Cryptography

4 – Public-key encryption: RSA

G. Chênevert

October 7, 2019

ISEN
ALL IS DIGITAL!
LILLE
yncréa

Today

Principle

Cryptanalysis

Key generation

In practice

Asymmetric cryptography

Two different keys are used: one for encryption, one for decryption

if knowledge about one gives no information about the other

$\Longrightarrow$ one of them can be made public

Public-key encryption

The encryption key k_e is made public (k_d kept private)

anyone can write to Bob, but only he can read

As implemented by *e.g.* PGP/GPG

Famous "asymmetric" problems

- factorization of large integers

  $\Longrightarrow$ RSA

- discrete logarithm problem (DLP)

  $\Longrightarrow$ Diffie-Hellman, ElGamal, DSA

- DLP over an elliptic curve

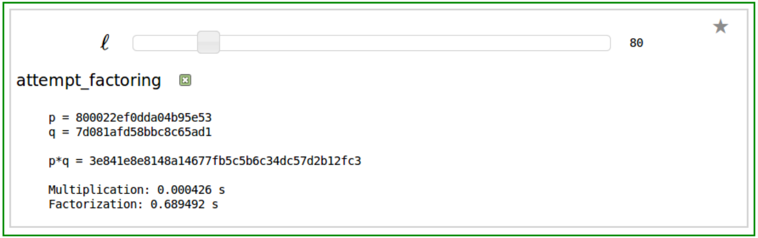
  $\Longrightarrow$ elliptic curve cryptography (ECC): ECDH, ECDSA, ...

- shortest vector problem

  $\Longrightarrow$ lattice-based cryptography ...

For two ℓ-bit factors

Factorization is asymptotically *much* slower than multiplication



Try it for yourself

Modular arithmetic

Recall (?)

Definition

We say that $a \underset{n}{\equiv} b$ when n divides $b - a$, *i.e.* $b = a + kn$ for some integer k

i.e. a and b are equal, up to ("modulo") a multiple of n

Remarks:

- $a \underset{n}{\equiv} b$ if and only if $a \,\%\, n = b \,\%\, n$
- If $a \underset{n}{\equiv} b$ and $c \underset{n}{\equiv} d$, then $(a + c) \underset{n}{\equiv} (b + d)$ and $(ac) \underset{n}{\equiv} (bd)$

Rivest-Shamir-Adleman (1977)

Fix some (large) integer n.

$\mathcal{M} = \mathcal{C} = \mathbb{Z}/n\mathbb{Z}$, identified with $[\![0, n[\![$

$$\begin{cases} E(e, m) :\underset{n}{\equiv} m^e \\ D(d,\ c) :\underset{n}{\equiv} c^d \end{cases}$$

based on **modular exponentiation**

Easy enough!

ℓ 20

```
n = cdf45
m = 34d78
e = b8ee6
m**e mod n = 16b88
```

Or is it? (try a larger ℓ)

Modular exponentiation

Naive algorithm to compute $m^e \,\%\, n$:

$$
\begin{aligned}
& r = 1 \\
& \text{for } i \text{ in } [\![1, e]\!]: \\
& \quad r = r * m \\
& \text{return } r \,\%\, n
\end{aligned}
$$

Problems:

- intermediate result r gets LARGE
- takes e iterations

Modular exponentiation (again)

Better algorithm to compute $m^e \,\%\, n$:

$$
\begin{aligned}
&r = 1\\
&\text{for } i \text{ in } [\![1, e]\!]:\\
&\quad r = (r * m) \,\%\, n\\
&\text{return } r
\end{aligned}
$$

But:

- still takes e modular multiplications ...

Fast exponentiation, v.1 (R to L)

Write $e = b_\ell \cdots b_0$ in base 2, so that $m^e \underset{n}{\equiv} m^{b_0}(m^2)^{b_1}(m^4)^{b_2} \cdots (m^{2^\ell})^{b_\ell}$.

$$
\begin{aligned}
&r = 1, q = m \\
&\text{for } i \text{ in } [\![0, \ell]\!]: \\
&\quad \text{if } b_i = 1: \\
&\quad\quad r = (r * q) \,\%\, n \\
&\quad q = q^2 \,\%\, n \\
&\text{return } r
\end{aligned}
$$

at most $2(\ell + 1)$ modular multiplications!

Example (v.1)

Let's compute 33^{29} modulo 227.

With $m = 33$, $n = 227$ and $e = 29 = 11101$:

i		0	1	2	3	4
b		1	0	1	1	1
q		33	181	73	108	87
r	1	33	33	139	30	113

so $33^{29} \underset{227}{\equiv} 113$ (indeed).

Fast exponentiation, v.2 (L to R)

Can get rid of the running variable q by writing

$$m^e \underset{n}{\equiv} \left(\cdots\left(\left(m^{b_\ell}\right)^2 m^{b_{\ell-1}}\right)^2 m^{b_{\ell-2}} \cdots m^{b_1}\right)^2 m^{b_0}.$$

$r = 1$
for i in $[\![0, \ell]\!]$:
    $r = r^2 \,\%\, n$
    if $b_{\ell-i} = 1$:
        $r = (r * m) \,\%\, n$
return r

In both cases: running time in $\mathcal{O}(\log_2 e)$

Example (v.2)

With the same values as before:

i		0	1	2	3	4
b		1	1	1	0	1
r	1	33	71	189	82	113

which is coherent with previous results (but uses half the memory).

Ok: that's fast

ℓ 33

```
n = cda20ec6
m = 89d3c768
e = a2f8ff3c
m**e mod n = 86df516
```

Indeed!

The RSA cipher (again)

$$\begin{cases} E(e, m) \underset{n}{\equiv} m^e \\ D(d, c) \underset{n}{\equiv} c^{\,d} \end{cases}$$

Correct decryption:

Why should there exist such exponents such that

$$m^{de} \underset{n}{\equiv} m \qquad \forall_m \quad ??$$

Chinese Remainder Theorem

If n can be written as a product of coprime factors

$$n = n_1 \cdots n_k,$$

then there is an *isomorphism of rings*

$$\mathbb{Z}/n\mathbb{Z} \cong \mathbb{Z}/n_1\mathbb{Z} \times \cdots \times \mathbb{Z}/n_k\mathbb{Z}.$$

- $(\rightarrow)$ take remainders
- $(\leftarrow)$ use Bézout's relation

Example

$$\mathbb{Z}/12\mathbb{Z} \cong \mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$$

	0	1	2	3
0	0	9	6	3
1	4	1	10	7
2	8	5	2	11

Euler's φ function

Consider the number $\varphi(n)$ of integers in $[\![1, n]\!]$ that are coprime with n.

Theorem (Fermat)

For all x coprime with n,

$$x^{\varphi(n)} \underset{n}{\equiv} 1.$$

i.e., modular exponents work modulo $\varphi(n)$: $x^a \underset{n}{\equiv} x^b$ when $a \underset{\varphi(n)}{\equiv} b$.

Almost there

Special case: suppose $n = p_1 \cdots p_k$ is a product of distinct prime factors, so that

$$\varphi(n) = (p_1 - 1) \cdots (p_k - 1).$$

Corollary

In this case, if $f \underset{\varphi(n)}{\equiv} 1$ *then* $x^f \underset{n}{\equiv} x \quad \forall_x$.

Hence: it is sufficient to ask that the RSA exponents satisfy

$$de \underset{\varphi(n)}{\equiv} 1.$$

A small (thus very insecure) working example

```
n   = 74989
phi = 69600
e   = 52027
d   = 10963

d*e mod phi = 1

message:    60211
encryption: 13247
decryption: 60211
```

Try here

Today

Principle

Cryptanalysis

Key generation

In practice

Security of RSA

Public: n, e, c.

The attacker would like to recover m.

- Brute force on m: search for x such that

$$x^e \underset{n}{\equiv} c.$$

$\Longrightarrow$ Impractical if n large

- Better: try to recover the decryption exponent d, then decrypt m like Bob

$$m \underset{n}{\equiv} \sqrt[e]{c} \underset{n}{\equiv} c^d.$$

Trivial is $\varphi(n)$ is known

Given e and $\varphi(n)$, the extended Euclidean algorithm easily solves

$$de \underset{\varphi(n)}{\equiv} 1.$$

But: computing $\varphi(n)$ from n is (assumed to be) hard.

Best known algorithm: **factor** $n = p_1 \cdots p_k$ and use

$$\varphi(n) = (p_1 - 1) \cdots (p_k - 1).$$

Factoring vs. splitting

Factoring n: finding the complete list of prime factors $(p_1, \ldots, p_k)$ for which

$$n = p_1 \cdots p_k.$$

Splitting n: finding *one* prime factor p of n.

Essentially all known factorization algorithms are of the form

$$\begin{array}{l} \text{factors} = [\ ] \\ \text{while } n > 1\text{:} \\ \quad p = \text{split}(n) \\ \quad \text{factors} \mathrel{+}= [\ p\ ] \\ \quad n = n \ /\!/ \ p \end{array}$$

Trial division

The simplest splitting algorithm:

$$
\begin{aligned}
&p = 2\\
&\text{while } p \leq \sqrt{n}\text{:}\\
&\quad \text{if } n \,\%\, p = 0 \text{ return } p\\
&\quad p \mathrel{+}= 1\\
&\text{return } n
\end{aligned}
$$

Quickly finds small ($\leq 2^{64}$) prime factors

$\implies$ smallest prime factor should be as large as possible

$p_i \approx \sqrt[k]{n} \implies$ take $k = 2$! (why not $k = 1$?)

Other factorization algorithms

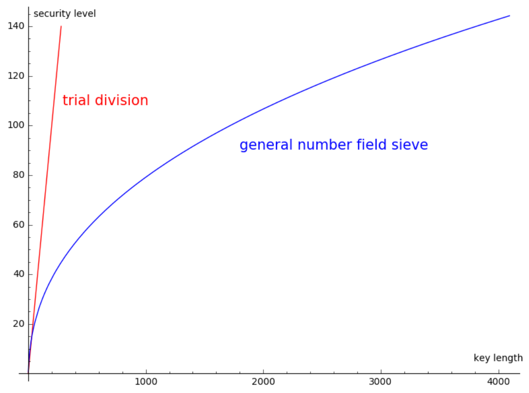
There is a very large litterature devoted to the subject of integer factorization.

As of 2019, the best general purpose algorithm is the *General Number Field Sieve* (GNFS) that factors an ℓ-bit integer in

$$\approx 5.5^{\,\ell^{1/3}(\ln \ell)^{2/3}} \text{ time.}$$

Public factorization record: RSA-728 (2009)

Consequence on key length



According to RSA Security, Inc.

Symmetric key size	Equivalent RSA key size
80	1024
112	2028
128	3072
256	15360

Today

Principle

Cryptanalysis

Key generation

In practice

Key generation

Recovering the decryption key should be hard for the attacker...

...but easy for Alice and Bob!

Ok since they are free to choose the prime factors of n.

Key generation: produces a RSA triple (n, d, e)

Prime factors

To generate an ℓ-bit RSA modulus n:

- generate two random $\ell/2$-bit prime numbers p and q
- set $n := p \cdot q$

To generate a random prime number:

- generate random integers until you get a prime!

(there are some very fast primality tests)

Note: density of prime numbers around x is $\approx \frac{1}{\ln x}$

RSA exponents

- Knowing p and q, compute $\varphi(n) = (p-1)(q-1)$
- Pick e coprime with $\varphi(n)$ (doesn't even need to be chosen randomly)
- Compute d such that

$$de \underset{\varphi(n)}{\equiv} 1$$

  using the extended Euclidean algorithm (XGCD)

Today

Principle

Cryptanalysis

Key generation

In practice

Real-world RSA

The *plain RSA* described above has all sorts of problems:

- malleability: $E(e, m_1) \cdot E(e, m_2) = E(e, m_1 \cdot m_2)$
- lack of randomness
- fixed size of plaintext
- ...

In practice, a suitable padding scheme needs to be used.

$\implies$ use a library!