Aufgabenblatt Signaturen

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

LB-S 00. (nicht abzugeben)

(*Code::Blocks*-Template *libsodium project*)

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-S 01. (*Code::Blocks*-Template *GMP and libsodium project*)

Schreiben Sie ein Programm, das unter Verwendung der *GMP* und der *libsodium* 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 Anzeige 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 2.048 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  template <size_t N> //Number of bytes (array size)
2  void libsodium_to_GMP(const unsigned char (&libsodium_value)
       ↪ [N], mpz_class &GMP_value)
3  {
4    GMP_value = 0;
5    for (const auto &libsodium_byte : libsodium_value)
6    {
7      GMP_value *= 256;
8      GMP_value += libsodium_byte;
9    }
10 }
```

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-S 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-S 03. (*Code::Blocks*-Template *libsodium project*)

Schreiben Sie ein Programm, das analog zu Beispiel 01. Nachrichten signieren und verifizieren kann. Verwenden Sie statt Ihrer Eigenimplementierungen ausschließlich die Signierungsfunktionalität der *libsodium*. Verwenden Sie den so genannten *Combined Mode*, in dem die zu signierende Nachricht gemeinsam mit der Signatur ausgegeben wird und orientieren Sie sich zur Umsetzung an der Dokumentation (`https://download.libsodium.org/doc/public-key_cryptography/public-key_signatures.html`). Beachten Sie, dass in diesem Modus die Signatur und der Klartext der Nachricht kombiniert in der Ausgabe

enthalten sind und damit für die Verifikation nur ein Kommandozeilenparameter für beide benötigt wird, d.h., dass die Verifikation insgesamt nur drei Kommandozeilenparameter erwartet.

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 <string>
2
3  bool HexStringToArray(const std::string &hex_string,
       ↪ unsigned char array[], const size_t array_size)
4  {
5    if (hex_string.length() != 2 * array_size)
6      return false;
7    for (size_t i = 0; i < array_size; i++)
8    {
9      const std::string str_part(hex_string.c_str() + 2 * i,
         ↪ 2); //Process 2 characters (one byte) at a time
10     try
11     {
12       const auto byte = std::stoul(str_part, nullptr, 16);
13       array[i] = byte;
14     }
15     catch (...)
16     {
17       return false;
18     }
19   }
20   return true;
21 }
```