

De wiskunde en toepassing van de cryptologie

Honours Class

TU/e

4 Januari 2010

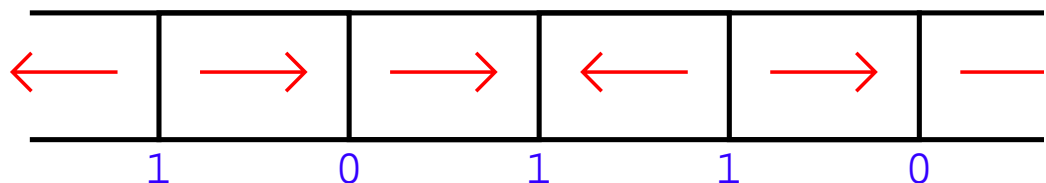
Henk C.A. van Tilborg

1 Beschermen van digitale gegevens.

Bijna alle informatie (muziek, video, foto's, documenten, bestanden) wordt tegenwoordig digitaal weergegeven.

Voorbeeld 1: Magnetische opslag, zoals op floppies en op de harde schijf van een computer.

De sporen op de floppies of schijf zijn in vakjes verdeeld, waarin het magnetisch veld op twee manieren georiënteerd kan worden: van links naar rechts of andersom. Met een leeskop, waarin een spoeltje zit, kunnen wisselingen in het magnetisch veld geregistreerd worden en aldus wordt die informatie weer uitgelezen.



Figuur 1: Een spoor op een floppy of harde schijf.

Voorbeeld 2: Optische opslag, zoals compact disc, cd-rom, dvd.

Een CD-rom heeft ook sporen, die verdeeld zijn in stukjes, maar deze kunnen al dan niet weggebrand worden. Zo ontstaan putjes en plateau'tjes die nullen of enen aangeven. Met een laserstraal kan die informatie weer uitgelezen worden.



Figuur 2: Diepteprofiel van een spoor van een audio cd.

Voorbeeld 3: Solid state, zoals memory sticks en flash cards.

Transistors; traditioneel slaat iedere cel 1 bit informatie op.

Ook bij het versturen van informatie (zowel over draden als via draadloze verbindingen) biedt een digitale weergave ongelooflijke mogelijkheden.

Maar er is een complicatie! Met digitaal verstuurd en opgeslagen gegevens kun je gemakkelijk **manipuleren**: je kunt er wat in veranderen, een stuk eruit halen, iets anders eraan toevoegen, ze dupliceren en opnieuw gebruiken, etc.

Welke elementaire garanties wil je uit veiligheidsoverwegingen aan digitale gegevens opleggen?

- 1) **Geheimhouding**: iemand die zich toegang verschafft tot jouw gegevens (door de communicatie te onderscheppen of door je computersysteem binnen te dringen) moet die informatie niet kunnen begrijpen.
- 2) **Handtekeningeigenschap**: de ontvanger van een boodschap wil graag een (digitaal) bewijs hebben, dat die boodschap inderdaad van een bepaalde persoon komt. Dit bewijs zou een rechter moeten kunnen overtuigen.
- 3) **Integriteit**: de ontvanger van een boodschap wil graag een (digitaal) bewijs dat er niet met die boodschap geknoeid is.

De oplossing wordt gegeven door de de moderne cryptologie.

2 Cryptografische bouwwerken

We onderscheiden drie niveau's.

2.1 De bouwstenen: cryptografische basisfuncties

De basisfuncties die gebruikt worden om privacy of integriteit te garanderen. Hier gaan we het vandaag uitvoerig over hebben.

2.2 Cryptografische protocollen

Cryptografische protocollen bestaan uit een aantal communicatie stappen tussen verschillende partijen. Ze beogen een concreet (deel-) doel maar . . . met bepaalde veiligheids garanties. Ze maken gebruik van de basisfuncties.

Als simpel voorbeeld wordt straks uitgelegd hoe je de identiteit van iemand/iets kunt controleren.

Een ander voorbeeld is hoe je een geheim nummer over twee personen (Alice en Bob) kunt verdelen zodanig dat:

- 1) elke persoon absoluut niets weet over het geheim,
- 2) als ze samen werken, ze het geheim kunnen reconstrueren.

We werken modulo 10. Stel dat het geheim is: **040120101730**

Je kunt natuurlijk aan Alice **040120** geven en aan Bob **101730**, maar dan is niet aan 1) voldaan.

Alice krijgt een volstrekt willekeurig rijtje getallen van 10 cijfers

```
a = Table[Random[Integer, {0, 9}], {i, 1, 12}]
```

```
{5, 4, 1, 8, 7, 4, 7, 9, 8, 0, 1, 5}
```

Bob krijgt het rijtje van 12 cijfers dat opgeteld bij A's rijtje (modulo 10) het geheim oplevert.

```
s = {0, 4, 0, 1, 2, 0, 1, 0, 1, 7, 3, 0};  
b = Mod[s - a, 10]
```

```
{5, 0, 9, 3, 5, 6, 4, 1, 3, 7, 2, 5}
```

Alice en Bob kunnen het rijtje reconstrueren door de getallen gewoon op te tellen modulo 10.

$\text{Mod}[a + b, 10]$

$\{0, 4, 0, 1, 2, 0, 1, 0, 1, 7, 3, 0\}$

Protocol:	Gegeven een geheim getal s , bekend bij de autoriteit.
Initialisatie	Autoriteit kiest een random a en berekent b zodanig dat $a + b = s$ modulo 10.
Communicatie	Autoriteit zendt a in het geheim naar deelnemer Alice. Autoriteit zendt b in het geheim naar deelnemer Bob.
Reconstructie	Alice en Bob berekenen $a + b$ en vinden zo s terug,

Je kunt nu eigenschappen bewijzen zoals dat Alice alleen, of Bob alleen, niet eens 1 bit van s weet.

2.3 Cryptografische systemen

Bij volledige cryptografische systemen moet je denken aan

elektronisch stemmen

met eigenschappen als volledige anonimiteit, stemmen kunnen niet hergebruikt worden, iedereen kan nagaan of zijn/haar stem in de uitslag meegeteld is en een controle of je mag stemmen.

digitaal geld

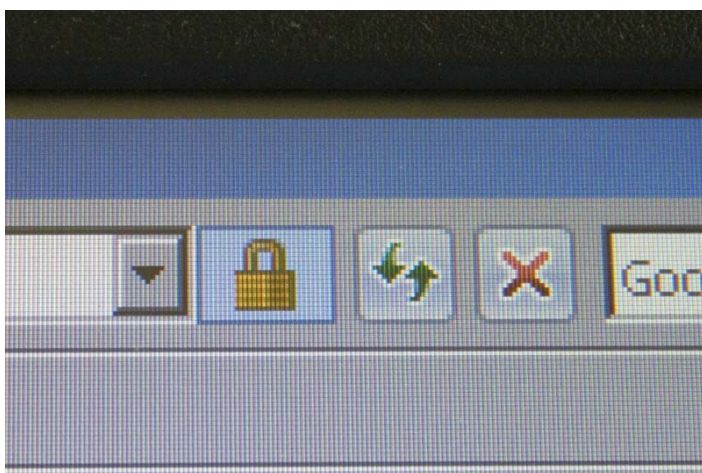
"opnemen" bij je bank, anonimiseren van het "geld", winkel kan checken of de cijferreeks geld representeert en hoeveel, maar je kunt anoniem betalen, winkel kan het verzilveren bij een bank, je kunt het maar één keer uitgeven, maar ook nog allerlei deelproblemen als wisselgeld terugkrijgen, een winkelier die een andere bank heeft, etc.

Merk op dat de eisen vaak tegenstrijdig lijken.

2.4 Dagelijkse toepassingen

"Geheimschriften" worden nu op allerlei plaatsen gebruikt om digitale informatie te beschermen.

2.4.1 Internetbankieren



2.4.2 GSM





3 Fundamenteel verschillende cryptosystemen

3.1 Heeft de "vijand" onbeperkte rekencapaciteit?

Als de vijand beschikt over onbeperkte rekenkracht, dan valt er maar een ding te doen en dat is het **one-time pad** (e.g. het Vernam cijfer) gebruiken.

Voorbeeld:

De "klare" tekst bestaat enkel uit de 26 kleine letters van het alfabet.

De sleutel bestaat uit een even lange rij letters, maar elke letter is willekeurig gekozen .

De optelling gaat paarsgewijs met behulp van de zgn. Vigenère tabel:

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

The Vigenère Table

Voorbeeld:

```

c r y p t o s y m p o s i u m
p f l z c g q d s g u g w k i
r w j o v u i b e v i y e e u

```

Eigenlijk ben je gewoon modulo 26 aan het optellen.

```

klaretekst = "hierkomteeneigenbericht";
L = StringLength[klaretekst]
tekst = ToCharacterCode[klaretekst] - 97

```

```
23
```

```

{7, 8, 4, 17, 10, 14, 12, 19, 4, 4,
 13, 4, 8, 6, 4, 13, 1, 4, 17, 8, 2, 7, 19}

```

```
sleutel = Table[Random[Integer, {0, 25}], {i, 1, L}]
```

```
{11, 1, 5, 21, 23, 14, 5, 7, 3, 25,
 10, 15, 18, 22, 3, 2, 11, 2, 1, 6, 9, 18, 6}
```

```
cijfertekst = Mod[tekst + sleutel, 26]
```

```
{18, 9, 9, 12, 7, 2, 17, 0, 7, 3, 23,
 19, 0, 2, 7, 15, 12, 6, 18, 14, 11, 25, 25}
```

wat overeenkomt met

```
FromCharCode[cijfertekst + 97]
```

```
sjjmhcrahdxtachpmsgsolzz
```

Dit systeem is bewijsbaar veilig. Alle letters worden op een onafhankelijke manier gecijferd en elke uitkomst is even waarschijnlijk.

Anders gezegd: bij elke cijfertekst is elke klaretekst even waarschijnlijk. Bijvoorbeeld

```
altklaretekst = "tueindhovenhonoursclass";
LL = StringLength[altklaretekst]
L == LL
alttekst = ToCharCode[altklaretekst] - 97
```

```
23
```

```
True
```

```
{19, 20, 4, 8, 13, 3, 7, 14, 21, 4, 13,
 7, 14, 13, 14, 20, 17, 18, 2, 11, 0, 18, 18}
```

was ook een mogelijke klaretekst maar nu bij sleutel

```
altsleutel = Mod[cijfertekst - alttekst, 26]
```

```
{25, 15, 5, 4, 20, 25, 10, 12, 12, 25,
 10, 12, 12, 15, 19, 21, 21, 14, 16, 3, 11, 7, 7}
```

Inderdaad geven deze twee samen dezelfde cijfertekst.

```
FromCharCode[Mod[alttekst + altsleutel, 26] + 97]
```

```
sjjmhcrahdxtachpmsgsolzz
```

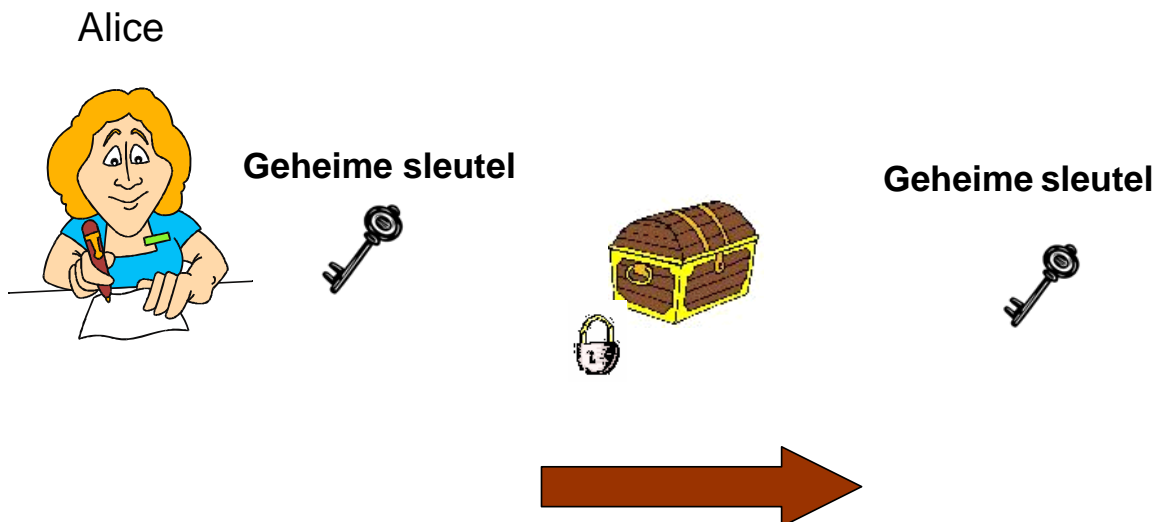
Dit is wel leuk als je gedwongen wordt je sleutel af te geven.

3.2 De vijand heeft "beperkte" rekentijd en geheugenruimte ter beschikking

3.2.1 Symmetrische systemen

Een **symmetrisch** cryptosysteem is als een goed slot waarmee je een kistje kunt afsluiten. Alice en Bob hebben de sleutel van dit slot. Ze kunnen zo'n afgesloten kistje laten staan tot de ander hem aantreft of hem ook versturen met een koerier.

SYMMETRIC KEY CRYPTOGRAPHY



Deze systemen maken gebruik van Shannon's voorstel om te zorgen voor "diffusion" and "confusion".

Je wilt in een tekst de ongelijke symboolfrequenties en onderlinge letterafhankelijkheden weghalen.

De huidige AES standaard Rijndael maakt gebruik van de mogelijkheid dat je bytes ook kunt vermenigvuldigen met elkaar of dat je de inverse van een byte kunt nemen. We maken een excursie.

Rijndael is een blokcijfer en er is wat variatie in mogelijk. Hier laten we het steeds werken op een groepje van 192 bits, ofwel 24 bytes. Deze worden eerst in een 4×6 array geschreven

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

De eerste 8 bits gaan naar $a_{0,0}$, de volgende eronder, etc. Bijvoorbeeld:

$$a_{0,0} = (1, 0, 1, 1, 0, 0, 0, 1).$$

Merk op dat een kolom overeenkomt met een register van 32 bits.

Je gaat nu 12 vercijferingsrondes door. Iedere ronde bestaat uit de volgende vier operaties:

□ **ByteSub**

Dit is het enige niet-lineaire bestanddeel van Rijndael.

Pas op iedere byte $a_{i,j}$ de volgende operatie toe:

Vervang de byte door zijn inverse (behalve de 0-byte, doe daar niets mee).

```
in = {0, 1, 0, 0, 1, 1, 0, 0};
pol = \sum_{i=1}^8 in[[i]] alpha^{i-1}
inver = 1 / pol
```

```
{0, 1, 0, 0, 1, 1, 0, 0}_2
```

```
{0, 1, 0, 0, 1, 0, 0, 1}_2
```

Wat je ook nog doet is het resultaat, zeg de byte (x_0, x_1, \dots, x_7) , te vervangen door

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

Op die manier verandert de 0-byte in een niet-0 byte.

□ ShiftRow

Vervang

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$

door

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$	$a_{0,4}$	$a_{0,5}$
$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	$a_{1,4}$	$a_{1,5}$	$a_{1,0}$
$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$	$a_{2,0}$	$a_{2,1}$
$a_{3,3}$	$a_{3,4}$	$a_{3,5}$	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$

□ MixColumn

Iedere kolom wordt op een vaste manier vervangen door een andere kolom.

Bijvoorbeeld, als een kolom er zo uit ziet

```
kol = {1 + alpha + alpha3 + alpha6 + alpha7,
      alpha + alpha4 + alpha5 + alpha7,
      alpha + alpha2 + alpha3 + alpha7, 1 + alpha + alpha7};
kol // TableForm
```

```
{1, 1, 0, 1, 0, 0, 1, 1}2
{0, 1, 0, 0, 1, 1, 0, 1}2
{0, 1, 1, 1, 0, 0, 0, 1}2
{1, 1, 0, 0, 0, 0, 0, 1}2
```

dan hoort daarbij de veelterm

```
kolompol[x_] = kol[[1]] + kol[[2]] x + kol[[3]] x^2 + kol[[4]] x^3
```

```
x {0, 1, 0, 0, 1, 1, 0, 1}_2 + x^2 {0, 1, 1, 1, 0, 0, 0, 1}_2 +
x^3 {1, 1, 0, 0, 0, 0, 0, 1}_2 + {1, 1, 0, 1, 0, 0, 1, 1}_2
```

Die vermenigvuldigen we dan met

$$g(x) = (1 + \alpha)x^3 + x^2 + x + \alpha$$

```
g[x_] = (one + alpha) x^3 + one x^2 + one x + alpha
```

```
{0, 1, 0, 0, 0, 0, 0, 0}_2 + x {1, 0, 0, 0, 0, 0, 0, 0}_2 +
x^2 {1, 0, 0, 0, 0, 0, 0, 0}_2 + x^3 {1, 1, 0, 0, 0, 0, 0, 0}_2
```

en krijgen

```
pr[x_] = ownexpand[kolompol[x] * g[x]]
```

```
x^4 {0, 0, 0, 0, 0, 0, 1, 1}_2 +
x {0, 0, 1, 0, 1, 1, 0, 1}_2 + x^5 {0, 1, 0, 1, 0, 0, 0, 0}_2 +
x^6 {0, 1, 1, 1, 1, 0, 0, 1}_2 + x^2 {0, 1, 1, 1, 1, 1, 1, 0}_2 +
{1, 0, 1, 1, 0, 0, 0, 1}_2 + x^3 {1, 1, 1, 0, 0, 1, 1, 0}_2
```

Deel door $x^4 - 1$ en neem de rest:

```
prod = PolynomialMod[pr[x], x^4 - 1]
```

```
x^2 {0, 0, 0, 0, 0, 1, 1, 1}_2 + x {0, 1, 1, 1, 1, 1, 0, 1}_2 +
{1, 0, 1, 1, 0, 0, 1, 0}_2 + x^3 {1, 1, 1, 0, 0, 1, 1, 0}_2
```

```
CoefficientList[prod, x] // TableForm
```

```
{1, 0, 1, 1, 0, 0, 1, 0}_2
{0, 1, 1, 1, 1, 1, 0, 1}_2
{0, 0, 0, 0, 0, 1, 1, 1}_2
{1, 1, 1, 0, 0, 1, 1, 0}_2
```

RoundKeyAddition

XOR de hele matrix met een matrix van dezelfde grootte die uit de sleutel vervaardigd is.

$$\begin{array}{|c|c|c|c|c|c|} \hline \mathbf{a}_{0,0} & \mathbf{a}_{0,1} & \mathbf{a}_{0,2} & \mathbf{a}_{0,3} & \mathbf{a}_{0,4} & \mathbf{a}_{0,5} \\ \hline \mathbf{a}_{1,0} & \mathbf{a}_{1,1} & \mathbf{a}_{1,2} & \mathbf{a}_{1,3} & \mathbf{a}_{1,4} & \mathbf{a}_{1,5} \\ \hline \mathbf{a}_{2,0} & \mathbf{a}_{2,1} & \mathbf{a}_{2,2} & \mathbf{a}_{2,3} & \mathbf{a}_{2,4} & \mathbf{a}_{2,5} \\ \hline \mathbf{a}_{3,0} & \mathbf{a}_{3,1} & \mathbf{a}_{3,2} & \mathbf{a}_{3,3} & \mathbf{a}_{3,4} & \mathbf{a}_{3,5} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|c|c|} \hline \mathbf{k}_{0,0} & \mathbf{k}_{0,1} & \mathbf{k}_{0,2} & \mathbf{k}_{0,3} & \mathbf{k}_{0,4} & \mathbf{k}_{0,5} \\ \hline \mathbf{k}_{1,0} & \mathbf{k}_{1,1} & \mathbf{k}_{1,2} & \mathbf{k}_{1,3} & \mathbf{k}_{1,4} & \mathbf{k}_{1,5} \\ \hline \mathbf{k}_{2,0} & \mathbf{k}_{2,1} & \mathbf{k}_{2,2} & \mathbf{k}_{2,3} & \mathbf{k}_{2,4} & \mathbf{k}_{2,5} \\ \hline \mathbf{k}_{3,0} & \mathbf{k}_{3,1} & \mathbf{k}_{3,2} & \mathbf{k}_{3,3} & \mathbf{k}_{3,4} & \mathbf{k}_{3,5} \\ \hline \end{array}$$

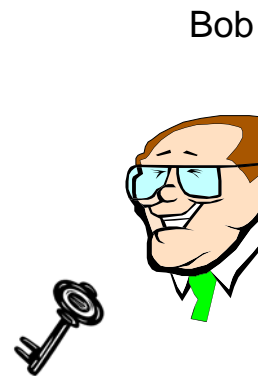
$$= \begin{array}{|c|c|c|c|c|c|} \hline \mathbf{u}_{0,0} & \mathbf{u}_{0,1} & \mathbf{u}_{0,2} & \mathbf{u}_{0,3} & \mathbf{u}_{0,4} & \mathbf{u}_{0,5} \\ \hline \mathbf{u}_{1,0} & \mathbf{u}_{1,1} & \mathbf{u}_{1,2} & \mathbf{u}_{1,3} & \mathbf{u}_{1,4} & \mathbf{u}_{1,5} \\ \hline \mathbf{u}_{2,0} & \mathbf{u}_{2,1} & \mathbf{u}_{2,2} & \mathbf{u}_{2,3} & \mathbf{u}_{2,4} & \mathbf{u}_{2,5} \\ \hline \mathbf{u}_{3,0} & \mathbf{u}_{3,1} & \mathbf{u}_{3,2} & \mathbf{u}_{3,3} & \mathbf{u}_{3,4} & \mathbf{u}_{3,5} \\ \hline \end{array} .$$

met $u_{0,0} = a_{0,0} \oplus k_{0,0}$, de coördinaatsgewijze optelling

$$\mathbf{a}_{0,0} = \{1, 1, 1, 1, 0, 0, 0, 0\}; \mathbf{k}_{0,0} = \{1, 1, 0, 0, 1, 0, 1, 0\}; \\
 \text{Mod}[\mathbf{a}_{0,0} + \mathbf{k}_{0,0}, 2]$$

$$\{0, 0, 1, 1, 1, 0, 1, 0\}$$

3.2.2 Asymmetrische systemen (public key cryptography)

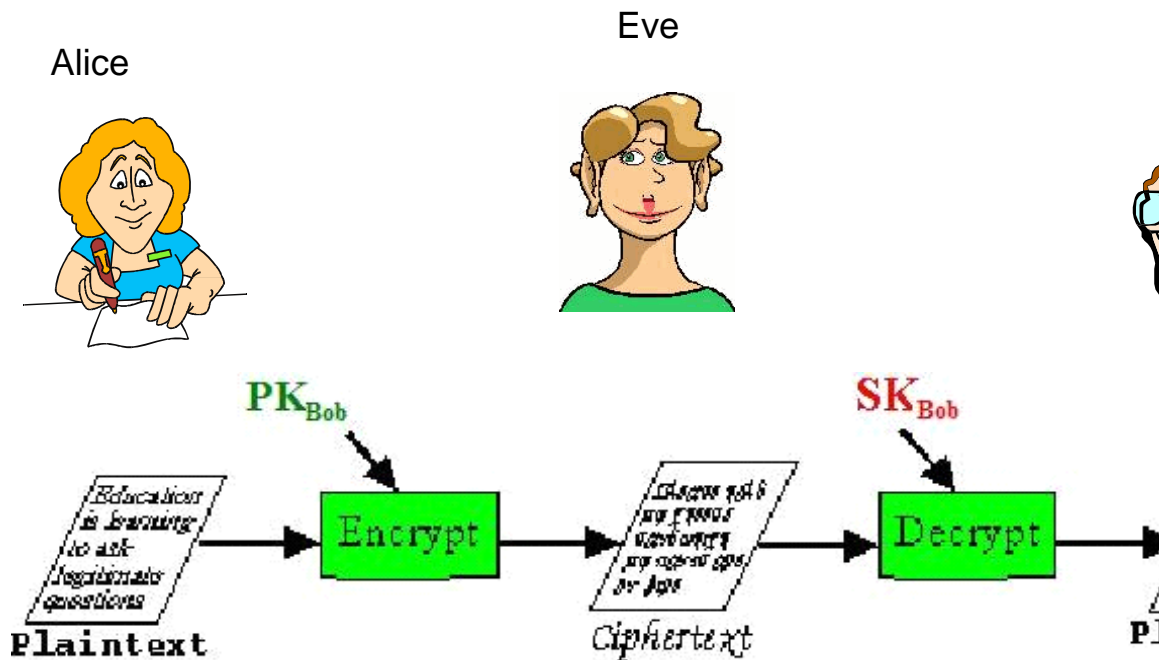


Een **asymmetrisch** cryptosysteem, dat gebruikt wordt voor **geheimhouding**, is als een groot aantal identieke sloten die Bob laat maken en verspreidt. Alleen hij heeft een sleutel. Als Alice een boodschap voor Bob heeft, stopt ze die in een kistje, hangt Bob's slot eraan en doet het dan op slot. Alleen Bob kan het openmaken.



Een **asymmetrisch** cryptosysteem, dat gebruikt wordt voor het zetten van een **handtekening**, is als een kistje van Alice met een uniek slot van haar waarmee ze het afsluit. Iedereen die maar wil krijgt een sleutel, maar alleen Alice weet het goede slot erop te hangen.

PUBLIC KEY CRYPTOGRAPHY



□ Een wiskundige aanname

Deze systemen maken gebruik van bepaalde wiskundige aannames. De meest bekende is de moeilijkheid om getallen te ontbinden in priemfactoren. (Een **priemgetal** is een getal dat niet te schrijven is als product van twee kleinere getallen.)

Het vermenigvuldigen van twee grote getallen is heel gemakkelijk.
 Uit de uitkomst weer die getallen terugvinden is erg moeilijk.

- 15 = 3 × 5
- 91 = ? × ?
- 667 = ? × ?
- 9167 = ? × ?
- ⋮

FactorInteger [50 514 360 227]

{{224 737, 1}, {224 771, 1}}

De rekentijd die hiervoor nodig is verloopt globaal als volgt: $e^{1.923 \sqrt[3]{\ln n} \sqrt[3]{(\ln \ln n)^2}}$

```
Table[{k, " ", " ",
  N[Exp[1.923 *  $\sqrt[3]{\text{Log}[10^k]}$  *  $\sqrt[3]{(\text{Log}[\text{Log}[10^k]])^2}$ ], 3]],
  {k, 100, 300, 50}] //
TableForm
```

100	6.78626×10^{15}
150	1.02919×10^{19}
200	4.04682×10^{21}
250	6.86786×10^{23}
300	6.49464×10^{25}

Het groeit harder dan elke veelterm, maar groeit niet exponentieel.

Het factorizeren van een groot getal is moeilijk en voor een heel grote modulus ondoenlijk.

□ Het RSA systeem

Dit systeem dateert van 1978 en is genoemd naar de drie uitvinders: Rivest, Shamir en Adleman.

Vorbereidingen

Deze voorbereidingen moeten door alle deelnemers uitgevoerd worden. Hier laten we alleen zien wat Bob doet.

Bob kiest twee verschillende priemgetallen, zeg p_B en q_B . De andere deelnemers kiezen ook hun eigen priemgetallen. We gebruiken de *Prime* functie, die het zoveelste priemgetal oplevert.

```
Prime[3]
```

```
5
```

```
pB = Prime[9591]
```

```
qB = Prime[9592]
```

99 989

99 991

Bob rekent ook het product $n_B = p_B q_B$ uit.

$n_B = p_B * q_B$

9 998 000 099

Bob kiest een random vercijferingsexponent e_B uit. (Deze mag echter geen factor gemeen hebben met $(p_B - 1)(q_B - 1)$.)

$e_B = 11\,111\,357$

11 111 357

Bob berekent een bijbehorende ontcijferexponent d_B . Deze moet voldoen aan

$$e_B \times d_B \equiv 1 \pmod{(p_B - 1)(q_B - 1)}.$$

We schrijven $\phi(n_B) = (p_B - 1)(q_B - 1)$.

Omdat Bob de twee priemgetallen p_B en q_B kent, kan hij gemakkelijk de decryptie exponent d_B bepalen.

De functie `PowerMod[g, e, m]` berekent $g^e \pmod{m}$ heel efficiënt.

Hier moeten we $(e_B)^{-1}$ uitrekenen modulo $(p_B - 1)(q_B - 1)$.

$d_B = \text{PowerMod}[e_B, -1, (p_B - 1)(q_B - 1)]$

42 643 373

Bob maakt nu: $n_B = 9\,998\,000\,099$ en $e_B = 11\,111\,357$ algemeen bekend, maar houdt $d_B = 42\,643\,373$ geheim (ook de priemgetallen $p_B = 99\,989$ en $q_B = 99\,991$ mogen niet bekend worden).

Vercijfering

Stel nu dat een andere persoon, zeg Alice, een boodschap naar Bob wil sturen. Voor het gemak is de boodschap het getal **1122334455**. In zijn algemeenheid zal de echte boodschap eerst vertaald moeten worden in een getal, bijv. door middel van de ASCII-code. Dit getal moet wel kleiner zijn dan n_B , want je rekent modulo n_B .

Alice zoekt de openbare waarden van Bob op: $e_B = 11\,111\,357$ en $n_B = 9\,998\,000\,099$.
Vervolgens berekent ze

$$1\,122\,334\,455^{11\,111\,357} \pmod{9\,998\,000\,099}$$

(ze verheft dus de boodschap tot de openbare vercijferingsexponent van Bob, maar modulo zijn n) en stuurt het antwoord naar Bob.

```
cijfertekst = PowerMod[1 122 334 455, 11 111 357, nB]
```

```
4 935 662 754
```

Ontcijfering

Bob kent zijn geheime ontcijferingsexponent $d_B=42643373$, en kan dus het volgende uitrekenen:

$$4\,935\,662\,754^{42\,643\,373} \pmod{9\,998\,000\,099}.$$

```
PowerMod[cijfertekst, 42 643 373, nB]
```

```
1 122 334 455
```

Dit is inderdaad de originele boodschap en deze methode werkt altijd.

Dit is het gevolg van een stelling van Euler. We schrijven c voor de cijfertekst en m voor de plaintext.

$$c^{d_B} \equiv (m^{e_B})^{d_B} \equiv m^{e_B d_B} \stackrel{\text{def. } d_B}{\equiv} m^{1+l \cdot \phi(n_B)} \equiv m \cdot (m^{\phi(n_B)})^l \stackrel{\text{Euler}}{\equiv} m \cdot 1 \equiv m \pmod{n_B}.$$

Afluisteraar

De afluisteraar (eavesdropper in het Engels), die Eva heet, onderschept deze waarde 4935662754 . Zij kent ook de openbare waarden van Bob ($e_B = 11\,111\,357$ en $n_B = 9\,998\,000\,099$) en weet dus dat

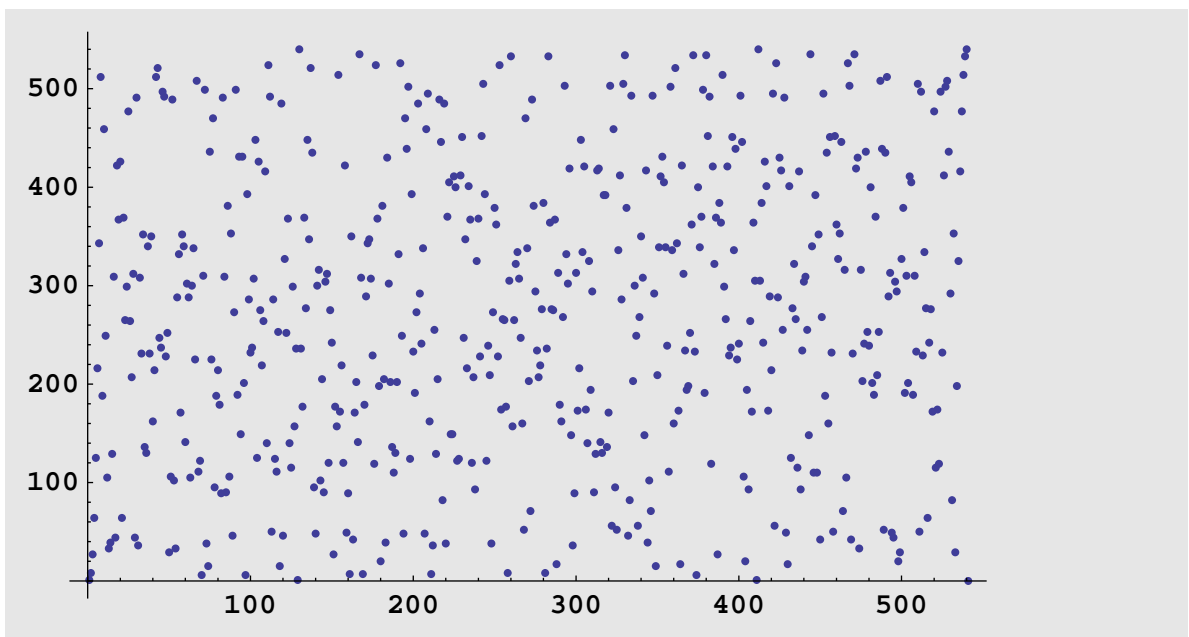
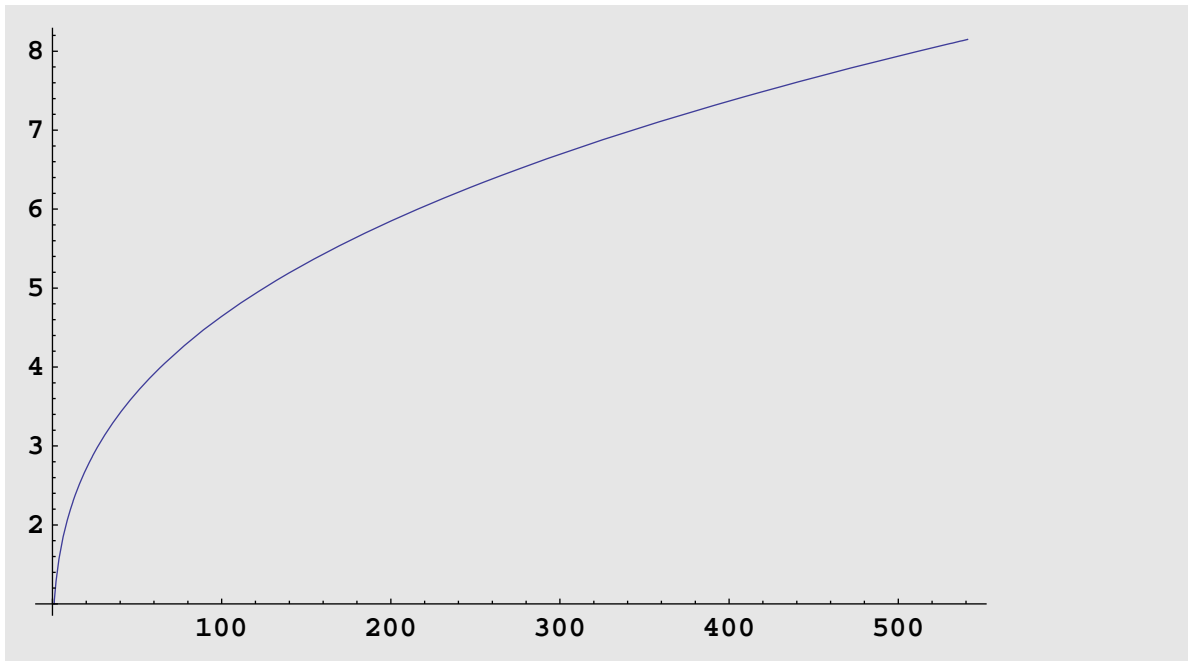
$$\text{Boodschap}^{11\,111\,357} \equiv 4\,935\,662\,754 \pmod{9\,998\,000\,099}.$$

Er zit nu voor haar niet veel meer op dan voor de boodschap 1, 2, 3, ..., 9 998 000 098 uit te proberen. Dit is niet leuk, maar zeker ondoenlijk als n_B en de boodschap 200 tot 500 cijfers lang is.

```

n = 541;
Plot[x1/3, {x, 1, n}]
ListPlot[Table[{j, Mod[j3, n]}, {j, 1, n}]]

```



Als Eva de priemfactoren p_B en q_B van n_B kende, kon ze ook d_B uit e_B berekenen, net zoals Bob dat zelf gedaan heeft, namelijk uit de vergelijking $e_B \times d_B \equiv 1 \pmod{(p_B - 1)(q_B - 1)}$. Echter, Eva kent p_B en q_B niet en kan ze ook niet vinden!

De cryptanalist kan zelf willekeurig veel paren maken van klaretekst met bijbehorende cijfertekst.

3.2.3 Een vergelijking

	symmetrisch cryptosysteem	asymmetrisch cryptosysteem
	<i>2 mensen delen dezelfde geheime sleutel</i>	<i>iedere persoon heeft een geheime en een publieke sleutel</i>
beschermt	tegen derden	ook tegen elkaar
snelheid	snel	langzaam
gevaar / probleem	onderling dispuut	man - in - the - middle aanval
oplossing	juridisch	PKI
technieken	confusion and diffusion	wiskundige aannamen
sleutellengte	klein	groot
nadeel	veilig kanaal nodig	wiskundige aannamen
oplossing	public key cryptography	lange geschiedenis; verder onderzoek

```
Table[ {k, " ", N[2^k, 3], " ",
  N[Exp[1.923 *  $\sqrt[3]{\text{Log}[2^k]}$  *  $\sqrt[3]{(\text{Log}[\text{Log}[2^k]])^2}$ ], 3],
  " ", N[( $\sqrt{2}$ )^k, 3] },
  {k, 50, 1450, 100} ] //
TableForm
```

50	1.13×10^{15}	2.14353×10^6	3.36×10^7
150	1.43×10^{45}	8.53679×10^{10}	3.78×10^{22}
250	1.81×10^{75}	7.83871×10^{13}	4.25×10^{37}
350	2.29×10^{105}	1.63135×10^{16}	4.79×10^{52}
450	2.91×10^{135}	1.45562×10^{18}	5.39×10^{67}
550	3.69×10^{165}	7.40167×10^{19}	6.07×10^{82}
650	4.67×10^{195}	2.50582×10^{21}	6.84×10^{97}
750	5.92×10^{225}	6.21563×10^{22}	7.70×10^{112}
850	7.51×10^{255}	1.20397×10^{24}	8.66×10^{127}
950	9.52×10^{285}	1.90471×10^{25}	9.76×10^{142}
1050	1.21×10^{316}	2.5436×10^{26}	1.10×10^{158}
1150	1.53×10^{346}	2.94×10^{27}	1.24×10^{173}
1250	1.94×10^{376}	2.99921×10^{28}	1.39×10^{188}
1350	2.46×10^{406}	2.74276×10^{29}	1.57×10^{203}
1450	3.12×10^{436}	2.2771×10^{30}	1.77×10^{218}

lengte

symmetrisch

asymmetrisch

elliptische krommen

4 Twee identificatie protocollen gebaseerd op hetzelfde verschil

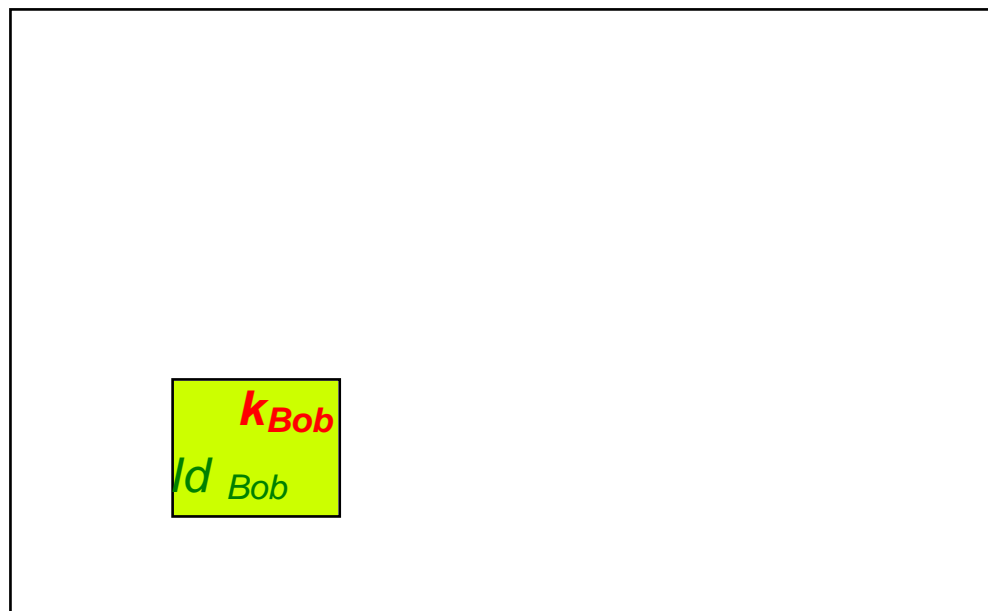
Dezelfde twee manieren van aanpak van het vorige hoofdstuk worden nu gebruikt om de identiteit van iemand te controleren.

We kunnen nu al zeggen dat dezelfde voor- en nadelen van de twee manieren van aanpak hier ook gelden.

4.1 Een identiteitscontrole gebaseerd op een symmetrisch systeem

Als een smartcard in een kaartlezer gestopt wordt, bijvoorbeeld bij een betaling met de Chippas, gaan de kaart en de kaartlezer eerst van elkaar controleren of ze wel door een officiële instantie, zeg de eigen bank, uitgegeven zijn (en dus geen vervalsingen zijn). Op elke kaart en in elke kaartlezer zit het blokcijfer **BC** (dit is nu nog DES).

Als de bank een kaart aanmaakt voor klant Bob, berekent zij een unieke, bij Bob's identiteit Id_{Bob} behorende geheime sleutel k_{Bob} en slaat dat op een ontoegankelijke plaats op de kaart op. De identiteit van Bob is geen geheim.



De bank gebruikt voor de berekening van de geheime sleutel k_{Bob} van Bob het blokcijfer **BC**. Het blokcijfer **BC** heeft als sleutel een supergeheime waarde **MK** (master key). In het identiteitsnummer Id_{Bob} kunnen naam en rekeningnummer verwerkt zijn. De berekening is simpelweg:

$$k_{Bob} = BC_{MK}(Id_{Bob}).$$

De meestersleutel **MK** is ook aanwezig in de kaartlezer, maar dan extra zwaar beschermd. Als de kaart van Bob in de kaartlezer gestopt wordt, leest de lezer eerst het identiteitsnummer Id_{Bob} uit van Bob. Nu kan de kaartlezer ook k_{Bob} uitrekenen. Hij gebruikt hiervoor dezelfde methode.

Controle van de smartcard door de kaartlezer.

De kaartlezer maakt een willekeurig rijtje van 64 nullen en enen. Dit rijtje noemen we r . De kaartlezer presenteert r als "uitdaging" aan de kaart.

De kaart past hierop het blokcijfer toe onder zijn geheime sleutel k_{Bob} . Het resultaat c geeft de kaart aan de kaartlezer als zijn "antwoord" op de uitdaging.

$$c = \text{BC}_{k_{\text{Bob}}}(r)$$

Een andere kaart zal niet in staat zijn het goede antwoord c te geven!

De kaartlezer controleert het antwoord met de k_{Bob} die hij net uitgerekend heeft.

Controle van de kaartlezer door de smartcard.

De kaart maakt zijn eigen willekeurig rijtje van 64 nullen en enen en presenteert dat als zijn "uitdaging" aan de kaartlezer. Alleen een legitiem vervaardigde kaartlezer kan k_{Bob} uitrekenen en dus de uitdaging goed beantwoorden, omdat alleen een officieel geïnstalleerde kaartlezer MK kent en de sleutel k_{Bob} van Bob kan bepalen uit diens identiteitsnummer Id_{Bob} .

4.2 Een identiteitscontrole gebaseerd op een asymmetrisch systeem

Het maakt gebruik van een andere wiskundige aanname:

Het nemen van een modulaire logaritme is moeilijk en voor een zeer grote priemgetal modulus ondoenlijk.

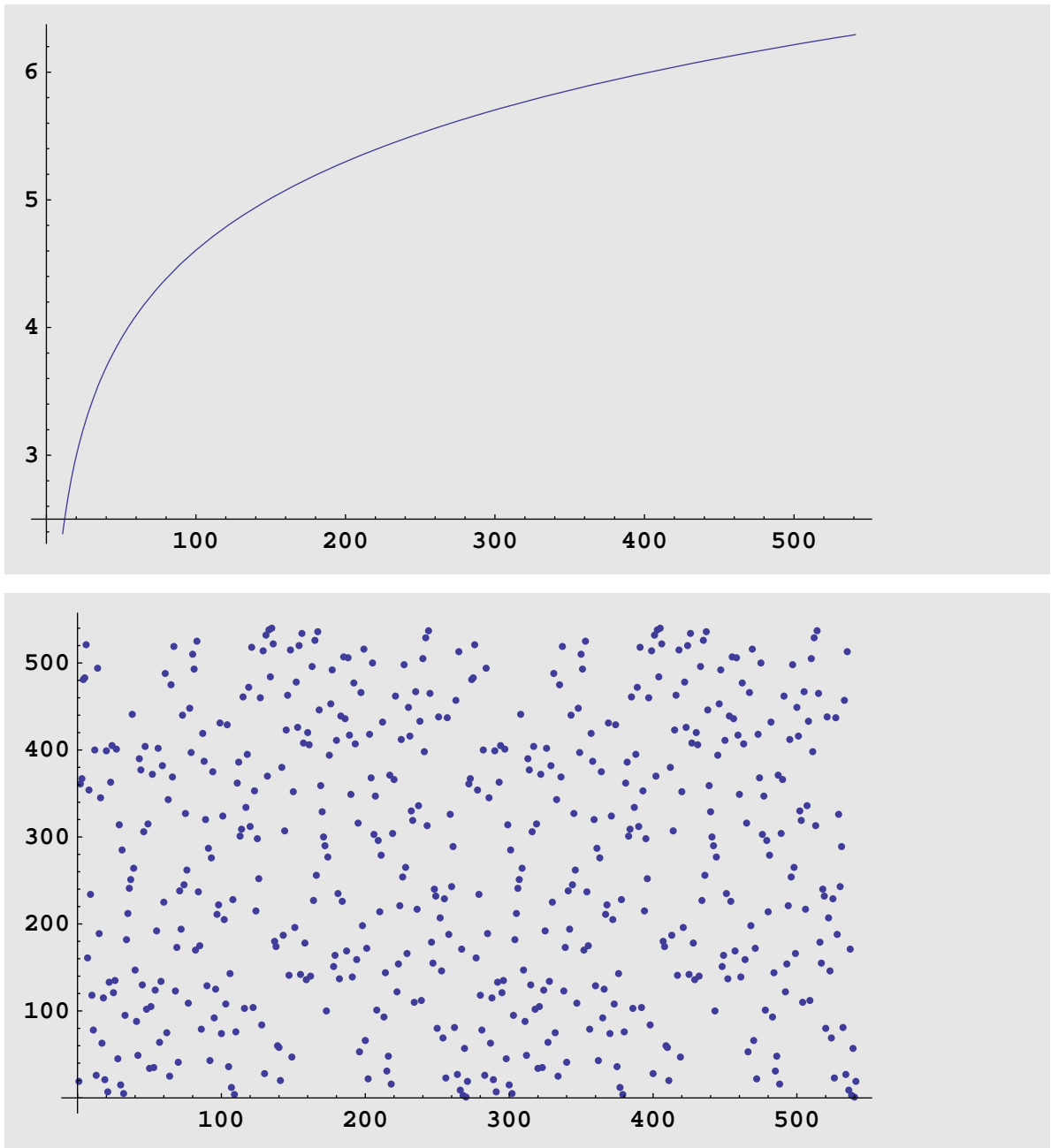
Bepaal e zodat

$$6^e \equiv 23\,456 \pmod{99991}.$$

Merk op dat e is $\log_6 23\,456$.

Je kunt niet heel veel meer doen dan $e = 1, 2, 3, \dots$ uitproberen.

```
p = 541; g = 19;
Plot[Log[x], {x, 1, p}]
ListPlot[Table[{j, Mod[g^j, p]}, {j, 1, p}]]
```



Het grote probleem is steeds dat je niet weet hoe vaak 99 991 van het antwoord is afgetrokken voordat je op 23456 kwam. Zonder de modulus zijn er geen problemen.

Het Schnorr Identificatie Protocol

Deelnemer A (Alice) heeft een geheime exponent S_{Alice} gekozen, de bijbehorende $P_{\text{Alice}} \equiv 6^{S_{\text{Alice}}} \pmod{99\,991}$ uitgerekend en deze als openbare sleutel bekend gemaakt.

(Bij een autoriteit die vertrouwd wordt, is de combinatie van persoonsgegevens I_{Alice} van Alice en bijbehorende P_{Alice} gedeponereerd. Iedereen kan dat daar nazoeken.)

Nu komt Alice bij een andere persoon (we noemen die de Verifier V) en die wil controleren dat de persoon P (we noemen die de Prover), die Alice beweert te zijn en I_{Alice} presenteert,

inderdaad over de geheime waarde S_{Alice} beschikt die van Alice is. Iedereen kan dit namelijk zeggen.

Uiteraard wil Alice (P) niet de waarde van S_{Alice} geven. Zij wil die vaker gebruiken en wil ook niet dat de Verifier na afloop zich elders als Alice kan voordoen. Een "valse" Prover kan niet de waarde van S_{Alice} geven.

P gaat een cryptografisch protocol doorlopen om de Verifier ervan te overtuigen dat zij S_{Alice} kent.

Voorbeeld:

$p = 99\,991$, $g = 6$, $P_{\text{Alice}} = 56\,230$

```
p = 99 991; g = 6;
SAlice = 13 531;
PAlice = PowerMod[6, 13 531, 99 991]
```

56 230

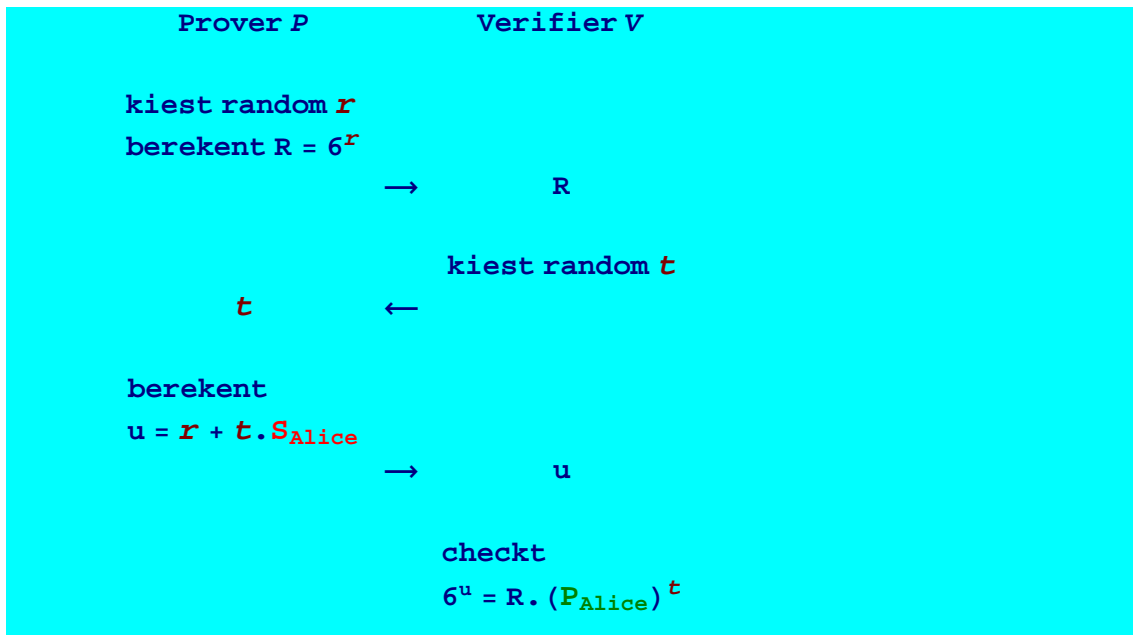
De Verifier weet dat $P_{\text{Alice}} = 56\,230$ de openbare waarde van de echte Alice is.

Protocol:

- 1) De prover P kiest een random exponent r , $0 \leq r < 9991$, berekent $R = 6^r$ en geeft deze waarde (als "getuige" van de random exponent r) aan V .
- 2) De verifier V kiest een random getal t , $0 \leq t < p$, en geeft deze als "uitdaging" aan P .
- 3) Prover P reageert door $u \equiv r + t \cdot S_{\text{Alice}}$ (modulo p) uit te rekenen en de uitkomst aan V te geven.
- 4) Verifier V checkt dat $6^u = R \cdot (P_{\text{Alice}})^t$.

Inderdaad hoort te gelden dat $6^u \equiv 6^{r+t \cdot S_{\text{Alice}}} \equiv 6^r \cdot (6^{S_{\text{Alice}}})^t \equiv R \cdot (P_{\text{Alice}})^t$ (modulo p).





```

r = Random[Integer, 99 990];
Print["random getal r van Prover is ", r]
R = PowerMod[6, r, 99 991];
Print["getuige R van Prover is ", R]
t = Random[Integer, 99 990];
Print["uitdaging t van Verifier is ", t]
u = Mod[r + t * SAlice, 99 990];
Print["reactie u van P is ", u]
{PowerMod[6, u, 99 991],
 Mod[R * PowerMod[PAlice, t, 99 991], 99 991]}

```

```

random getal r van Prover is 1807
getuige R van Prover is 82629
uitdaging t van Verifier is 81892
reactie u van P is 93269

```

```
{48 931, 48 931}
```

De prover P kan alleen maar goed op de uitdaging reageren als hij de goede waarde van S_{Alice} kent. Als hij die niet kent, zal zijn reactie alleen maar met kans $1/p$ geaccepteerd worden. Het priemgetal is echter heel groot en dus is $1/p$ heel klein.

Merk op dat in de relatie $u \equiv r + t \cdot S_{\text{Alice}}$ (modulo p) alleen de waarden van u en t aan V bekend zijn. Met andere woorden, de random waarde van r zorgt ervoor dat geen informatie over S_{Alice} aan V doorlekt.

Als de prover niet Alice is maar weet welke waarde de Verifier t gaat geven, kan hij/zij aan

het protocol voldoen, door een willekeurige u te nemen en aan de Verifier als waarde voor R het getal $6^u / (P_{\text{Alice}})^t$ te geven.

5 Digitale handtekeningen

Met een symmetrisch cryptosysteem kun je nooit een unieke handtekening maken. Er zijn immers twee personen die dezelfde geheime sleutel hebben!

Je kunt wel symmetrische systemen gebruiken om je te beschermen tegen anderen.

5.1 Een digitale handtekening (gebaseerd op een asymmetrisch systeem)

□ Het RSA systeem

We gebruiken dezelfde parameters van Bob als eerder:

Bob maakt bekend: $n_B = 9\,998\,000\,099$ en $e_B = 11\,111\,357$

Bob houdt geheim: $d_B = 42\,643\,373$ (ook de priemgetallen $p_B = 99\,989$ en $q_B = 99\,991$).

Handtekening

Stel dat Bob een handtekening wil toevoegen aan een boodschap. Voor het gemak is de boodschap het getal **1234512345**. Hij berekent gewoon

$$1234512345^{42643373} \pmod{9998000099}.$$

```
PowerMod [1 234 512 345, 42 643 373, nB]
```

```
5 615 829 608
```

met zijn geheime d_B . Hij verstuurt vervolgens het antwoord **5 615 829 608**. Alleen Bob kan dit antwoord uitrekenen!

Verificatie

Alice (of iemand anders) wil de handtekening controleren en zoekt de openbare waarden van Bob op: $e_B = 11\,111\,357$ en $n_B = 9\,998\,000\,099$. Vervolgens berekent ze

$$5\,615\,829\,608^{11\,111\,357} \pmod{9998000099}$$

(ze verheft dus de handtekening tot de openbare vercijferingsexponent van Bob, maar

modulo zijn n) en vindt inderdaad **1 234 512 345**.

```
PowerMod [5 615 829 608, 11 111 357, nB]
```

```
1 234 512 345
```

□ **Een subtiel probleem**

Hier is nog een subtiel probleem. Alice kan een random getal opschrijven

```
zomaar = Random [Integer, nB]
```

```
4 862 717 197
```

Zij verheft dit tot de macht $e_B = 11\,111\,357$ modulo $n_B = 9\,998\,000\,099$

```
nepboodschap = PowerMod [zomaar, 11 111 357, nB]
```

```
7 835 299 255
```

en dan claimen dat "**zomaar**" de handtekening is voor de boodschap "**nepboodschap**".
Inderdaad geldt dat

$$\text{nepboodschap}^{e_B} \equiv \text{zomaar} \pmod{n_B}$$

en dus geldt

$$\text{zomaar}^{d_B} \equiv \text{nepboodschap} \pmod{n_B}$$

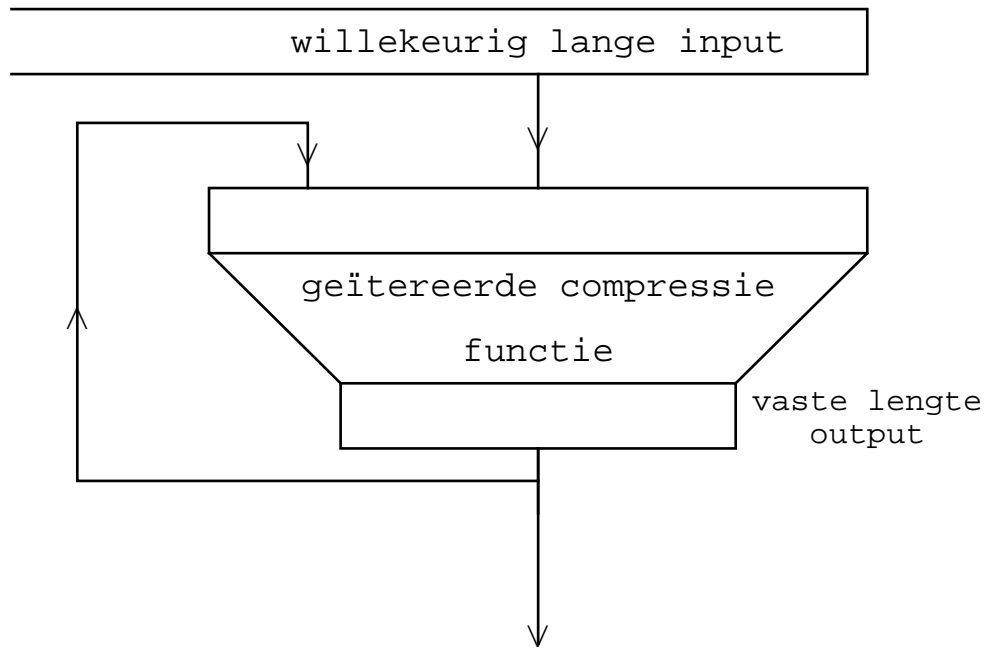
maar de kans is groot dat "**quasi**" geen betekenisvolle boodschap zal zijn.

6 Cryptografische hash functies

Heel vaak zijn boodschappen/files veel te lang om er een digitale handtekening overheen te zetten. Wat je graag doet is ze comprimeren met een **cryptografische hash functie** en dan een digitale handtekening zetten over de hashwaarde. De hash bestaat uit bijv. **256 bits**.

Je werkt nu niet met een sleutel. Iedereen kan de hashwaarde uitrekenen en controleren.

Als je wel met sleutels werkt, noem je deze functies: Message Authentication Codes.



Je kunt bijv. de openbare sleutels van je RSA systeem bij een Certification Authority laten voorzien van een digitale handtekening, zodat ieder ander met de openbare sleutel van de CA kan controleren dat hij inderdaad jouw openbare sleutels heeft. Hiermee kun je de "man-in-the middle attack" voorkomen. Omdat die openbare sleutel zelf al lang is, zet je alleen een handtekening over een hash waarde ervan in combinatie met de identiteit van de eigenaar.

Het moet natuurlijk niet zo zijn dat iemand een tweede boodschap (*andere openbare sleutels*) kan vinden met dezelfde hashwaarde, want dan is het geen controle mechanisme meer op de integriteit van de boodschap!

Verschillende niveaus van sterkte:

- pre-image resistance:** vind bij een gegeven hashwaarde een boodschap m die deze hash waarde oplevert,
- weak collision resistance:** gegeven een boodschap m vind een andere boodschap m' met dezelfde hash waarde.
- strong collision resistance:** vind twee boodschappen m en m' met dezelfde hash waarde.

(Hier zou je eigenlijk steeds het woord "zinnvolle" aan willen toevoegen.)

MD5 and SHA1 beslaan samen bijna de hele markt. Andere namen zijn HAVAL en RIPEMD. Je hebt ook hashfuncties die op blokcijfers zijn gebaseerd, maar daar kleven nadelen aan.

Collisions (botsingen) voor MD5 (128 bits) kunnen sinds een paar jaar met niet te veel moeite gevonden worden. *Zelfs betekenisvolle botsingen!*

Voor SHA-1 (160 bits) zijn nog geen botsingen gevonden, maar dezelfde methodes werken! Beide systemen hebben nog steeds de "pre-image resistance" eigenschappen dat is het enige van belang in de praktijk. Echter iedereen heeft altijd het "strong collision resistant" zijn gezien als teken dat een hashfunctie echt robuust is.

7 Twee problemen

7.1 De man-in-the-middle aanval

Een probleem van toepassingen van public key cryptografie over internet is de **man-in-the-middle** aanval:

alle communicatie tussen Alice en Bob wordt onderschept door Eve.

Alice communiceert in feite met Eve

Eve communiceert met Bob

Deze aanval is mogelijk als je geen garantie hebt dat de openbare sleutel van Bob echt bij Bob hoort. Eve maakt je wijs dat haar openbare sleutel van Bob is.

7.2 Key escrow

Een andere zorg is de toegankelijkheid van bestanden als een werknemer deze beveiligd heeft en er gebeurt iets met die persoon (vergeet sleutel, overlijdt, is kwaadwillend etc.). Hoe kun je als bedrijf nog bij zijn bestanden?

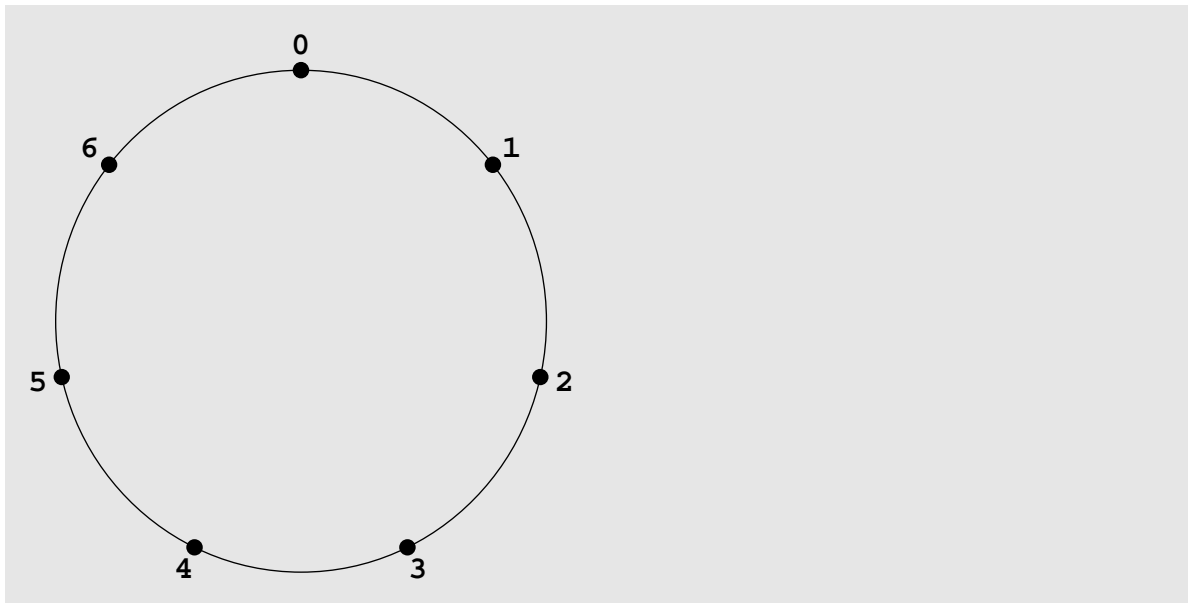
7.3 Een oplossing: PKI

De oplossing voor beide problemen is een goed opgezette **Public Key Infrastructure** (PKI)

Deze regelt allerlei praktische zaken, zoals sleutel generatie (als dat nodig is), het toekennen van rechten, het certificeren van sleutels, maar ook key escrow en het zonnodig herroepen van gecertificeerde sleutels.

8 Modulair rekenen (klokrekenen)

Modulo 7 rekenen betekent dat je alleen met de getallen 0, 1, 2, 3, 4, 5, 6 werkt.



Zo gauw een antwoord hier buiten valt, trek je er net zo vaak 7 van af (of telt het er bij op) totdat je weer in 0, 1, 2, 3, 4, 5, 6 zit. Dus

$$6+5 \equiv 11 \equiv 4 \pmod{7}.$$

$$6 \times 5 \equiv 30 \equiv 2 \pmod{7}.$$

$$6^5 \equiv 7776 \equiv 6 \pmod{7}.$$

We schrijven \equiv in plaats van $=$ en voegen "mod 7" toe om niet te vergeten dat het niet om een echte gelijkheid gaat maar alleen om een gelijkheid modulo 7

In feite deel je het antwoord door 7 en schrijf je de rest op.

In *Mathematica* kan dit met de Mod functie.

```
Mod [ 6 + 5 , 7 ]
```

```
Mod [ 6 * 5 , 7 ]
```

```
Mod [ 6^5 , 7 ]
```

4

2

6

Merk op dat

Mod [3 * 5, 7]

1

$$3 \times 5 \equiv 1 \pmod{7}$$

We schrijven rustig $3 \equiv \frac{1}{5} \pmod{7}$.

Anders gezegd: delen door 5 is hetzelfde als vermenigvuldigen met 3.

□ Modulo 2

We rekenen heel vaak modulo 2 (de \oplus of **XOR** voor EE):

$$0 + 0 = 0, \quad 0 + 1 = 1 + 0 = 1, \quad 1 + 1 = 0.$$

Modulo 2 zijn optellen en aftrekken hetzelfde.

$$0 \times 0 = 0 \times 1 = 1 \times 0 = 0 \quad \text{en} \quad 1 \times 1 = 1.$$

Vermenigvuldigen modulo 2 is niet erg interessant.

9 Rekenen met bytes

Met bytes kun je net zo goed optellen, aftrekken, vermenigvuldigen en delen als met de reële getallen en de complexe getallen.

Bytes bestaan uit groepjes van 8 bits:

$$a = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7).$$

Met de bits werken we modulo 2: $0 + 0 = 0, \quad 0 + 1 = 1 + 0 = 1, \quad 1 + 1 = 0.$

Optellen

Als $b = (b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7)$ dan is

$$\underline{a} + \underline{b} = (a_0 + b_0, a_1 + b_1, a_2 + b_2, a_3 + b_3, a_4 + b_4, a_5 + b_5, a_6 + b_6, a_7 + b_7) \pmod{2}.$$

Bijvoorbeeld:

$$\mathbf{a} = \{1, 0, 0, 1, 1, 1, 1, 1\}; \mathbf{b} = \{0, 0, 0, 0, 1, 1, 0, 1\};$$

$$\text{Mod}[\mathbf{a} + \mathbf{b}, 2]$$

$$\{1, 0, 0, 1, 0, 0, 1, 0\}$$

Modulo 2 is optellen en aftrekken hetzelfde.

□ **Vermenigvuldigen**

Maar hoe kun je nu een byte vermenigvuldigen met een andere byte of er door delen. Het antwoord moet weer een byte zijn!

Heel graag heb je dat de vergelijking $A.X = B$ precies één oplossing heeft, behalve natuurlijk als $A = (0, 0, 0, 0, 0, 0, 0, 0)$.

We gebruiken een truc (eigenlijk de prachtige theorie van Galois lichamen).

We maken van de bytes **veeltermen** (in de variable α):

$$\underline{a} = (a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \iff$$

$$A(\alpha) = a_0 + a_1 \alpha + a_2 \alpha^2 + a_3 \alpha^3 + a_4 \alpha^4 + a_5 \alpha^5 + a_6 \alpha^6 + a_7 \alpha^7 = \sum_{i=0}^7 a_i \alpha^i.$$

$$\mathbf{A} = \sum_{i=1}^8 \mathbf{a}[[i]] \alpha^{i-1}$$

$$1 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^6 + \alpha^7$$

Als je echter twee veeltermen van graad 7 vermenigvuldigt kun je iets van graad 14 krijgen en dat correspondeert natuurlijk niet meer met een byte.

$$B = \sum_{i=1}^8 b[[i]] \alpha^{i-1}$$

$$\text{PolynomialMod}[A * B, \{2\}]$$

$$\alpha^4 + \alpha^5 + \alpha^7$$

$$\alpha^4 + \alpha^5 + \alpha^{10} + \alpha^{11} + \alpha^{13} + \alpha^{14}$$

Om weer op een veelterm van graad 7 uit te komen delen we de uitkomst van net door $m(\alpha) = 1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8$. De veelterm $m(\alpha)$ is onontbindbaar. Er geldt

$$\alpha^4 + \alpha^5 + \alpha^{10} + \alpha^{11} + \alpha^{13} + \alpha^{14} = (\alpha^6 + \alpha^5 + \alpha^3 + 1)(1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8) + (1 + \alpha + \alpha^4 + \alpha^6)$$

$$P = \text{PolynomialMod}[A * B, \{1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8, 2\}]$$

$$1 + \alpha + \alpha^4 + \alpha^6$$

En dat geeft dus de byte **(1, 1, 0, 0, 1, 0, 1, 0)**.

De eenheid bij de vermenigvuldiging is de byte (1, 0, 0, 0, 0, 0, 0, 0), horend bij de veelterm 1.

□ Delen

Hoe kun je delen?

Als je a een byte is en je wilt $1/a$ uitrekenen, dan zoek je de byte u met

$$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) (u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7) = (1, 0, 0, 0, 0, 0, 0, 0).$$

Je zoekt dus de veelterm $U(\alpha)$ zodat

$$A(\alpha)U(\alpha) \equiv 1 \pmod{1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8}.$$

Het antwoord is $U(\alpha) = \alpha + \alpha^2 + \alpha^3 + \alpha^6$. Inderdaad geldt

$$U = \alpha^2 + \alpha^3 + \alpha^4 + \alpha^7;$$

$$\text{PolynomialMod}[A * U, 2]$$

$$P = \text{PolynomialMod}[A * U, \{1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8, 2\}]$$

$$\alpha^2 + \alpha^3 + \alpha^4 + \alpha^5 + \alpha^8 + \alpha^9 + \alpha^{10} + \alpha^{12} + \alpha^{13} + \alpha^{14}$$

1

Dus de inverse van byte a is byte u :

```
CoefficientList[U,  $\alpha$ ]
```

```
{0, 0, 1, 1, 1, 0, 0, 1}
```

Hoe je die moet vinden is weer een ander verhaal, maar met de goede theorie gaat het heel gemakkelijk. We herhalen dat $m(\alpha) = 1 + \alpha + \alpha^3 + \alpha^4 + \alpha^8$.

```
<< FiniteFields`
```

GF::shdw :

Symbol GF appears in multiple contexts {FiniteFields`, Global`}; definitions in context FiniteFields` may shadow or be shadowed by other definitions. >>

```
f256 = GF[2, {1, 1, 0, 1, 1, 0, 0, 0, 1}];
one = f256[{1, 0, 0, 0, 0, 0, 0, 0, 0}]
alpha = f256[{0, 1, 0, 0, 0, 0, 0, 0, 0}]
```

```
{1, 0, 0, 0, 0, 0, 0, 0, 0}_2
```

```
{0, 1, 0, 0, 0, 0, 0, 0, 0}_2
```

```
a = {1, 0, 0, 1, 1, 1, 1, 1};
```

$$AA = \sum_{i=1}^8 a[[i]] \alpha^{i-1};$$

```
inver = 1 / AA
```

```
{0, 0, 1, 1, 1, 0, 0, 1}_2
```

