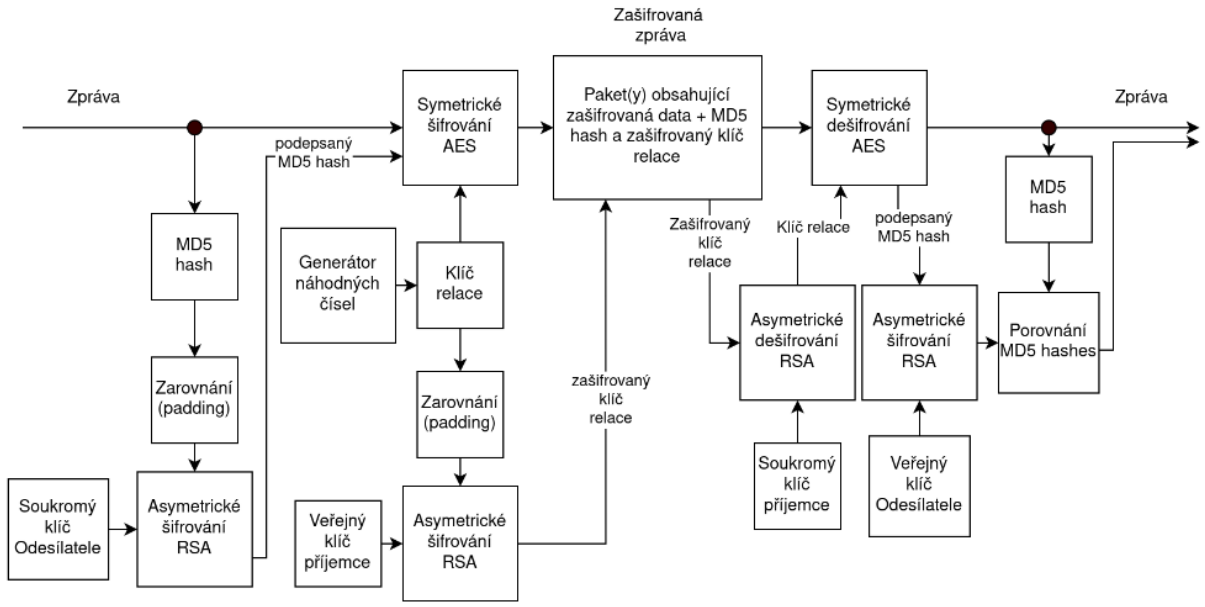
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Kryptografie 2. Projekt 2023 / 24

Bc. Petr Pouč - xpoucp01

21. dubna 2023

Cílem projektu bylo vytvořit hybridní šifrování pro komunikaci mezi klientem a serverem. Při implementaci hybridního šifrování byly použity algoritmy AES 128 bit, RSA 2048 a MD5 hash.



Obrázek 1: Schéma implementovaného hybridního šifrování

Vytvoření klíčů

Pro symetrickou kryptografii se používá funkce `generate_session_key()`, která vytváří náhodný 128 bitový klíč pomocí funkce `get_random_bytes()`.

Pro asymetrickou kryptografii se používají klíče RSA. Generování klíčů probíhá ve funkci `generate_rsa_keys()`, která vytvoří dvě dvojice klíů, jednu pro odesílatele a druohu pro příjemce. Klíče jsou následně uloženy ve složce cert jako sender_id_rsa (soukromý klíč odesílatele), id_rsa.pub (veřejný klíč), receiver_id_rsa (soukromý klíč odesílatele), id_rsa.pub (veřejný klíč).

Symetrický klíč (klíč relace) se používá k šifrování zpráv AES v režimu EAX. Je generován při každé nové komunikaci mezi klientem a serverem a následně je šifrován RSA veřejným klíčem příjemce a poslán serveru.

Asymetrické klíče jsou použity pro šifrování a dešifrování symetrického klíče použitého pro šifrování zpráv. Veřejný klíč je použit klientem k šifrování symetrického klíče a soukromý klíč je použit serverem k dešifrování symetrického klíče.

Dále je použit soukromý klíč použit k asymetrickému zašifrování MD5 hashe.

Padding

K vyplnění velikosti bloku šifrovacího algoritmu jsem implementoval vlastní padding na principu OAEP. Padding se aplikuje za hash. Hash doplňuji náhodnými čísly na délku 255 bajtů.

```
def pad_hash(hash):
    current_length = len(hash)
    padding_length = 255 - current_length
    padding = get_random_bytes(padding_length)
    padded_hash = hash + padding
    return padded_hash
```

Zajištění integrity

Pro zajištění integrity je k odeslané zprávě přidán asymetricky šifrovaný MD5 hash. Ten je dále spolu s klíčem relace a zprávou symetricky šifrován AES šifrou.

Po příjmu zprávy je použit soukromý klíč příjemce k dešifrování klíče pro symetrickou šifru AES (klíče relace). Pomocí něj je následně provedeno dešifrování paketu, který byl šifrován pomocí AES. Z něj lze získat podepsaný MD5 hash, který nám pomůže určit zachování integrity zprávy. Pokud se získaný MD5 hash rovná zakódovanému MD5 hashi zprávy, zpráva je považována za platnou.

```
decrypted_message = cipher.decrypt(ciphertext)
#extract and decrypt MD5 hash
received_md5 = int.from_bytes(decrypted_message[-256:], byteorder='big')
received_md5 = public_key._encrypt(received_md5)
received_md5_bytes = received_md5.to_bytes((received_md5.bit_length() + 7) // 8, 'big')

#remove MD5 hash from message, only first 16 bytes are hash
recv_md5_cut = received_md5_bytes[:16]
received_md5_bytes = received_md5.to_bytes((received_md5.bit_length() + 7) // 8, 'big')

 if recv_md5_cut != md5_hash:
    print("The integrity of the report has been compromised.")
    # send NACK
    client_socket.send(b'NACK')
    break
```

K šifrovování a dešifrování pomocí algoritmu RSA jse použil funkce `_encrypt` a `_decrypt` z modulu RSA.

Pro zajištění doručení (a přečtení) celého paketu přikládám k zašifrované zprávě velikost celého paketu, kterou šifruji na 4 bajty. Server po přijetí dat tedy extrakture oekávanou délku paketu a data jsou načítaná dokud není zpracován celý paket.

```
while True:
    print("s: \"Client has joined\"")
    recv_buf = b''
    try:
        # receive data from client in while loop, until it reaches end
        packet = b''
        while True:
            new_recv = client_socket.recv(4096)
            print(f"s: Received {len(new_recv)} bytes from client")

            if not new_recv: # client disconnected
                break

            recv_buf += new_recv
            #extract first 4 bytes as length of message
            print(f"Now have {len(recv_buf)} bytes in buffer")

            length = int.from_bytes(recv_buf[:4], byteorder='big')
            print(f"Message length is {length}")
            if len(recv_buf) < length + 4:
                continue # not enough data

            # extract message
            packet = recv_buf[4:length + 4]
            # remove message from buffer
            recv_buf = recv_buf[length + 4:]
            break
```

Spojení je ukončeno zadáním prázdné zprávy.

Výměna klíčů

Proces výměny klíčů může probíhat několika různými způsoby. V běžném svetě se pro ověření podepsaných klíčů využívá certifikační autorita (CA), která zajišťuje důvěryhodnost veřejných klíčů a potvrzuje totožnost majitelů klíčů. CA v implementaci zahrnuta není, je počítano s tím, že si strany důvěřují a klíče si vyměnily bezpečně.

Zhodnocení bezpečnosti

Bezpečnost tohoto hybridního šifrování by mohla být ohrožena, pokud by útočník získal přístup k soukromému klíči. Délka klíče je 128 bitů pro AES a 2048 bitů pro RSA. Tyto délky jsou obecně považovány za dostatečně bezpečné a jsou běžně používány v moderní kryprografii.

Potenciálním vylepšením této implementace hybridního šifrování by bylo nahrazení šifrování pomocí MD5, o kterém se ví, že není bezpečné. Tento algoritmus má několik nevýhod, jako jsou například kolidující hashe (mohou existovat 2 stejné zprávy, které mají stejný hash), není příliš odolný vůči útokům hrubou silou, nebo útokům s využitím kolizí. Vhodnější by bylo pro tyto účely použít například SHA-256.