Praktikum zur Kryptografie, 2008
==========

Generelle Hinweise: Alle Aufgaben dieses Praktikums werden in der Vorlesung noch im Detail erläutert. Bilden Sie Zweierteams fürs Praktikum, und bemühen Sie sich um gut strukturierten und klaren Code!

1. Aufgabe (Known-Plaintext Angriff gegen die Tausch-Chiffre)

Buchstabe	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Zahlcode	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Alice und Bob kennen **Modul** m(=26) und **Schlüssel** s(=7), t(=3).

Tauschchiffre verschlüsselt zeichenweise gemäß:

$$\textbf{geheim} = (\textbf{klar}+s)*t \bmod m$$

und entschlüsselt gemäß:

$$\textbf{klar} = \textbf{geheim}*t^{-1} - s \bmod m$$

Klartext:	h	a	l	l	o	b	o	b		**Alice**
	7	0	11	11	14	1	14	1	| +7	|
	14	7	18	18	21	8	21	8	| *3	|
	16	21	2	2	11	24	11	24		|
Geheimtext:	Q	V	C	C	L	Y	L	Y		**Bob**
	16	21	2	2	11	24	11	24	| *9(=inv 3)	|
	14	7	18	18	21	8	21	8	| -7	|
	7	0	11	11	14	1	14	1		|
Klartext:	h	a	l	l	o	b	o	b		**(ok)**

Zum Angriff reichen der AngreiferIn neben dem Geheimtext in vielen Fällen zwei (warum gerade zwei?) zugehörige Klartextzeichen (*Known Plaintext*), um die Schlüssel s und t herauszubekommen:

klar:	h	a			
	7	0			
geheim:	Q	V	C	C	...
	16	21	2	2	

Also gilt

$$(7+s)*t = 16 \bmod 26$$
$$(0+s)*t = 21$$
$$---$$
$$7*t = -5 \quad | *15(=\text{inv } 7)$$
$$t = -5*15$$
$$= -5*(-11) = 55 = 3$$
$$s*3 = 21 \quad | *9(=\text{inv } 3)$$
$$s = 21*9 = -5*9 = -45 = 7$$

Also wurden die Schlüssel **s=7, t=3** richtig rekonstruiert.

An diesem Beispiel erkennt man, was man zur Implementierung der Tauschchiffre alles benötigt:
Die Buchstaben-Zahl Codierung muß realisiert werden. Als Modul m wählen Sie am besten eine 8 Bit breite Primzahl, so daß Sie den ganzen ASCII-Zeichenbereich verfügbar haben.
Kern der Aufgabe ist die Implementierung der modularen Arithmetik (Operatoren **+,-,*** und **inv**, evtl. auch **div** und **mod**), die Sie am besten per Klasse implementieren. Den Modul m sollten Sie (als Attribut) flexibel halten.
Zentrales Feature bei der modularen Arithmetik ist sicherlich die modulare Inverse **inv**, die nicht irgendwie gemacht werden soll, sondern so wie im Kurs beschrieben (mit der direkten, sehr effizienten Methode).
Man braucht seriöse Fehlerbehandlung für den Fall, daß eine geforderte Inverse nicht existiert.
Eine klare Bedienstruktur des Programms ist von Vorteil, sinnvolle Menüpunkte könnten sein:
- Klartexteingabe (per Tastatur oder von Datei)
- Schlüsseleingabe (mit Kontrolle von t^{-1})
- verschlüsseln
- entschlüsseln
- Known-Plaintext-Angriff (mit Fehlerkontrolle, falls die dort geforderten Inversen nicht existieren)

Bei der Implementierung des Known-Plaintext-Angriffs kann man sich an obigem Beispiel orientieren und die nötigen Formeln gleich allgemein in den Code eintragen, weitere Details hierzu im Kurs.

Im anschließenden zweiten Praktikum wird es um genau die gleiche Tauschchiffre gehen, allerdings über einem anderen Zahlbereich, nämlich 8 Bit breite Galoiszahlen statt 8 Bit breite Modularzahlen. Die Programmstruktur wird genau die gleiche sein, dem Programm werden lediglich andere Zahlen *unterschoben*.
Es wäre insofern sinnvoll, die Arithmetikklasse gleich möglichst allgemein zu halten.

Tip: Starten Sie vielleicht mit der Implementierung der modularen Arithmetik und einer einfachen Testklasse dafür. Wenn das stimmt, ist der Rest simpel :-)

2. Aufgabe (Tausch-Chiffre mit Galoiszahlen)
Diese Aufgabe unterscheidet sich von der ersten nur dadurch, daß anstelle von Modularzahlen 8 Bit breite Galoiszahlen eingesetzt werden sollen, als Modul soll der Rijndael-Modul

$$p(x) = x^8+x^4+x^3+x+1 \approx 1.0001.1011 \approx \text{'11B'}$$

fest gewählt werden.
Einige Routinen, die bei Modularzahlen vom System bereitgestellt werden, müssen Sie diesmal selber implementieren:

- Multplizieren mit einer anderen Galoiszahl
- Reduzieren (auf Rechenbreite 8) nach Multiplikation
- Teilen durch eine andere Galoiszahl, also Berechnen von *div* und *mod* und zwar am besten in einer einzigen Routine, die beides bereitstellt.

Die Inverse und auch die zur Tauschchiffre gehörenden Programmteile sollten Sie eigentlich vom ersten Programm unverändert übernehmen können (vorausgesetzt Sie haben dort die Arithmetik trennscharf gekapselt).

Um die Effizienz Ihres Programms zu testen, entwerfen Sie einen Härtetest und zwar
entweder für die Inverse: Wählen Sie größenordnungsmäßig (je nach Effizienz Ihres Programms) eine Million Zufalls-Galoiszahlen, rufen Sie jeweils die Inverse auf und messen Sie die Zeit für das Ganze, am besten mit der Systemuhr.
oder für die Potenzierung: Wählen Sie wieder Zufalls-Galoiszahlen und zufällige, genauso breite Exponenten und potenzieren Sie beide miteinander mittels *schneller Potenzierung* (wird im Kurs genauer erläutert). - Messen Sie wieder die Zeit.

Hinweis: Bitte benutzbare Kontrollstrukturen zum Testen bereitstellen!