ŠIFROVANÁ KOMUNIKACE MEZI DVĚMA UŽIVATELI

Autoři: Petra Kajánková, Matúš Sádovský
Erik Chovanec, Samo Kopecký

Datum odevzdání: 15.11.2019

OBSAH:

1. Úvod a popis projektu str – 3
2. Cíle projektu, kterých má být dosaženo str – 4
3. Teoretická část popisující dané téma str – 4-6
4. Aktuálny stav riešenia str – 7-10
5. Záver str – 10
6. Autori a ich prínos do projekt str – 10

1. Úvod a popis projetu

V projektu je naším cílem vytvořit šifrovou komunikaci mezi dvěma uživateli. Přičemž jsou podmínky takové, že stanovení veřejného klíče bude probíhat přes asymetrickou kryptografii. Další podmínkou je, že zde bude certifikační autorita, která nám vytvoří certifikát a podepíše ho. Poslední podmínkou je stanovení soukromého klíče pro komunikaci, která bude probíhat přes symetrickou kryptografii.

V projektu budeme demonstrovat komunikaci mezi uživateli pomocí dvou konzolových oken. Naším prvním úkolem v projektu je vytvoření veřejného klíče, přes který budou uživatelé stanovovat klíč pro symetrickou kryptografii, kterým se pak bude šifrovat komunikace.

Tento konkrétní krok bude probíhat díky asymetrickému protokolu RSA. Uživatel číslo jedna si tedy vygeneruje soukromý a veřejný klíč.

Dalším úkolem bude podepsání certifikátu certifikační autoritou. Tento krok bude probíhat tak, že nejprve uživatel jedna pošle svoji žádost o certifikát vybrané certifikační autoritě. V tomto kroku je důležité, aby si obě strany vybrali certifikační autoritu, které budou oba uživatelé důvěřovat.

Certifikační autorita podepíše certifikát svým soukromým RSA klíčem, tak aby zde byla zajištěna autentičnost, autorita a žádný jiný uživatel se za ni nemohl vydávat. Následně tento certifikát odešle uživateli jedna.

Uživatel jedna si pomocí veřejného klíče RSA certifikační autority může zkontrolovat, či mu jej opravdu podepsala vybraná autorita. Následně tento certifikát odešle uživateli dva, který si opět může ověřit autentičnost certifikátu uživatele jedna.

Jestliže i tento krok proběhne v pořádku, uživatel dva vygeneruje klíč pro symetrické šifrování AES. Tento klíč je náhodně vygenerovaný. Aby tento klíč nemohl zjistit kdokoliv jiný, je třeba jej opět zašifrovat přes protokol RSA.

Uživatel tedy vezme z certifikátu veřejný klíč a zašifruje s ním vygenerovaný AES klíč. Následně tuto zprávu s klíčem odešle uživateli číslo jedna.

Uživatel číslo jedna teď musí dešifrovat přijatou zprávu. Tento krok uskuteční pomocí svého privátního klíče. Po dešifrování dojde k tomu, že obě strany nyní znají AES klíč a tak spolu nyní mohou komunikovat.

2. Cíle projektu, kterých by mělo být dosaženo

V tomto projektu by mělo být dosaženo toho, abychom vytvořili zabezpečenou komunikaci mezi dvěma uživateli.

3. Teoretická část popisující téma

V této části si přiblížíme dané kryptografické šifry společně s certifikační autoritou.

SYMETRICKÁ ŠIFRA AES - *Advenced Encryption standard*

Jedná se o symetrickou šifru, což znamená, že pro komunikaci je stanoven pouze jeden klíč. Tedy jedním klíčem se šifruje a opět dešifruje.

AES je federální standardem USA od roku 2002. Jeho původní název byl – Rijndael, který byl složen ze jmen jeho autorů, tedy Joana Daemena a Vincenta Rijmena.

Hlavní výhodou symetrických šifer je, že jsou rychlé a dají se tak použít pro šifrování velkého objemu dat.

Zásadní nevýhodou je ovšem samotné použití klíče, kdy jeden uživatel může šifrovat i dešifrovat, je tedy důležité aby si tento klíč předali důvěrnou cestou, kterou není možné odposlechnout.

AES je bloková šifra. Využívá stanovené délky klíčů pro komunikaci 128, 196, či 256 bitů. Tato šifra je aplikovaná na data s pevně danou délkou, což je 128 b. Pokud jsou data delší, jsou rozdělena do několika bloků. Jestliže jsou data kratší musí dojít k tak zvanému paddingu – doplnění. To může probíhat nejrůznějšími metodami. Nejjednodušší metoda je doplnění nulami, ale se moc často nepoužívá. Nejčastěji se používá mechanismus PKCS#7.

V praxi se nejčastěji používá CBC – otevřený text nejprve projde operací XOR předcházejícího zašifrovaného textu. První blok šifrován pomocí náhodně vygenerovaného nultého bloku – inicializační vektor.

Následné šifrování pak probíhá ve čtyřech krocích:

1. **SubBytes** – jedná se o jednoduchou substituci kde, každý byt je nahrazen jiným, dle předem daného klíče. Tento krok nám zajišťuje nelineárnost šifry a zabrání útoku na jednoduchých algebraických vlastnostech.
2. **ShiftRows** – v tomto kroku dojde k přeházení jednotlivých bitů (první řádek se zachová, druhý se přehází o jedno místo, třetí o dvě místa a čtvrtý o 3 místa).
3. **MixComlumns** – při této operaci jsou přeházeny sloupce a násobeny polynomem c(x).
4. **AddRoundKey** - každý byt zkomprimujeme se subklíčem, který získáme z původního klíče pomocí Rijndaelovy tabulky a tak získáme výslednou šifru.

ASYMETRICKÁ ŠIFRA RSA:

Asymetrická šifra je taková šifra, kde se nachází klíč pro šifrování, tak klíč pro dešifrování. Oproti symetrickým šifrám je pomalejší a tak se používá pro malé bloky dat, obvykle pro ustanovení klíčů. RSA je nejstarší a nejznámější asymetrický algoritmus. Je pojmenován dle svých třech autorů – Rivest, Shamir, Adleman. RSA je založeno na problému faktorizace velkých čísel. RSA se nejen používá pro šifrování, ale dokáže zajistit i autentičnost, takže se používá k podpisům. Klíče při šifrování: Jestliže šifru chceme využít při šifrování, je nutné tedy soukromým klíčem zprávu podepsat a veřejným klíčem zprávu dešifrovat. Při podpisu opět využijeme soukromý klíč a pomocí veřejného můžeme ověřit to, jestli daný dokument opravdu podepsala daná osoba.

Průběh – Nejprve vygenerujeme dostatečně velká prvočísla r a s.

- Následně vypočítáme n, které je rovno násobku prvočísel.
- Poté si spočítáme $\phi(n) = (r - 1)(s - 1)$.
- Z těchto parametrů si stanovým soukromý klíč tak, že jejich největší společný dělitel soukromého klíče $\phi(n)$ bude rovno jedna.
- Posledním krokem je stanovení veřejného klíče pomocí vztahu $Pk = Sk{-}1 \bmod \phi(n)$
- Podmínkou je, aby n mělo alespoň 2048 bitů, takže r a s musíme volit dostatečně veliká.

Certifikační autorita - CA:

Využívá se v asymetrické kryptografii pro vydání certifikátu k veřejnému klíči, tak svou autoritou potvrzuje pravdivost údajů, které jsou ve veřejném klíči uvedeny.

Certifikát je podepsán soukromým klíčem vydavatele certifikátu, aby nemohlo dojít k jeho přepsání. Uživatelé, kteří dostanou takto podepsaný klíč si mohou pomocí veřejného klíče autority ověřit, jestli skutečně klíč podepsala ona.

Existují celkem 3 třídy:

1. Certifikační autorita ručí pouze za jednoznačnost certifikátu. To znamená, že daný certifikát je vydán anonymnímu uživateli. Ten musí vyplnit formulář serveru a ten následně poslat přes HTTPS. Autorita tak nezajišťuje autentičnost.
2. CA kontroluje totožnost uživatele. Tento certifikát je vydám pouze po osobní návštěvě registrační sítě autorit.
3. Vydané certifikáty jsou pouze pro konkrétní aplikace.

Certifikát obsahuje jak údaje o majiteli certifikátu, tak i údaje o certifikační autoritě, ale i datum začátek platnosti certifikátu, tak i datum konce platnost

4. Aktuálny stav riešenia

Na začiatku si otvoríme 3 pracovné terminály kde vytvoríme dvoch užívateľov a jednu certifikačnú autoritu. Certifikačná autorita je schopná počúvať na zvolenom porte a prijímať požiadavky od užívateľov o vydanie certifikátu pre overenie identity. Užívateľ má funkciu, pomocou ktorej žiada o vydanie certifikátu od certifikačnej autority. Pri incializácii objektu užívateľa sa vygeneruje aj privátny a verejný kľúč.

```
def __init__(self):
    """
        Creating variables to store things in later in the code, or initializing them right away
        other_* = variables of the other user we are communicating with
        active_socket = socket to communicate trough
    """
    self.aes_key = bytes()
    self.aes_iv = bytes()
    self.cipher = utils.Cipher
    self.my_certificate = utils.crypto.X509()
    self.private_key, self.public_key = utils.generate_cryptography_rsa_keys()
    self.other_certificate = utils.crypto.X509()
    self.other_public_key = utils.rsa.RSAPublicKey
    self.active_socket = utils.socket.socket()
    self.name = input('enter your name : ')
    self.received_messages = []
```

Následne pri žiadaní o vydanie certifikátu si vytvorí objekt typu crypto.x509req, ktorý naplní svojimi údajmi a podpíše ho svojim privátnym kľúčom.

```
def create_certificate_request(self):
    """
        We create certificate request in this function and convert keys from one lib to another
        request.get_subject is our info that the CA will put as issuer
        :return: returns created certificate request
    """
    ssl_public_key = utils.crypto.PKey.from_cryptography_key(self.public_key)
    ssl_private_key = utils.crypto.PKey.from_cryptography_key(self.private_key)
    request = utils.crypto.X509Req()
    request.get_subject().countryName = 'CZ'
    request.get_subject().stateOrProvinceName = 'Czech Republic'
    request.get_subject().localityName = 'Brno'
    request.get_subject().organizationName = 'University of Technology'
    request.get_subject().organizationalUnitName = 'VUT'
    request.get_subject().commonName = '{}-vut.cz'.format(self.name)
    request.get_subject().emailAddress = '{}@vut.cz'.format(self.name)
    request.set_pubkey(ssl_public_key)
    request.sign(ssl_private_key, 'sha256')
    return request
```

Potom sa pomocou ďalšej metódy vytvorí spojenie s certifikačnou autoritou kde sa žiadosť prekonvertuje do formátu PEM a pošle ho certifikačnej autorite.

```
def send_request_to_ca(self):
    """
        In here we get ready for communication, convert the certificate request to PEM format so
        that it can be sent and then we send it and then we close the connection with CA
    """
    client_socket = utils.start_sending()
    utils.send_data(client_socket, b'sending cert request', 'request to start communication')
    data_to_send = utils.crypto.dump_certificate_request(
        utils.PEM_FORMAT,
        self.create_certificate_request()
    )
    utils.send_data(client_socket, data_to_send, 'cert req')
    data = utils.receive_data(client_socket, 'cert')
    self.my_certificate = utils.crypto.load_certificate(utils.PEM_FORMAT, data)
    utils.finish_connection(client_socket)
```

Skôr ako certifikačná autorita bude komunikovať je nutné stanoviť, ako už bolo spomenuté, na akom porte bude certifikačná autorita počúvať, na čo slúži prvá spustená metóda pri spúšťaní certifikačnej autority.

```
def use_ca():
    """
        first function to run when CA.py is ran and before we start the infinite loop we
        get the the port to listen to for the rest of the runtime so we don't have to enter
        it every time
    """
    ca = CA()
    port = int(input('choose port to listen to : '))
    while True:
        ca.receive_certificate_request(port)
```

Certifikačná autorita teda už môže na základe prijatých užívateľovych údajov overiť užívateľa. Najskôr odošle správu užívateľovi s potvrdením nadviazania komunikácie a ako prvé pri prijatí žiadosti prekonvertuje túto žiadosť vo formáte PEM naspäť do formátu x509Req.

```
def send_certificate(self, connection):
    """
        first we convert the certificate request form PEM to x509Req format, create the certificate
        send the certificate back to the user
        :param connection: port to send data with
    """
    print('ready to accept, sending ack')
    utils.send_acknowledgement(connection)
    data = utils.receive_data(connection, 'cert req')
    cert_req = utils.crypto.load_certificate_request(utils.PEM_FORMAT, data)
    cert = self.create_certificate_from_request(cert_req)
    data_to_send = utils.crypto.dump_certificate(utils.PEM_FORMAT, cert)
    utils.send_data(connection, data_to_send, 'cert')
```

Pred tým, než certifikačná autorita vytvorí certifikát, vyplní informácie o sebe a podpíše ho svojim súkromným kľúčom. Následne certifikát prekonvertuje do formátu PEM a odošle certifikát používateľovi. Obdobne funguje vytvorenie certifikátu aj s ďalším užívateľom.

```
def create_certificate_from_request(self, request):
    """
        before we create the certificate from the certificate request we fill the CA's info as a subj
        :param request: certificate request
        :return: returns created certificate
    """
    self.verify_certificate_request(request)
    cert = utils.crypto.X509()
    cert.set_serial_number(1000)
    cert.get_subject().countryName = 'CZ'
    cert.get_subject().stateOrProvinceName = 'Czech Republic'
    cert.get_subject().localityName = 'Brno'
    cert.get_subject().organizationName = 'University of Technology'
    cert.get_subject().organizationalUnitName = 'VUT'
    cert.get_subject().commonName = 'CA-vut.cz'
    cert.get_subject().emailAddress = 'CA@vut.cz'
    cert.set_issuer(request.get_subject())
    cert.gmtime_adj_notBefore(0)
    cert.gmtime_adj_notAfter(60 * 60 * 24)  # 24 hours
    cert.set_pubkey(request.get_pubkey())
    cert.sign(self.private_key, 'sha256')
    self.list_of_certs.append(cert)
    return cert
```

Keď už obaja užívatelia majú svoje certifikáty môže sa vytvoriť spojenie medzi jednotlivými užívateľmi, pomocou ktorého budú spolu komunikovať. To znamená, že sme najskôr vytvorili metódu pre vytvorenie spojenia medzi danými užívateľmi. Vytvorenie spojenia prebieha obdobne ako vytváranie spojenia s certifikačnou autoritou. Užívatelia si stanovujú, že kto bude prijímať a kto posielať správy a na akom porte bude komunikácia prebiehať. V priebehu vytvárania spojenia si užívatelia navzájom posielajú správy s potvrdením prijatia správ pre vytvorenie spojenia. Akonáhle je spojenie vytvorené užívatelia si najskôr vymenia certifikáty pre overenie identity či je podpísaný požadovanou certifikačnou autoritou. Po overení identity užívateľov navzájom je užívateľom vygenerovaný zdieľaný kľúč AES, zašifrovaný pomocou privátneho kľúču RSA a poslaný druhému užívateľovi s inicializačným nezašifrovným vektorom.

```
def sending_aes_key(self):
    """
        this method generates the shared AES key and vector which it will then send to the other user
        encrypted with RSA
        key is 32 bytes long
        iv is 16 bytes long and can be sent in plain text
    """
    self.aes_key, self.aes_iv = os.urandom(32), os.urandom(16)
    self.other_public_key = utils.convert_key_from_ssl_to_cryptography(self.other_certificate.get_pubkey())
    data_to_send = utils.rsa_encrypt(self.aes_key, self.other_public_key)
    utils.send_data(self.active_socket, data_to_send, 'aes key')
    utils.send_data(self.active_socket, self.aes_iv, 'aes iv')
```

Druhý užívateľ zdieľaný kľúč po prijatí dešifruje.

5. Záver

Pre dokončenie projektu je potrebné implementovať tieto funkcionality:

- Preposielanie zašifrovaných správ a súborov pomocou zdieľaného AES kľúču
- Verifikácia vymenených certifikátov u certifikačnej autority
- Verifikácia žiadosti o certifikát certifikačnou autoritou pri vytváraní certifikátu

6. Autori a ich prínos do projektu

Na tomto projekte sme pracovali ako tým a každý týždeň sme sa stretávali a pracovali na projekte spolu.