DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Segurança e Privacidade de Dados

Trabalho 2

Mestrado em Engenharia Informática – Engenharia de Dados

2023 / 2024

1190384 – André Teixeira

1190818 – Luís Pinto

1231925 – João Henriques

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Resumo

O relatório aborda a implementação de uma Infraestrutura de Chave Pública (PKI) para garantir a segurança das comunicações e dos dados em repouso de uma empresa virtual, seguindo as melhores práticas de segurança. O problema central é a complexidade e a gestão desafiadora da criptografia em ambientes empresariais, o que pode levar à diminuição dos controles de segurança e ao aumento das ameaças aos sistemas de informação. A metodologia inclui a criação de um Certificado de Autoridade (CA) interno e externo, a definição de políticas de segurança, a implementação de autenticação por chave em uma aplicação protegida por um proxy reverso e o desenvolvimento de procedimentos para *onboarding*, revogação e recuperação de identidades.

Palavras-chave:

PKI, segurança, criptografia, certificados, *CA*, *TLS*, políticas de segurança, autenticação, revogação de identidade, recuperação de identidade, gestão de certificados, infraestrutura de chave pública, aplicação de autenticação.

ÍNDICE

ÍNDICE DE FIGURAS

1. Introdução

Este relatório foi realizado no âmbito da unidade curricular de Segurança e Privacidade de Dados, incluída no primeiro ano do mestrado em Engenharia Informática, ramo de Engenharia de Dados, do Instituto Superior de Engenharia do Porto.

O nosso objetivo principal foi implementar uma Infraestrutura de Chave Pública (*PKI*) seguindo as melhores práticas de segurança, garantindo a segurança das comunicações e dados em repouso. Isso inclui a criação de um Certificado de Autoridade (*CA*) interno e externo, emissão e gestão de certificados para utilizadores e serviços, implementação de políticas de segurança, autenticação por chave em vez de senha numa aplicação protegida por um *proxy* reverso e recuperação de identidades.

2. Implementação

Este comando gera uma chave privada para a Autoridade de Certificação com o nome do arquivo *rootCA.key* e um comprimento de chave de 4096 bits.

```
C:\Users\joao_\Desktop\SEGPRD>openssl genrsa -aes256 -out rootCA.key 4096
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

C:\Users\joao_\Desktop\SEGPRD>
```

Figura 1 - rootCA

```
C:\Users\joao_\Desktop\SEGPRD>openssl req -new -key rootCA.key -out rootCA.csr
Enter pass phrase for rootCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PT
State or Province Name (full name) [Some-State]:Porto
Locality Name (eg, city) []:Porto
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ISEP
Organizational Unit Name (eg, section) []:Mestrado Engenharia de Dados
Common Name (e.g. server FQDN or YOUR name) []:JH, LP & AT
Email Address []:1231925@isep.ipp.pt

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:tp2
String too short, must be at least 4 bytes long
A challenge password []:tpratico2
An optional company name []:ISEP MEI
```

Passe: segprd

```
C:\Users\joao_\Desktop\SEGPRD>openssl x509 -req -days 3650 -in rootCA.csr -signkey rootCA.key -out rootCA.crt
Enter pass phrase for rootCA.key:
Certificate request self-signature ok
subject=C=PT, ST=Porto, L=Porto, O=ISEP, OU=Mestrado Engenharia de Dados, CN=JH, LP & AT, emailAddress=1231925@isep.ipp.pt
```

Irá ser explicado o processo de implementação através da explicação de *snipets* do código criado para o problema exposto.

Começamos por fazer os *imports* das bibliotecas *python* utilizadas para o projeto.

```
import sys
import json
from PyQt5.QtWidgets import QApplication, QWidget, QLabel, QLineEdit, QPushButton, QMessageBox
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes
```

Figura 2 - Imports do projeto

Este código define uma classe "CertificateAuthority" que representa uma Autoridade Certificadora (*CA*). Uma *CA* é uma entidade confiável que emite certificados digitais para autenticar a identidade de entidades em um sistema de segurança, como em comunicações seguras pela internet.

- A classe "CertificateAuthority" possui um método "__init__ " que inicializa um dicionário(estrutura de dados que mapeia chaves para valores) vazio chamado *certificates*. Este dicionário será usado para armazenar os certificados emitidos pela autoridade certificadora.
- O método "issue_certificate" é responsável por emitir um certificado para um determinado sujeito (identidade). Ele aceita dois argumentos: subject_name (o nome do sujeito para quem o certificado está sendo emitido) e validity_period (o período de validade do certificado).

  No interior deste método, é chamada uma função generate_key_pair() para gerar um par de chaves público-privadas para o utilizador.

  Depois de gerar o par de chaves, o método cria um certificado contendo informações sobre o *user*, a chave pública e o período de validade.

  A chave pública é convertida num formato PEM (*Privacy-Enhanced Mail*) usando a biblioteca *serialization* do módulo *cryptography*.

  O certificado é então armazenado no dicionário *certificates*, utilizando o nome do sujeito como chave.

  Finalmente, o método retorna o certificado emitido.

```python
class CertificateAuthority:
    def __init__(self):
        self.certificates = {}

    def issue_certificate(self, subject_name, validity_period):
        # Emite um certificado para o sujeito com o periodo de validade especificado
        private_key, public_key = generate_key_pair()
        certificate = {
            'subject': subject_name,
            'public_key': public_key.public_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PublicFormat.SubjectPublicKeyInfo
            ).decode('utf-8'),
            'validity_period': validity_period
        }
        self.certificates[subject_name] = certificate
        return certificate
```

Figura 3 - issue_certificate

Este método revoga o certificado associado ao nome do sujeito especificado. Ele verifica se o nome do sujeito está presente no dicionário de certificados da *CA*. Se estiver presente, o certificado é removido usando *del*

```python
def revoke_certificate(self, subject_name):
    # Revoga o certificado para o sujeito especificado
    if subject_name in self.certificates:
        del self.certificates[subject_name]
        QMessageBox.information(None, "Success", "Certificate for {} revoked.".format(subject_name))
    else:
        QMessageBox.critical(None, "Error", "Certificate not found for {}.".format(subject_name))
```

Figura 4 - revoke_certificate

Este método simplesmente retorna todos os certificados emitidos pela CA, que são armazenados no dicionário “self.certificates”. Retorna o dicionário completo contendo todos os certificados.

```
def list_certificates(self):
    # Lista todos os certificados emitidos pela CA
    return self.certificates
```

Figura 5 - list_certificates

O método "__init__" é o construtor da classe. Inicializa um dicionário vazio chamado *user_identities*, que será usado para armazenar as identidades dos utilizadores, juntamente com os seus pares de chaves público-privadas.

- Já o método *generate_user_certificate* é usado para gerar um certificado para um utilizador específico. Ele aceita o nome de utilizador como entrada.

- Dentro deste método, um par de chaves RSA é gerado usando uma função chamada "generate_key_pair()". Depois de gerar o par de chaves, um certificado é criado contendo o nome de utilizador e a chave pública correspondente.
  A chave pública é convertida para o formato *PEM* usando a biblioteca *serialization* do módulo *cryptography*.
  O par de chaves (chave privada e pública) é armazenado no dicionário "user_identities", onde a chave é o nome de utilizador e o valor é uma tupla(semelhante a uma lista, mas com a distinção principal de que é imutável)contendo a chave privada e a chave pública.
  Finalmente, o método retorna o certificado gerado para o utilizador.

```python
class IdentityProvider:
    def __init__(self):
        self.user_identities = {}

    def generate_user_certificate(self, username):
        # Gera um par de chaves RSA para o utilizador
        private_key, public_key = generate_key_pair()

        # Emite um certificado para o utilizador
        certificate = {
            'username': username,
            'public_key': public_key.public_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PublicFormat.SubjectPublicKeyInfo
            ).decode('utf-8')
        }
        self.user_identities[username] = (private_key, public_key)
        return certificate
```

Figura 6 - generate_user_certificate

Esta função faz parte da classe “IdentityProvider” e é responsável por revogar o certificado de um utilizador específico, identificado pelo nome de utilizador fornecido

```python
def revoke_user_certificate(self, username):
    # Revoga o certificado do utilizador
    if username in self.user_identities:
        del self.user_identities[username]
        QMessageBox.information(None, "Success", "Certificate for {} revoked.".format(username))
    else:
        QMessageBox.critical(None, "Error", "Certificate not found for {}.".format(username))
```

Figura 7 - revoke_user_certificate

A função “recover_user_identity “ também faz parte da classe “IdentityProvider” e é usada para recuperar a identidade de um utilizador específico, identificado pelo nome de utilizador fornecido como argumento. É útil em casos de perda ou comprometimento da identidade do *user*.

```python
def recover_user_identity(self, username):
    # Recupera a identidade do utilizador em caso de perda ou comprometimento
    if username in self.user_identities:
        return self.user_identities[username]
    else:
        return None
```

Figura 8 - recover_user_identity

Este método é usado para criptografar uma mensagem usando a chave pública do destinatário. Ele aceita dois argumentos: a mensagem a ser criptografada e a chave pública do destinatário.

```python
def encrypt_message(self, message, public_key):
    # Criptografa uma mensagem usando a chave pública do destinatário
    ciphertext = public_key.encrypt(
        message.encode(),
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return ciphertext
```

Figura 9 - encrypt_message

Já por outro lado, este método é usado para descriptografar uma mensagem usando a chave privada do destinatário. Ele aceita dois argumentos: o texto cifrado e a chave privada do destinatário

```python
def decrypt_message(self, ciphertext, private_key):
    # Descriptografa uma mensagem usando a chave privada do destinatário
    plaintext = private_key.decrypt(
        ciphertext,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    return plaintext.decode()
```

Figura 10 - decrypt_message

Esta função gera um par de chaves *RSA*, composto por uma chave privada e uma chave pública. Usa a biblioteca *cryptography* para gerar as chaves com um tamanho de chave de 2048 bits. A chave privada é retornada como “private_key” e a chave pública é derivada da chave privada e retornada como “public_key”.

```python
def generate_key_pair():
    # Gera um par de chaves RSA
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=2048,
        backend=default_backend()
    )
    public_key = private_key.public_key()
    return private_key, public_key
```

Figura 11 - generate_key_pair

Esta função guarda os certificados emitidos pela *CA* num arquivo *JSON* chamado “certificates.json”

```python
def save_certificates(ca):
    # Salva os certificados emitidos pela CA num arquivo JSON
    with open('certificates.json', 'w') as f:
        json.dump(ca.list_certificates(), f, indent=4)
```

Figura 12 - save_certificates

Esta função carrega os certificados emitidos pela CA a partir do arquivo JSON “certificates.json”

```python
def load_certificates():
    # Carrega os certificados emitidos pela CA a partir de um arquivo JSON
    try:
        with open('certificates.json', 'r') as f:
            certificates = json.load(f)
        return certificates
    except FileNotFoundError:
        return {}
```

Figura 13 - load_certificates

Este método é chamado quando o utilizador deseja emitir um certificado para a própria Autoridade Certificadora. Ele obtém o nome do sujeito (no caso, a Autoridade Certificadora) e o período de validade a partir da interface do *user*. Em seguida, chama o método "issue_certificate" da instância ca (que é uma instância da classe *CertificateAuthority*) para emitir o certificado para a *CA*. Depois de emitir o certificado, chama a função "save_certificates" para salvar os certificados atualizados (incluindo o novo certificado emitido para a *CA*) num arquivo *JSON*. Por fim, exibe uma mensagem informativa para o utilizador indicando que o certificado da *CA* foi emitido com sucesso.

```
def issue_ca_certificate(self):
    # Emite um certificado para a Autoridade de Certificação (CA)
    subject_name = self.subject_entry.text()
    validity_period = self.validity_entry.text()
    certificate = self.ca.issue_certificate(subject_name, validity_period)
    save_certificates(self.ca)
    QMessageBox.information(self, "Success", "CA Certificate issued for {}.".format(subject_name))
```

Figura 14 - issue_ca_certificate

Este método é chamado quando o utilizador deseja revogar o certificado da Autoridade Certificadora. Ele obtém o nome do sujeito (novamente, a *CA*) da interface do *user* e, em seguida, chama o método "revoke_certificate" da instância ca para revogar o certificado da *CA*. Depois de revogar o certificado, chama a função save_certificates para salvar os certificados atualizados (com o certificado da *CA* revogado) num arquivo *JSON*.

```
def revoke_ca_certificate(self):
    # Revoga o certificado da Autoridade de Certificação (CA)
    subject_name = self.subject_entry.text()
    self.ca.revoke_certificate(subject_name)
    save_certificates(self.ca)
```

Figura 15 - revoke_ca_certificate

Este método é chamado quando o utilizador deseja listar os certificados emitidos pela Autoridade Certificadora (*CA*). Ele obtém os certificados da *CA* chamando o método "list_certificates" da instância *CA* (que é uma instância da classe CertificateAuthority).

```
def list_ca_certificates(self):
    # Lista os certificados emitidos pela Autoridade de Certificação (CA)
    certificates = self.ca.list_certificates()
    QMessageBox.information(self, "CA Certificates", json.dumps(certificates, indent=4))
```

Figura 16 - list_ca_certificates

Este método é chamado quando o utilizador deseja emitir um certificado para um *user*. Ele obtém o nome do utilizador da interface do *user* e, em seguida, chama o método "generate_user_certificate" da instância *identity_provider* (que é uma instância da classe *IdentityProvider*) para gerar o certificado para o utilizador.

```
def issue_user_certificate(self):
    # Emite um certificado para o utilizador
    username = self.subject_entry.text()
    certificate = self.identity_provider.generate_user_certificate(username)
    QMessageBox.information(self, "Success", "User Certificate issued for {}.".format(username))
```

Figura 17 - issue_user_certificate

Este método é chamado quando o *user* deseja revogar o certificado de um utilizador.

```
def revoke_user_certificate(self):
    # Revoga o certificado do utilizador
    username = self.subject_entry.text()
    self.identity_provider.revoke_user_certificate(username)
```

Figura 18 - revoke_user_certificate

Este método é chamado quando o utilizador deseja listar os certificados emitidos para os utilizadores. Ele itera sobre o dicionário "user_identities" da instância identity_provider para obter todos os certificados dos usuários. Para cada utilizador, obtém a chave pública associada e a converte em uma *string PEM*

```
def list_user_certificates(self):
    # Lista os certificados emitidos para utilizadores
    user_certificates = {}
    for username, (private_key, public_key) in self.identity_provider.user_identities.items():
        user_certificates[username] = {
            'public_key': public_key.public_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PublicFormat.SubjectPublicKeyInfo
            ).decode('utf-8')
        }
    QMessageBox.information(self, "User Certificates", json.dumps(user_certificates, indent=4))
```

Figura 19 - list_user_certificates

Este método criptografa uma mensagem usando a chave pública do destinatário. Ele obtém o nome do destinatário e a mensagem da interface do utilizador. Em seguida, verifica se o destinatário está presente no dicionário "user_identities" da instância *identity_provider*, que contém as identidades dos utilizadores. Se o destinatário estiver presente, recupera sua chave pública e usa a instância da classe "EndToEndEncryption" para criptografar a mensagem usando a chave pública do destinatário.

```
def encrypt_message(self):
    # Criptografa uma mensagem usando a chave pública do destinatário
    recipient = self.subject_entry.text()
    message = self.validity_entry.text()
    if recipient in self.identity_provider.user_identities:
        recipient_public_key = self.identity_provider.user_identities[recipient][1]
        ciphertext = self.end_to_end_encryption.encrypt_message(message, recipient_public_key)
        QMessageBox.information(self, "Success", "Message encrypted successfully.")
    else:
        QMessageBox.critical(self, "Error", "Recipient not found.")
```

Figura 20 - encrypt_message

Por outro lado, este método decriptografa uma mensagem usando a chave privada do destinatário. Ele também obtém o nome do destinatário e o texto cifrado da interface do utilizador. Da mesma forma que no método “encrypt_message”, verifica se o destinatário está presente no dicionário “user_identities”. Se o destinatário estiver presente, recupera sua chave privada e usa a instância da classe “EndToEndEncryption” para descriptografar o texto cifrado usando a chave privada do destinatário

```
def decrypt_message(self):
    # Descriptografa uma mensagem usando a chave privada do destinatário
    recipient = self.subject_entry.text()
    ciphertext = self.validity_entry.text()
    if recipient in self.identity_provider.user_identities:
        recipient_private_key = self.identity_provider.user_identities[recipient][0]
        plaintext = self.end_to_end_encryption.decrypt_message(ciphertext, recipient_private_key)
        QMessageBox.information(self, "Success", "Decrypted Message: {}".format(plaintext))
    else:
        QMessageBox.critical(self, "Error", "Recipient not found.")
```

Figura 21 - decrypt_message

Já na classe *User*, também se começa por uma função “__init__” responsável por criar o objeto do user com esse nome. Bem como a função “generate_key_par” que cria um par de chaves *RSA* pública e privada para o utilizador.

```
class User:
    def __init__(self, username):
        self.username = username
        self.private_key, self.public_key = self.generate_key_pair()

    def generate_key_pair(self):
        # Gera um par de chaves RSA para o utilizador
        private_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size=2048,
            backend=default_backend()
        )
        public_key = private_key.public_key()
        return private_key, public_key
```

Figura 22 – generate_key_pair

A seguir, existe a função “sign_message” que recebe uma mensagem e assina com a chave privada do utilizador.

```python
def sign_message(self, message):
    # Assina uma mensagem usando a chave privada do utilizador
    signature = self.private_key.sign(
        message.encode(),
        padding.PSS(
            mgf=padding.MGF1(hashes.SHA256()),
            salt_length=padding.PSS.MAX_LENGTH
        ),
        utils.Prehashed(hashes.SHA256())
    )
    return signature
```

Figura 23 – sign_message

Por fim, existe ainda a função “verify_signature” que verifica se a assinatura fornecida corresponde à mensagem fornecida. A chave pública é utilizada para verificar a assinatura e retorna “True” caso essa verificação seja bem-sucedida.

```python
def verify_signature(self, message, signature, public_key):
    # Verifica a assinatura de uma mensagem usando a chave pública do remetente
    try:
        public_key.verify(
            signature,
            message.encode(),
            padding.PSS(
                mgf=padding.MGF1(hashes.SHA256()),
                salt_length=padding.PSS.MAX_LENGTH
            ),
            utils.Prehashed(hashes.SHA256())
        )
        return True
    except:
        return False
```

Figura 24 – verify_signature

A função “login_or_create_account” é responsável por tratar tanto da autenticação de utilizadores existentes quanto a criação de novas contas, o que garante o acesso à aplicação de forma mais controlada e segura.

```python
def login_or_create_account(self):
    # Verifica se o utilizador já existe, caso contrário, cria uma nova conta
    username = self.username_entry.text()
    if username in self.users:
        # Verificar se o certificado do utilizador é válido
        if self.ca.verify_certificate(username):
            self.current_user = self.users[username]
            QMessageBox.information(self, "Success", "Logged in as {}.".format(username))
        else:
            QMessageBox.critical(self, "Error", "Invalid certificate for {}.".format(username))
    else:
        # Emitir um certificado para o novo utilizador
        certificate = self.ca.issue_certificate(username, 10)
        if certificate:
            new_user = User(username)
            self.users[username] = new_user
            self.current_user = new_user
            self.save_users()  # Salva os novos dados de login
            QMessageBox.information(self, "Success", "Account created for {}.".format(username))
        else:
            QMessageBox.critical(self, "Error", "Failed to issue certificate for {}.".format(username))
```

Figura 25 – login_or_create_account

As funções “save_users” e “load_users” são responsáveis por guardar e carregar os dados de *login* dos utilizadores no ficheiro *JSON* associado.

```python
def save_users(self):
    # Guarda os dados de login dos utilizador num arquivo JSON
    with open('users.json', 'w') as f:
        user_data = {user.username: {"private_key": user.private_key.private_bytes(
                                        encoding=serialization.Encoding.PEM,
                                        format=serialization.PrivateFormat.TraditionalOpenSSL,
                                        encryption_algorithm=serialization.NoEncryption()
                                    ).decode('utf-8'),
                                    "public_key": user.public_key.public_bytes(
                                        encoding=serialization.Encoding.PEM,
                                        format=serialization.PublicFormat.SubjectPublicKeyInfo
                                    ).decode('utf-8')} for user in self.users.values()}
        json.dump(user_data, f, indent=4)

def load_users(self):
    # Carrega os dados de login dos utilizadores do arquivo JSON
    try:
        with open('users.json', 'r') as f:
            user_data = json.load(f)
        for username, keys in user_data.items():
            private_key = serialization.load_pem_private_key(
                keys["private_key"].encode('utf-8'),
                password=None,
                backend=default_backend()
            )
            public_key = serialization.load_pem_public_key(
                keys["public_key"].encode('utf-8'),
                backend=default_backend()
            )
            self.users[username] = User(username)
            self.users[username].private_key = private_key
            self.users[username].public_key = public_key
    except FileNotFoundError:
        pass
```

Figura 26 – save_users e load_users

Por último, a função “send_message” é responsável por enviar a mensagem assinada pelo utilizador atual.

```python
def send_message(self):
    # Envia uma mensagem assinada pelo utilizador atual
    if self.current_user:
        message = self.message_entry.text()
        signature = self.current_user.sign_message(message)
        self.messages_text.append("Sent: {}".format(message))
        self.messages_text.append("Signature: {}".format(signature.hex()))
        self.message_entry.clear()
    else:
        QMessageBox.critical(self, "Error", "No user logged in.")
```

Figura 27 – send_message

3. Conclusão

Apesar do esforço conjunto dos membros neste projeto, os resultados não foram os esperados. Verificou-se que o trabalho foi complexo e difícil de compreender e concluir. Como alunos do Mestrado de Engenharia de Dados sentimos que a fundação de alguns dos termos ou as bases necessárias para a construção do projeto, foram dadas em outras unidades curriculares do plano de estudo do Mestrado de Cibersegurança e Administração de Sistemas. No primeiro projeto já foi sentido esse problema, mas neste segundo realçou-se ainda mais esse facto.

Contudo, ao longo deste trabalho foi possível compreender as ameaças à informação e as estratégias utilizadas para as mitigar. A necessidade de proteger a privacidade e confidencialidade dos dados é cada vez mais importante dada a crescente quantidade de dados que o utilizador partilha nas diferentes plataformas.

Percebemos também que a segurança é uma jornada continua e que ferramentas como a encriptação e certificados são importantes para a segurança de cada utilizador individual.

O trabalho foi dividido de forma justa e equilibrada de forma que todos os elementos do grupo tivessem impacto no desenvolvimento e que o resultado cumprisse os requisitos pedidos no enunciado.

Referências

MoodleISEP. (n.d.). Moodle.isep.ipp.pt. Retrieved March 24, 2024, from https://moodle.isep.ipp.pt/course/view.php?id=4744

Schneier, B. (2007). *Applied cryptography*. John Wiley & Sons.

Ristic, I. (2022). *Bulletproof TLS and PKI*. London: Feisty Duck.

Campbell, B., & Bishop, M. (2023, July 1). *Client-Cert HTTP Header Field*. IETF. https://datatracker.ietf.org/doc/rfc9440/

Komar, B. (2008). *Windows Server 2008 PKI and Certificate Security*. Pearson Education.

What is PKI? A Public Key Infrastructure Definitive Guide. (n.d.). Keyfactor. https://www.keyfactor.com/education-center/what-is-pki/

What is Public Key Infrastructure (PKI)? | Thales. (n.d.). Cpl.thalesgroup.com. https://cpl.thalesgroup.com/faq/public-key-infrastructure-pki/what-public-key-infrastructure-pki

OpenSSL Foundation, Inc. (2019). */index.html*. Openssl.org. https://www.openssl.org/

O que é certificado SSL – definição e explicação. (2022, January 10). Www.kaspersky.com.br. https://www.kaspersky.com.br/resource-center/definitions/what-is-a-ssl-certificate

Cloudflare. (2021). What is SSL (Secure Sockets Layer)? | Cloudflare. *Cloudflare*. https://www.cloudflare.com/learning/ssl/what-is-ssl/