RSA en Diffie-Hellman

RSA en Diffie-Hellman

1. Inleiding

Wat je verstuurt en ontvangt via **Wifi** in de trein en cafés kan door iedereen **afgeluisterd en gemanipuleerd** worden.

https zorgt voor o.a.

- **autenticiteit** van de webpagina
- **encryptie** tegen afluisteren van wachtwoorden etc.

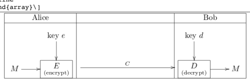
Onderdelen:

- **RSA (Rabobank, ING, ABN Amro, ...)**
- **Diffie-Hellman** met de groep $(\mathbf{Z}/p\mathbf{Z})^*$
- Diffie-Hellman met een **elliptische kromme (Google, WhatsApp, Facebook, ...)**

2. Symmetrische versus asymmetrische encryptie

```
%hide
%auto
latex.add_package_to_preamble_if_available("xypic")
```

```
%hide
%auto
%latex
\[\begin{array}[b]{|c|c|c|}
\hline
\text{Alice} & & \text{Bob}\\
\hline
%\rule[-7ex]{0pt}{0pt}
\xymatrix{
 & \text{key $e$}
%\ \text{(key)}
 \ar[d] \\
M
%\ \text{(message)}
 \ar[r]  & *+[F]{{\displaystyle{E} \atop  \text{(encrypt)}}}
 } &
%$\xymatrix{ \ar[r]^{C} & }$ &
\!\!\!\!\!\!\!\xymatrix{
 %\ \text{(key)}
% \phantom{
{{{{{ \atop } \atop } \atop } \atop } \atop { \atop }}
%}
\\
%\ \text{(message)}
    \ar[rrrr]^{C} &  & & &
    }
    \!\!\!\!\!\!\!
    &
\xymatrix{
\text{key $d$}
%\ \text{(key)}
 \ar[d] \\
%\ \text{(message)}
    *+[F]{{\displaystyle{D} \atop  \text{(decrypt)}}}\ar[r] & M
  }\\
\hline
\end{array}\]
```



2.1 Symmetrisch:

één sleutel **d = e = k** (private key)

Voorbeeld:

- zij $\sigma = (ABCDEFGHIJKLMNOPQRSTUVWXYZ) \in S_{26}$
- **Caesar**-encryptie is σ^k.
- Bijvoorbeeld σ^3: DELFT $\mapsto$ GHOIW
- decryptie is dan σ^{-k}: GHOIW $\mapsto$ DELFT
- eenvoudig te kraken

Voorbeeld:

- **AES** (Advanced Encryption Standard, aka Rijndael)
- wordt overal gebruikt (o.a. https)
- snel en veilig

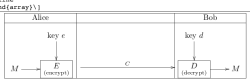
Nadeel: je moet wel **eerst samen een sleutel afspreken!**

Oplossingen:

- asymmetrische encryptie
- sleuteluitwisseling

(herhaling van het plaatje)


```
%hide
%auto
%latex
\[\begin{array}[b]{|c|c|c|}
\hline
\text{Alice} & & \text{Bob}\\
\hline
%\rule[-7ex]{0pt}{0pt}
\xymatrix{
 & \text{key $e$}
%\ \text{(key)}
 \ar[d] \\
M
%\ \text{(message)}
 \ar[r]  & *+[F]{{\displaystyle{E} \atop  \text{(encrypt)}}}
 } &
%$\xymatrix{ \ar[r]^{C} & }$ &
\!\!\!\!\!\!\!\xymatrix{
 %\ \text{(key)}
% \phantom{
{{{{{ \atop } \atop } \atop } \atop } \atop { \atop }}
%}
\\
%\ \text{(message)}
    \ar[rrrr]^{C} &  & & &
    }
    \!\!\!\!\!\!\!
    &
\xymatrix{
\text{key $d$}
%\ \text{(key)}
 \ar[d] \\
%\ \text{(message)}
    *+[F]{{\displaystyle{D} \atop  \text{(decrypt)}}}\ar[r] & M
  }\\
\hline
\end{array}\]
```

2.2 Asymmetrisch:

De encryptiesleutel **e is publiek bekend** (public key)

Paradoxaal:

- de afbeelding "encryptie met e" heeft een inverse,
- maar die **inverse is computationeel moeilijk** voor wie d niet kent.

3. Snel versus langzaam

Met de schoolmethoden kan je snel

- optellen
- vermenigvuldigen
- delen met rest

Allemaal in een hoeveelheid tijd die polynomiaal is in het aantal cijfers en haalbaar voor duizenden cijfers.

```
a = randint(10^100000,10^100001)
b = randint(10^100000,10^100001)
timeit("c = a+b; d = a*b; q = a/b; r = a%b")
```

```
5 loops, best of 3: 131 ms per loop
```

Dus: optellen en vermenigvuldigen in $\mathbf{Z}/n\mathbf{Z}$ is snel zelfs als n duizenden cijfers heeft.

Hoe zit het met **exponentiatie** in $\mathbf{Z}/n\mathbf{Z}$?

Laat $a = \overline{51} \in \mathbf{Z}/101\mathbf{Z}$. Wat is a^{73}?

Het is per definitie $a^{73} = aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa$, dus 72 vermenigvuldigingen (terwijl 73 maar 2 cijfers heeft).

```
a = Zmod(101)(51)
a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a*a
```

```
40
```

Wat als je a^k wilt uitrekenen met $k \approx 10^{1000}$?

Eerst nogmaals a^{73}.

```
print 73.binary()
print 2^6 + 2^3 + 2^0
```

```
1001001
73
```

$a^{73} = a^{2^6+2^3+2^0} = a^{2^6} \cdot a^{2^3} \cdot a^{2^0}$

Merk op: $a^{2^0} = a$ en $a^{2^{n+1}} = a^{2^n \cdot 2} = (a^{2^n})^2$.

```
l = [a]
for i in range(6):
    l.append(l[-1]^2)
l[6]*l[3]*l[0]
```

```
40
```

Dus het aantal vermenigvuldigingen om zo a^k uit te rekenen is lineair in het aantal (binaire) cijfers van k.

```
a = Zmod(101123456)(51)
time prod([a for i in range(7312333)])
```

```
19282755
Time: CPU 1.61 s, Wall: 8.39 s
```

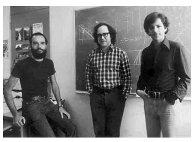
```
time power_mod(51,7312333,101123456)
```

```
19282755
Time: CPU 0.00 s, Wall: 0.00 s
```

```
time
power_mod(519837943817310938739874139874318973124987231498234179829184,1013873894714239873428972341987123183190784398739184723148939749831734987329184734219873421987423149823479843219873871439874198734,19837319048734938173241897234198743218947191893749132897438973142987328973219832749812347893481379834798231749382417349874231891234798873198473)
```

```
13455770051990752342540135992527333675217335255574440511051472134713\
94292313963007032906498773209768865846487338013966803335497902184425\
1139199
Time: CPU 0.00 s, Wall: 0.02 s
```

4. RSA: Rivest, Shamir, Adleman



Voor p en q ongelijke priemgetallen, laat $N = pq$. Dan geldt $\varphi(N) = \varphi(p)\varphi(q) = (p-1)(q-1)$.

Stelling:

Zij N als boven en zijn e en d gehele getallen met $ed \equiv 1 \bmod \varphi(N)$, dan geldt voor alle $a \in \mathbf{Z}$:

$$(a^e)^d \equiv a \bmod N.$$

Bewijs:

We bewijzen $(a^e)^d \equiv a \bmod p$.

Modulo q gaat hetzelfde, en samen geeft dit $N \mid (a^e)^d - a$.

Als $p \mid a$, dan triviaal. Zo niet, dan:

$ed = k(p-1)(q-1) + 1$ voor een $k \in \mathbf{Z}$.

Dus $(a^e)^d = a^{ed} = (a^{p-1})^{k(q-1)} \cdot a \equiv 1^{k(q-1)} \cdot a \equiv a \bmod p$ wegens Euler/Lagrange. QED

```
# klein voorbeeld:
p = 5; q = 11
N = p*q # 55
e = 13; f = 37
# controle:
phi = (p-1)*(q-1); print phi
print e*f
```

```
40
481
```

```
# encryptie van het getal 3
# 3^e modulo N
3^13 % 55
```

```
38
```

```
# decryptie
# 38^f modulo N
38^37 % 55
```

```
3
```

Voorbeeld inclusief initialisatie:

```
# Grote priemgetallen zijn gemakkelijk te maken.
# (disclaimer: in de praktijk moet er iets meer gebeuren)
p = random_prime(10^400, proof=False)
q = random_prime(10^400, proof=False)
```

```
# De publieke sleutel:
N = p*q
e = 2^16+1 # dit is echt het getal dat je bank gebruikt!
(N, e)
```

```
(29753725955410067038090183940111764985457774854409186570068518582326\
9934101786604135569501280462354127914255879229783403264001108088675414\
8682404569275441077498480025330041292583450245540757573707149645510 3\
8943471883877687894512668817931663036539727372547718417547606427473\
7911575003541373166609128098239954750214746275019408372096695186732 3\
8155154353058348886942852268139894944683337832776349736685937121401\
2540847864531769332600327673610295882769761078738001704087325088503 7\
7075864712335800727231830352746173325165436134559576478182435287\
8929320583527836892674764139742986663733253285283334188586668585040\\
8561531474609464167177207534360263509127505650736215156681039875 95\
4979639973226644637279291922615091293369593684143590223816822977309 5\
0448403620124743646202481254938921001569046739405 3, 65537)
```

```
# Het maken van een geheime sleutel:
phi = (p-1)*(q-1)
D, x, y = xgcd(e, phi) # algoritme van Euclides!
print D == 1
print x * e + y * phi == 1
```

```
True
True
```

```
# Nu is x de inverse van e modulo phi, dus de geheime sleutel is:
f = x
```

```
# Schrijf je bericht als getal:
# a = 01, b = 02, c = 03, d = 04, e = 05, ..., spatie = 00.
a = texttonum("het wachtwoord is eesam open u")
a
```

```
80520002301030820231515180400091900190519011300151605140021
```

```
# versleutelen:
b = power_mod(a, e, N) # a^e modulo N op een slimme manier berekend
b
```

```
376287514632639382044718167928427318917863947304151125972088663482087\
6429908741100867247863648634971175649833351035666961788562091920100572\
6021488222680381792351739440362217495550780432729039601567949429 9\
3711303881329440311411607046309062876948639578642769384986324807321 5\
6440522587980748997492040335978102133000971906044170226945280925160 7\
7819922886139877738816658500960184114343511818409324827325492942907 9\
9718913297557360075276920784050096618895604752223472736994775512651\
7491257773944264477104864008648851677875063950251862149130004772338\\
1347619756457051393344938818277704157764169296143270365887133128248 7\
1742835535782868465871287922045472577674354409756171736601118526 92\\
5776369785360917605707439394425718833284411393386989590205783968\\
5207189854600012397614848295062797526613015786282
```

```
# ontsleutelen:
c = power_mod(b, f, N) # b^f modulo N, dat zou (a^e)^f, dus a moeten zijn
print c
print numtotext(c)
```

```
80520002301030820231515180400091900190519011300151605140021
het wachtwoord is eesam open u
```

```
factor(65537-1)
```

```
2^16
```

5. Diffie-Hellman

Doel: een sleutel k afspreken om symmetrische encryptie mee te doen.

Zij G een groep en $g \in G$.

1. Alice kiest een willekeurig geheim getal a en maakt g^a openbaar.
2. Bob kiest een willekeurig geheim getal b en maakt g^b openbaar.
3. Alice rekent $(g^b)^a$ uit en noemt dit k.
4. Bob rekent $(g^a)^b$ uit en noemt dit k.

Merk op: $(g^b)^a = g^{ab} = (g^a)^b$.

Hier is a een discrete logaritme van g^a met basis g.

Als je discrete logaritmen kan uitrekenen in G, dan kan je de sleutel kraken.

In $G = (\mathbf{Z}/p\mathbf{Z})^*$ voor goed gekozen priemgetallen p van ruim 600 cijfers is het discretelogaritmeproblem moeilijk met de huidige technieken.

6. Elliptische krommen, en verder.

Cryptografie met $(\mathbf{Z}/n\mathbf{Z})^*$ (zeals RSA en Diffie-Hellman met $(\mathbf{Z}/p\mathbf{Z})^*$) heeft zwakke plekken die komen van geavanceerde wiskunde met (uitbreidingen van) de ring $\mathbf{Z}$ (zie "Number Field Sieve"). Om toch veilig te zijn, worden getallen van ruim 600 cijfers gebruikt.

Elliptische krommen over $(\mathbf{Z}/p\mathbf{Z})$ zijn groepen waarbij dergelijke zwakke plekken niet bekend zijn. Daarom zijn die al veilig bij maar 80 cijfers en dus veel sneller.

Een praktische kwantumcomputer van enkele duizenden qubits zou al deze asymmetrische algoritmen kraken. Voor het geval zoiets mogelijk wordt, wordt gekeken naar alternatieven, zeals cryptografie op basis van roosters.

Alles op basis van algebra.