Aufgabenblatt Signaturen

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

LB-HVS 00. (nicht abzugeben)

a) Legen Sie ein neues Projekt nach dem Musterprojekt für *libsodium* an und führen Sie es mit es Ihrem Vornamen als Argument aus. Die Ausgabe des Programmes ist der SHA-256-Hash des übergebenen Argumentes. Verifizieren Sie die Ausgabe mit dem Befehl `echo -n <Vorname> | sha256sum` (ohne spitze Klammern!) auf der Kommandozeile.

b) Passen Sie den Code aus a) so an, dass SHA-512 an Stelle von SHA-256 verwendet wird. Gehen Sie dazu den Code mit Hilfe der Dokumentation von *libsodium* schrittweise durch und passen Sie das Programm entsprechend an. Verwenden Sie zur Verifikation auf der Kommandozeile `sha512sum` an Stelle von `sha256sum` – die restliche Verwendung bleibt unverändert.

Hinweis: Die Dokumentation von libsodium *finden Sie unter* `https://download.libsodium.org/doc/advanced/sha-2_hash_function.html`

LB-HVS 01.

Schreiben Sie ein Programm unter Verwendung des Musterprojektes für *GMP* und *libsodium*, das die folgenden Funktionen implementiert:

- `Sign` zum Signieren einer Nachricht `message` mit dem privaten Schlüssel `(d, N)` aus einem RSA-Schlüsselpaar. Die Ausgabe der Signatur erfolgt in dezimaler Darstellung über `std::cout` in **exakt** folgendem Format (Beispielausgabe) **ohne** weitere Ausgaben:

  ```
  4272599298832472[...]
  ```

- `Verify` zum Überprüfen der Gültigkeit einer zur Nachricht `message` gehörenden Signatur `signature` mit dem öffentlichen Schlüssel `(e, N)` aus einem RSA-Schlüsselpaar. Die Ausgabe der Gültigkeit erfolgt in Form der Textausgaben *Signature valid.* bzw. *Signature invalid.* auf `std::cout`.

Das Programm soll mit vier bzw. fünf Parametern über die Kommandozeile wie folgt aufgerufen werden können: `./test Sign <message> <d> <N>` bzw. `./test Verify <message> <signature> <e> <N>`. Kombinieren Sie für die Erstellung dieses Programmes Ihren Code aus Beispiel 00. b) zur Berechnung des Hashes sowie aus LB-KÖS 01. für die Ver- bzw. Entschlüsselung mit RSA. Testen Sie Ihr Programm mit einem RSA-Schlüsselpaar, z.B. erzeugt mit Ihrem Programm aus LB-KÖS 02. mit einer Schlüssellänge von 2048 Bit.
Hinweis: Um den von der libsodium *ermittelten Hash in eine für die Berechnungen mit der* GMP *kompatible Darstellung umzuwandeln, muss das von der*

Hashfunktion zurückgegebene Array byteweise als Zahl dargestellt und dann in seiner Gesamtheit als (große) Zahl interpretiert werden. Sie können folgende Funktion oder eine Variation davon verwenden, um dies zu bewerkstelligen:

```
1  #include <sstream>
2  #include <iomanip>
3
4  void libsodium_to_GMP(const unsigned char (&libsodium_value)
       ↪ [crypto_hash_sha512_BYTES], mpz_class &GMP_value)
5  {
6    stringstream s;
7    s << hex;
8    for (size_t i = 0; i < sizeof libsodium_value; i++)
9      s << setw(2) << setfill('0') << (int)libsodium_value[i];
10   const string string_as_hex = s.str();
11   mpz_set_str(GMP_value.get_mpz_t(), string_as_hex.c_str(),
       ↪ 16);
12 }
```

Beispielaufruf:

```
1  unsigned char hash[crypto_hash_sha512_BYTES];
2  /* TODO: Calculate hash as usual */
3  mpz_class hash_value;
4  libsodium_to_GMP(hash, hash_value);
```

LB-HVS 02. (nicht abzugeben)

Schließen Sie sich zu den von den Lehrveranstaltungsleitern bestimmten Zweiergruppen zusammen und überprüfen Sie die Korrektheit und Interoperabilität Ihrer Programme aus Beispiel 01. durch einen Emailaustausch signierter Nachrichten. Person P_A generiert dazu ein RSA-Schlüsselpaar (d_A, e_A, N_A) sowie eine beliebige Nachricht m_a, aus denen in Kombination mittels der Implementierung (von P_A) aus 01. die Signatur s_a erzeugt wird. Anschließend sendet P_A per Email m_a, s_a und den öffentlichen Schlüssel (e_A, N_A) an Person P_B. Diese übergibt die übersendeten Informationen an die Implementierung (von P_B), um die Signatur zu überprüfen. Vertauschen Sie anschließend die Rollen von P_A und P_B und wiederholen Sie den Versuch.

LB-HVS 03.

Schreiben Sie ein Programm unter Verwendung des Musterprojektes für *libsodium*, das analog zu Beispiel 01. Nachrichten signieren und verifizieren kann. Verwenden Sie statt Ihrer Eigenimplementierungen ausschließlich die Signierungsfunktionalität der *libsodium*. Orientieren Sie sich dazu an der Dokumentation (*combined mode*) unter `https://download.libsodium.org/doc/public-key_cryptography/public-key_signatures.html`.

Die Schlüsselgenerierung soll direkt beim Signieren erfolgen, wobei der öffentliche Schlüssel zusammen mit der Signatur in **exakt** folgendem Format ausgegeben werden soll:

```
Signed message: d584bb849f3a8d96[...]
Public key: 8ab03757373db7a0[...]
```

Beachten Sie sowohl beim Signieren als auch beim Überprüfen der Signatur, dass die *libsodium* elliptische Kurven anstatt RSA verwendet, wodurch die Schlüssellängen kürzer sind und die Schlüssel aus nur einem einzigen Wert bestehen.
Hinweise: Verwenden Sie zur Ausgabe der Signatur und des öffentlichen Schlüssels die Schleife zur hexadezimalen Ausgabe aus Beispiel 00. Zum Einlesen der hexadezimalen Ziffernkolonnen in einem für die libsodium *kompatiblen Format können Sie folgende Funktion oder eine Variation davon verwenden:*

```
1  #include <sstream>
2  #include <iomanip>
3
4  void HexStringToArray(const char * const text, unsigned char
       ↪  * const array, const size_t array_size)
5  {
6    for (size_t i = 0; i < array_size; i++)
7    {
8      const string text_part(text + 2 * i, 2); //Process 2
           ↪ chars at a time
9      stringstream s(text_part);
10     s >> hex;
11     int value;
12     s >> value;
13     array[i] = value;
14   }
15 }
```