Public Key Cryptography

Introduction

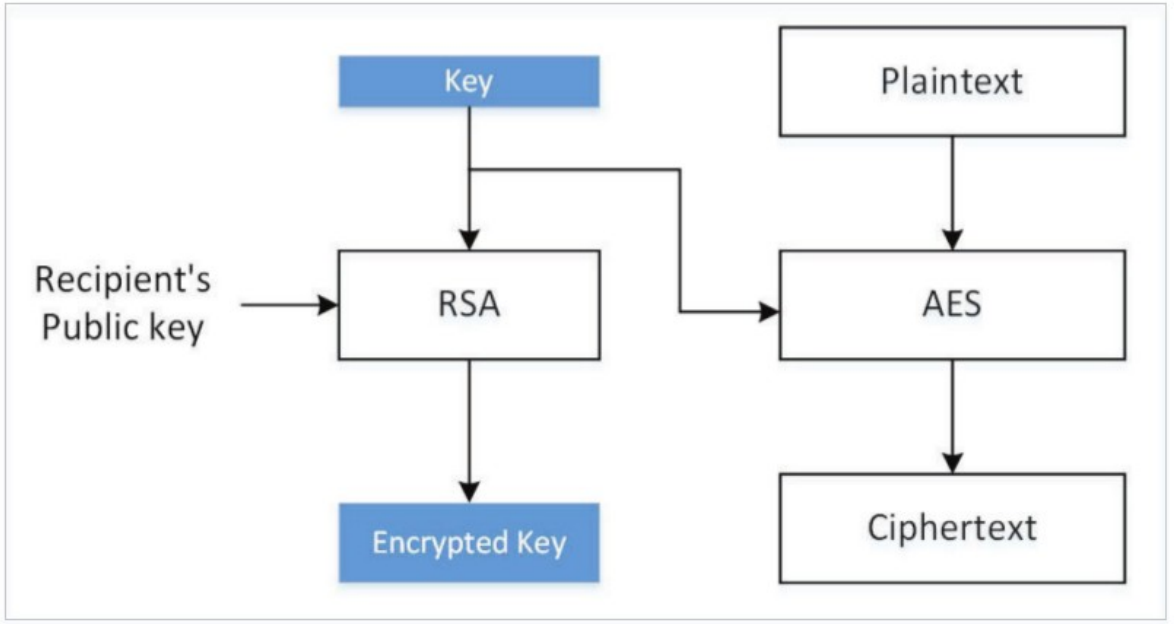
- Foundation of today’s secure communication
- Allows communicating parties to obtain a shared secret key
- Public key (for encryption) and Private key (for decryption)
- Private key (for digital signature) and Public key (to verify signature)

Brief History Lesson

- Historically same key was used for encryption and decryption
- Challenge: exchanging the secret key (e.g. face-to-face meeting)
- 1976: Whitfield Diffie and Martin Hellman
  - key exchange protocol
  - proposed a new public-key cryptosystem
- 1978: Ron Rivest, Adi Shamir, and Leonard Adleman (all from MIT)
  - attempted to develop a cryptosystem
  - created RSA algorithm

Hybrid Encryption

- High computation cost of public-key encryption
- Public key algorithms used to exchange a secret session key
- Key (content-encryption key) used to encrypt data using a symmetric-key algorithm



Using OpenSSL Tools to Conduct RSA Operations

We will cover:

- Generating RSA keys
- Extracting the public key
- Encryption and Decryption

OpenSSL Tools: Generating RSA keys

Example: generate a 1024-bit public/private key pair

- `openssl genrsa -aes128 -out private.pem 1024`
- private.pem: Base64 encoding of DER generated binary output

```
$ more private.pem
-----BEGIN RSA PRIVATE KEY-----
MIICWgIBAAKBgQCuXJawrRzJNG9vt2Zqe+/TCT3OxuEKRWkHfE5uZBkLCMgGbYzK
...
mesOrjIfm0ljUNL4VRnrLxrl/1xEBGWedCuCPqeV
-----END RSA PRIVATE KEY-----
```

OpenSSL Tools: Generating RSA keys (Contd.)

Actual content of private.pem

```
$ openssl rsa -in private.pem -noout -text
Enter pass phrase for private.pem:
Private-Key: (1024 bit)
modulus:
    00:c4:5a:9d:8d:f7:ad:0d:e7:60:4e:b3:9c:76:93: ...
publicExponent: 65537 (0x10001)
privateExponent:
    00:a5:86:fe:6b:3f:f0:53:58:4a:88:0e:42:48:74: ...
prime1:
    00:ec:a0:f7:02:8d:79:a0:8b:c5:5b:e6:a0:25:2c: ...
prime2:
    00:d4:6d:9c:4a:35:6b:fb:db:42:20:d8:6e:45:a9: ...
exponent1:
    06:72:d4:88:73:46:8f:43:7f:db:63:4b:95:f7:c4: ...
exponent2:
    00:d1:3c:45:bd:32:71:72:59:bd:00:ed:2d:70:a0: ...
coefficient:
    22:f5:95:05:81:c4:fd:3e:52:99:16:b5:66:92:52: ...
```

OpenSSL Tools: Extracting Public Key

- `openssl rsa -in private.pem -pubout > public.pem`
- Content of public.pem:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDEWp2N960N52BOs5x2k53WglVn
iAv5oUemZdfnGP1qUhTMZfhSbD27eOUJZAEdrMS/4Nax/BJIxz6N+L2K2cQQasJY
Gqf1PetXKtYakzgd5dBuB3aogOTJaBSt8/A0DBK2MtwNMnBxeZWnf4DK8Glsbp2S
nsGmCdceQ4nelGZbIwIDAQAB
-----END PUBLIC KEY-----
```

```
$ openssl rsa -in public.pem -pubin -text -noout
Public-Key: (1024 bit)
Modulus:
    00:af:1a:d9:ca:91:91:6b:b6:d0:1d:56:7a:1b:2d: ...
Exponent: 65537 (0x10001)
```

OpenSSL Tools: Encryption and Decryption

- Plain Text

```
$ echo "This is a secret." > msg.txt
```

- Encryption

```
$ openssl rsautl -encrypt -inkey public.pem -pubin \
                 -in msg.txt -out msg.enc
```

- Decryption

```
$ openssl rsautl -decrypt -inkey private.pem -in  msg.enc
Enter pass phrase for private.pem:
This is a secret.
```

Paddings for RSA

- Secret-key encryption uses encryption modes to encrypt plaintext longer than block size.
- RSA used in hybrid approach (Content key length << RSA key length)
- To encrypt:
  - short plaintext: treat it a number, raise it to the power of e (modulo n)
  - large plaintext: use hybrid approach (treat the content key as a number and raise it to the power of e (modulo n)
- **Treating plaintext as a number and directly applying RSA is called plain RSA or textbook RSA**

Attacks Against Textbook RSA

- RSA is deterministic encryption algorithm
  - same plaintext encrypted using same public key gives same ciphertext
  - secret-key encryption uses randomized IV to have different ciphertext for same plaintext
- For small e and m
  - if m^e < modulus n
  - e-th root of ciphertext gives plaintext
- If same plaintext is encrypted e times or more using the same e but different n, then it is easy to decrypt the original plaintext message via the Chinese remainder theorem

Paddings: PKCS#1 v1.5 and OAEP

- Simple fix to defend against previous attacks is to add randomness to the plaintext before encryption
- Approach is called padding
- Types of padding:
  - PKCS#1 (up to version 1.5): weakness discovered since 1998
  - Optimal Asymmetric Encryption Padding (OAEP): prevents attacks on PKCS
- `rsautl` command provides options for both types of paddings (PKCS#1 v1.5 is default)

PKCS Padding

- Plaintext is padded to 128 bytes
- Original plaintext is placed at the end of the block
- Data inside the block (except the first two bytes) are all random numbers
- First byte of the padding is always 00 (so that padded plaintext as integer is less than modulus n)
- Second byte is 00, 01, and 02 (different strings used for padding for different types)

PKCS Padding (Contd.)

```
$ openssl rsautl -encrypt -inkey public.pem -pubin \
                 -in msg.txt -out msg.enc -pkcs


$ openssl rsautl -decrypt -inkey private.pem \
                 -in msg.enc -out newmsg.txt -raw

$ xxd newmsg.txt
00000000: 0002 1b19 331a 1ea8 049e 8667 3b55 057c  ....3......g;U.|
00000010: 1072 e2bb 0aca 9af0 dd0e 5706 b34d e4a3  .r........W..M..
00000020: 7df6 b4d3 5f9b 8303 5ce7 67ee 150e 0fe1  }..._...\.g.....
00000030: f73f 6dc4 af36 117d 0d63 72f1 88f2 337f  .?m..6.}.cr...3.
00000040: 100b afac 8b26 fa65 d5a6 10b3 cf10 0b35  .....&.e.......5
00000050: 171b 9cc2 3409 c3b6 d953 a8a4 4617 4356  ....4....S..F.CV
00000060: 3f5f 1a91 9a97 5863 eae2 8ec5 4a00 5468 ?_....Xc....J.Th
00000070: 6973 2069 7320 6120 7365 6372 6574 2e0a  is is a secret..
```

OAEP Padding

- Original plaintext is not directly copied into the encryption block
- Plaintext is XORed with a value derived from random padding data

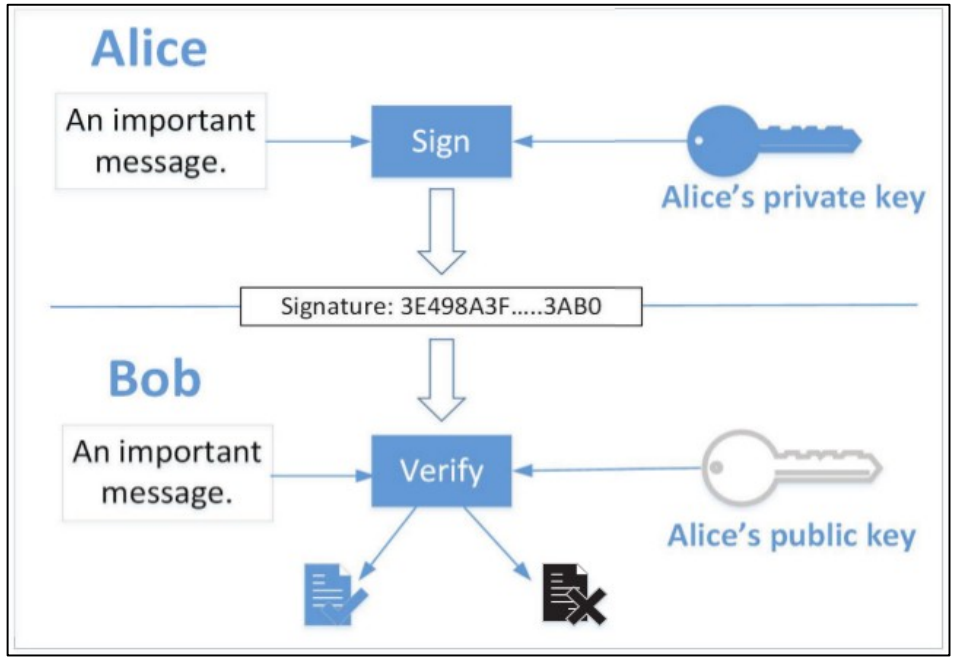
```
$ openssl rsautl -encrypt -inkey public.pem -pubin \
                 -in msg.txt -out msg.enc -oaep

$ openssl rsautl -decrypt -inkey private.pem \
                 -in msg.enc -out newmsg.txt -raw

$ xxd newmsg.txt
00000000: 006f 5f5e 5e0d e813 7fb0 3d45 e1ed d4fa  .o_^^.....=E....
00000010: 0688 1196 bb47 4501 b815 8922 51a0 5184  .....GE...."Q.Q.
00000020: d6b1 9819 4c00 07d1 b985 0248 8822 7b4f  ....L......H."{O
00000030: 8470 b195 1e4e 288f db91 f905 9d70 01de  .p...N(......p..
00000040: e0f4 5b4c 5b8a 26df 7031 b4a6 6547 d07d  ..[L[.&.p1..eG.}
00000050: e8ca 0006 3b65 a3ba 0f9f f865 6e80 6e0d  ....;e.....en.n.
00000060: 04ff 82a1 2c0b 3d1d 8d63 19b1 56f7 14f8  ....,.=..c..V...
00000070: 880e d003 d0e8 003c 9818 b083 7ba0 c6e6  .......<....{...
```

Digital Signature

- Goal: provide an authenticity proof by signing digital documents
- Diffie-Hellman authors proposed the idea, but no concrete solution
- RSA authors developed the first digital signature algorithm



Digital Signature using RSA

- Apply private-key operation on m using private key, and get a number s, everybody can get the m back from s using our public key
- For a message m that needs to be signed:

  Digital signature = $m^d \bmod n$

- In practice, message may be long resulting in long signature and more computing time
- Instead, we generate a cryptographic hash value from the original message, and only sign the hash

Digital Signature using RSA (Contd.)

Generate message hash

```
# Generate the hash from the message
$ openssl sha256 -binary msg.txt > msg.sha256
$ xxd msg.sha256
00000000: 8272 61ce 5ddc 974b 1b36 75a3 ed37 48cd  .ra.]..K.6u..7H.
00000010: 83cd de93 85f0 6aab bd94 f50c db5a b460  ......j......Z.`
```

Digital Signature using RSA (Contd.)

Generate and verify the signature

```
# Sign the hash
$ openssl rsautl -sign -inkey private.pem -in msg.sha256 -out msg.sig

# Verify the signature
$ openssl rsautl -verify -inkey public.pem -in msg.sig -pubin \
                 -raw | xxd
00000000: 0001 ffff ffff ffff ffff ffff ffff ffff  ................
00000010: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000020: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000030: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000040: ffff ffff ffff ffff ffff ffff ffff ffff  ................
00000050: ffff ffff ffff ffff ffff ffff ffff ff00  ................
00000060: 8272 61ce 5ddc 974b 1b36 75a3 ed37 48cd  .ra.]..K.6u..7H.
00000070: 83cd de93 85f0 6aab bd94 f50c db5a b460  ......j......Z.`
```

Attack Experiment on Digital Signature

- Attackers cannot generate a valid signature from a modified message because they do not know the private key
- If attackers modifies the message, the hash will change and it will not be able to match with the hash produced from the signature verification
- Experiment: modify 1 bit of signature file msg.sig and verify the signature

Attack Experiment on Digital Signature (Contd.)

After applying the RSA public key on the signature, we get a block of data that is significantly different

```
$ openssl rsautl -verify -inkey public.pem -in msg.sig -pubin \
                 -raw | xxd
00000000: 8116 cdc6 6b45 bcfc 98c3 7b09 514e 82fd  ....kE....{.QN..
00000010: 88a2 170b 414d 1ce8 7d18 d031 f03e db9f  ....AM..}..1.>..
00000020: 6f0f 3209 c1bc d2a6 a9d9 3f06 1e2c f970  o.2.......?..,.p
00000030: 1d90 ae31 bc5c 010d de8b 9a4b 6060 71b6  ...1.\.....K``q.
00000040: 71ce 43eb 505e 7759 42b9 e6c1 6bf5 06b9  q.C.P^wYB...k...
00000050: bd70 94fd 990f 2261 1257 76c2 7441 cbe0  .p....."a.Wv.tA..
00000060: 8538 8d9d 753e 4bd0 5c16 cb9c 57ea 8b62  .8..u>K.\...W..b
00000070: f804 76a2 d33b 7044 4ec7 93aa 56eb c0c1  ..v..;pDN...V...
```

Applications

- Authentication
- HTTPS and TLS/SSL
- Chip Technology Used in Credit Cards