttons: a blue "Entrar" button and a grey "Criar nova conta" button.

Fig. 5

Para cada usuário, uma forma de verificarmos se cada chave estava a ser criada e troca de forma correta, foi adotado uma medida de que se cada lado clicar correntemente no chat da pessoa que quer conversar, é apresentado justamente um alert, indicando qual a chave sendo usada para esta troca de mensagens, de modo que conseguissemos garantir seguramente a funcionalidade do nosso sistema (Figura 6).

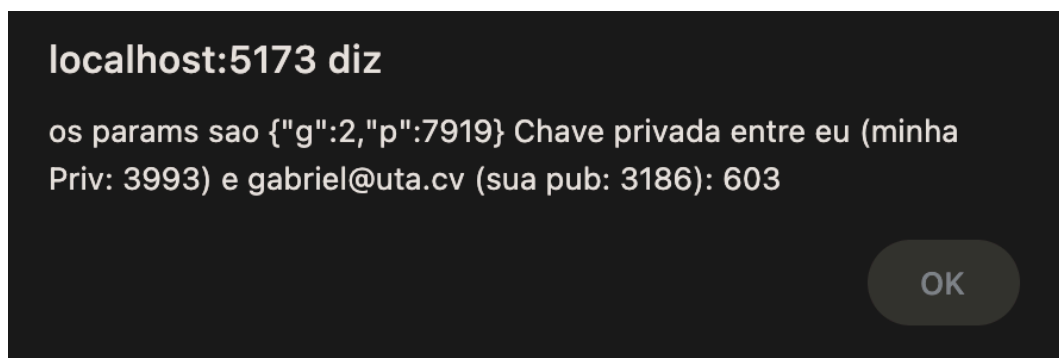


Fig. 6

Conclusão

Desenvolvendo este sistema de comunicação segura permitiu que colocássemos um pouco a prova os conceitos abordados na cadeira de Gestão e Segurança de Redes. As diferentes técnicas usadas seja para a verificação de integridade, seja para a troca de chaves ou mesmo para garantir a confidencialidade, autenticidade e integridade dos dados proporcionou com que criássemos uma arquitetura robusta.

Também usar ferramentas conhecidas como Java e Typescript para a integração do backend e frontend, também exigiu uma abordagem cuidadosa, sobretudo na interoperabilidade entre diferentes bibliotecas criptograficas e na definição de formatos de dados compatíveis. O uso de WebSockets foi essencial para garantir comunicação em tempo real, enquanto a aplicação de padding com PKCS#7 e a validação da segurança dos PRNGs contribuíram para aumentar a resiliência do sistema contra ataques.

Este projeto permitiu compreender os desafios reais envolvidos na construção de sistemas seguros. A prática mostrou que segurança não se resume à aplicação de algoritmos, mas sim ao correto encadeamento de técnicas e à atenção aos detalhes de implementação.