Kryptoanalyse Laborarbeit

Aufgabe 4

2024-11-15

Hintergrund

In der Vorlesung hatten wir die Angriffe auf GCM bei Wiederverwendung der Nonce besprochen. Dies sollen Sie in dieser Übung implementieren, konkret den Algorithmus von Cantor-Zassenhaus zur polynomiellen Faktorisierung über finiten Feldern. Die Faktorisierung von Polynomen ist ein Werkzeug zum finden der Nullstellen von polynomiellen Gleichungssystemen. Betrachten Sie beispielsweise in $\mathbb{R}$ die einfache Gleichung:

$$x^4 + 13x^3 + 19x^2 - 213x - 540 = 0$$

Durch Faktorisierung des Polynoms auf der linken Seite erhalten wir:

$$(x + 3)(x + 9)(x - 4)(x + 5) = 0$$

Hieraus ergibt sich direkt, dass $-3, -9, 4, -5$ gültige Lösungen für das Gleichungssystem sind. In GCM kommt bekanntlich Arithmetik über dem $\mathbb{F}_{2^{128}}$ zum Einsatz. Der Ansatz ist jedoch exakt derselbe.
Dieses finden einer Lösung bei einem Polynom mit Koeffizienten im $\mathbb{F}_{2^{128}}$ kann zu einem Komplettbruch von GCM verwendet werden, falls eine Nonce wiederverwendet wird. In der Vorlesung haben wir hierfür die Algorithmen besprochen, die in einem dreistufigen Verfahren eigesetzt werden. Im ersten Schritt ist dies die quadratfreie Faktorisierung (squarefree factorization), danach die gradeindeutige Faktorisierung (distinct degree factorization) um zuletzt den Cantor-Zassenhaus-Algorithmus, also die gradgleiche Faktorisierung (equal degree factorization) anzuwernden.
Um den Knackpunkt des Algorithmus zu verstehen, müssen wir ein wenig ausholen. Wie in der Vorlesung besprochen wird im GHASH des GCM ja ein Polynom folgender Form berechnet:

$$f = c_3X^3 + c_2X^2 + c_1X + c_0, \qquad c_3 \neq 0$$

Bei GCM ist $H = X$, das heißt, die Nullstellen des Polynoms sind jeweils Kandidaten für den Schlüssel H. Da es bei einem Polynom n-ten Grades nur maximal n Nullstellen gibt, ist es recht einfach, unter den sehr wenigen Kandidaten den korrekten Schlüssel zu ermitteln.
Beachten Sie hierbei, dass die Koeffizienten des Polynoms Feldelemente im $\mathbb{F}_{2^{128}}$ sind; wir haben also ein Polynom in X, welches wiederum Polynome als Koeffizienten hat, welche Potenzen des Generators x (manchmal auch als α bezeichnet)

sind. Wir bauen also auf der in den vorigen Aufgaben gebauten polynomiellen Finitfeldarithmetik auf.
Die Grundidee des Algorithmus von Cantor-Zassenhaus ist bestechend: Sie basiert auf der Tatsache, dass für einen Wert q, welcher die Ordnung des darunterliegenden Feldes angibt (in unserem Fall Charakteristik 2, d.h. $q = 2^{128}$) das Polynom

$$X^{\frac{q^i-1}{3}} - X$$

garantiert alle irreduziblen Polynome i-ten Grades (also eben genau aller Generatorpoynome) teilt.
Wenn nun ein randomisiertes Polynom gewählt wird und dies hoch q^i exponiert wird, zwingen wir das Ergebnis mit vergleichsweise hoher Wahrscheinlichkeit in beiden enthaltenen Untergruppen der Faktoren auf nichttriviale Elemente. Dann lassen sich die Faktoren durch einfache Anwendung des euklidischen Algorithmus zur Ermittlung des ggT finden.
Dies wirkt zunächst nicht wie eine sinnvolle Eigenschaft, da Arithmetik mit einem Polynom vom Grad 2^{128} natürlich prohibitiv ist. Allerdings wird die Berechnung möglich, wenn man erkennt, dass die Berechnung der polynomiellen Exponentiation eben modulo f geschehen kann und das Ergebnis identisch bleibt. Das heißt, Sie müssen hier eine modulare polynomielle Exponentiation (analog des square/multiply-Algorithmus) entwickeln, der nach jedem Schritt eine polynomielle Reduktion (also Polynomdivision) durchführt. Die hierfür benötigten Algorithmen sind mit aus der Schule bekannter Mathematik durchführbar; dass die Koeffizienten Elemente des $F_{2^{128}}$ sind, spielt hier keine Rolle.

`gfpoly_sort`: Totale Ordnung von Polynomen

Für die Implementierung insbesondere probabilistischer Algorithmen ist es notwendig, sich auf eine Konvention zu einigen, in der Polynome sortiert zurückgegeben werden sollen. Wir definieren hierfür eine Relation $\overset{\circ}{<}$ welches eine Menge paarweiser ungleicher Polynome in eine totale Ordnung überführt. Es gilt zunächst, dass nach Grad sortiert wird, also grundsätzlich gilt für zwei Polynome Q, W:

$$\deg Q < \deg W \implies Q \overset{\circ}{<} W$$

Im Fall zweier gradgleichen Polynome Q, W, also der Form:

$$Q = q_0 X^0 + q_1 X^1 + q_2 X^2 + \ldots + q_n X^n \quad q_n \neq 0$$
$$W = w_0 X^0 + w_1 X^1 + w_2 X^2 + \ldots + w_n X^n \quad w_n \neq 0$$
$$\deg Q = \deg W = n$$

Hierbei wird weiter nach Koeffizienten verglichen, aufsteigend von der größten Potenz zur kleinsten Potenz. Das heißt konkret, wenn die höchsten e Koeffizienten von Q und W identisch sind, gilt:

$$q_i = w_i \quad \forall i \in \{n - e + 1 \ldots n\}$$

$$q_{n-e} \overset{\circ}{<} w_{n-e} \implies Q \overset{\circ}{<} W$$

Zwei Feldelemente im $\mathbb{F}_{2^{128}}$ werden ebenfalls nach der gleichen Konvention verglichen, d.h. Koeffizientenvergleich angefangen von α^{127} bis hin zu α^0.

```
{
    "action": "gfpoly_sort",
    "arguments": {
        "polys": [
            [
                "NeverGonnaGiveYouUpAAA==",
                "NeverGonnaLetYouDownAA==",
                "NeverGonnaRunAroundAAA==",
                "AndDesertYouAAAAAAAAAA=="
            ],
            [
                "WereNoStrangersToLoveA==",
                "YouKnowTheRulesAAAAAAA==",
                "AndSoDoIAAAAAAAAAAAAAA=="
            ],
            [
                "NeverGonnaMakeYouCryAA==",
                "NeverGonnaSayGoodbyeAA==",
                "NeverGonnaTellALieAAAA==",
                "AndHurtYouAAAAAAAAAAAA=="
            ]
        ]
    }
}
```

Erwartete Antwort:


```
{
    "sorted_polys": [
        [
            "WereNoStrangersToLoveA==",
            "YouKnowTheRulesAAAAAAA==",
            "AndSoDoIAAAAAAAAAAAAAA=="
        ],
        [
            "NeverGonnaMakeYouCryAA==",
            "NeverGonnaSayGoodbyeAA==",
            "NeverGonnaTellALieAAAA==",
            "AndHurtYouAAAAAAAAAAAA=="
        ],
        [
            "NeverGonnaGiveYouUpAAA==",
            "NeverGonnaLetYouDownAA==",
            "NeverGonnaRunAroundAAA==",
            "AndDesertYouAAAAAAAAAA=="
        ]
    ]
}
```

`gfpoly_make_monic`: Polynom in monische Form überführen

Diese Funktion teilt ein Polynom A durch ihren führenden Term (den Koeffizienten des höchsten Exponenten von X).

```
{
    "action": "gfpoly_make_monic",
    "arguments": {
        "A": [
            "NeverGonnaGiveYouUpAAA==",
            "NeverGonnaLetYouDownAA==",
            "NeverGonnaRunAroundAAA==",
            "AndDesertYouAAAAAAAAAA=="
        ]
    }
}
```

Erwartete Antwort:

```
{
    "A*": [
        "edY47onJ4MtCENDTHG/sZw==",
        "oaXjCKnceBIxSavZ9eFT8w==",
        "1Ial5rAJGOucIdUe3zh5bw==",
        "gAAAAAAAAAAAAAAAAAAAAA=="
    ]
}
```

`gfpoly_sqrt`: Quadratwurzel eines Polynoms

Diese Funktion gibt für ein Polynom Q, welches nur für gerade Exponenten von X Koeffizienten ungleich Null hat, die Quadratwurzel $S = \sqrt{Q}$ zurück.

```
{
    "action": "gfpoly_sqrt",
    "arguments": {
        "Q": [
            "5TxUxLHO1lHE/rSFquKIAg==",
            "AAAAAAAAAAAAAAAAAAAAAA==",
            "0DEUJYdHlmd4X7nzzIdcCA==",
            "AAAAAAAAAAAAAAAAAAAAAA==",
            "PKUa1+JHTxHE8y3LbuKIIA==",
            "AAAAAAAAAAAAAAAAAAAAAA==",
            "Ds96KiAKKoigKoiKiiKAiA=="
        ]
    }
}
```

Erwartete Antwort:

```
{
    "S": [
        "NeverGonnaGiveYouUpAAA==",
        "NeverGonnaLetYouDownAA==",
        "NeverGonnaRunAroundAAA==",
        "AndDesertYouAAAAAAAAAA=="
    ]
}
```

`gfpoly_diff`: Polynom differenzieren

Diese Funktion gibt für ein gegebenes Polynom F dessen erste Ableitung F' zurück.

```
{
    "action": "gfpoly_diff",
    "arguments": {
        "F": [
            "IJustWannaTellYouAAAAA==",
            "HowImFeelingAAAAAAAAAA==",
            "GottaMakeYouAAAAAAAAAA==",
            "UnderstaaaaaaaaaaaaanQ=="
        ]
    }
}
```

Erwartete Antwort:

```
{
    "F'": [
        "HowImFeelingAAAAAAAAAA==",
        "AAAAAAAAAAAAAAAAAAAAAA==",
        "UnderstaaaaaaaaaaaaanQ=="
    ]
}
```

`gfpoly_gcd`: ggT von zwei Polynomen berechnen

Diese Funktion gibt für zwei gegebenee Polynome A, B das monische Polynom $G = \gcd(A, B)$ zurück.

```
{
    "action": "gfpoly_gcd",
    "arguments": {
        "A": [
            "DNWpXnnY24XecPa7a8vrEA==",
            "I8uYpCbsiPaVvUznuv1IcA==",
            "wsbiU432ARWuO93He3vbvA==",
            "zp0g3o8iNz7Y+8oUxw1vJw==",
            "J0GekE3uendpN6WUAuJ4AA==",
            "wACdOe6u1ii4AAAAAAAAAA==",
            "ACAAAAAAAAAAAAAAAAAAAA=="
        ],
        "B": [
            "I20VjJmlSnRSe88gaDiLRQ==",
            "0Cw5HxJm/pfybJoQDf7/4w==",
            "8ByrMMf+vVj5r3YXUNCJ1g==",
            "rEU/f2UZRXqmZ6V7EPKfBA==",
            "LfdALhvCrdhhGZWl9l9DSg==",
            "KSUKhN0n6/DZmHPozd1prw==",
            "DQrRkuA9Zx279wAAAAAAAA==",
            "AhCEAAAAAAAAAAAAAAAAAA=="
        ]
    }
}
```

Erwartete Antwort:

```
{
    "G": [
        "NeverGonnaMakeYouCryAA==",
        "NeverGonnaSayGoodbyeAA==",
        "NeverGonnaTellALieAAAA==",
        "AndHurtYouAAAAAAAAAAAA==",
        "gAAAAAAAAAAAAAAAAAAAAA=="
    ]
}
```

`gfpoly_factor_sff`: Square-free Factorization

Implementieren Sie den Algorithmus, der ein monisches Polynom f in seine quadratfreien Faktoren zerlegt, wie in Alg. 1 gezeigt.

Algorithm 1 SFF-Algorithmus

Eingabe: Monisches Polynom f mit Koeffizienten in $\mathbb{F}_{2^{128}}$
Ausgabe: Menge an Faktoren und deren respektive Multiplizitäten

1: **procedure** SFF(f)
2: $c \leftarrow \gcd(f, f')$
3: $f \leftarrow \frac{f}{c}$
4: $z \leftarrow \varnothing$
5: $e \leftarrow 1$ ▷ Multiplizität
6: **while** $f \neq 1$ **do**
7: $y \leftarrow \gcd(f, c)$
8: **if** $f \neq y$ **then** ▷ Faktor gefunden
9: $z \leftarrow z \cup \{(\frac{f}{y}, e)\}$
10: **end if**
11: $f \leftarrow y$
12: $c \leftarrow \frac{c}{y}$
13: $e \leftarrow e + 1$
14: **end while**
15: **if** $c \neq 1$ **then** ▷ Rekursiver Aufruf
16: $r \leftarrow \text{SFF}(\sqrt{c})$
17: **for** $(f^*, e^*) \in r$ **do**
18: $z \leftarrow z \cup \{(f^*, 2e^*)\}$ ▷ Alle Exponenten verdoppeln
19: **end for**
20: **end if**
21: **return** z
22: **end procedure**

Die Rückgabeliste der Faktoren soll aufsteigend nach Polynomen sortiert sein.

```
{
    "action": "gfpoly_factor_sff",
    "arguments": {
        "F": [
            "vL77UwAAAAAAAAAAAAAAAA==",
            "mEHchYAAAAAAAAAAAAAAAA==",
            "9WJaOMAAAAAAAAAAAAAAAA==",
            "akHfwWAAAAAAAAAAAAAAAA==",
            "E12o/QAAAAAAAAAAAAAAAA==",
            "vKJ/FgAAAAAAAAAAAAAAAA==",
            "yctWwAAAAAAAAAAAAAAAA==",
            "c1BXYAAAAAAAAAAAAAAAA==",
            "oOAtAAAAAAAAAAAAAAAA==",
            "AbP2AAAAAAAAAAAAAAAA==",
            "k2YAAAAAAAAAAAAAAAAA==",
            "vBYAAAAAAAAAAAAAAAAA==",
```

```
            "dSAAAAAAAAAAAAAAAAAAAA==",
            "69gAAAAAAAAAAAAAAAAAA==",
            "VkAAAAAAAAAAAAAAAAAAAA==",
            "a4AAAAAAAAAAAAAAAAAAAA==",
            "gAAAAAAAAAAAAAAAAAAAA=="
        ]
    }
}
```

Erwartete Antwort:

```
{
    "factors": [
        {
            "factor": [
                "q4AAAAAAAAAAAAAAAAAAAA==",
                "gAAAAAAAAAAAAAAAAAAAA=="
            ],
            "exponent": 1
        },
        {
            "factor": [
                "iwAAAAAAAAAAAAAAAAAAAA==",
                "CAAAAAAAAAAAAAAAAAAAAA==",
                "AAAAAAAAAAAAAAAAAAAAAA==",
                "gAAAAAAAAAAAAAAAAAAAA=="
            ],
            "exponent": 2
        },
        {
            "factor": [
                "kAAAAAAAAAAAAAAAAAAAAA==",
                "CAAAAAAAAAAAAAAAAAAAAA==",
                "wAAAAAAAAAAAAAAAAAAAAA==",
                "gAAAAAAAAAAAAAAAAAAAA=="
            ],
            "exponent": 3
        }
    ]
}
```

`gfpoly_factor_ddf`: Distinct-degree Factorization

Implementieren Sie den Algorithmus, der ein monisches Polynom f in Faktoren zerlegt, die jeweils ein Produkt aus gradgleichen Polynomen sind. Diesen Algorithmus sehen Sie in Alg. 2 abgebildet.

Algorithm 2 DDF-Algorithmus

Eingabe: Monisches, quadratfreies Polynom f mit Koeffizienten in $\mathbb{F}_{2^{128}}$
Ausgabe: Menge an Faktoren und der Grad derjenigen Polynome, aus denen die Faktoren ein Produkt sind

1: **procedure** DDF(f)
2: $q \leftarrow 2^{128}$
3: $z \leftarrow \varnothing$
4: $d \leftarrow 1$ ▷ Grad der beinhalteten Polynome
5: $f^* \leftarrow f$
6: **while** $\deg f^* \geq 2d$ **do**
7: $h \leftarrow X^{q^d} - X \mod f^*$
8: $g \leftarrow \gcd(h, f^*)$
9: **if** $g \neq 1$ **then** ▷ Faktor gefunden
10: $z \leftarrow z \cup \{(g, d)\}$
11: $f^* \leftarrow \frac{f^*}{g}$
12: **end if**
13: $d \leftarrow d + 1$
14: **end while**
15: **if** $f^* \neq 1$ **then**
16: $z \leftarrow z \cup \{(f^*, \deg f^*)\}$
17: **else if** $\#z = 0$ **then**
18: $z \leftarrow z \cup \{(f, 1)\}$
19: **end if**
20: **return** z
21: **end procedure**

Die Rückgabeliste der Faktoren soll aufsteigend nach Polynomen sortiert sein.

```
{
    "action": "gfpoly_factor_ddf",
    "arguments": {
        "F": [
            "tpkgAAAAAAAAAAAAAAAAAA==",
            "m6MQAAAAAAAAAAAAAAAA==",
            "8roAAAAAAAAAAAAAAAAA==",
            "3dUAAAAAAAAAAAAAAAAA==",
            "FwAAAAAAAAAAAAAAAAAA==",
            "/kAAAAAAAAAAAAAAAAAA==",
            "a4AAAAAAAAAAAAAAAAAA==",
            "gAAAAAAAAAAAAAAAAAAA=="
        ]
    }
}
```

Erwartete Antwort:


```
{
    "factors": [
        {
            "factor": [
                "q4AAAAAAAAAAAAAAAAAAAA==",
                "gAAAAAAAAAAAAAAAAAAAAA=="
            ],
            "degree": 1
        },
        {
            "factor": [
                "mmAAAAAAAAAAAAAAAAAAAA==",
                "AbAAAAAAAAAAAAAAAAAAAA==",
                "zgAAAAAAAAAAAAAAAAAAAA==",
                "FwAAAAAAAAAAAAAAAAAAAA==",
                "AAAAAAAAAAAAAAAAAAAAAA==",
                "wAAAAAAAAAAAAAAAAAAAAA==",
                "gAAAAAAAAAAAAAAAAAAAAA=="
            ],
            "degree": 3
        }
    ]
}
```

`gfpoly_factor_edf`: Equal Degree Factorization

Implementieren Sie den Algorithmus von Cantor-Zassenhaus zur Equal Degree Factorization wie in Alg. 3 gezeigt.

Algorithm 3 EDF-Algorithmus von Cantor-Zassenhaus

Eingabe: Quadratfreies monisches Polynom f welches ein Produkt aus Polynomen des Grades d ist
Ausgabe: Menge der beinhalteten Faktoren

1: **procedure** EDF(f, d)
2: $q \leftarrow 2^{128}$
3: $n \leftarrow \frac{\deg f}{d}$ ▷ Anzahl beinhalteter Polynome
4: $z \leftarrow \{f\}$ ▷ Gefundene Faktoren
5: **while** $\#z < n$ **do** ▷ Noch nicht alle Faktoren gefunden
6: $h \leftarrow$ RandPoly() ▷ zufälliges Polynom mit $\deg h < \deg f$
7: $g \leftarrow h^{\frac{q^d-1}{3}} - 1 \mod f$
8: **for** $u \in z$ **do**
9: **if** $\deg u > d$ **then**
10: $j = \gcd(u, g)$
11: **if** $j \neq 1$ and $j \neq u$ **then**
12: $z = (z \setminus \{u\}) \cup \{j, \frac{u}{j}\}$
13: **end if**
14: **end if**
15: **end for**
16: **end while**
17: **return** z
18: **end procedure**

Die Rückgabeliste der Faktoren soll aufsteigend nach Polynomen sortiert sein.

```
{
    "action": "gfpoly_factor_edf",
    "arguments": {
        "F": [
            "mmAAAAAAAAAAAAAAAAAAAA==",
            "AbAAAAAAAAAAAAAAAAAAAA==",
            "zgAAAAAAAAAAAAAAAAAAAA==",
            "FwAAAAAAAAAAAAAAAAAAAA==",
            "AAAAAAAAAAAAAAAAAAAAAA==",
            "wAAAAAAAAAAAAAAAAAAAAA==",
            "gAAAAAAAAAAAAAAAAAAAAA=="
        ],
        "d": 3
    }
}
```

Erwartete Antwort:

```
{
    "factors": [
```

```
        [
            "iwAAAAAAAAAAAAAAAAAAAA==",
            "CAAAAAAAAAAAAAAAAAAAAA==",
            "AAAAAAAAAAAAAAAAAAAAAA==",
            "gAAAAAAAAAAAAAAAAAAAAA=="
        ],
        [
            "kAAAAAAAAAAAAAAAAAAAAA==",
            "CAAAAAAAAAAAAAAAAAAAAA==",
            "wAAAAAAAAAAAAAAAAAAAAA==",
            "gAAAAAAAAAAAAAAAAAAAAA=="
        ]
    ]
}
```

`gcm_crack`: Vollständiger Bruch von GCM

Verwenden Sie die Implementierung Ihrer Algorithmen um den Hashschlüssel H aus GCM zu errechnen. Ihnen ist garantiert dass die Nonces von der Nachrichten `m1`, `m2`, `m3` und der von Ihnen zu fälschenden Nachricht `forgery` identisch sind. Die Ciphertexte und/oder assozierte Daten der drei Nachrichten sind allerdings verschieden. Sie müssen also die GHASH-Polynome für die verschiedenen Nachrichten aufstellen:

$$c_n H^n + c_{n-1} H^{n-1} + \ldots + c_2 H^2 + LH + E_K(Y_0) = T$$

Diese Gleichungen müssen Sie umformen und das sich ergebende Gleichungssystem lösen. Hierfür finden Sie die Nullstellen, indem Sie die das Polynom mittels SFF, DDF und EDF in Ihre Faktoren zerlegen. Die Faktoren, die für uns hier relevant sind, sind diejenigen des Grades 1, also Faktoren der Form:

$$X + c$$

Jeder dieser Koeffizienten c ist ein Kandidat für den GCM Authentisierungsschlüssel H. Um den korrekten Kandidaten zu ermitteln, müssen Sie zunächst die Gleichungen nach $E_K(Y_0)$ auflösen. Unter Kenntnis von einem H-Kandidaten und dem entsprechenden $E_K(Y_0)$-Kandidaten können Sie dann überprüfen, ob Sie damit eine dritte Nachricht `m3` erfolgreich authentisieren können. Wenn dies der Fall ist, haben Sie den korrekten GHASH-Schlüssel H gefunden und können diese korrekte $H/E_K(Y_0)$-Kombination verwenden, um die vierte Nachricht zu authentisieren. Sie geben in Ihrer Antwort dann die Maske (also $E_K(Y_0)$), den Authentisierungsschlüssel H sowie das Authentisierungs-Tag für die gefälschte Nachricht zurück.

Testcase für `gcm_crack`:


```
{
    "action": "gcm_crack",
    "arguments": {
        "nonce": "4gF+BtR3ku/PUQci",
        "m1": {
            "ciphertext": "CGOkZDnJEt24aVV8mqQq+P4pouVDWhAYj0SN5MDAgg==",
            "associated_data": "TmFjaHJpY2h0IDE=",
            "tag": "GC9neV3aZLnmznTIWqCC4A=="
        },
        "m2": {
            "ciphertext": "FnWyLSTfRrO8Y1MuhLIs6A==",
            "associated_data": "",
            "tag": "gb2ph1vzwU85/FsUg51t3Q=="
        },
        "m3": {
            "ciphertext": "CGOkZDnJEt25aV58iaMt6O8+8chKVh0Eg1XFxA==",
            "associated_data": "TmFjaHJpY2h0IDM=",
            "tag": "+/aDjsAzTseDLuM4jt5Q6Q=="
        },
        "forgery": {
            "ciphertext": "AXe/ZQ==",
            "associated_data": ""
        }
    }
}
```


Erwartete Antwort:


```
{
    "tag": "Y16EEEO1sgJX3IsJSwEXlA==",
    "H": "Nxn7h7ruk8eiNAG6AfhUFg==",
    "mask": "tXjFK5vCqIPl6fKAJAyy9A=="
}
```