

Aufgabenblatt Hybride Verschlüsselung

Lösen Sie die nachfolgenden Aufgaben und bereiten Sie diese bis zum nächsten Lehrveranstaltungstermin vor.

LB-HVS 04.

Ein Server (Sender) möchte eine verschlüsselte Nachricht an einen Client (Empfänger) verschicken. Dabei soll aus Performancegründen ein symmetrisches Verschlüsselungsverfahren, konkret AES, verwendet werden. Um den Schlüssel zwischen den beiden Kommunikationsteilnehmern vorab auszutauschen, wird Diffie-Hellman als Schlüsseltauschverfahren verwendet. Von der Schlüsselgenerierung bis zur symmetrischen Verschlüsselung sollen alle Schritte umgesetzt werden. Schreiben Sie dazu ein Programm unter Verwendung der *GMP* und der *libso-dium* (vgl. Beispiel 01.), das die folgenden Operationen implementiert und sich nach dem Muster `<Operation> [<Operand 1> <Operand 2>]` aufrufen lässt:

- `ServerGeneratePartialKey` zum Erzeugen einer Zahl $x \in \mathbb{Z}_p^*$ und zum anschließenden Berechnen des Teilschlüssels $S := g^x \bmod (p)$. Die Ausgabe von x und S erfolgt in dezimaler Darstellung über `std::cout` in **exakt** folgendem Format (Beispielausgabe):

```
x: 1559258775283944 [...]
g^x: 1311271927357378 [...]
```

- `ClientGeneratePartialKey` zum Erzeugen einer Zahl $y \in \mathbb{Z}_p^*$ und zum anschließenden Berechnen des Teilschlüssels $C := g^y \bmod (p)$. Die Ausgabe von y und C erfolgt analog zu der des Servers (Beispielausgabe):

```
y: 1175596360350942 [...]
g^y: 2826535490322092 [...]
```

Beachten Sie, dass S und C paarweise verschieden sein **müssen**, d.h. $g^x \not\equiv g^y \bmod (p)$.

- `ServerGenerateSessionKey` zum Berechnen des 256 Bit langen Sitzungsschlüssels $k := (C^x \equiv (g^y)^x \bmod (p)) \bmod (2^{256})$ für den Sender basierend auf C und x , die in dieser Reihenfolge als Kommandozeilenparameter angegeben sind. Die Ausgabe des Schlüssels erfolgt in hexadezimaler Darstellung über `std::cout` in **exakt** folgendem Format (Beispielausgabe):

```
d319bbb924009b66 [...]
```

- `ClientGenerateSessionKey` zum Berechnen des 256 Bit langen Sitzungsschlüssels $k := (S^y \equiv (g^x)^y \bmod (p)) \bmod (2^{256})$ für den Client basierend auf S und y , die in dieser Reihenfolge als Kommandozeilenparameter angegeben sind. Die Ausgabe des Schlüssels erfolgt analog zu der des Servers (Beispielausgabe):

d319bbb924009b66 [...]

Beachten Sie, der Sitzungsschlüssel k bei Server und Client identisch sein **muss**.

- `ServerEncrypt` zum Verschlüsseln einer Nachricht mit dem (zuvor separat generierten) Sitzungsschlüssel. Die Nachricht als Text sowie der Sitzungsschlüssel in Hexadezimaldarstellung sind als Kommandozeilenparameter in exakt dieser Reihenfolge zu übergeben. Die verschlüsselte Nachricht wird in Hexadezimaldarstellung auf `std::cout` ausgegeben.
- `ClientDecrypt` zum Entschlüsseln einer Nachricht mit dem (zuvor separat generierten) Sitzungsschlüssel. Die verschlüsselte Nachricht als Text sowie der Sitzungsschlüssel in Hexadezimaldarstellung sind als Kommandozeilenparameter in exakt dieser Reihenfolge zu übergeben. Die entschlüsselte Nachricht wird als Text auf `std::cout` ausgegeben.

Beispielaufrufe:

- `ServerGeneratePartialKey`
- `ClientGeneratePartialKey`
- `ServerGenerateSessionKey 2826535490322092[...] 1559258775283944[...]`
- `ClientGenerateSessionKey 1311271927357378[...] 1175596360350942[...]`
- `ServerEncrypt Hallo d319bbb924009b66[...]`
- `ClientDecrypt e7e25195d4bcff8c[...] d319bbb924009b66[...]`

Hinweise: Um voneinander verschiedene Zufallszahlen mittels der GMP zu erzeugen, rufen Sie nach `gmp_randinit_default(prng_state)`; die Funktion `gmp_randseed_ui(prng_state, time(NULL))`; auf, die den Pseudozufallszahlengenerator mit der aktuellen Systemzeit initialisiert. Verwenden Sie für g und p die folgenden Werte (übernommen von <https://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html>):

```

1  const char * const g_as_text = "\
2  f7e1a085d69b3ddecbbcab5c36b857b97994afbbfa3aea82f\
3  9574c0b3d0782675159578ebad4594fe67107108180b44916\
4  7123e84c281613b7cf09328cc8a6e13c167a8b547c8d28e0a\
5  3ae1e2bb3a675916ea37f0bfa213562f1fb627a01243bcc4\
6  f1bea8519089a883dfe15ae59f06928b665e807b552564014\
7  c3bfecf492a";
8  const mpz_class g(g_as_text, 16);
9
10 const char * const p_as_text = "\
11 fd7f53811d75122952df4a9c2eece4e7f611b7523cef4400c\
12 31e3f80b6512669455d402251fb593d8d58fabfc5f5ba30f6\
13 cb9b556cd7813b801d346ff26660b76b9950a5a49f9fe8047\

```

```

14  b1022c24fbba9d7feb7c61bf83b57e7c6a8a6150f04fb83f6\
15  d3c51ec3023554135a169132f675f3ae2b61d72aef222031\
16  99dd14801c7";
17  const mpz_class p(p_as_text, 16);

```

Um das 1024 Bit lange Ergebnis $(g^x)^y \equiv (g^y)^x \pmod{p}$ auf 256 Bit, d.h. k , zu reduzieren, verwenden Sie folgende Funktion oder eine Variation davon:

```

1  string ExtractKey(const mpz_class &g_xy)
2  {
3      mpz_class two_256;
4      mpz_ui_pow_ui(two_256.get_mpz_t(), 2, 256);
5      mpz_class g_xy_short;
6      mpz_mod(g_xy_short.get_mpz_t(), g_xy.get_mpz_t(), two_256.
7          ↪ get_mpz_t());
8      stringstream s;
9      s << hex << g_xy_short << endl;
10     return s.str();

```

Verwenden Sie vor der Ver- und Entschlüsselung die Funktion `HexStringToArray` aus Beispiel 03. Um Nachrichten zu ver- bzw. entschlüsseln, verwenden Sie AES-256 im GCM-Modus aus der libsodium. Die Dokumentation unter https://download.libsodium.org/doc/secret-key_cryptography/aes-256-gcm.html beschreibt authentifizierte Verschlüsselung, die zusätzlich eine Verifikation bei der Entschlüsselung durchführt, um sicherzustellen, dass der korrekte Schlüssel verwendet wurde. Für die Ver- und Entschlüsselung verwenden Sie dabei eine Null-Nonce wie folgt:

```

1  const unsigned char nonce[ crypto_aead_aes256gcm_NPUBBYTES ] =
    ↪ {0};

```

Achtung: Bitte verwenden Sie in sicherheitskritischen Anwendungen **niemals** eine Konstante als Nonce. Hier wird sie lediglich verwendet, um die Implementierung im Rahmen des Laboratoriums stark zu vereinfachen.

LB-HVS 05. (nicht abzugeben)

Schließen Sie sich zu den von den Lehrveranstaltungsleitern bestimmten Zweiergruppen zusammen, wobei je eine Person die Rolle des Servers und die andere entsprechend die des Clients übernimmt. Überprüfen Sie die Korrektheit und Interoperabilität Ihrer Programme aus Beispiel 04. in zwei separaten Schritten. Tauschen Sie zuerst per Email alle für den Schlüsseltausch notwendigen Daten aus. Beachten Sie, dass Sie dabei keine Daten austauschen, die im Protokoll nach Diffie-Hellman geheim gehalten werden müssen. Senden Sie danach mit dem vereinbarten Sitzungsschlüssel eine Nachricht vom Server an den Client. Stellen Sie sicher, dass diese vom Client korrekt entschlüsselt werden kann.