Kryptoanalyse Laborarbeit

Aufgabe 3

2024-11-05

In der folgenden Aufgabe sollen Sie die grundlegenden Funktionen implementieren, die dann für das nächste Aufgabenblatt notwendig werden. Grundsätzlich betrachten wir hier Polynome des Grades n mit folgender Form:

$$c_0X^0 + c_1X^1 + c_2X^2 + \ldots + c_nX^n \qquad c_i \in \mathbb{F}_{2^{128}} \forall i \in \{0 \ldots n\}$$

Die Koeffizienten der Polynome sind hier also Feldelemente im $F_{2^{128}}$. Als Repräsentation der Koeffizienten innerhalb der Testfälle wird grundsätzlich die GCM-Konvention verwendet. Innerhalb der Testcases wird ein solches Polynom als Array von Strings repräsentiert, wobei das Arrayelement mit Index i jeweils dem Koeffizient c_i entspricht. Als Beispiel sei folgendes Polynom gegeben:

$$(\alpha^{10} + \alpha^6 + \alpha^3) + (\alpha^8 + \alpha^4 + 1)X + (\alpha^{61} + \alpha^{36} + \alpha^2)X^2$$

Welches in kompakter Schreibweise so aussähe:

$$(\{448\}) + (\{111\})X + (\{2000001000000004\})X^2$$

Und innerhalb eines Testcases dann eben so repräsentiert würde:

```
[
    "EiAAAAAAAAAAAAAAAAAAAA==",
    "iIAAAAAAAAAAAAAAAAAAAA==",
    "IAAAAAgAAAQAAAAAAAAAAA=="
]
```

Innerhalb der Testcases wird ein Polynom des Grades n, wobei $n \geq 0$, stets mit exakt $n + 1$ Array-Elementen dargestellt.

`gfpoly_add`: Polynomaddition

Implementieren Sie die Addition zweier Polynome $S = A + B$:

```
{
    "action": "gfpoly_add",
    "arguments": {
        "A": [
            "NeverGonnaGiveYouUpAAA==",
            "NeverGonnaLetYouDownAA==",
            "NeverGonnaRunAroundAAA==",
            "AndDesertYouAAAAAAAAAA=="
        ],
        "B": [
            "KryptoanalyseAAAAAAAAA==",
            "DHBWMannheimAAAAAAAAAA=="
        ]
    }
}
```

Erwartete Antwort:

```
{
    "S": [
        "H1d3GuyA9/0OxeYouUpAAA==",
        "OZuIncPAGEp4tYouDownAA==",
        "NeverGonnaRunAroundAAA==",
        "AndDesertYouAAAAAAAAAA=="
    ]
}
```

`gfpoly_mul`: Polynommultiplikation

Implementieren Sie die Multiplikation zweier Polynome, sodass deren Produkt $P = A \cdot B$:

```
{
    "action": "gfpoly_mul",
    "arguments": {
        "A": [
            "JAAAAAAAAAAAAAAAAAAAA==",
            "wAAAAAAAAAAAAAAAAAAAA==",
            "ACAAAAAAAAAAAAAAAAAAA=="
        ],
        "B": [
            "OAAAAAAAAAAAAAAAAAAAA==",
            "IQAAAAAAAAAAAAAAAAAAA=="
        ]
    }
}
```

Erwartete Antwort:


```
{
    "P": [
        "MoAAAAAAAAAAAAAAAAAAAA==",
        "sUgAAAAAAAAAAAAAAAAAAA==",
        "MbQAAAAAAAAAAAAAAAAAAA==",
        "AAhAAAAAAAAAAAAAAAAAAA=="
    ]
}
```


`gfpoly_pow`: Polynomexponentiation

Implementieren Sie die Exponentiation eines Polynoms, sodass $Z = A^k$ für eine Ganzzahl $0 \leq k \leq 100$:


```
{
    "action": "gfpoly_pow",
    "arguments": {
        "A": [
            "JAAAAAAAAAAAAAAAAAAAAA==",
            "wAAAAAAAAAAAAAAAAAAAAA==",
            "ACAAAAAAAAAAAAAAAAAAAA=="
        ],
        "k": 3
    }
}
```


Erwartete Antwort:


```
{
    "Z": [
        "AkkAAAAAAAAAAAAAAAAAAA==",
        "DDAAAAAAAAAAAAAAAAAAAA==",
        "LQIIAAAAAAAAAAAAAAAAAA==",
        "8AAAAAAAAAAAAAAAAAAAAA==",
        "ACgCQAAAAAAAAAAAAAAAAA==",
        "AAAMAAAAAAAAAAAAAAAAAA==",
        "AAAAAgAAAAAAAAAAAAAAAA=="
    ]
}
```

`gfdiv`: Division von Feldelementen im $\mathbb{F}_{2^{128}}$

Implementieren Sie die Division von Feldementen $q = \frac{a}{b}$ im $\mathbb{F}_{2^{128}}$. Das Ergebnis der Division ist selbstverständlich nur für $b \neq 0$ definiert, Sie können also davon ausgehen, dass b nie Null wird. Die Semantik bei `gfdiv` ist, wie oben angegeben, implizit immer GCM.

```
{
    "action": "gfdiv",
    "arguments": {
        "a": "JAAAAAAAAAAAAAAAAAAAAA==",
        "b": "wAAAAAAAAAAAAAAAAAAAAA=="
    }
}
```

Erwartete Antwort:

```
{
    "q": "OAAAAAAAAAAAAAAAAAAAAA=="
}
```

`gfpoly_divmod`: Polynomdivision

Implementieren Sie die Polynomdivision zweier Polynome $Q = \frac{A}{B}$, wobei der Rest nach Division R sein soll. Es gilt also nach der Division, dass $A = Q \cdot B + R$.

```
{
    "action": "gfpoly_divmod",
    "arguments": {
        "A": [
            "JAAAAAAAAAAAAAAAAAAAAA==",
            "wAAAAAAAAAAAAAAAAAAAAA==",
            "ACAAAAAAAAAAAAAAAAAAAA=="
        ],
        "B": [
            "OAAAAAAAAAAAAAAAAAAAAA==",
            "IQAAAAAAAAAAAAAAAAAAAA=="
        ]
    }
}
```

Erwartete Antwort:

```
{
    "Q": [
        "nAIAgCAIAgCAIAgCAIAgCg==",
        "m85zn0c5zn0c5zn0c5zn0Q=="
    ],
    "R": [
        "lQNA0DQNA0DQNA0DQNA0Dg=="
    ]
}
```

`gfpoly_powmod`: Modulare Polynomexponentiation

Implementieren Sie die *modulare* Exponentiation eines Polynoms, sodass $Z = A^k \mod M$ für eine Ganzzahl $k \geq 0$ und zwei Polynome A, M. k kann hier sehr groß werden (Größenordnung 2^{128}) und dies *muss* von Ihrer Implementierung unterstützt werden. Sie dürfen hier also nicht einfach zunächst `gfpoly_pow` und danach `gfpoly_divmod` aufrufen – wenngleich das für kleine Zahlenwerte natürlich funktioniert und also für Testzwecke eingesetzt werden kann. Stattdessen verwenden Sie den altbekannten square-multiply Trick, den Sie bereits von der modularen Exponentiation kennen. Beispielaufgabe:

```
{
    "action": "gfpoly_powmod",
    "arguments": {
        "A": [
            "JAAAAAAAAAAAAAAAAAAAA==",
            "wAAAAAAAAAAAAAAAAAAAA==",
            "ACAAAAAAAAAAAAAAAAAAA=="
        ],
        "M": [
            "KryptoanalyseAAAAAAAAA==",
            "DHBWMannheimAAAAAAAAAA=="
        ],
        "k": 1000
    }
}
```

Erwartete Antwort:

```
{
    "Z": [
        "oNXl5P8xq2WpUTP92u25zg=="
    ]
}
```

Implementierungshinweise

Die Operationen in diesem Aufgabenblatt sind schwieriger korrekt zu implementieren als die vorherigen Übungsaufgaben. Verwenden Sie daher zur Erstellung geeigneter Testcases ein Computeralgebrasystem wie bspw. Sage, mit dem Sie polynomielle Arithmetik über Körpern leicht durchführen können:

```
_.<K> = GF(2)[]
F.<a> = GF(2^128, modulus = K^128 + K^7 + K^2 + K + 1)
_.<X> = PolynomialRing(F, "X")
```

Sie können sich hier jetzt also Feldelemente $a_0, a_1, a_2, b_0, b_1, b_2$ definieren, wobei in Sage statt α einfach a genommen wird.

```
a0 = a^5 + a^2
a1 = a + 1
a2 = a^10

b0 = a^3 + a + 1
b1 = a^7 + a^2
b2 = 0
```

Hier können Sie beispielsweise bereits die Invertierung von Feldelementen testen, die Sie für `gfdiv` benötigen:

```
sage: a0
a^5 + a^2

sage: a1
a + 1

sage: a0 / a1
a^4 + a^3 + a^2

sage: a1_inv = a1^-1

sage: a1_inv
a^127 + a^126 + a^125 + a^124 + a^123 + a^122 + a^121 + a^120 + a^119 + a^118 +
    a^117 + a^116 + a^115 + a^114 + a^113 + a^112 + a^111 + a^110 + a^109 + a^108
    [...]
    + a^20 + a^19 + a^18 + a^17 + a^16 + a^15 + a^14 + a^13 + a^12 + a^11 + a^10
    + a^9 + a^8 + a^7 + a

sage: a1 * a1_inv
1
```

Aus den Werten $a_0, a_1, a_2, b_0, b_1, b_2$ können Sie zwei Polynome A und B bilden:

```
A = (a0*X^0) + (a1*X^1) + (a2*X^2)
B = (b0*X^0) + (b1*X^1) + (b2*X^2)
```

Nun können Sie beispielsweise `gfpoly_add` und `gfpoly_mul` testen:

```
sage: A
a^10*X^2 + (a + 1)*X + a^5 + a^2

sage: B
(a^7 + a^2)*X + a^3 + a + 1

sage: A + B
a^10*X^2 + (a^7 + a^2 + a + 1)*X + a^5 + a^3 + a^2 + a + 1

sage: A * B
(a^17 + a^12)*X^3 + (a^13 + a^11 + a^10 + a^8 + a^7 + a^3 + a^2)*X^2 +
    (a^12 + a^9 + a^7 + a^3 + a^2 + 1)*X + a^8 + a^6 + a^3 + a^2
```

Um die Polynomdivision zu testen können Sie ebenfalls Sage verwenden:

```
sage: Q = A // B

sage: Q
(a^127 + a^124 + a^123 + a^122 + a^119 + a^118 + a^117 + a^114 + a^113 +
    a^112 + a^109 + a^108 + a^107 + a^104 + a^103 + a^102 + a^99 + a^98 +
    a^97 + a^94 + a^93 + a^92 + a^89 + a^88 + a^87 + a^84 + a^83 + a^82 +
    a^79 + a^78 + a^77 + a^74 + a^73 + a^72 + a^69 + a^68 + a^67 + a^64 +
    [...]
    + a^14 + a^13 + a^12 + a^9 + a^8 + a^7 + a^6 + a^4 + a^3 + 1)*X + a^126 +
    a^124 + a^114 + a^104 + a^94 + a^84 + a^74 + a^64 + a^54 + a^44 + a^34 +
    a^24 + a^14 + a^5 + a^4 + a^3 + 1

sage: R = A % B

sage: R
a^126 + a^125 + a^124 + a^117 + a^115 + a^114 + a^107 + a^105 + a^104 + a^97 +
    a^95 + a^94 + a^87 + a^85 + a^84 + a^77 + a^75 + a^74 + a^67 + a^65 + a^64 +
    a^57 + a^55 + a^54 + a^47 + a^45 + a^44 + a^37 + a^35 + a^34 + a^27 +
    a^25 + a^24 + a^17 + a^15 + a^14 + a^7 + a^5 + a^3 + 1

sage: A == (Q * B) + R
True
```

Als letzte Aufgabe unterstützt Sage selbstverständlich auch die Exponentiation und modulare Exponentiation:

```
sage: A^3
a^30*X^6 + (a^21 + a^20)*X^5 + (a^25 + a^22 + a^12 + a^10)*X^4 +
    (a^3 + a^2 + a + 1)*X^3 + (a^20 + a^14 + a^7 + a^5 + a^4 + a^2)*X^2 +
    (a^11 + a^10 + a^5 + a^4)*X + a^15 + a^12 + a^9 + a^6

sage: pow(A, 5000, B)
a^125 + a^124 + a^121 + a^118 + a^117 + a^113 + a^112 + a^108 + a^107 + a^104 +
    a^103 + a^102 + a^100 + a^97 + a^96 + a^95 + a^94 + a^93 + a^90 + a^89 +
    a^88 + a^84 + a^82 + a^81 + a^79 + a^75 + a^73 + a^70 + a^69 + a^68 + a^66
    + a^65 + a^63 + a^62 + a^59 + a^56 + a^54 + a^52 + a^50 + a^49 + a^47 +
    a^45 + a^43 + a^40 + a^39 + a^38 + a^37 + a^28 + a^26 + a^21 + a^20 + a^19
    + a^18 + a^16 + a^11 + a^9 + a^7 + a^6 + a^5 + a
```