Application of Cryptography

Yunfan Tian

Date: 10/05/2018

This cryptography application implenments **pretty good privacy** protocol to enceypt and decrypt plaintext

Pretty Good Privacy (PGP) is an encryption program that provides cryptographic privacy and authentication for data communication.[1]

Sender encrypt:

1. plaintext --> padding --> AES256(aes_key, aes_iv) encrypt = ciphertext
2. aes_key --> RSA2048(dst_public.pem) encrypt = encrypted_aes_key
3. ciphertext + encrypt_ae_key --> RSA2048(send_private.pem) sign= signature
4. signature + encrypted_aes_key + aes_iv + ciphertext = output_ciphertext

Destnation decrypt:

1. output_ciphertext --> verify signature using RSA2048(send_public.pem)
2. encrypted_aes_key --> RSA2048(dst_private.pem) = aes_key
3. ciphertext --> AES256(aes_key, aes_iv) --> unpadding = output_plaintext

Details

Generate two pairs of RSA keys for encrypt and signature

keysize: 2048

```
>> openssl genrsa -out send_private.pem 2048
>> openssl genrsa -out dst_private.pem 2048
>> openssl rsa -in send_private.pem -outform PEM -pubout -out
send_public.pem
>> openssl rsa -in dst_private.pem -outform PEM -pubout -out
dst_public.pem
```

Sender Encrypt

plaintext --> padding = padder_data

padding block: 128

```
plaintext = f.read()
padder = padding.PKCS7(algorithms.AES.block_size).padder()
padded_data = padder.update(plaintext) + padder.finalize()
```

AES encrypt:

keysize: 256 bits; initial vector: 128 bits

mode: CBC

```
# key and iv for AES cbc mode
aes_key = os.urandom(32)
aes_iv = os.urandom(16)
# AES cbc mode
cipher = Cipher(algorithms.AES(aes_key), modes.CBC(aes_iv),
backend=default_backend())
encryptor = cipher.encryptor()
ciphertext_1 = encryptor.update(padded_data) + encryptor.finalize()
```

RSA (dst_public.pem) encrypt aes_key using OAEP padding

```
encrypted_aes_key = public_key.encrypt(
    aes_key,
    asyc_padding.OAEP(
        mgf=asyc_padding.MGF1(algorithm=hashes.SHA1()),
        algorithm=hashes.SHA1(),
        label=None
    )
)
```

Use RSA(send_private.pem) to sign ciphertext + aes_iv +encrypted_aes_key = signature

```
message = base64.b64encode(encrypted_aes_key) + '\n' + \
          base64.b64encode(aes_iv) + '\n' + \
          base64.b64encode(ciphertext_1)

signature = private_key.sign(
    message,
    asyc_padding.PSS(
        mgf=asyc_padding.MGF1(hashes.SHA256()),
        salt_length=asyc_padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
```

signature + encrypted_aes_key + aes_iv + ciphertext = output_ciphertext

```
with open(OUTPUT_FILE_NAME, 'wb') as f:
    f.write(base64.b64encode(signature))
    f.write('\n' + message)
```

Destination Decrypt

output_ciphertext --> verify signature using RSA2048(send_public.pem)

```
public_key.verify(
    signature,
    signed_message,
    asyc_padding.PSS(
        mgf=asyc_padding.MGF1(hashes.SHA256()),
        salt_length=asyc_padding.PSS.MAX_LENGTH
    ),
    hashes.SHA256()
)
```

encrypted_aes_key --> RSA2048(dst_private.pem) = aes_key

```
# decrypt the aes key
aes_key = private_key.decrypt(
    encrypted_aes_key,
    asyc_padding.OAEP(
        mgf=asyc_padding.MGF1(algorithm=hashes.SHA1()),
        algorithm=hashes.SHA1(),
        label=None
    )
)
```

ciphertext --> AES256(aes_key, aes_iv) --> unpadding = output_plaintext

```
cipher = Cipher(algorithms.AES(aes_key), modes.CBC(aes_iv),
backend=default_backend())
decryptor = cipher.decryptor()
res = decryptor.update(ciphertext) + decryptor.finalize()
# unpadding
unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
plaintext = unpadder.update(res) + unpadder.finalize()
```

---

1. Pretty Good Privacy. https://en.wikipedia.org/wiki/Pretty_Good_Privacy↩