Aufgabenblatt 08

PS Einführung in die Kryptographie

Andreas Schlager

18. Mai 2025

Inhaltsverzeichnis

Aufgabe 27. Zur Known-Plaintext Attacke gegen 2 Key EDE TripleDES (Slide 138f): Erklären/beweisen sie die angegebene Angriffskomplexität / Laufzeit der Oorschot, Wiener Attacke und die Anzahl der im Mittel notwendigen Versuche für a. Hinweis: sehen sie sich die Originalarbeit dazu an.

Zeitkomplexität Zur Berechnung der asymptotischen Zeitkomplexität des Angriffs, wird für jeden Teilschritt die Komplexität angegeben.

1. Gegeben eine Tabelle mit Plaintext- und Ciphertext Paaren (Tabelle 1, sortiert nach Plaintexten).
   (a) Sortieren benötigt bei p Paaren $\mathcal{O}(p \log p)$.
2. Erraten/wählen Sie einen fixen ersten Zwischenwert $a = E_{K_1}(P)$.
   (a) Ein fixes a wählen geht in $\mathcal{O}(1)$.
3. Bestimmen sie durch die Berechnung von $P = D_{K_1}(a)$ den Plaintext und suchen den dazugehörenden Ciphertext C in Tabelle 1, dies für alle K_1.
   (a) Pro K_1:
      i. Berechnung von $P = D_{K_1}(a)$ in $\mathcal{O}(1)$.
      ii. In der Tabelle suchen $\mathcal{O}(\log p)$.
   (b) Insgesamt gibt es für $K \in \{0,1\}^n$ 2^n Durchläufe.

   Somit ist die Gesamtkomplexität für diesen Schritt $\mathcal{O}(2^n \log p)$.
4. Tabulieren Sie für alle K_1 (für das fixe a) den zweiten Zwischenwert, $b = D_{K_1}(C)$ (Tabelle 2).

   Entspricht wieder einem gesamten Durchlauf der Schlüssel (2^n). Bei jedem wird b in $\mathcal{O}(1)$ berechnet. Die Komplexität beträgt daher $\mathcal{O}(2^n)$.
5. Suchen Sie in der Tabelle 2 für alle möglichen K_2 Elemente mit passendem zweiten Zwischenwert $b = D_{K_2}(a)$.
   (a) Pro K_2:
      i. Berechne $b = D_{K_2}(a)$ in $\mathcal{O}(1)$.
      ii. Prüfe b in Tabelle 2 (angenommen Hashtabelle) in $\mathcal{O}(1)$.
   (b) Insgesamt gibt es wieder 2^n K_2, also auch 2^n Durchläufe.

   Gesamtkomplexität ist $\mathcal{O}(2^n)$.

6. Bei Übereinstimmung ist ein mögliches Schlüsselpaar gefunden (mit weiteren Plaintext / Ciphertextpaaren verifizieren). Ansonsten wird ein neues a gewählt.

   Verifikation durch p bekannte Paare in $\mathcal{O}(p)$.

Durch die Schritte 3, 4 & 5 ergeben sich folgende Laufzeiten

$$\mathcal{O}(2^n \log p), \mathcal{O}(2^n), \mathcal{O}(2^n).$$

Im schlechtesten Fall müssen alle Zwischenwert für $a \in \{0,1\}^m$ überprüft werden.

$$\mathcal{O}(2^n \cdot 2^m) = \mathcal{O}(2^{n+m})$$

Wahrscheinlichkeit Es existieren 2^m mögliche Werte für a. In der Datenbank befinden sich p Einträge. Unter der Annahme, dass jeder Zwischenwert der Klartextpaare gleichverteilt ist, ist die Wahrscheinlichkeit einen Treffer zu landen

$$P(\text{Treffer}) = \frac{p}{2^m}.$$

Nimmt man dieses Wahrscheinlichkeit als die Wahrscheinlichkeit für ein Bernoulli-Experiment an, gibt uns die geometrische Verteilung den Erwartungswert für die Anzahl N an unterschiedlichen Werten für a.

$$\mathbb{E}[N] = \frac{1}{P(\text{Treffer})} = \frac{2^m}{p}$$

Im Schnitt müssen wir also nicht alle 2^m Werte für a probieren (wie im worst-case), sondern nur $\mathbb{E}[N]$ viele. Die erwartete Laufzeitkomplexität ist

$$\mathcal{O}\left(2^n \cdot \frac{2^m}{p}\right) = \mathcal{O}\left(\frac{2^{n+m}}{p}\right)$$

Fazit Je größer die Datenbank an Plaintext-Ciphertext-Paaren ist, desto schneller geht der Angriff.

Aufgabe 28. Erklären sie warum die Berechnung von Diskrete Logarithmen und Quadratwurzeln in Modulararithmetik "computationally infeasible" ist, während in klassischer Arithmetik die entsprechenden Berechnungen nicht besonders aufwändig sind.

Diskreter Logarithmus Der diskrete Logarithmus einer Zahl m zur Basis a modulo p ist der kleinste Exponent x der Gleichung

$$a^x \equiv m \mod p.$$

Das Problem bei der Berechnung ist, dass die Funktion $f(x) = a^x \mod p$ aufgrund der Modulararithmetik nicht stetig wächst oder fällt. Die Ergebnisse für unterschiedliche x springen ohne klare Richtung. Somit kann man sich der korrekten Lösung nicht annähern und es bleibt nur eine Brute-Force-Approach.

Example 1. Sei $m = 3, p = 11$ und $a = 2$.[1] Man sieht in den Ergebnissen kein klar erkennbares Muster.

$$\begin{aligned}
2^1 &= 2 \equiv 2 \mod 11 \\
2^2 &= 4 \equiv 4 \mod 11 \\
2^3 &= 8 \equiv 8 \mod 11 \\
2^4 &= 16 \equiv 5 \mod 11 \\
2^5 &= 32 \equiv 10 \mod 11 \\
2^6 &= 64 \equiv 9 \mod 11 \\
2^7 &= 128 \equiv 7 \mod 11 \\
2^8 &= 256 \equiv 3 \mod 11 \\
2^9 &= 512 \equiv 6 \mod 11 \\
2^{10} &= 1024 \equiv 1 \mod 11
\end{aligned}$$

Quadartwurzel Das ziehen der Quadartwurzel bereitet im Restklassenringen mehrere Schwierigkeiten. Eine davon ist, dass viele Zahlen überhaupt keine Wurzel besitzen. Betrachtet man etwa den Restklassenring $\mathbb{Z}_7$ und sucht x, sodass $x^2 \equiv 3 \mod 7$, wird man keine Lösung finden (siehe Tab 1). Besonders interessant ist allerdings auch, dass sich das Ziehen der Quadartwurzel im Restklassenring auf das Faktorisierungsproblem reduzieren lässt, welches bekanntlich in NP liegt.

[1] Werte und Zahlen von Wikipedia übernommen und nachgerechnet.

x	$x^2 \mod 7$
0	0
1	1
2	4
3	2
4	2
5	4
6	1

Tabelle 1: Ergebnisse für x^2 in $\mathbb{Z}_7$

Aufgabe 29. Erklären sie (und rechnen sie ein konkretes Beispiel durch), wie modulare Inversion mit dem erweiterten Euklidischen Algorithmus realisiert werden kann.

Der erweiterte euklidische Algorithmus berechnet für zwei Zahlen $a, b \in \mathbb{N}$ den größten gemeinsamen Teiler $\gcd(a, b)$. Außerdem liefert der Algorithmus zwei weitere Zahlen $s, t \in \mathbb{Z}$, sodass der größte gemeinsame Teiler als Linearkombination dieser Zahlen geschrieben werden kann (Lemma von Bézout).

$$\gcd(a, b) = s \cdot a + t \cdot b$$

Das modulare Inverse einer Zahl $a \in \mathbb{N}$ ist eine Zahl $x \in \mathbb{Z}_n$, sodass

$$a \cdot x \equiv 1 \mod n.$$

Um x zu bestimmen, berechnet man zuerst über den erweiterten euklidischen Algorithmus das Tupel $(\gcd(a, n), s, t)$. Ein modulares Inverses existiert genau dann, wenn $\gcd(a, n) = 1$ (also a und n teilerfremd sind). Nach dem Lemma, lässt sich der größte gemeinsame Teiler dann wie folgt schreiben.

$$\begin{aligned} \gcd(a, n) = 1 &= s \cdot a + t \cdot n \\ 1 &\equiv s \cdot a + t \cdot n \mod n \\ 1 &\equiv s \cdot a \mod n \end{aligned}$$

Somit ist s das modulare Inverse zu a modulo n.

Algorithmus Pseudocode des erweiterten euklidischen Algorithmus [2]

[2] Wikipedia, rekurisver, erweiterter euklidischer Algorithmus

```
a,b: zwei Zahlen für die der erweiterte euklidische Algorithmus durchgeführt wird

extended_euclid(a,b)
1  wenn b = 0
2      dann return (a,1,0)
3  (d',s',t') ← extended_euclid(b, a mod b)
4  (d,s,t) ← (d',t',s' - (a div b)t')
5  return (d,s,t)
```

Example 2. Seien $a = 7, n = 15$. Zuerst berechnet man den größten gemeinsamen Teiler wie beim regulären euklidischen Algorithmus. q ist dabei das Ergebnis der ganzzahligen Division von a und n.

Iteration	a	n	q
1	7	15	0
2	15	7	2
3	1	1	1
4	1	0	

Sobald $n = 0$ ist, steht in a der ggT. Beginnend vom Tabellenende werden nun s, t berechnet. Dafür soll in jeder Zeile $1 = s \cdot a + t \cdot n$ gelten. Dementsprechend wird in der letzten Zeile für $s = 1$ und $t = 0$ eingetragen. Nun arbeitet man sich in der Tabelle hoch und berechnet die weiteren s, t wie folgt

$$s = t_{\text{alt}}$$
$$t = s_{\text{alt}} - q \cdot t_{\text{alt}}.$$

Das q ist immer aus der aktuellen Zeile.

Iteration	a	n	q	s	t
1	7	15	0	-2	1
2	15	7	2	1	-2
3	1	1	1	0	1
4	1	0		1	0

In der obersten Zeile steht nun das Ergebnis für s, t.

$$\gcd(7, 15) = 1 = -2 \cdot 7 + 1 \cdot 15$$

Entsprechend der obigen Erklärung wäre das modulare Inverse zu 7 mod 15 also -2.

$$7 \cdot (-2) \equiv 1 \mod 15$$

Aufgabe 30. Beweisen Sie: Ist $n = p \cdot g$ mit $p, g \in \mathbb{P}$: $\phi(n) = (p-1)(g-1)$.

Beweis. $\phi(n) = (p-1)(g-1)$.

n ist eine zusammengesetzte Zahl, deren Primfaktorenzerlegung genau $p \cdot g$ ist. Somit sind alle Vielfachen der Primzahlen, die nicht größer als n sind, **nicht** teilerfremd zu n (der gemeinsame Teiler sind die entsprechenden Primzahlen).

$$|\{k \cdot p \mid 1 \leq k \leq g\}| = |\{p, 2p, 3p, \ldots, g \cdot p\}| = g$$
$$|\{k \cdot g \mid 1 \leq k \leq p\}| = |\{g, 2g, 3g, \ldots, p \cdot g\}| = p$$

Ausgehend von alle in Frage kommenden Zahlen $p \cdot g$, ziehen wir jeweils obigen Anzahl ab und achten darauf, die Schnittmenge (n) nur einmal abzuziehen.

$$\begin{aligned}\phi(n) &= p \cdot g - 1 - p + 1 - g + 1 \\ &= p \cdot g - p - g + 1 \\ &= (p-1)(g-1)\end{aligned}$$

□