Kryptoanalyse Laborarbeit

In Ihrer Laborarbeit erstellen Sie ein Programm namens `kauma`, welches als Parameter einen Dateinamen bekommt. Die Datei, die Ihnen immer übergeben wird, ist im JSON-Format und beinhaltet Testcases, die Sie ausführen sollen. Jeder einzelne Testcase ist ein Dictionary innerhalb der JSON-Struktur.
Allgemein sieht die JSON Testcase-Datei wie folgt aus:

```
{
        "testcases": {
                "b856d760-023d-4b00-bad2-15d2b6da22fe": {
                        "action": "add_numbers",
                        "arguments": {
                                "number1": 123,
                                "number2": 234
                        }
                },
                "254eaee7-05fd-4e0d-8292-9b658a852245": {
                        "action": "add_numbers",
                        "arguments": {
                                "number1": 333,
                                "number2": 444
                        }
                },
                "affbf4fc-4d2a-41e3-afe0-a79e1d174781": {
                        "action": "subtract_numbers",
                        "arguments": {
                                "number1": 999,
                                "number2": 121212
                        }
                }
        }
}
```

In diesem Beispiel gäbe es zwei Aufgabenstellungen, bei der Sie einerseits die Summe zweier Zahlen (`add_numbers`) zurückliefern sollen sowie andererseits die Differenz zweier Zahlen (action `subtract_numbers`) ausrechnen. Jeder der Testfälle hat eine eindeutige ID (hier eine UUID), welcher als Schlüssel des `"testcases"` Dictionaries verwendet wird. Ihre Antwort für den respektiven Testcase muss dieseble ID beinhalten. Eine Datei kann beliebig viele Testcases beinhalten. In diesem Fall würde Ihre Antwort, die Sie als JSON-Datei auf der Standardausgabe ausgeben sollen, wie folgt lauten:

```
{
        "responses": {
                "b856d760-023d-4b00-bad2-15d2b6da22fe": {
                        "sum": 357
                },
                "254eaee7-05fd-4e0d-8292-9b658a852245": {
                        "sum": 777
                },
```

```
                "affbf4fc-4d2a-41e3-afe0-a79e1d174781": {
                        "difference": -120213
                }
        }
}
```

Die genaue Struktur für die jeweiligen Actions ist den einzelnen Aufgabenstellungen zu entnehmen. Ihr Programm soll kumulativ mit allen actions benatworten können; das heißt, Sie sollten Ihr Programm von Anfang an so modular gestalten, dass Sie einfach neue Actions hinzufügen können.
Binärdaten werden in allen Aufgaben ausschließlich als Base64-encodete Strings übermittelt.

`poly2block`: 16-byte Block in ein Polynom umwandeln

Gegeben ist ein Polynom mit seinen Koeffizienten (nicht notwendigerweise sortiert), welches Sie in einen 16 Bytes langen Binärblock umwandeln sollen. Die Semantik hierbei ist für den Augenblick ausschließlich *ex" (später sollen Sie dies aber um die GCM-Semantik erweitern können). Beispiel-Testcase:

```
{
        "action": "poly2block",
        "arguments": {
                "semantic": "xex",
                "coefficients": [
                        12,
                        127,
                        9,
                        0
                ]
        }
}
```

Erwartete Antwort:

```
{
        "block": "ARIAAAAAAAAAAAAAAAAgA=="
}
```

`block2poly`: Block in ein Polynom umwandeln

In diesem Aufgabentyp wird Ihnen ein 16-Byte großer Binärblock gegeben und Sie ermitteln die Koeffizienten des respektiven Polynoms. Es ist also exakt die Inverse Operation zur letzten Aufgabe. Bei Ihrer Antwort müssen die Koeffizienten sortiert sein. Auch hier wird aktuell nur die XEX-Semantik unterstützt, aber später wird GCM dazukommen.

```
{
        "action": "block2poly",
        "arguments": {
                "semantic": "xex",
                "block": "ARIAAAAAAAAAAAAAAAAgA=="
        }
}
```

Erwartete Antwort:

```
{
        "coefficients": [
                0
                9,
                12,
```

```
                127
        ]
}
```

`gfmul`: Multiplikation im $\mathbb{F}_{2^{128}}$

Implementieren Sie die Galoisfeldmultiplikation zweier 16-Bytes Blöcke im $\mathbb{F}_{2^{128}}$ mit dem in GCM/XEX verwendeten üblichen Reduktionspolynom $\alpha^{128} + \alpha^7 + \alpha^2 + \alpha + 1$. Die Semantik der Blockinterpretation wird Ihnen gegeben (aktuell immer xex). Aktuell wird der zweite Block b immer der Konstante α Block sein, jedoch wird es für die folgenden Aufgaben notwendig sein, eine generische Multiplikation (zwei beliebige Blöcke) miteinander multiplizieren zu können.

```
{
        "action": "gfmul",
        "arguments": {
                "semantic": "xex",
                "a": "ARIAAAAAAAAAAAAAAAAAgA==",
                "b": "AgAAAAAAAAAAAAAAAAAAAA=="
        }
}
```

Erwartete Antwort:

```
{
        "product": "hSQAAAAAAAAAAAAAAAAA=="
}
```

`sea128`: Modifizierter AES-Algorithmus

Implementieren Sie SEA-128, welches eine abgeänderte AES-128 Variante ist. Ihre Eingabe muss jeweils nur einen 16-Bytes Block verarbeiten können. Die Definition der SEA-128 Verschlüsselung S basiert auf der AES-128 Verschlüsselung Ëünd lautet wie folgt:

$$S_K(P) = E_K(P) \oplus \texttt{c0ffeec0ffeec0ffeec0ffeec0ffee11}$$

Beispiel für Verschlüsselung:

```
{
        "action": "sea128",
        "arguments": {
                "mode": "encrypt",
                "key": "istDASeincoolerKEYrofg==",
                "input": "yv66vvrO263eyviIiDNEVQ=="
        }
}
```

Erwartete Antwort bei Verschlüsselung:

```
{
        "output": "D5FDo3iVBoBN9gVi9/MSKQ=="
}
```

Beispiel für Entschlüsselung:

```
{
        "action": "sea128",
        "arguments": {
                "mode": "decrypt",
                "key": "istDASeincoolerKEYrofg==",
                "input": "D5FDo3iVBoBN9gVi9/MSKQ=="
        }
}
```

Erwartete Antwort bei Entschlüsselung:

```
{
        "output": "yv66vvrO263eyviIiDNEVQ=="
}
```

`fde`: FDE mittels XEX

Implementieren Sie XEX auf Basis von SEA-128. Die Länge der jeweiligen Eingabedaten ist garantiert ein ganzzahliges Vielfaches der Blockgröße (128 Bit).

```
{
        "action": "xex",
        "arguments": {
                "mode": "encrypt",
                "key": "B1ygNO/CyRYIUYhTSgoUysX5Y/wWLi4UiWaVeloUWs0=",
                "tweak": "6VXORr+YYHrd2nVe0OlA+Q==",
                "input": "/aOg4jMocLkBLkDLgkHYtFKc2L9jjyd2WXSSyxXQikpMY9Z
                          RnsJE76e9dW9olZIW"
        }
}
```

Erwartete Antwort:

```
{
        "output": "mHAVhRCKPAPx0BcufG5BZ4+/CbneMV/gRvqK5rtLe0OJgpDU5i
                   T7z2P0R7gEeRDO"
}
```

Analog zu SEA-128 soll ebenfalls zum Modus `encrypt` der Modus `decrypt` von Ihnen implementiert werden:

```
{
        "action": "xex",
        "arguments": {
                "mode": "decrypt",
```

```
                "key": "B1ygNO/CyRYIUYhTSgoUysX5Y/wWLi4UiWaVeloUWs0=",
                "tweak": "6VXORr+YYHrd2nVe0OlA+Q==",
                "input": "lr/ItaYGFXCtHhdPndE65yg7u/GIdM9wscABiiFOUH2Sbyc
                          2UFMlIRSMnZrYCW1a"
        }
}
```

Erwartete Antwort:

```
{
        "output": "SGV5IHdpZSBrcmFzcyBkYXMgZnVua3Rpb25pZXJ0IGphIG9mZm
                   VuYmFyIGVjaHQu"
}
```