Übungen zu Systemsicherheit

Jürgen Kleinöder,
Michael Gernoth, Reinhard Tartler
Universität Erlangen-Nürnberg, Informatik 4

SYSSEC

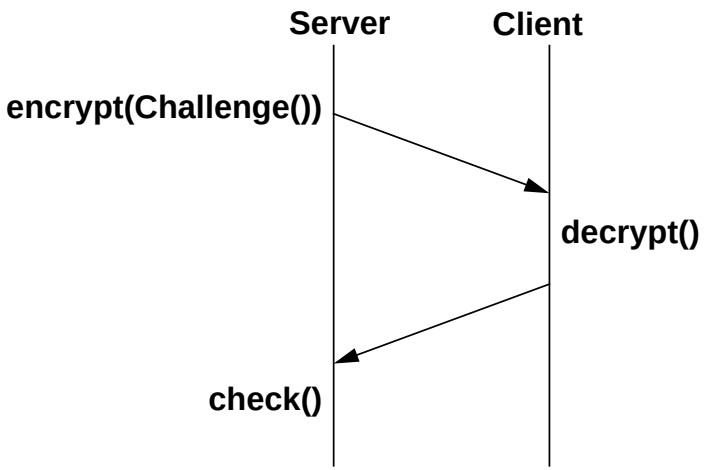
D.1 Besprechung der Aufgabe ’filecrypt’

- Wieviele Passwörter pro Sekunde? :-)

D.2 Challenge Response Verfahren

- Authentifizierung eines Clients an einem Serverdienst
- Zufallszahl (Nonce) wird verschlüsselt an den Client gesendet
  - symmetrische Verschlüsselung
  - asymmetrische Verschlüsselung
- Ergebnis wird zurückgesendet

D.3 Ablauf



D.4 Aufgabenstellung

- 2 OpenSSL gestützte Konsolenanwendungen:
  - Server
    - Kennt den Public Key
    - Verschlüsselt eine Zufallszahl (Nonce)
    - Erwartet die Antwort in einer sinnvollen Zeit
  - Client
    - Hat Private Key
    - Entschlüsselt
    - gibt die entschlüsselte Nonce aus
- Anwendungen arbeiten auf `stdin`/`stdout`, Bedienung mittels zwei `x-terms` und Copy&Paste

D.5 Schwächen/Probleme des RSA Verfahrens

- RSA ist ein deterministisches Verfahren
  - Keine Verwendung von Zufallswerten
  - "Known-Plaintext Attack" ist möglich
- Daher wird RSA mit einem Padding Scheme verwendet
  - Bekanntestes Verfahren ist PKCS
  - Standardisiert in RFC 3447 (definiert OAEP und PKCS1-V1_5)
- Maximallänge der zu verschlüsselnden Nachricht ist beschränkt durch
  - Schlüssellänge
  - Padding Verfahren

D.6 Erstellung von RSA Keys

- Mittels openssl Kommandozeile

```
openssl genrsa -out privkey -aes256 1024
```

- Die Datei `privkey` enthält nun beide Schlüsselteile
  - Serveranwendung benötigt aber nur den öffentlichen Teil
  - Extraktion ebenfalls mit dem `openssl` Werkzeug:

```
openssl rsa -in privkey -pubout -out pubkey.pem
```

- **NB**: Private Key ist ggf. mit einem Passwort verschlüsselt!
  - Das Token muss also den Key erst entschlüsseln
  - Dafür müssen die entsprechenden Module in OpenSSL geladen werden:

```
ERR_load_crypto_strings();
OpenSSL_add_all_algorithms();
```


D.7 Alleinstehendes Cryptotoken

- Kann z.B. von einem USB-Stick, etc. verwendet werden.
- Schlüsselhälften in die ausführbaren Programme einbinden
  - Generierung eines Headers, Einbinden per `#include`
  - Umwandlung des Schlüssels in ein Object-File und ins Binary binden

```
ld -r -b binary -o privkey.o privkey.pem
```

  - Schlüssel dann verfügbar als Symbol

```
extern char _binary_privkey_pem_start[];
```

D.8 Umgang mit RSA in openssl

- OpenSSL bietet diverse Funktionen zum Umgang von PEM Schlüsseln an:
- Dokumentiert in `pem(3ssl)`
  - ◆ Diverse Variationen von Formaten (Enkodierung und Verschlüsselung)
  - ◆ Für diese Aufgabe sinnvolle Funktionen:

```
RSA *PEM_read_RSAPrivateKey(FILE *fp, x, cb, u)
RSA *PEM_read_RSA_PUBKEY(FILE *fp, x, cb, u)
RSA *PEM_read_bio_RSAPrivateKey(BIO *bp, *x, cb , u);
...
```

- x, cb, u werden nur für benutzerspezifische Passphrase-Callback Funktion benötigt. `NULL` ist ein sinnvoller und akzeptierter Wert.

D.9 RSA Verschlüsselung in Openssl

- Verschlüsselung an einen Key

```
int RSA_public_encrypt(int flen, unsigned char *from,
           unsigned char *to, RSA *rsa, int padding);
```

- Entschlüsselung einer Nachricht:

```
int RSA_private_decrypt(int flen, unsigned char *from,
            unsigned char *to, RSA *rsa, int padding);
```

- Als Padding Modus wird `RSA_PKCS1_OAEP_PADDING` verwendet
- Die Challenge ist genauso lang wie der Schlüssell

```
int RSA_size(const RSA *rsa);
```

- Minus Paddinglänge (2x SHA1 + 2Bytes)

```
#define RSA_PKCS1_OAEP_PADDING_SIZE 42
```