Kryptographie – ein Überblick für Fortgeschrittene

1. Caesar-Verschlüsselung: Idee, Angriff (bruteforce, Häufigkeit)

Einführung (Wikipedia): https://de.wikipedia.org/wiki/Caesar-Verschl%C3%BCsselung
Zitat Wikipedia:
Da die Größe des Schlüsselraums nur 25 beträgt, liegt nach Ausprobieren spätestens nach dem 25. Versuch der Klartext vor. Eine erschöpfende Schlüsselsuche (Exhaustion) ist bei der Caesar-Verschlüsselung trivial realisierbar. Da dies auch ohne Computer oder Rechenmaschine mit geringem Aufwand möglich ist, bestand die Sicherheit der Caesar-Verschlüsselung schon zu ihren Anfängen nicht auf der Geheimhaltung des Schlüssels, sondern im Wesentlichen auf der Geheimhaltung des Verfahrens, und entspricht damit nicht dem im 19. Jahrhundert postulierten Prinzip von Auguste Kerckhoffs.

Begriffe:
Klartext: Die von Menschen lesbare Nachricht
Geheimtext: Die verschlüsselte Nachricht (nicht unmittelbar menschenlesbar)
Kryptosystem: Ein Verfahren, welches einen Klartext durch Anwendung eines Schlüssels in einen Geheimtext wandelt und auch wieder zurückwandelt
Schlüssel: (s. Kryptosystem)
Klartextalphabet (Alle möglichen Symbole, die im Klartext vorkommen dürfen)
Geheimtextalphabet (Alle möglichen Symbole, die im Geheimtext vorkommen dürfen)
Schlüsselalphabet (Alle möglichen Symbole, die im Schlüssel vorkommen dürfen)

Beim Kryptosystem Caesar gilt: Sowohl Klartext als auch Geheimtext sind die 26 Buchstaben des lat. Alphabets. Schlüsselalphabet sind alle ganze Zahlen im Intervall [0;25]. Dieser fungiert als Verschiebungssummand.
Beispiel:

```
Klartext:                t i m i s t s p i t z e l
Schluessel:              2
Geheimtext:              V K O K U V U R K V B G N
```

2. Kerckhoffsches Prinzip (1883)

Verkürzt: "Die Sicherheit eines Verschlüsselungsverfahrens beruht auf der Geheimhaltung des Schlüssels anstatt auf der Geheimhaltung des Verschlüsselungsalgorithmus."
Problematisches Gegenprinzip: "Security through obscurity": Sicherheit durch Geheimhaltung des Verschlüsselungsalgorithmus selbst.
Warum ist das Problematisch? U.a. wegen:
* Es ist viel schwieriger, einen Algorithmus geheim zu halten als einen Schlüssel.
* Es ist schwieriger, einen enttarnten Algorithmus durch einen anderen zu ersetzen als einen enttarnten Schlüssel.
Beispiele für „Security through obscurity“:
• In gewisser Weise ist Caeser „Security through obscurity“, denn wenn man weiß, dass mit Caesar verschlüsselt worden ist, hat man den Klartext mit vertretbarem Aufwand gefunden.

• „Navajo-Code-Talkers“ (Film „Windtalkers“) im 2. WK: Angehörige eines Indianervolkes kommunizieren in ihrer Sprache, die vom Feind nicht verstanden werden kann.

3. Vigenere Verschlüsselung

Einführung (Wikipedia): https://de.wikipedia.org/wiki/Vigen%C3%A8re-Chiffre
Idee:
• Statt Verschiebungssummanden (0 – 25 wie bei Caesar) benutzt man die Buchstaben a-z
• Man benutzt nicht nur einen einzigen Schlüsselbuchstaben sondern ein Schlüsselwort und verschlüsselt aufeinanderfolgende Zeichen des Klartextes mit aufeinanderfolgenden Zeichen des Schlüssels.
Angriff durch Häufigkeitsanalyse bei kurzen(!) Schlüsseln und langen(!) Texten.
Wenn Schlüssellänge >= Klartextlänge und Schlüssel echt zufällig gewählt (nicht aus dem Wörterbuch) gilt, dann ist Vigenere tatsächlich unknackbar (da identisch mit One-Time-Pad)

Beispiel (ohne aufwändiges Vigenere-Quadrat): Klartext: "tim ist spitzel", Schlüssel "bagdad"

Klartext:	t	i	m	i	s	t	s	p	i	t	z	e	l
Schluessel:	b	a	g	d	a	d	*b*	*a*	*g*	*d*	*a*	*d*	*b*
(Verschiebung)	1	0	6	3	0	3	1	0	6	3	0	3	1
Geheimtext:	U	I	S	L	S	W	T	P	O	W	Z	H	M

4. Kryptosystem Digi-Vigenere oder: XOR-Verknüpfung

Den Namen „Digi-Vigenere“ habe ich mir nur ausgedacht. Das Prinzip wird aber überall angewandt:
Idee: Alle Alphabete (Klartext-, Schlüssel- und Geheimtextalphabet) bestehen nur aus {0,1}.
Beispiel:

Klartext:	00100110
Schluessel:	01001100
Geheimtext:	01101010

Digi-Vigenere ist also nichts anderes, als eine XOR-Verknüpfung ("entweder-oder" - aber nicht beides) von Klartext und Schlüssel. Hier ist der Schlüssel genauso lang wie der Klartext, so dass in diesem Fall die Sicherheit eines One-Time-Pads gegeben ist – s.u.)

5. One-Time-Pad

Recherchiere: https://de.wikipedia.org/wiki/One-Time-Pad
Das One-Time-Pad ist praktisch „Digi-Vigenere“ mit einem Schlüssel, der mindestens so lang ist, wie die Nachricht selbst.
Ein One-Time-Pad ist theoretisch unknackbar. Dies erkauft man sich mit folgenden Nachteilen:
Der Einmalschlüssel muss
- mindestens so lang sein wie die Nachricht,
- gleichverteilt zufällig gewählt werden,
- geheim bleiben und
- darf nicht wiederverwendet werden, auch nicht teilweise.

6. Kryptosystem „Vigenere mit Autokey“

Problem und Idee: Das Standard-Vigenere-System verwendet das Schlüsselwort bei langen Klartexten mehrmals hintereinander. Damit ist das Verfahren angreifbar (siehe https://de.wikipedia.org/wiki/Vigen%C3%A8re-Chiffre#Kryptanalyse). Es wäre also besser, wenn man den Schlüssel so erweitern könnte, dass nicht immer die gleichen Buchstaben vorkommen. Eine Methode ist „Autokey“: Man benutzt den Klartext selbst als Schlüssel (https://en.wikipedia.org/wiki/Autokey_cipher)!
Beispiel:
Klartext: `treffenumneunamturm`
Schlüssel: `otto`
Die ersten vier Buchstaben des Geheimtexts lauten zunächst: **HKXT**
Wenn der Empfänger die ersten vier Zeichen der Nachricht mit Standard-Vigenere entschlüsselt, so erhält dieser auch die ersten vier Zeichen des Klartexts, nämlich „tref“. Da der Empfänger diese Zeichen nun kennt, können diese also selbst als Schlüssel verwendet werden! Wenn der Absender den Schlüssel „otto“ verwendet, so wird beim Kryptosystem „Vigenere mit Autokey“ faktisch folgendermaßen verschlüsselt:
Klartext: `treffenumneunamturm`
Schlüssel: `ottotreffenumneunam`
Geheimtext: HKXTYVRZRRROZNQNHRY
Dieses konkrete System ist gegen klassische Vigenere-Angriffe sicher (aber nach aktuellem Standard längst nicht unknackbar: Siehe entsprechenden Abschnitt in obigem Autokey-Artikel!). Dennoch wird die Idee, praktisch unendlich lange Schlüssel zu verwenden, die irgendwie aus dem Passwort und Teilen des Klartextes erzeugt werden, auch heutzutage in der Praxis verwendet.

7. Stromchiffre (oder „Stream Cipher“) und Zufallsgeneratoren

Die Idee mit „unendlich langen Passwörtern“ lässt sich auch folgendermaßen umsetzen: Man entwirft einen Zufallszahlen-Generator, der sowohl bei Sender als auch Empfänger die gleiche Zufallszahlen-Folge erzeugt, die dann als Schlüssel verwendet wird. Weil dieser Generator auf beiden Seiten zuverlässig die gleichen Zahlen erzeugen muss, ist der Begriff „Zufall“ nicht mehr ganz zutreffend. Man spricht daher von „Pseudo-Zufallszahlen“. Diese kann man folgendermaßen erzeugen:
(Das Beispiel kann man direkt in der Windows-Powershell laufen lassen:)

```
$seed = 123;  # (Geheimer) Startwert
$c    = 8;    # c und m sollten
$m    = 15;   # teilerfremd sein!
$a    = 31;   # (a-1) sollte durch alle
              # Primfaktoren von m teilbar sein
              # (und durch 4, falls m durch 4 teilbar ist)

for ($i=0; $i -le 18; $i++) {
    $seed = ($a * $seed + $c) % $m
    Write-Host (" " + $i + " current seed: "+ $seed)
}
```

Die enthaltene Formel, die immer wieder neu mit dem aktualisierten Seed-Wert aufgerufen wird, erzeugt Zahlen zwischen 0 und 14 (also 15 verschiedene Zahlen, vgl. Wert für m). Wenn sich Sender und Empfänger auf einen geheimen Startwert einigen, können im Prinzip beliebig lange

Zufallszahlfolgen (man muss m nur genügend groß wählen) erzeugt werden.
ACHTUNG: Wie so oft, ist diese einfache Idee nicht sicher. Kryptografisch sichere Pseudo-Zufallszahl-Generatoren arbeiten ganz anders.

8. Zwischenfazit zu symmetrischen Verschlüsselungsverfahren

Wikipedia: „Ein symmetrisches Kryptosystem ist ein Kryptosystem, bei welchem im Gegensatz zu einem asymmetrischen Kryptosystem beide Teilnehmer denselben Schlüssel verwenden.“
Bereits Digi-Vigenere erlaubt es uns, beliebige Daten ziemlich sicher zu Verschlüsseln. (Vorsicht: In der Praxis sind Klartexte lang und Schlüssel kurz. In diesem Fall ist Digi-Vigenere sehr unsicher. Es gibt für diese Fälle wesentlich bessere - aber auch kompliziertere Kryptosysteme).
Allgemeines Problem bleibt bislang, dass ein Schlüssel immer im Geheimen ausgetauscht werden muss.

9. Diffie-Hellman-Schlüsselaustausch

Ein Verfahren, mit dem sich zwei Kommunikationspartner (online, für alle einsehbar) auf einen Schlüssel (z.B. für ein Strom-Chiffre) einigen können, ohne sich im Geheimen treffen zu müssen.
Wir nennen die Kommunikationspartner Alice und Bob. Ein Angreifer namens Carlo kann die Kommunikation belauschen.
Grundlegendes Prinzip ist, dass man eine mathematische Operation benutzt, die einfach zu berechnen ist (Potenzieren mit anschließendem Modulo), deren Umkehroperation aber nicht einfach ist.

Einführendes Beispiel: Angenommen, aus irgendwelchem Grund ist es auf dem Planeten Melmac ganz einfach, Additionen durchzuführen. Die Umkehroperation (Subtraktion) ist dort aber noch nicht erfunden worden.

Der „Diffie-Hellman-Schlüsselaustausch auf Melmac“ sieht nun so aus:

A) Alice schlägt eine Zahl g vor und schickt diese offen an Bob.	z.B.	g= 100
B) Alice denkt sich im geheimen für sich die Zahl a aus.	z.B.	a= 3
Alice rechnet x = a + g. Die Zahl x wird offen an Bob gesandt.	also	x=103
C) Bob denkt sich im geheimen für sich die Zahl b aus.	z.B.	b=8
Bob rechnet y = b+ g. Die Zahl y wird offen an Alice gesandt.	also	y=108
D) Alice rechnet s = y + a	also	s= 108 + 3 = 111
Bob rechnet s = x + b	also	s= 103 + 8 = 111

Der Clou ist: s ist für Angreifer Carlo nicht berechenbar, weil er nicht x – g subtrahieren kann um a zu errechnen.
Nun können wir auf der Erde sehr einfach Addieren UND Subtrahieren. Aber die Potenzierung mit anschließender Modulo-Rechnung ist nicht einfach umkehrbar.

Der echte Diffie-Hellman-Schlüsselaustausch funktioniert so:

A) Alice schlägt zwei Zahlen vor und schickt diese an Bob:
Eine (große) Primzahl p und eine ganze Zahl g aus dem Intervall [1;p[
Diese beiden Zahlen kann Carlo abhören.

B) Alice denkt sich im geheimen für sich eine Zahl a aus dem Intervall [1;p[aus.
Alice berechnet x = g^a mod p
Alice schickt x (für Carlo einsehbar) an Bob

C) Bob denkt sich im geheimen für sich eine Zahl b aus dem Intervall [1;p[aus.
Bob berechnet y = g^b mod p
Bob schickt y (für Carlo einsehbar) an Alice

D) Alice berechnet s = y^a mod p
Bob berechnet s = x^b mod p
Die berechnete Zahl s ist bei Bob und Alice identisch, aber für Carlo nicht ohne Weiteres ermittelbar (§§ Details später)

E) Die Zahl s kann nun als Schlüssel z.B. für einen Stromchiffre benutzt werden.

Eigenschaften der Modulo-Rechnung:
Beispiel: (4*4) mod 3 = ((4 mod 3) * (4 mod 3)) mod 3 = 1
(hilfreich um Zwischenergebnisse z.B. bei der Programmierung klein zu halten)

Idee beim obigen Schlüsselaustausch:
Alice rechnet: x = g^a mod p
Bob rechnet: y = g^b mod p
Alice rechnet: s = y^a = (g^b)^a mod p
Bob rechnet: s = x^b = (g^a)^b mod p

Warum ist das Diffie-Hellman-Verfahren „belauschungssicher“?
2^4 mod 5 zu berechnen ist einfach (hier: 2^4 mod 5 = 16 mod 5 = 1)
Aber aus 2^d mod 5 = 1 auf den Exponenten d zu schließen ist schwierig.
Im Allgemeinen muss man probieren:

```
Wertetabelle:   d  1 2 3 4
        2^d mod 5  2 4 3 1     also muss d gleich 4 sein.
```

Es gibt im Allgemeinen kein Verfahren, um schneller an das d zu gelangen! In der Praxis sind die beteiligten Zahlen natürlich riesig, so dass ein Durchprobieren nicht praktikabel ist.

10. Angriffsmöglichkeiten für den Lauscher Carlo:

Beispielkommunikation:
Alice schlägt vor: p = 13 g = 8 Alice denkt sich aus: a = 5
Alice rechnet: x = (8^5) mod 13 = 8
Bob denkt sich aus: b = 12
Bob rechnet: y = (8^12) mod 13 = 1
Alice rechnet den Schlüssel aus: s = y^a mod 13 = 1^5 mod 13 = 1
Bob rechnet den Schlüssel aus: s = x^b mod 13 = 8^12 mod 13 = 1

Carlo kennt die Zahlen: p = 13, g = 8, x = 8 , y = 1
Carlo versucht nun, die Zahl a zu ermitteln, indem er versucht, folgende Gleichung zu lösen:
8 = 8^a mod 13
Carlo MUSS die Gleichung durch probieren lösen:

```
Wertetabelle:
         a: 1 2  3  4  5
8^a mod 13: 8 12 5  1  8
```