Kryptoanalyse Laborarbeit

Aufgabe 5 – Bonusaufgabe

2024-12-02

Hintergrund

Wir haben in der Vorlesung eine Möglichkeit Besprochen, eine Backdoor in RSA-Schlüssel einzubauen. In diesem Dokument wird „Glasskey“ beschrieben, eine leicht abgeänderte (insbesondere voll deterministische) Variante des in der Vorlesung besprochenen Systems.

`glasskey_prng`: PRNG

Wir definieren zunächst einen PRNG, welcher Binärdaten in einem kontinuierlichen Strom liefert. Hierfür gibt es eine Funktion, die auf Basis von HMAC-SHA256, dem `agency_key` K und dem `seed` s Binärdatenblöcke B_i der Länge 32 Bytes erzeugt:

Algorithm 1 PRNG Block-Erzeugung

Eingabe: Schlüssel K, Seed s, Blocknummer $i = 0 \dots 2^{64} - 1$
Ausgabe: 32 Bytes an PRNG-Daten

1: **procedure** GLASSKEY_PRNG_BLOCK(K, s, i)
2: $i^* \leftarrow$ INT2BYTESLITTLEENDIAN(i)
3: $K^* \leftarrow$ SHA256(K) || SHA256(s)
4: **return** HMAC-SHA256($m = i^*, K = K^*$)
5: **end procedure**

Ein kontinuierlicher Keystrom `glasskey_prng` wird definiert als Sequenz von Bytes die mit aufsteigenden Zählerwerten i, beginnend ab $i = 0$, aneinanderkonkateniert würden. Hierbei wird also, entgegen der Definition in der Vorlesung, pro HMAC-Aufruf immer 32 statt nur einem Byte erzeugt. Ihre PRNG-Funktion soll mehrere aufeinanderfolgende Datenströme mit deren respektiven Längen l_k zurückliefern. Beachten Sie, dass Sie hierfür den PRNG nur einmal an Anfang seeden dürfen.

Testfall:

```
{
    "action": "glasskey_prng",
    "arguments": {
        "agency_key": "T01HV1RG",
        "seed": "ur1EoxDElJs=",
        "lengths": [
            4,
            8,
            13,
            12,
            1,
            9
        ]
    }
}
```

Erwartete Antwort:

```
{
    "blocks": [
        "9q32ZQ==",
        "r2I4mx+M13E=",
        "RvMXtSbjaKkuBXoUsQ==",
        "gwdanlsoBDPlMuzk",
        "ZQ==",
        "P3ixhiNIbxur"
    ]
}
```

`glasskey_prng_int_bits`: Integerwerte aus dem PRNG

Aus dem Datenstrom sollen nun ganzzahlige Werte extrahiert werden:

Algorithm 2 PRNG Ganzzahl-Erzeugung

Eingabe: Schlüssel K, Seed s, Maximalbitlänge b
Ausgabe: Ganzzahliger Wert der Bitlänge $1 \dots b$

1: **procedure** GLASSKEY_PRNG_INT_BITS(K, s, b)
2: $l \leftarrow \lceil \frac{b}{8} \rceil$
3: $s \leftarrow$ GLASSKEY_PRNG(l)
4: $s^* \leftarrow$ BYTES2INTLITTLEENDIAN(s)
5: **return** EXTRACTLOWESTBITS(s^*, b)
6: **end procedure**

```
{
    "action": "glasskey_prng_int_bits",
    "arguments": {
        "agency_key": "T01HV1RG",
        "seed": "ur1EoxDElJs=",
        "bit_lengths": [
            5,
            6,
            7,
            8
        ]
    }
}
```

Erwartete Antwort:

```
{
    "ints": [
        22,
        45,
        118,
        101
    ]
}
```

`glasskey_prng_int_min_max`: Ganzzahlen in Bereich

Während wir für die vorige Aufgabe nur Ganzzahlen benötigen, die Werte bis $2^b - 1$ annehmen können sollen nun Ganzzahlen in einem beliebigen Wertebereich $m \ldots M$ zurückgeliefert werden können.

Algorithm 3 PRNG Ganzzahl-Erzeugung in spezifischem Bereich

Eingabe: Schlüssel K, Seed s, Minimalwert m, Maximalwert M
Ausgabe: Ganzzahliger Wert r mit $m \leq r \leq M$

```
1:  procedure GLASSKEY_PRNG_INT_MIN_MAX(K, s, m, M)
2:      s ← M − m + 1
3:      b ← BITCOUNT(s)
4:      while true do
5:          r ← GLASSKEY_PRNG_INT_BITS(b)
6:          if r < s then
7:              return r + m
8:          end if
9:      end while
10: end procedure
```

```
{
    "action": "glasskey_prng_int_min_max",
    "arguments": {
        "agency_key": "T01HV1RG",
        "seed": "ur1EoxDElJs=",
        "specification": [
            {
                "min": 0,
                "max": 10
            },
            {
                "min": 12,
                "max": 14
            },
            {
                "min": 0,
                "max": 3
            },
            {
                "min": 0,
                "max": 1
            },
            {
                "min": 0,
                "max": 0
            },
            {
                "min": 14,
                "max": 99
            }
        ]
    }
}
```

Erwartete Antwort:

```
{
    "ints": [
        6,
        13,
        2,
        0,
        0,
        84
    ]
}
```

`glasskey_genkey`: Generierung eines Schlüsselpaars

Um ein Schlüsselpaar zu erzeugen wird folgende Funktion definiert, welche mindestens 128 Bit lange RSA Private Keys erzeugt und immer einen statischen 64-Bit Seed verwendet, der in den MSBs des Modul n auftauchen muss. Ihnen ist garantiert, dass die zwei MSBs des Seeds den Wert 1 haben.

Algorithm 4 Backdoor Schlüsselerzeugung

Eingabe: Schlüssel K, 64 Bit Seed s, Bitlänge l von n
Ausgabe: Primzahlen p, q sodass $n = pq$ und $\texttt{MSB}(n) = s$

procedure GLASSKEY_GENKEY(K, s, l)
  $l_p \leftarrow \lfloor \frac{l}{2} \rfloor$ ▷ Bitlänge von p
  $p \leftarrow$ GLASSKEY_PRNG_INT(l_p)
  $p \leftarrow$ SETLSB(p)
  $p \leftarrow$ SETTWOMSB(p)
  while ISNOTPRIME(p) **do** ▷ Miller-Rabin, 20 Runden
    $p \leftarrow p + 2$
  end while

  $r \leftarrow l - 64$ ▷ Freie Bits von n
  $n_l \leftarrow s \cdot 2^r$ ▷ Minimaler Wert für n
  $n_h \leftarrow n_l + (2^r - 1)$ ▷ Maximaler Wert für n
  $q_l \leftarrow \lfloor \frac{n_l}{p} \rfloor + 1$ ▷ Minimaler Wert für q
  $q_h \leftarrow \lfloor \frac{n_h}{p} \rfloor$ ▷ Maximaler Wert für q

  $q \leftarrow$ GLASSKEY_PRNG_INT_MIN_MAX(q_l, q_h)
  $q \leftarrow$ SETLSB(q)
  while ISNOTPRIME(q) **do**
    $q \leftarrow q + 2$
  end while
  return (p, q)
end procedure

Der Seed wird Ihnen als Base64-kodierte Binärdaten übergeben, welche Sie *big endian* interpretieren müssen.
Erzeugen Sie aus den so erhaltenen Faktoren p, q einen RSA Private Key in DER-Format. Die entsprechende ASN.1-Syntax können Sie RFC8017 im Appendix A.1.2 entnehmen. Prüfen Sie, dass Ihre so erzeugten RSA Private Keys direkt mit OpenSSL kompatibel sind.

```
{
    "action": "glasskey_genkey",
    "arguments": {
        "agency_key": "T01HV1RG",
        "seed": "3q2+78wA/+4=",
        "bit_length": 1024
    }
}
```


Erwartete Antwort:


```
{
    "der": "MIICXAIBAAKBgQDerb7vzAD/7icmAUKB1ufxU7rSB1KCvFJbYI4
       MfgMGIS7y1Ju/EOJ0snGit9FSsiskHIU0E8Co28mvpV69qwo5fdiOaL/c1oL
       p/ANVk+qmaOTKL/xRIUkXfh6X1LwfljejHxKw8N977rZ/ChT4srM4nNFrXdV
       6fAiE5NrZ+lDjfQIDAQABAoGBAIVLP+ZPGbIZnvmS+3q5Z/H/iu86TtvPWwU
       z0RQNQkYjCvV06x3+P3BXeMpMelKkMErXf6zivgrWNe6ccQ/cGKuZmmFVWAI
       uIsM5GTiEz0Ov7unkoqBoP0+9dUEiL6GvPataViHmiNscLf2JotD00hFqDZ2
       6RINmBeWYazwZr0fBAkEAwodaEDHYM+Liw11EBsEUNBDc5wUAMDylJRdN0Co
       JEWhQCl/07fJcdOAMezjzh/LPMbg9IKTAxD7dk+nEdbl0KQJBASULn0of17m
       aKMo7e0rWYRmL/VdK/nBkMLHtUDUxz1GP1uX3DlxBzTu7k6vlacgGftJTsNS
       YnZAaftRFWj1f/zUCQF+5r24suRG/Yot0x9bzCHgenDXq1g7mqPW5pAb9yHy
       ScmDIm4TEMQ8qebnhaqXJrH/xA9Oef2WS8gKplI3B9xECQDbgy07FUc/XN8Z
       Ph1JHfV2cYrAjQizoBlp7t6aOkmWSy0q7jnvmcrm58fih+MJVvRBETfwyLGe
       dHp0/85tEy/UCQHSvm3HtEvR7u7FQxT3shRZYr0BbSw5mZMJAnIvjvAg6c4R
       tVNOxOMimiovaL+R6BoiPAdDmGUkYWVNNAEC/9S0="
}
```

glasskey_break: Bruch von CMS-Verschlüsselung

Wenn CMS verwendet wird, um Nachrichten zu verschlüsseln und die RSA-Schlüssel innerhalb des Zertifikats mittels Glasskey ein Backdoor eingebaut haben, können solche Nachrichten von jeder Person entschlüsselt werden, die den `agency_key` besitzt. Zunächst eine kleine Einführung in CMS und dessen Verwendung mit OpenSSL. Eine RSA-Schlüsselerzeugung (Key ist hier offensichtlich nicht mit einem Backdoor versehen):

```
$ openssl genrsa -out goodkey.key 1024
```

Daraufhin erstellen wir uns ein selbstsigniertes Zertifikat:

```
$ openssl req -key goodkey.key -new -x509 -subj '/CN=Good Cert' \
    -out goodkey.crt
```

Und können nun Nachrichten an dieses Zertifikat verschlüsseln:

```
$ echo Hallo | openssl cms -encrypt -binary -outform der \
    -out encrypted_msg.bin -aes128-wrap -aes128 goodkey.crt
```

Die verschlüsselte Nachricht liegt nun in der Binärdatei `encrypted_msg.bin`. Um diese zu entschlüsseln benötigen wir den privaten Schlüssel:

```
$ openssl cms -decrypt -in encrypted_msg.bin -inform der \
    -inkey goodkey.key
Hallo
```

Ihnen wird nun eine so erstellte CMS-Nachricht gegeben sowie das öffentliche X.509 Zertifikat des respektiven Empfängers. Ihre Aufgabe ist es, mit Hilfe des `agency_key`, welcher zur Erstellung des verwendeten RSA-Keypaars genutzt wurde, den öffentlichen Schlüssel zu brechen und die Nachricht zu entschlüsseln.

```
{
    "action": "glasskey_break",
    "arguments": {
        "agency_key": "T01HV1RG",
        "cms_msg": "MIIBJwYJKoZIhvcNAQcDoIIBGDCCARQCAQAxgdAwgc0CAQ
           AwNjAeMRwwGgYDVQQDDBNCYWNrZG9vcmVkIEdsYXNza2V5AhRcmnGJVeWTER
           6dfauMTBuwca/wNzANBgkqhkiG9w0BAQEFAASBgIWOQpYga5Ixa0s74wtDZQ
           trtjQCEm/kxnPHhkHZf4Sl627pPe8dtzxh8B4qC7Fmu73UugMDS0lbbeWABt
           7Wu2fOnf2fXRXBFfYiJfmVM4bBBLW9gcPzjNsswTfw48dQzw+L1oi/+PZmCx
           UQ7NztkAhPWawj/iFRxRrAtmhDNLZVMDwGCSqGSIb3DQEHATAdBglghkgBZQ
           MEAQIEEGasAhJ6oXdmSsElQV3686qAECE9U0KMqoujqKLgHQapxZc=",
        "x509_crt": "MIICGDCCAYGgAwIBAgIUXJpxiVXlkxEenX2rjEwbsHGv8
           DcwDQYJKoZIhvcNAQELBQAwHjEcMBoGA1UEAwwTQmFja2Rvb3JlZCBHbGFzc
           2tleTAeFw0yNDEyMDIxODU3MTNaFw0yNTAxMDExODU3MTNaMB4xHDAaBgNVB
           AMME0JhY2tkb29yZWQgR2xhc3NrZXkwgZ8wDQYJKoZIhvcNAQEBBQADgY0AM
           IGJAoGBAN6tvu/MAP/uJyYBQoHW5/FTutIHUoK8Ultgjgx+AwYhLvLUm78Q4
           nSycaK30VKyKyQchTQTwKjbya+lXr2rCjl92I5ov9zWgun8A1WT6qZo5Mov/
           FEhSRd+HpfUvB+WN6MfErDw33vutn8KFPiyszic0Wtd1Xp8CITk2tn6UON9A
           gMBAAGjUzBRMB0GA1UdDgQWBBTL5u3Jeoj2RjQP57PdTR8Twt+kTzAfBgNVH
           SMEGDAWgBTL5u3Jeoj2RjQP57PdTR8Twt+kTzAPBgNVHRMBAf8EBTADAQH/M
           A0GCSqGSIb3DQEBCwUAA4GBAJxUCHVwawZotoi8/qhFTBNz0iD3DwkYN6SFD
           aQbgQXtLfi8JgVM44IYRTtNtC1uD9KYOhGzal3C7+Xk1qVukzHDM1xeZ0idx
           wbvnv/dyDSYGoqpcedfy8zGU/xG1QDcCeKjzePV7Y6eZufnFGVsN0zUu5cRF
           NnFyNKa+HKalQcW"
    }
}
```


Erwartete Antwort:


```
{
    "plaintext": "U2FwcGVybG90dAo="
}
```