Cryptology Part III: Asymmetric Ciphers

Klaus Pommerening
Fachbereich Mathematik
der Johannes-Gutenberg-Universität
Saarstraße 21
D-55099 Mainz

November 27, 2000—English version: January 13, 2016
last change: March 5, 2021

Asymmetric encryption introduces a new idea into cryptography that makes a fundamental difference with the formerly treated classic or bitblock ciphers:

> Encryption and decryption are *significantly different* processes. Who knows the encryption function (including its key) has no means to *efficiently* derive the decryption function (or key).

The derivation of the decryption function from the encryption function is a "one-way" process. As an everyday analog think of a postbox: Who can insert letters in it is not able to get something out of the box, except when she has the key. In this situation for every participant there exists a pair of functions. The first part of this pair is the encryption function—represented by a parameter called "public key"—and is publically available and usable by *everyone*. The second part is the decryption function—represented by a "private key"—and is a personal secret shared by *no one else*.

The existence of a strictly personal secret has further interesting applications:

- secure proof of identity ("strong authentication"),
- digital signature.

The latter is simply the reverse application of private and public keys:

- *No one except the owner* can encrypt with his private key.
- *Everyone* can decrypt with the public key, and in this way convince herself without any doubt that *the author and the content* of the message are authentic.

Historical notes:

- Invented by Diffie/Hellman in 1976 (in the public science).
- Best known algorithm: RSA 1978.
- Invented by James Ellis (British secret service CESG) in 1970, declassified 1997.
- Potentially known at NSA in 1965.
  - Potential idea: the "codebook paradigm"—*reversing a function may be difficult.*
  - Potential application: Engineers can encrypt the code for operating nuclear weapons. But only the serving commander can decrypt, and release the bomb.

Chapter 1

The RSA Cipher and its Algorithmic Foundations

The most important—that is, most applied and most analyzed—asymmetric cipher is RSA, named after its inventors Ron Rivest, Adi Shamir, and Len Adleman. It uses elementary number theoretic algorithms, and its supposed security relies on the hardness of factoring large numbers into primes, although breaking RSA might be easier then factoring, see Section 2.2.

The three fundamental arithmetic algorithms for implementing RSA are

- binary power algorithm,
- Euclidean algorithm,
- chinese remainder algorithm,

the last two of them known from Part I of these lectures (in the context of linear ciphers). These algorithms are basic not only for cryptography but more generally for algorithmic algebra and number theory ("computer algebra"). Moreover they are fundamental also for numerical mathematics—for problems that don't require approximate numerical solutions in floating-point numbers but exact solutions in integers, rationals, or symbolic expressions.

1.1 Description of the RSA Cipher

Parameters

The three parameters

- n = **module**,
- e = **public exponent**,
- d = **private exponent**,

are positive integers with

$$(1) \qquad m^{ed} \equiv m \pmod n \qquad \text{for all } m \in [0 \ldots n-1].$$

Naive Description

The first idea is to set

$$M = C = \mathbb{Z}/n\mathbb{Z}, \quad K \subseteq [1 \ldots n-1] \times [1 \ldots n-1].$$

For $k = (e, d)$ we have

$$E_k : M \longrightarrow C, \qquad m \mapsto c = m^e \bmod n,$$

$$D_k : C \longrightarrow M, \qquad c \mapsto m = c^d \bmod n.$$

This description is naive for n is variable, and (necessarily, as we'll see soon) a part of the public key. In particular the sets M and C vary.

More Exact Description

We want to describe RSA in a form that fits the general definition of a cipher. To this end we note that for an l bit number n we have $2^{l-1} \le n < 2^l$, thus fix the parameters:

- l = bit length of the module (= "key length"),
- $l_1 < l$ bit length of plaintext blocks,
- $l_2 \ge l$ bit length of ciphertext blocks.

We construct a block cipher $M \longrightarrow C$ over the alphabet $\Sigma = \mathbb{F}_2$ with

$$M = \mathbb{F}_2^{l_1} \subseteq \mathbb{F}_2^{l_2} = C.$$

The key $k = (n, e, d) \in \mathbb{N}^3$ is chosen with ($2^{l-1} \le n < 2^l$ or equivalently:)

$$\ell(n) := \lfloor \log_2 n \rfloor + 1 = l, \quad 1 \le e \le n-1, \quad 1 \le d \le n-1,$$

such that equation (1) holds. The symbol $\ell(n)$ denotes the number of bits, that is, the length of the binary representation of n.

To encrypt a plaintext block m of length l_1 by E_k we interpret it as the binary representation of an integer. The result c, a non-negative integer $< n$, has a binary representation by l_2 bits—completed with leading zeroes if necessary, or better yet, with random leading bits.

To decipher the ciphertext block c we interpret it as a non-negative integer $c < n$ and transform it into $m = c^d \bmod n$.

Really Exact Description

See PKCS = 'Public Key Cryptography Standard' #1:
`https://tools.ietf.org/html/rfc8017`.

Questions to Address

- How to find suitable parameters n, d, e such that (1) holds?
- How to efficiently implement the procedures for encryption and decryption?
- How to assess the security?

Speed

Note that encryption and decryption are significantly slower than for common symmetric ciphers. (Estimates range up to a factor of roughly 10^4.)

1.2 The Binary Power Algorithm

The procedure for raising powers in a quite efficient way has a natural description in the abstract framework of a multiplicative semigroup H. The task is: Compute the power x^n for $x \in H$ and a positive integer n—the product of n factors x—by *as few multiplications as possible.* The naive direct method,

$$x^n = x \cdot (x \cdots x)\,,$$

involves $n-1$ multiplications. The expense is proportional with n, hence grows *exponentially* with the number $\ell(n)$ of bits (or decimal places) of n. A much better idea is the **binary power algorithm**. In the case of an additively written operation (strictly speaking for the semigroup $H = \mathbb{N}$) it is known also as Russian peasant multiplication, and was known in ancient Egypt as early as 1800 B. C., in ancient India earlier than 200 B. C.

The specification starts from the binary representation of the exponent n,

$$n = b_k 2^k + \cdots + b_0 2^0 \quad \text{with } b_i \in \{0,1\},\ b_k = 1,$$

thus $k = \lfloor \log_2 n \rfloor = \ell(n) - 1$. Then

$$x^n = (x^{2^k})^{b_k} \cdots (x^2)^{b_1} \cdot x^{b_0}.$$

This suggests the following procedure: Compute $x, x^2, x^4, \ldots, x^{2^k}$ in order by squaring k times (and keeping the intermediate results), and then multiply the x^{2^i} that have $b_i = 1$. The number of factors is $\nu(n)$, the number of 1's in the binary representation. In particular $\nu(n) \leq \ell(n)$. This makes a total of $\ell(n) + \nu(n) - 2$ multiplications.

We have shown:

Proposition 1 *Let H be a semigroup. Then for all $x \in H$ and $n \in \mathbb{N}$ we can compute x^n by at most $2 \cdot \lfloor \log_2 n \rfloor$ multiplications.*

This expense is only *linear* in the bit length of n. Of course to assess the complete expense we have to account for the cost of multiplication in the semigroup H.

Here is a description as pseudocode:

```
Procedure BinPot
    Input parameters:
        $x$ = base
            [locally used for storage of the iteratively computed squares]
        $n$ = exponent
    Output parameters:
        $y$ = result $x^n$
            [locally used for accumulation of the partial product]
    Instructions:
        $y := 1$.
        while $n > 0$:
            if $n$ is odd: $y := yx$.
            $x := x^2$.
            $n := \lfloor n/2 \rfloor$.
```

Remarks

1. The algorithm is almost optimal, but not completey. The theory of "addition chains" in number theory yields an asymptotic behaviour of $\log_2 n$ for the average minimum number of multiplications, roughly half the value from Proposition 1.

2. That the numbers of involved multiplications differ depending on the exponent is the starting point of *timing* and *power attacks* invented by Paul KOCHER. Imagine a device, say a smart card, that computes powers with a secret exponent. Then the different timings or power consumptions reveal information about the exponent.

1.3 The Carmichael Function

We assume $n \geq 2$.

The Carmichael function is defined as the exponent of the multiplicative group $\mathbb{M}_n = (\mathbb{Z}/n\mathbb{Z})^\times$:

$$\lambda(n) := \exp(\mathbb{M}_n) = \min\{s \geq 1 \mid a^s \equiv 1 \pmod{n} \quad \text{for all } a \in \mathbb{M}_n\};$$

in other words, $\lambda(n)$ is the maximum of the orders of the elements of $\mathbb{M}_n$.

Remarks

1. Euler's theorem may be expressed as $\lambda(n)|\varphi(n)$ ("exponent divides order"). A common way of expressing it is

   $$a^{\varphi(n)} \equiv 1 \pmod{n} \quad \text{for all } a \in \mathbb{Z} \text{ with } \gcd(a,n) = 1.$$

   Both versions follow immediately from the definition.

2. If p is prime, then $\mathbb{M}_p$ is cyclic—see Proposition 2 below—, hence

   $$\lambda(p) = \varphi(p) = p - 1.$$

By the chinese remainder theorem we have $\mathbb{M}_{mn} \cong \mathbb{M}_m \times \mathbb{M}_n$, hence by Lemma 22 of Appendix A.10:

Corollary 1 *For coprime $m, n \in \mathbb{N}_2$*

$$\lambda(mn) = \operatorname{lcm}(\lambda(m), \lambda(n)).$$

Corollary 2 *If $n = p_1^{e_1} \cdots p_r^{e_r}$ is the prime decomposition of $n \in \mathbb{N}_2$, then*

$$\lambda(n) = \operatorname{lcm}(\lambda(p_1^{e_1}), \ldots, \lambda(p_r^{e_r})).$$

Remarks

3. The Carmichael function for powers of 2 (proof as **exercise** or in Appendix A.1):

   $$\lambda(2) = 1, \quad \lambda(4) = 2, \quad \lambda(2^e) = 2^{e-2} \quad \text{for } e \geq 3.$$

4. The Carmichael function for powers of odd primes (proof as **exercise** or in Appendix A.3):

   $$\lambda(p^e) = \varphi(p^e) = p^{e-1} \cdot (p-1) \quad \text{for } p \text{ prime} \geq 3.$$

To prove the statement in Remark 2 we have to show that the multiplicative group $\bmod p$ is indeed cyclic. We prove a somewhat more general standard result from algebra:

Proposition 2 *Let K be a field and $G \leq K^\times$ be a finite subgroup of order $\#G = n$. Then G is cyclic and consists exactly of the n-th roots of unity in K.*

Proof. For $a \in G$ we have $a^n = 1$, hence G is contained in the set of roots of the polynomial $T^n - 1 \in K[T]$. Thus K has exactly n different n-th roots of unity, and G contains all of them.

Now let m be the exponent of G, in particular $m \leq n$. Lemma 24 of Appendix A.10 yields that all $a \in G$ are even m-th roots of unity. Hence $n \leq m$, so $n = m$, and G has an element of order n. $\diamond$

1.4 Suitable Parameters for RSA

Proposition 3 *Let $n \geq 3$ be an integer. The following statements are equivalent:*

(i) *n is squarefree.*

(ii) *There exists an $r \geq 2$ with $a^r \equiv a \pmod{n}$ for all $a \in \mathbb{Z}$.*

(iii) **[RSA equation]** *For every $d \in \mathbb{N}$ and $e \in \mathbb{N}$ with $de \equiv 1 \pmod{\lambda(n)}$ we have $a^{de} \equiv a \pmod{n}$ for all $a \in \mathbb{Z}$.*

(iv) *For each $k \in \mathbb{N}$ we have $a^{k\cdot\lambda(n)+1} \equiv a \pmod{n}$ for all $a \in \mathbb{Z}$.*

Proof. "(iv) $\Longrightarrow$ (iii)": Since $de \equiv 1 \pmod{\lambda(n)}$, we have $de = k \cdot \lambda(n) + 1$ for some k. Hence $a^{de} \equiv a \pmod{n}$ for all $a \in \mathbb{Z}$.

"(iii) $\Longrightarrow$ (ii)": Since $n \geq 3$, we have $\lambda(n) \geq 2$. Choosing an arbitrary d with $\gcd(d, \lambda(n)) = 1$ and a corresponding e by congruence division mod$\lambda(n)$ we get (ii) with $r = de$.

"(ii) $\Longrightarrow$ (i)": Assume there is a prime p with $p^2|n$. Then by (ii) we have $p^r \equiv p \pmod{p^2}$. But because of $r \geq 2$ we have $p^r \equiv 0 \pmod{p^2}$, contradiction.

"(i) $\Longrightarrow$ (iv)": By the chinese remainder theorem we only have to show that $a^{k\cdot\lambda(n)+1} \equiv a \pmod{p}$ for all prime divisors $p|n$.

Case 1: $p|a$. Then $a \equiv 0 \equiv a^{k\cdot\lambda(n)+1} \pmod{p}$.

Case 2: $p \nmid a$. Because of $p-1|\lambda(n)$, we have $a^{\lambda(n)} \equiv 1 \pmod{p}$, hence $a^{k\cdot\lambda(n)+1} \equiv a \cdot (a^{\lambda(n)})^k \equiv a \pmod{p}$. $\diamond$

Corollary 1 *The RSA procedures work for a module n if and only if n is squarefree.*

To find suitable exponents d and e we have to know $\lambda(n)$ or, better yet (and necessarily as it will turn out) the prime decomposition of n. Then the procedure of key generation suggests itself:

1. Choose different primes $p_1, \ldots, p_r$ and form the module $n := p_1 \cdots p_r$.
2. Compute $\lambda(n) = \operatorname{lcm}(p_1 - 1, \ldots, p_r - 1)$ using the Euclidean algorithm.
3. Choose the public exponent $e \in \mathbb{N}_2$, coprime with $\lambda(n)$.
4. Compute the private exponent d with $de \equiv 1 \pmod{\lambda(n)}$ by congruence division.

Then take the pair (n, e) as public key, and the exponent d as private key.

Corollary 2 *Who knows the prime decomposition of n can compute the private key d from the public key (n, e).*

Practical Considerations

1. The usual choice is $r = 2$. Then the module has only two prime factors p and q that, as a compensation, are very large. Factoring this kind of integers $n = pq$ seems especially hard. It is crucial that the primes are chosen completely at random. Then an attacker has no hint for a guess.

2. For e we may choose a prime with $e \nmid \lambda(n)$, or a "small" integer say $e = 3$—more on the dangers of this choice later. A common standard choice is the prime $e = 2^{16} + 1$, provided $e \nmid \lambda(n)$. The binary representation of this integer contains only two 1's, making the binary power algorithm for enryption very fast. (For digital signature this is the verification of the signature.) However this choice of e doesn't make decryption (or generating a digital signature) more efficient.

3. After generating the keys we don't need p, q, and $\lambda(n)$ anymore, so we could destroy them.

   However: Since d is a "random" integer in the interval $[1 \ldots n]$ taking d-th powers is costly even with the binary power algorithm. It becomes somewhat faster when the owner of the private key computes $c^d \bmod p$ and $\bmod q$—using integers of about half the size—and then composes the result $\bmod n$ with the chinese remainder theorem. This procedure yields a small advantage in speed for decryption (or generating a digital signature).

4. Instead of $\lambda(n)$ we could use its multiple $\varphi(n) = (p-1)(q-1)$ for calculating the exponent.

   Advantage: We save (one) lcm computation.

   Drawback: In general we get a larger exponent d, slowing down each single decryption.

5. Notwithstanding Corollary 1 the RSA procedure works in a certain sense even if the module n is not squarefree. Decryption using the chinese remainder theorem is slightly more complex, involving an additional "Hensel lift." However decryption breaks down for plaintexts a that are multiples of a prime p with $p^2 | n$. Note that this effect is compatible with Corollary 1!

   The danger of hitting a plaintext divided by a multiple prime factor of n by chance is negligeable but grows with the number of prime factors. Even for a squarefree module n a plaintext divided by a prime factor would immediately yield a factorization of n, and hence reveal the private key.

Attention

The cryptanalytic approaches of the following chapter result in a set of side conditions that should be strictly respected when generating RSA keys.

Exercises

1. Let p and q be two different odd primes, and $n = p^2q$. Characterize the plaintexts $a \in \mathbb{Z}/n\mathbb{Z}$ that satisfy the RSA equation $a^{de} \equiv a \pmod{n}$. Generalize the result to arbitrary n.

2. Show that an integer $d \in \mathbb{N}$ is coprime with $\lambda(n)$ if and only if d is coprime with $\varphi(n)$.

Chapter 2

Cryptanalysis of RSA

"Cryptanalysis of RSA" *doesn't break* the cipher—except in a few exceptional situations—but traces out the framework for applying it in a secure way according to our best judgment. In particular it helps avoiding some traps. We want answers to the questions:

- Do there exist sufficiently many keys to evade an exhaustion attack?
- Which mathematical results might lead to breaking an RSA ciphertext? Or to a computation of the private key?
- How to choose the parameters in order to avoid weaknesses?

There is a good overview in:

D. Boneh: *Twenty years of attacks on the RSA cryptosystem.* Notices of the American Mathematical Society 46 (1999), 203–213.

2.1 The Prime Number Theorem

Let $\pi(x)$ be the number of primes $p \leq x$. Somewhat more generally let $\pi_{a,b}(x)$ be the number of primes $p \leq x$ of the form $p = ak + b$ (in other words: congruent to b modulo a). The prime number theorem states the asymptotic relation ()

$$\pi_{a,b}(x) \sim \frac{1}{\varphi(a)} \cdot \frac{x}{\ln(x)}$$

provided a and b are coprime. The special case $a = 1$, $b = 0$, is:

$$\pi(x) \sim \frac{x}{\ln(x)}.$$

There are many theoretical and empirical results concerning the quality of this approximation. An instance is a formula by Rosser and Schoenfeld:

$$\frac{x}{\ln(x)} \cdot \left(1 + \frac{1}{2\ln(x)}\right) < \pi(x) < \frac{x}{\ln(x)} \cdot \left(1 + \frac{3}{2\ln(x)}\right) \quad \text{for } x \geq 59.$$

The prime number theorem helps for answering the following questions (albeit not completely exactly):

How many prime numbers $< 2^k$ do exist?

Answer: $\pi(2^k)$, that is about

$$\frac{2^k}{k \cdot \ln(2)},$$

at least (for $k \geq 6$)

$$\frac{2^k}{k \cdot \ln(2)} \cdot \left(1 + \frac{1}{2k\ln(2)}\right).$$

For $k = 128$ this number is about $3.8 \cdot 10^{36}$, for $k = 256$, about $6.5 \cdot 10^{74}$.

How many k-bit primes do exist?

Answer: $\pi(2^k) - \pi(2^{k-1})$, that is about

$$\frac{2^k}{k \cdot \ln(2)} - \frac{2^{k-1}}{(k-1) \cdot \ln(2)} = \frac{2^{k-1}}{\ln(2)} \cdot \frac{k-2}{k(k-1)} \approx \frac{1}{2} \cdot \pi(2^k).$$

For $k = 128$ this amounts to about $1.9 \cdot 10^{36}$, for $k = 256$, to about $3.2 \cdot 10^{74}$. In other words, a randomly chosen k-bit integer is prime with probability

$$\frac{\pi(2^k) - \pi(2^{k-1})}{2^{k-1}} \approx \frac{\pi(2^k)}{2^k} \approx \frac{1}{k \cdot \ln(2)} \approx \frac{1.44}{k}.$$

For $k = 256$ this is about 0.0056.

The inequality

$$\pi(2^k) - \pi(2^{k-1}) > 0.71867 \cdot \frac{2^k}{k} \qquad \text{for } k \geq 21.$$

gives a reliable lower bound.

In any case the number of primes of size relevant for RSA is huge and makes an exhaustion attack completely obsolete.

Special Primes

Often cryptologists want their primes to have special properties:

Definition A **special prime** (or **safe prime**) is a prime of the form $p = 2p' + 1$ where p' is an odd prime (then p' is also called a GERMAIN prime).

Remark Let p be special. Then $p \equiv 3 \pmod 4$, for $p = 2p' + 1 \equiv 2 \cdot a + 1$ where $a = 1$ or 3.

Definition A **superspecial prime** is a prime of the form $p = 2p' + 1$ where $p' = 2p'' + 1$ is a special prime.

Examples The two smallest superspecial primes are $p = 23$ (with $p' = 11$, $p'' = 5$) and $q = 47$ (with $q' = 23$, $q'' = 11$).

Are there enough primes to fulfill these special or superspecial requests?

Frankly speaking, there is no exact answer. However we can give (unproven!) fairly exact estimates for these numbers:

- As we saw, a (positive) k-bit integer is prime with probability $\frac{\alpha}{k}$ where $\alpha \approx 1.44$.
- If $p = 2p' + 1$ is special, then p' is a $k/2$-bit integer, and is prime (heuristically, but in fact unknown) with probability $\frac{2\alpha}{k}$.
- Thus we estimate that a random k-bit integer is a special prime with probability $\frac{\alpha}{k} \cdot \frac{2\alpha}{k} = \frac{2\alpha^2}{k^2}$, and we expect that $\frac{\alpha^2}{k^2} \cdot 2^k$ of the 2^{k-1} k-bit integers are special primes (assuming that the "events" p prime and $(p-1)/2$ prime are independent).
- Moreover $p'' = (p' - 1)/2$ is a $k/4$-bit integer, hence prime with probability $\frac{4\alpha}{k}$.

- This makes up for a probability of

$$\frac{\alpha}{k} \cdot \frac{2\alpha}{k} \cdot \frac{4\alpha}{k} = \frac{8\alpha^3}{k^3}$$

for a k-bit integer to be a superspecial prime.

- By this consideration—although we have no mathematical proof for it—we expect that

$$\frac{\alpha^3}{k^3} \cdot 2^{k+2}$$

of the 2^{k-1} k-bit integers are superspecial primes.

- For $k = 256 = 2^8$ (and $\alpha^2 \approx 2$, $\alpha^3 \approx 3$) we may hope for

$$\begin{array}{rcll} 2 \cdot 2^{256} \cdot 2^{-16} & \approx & 3.5 \cdot 10^{72} & \text{special primes,} \\ 3 \cdot 2^{258} \cdot 2^{-24} & \approx & 8.3 \cdot 10^{70} & \text{superspecial primes.} \end{array}$$

Extensions

Let p_n be the n-th prime, thus $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, ... Let $\vartheta(x)$ be the sum of the logarithms of the primes $\leq x$,

$$\vartheta(x) = \sum_{p \leq x,\, p \text{ prime}} \ln(p).$$

Then we have the asymyptotic formulas

$$\begin{array}{rcl} p_n & \sim & n \cdot \ln(n), \\ \vartheta(x) & \sim & x, \end{array}$$

and the error bounds due to Rosser/Schoenfeld:

(1)
$$n \cdot \left(\ln(n) + \ln\ln(n) - \frac{3}{2}\right) < p_n < n \cdot \left(\ln(n) + \ln\ln(n) - \frac{1}{2}\right) \quad \text{for } n \geq 20,$$

(2)
$$x \cdot \left(1 - \frac{1}{\ln(x)}\right) < \vartheta(x) < x \cdot \left(1 - \frac{1}{2\ln(x)}\right) \quad \text{for } n \geq 41.$$

For a proof of the prime number theorem see any textbook on analytic number theory, for example

Apostol, T. M. *Introduction to Analytic Number Theory.* Springer-Verlag, New York 1976.

2.2 Computing the Key and Factorization

Question: *How to compute the private RSA exponent d, given the public exponent e and the module n?*

Answer: Each of the following tasks (A) – (D) is efficiently reducible to each of the other ones:

(A) Computing the private key d.

(B) Computing $\lambda(n)$ (CARMICHAEL function).

(C) Computing $\varphi(n)$ (EULER function).

(D) Factoring n.

Breaking RSA is the (possibly properly) easier task:

(E) Computing e-th roots in $\mathbb{Z}/n\mathbb{Z}$.

The "proof" (not an exact proof in the mathematical sense) follows the roadmap:

$$\begin{tikzcd}
C \arrow[r] & A \arrow[r] \arrow[dl] & E \\
D \arrow[u] \arrow[r] & B \arrow[u] &
\end{tikzcd}$$

We always assume that n and the public exponent e are known, and $n = p_1 \cdots p_r$ with different primes $p_1, \ldots, p_r$.

Clearly "A $\longrightarrow$ E": Taking an e-th root means raising to the d-th power. So if d is known, computing e-th roots is easy.

Note that the converse implication is unknown: *Breaking RSA could be easier than factoring.*

"D $\longrightarrow$ C": $\varphi(n) = (p_1 - 1) \cdots (p_r - 1)$.

"D $\longrightarrow$ B": $\lambda(n) = \mathrm{kgV}(p_1 - 1, \ldots, p_r - 1)$.

"B $\longrightarrow$ A": Compute d by congruence division from $de \equiv 1 \pmod{\lambda(n)}$.

"C $\longrightarrow$ A": Since $\varphi(n)$ has exactly the same prime factors as $\lambda(n)$, also $\varphi(n)$ is coprime with e. From $de \equiv 1 \pmod{\varphi(n)}$ we get a solution for d by congruence division. This might not be the "true" exponent, but works in the same way as private key since a forteriori $de \equiv 1 \pmod{\lambda(n)}$.

"A $\longrightarrow$ D" is significantly more involved. Moreover we only construct a probabilistic algorithm.

Preliminary Remarks

1. *It suffices to decompose n into two proper factors.*

   (a) Let $n = n_1 n_2$ be a proper decomposition, and assume for simplicity that $n_1 = p_1 \cdots p_s$ with $1 < s < r$. Then

   $$\lambda(n_1) = \mathrm{kgV}(p_1 - 1, \ldots, p_s - 1) \,|\, \mathrm{kgV}(p_1 - 1, \ldots, p_r - 1) = \lambda(n),$$

   thus also $de \equiv 1 \pmod{\lambda(n_1)}$. This reduces the problem to the analoguous ones for n_1 and n_2.

   (b) Since the number of prime factors of n is at most $\log_2(n)$ the recursive reduction suggested by (a) is efficient.

2. *How can a residue class $w \in \mathbb{Z}/n\mathbb{Z}$ help with factoring n?*

   (a) Finding a $w \in [1 \ldots n-1]$ with $\gcd(w, n) > 1$ decomposes n since $\gcd(w, n)$ is a proper divisor of n.

   (b) Finding a $w \in [2 \ldots n-2]$ with $w^2 \equiv 1 \pmod{n}$ (that is a non-trivial square root of 1 in $\mathbb{Z}/n\mathbb{Z}$) likewise decomposes n:
   Since $n | w^2 - 1 = (w + 1)(w - 1)$ and $n \nmid w \pm 1$ we have $\gcd(n, w + 1) > 1$, and this decomposes n by (a).

Now let (d, e) be a pair of RSA exponents. Then also $u := ed - 1 = k \cdot \lambda(n)$ is known (with unknown k and $\lambda(n)$). Since $\lambda(n)$ is even we may write

$$u = r \cdot 2^s \qquad \text{with } s \geq 1 \text{ and } r \text{ odd.}$$

If we choose a random $w \in [1 \ldots n - 1]$, then we have to deal with two possibilities:

- $\gcd(w, n) > 1$—then n is decomposed.
- $\gcd(w, n) = 1$—then $w^{r2^s} \equiv 1 \pmod{n}$.

In the second case we efficiently find the minimal $t \geq 0$ with

$$w^{r2^t} \equiv 1 \pmod{n}.$$

Again we distinguish two cases:

- $t = 0$—bad luck, choose another w.
- $t > 0$—then $w^{r2^{t-1}}$ is a square root $\neq 1$ of 1 in $\mathbb{Z}/n\mathbb{Z}$.

In the second case we distiguish:

- $w^{r2^{t-1}} \equiv -1 \pmod{n}$—bad luck, choose another w.

- $w^{r2^{t-1}} \not\equiv -1 \pmod{n}$—then n is decomposed by preliminary remark 2.

Thus every choice of $w \in [1 \dots n-1]$ has one of four possible outcomes, two of them decompose n, and the other two flop. Denote the last two events by

$$\begin{array}{ll} (\mathrm{E}_{n,u}(w)/\mathrm{I}) & w^r \equiv 1 \pmod{n} \\ (\mathrm{E}_{n,u}(w)/\mathrm{II}) & w^{r2^{t-1}} \equiv -1 \pmod{n} \quad \text{for a } t \text{ with } 1 \le t \le s. \end{array}$$

Altogether this yields a tree-like structure:

```
$w \in [1 \dots n-1] \longrightarrow$
    $\gcd(w, n) > 1 \longrightarrow n$ decomposed SUCCESS
    $w \in \mathbb{M}_n \longrightarrow$
        $w^r \equiv 1 \pmod{n} \longrightarrow (\mathrm{E}_{n,u}(w)/\mathrm{I})$ FLOP
        $w^r \not\equiv 1 \pmod{n} \longrightarrow$
            $w^{r2^{t-1}} \equiv -1 \pmod{n} \longrightarrow (\mathrm{E}_{n,u}(w)/\mathrm{II})$ FLOP
            $w^{r2^{t-1}} \not\equiv -1 \pmod{n} \longrightarrow n$ decomposed SUCCESS
```

Thus our procedure decomposes n “with high probability” if there are only “few” “bad” integers w with $(\mathrm{E}_{n,u}(w)/\mathrm{I,II})$. The next section will provide an upper bound for their number.

2.3 The Probability of Flops

Let $n \in \mathbb{N}_3$. Furthermore assume that $u \in \mathbb{N}_2$ is even, $u = r \cdot 2^s$ with odd r and $s \geq 1$. We introduce the sets:

$$\begin{array}{rcl}
A_u^{(0)} & = & B_u^{(0)} := \{w \in \mathbb{M}_n \mid w^r = 1\} \quad [\text{case } (\mathrm{E}_{n,u}/\mathrm{I})], \\
A_u^{(t)} & := & \{w \in \mathbb{M}_n \mid w^{r\cdot 2^t} = 1, w^{r\cdot 2^{t-1}} \neq 1\} \quad \text{for } 1 \leq t \leq s, \\
B_u^{(t)} & := & \{w \in A_u^{(t)} \mid w^{r\cdot 2^{t-1}} = -1\} \quad [\text{case } (\mathrm{E}_{n,u}/\mathrm{II})], \\
A_u & := & \bigcup_{t=0}^{s} A_u^{(t)} = \{w \in \mathbb{M}_n \mid w^u = 1\}, \\
B_u & := & \bigcup_{t=0}^{s} B_u^{(t)} \quad [\text{case } (\mathrm{E}_{n,u}) \text{ (I or II)}]. \\
C_0 & := & \{w \in \mathbb{M}_n \mid \operatorname{ord} w \text{ odd}\}, \\
C_1 & := & \{w \in \mathbb{M}_n \mid -1 \in \langle w \rangle\}, \\
C & := & C_0 \cup C_1.
\end{array}$$

Remarks

- $A_u^0 \leq A_u \leq \mathbb{M}_n$ are subgroups, as are $A_u^0 \leq C_0 \leq \mathbb{M}_n$.
- $B_u^{(t)} = A_u^{(t)} \cap C$ for $t = 0, \ldots, s$, since a cyclic group $\langle w \rangle$ can contain at most one square root of 1 in addition to 1 itself.
- Hence also $B_u = A_u \cap C$.
- B_u is the exceptional set of "bad" integers with $(\mathrm{E}_{n,u})$ from Section 2.2 that flop with factoring n. The following proposition upper bounds by $\frac{1}{2}$ the probability of hitting an element of this set by pure chance. If we try k random candidate integers the probability of not factoring n is $< 1/2^k$, hence *extremely* small even for moderate sizes of k

Proposition 4 *Let n be odd and not a prime power. Let $u = r \cdot 2^s$ be a multiple of $\lambda(n)$ with odd r. Then*

$$\#B_u \leq \frac{1}{2} \cdot \varphi(n).$$

Proof. By the following lemma C, and a forteriori B_u, is contained in a proper subgroup of $\mathbb{M}_n$. $\diamond$

Lemma 1 (Dixon, AMM 1984) *Let $n \in \mathbb{N}_3$. Assume $\langle C \rangle = \mathbb{M}_n$. Then n is a prime power or even.*

Proof. For this proof let $\lambda(n) = r \cdot 2^s$ with odd r. (Since $n \geq 3$, we have $s \geq 1$. The "old" meanings of r and s don't occur in this proof.) Consider the map

$$h : \mathbb{M}_n \longrightarrow \mathbb{M}_n, \qquad w \mapsto w^{r \cdot 2^{s-1}}.$$

This h is a group homomorphism with $h(\mathbb{M}_n) \subseteq \{v \in \mathbb{M}_n \mid v^2 = 1\}$ (group of square roots of 1 mod$n$). Since the $w \in C_0$ have odd order $h(C_0) \subseteq \{1\}$.

For $w \in C_1$ we have $h(w) \in \langle w \rangle$ and $h(w)^2 = 1$, hence $h(w)$ is one of the two roots of unity $\pm 1 \in \langle w \rangle$.

Together we have $h(C) \subseteq \{\pm 1\}$.

If n is not a prime power (and a forteriori not a prime) there is a decomposition $n = pq$ into coprime factors $p, q \in \mathbb{N}_2$. Since $2^s | \lambda(n) = \operatorname{lcm}(\lambda(p), \lambda(q))$ we may assume $2^s | \lambda(p)$. The chinese remainder theorem provides a $w \in \mathbb{M}_n$ with $w \equiv 1 \pmod q$ such that $w \bmod p$ has order 2^s. Then $h(w) \not\equiv 1 \pmod p$, a forteriori $h(w) \neq 1$. Since $h(w) \equiv 1 \pmod q$ we also have $h(w) \neq -1$—except when $q = 2$.

Therefore if n is not even nor a prime power we have the contradiction $h(\mathbb{M}_n) \not\subseteq \{\pm 1\}$. $\diamond$

This also completes the missing step of Section 2.2: Who knows the private RSA key is able to factor the module n.

2.4 Factoring Algorithms

A crucial question for the security of RSA is: *How fast can we factorize large integers?*

- There are "fast" factoring algorithms for integers of the form $a^b \pm c$ with "small" values a and c, the most prominent examples are the MERSENNE and FERMAT primes $2^b \pm 1$. The probability that the generation of RSA keys from random primes yields such a module is extremely low and usually neglected.
- FERMAT factoring of n: Find an integer $a \geq \sqrt{n}$ such that $a^2 - n$ is a square $= b^2$. This yields a decomposition

  $$n = a^2 - b^2 = (a+b)(a-b).$$

  Example: $n = 97343$, $\sqrt{n} \approx 311.998$, $312^2 - n = 1$, $n = 313 \cdot 311$. This method is efficient provided that we find an a close to $\sqrt{n}$, or $a^2 \approx n$. In the case $n = pq$ of two factors this means a small difference $|p - q|$. (Un-) fortunately finding a seems to be hard.
- The fastest general purpose factoring algorithms
  - number field sieve (SILVERMAN 1987, POMERANCE 1988, A. K. LENSTRA/ H. W. LENSTRA/ MANASSE/ POLLARD 1990),
  - elliptic curve factoring (H. W. LENSTRA 1987, ATKIN/ MORAIN 1993),

  need time of size

  $$L_n := e^{\sqrt[3]{\ln n \cdot (\ln \ln n)^2}},$$

  hence are "subexponential", but also "superpolynomial". Anyway they show that *factoring is a significantly more efficient attack on RSA than exhaustion ("brute force").*

This results in the following estimates for factoring times:

integer	bits	decimal places	expense (MIPS years)	status
rsa120	399	120	100	$<$ 1 weak on a PC
rsa154	512	154	100 000	TE RIELE 1999
rsa200	665	200	$(*)$	FRANKE 2005
	1024	308	10^{11}	insecure
	2048	616	10^{15}	for short-term security

$(*)$ 80 CPUs à 2.2 GHz in 4.5 months

When we extrapolate these estimates we should note:

- they are rough approximations only,
- they hold only as long as nobody finds significantly faster factoring algorithms.

Remember that the existence of a polynomial factoring algorithm is an *open problem.*

Some recent developments are already incorporated into the table:

- A paper by A. K. Lenstra/ E. Verheul, *Selecting cryptographic key sizes* summarizes the state of the art in the year 2000 and extrapolates it.
- A proposal by Bernstein, *Circuits for integer factorization* triples (!) the length of integers that can be factorized with a given expense, using the fastest factoring algorithms.
- Special-purpose hardware designs by Shamir and his collaborators:
  - TWINKLE (The Weizmann Institute Key Locating Machine) (1999) is the realization in hardware of an idea by Lehmer from the 1930s that accelerates factoring 100–1000 times,
  - TWIRL (The Weizmann Institute Relation Locator) (2003) accelerates factoring another 1000–10000 times following Bernstein's idea.

  Taken together these approaches make factoring 10^6 (or 2^{20}) times faster using the number field sieve. However the order of magnitude L_n of the complexity is unaffected.

This progress lets the Lenstra/ Verheul estimates look overly optimistic. 1024-*bit keys should no longer be used.* 2048-bit keys might be secure enough to protect information for a few years.

Recommendation: Construct your RSA module $n = pq$ from primes p and q that have bit lengths of at least 2048 bits, and choose them such that also their difference $|p - q|$ has a bit length of about 2048 bits.

2.5 Iteration Attack

Consider a bijective map $E\colon M \longrightarrow M$ of a finite set M onto itself and its inverse $D = E^{-1}$ (think of E as an encryption function). Then E is an element of the full symmetric group $\mathfrak{S}(M)$ that has the (huge) order $\#\mathfrak{S}(M) = (\#M)!$. Nevertheless this group is finite, thus there is an $s \in \mathbb{N}_1$ with $E^s = \mathbf{1}_M$, hence

$$D = E^{s-1}.$$

As a consequence an attacker can compute D from E by sufficiently many iterations. This attack is relevant only for asymmetric ciphers where the attacker knows E. The only protection against it is *to choose the order of E, the smallest $s \geq 1$ with $E^s = \mathbf{1}_M$, as large as possible.*

The Example of RSA

Let $M = \mathbb{Z}/n\mathbb{Z}$, then $\#\mathfrak{S}(M) = n!$, where n itself is a very large integer. The attacker could compute $E^{n!-1}$, but even the fastest power algorithm is not fast enough to accomplish this task in this universe. So the attack doesn't seem to put RSA into immediate danger.

However, as a closer look reveals, RSA encryption functions are contained in a significantly smaller subgroup of $\mathfrak{S}(M)$—fortunately the attacker doesn't know its order. To see this consider the map

$$\Phi\colon \mathbb{N} \longrightarrow \operatorname{map}(M, M), \quad e \mapsto E_e \quad \text{with } E_e(a) = a^e \bmod n.$$

Here are some of its properties:

1. For $e, f \in \mathbb{N}$ we have $E_{ef} = E_e \circ E_f$ since $a^{ef} \equiv (a^f)^e \pmod{n}$ for all $a \in M$. Hence Φ is a homomorphism of the multiplicative semigroup $\mathbb{N}$.

2. If $e \equiv f \pmod{\lambda(n)}$, then $E_e = E_f$: Assume $f = e + k\lambda(n)$, then $a^f = a^{e+k\lambda(n)} \equiv a^e \pmod{n}$ for all $a \in M$.

3. If $e \bmod \lambda(n)$ is invertible, then E_e is bijective: Assume $de \equiv 1 \pmod{\lambda(n)}$, then $E_d \circ E_e = E_1 = \mathbf{1}_M$. Hence the map

   $$\bar{\Phi}\colon \mathbb{M}_{\lambda(n)} \longrightarrow \mathfrak{S}(M)$$

   induced by Φ is a group homomorphism.

4. $\bar{\Phi}$ is injective: For if $\Phi(e) = E_e = \mathbf{1}_M$, then $a^e \equiv a \pmod{n}$ for all $a \in M$, hence $a^{e-1} \equiv 1 \pmod{n}$ for all $a \in \mathbb{M}_n$, hence $\lambda(n) | e-1$, thus $e \equiv 1 \pmod{\lambda(n)}$.

These remarks prove:

Proposition 5 *The RSA encryption functions E_e form a subgroup $H_n \leq \mathfrak{S}(M)$ that is isomorphic with $\mathbb{M}_{\lambda(n)}$ and has order $\varphi(\lambda(n))$ and exponent $\lambda(\lambda(n))$.*

Of course the order of a single encryption function E_e could be even much smaller: All we can say is that the cyclic subgroup $\langle e \rangle \leq \mathbb{M}_{\lambda(n)}$ has order $s := \operatorname{ord}(e) \mid \lambda(\lambda(n))$.

This observation raises two problems:

1. How large is $\lambda(\lambda(n))$?
2. Under what conditions is $\operatorname{ord}(e) = \lambda(\lambda(n))$? Or at least not significantly smaller?

Answer to 1 (without proof): “In general” $\lambda(\lambda(n)) \approx \frac{n}{8}$.

If we want to be sure about this we should choose p, q as special primes $p = 2p' + 1$, $q = 2q' + 1$ with different primes $p', q' \geq 3$. Then for $n = pq$ we have

$$\lambda(n) = \operatorname{kgV}(2p', 2q') = 2p'q' = \frac{(p-1)(q-1)}{2} \approx \frac{n}{2}.$$

If moreover p and q are superspecial primes, that is, $p' = 2p'' + 1$ and $q' = 2q'' + 1$ are special primes too, then

$$\lambda(\lambda(n)) = 2p''q'' = \frac{(p-3)(q-3)}{8} \approx \frac{n}{8}.$$

By the prime number theorem, see Section 2.1, we may expect that superspecial primes exist in astronomic quantities.

Answer to 2: in most cases (also without general proof).

Again, if we want to be sure, we should confine our choices to special or even superspecial primes. We use some elementary results on finite groups, see Lemmas 21, 22, and 23 of Appendix A.10.

Let p be an odd prime number. In the additive cyclic group $\mathbb{Z}/2p\mathbb{Z}$ we consider the subsets:

$$\begin{aligned} E_p &= \{a \bmod 2p \mid 0 \leq a < p,\ a \text{ even}\} - \{0\}, \\ O_p &= \{a \bmod 2p \mid 0 \leq a < p,\ a \text{ odd}\} - \{p\}. \end{aligned}$$

Clearly, $\mathbb{Z}/2p\mathbb{Z} = \{0, p\} \cup E_p \cup O_p$, and

$$\#E_p = \#O_p = p - 1.$$

The order of an element $x \in \mathbb{Z}/2p\mathbb{Z}$ is

$$\operatorname{ord} x = \begin{cases} 1 & \Longleftrightarrow x = 0, \\ 2 & \Longleftrightarrow x = p, \\ p & \Longleftrightarrow x \in E_p, \\ 2p & \Longleftrightarrow x \in O_p. \end{cases}$$

We transfer this result to an abstract cyclic group $\mathcal{Z}_{2p}$ with generating element g via the isomorphism

$$\tau : \mathbb{Z}/2p\mathbb{Z} \longrightarrow \mathcal{Z}_{2p}, \quad x \mapsto g^x.$$

Let $\mathcal{E}_p = \tau E_P$ and $\mathcal{O}_p = \tau O_P$. Then the result is:

Lemma 2 *The order of an element $h \in \mathcal{Z}_{2p}$ is*

$$\operatorname{ord} h = \begin{cases} 1 & \Longleftrightarrow h = \mathbf{1}, \\ 2 & \Longleftrightarrow h = g^p, \\ p & \Longleftrightarrow h \in \mathcal{E}_p, \\ 2p & \Longleftrightarrow h \in \mathcal{O}_p. \end{cases}$$

Next we study the orders of the elements of the direct product $\mathcal{Z}_{2p} \times \mathcal{Z}_{2q}$ for two different odd primes p and q. Applying Lemma 21 we see that the order of a pair (g, h) for $g \in \mathcal{Z}_{2p}$ and $h \in \mathcal{Z}_{2q}$ is given by the following table:

		ord $g =$			
		1	2	p	$2p$
ord $h =$	1	1	2	p	$2p$
	2	2	2	$2p$	$2p$
	q	q	$2q$	pq	$2pq$
	$2q$	$2q$	$2q$	$2pq$	$2pq$

An obvious count yields:

Proposition 6 *Let p and q be two different odd primes. Then the direct product group $\mathcal{Z}_{2p} \times \mathcal{Z}_{2q}$ has*

(i) 1 *element of order* 1,

(ii) 3 *elements of order* 2,

(iii) $p-1$ *elements of order* p,

(iv) $3 \cdot (p-1)$ *elements of order* $2p$,

(v) $q-1$ *elements of order* q,

(vi) $3 \cdot (q-1)$ *elements of order* $2q$,

(vii) $(p-1) \cdot (q-1)$ *elements of order* pq,

(viii) $3 \cdot (p-1) \cdot (q-1)$ *elements of order* $2pq$.

Again let p be a prime number. Then the multiplicative group $\mathbb{M}_p = (\mathbb{Z}/p\mathbb{Z})^\times$ of the finite field $\mathbb{Z}/p\mathbb{Z}$ is cyclic of order $p-1$. Let q be a prime different from p and let $n = p \cdot q$. Then by the Chinese Remainder Theorem $\mathbb{M}_n \cong \mathbb{M}_p \times \mathbb{M}_q$ is (up to isomorphy) the direct product of two cyclic groups of orders $p-1$ and $q-1$. Hence:

Lemma 3 *Let $n = pq$ be the product of two different odd primes p and q. Then the multiplicative group $\mathbb{M}_n = (\mathbb{Z}/n\mathbb{Z})^\times$ of the quotient ring $\mathbb{Z}/n\mathbb{Z}$ has order $\varphi(n) = (p-1)(q-1)$ and exponent $\lambda(n) = \operatorname{lcm}(p-1, q-1)$. In particular $\mathbb{M}_n$ is not cyclic.*

The latter statement is due to the common divisor 2 of $p-1$ and $q-1$.

We now consider the case where $p = 2p' + 1$ and $q = 2q' + 1$ are special primes. Then

$$\varphi(n) = 4p'q' \quad \text{and} \quad \lambda(n) = 2p'q'.$$

By Proposition 5 the RSA encryption functions for the module $n = pq$ make up a group H_n isomorphic with $\mathbb{M}_{\lambda(n)}$. For special primes we therefore have by Theorem 2 in Appendix A.4:

Proposition 7 *Let $n = pq$ be the product of two different special primes $p = 2p' + 1$ and $q = 2q' + 1$. Then the RSA group*

$$H_n \cong \mathbb{M}_{\lambda(n)} \cong \mathcal{Z}_{p'-1} \times \mathcal{Z}_{q'-1}$$

is the product of two cyclic groups of orders $p' - 1$ and $q' - 1$.

In order to derive some more easy results we assume that p and q are superspecial primes, with $p' = 2p'' + 1$ and $q' = 2q'' + 1$. Then

$$H_n \cong \mathbb{M}_{\lambda(n)} \cong \mathcal{Z}_{2p''} \times \mathcal{Z}_{2q''},$$

and Proposition 6 applies for the primes p'' and q'':

Proposition 8 *Let $n = pq$ be the product of two different superspecial primes $p = 2p' + 1$ and $q = 2q' + 1$ with $p' = 2p'' + 1$ and $q' = 2q'' + 1$. Then the RSA group H_n consists of*

(i) 1 *element of order* 1,

(ii) 3 *elements of order* 2,

(iii) $p'' - 1$ *elements of order* p'',

(iv) $3 \cdot (p'' - 1)$ *elements of order* $2p''$,

(v) $q'' - 1$ *elements of order* q'',

(vi) $3 \cdot (q'' - 1)$ *elements of order* $2q''$,

(vii) $(p'' - 1) \cdot (q'' - 1)$ *elements of order* $p''q''$,

(viii) $3 \cdot (p'' - 1) \cdot (q'' - 1)$ *elements of order* $2p''q''$.

Since $2p''q'' = \lambda(\lambda(n))$ is the exponent of H_n we see that almost all of its elements have their orders near the maximum. More precisely the number of elements of order $< \frac{1}{2}\lambda(\lambda(n)) = p''q''$ is

$$1 + 3 + 4 \cdot (p'' - 1) + 4 \cdot (q'' - 1) = 4 \cdot (p'' + q'' - 1).$$

Corollary 1 *The number of elements of* H_n *with order* $< \frac{1}{2}\lambda(\lambda(n))$ *is* $p + q - 7$.

Proof. Note that $p'' = (p - 3)/4$. $\diamond$

Thus this number is $\approx 2 \cdot \sqrt{n}$ if p and q—as recommended in Section 2.4—are chosen near $\sqrt{n}$. Then the proportion of elements of "small" orders is $\approx 2/\sqrt{n}$, and this proportion asymptotically tends to 0 with growing values of n.

As a consequence we resume: *With negligeable exceptions s has the order of magnitude of $n/8$.* The best known general results are in Chapter 23 of Shparlinski's book, see the references for these lecture notes.

In addition to Section 2.2 we formulate the task

(F) Finding the order s of the encryption function.

In the sense of complexity theory we have the implication

$$\text{(F)} \longrightarrow \text{(A)}$$

but maybe not the reverse implication. If the order s is known, then $D = E^{s-1}$ and thus $d = e^{s-1}$ are efficiently computable. *Finding the order of the encryption function is at least as difficult as factoring the module.*

2.6 Breaking Single Ciphertexts

Breaking a single ciphertext (without necessarily computing the private key) could be even easier: For a given ciphertext c we could have $E_e^r(c) = c$ even if $E_e^r \neq \mathbf{1}_M$. If a is the corresponding plaintext, $c = E_e(a)$, then the cryptanalyst can compute:

$$E_e^{r-1}(c) = D_e(E_e^r(c)) = D_e(c) = a.$$

From a mathematical viewpoint we have the situation:

- The group $\mathbb{M}_{\lambda(n)}$ acts on the set $M = \mathbb{Z}/n\mathbb{Z}$, as does its cyclic subgroup $G := \langle e \rangle \leq \mathbb{M}_{\lambda(n)}$.
- For $a \in M$ the orbit is $G \cdot a = \{a^{e^k} \mid 0 \leq k < s\}$ (where s is the order of e in the multiplicative group $\mathbb{M}_{\lambda(n)}$).
- The stabilizer $G_a = \{f \in G \mid a^f \equiv a \pmod{n}\}$ is a subgroup of G. We have a natural bijective correspondence between the sets $G \cdot a$ and G/G_a.
- For the orbit length $t = \#G \cdot a$ we have

$$\begin{aligned} t = \frac{s}{\#G_a}, &\qquad t|s|\lambda(\lambda(n)) \\ E_e^r(c) = c &\iff E_e^r(a) = a \iff t|r. \end{aligned}$$

- $G \cdot c = G \cdot a$ and $\#G_c = \#G_a$. (The two stabilizers are conjugate.)
- Finding the orbit length t of a and c is at least as difficult as breaking the ciphertext c.

This suggests yet another problem:

3. Under what conditions is $t = s$, in other words, which stabilizers G_a are trivial? Or at least quite small?

Answer once more (without proof): in most cases. For superspecial primes p and q where $\lambda(\lambda(n)) = 2p''q''$ we expect by similar considerations as in Section 2.5 that $t < p''q''$ only for a negligeable set of exceptions.

The following two papers show how low is the risk of hitting a small orbit length by pure chance, enabling an iteration attack:

- J. J. Brennan/ Bruce Geist, Analysis of iterated modular exponentiation: The orbits of x^α mod N. **Designs, Codes and Cryptography** 13 (1998), 229–245.
- John B. Friedlander/ Carl Pomerance/ Igor E. Shparlinski, Period of the power generator and small values of Carmichael's function. **Mathematics of Computation** 70 (2001), 1591–1606, + 71 (2002), 1803–1806.

2.7 Re-Use of a Module

Question: What happens if two different participants use the same RSA module n?

In other words, A and B use (n, e_{A}) and (n, e_{B}) as public keys.

Obviously A and B may read each other's messages since both can factorize n and hence compute the other's private key. Thus a common module makes sense only in a cooperative situation where A and B absolutely trust each other.

However it's even worse: A message a sent to both A and B is readable *by everyone.* The ciphertexts are:

$$c_{\mathrm{A}} = a^{e_{\mathrm{A}}} \bmod n, \quad c_{\mathrm{B}} = a^{e_{\mathrm{B}}} \bmod n.$$

Assuming e_{A} and e_{B} coprime is no significant loss of generality. Then the attacker, using the extended Euclidean algorithm, finds coefficients x and y with

$$xe_{\mathrm{A}} + ye_{\mathrm{B}} = 1.$$

Necessarily x and y have opposite signs, assume $x < 0$. If $\gcd(c_{\mathrm{A}}, n) > 1$, then the attacker can decompose n and is done. Otherwise she computes

$$g := c_{\mathrm{A}}^{-1} \bmod n$$

by congruence division and gets

$$g^{-x} \cdot c_{\mathrm{B}}^{y} \equiv (a^{e_{\mathrm{A}}})^{x} \cdot (a^{e_{\mathrm{B}}})^{y} \equiv a \pmod{n},$$

breaking the ciphertext without computing the private keys d_{A} and d_{B}.

Hence the common module n is secure only when A and B trust each other and moreover keep the module secret. But in this situation it makes much more sense to use a symmetric cipher.

2.8 Small Exponents

Question: Is RSA in danger if someone chooses a small public exponent e?

The exponent $e = 1$ is nonsensical since it leaves plaintexts unencrypted.

The exponent $e = 2$ doesn't work for RSA since it is even and thus not coprime with $\lambda(n)$. Nevertheless the related RABIN cipher uses $e = 2$. Here the receiver of the message must be able to take square roots $\bmod n$, and this works since he knows the prime factors of n (see later). (By the way he must also be able to recognize the true plaintext among several different square roots.)

Same Message for Several Receivers

For RSA the smallest potentially suited exponent is $e = 3$. However it enables an attack that applies as soon as someone sends the same message a to *three* different receivers A, B, and C. Let their public keys be $(n_\mathrm{A}, 3)$, $(n_\mathrm{B}, 3)$, and $(n_\mathrm{C}, 3)$. Assume the modules n_A, n_B, and n_C are pairwise coprime, otherwise the attacker factorizes at least two of them and reads a. Then (with some luck) she intercepts three ciphertexts

$$c_\mathrm{A} = a^3 \bmod n_\mathrm{A}, \quad c_\mathrm{B} = a^3 \bmod n_\mathrm{B}, \quad c_\mathrm{C} = a^3 \bmod n_\mathrm{C},$$

with $0 \leq a < n_\mathrm{A}, n_\mathrm{B}, n_\mathrm{C}$, thus $a^3 < n_\mathrm{A} n_\mathrm{B} n_\mathrm{C}$. Using the chinese remainder algorithm she constructs an integer $\tilde{c} \in \mathbb{Z}$ with

$$0 \leq \tilde{c} < n_\mathrm{A} n_\mathrm{B} n_\mathrm{C}$$

such that

$$\tilde{c} \equiv c_\mathrm{X} \bmod n_\mathrm{X} \quad \text{for } \mathrm{X} = \mathrm{A}, \mathrm{B}, \mathrm{C}.$$

By uniqueness $\tilde{c} = a^3$ in $\mathbb{Z}$. So she computes $a = \sqrt[3]{\tilde{c}}$ by taking the integer root in $\mathbb{Z}$. This is an efficient procedure. (In this situation she doesn't succeed with computing the private exponents.)

This attack obviously generalizes to other "small" shared public exponents e: If the same message is sent to e different people, then everybody can read it. This attack is not completely unrealistic: Think for example of fixed "protocol information" at the beginning of a larger message. Even in classical cryptography an important maxim was: *Never encrypt the same plaintext with different keys.*

In practice the exponent $e = 2^{16} + 1 = 65537$ is considered as sufficiently secure for "normal" situations.

Stereotypical Message Parts

Consider the key parameters (n, e, d). Imagine an attack with known plaintext that reads:

```
Der heutige Tagesschluessel ist:********
```

("The master key for today is: . . . ", example by Julia Dietrichs) with known (stereotypical) 32 byte part "`Der heutige Tagesschluessel ist:`", and unknown 8 byte part "`********`".

This message is encoded by the 8-bit character code ISO-8859-1 (used for German texts) as a sequence of 40 bytes or 320 bits, and for encryption by RSA interpreted as an integer $a \in [0 \ldots n-1]$ (assume n has more then 320 bits, and $e = 3$). Decompose a as $a = b + x$ where b corresponds to the known, and x, to the unknown part. Since the latter forms the end of the message and consists of 64 bits we know $x < 2^{64}$. Encryption yields the ciphertext

$$c = a^e \bmod n = (b + x)^e \bmod n.$$

Hence the secret x is a root of the polynomial

$$(T + b)^e - c \in (\mathbb{Z}/n\mathbb{Z})[T].$$

At first sight this observation doesn't seem alarming since we know of no general efficient algorithms that compute roots. However algorithms for certain special cases exist, for instance:

> COPPERSMITH's **algorithm**
> Let $f \in (\mathbb{Z}/n\mathbb{Z})[T]$ be a polynomial of degree r. The algorithm finds all roots x of f with $0 \leq x < \sqrt[r]{n}$ (or proves that there are none).
> The execution time is polynomial in $\log n$ and r.
> (The algorithm uses the "LLL algorithm" for reduction of lattice bases.)

In our example n has at least 321 bits, and $e = 3$. Thus the algorithm outputs x since $x^3 < 2^{192} < 2^{320} < n$.

This is only a simple example of a larger class of attacks for special situations that amount to a computation of roots mod n.

Exercise. Modify the attack for a situation where the unknown part of the plaintaxt consists of some contiguous letters surrounded by known plaintext sequences.

2.9 The Signature Trap

The signature trap doesn't challenge the security of RSA itself, but the frame conditions of its use: Since reversing the order of encryption and decryption is the basic mechanism of digital signatures the user has to take care that he doesn't inadvertently decrypt a ciphertext in the erroneous belief that he digitally signs a document. Would the standard input to the signature algorithm be a normal plaintext, the user would realize this situation at once. However for (at least) three reasons the situation is different:

1. To get acceptable performance usually a digital signature is applied to a (cryptographic) hash value of a document. This cannot be distiguished from a random bitstring.

2. Strong authentication requires digitally signing a random bitstring instead of entering a password to prove the user's identity. Even if the result was a decrypted plaintext—the user wouldn't see it at all since it is immediately sent to the communication partner (that might be a server, or a "man in the middle").

3. Moreover the attacker could present an arbitrary text that is "camouflaged" by some kind of encryption, and require the user to "sign" (i. e. decrypt) it. Even a close inspection of the result would not detect the fraud—see below. This is a otherwise very useful property of RSA: It is the basis for blind signatures and hence the generation of digital pseudonyms and anonymous transactions.

By the way this an instance of an *attack with chosen ciphertext.* To escape this attack in practice each of the three (or four) functions

- encryption,
- digital signature,
- strong authentication,
- (optionally) blind signature,

should use a different key pair.

Now for the "camouflage" that disguises the chosen ciphertext attack. Here is the procedure:

1. The attacker M ("Mallory") has an intercepted ciphertext $x = E_{\mathrm{A}}(a)$ and would like to read it. He encrypts it as $y = C(x)$ using a function C known only to him.

2. He presents y to his victim A ("Alice") and requires a digital signature. A generates $z = D_{\mathrm{A}}(y)$.

3. M removes the "camouflage" by a suitable inverse transformation C'. For this he needs a pair (C, C') of transformations such that

$$C' \circ D_{\mathrm{A}} \circ C = D_{\mathrm{A}}.$$

   Then $a = D_{\mathrm{A}}(x) = C'(z)$.

As a peculiarity of RSA such pairs (C, C') of transformations exist: Let $E_{\mathrm{A}}(a) = a^e \bmod n$, and take C as the shift by u^e on $\mathbb{M}_n = (\mathbb{Z}/n\mathbb{Z})^\times$, and C' as the multiplication by $u^{-1} \bmod n$ where $u \in \mathbb{M}_n$ is randomly chosen. Thus the attack runs through the steps:

1. M chooses u und computes $y = C(x) = u^e x \bmod n$.
2. A generates $z = y^d \bmod n$.
3. M computes

$$C'(z) = zu^{-1} = y^d u^{-1} = u^{ed} x^d u^{-1} = x^d = a$$

   in $\mathbb{Z}/n\mathbb{Z}$.

2.10 More Attacks

Finally we give a short overview over some other attacks on RSA. For a comprehensive treatment consult the paper by D. Boneh (see the introduction of this section):

1. **Small private exponent:** M. Wiener detected a way of efficiently computing the private key d from the public key (n, e) using continued fractions in the case $d < \frac{1}{3} \cdot \sqrt[4]{n}$.

2. **Related plaintexts** after Franklin/Reiter. Assume two different plaintexts a_1 and a_2 are related by an affine equation $a_2 = sa_1 + t$ with known coefficients s, $t \neq 0$. Then the corresponding plaintexts are efficiently computable from the public key (n, e), the coefficients s and t and the ciphertexts. Coppersmith found a situation that forces such an affine equation in the case where a_1 and a_2 originate from the same plaintext by "padding" differently.

3. **Partial leak** after Boneh/Durfee/Frankel/Coppersmith. If the last quarter of the bits of one of the integers d (the private exponent), p, or q (the prime factors of the module) are known, then n may be efficiently factorized.

4. **Timing and power attacks** after Kocher. The attacker observes the CPU during a decryption and measures the execution time or the power consumption. From this she gains informations about the bits of the private exponent. See the binary power algorithm, Section 1.2.

5. **Differential fault analysis** after Shamir et al. The attacker exposes the processor (for instance a smartcard) to environment conditions slightly outside the range where the specification guarantees a faultless operation, for instance by deforming, heating, radiation. Then the processor will produce single faulty bits that allow statistical inferences about the internal parameters.

Other attacks don't target the RSA algorithm itself but bugs in the implementation, faulty use in cryptographic protocols, flawed interaction with the system environment, and other mistakes.

In some situations using modules with more then two prime factors might even be advantageous, as the following paper suggests:

- M. Jason Hinek, Mo King Low, Edlyn Teske: On some attacks on multi-prime RSA. SAC 2002, 385–404.

Chapter 3

Primality Tests

A crucial question when implementing RSA is how to find the necessary primes for key generation. The answer will be given in form of efficient procedures. Start with a random integer of the desired length and test it for primality. If it is not prime take the next integer and so on. Eventually a prime will occur.

For this we need procedures that efficiently decide whether an integer is prime or not—primality tests.

We'll encounter a phenomenon that is familiar also with other mathematical problems (for instance linear optimization, numerical approximation of roots of polynomials over the real or complex numbers):

- There is an algorithm that gets by with polynomial cost.
- There is a "standard algorithm" (in the examples: the simplex method, the Newton algorithm) that is much more efficient for "most" instances, but needs more than polynomial cost in the "worst case". In practice this algorithm is the preferred one.

For primality testing the AKS algorithm is polynomial, but usually slower than the established Rabin algorithm. The latter is usually very efficient, but in the worst case even fails to deliver a correct result.

All these primality tests have a considerable overhead. Therefore for a practical implementation it makes sense to first check divisibility by "small" primes, say primes $< 10^6$, depending on the available storage (precompute a list L of small primes).

If we need a random prime of a certain size we randomly choose an integer r of this size. If r is even we increment it by 1. Then we sieve an interval $[r, r+s]$ for multiples of the primes in L by Eratosthenes' method. We test the remaining integers for primality until we find one that passes the test. In most cases this will be the first one already.

3.1 The Pseudoprime Test

How can we identify an integer as prime? The "naive" approach is trial divisions by all integers $\leq \sqrt{n}$, made perfect in the form of Eratosthenes sieve. An assessment of the cost shows that this approach is not efficient since $\sqrt{n} = \exp(\frac{1}{2}\log n)$ grows exponentially with the length $\log n$ of n.

An approach to identify primes without trial divisions is suggested by Fermat's theorem: If n is prime, then $a^{n-1} \equiv 1 \pmod{n}$ for all $a = 1, \ldots, n-1$. Note that this is a necessary condition only, not a sufficient one. Thus we say that n is a (Fermat) **pseudoprime to base** a if $a^{n-1} \equiv 1 \pmod{n}$. Hence a prime number is a pseudoprime to each base $a = 1, \ldots, n-1$.

Examples

1. The congruence $2^{14} \equiv 4 \pmod{15}$ shows that 15 is not prime.

2. We have $2^{340} \equiv 1 \pmod{341}$ although $341 = 11 \cdot 31$ is not prime. Anyway $3^{340} \equiv 56 \pmod{341}$, hence 341 fails the pseudoprime test to base 3.

The pseudoprime property is not sufficient for primality. Therefore we call n a Carmichael **number** if n is a pseudoprime to each base a that is coprime with n, but n is not a prime.

Another way to express pseudoprimality is that the order of a in $\mathbb{M}_n$ divides $n-1$. Thus n is a Carmichael number or prime if and only if $\lambda(n) \mid n-1$ with the Carmichael function λ.

Unfortunately there are many Carmichael numbers, so pseudoprimality cannot even considered as "almost sufficient" for primality. In 1992 Alford, Granville, and Pomerance proved that there are infinitely many Carmichael numbers.

The smallest Carmichael number is $561 = 3 \cdot 11 \cdot 17$. This is a direct consequence of the next proposition.

Proposition 9 *A natural number n is a* Carmichael *number if and only if it is not prime, squarefree, and $p-1 \mid n-1$ for each prime divisor p of n. An odd* Carmichael *number has at least* 3 *prime factors.*

Proof. "$\Longrightarrow$": Let p be a prime divisor of n.

Assume $p^2|n$. Then $\mathbb{M}_n$ contains a subgroup isomorphic with $\mathbb{M}_{p^e}$ for some $e \geq 2$, hence by Proposition 18 in Appendix A.3 also a cyclic subgroup of order p. This leads to the contradiction $p \mid n-1$.

Since $\mathbb{M}_n$ contains a cyclic group of order $p-1$ it has an element a of order $p-1$, and $a^{n-1} \equiv 1 \pmod{n}$, hence $p-1 \mid n-1$.

"$\Longleftarrow$": Since n is squarefree by the chinese remainder theorem the multiplicative group $\mathbb{M}_n$ is the direct product of the cyclic groups $\mathbb{F}_p^\times$ where p runs through the prime divisors of n. Since all $p-1 \mid n-1$ the order of each element of $\mathbb{M}_n$ divides $n-1$.

Proof of the addendum: Let n be an odd Carmichael number. Suppose $n = pq$ with two primes p and q, say $p < q$. Then $q-1 \mid n-1 = pq-1$, hence $p-1 \equiv pq-1 \equiv 0 \pmod{q-1}$. This contradicts $p < q$. $\diamond$

3.2 Strong Pseudoprimes

For a stronger pseudoprime test we use an additional characteristic property of primes.

Assume that n is odd, but not a prime nor a prime power. Then the residue class ring $\mathbb{Z}/n\mathbb{Z}$ contains non-trivial square roots of 1 besides ± 1. If we find one of these, then we have a proof that n is composite. But how to find non-trivial square roots of 1 when the prime decomposition of n is unknown?

Picking up an idea from Section 2.2 we decompose $n-1$ as

$$n - 1 = 2^s \cdot r \quad \text{with odd } r \tag{1}$$

(and call s the **2-order** of $n-1$). Let $a \in \mathbb{M}_n$. If n fails the pseudoprime test to base a, then it is identified as composite. Otherwise the order of a in the multiplicative group $\mathbb{M}_n$ divides $n-1$. Consider the sequence

$$a^r \bmod n, \quad a^{2r} \bmod n, \quad \ldots, \quad a^{2^s r} \bmod n = 1\,. \tag{2}$$

Possibly already $a^r \equiv 1 \pmod{n}$, and thus the complete sequence consists of 1's. Then we reject a without deciding on n. Otherwise the first 1 occurs at a later position. Then the element before it must be a square root of 1, but $\neq 1$. If we have bad luck, it is -1. In this case again we reject a without a decision. But if we are lucky we have found a non-trivial square root of 1, and identified n as a composite number.

Now let n be an arbitrary positive integer, and assume that $n-1$ is decomposed as in Equation (1). Then (after Selfridge ca 1975) we call n a **strong pseudoprime to base** a, if

$$a^r \equiv 1 \pmod{n} \quad \text{or} \quad a^{2^k r} \equiv -1 \pmod{n} \quad \text{for a } k = 0, \ldots, s-1. \tag{3}$$

Lemma 4 (i) *A prime number is a strong pseudoprime to each base that is not a multiple of this prime.*

(ii) *A pseudoprime to base a is a forteriori a pseudoprime to base a.*

Proof. (i) If n is prime and $a^r \not\equiv 1$, then in the sequence (2) we choose k maximal with $0 \leq k < s$ and $a^{2^k r} \not\equiv 1 \pmod{n}$. Since ± 1 are the only square roots of 1 mod n we conclude $a^{2^k r} \equiv -1 \pmod{n}$.

(ii) The definition (3) immediately yields $a^{n-1} \equiv 1 \pmod{n}$. $\diamond$

Now we face an analoguous situation as in Section 2.3 with $u = n-1$. The set

$$B_u = \bigcup_{t=0}^{s} \{w \in \mathbb{M}_n \mid w^{r \cdot 2^t} = 1,\ w^{r \cdot 2^{t-1}} = -1 \text{ (if } t > 0)\}$$

exactly consists of the bases to which n is a strong pseudoprime, thus has the property $(\mathrm{E}_{n,u})$. These bases are called **prime testimonials** for n.

The CARMICHAEL number $n = 561$ fails the test even with $a = 2$: We have $n - 1 = 560 = 16 \cdot 35$,

$$\begin{aligned} 2^{35} &\equiv 263 \pmod{561}, & 2^{70} &\equiv 166 \pmod{561}, \\ 2^{140} &\equiv 67 \pmod{561}, & 2^{280} &\equiv 1 \pmod{561}. \end{aligned}$$

Hence 561 is unmasked as a composite number since $67 \not\equiv \pm 1$. The smallest composite integer that is a strong pseudoprime to 2, 3, and 5, is $25326001 = 2251 \cdot 11251$. The only composite number $< 10^{11}$ that is a strong pseudoprime to the bases 2, 3, 5, and 7, is 3 215 031 751. This observations make us hope that the strong pseudoprime test is suited for detecting primes.

Proposition 10 *Let $n \geq 3$ be odd. Then the following statements are equivalent:*

(i) *n is prime.*

(ii) *n is a strong pseudoprime to each base a that is not a multiple of n.*

Proof. "(i) $\Longrightarrow$ (ii)": See Lemma 4 (i).

"(ii) $\Longrightarrow$ (i)": By Lemma 4 (ii) n is a prime or satisfies the definition of a CARMICHAEL number, in particular $\lambda(n) \mid n - 1 = u$, and n is squarefree, and a forteriori not a proper prime power. Since $B_u = \mathbb{M}_n$ by assumption, Lemma 1 says that n is a prime power. Hence n is prime. $\diamond$

Corollary 2 *If n is not prime, then the number of bases $< n$ to which n is a strong pseudoprime is at most $\frac{\varphi(n)}{2}$.*

Proof. If n is a CARMICHAEL number, then this follows from Proposition 4. Otherwise $A_u = \{w \in \mathbb{M}_n \mid w^{n-1} = 1\} < \mathbb{M}_n$ is a proper subgroup, and $B_u \subseteq A_u$. $\diamond$

With a little more care we even get the RABIN/MONIER bound $\frac{\varphi(n)}{4}$ (**Exercise**).

3.3 Miller's Primality Test

How can we exploit the criterion for strong pseudoprimes to sufficiently many bases and construct a practically usable test? First we formulate the algorithm for one base a and assess its cost.

Since we anyway compute a^{n-1} by the binary power algorithm it makes sense to compute the complete sequence of powers beginning with a^r in a passing strike. Then the effort is about the same as for the "weak" pseudoprime test alone. Thus the test for strong pseudoprimes to the base a runs as follows:

```
Procedure sPPT(a)
    [Strong pseudoprime test to base $a$ ]
    Input parameters:
        $n$ = the integer to be tested (odd $\geq 3$)
        $a$ = base (in the integer interval $[2 \ldots n-1]$)
    Output parameters:
        compo = a Boolean value with the meaning
            TRUE: $n$ is composite.
            FALSE: The test has no definite result
                [i. e. $n$ is a strong pseudoprime to base $a$].
    Instructions:
        Compute $s$ = 2-order of $n-1$.
        Compute $r$ = odd part of $n-1$.
        Compute $b = a^r \bmod n$ (using the binary power algorithm).
        Set $k = 0$.
        [Loop: entry condition $b = a^{2^k r} \bmod n$
        The Boolean variable 'done', initiated with FALSE, decides
        about repeating the loop.]
        While not done:
            If $b = 1$: set done = TRUE.
                If $k = 0$: set compo = FALSE,
                else: set compo = TRUE. [1 without preceding -1]
            If $b = n-1$ and $k < s$:
                set compo = FALSE, done = TRUE.
            If $k = s$ and $b \neq 1$:
                set compo = TRUE, done = TRUE.
            In all other cases [$k < s, b \neq 1, b \neq n-1$]
                replace $b$ by $b^2 \bmod n$,
                replace $k$ by $k+1$.
```

To assess the cost we break the procedure down into single steps that each multiply two integers mod n. Computing $a^r \bmod n$ takes at most $2 \cdot \log_2(r)$ steps. In each of the up to s loops we compute a square. Since

$\log_2(n-1) = s + \log_2(r)$ we have to compute at most $2 \cdot \log_2(n)$ products $\bmod n$. Each of these squares needs at most N^2 "primitive" integer multiplications where N is the number of places of n in the used representation of the number system. Computing r takes s divisions by 2 that can be neglected. Hence a coarse estimate of the total cost yields $\mathrm{O}(\log(n)^3)$ for a single base.

MILLER's primality test is the sequence of strong peudoprime tests to the bases $2, 3, 4, 5, \ldots$. This doesn't look efficient: In the worst case we test a true prime, then we run through all bases $< n$. However as MILLER showed, significantly less bases suffice—*presupposed that the extended* RIEMANN *hypothesis is true.* In the next section we'll see some explanation but without complete proofs.

3.4 The Extended Riemann Hypothesis (ERH)

A **(complex) character** $\bmod n$ is a function

$$\chi : \mathbb{Z} \longrightarrow \mathbb{C}$$

with the properties:

1. χ has period n.
2. $\chi(xy) = \chi(x)\chi(y)$ for all $x, y \in \mathbb{Z}$.
3. $\chi(x) = 0$ if and only if $\operatorname{ggT}(x, n) > 1$.

The characters $\bmod n$ bijectively correspond to the group homomorphisms

$$\bar{\chi} : \mathbb{M}_n \longrightarrow \mathbb{C}^\times$$

in a canonical way.

Examples are the **trivial character** $\chi(a) = 1$ for all a that are coprime with n, and the Jacobi **character** $\chi(a) = (\frac{a}{n})$ known from the theory of quadratic reciprocity, see Appendix A.5.

A character defines an **L-function** by the Dirichlet series

$$L_\chi(z) = \sum_{a=1}^{\infty} \frac{\chi(a)}{a^z}.$$

This series converges absolutely and locally uniformly in the half-plane $\{z \in \mathbb{C} \mid \operatorname{Re}(z) > 1\}$ because $a^{i \cdot \operatorname{Im}(z)} = e^{i \cdot \ln(a) \cdot \operatorname{Im}(z)}$ has absolute value 1, hence

$$\left| \frac{\chi(a)}{a^z} \right| = \left| \frac{\chi(a)}{a^{\operatorname{Re}(z)} \cdot a^{i \cdot \operatorname{Im}(z)}} \right| = \frac{1}{a^{\operatorname{Re}(z)}} \qquad \text{or} \quad = 0.$$

It admits an analytic continuation to the right half-plane $\operatorname{Re}(z) > 0$ as a holomorphic function, except for the trivial character where 1 is a simple pole.

The function L_χ has the Riemann **property** if all its zeroes in the strip $0 < \operatorname{Re}(z) \leq 1$ are on the line $\operatorname{Re}(z) = \frac{1}{2}$. The Riemann hypothesis states just this property for the Riemann zeta function, the **extended** Riemann **hypothesis (ERH)**, for all L-functions for characters $\bmod n$.

The zeta function is defined for $\operatorname{Re}(z) > 1$ by

$$\zeta(z) := \sum_{a=1}^{\infty} \frac{1}{a^z} = \prod_{p \text{ prime}} \frac{1}{1 - \frac{1}{p^z}}$$

where the last equation is Euler's product formula. Hence for the trivial character $\chi_1 \bmod n$ we have:

$$L_{\chi_1}(z) = \sum_{\gcd(a,n)=1} \frac{1}{a^z} \quad = \quad \zeta(z) \cdot \prod_{p|n \text{ prime}} \left(1 - \frac{1}{p^z}\right);$$

and this L-function has the same zeroes as ζ in $\operatorname{Re}(z) > 0$.

Proposition 11 (Ankeney/Montgomery/Bach) *Let $c = 2/\ln(3)^2 = 1.65707\ldots$. Let χ be a nontrivial character* $\bmod n$ *whose L-function L_χ has the* Riemann *property. Then there is a prime $p < c \cdot \ln(n)^2$ with $\chi(p) \neq 1$.*

We omit the proof.

Corollary 1 *Suppose ERH is true. Let $G < \mathbb{M}_n$ be a proper subgroup. Then there is a prime p with $p < c \cdot \ln(n)^2$ whose residue class* $\bmod\, n$ *is in the complement $\mathbb{M}_n - G$.*

Proof. There exists a nontrivial homomorphism $\mathbb{M}_n/G \longrightarrow \mathbb{C}^\times$, thus a character $\bmod\, n$ with $G \subseteq \ker \chi \subseteq \mathbb{M}_n$. $\diamond$

Proposition 12 (Miller) *Let the integer $n \geq 3$ be odd and a strong pseudoprime to all prime bases $a < c \cdot \ln(n)^2$ with c as in Proposition 11. Assume that the L-function of each character for each divisor of n has the* Riemann *property. Then n is prime.*

Proof. We first show that n is squarefree.

Assume $p^2 \mid n$ for some prime p. The multiplicative group $\mathbb{M}_{p^2}$ is cyclic of order $p(p-1)$. In particular the homomorphism

$$\mathbb{M}_{p^2} \longrightarrow \mathbb{M}_{p^2}, \quad a \mapsto a^{p-1} \bmod p^2,$$

is nontrivial. Its image is a subgroup $G < \mathbb{M}_{p^2}$ of order p, and is cyclic, hence isomorphic with the group of p-th roots of unity in $\mathbb{C}$. The composition of these two homomorphisms yields a character $\bmod\, p^2$. Thus Proposition 11 gives a prime $a < c \cdot \ln(p^2)^2$ with $a^{p-1} \not\equiv 1 \bmod p^2$. The order of a in $\mathbb{M}_{p^2}$ divides $p(p-1)$. Suppose $a^{n-1} \equiv 1 \bmod n$. Then the order also divides $n-1$. Since p is coprime with $n-1$ the order divides $p-1$, contradicting the definition of a. Hence $a^{n-1} \not\equiv 1 \bmod n$, and this in turn contradicts the strong pseudoprimality of n. Therefore n is squarefree.

Next we show that n doesn't have two different prime factors.

Assume p and q are two different prime divisors of n. Denote the 2-order of an integer x by $\nu_2(x)$. We may assume that $\nu_2(p-1) \geq \nu_2(q-1)$. Let

$$r = \begin{cases} p, & \text{if } \nu_2(p-1) > \nu_2(q-1), \\ pq, & \text{if } \nu_2(p-1) = \nu_2(q-1). \end{cases}$$

Again by Proposition 11 there is an $a < c \cdot \ln(r)^2$ with $(\frac{a}{r}) = -1$. If u is the odd part of $n-1$, and $b = a^u$, then also $(\frac{b}{r}) = -1$, in particular $b \neq 1$. By strong pseudoprimality there is a k with $b^{2^k} \equiv -1 \bmod n$. Thus b has order 2^{k+1} in $\mathbb{M}_p$ and in $\mathbb{M}_q$. In particular $2^{k+1} \mid q-1$.

In the case $\nu_2(p-1) > \nu_2(q-1)$ even $2^{k+1} \mid \frac{p-1}{2}$. We conclude $b^{(p-1)/2} \equiv 1 \pmod{p}$, but this contradicts $(\frac{b}{p}) = (\frac{b}{r}) = -1$ by EULER's criterion for quadratic residues.

In the case $\nu_2(p-1) = \nu_2(q-1)$ we have $(\frac{b}{p})(\frac{b}{q}) = (\frac{b}{r}) = -1$. Thus (without restriction) $(\frac{b}{p}) = -1$, $(\frac{b}{q}) = 1$. By EULER's criterion $b^{(q-1)/2} \equiv 1 \pmod{q}$, hence $2^{k+1} \mid \frac{q-1}{2}$, $k+2 \leq \nu_2(q-1) = \nu_2(p-1)$, hence also $b^{(p-1)/2} \equiv 1 \pmod{p}$, contradicting $(\frac{b}{p}) = -1$. $\diamond$

Therefore for MILLER's primality test it suffices to perform the strong pseudoprime test for all prime bases $a < c \cdot \ln(n)^2$. This makes total costs of $\mathrm{O}(\log(n)^5)$.

As an example, for a 512-bit integer, that is $n < 2^{512}$, testing the 18698 primes < 208704 is sufficient. Despite its efficiency this procedure takes some time. Therefore in practice this test is modified in way that is (in a sense yet to specify) not completely exact, but much faster. This is the subject of the next section.

3.5 Rabin's Probabilistic Primality Test

Rabin transferred an idea of Solovay and Strassen to Miller's test. As it later turned out Selfridge had used the method already in 1974.

If we choose a random base a in $[2 \dots n-1]$, then n "in general" fails the strong pseudoprime test to base a except when it is prime. But what means "in general"? How large is the probability? To answer this question we look at the corollary of Proposition 10 where the tighter bound $\frac{1}{4}$ was stated without proof.

Note that the bound $\frac{1}{4}$ is sharp. To see this we consider integers of the form

$$n = (1+2t)(1+4t)$$

with odd t (and assume that $p = 1+2t$ and $q = 1+4t$ are prime—example: $t = 24969$, $p = 49939$, $q = 99877$). Then $n-1 = 2r$ with $r = 3t+4t^2$, and

$$B_u = \{a \mid a^r \equiv 1 \pmod{n}\} \cup \{a \mid a^r \equiv -1 \pmod{n}\}.$$

Since $\gcd(r, p-1) = \gcd(3t+4t^2, 2t) = t = \gcd(r, q-1)$, each of these two congruences has exactly t^2 solutions. Hence $\#B_u = 2t^2$,

$$\frac{\#B_u}{n-1} = \frac{2t^2}{2 \cdot (3t+4t^2)} = \frac{t}{3+4t} = \frac{1}{4+\frac{3}{t}}.$$

However most composite integers don't even come close to this bound $\frac{1}{4}$.

In general assume we are given a family $(M_{(n)})_{n \geq 1}$ of sets $M_{(n)} \subseteq [1 \dots n-1]$ and a real number $\varepsilon \in]0,1[$ with

1. $M_{(n)} = [1 \dots n-1]$ if n is prime,
2. $\#M_{(n)} \leq \varepsilon \cdot (n-1)$ for all sufficiently large odd composite integers n.

Moreover we assume that the property $a \in M_{(n)}$ is efficiently decideable for all $a \in [1 \dots n-1]$, i.e. with costs that grow at most polynomially with $\log(n)$. Then we have a corresponding (abstract) pseudoprime test:

1. Choose a random $a \in [1 \dots n-1]$.
2. Check whether $a \in M_{(n)}$.
3. Output:
   (a) If **no**: n is composite.
   (b) If **yes**: n is pseudoprime to a.

The corresponding **probabilistic primality test** consists of a series of k of these pseudoprime tests to independently chosen bases a (note that this allows for accidental repetitions). If $a \notin M_{(n)}$, we call a a witness for compositeness of n. If always $a \in M_{(n)}$ (we find no witnesses), then n is almost certainly a prime. We may assign an "error probability" δ to this event. This is computed in the following way (no it is *not* $= \varepsilon^k$):

Consider the set of odd r-bit integers, that is odd positive integers $< 2^r$. Let X be the subset of *composite* numbers, and Y_k, the subset of integers that pass the first k of a given series of independent (abstract) pseudoprime tests. The probability that a composite integer makes it into this subset is the conditional probability $P(Y_k|X) \leq \varepsilon^k$.

Nevertheless more important for the practical application is the "converse" probability $\delta = P(X|Y_k)$ that a number n that passed all the tests is still composite. This probability is assessed using Bayes' formula:

$$P(X|Y_k) = \frac{P(X) \cdot P(Y_k|X)}{P(Y_k)} \leq \frac{P(Y_k|X)}{P(Y_k)} \leq \frac{1}{q} \cdot \varepsilon^k \leq r \cdot \ln(2) \cdot \varepsilon^k,$$

where we also used the density of primes estimated by the prime number theorem:

$$P(Y_k) \geq P(\text{prime}) =: q \geq \frac{1}{r \cdot \ln(2)}$$

(the latter inequality being rather tolerant since we consider only odd numbers). Thus the "error probability" $\delta = P(X|Y_k)$ might be larger than ε^k. We can (and should) reduce it by restricting the set we search for primes, thereby enlarging $P(Y_k)$. For example before starting the series of pesudoprime tests we could try to divide by all primes say $< 100r$.

For Rabin's **primality test** we take $M_{(n)}$ as the set of bases n is a strong pseudoprime to, and $\varepsilon = \frac{1}{4}$. If n passes 25 single tests then it is prime with a quite small error probability. The probability that an exact computation produces a false result due to a hardware nor software error is larger than the error probability of Rabin's algorithm. Knuth even doubts whether a future published proof of the extended Riemann hypothesis might ever be as trustworthy. Nevertheless from a mathematical viewpoint we are unsatisfied when we can't be sure that we really found a prime.

For further information on the error probability of a probabilistic primality test read

- S. H. Kim/C. Pomerance: The probability that a random probable prime is composite. Math Comp. 53 (1989), 721–741.
- Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone: *Handbook of Applied Cryptography.* CRC Press, Boca Raton 1997, p. 147.

3.6 RSA and Pseudoprimes

To use RSA we need primes. The probabilistic Rabin primality test solves the problem of finding them in a highly efficient, but not perfectly satisfying way: We could catch a “wrong” prime. What could happen in this case?

For an analysis of the situation let $n = pq$ be a putative RSA module where p and q are not necessarily primes, but at least coprime. For the construction of the exponents d, e with

$$de \equiv 1 \pmod{\lambda(n)} \quad (\text{or} \pmod{\varphi(n)})$$

we use the possibly wrong values

$$\tilde{\varphi}(n) := (p-1)(q-1), \quad \tilde{\lambda}(n) := \mathrm{kgV}(p-1, q-1)$$

instead of the true values $\varphi(n)$ and $\lambda(n)$ of the Euler and Carmichael functions.

How do the RSA algorithms work with the “false” values? Let $a \in \mathbb{Z}/n\mathbb{Z}$ be a plaintext. As usual the case $\gcd(a, n) > 1$ leads to a decomposition of the module, we ignore it because of its extremely low probability. So we assume $\gcd(a, n) = 1$, and ask whether

$$a^{de-1} \overset{?}{\equiv} 1 \pmod{n}$$

holds. By the chinese remainder theorem this holds if and only if

$$a^{de-1} \equiv 1 \pmod{p} \quad \text{and} \pmod{q}.$$

A sufficient condition is

$$a^{p-1} \equiv 1 \pmod{p} \quad \text{and} \quad a^{q-1} \equiv 1 \pmod{q}.$$

Thus a message a might be incorrectly decrypted only if p or q is not a pseudoprime to base a. Hence:

- If instead of a prime factor p we use a Carmichael number, then RSA works correctly despite the “false” parameters, at least if a is coprime with n, though the (extremely low) probability of accidentally factorizing the module n by catching an inept plaintext a is slightly enlarged.

- Otherwise p is not a prime nor a Carmichael number. Then there is a small chance that a message cannot be correctly decrypted.

For this reason many implementations of RSA execute a few trial encryptions and decryptions after generating a key pair relying on the probabilistic Rabin test. But the effect of this additional step simply boils down to a few additional pseudoprime tests. If something goes wrong, the module is rejected.

It is unknown whether this case yet occured in this universe.

3.7 The AKS Primality Test

Miller reduced the quest for an **efficient deterministic** primality test to the extended Riemann hypothesis. In August 2002 the three Indian mathematicians Manindra Agrawal, Neeraj Kayal und Nitin Saxena surprised the scientific community with a complete proof that relied on an astonishingly simple deterministic algorithm. It immediately was baptized "AKS primality test". The fastest known version costs $\mathrm{O}(\log(n)^6)$.

Proposition 13 (Basic criterion) *Let $a, n \in \mathbb{Z}$ be coprime, $n \geq 2$. Then the following statements are equivalent:*

(i) *n is prime.*

(ii) *$(X+a)^n \equiv X^n + a \pmod{n}$ in the polynomial ring $\mathbb{Z}[X]$.*

Proof. From the binomial theorem we have

$$(X+a)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} X^i$$

in $\mathbb{Z}[X]$.

"(i) $\Longrightarrow$ (ii)": If n is prime, then $n|\binom{n}{i}$ for $i = 1, \ldots, n-1$, hence $(X+a)^n \equiv X^n + a^n \pmod{n}$. By Fermat's theorem $a^n \equiv a \pmod{n}$.

"(ii) $\Longrightarrow$ (i)": If n is composite, then we choose a prime $q|n$, and k with $q^k|n$ and $q^{k+1} \not| n$. Then $q \neq n$ and

$$q^k \not| \binom{n}{q} = \frac{n \cdots (n-q+1)}{1 \cdots q}.$$

Hence the coefficient of X^q in $(X+a)^n$ is $\neq 0$ in $\mathbb{Z}/n\mathbb{Z}$. $\diamond$

Remarks

1. Looking at the absolute term in (ii) we see that the basic criterion generalizes Fermat's theorem.
2. Consider the ideal $\mathfrak{q}_r := (n, X^r - 1) \trianglelefteq \mathbb{Z}[X]$ for $r \in \mathbb{N}$. If n is prime, then $(X+a)^n \equiv X^n + a \pmod{\mathfrak{q}_r}$. This shows:

Corollary 1 *If n is prime, then in the polynomial ring $\mathbb{Z}[X]$*

$$(X+a)^n \equiv X^n + a \pmod{\mathfrak{q}_r}$$

for all $a \in \mathbb{Z}$ with $\gcd(a,n) = 1$ and all $r \in \mathbb{N}$.

Applying the basic criterion as a primality test in a naive way would cost about $\log_2 n$ multiplications of polynomials in $\mathbb{Z}/n\mathbb{Z}[X]$ using the binary power algorithm. But these multiplications become more and more expensive, in the last step we have to multiply two polynomials of degree about $\frac{n}{2}$ for an expense of size about n. The corollary bounds the degrees by $r-1$, but its condition is only necessary, not sufficient.

The sticking point of the AKS algorithm is a converse of the corollary that says that we need to try only "few" values of a, however sufficiently many, for a suitable fixed r:

Proposition 14 (AKS criterion, H. W. L<span style="font-variant:small-caps">enstra</span>**'s version)** *Let n be an integer ≥ 2. Let $r \in \mathbb{N}$ be coprime with n. Let $q := \operatorname{ord}_r n$ be the order of n in the multiplicative group $\mathbb{M}_r = (\mathbb{Z}/r\mathbb{Z})^\times$. Furthermore let $s \geq 1$ be an integer with $\gcd(n,a) = 1$ for all $a = 1, \ldots, s$ and*

$$\binom{\varphi(r)+s-1}{s} \geq n^{2d\cdot\lfloor\sqrt{\frac{\varphi(r)}{d}}\rfloor}$$

for each divisor $d|\frac{\varphi(r)}{q}$. Assume

$$(X+a)^n \equiv X^n + a \pmod{\mathfrak{q}} \quad \textit{for all } a = 1, \ldots s$$

with the ideal $\mathfrak{q} = \mathfrak{q}_r = (n, X^r - 1) \trianglelefteq \mathbb{Z}[X]$. Then n is a prime power.

We reproduce the proof by D. Bernstein, breaking it up into a series of lemmas and corollaries.

Lemma 5 *For all $a = 1, \ldots s$ and all $i \in \mathbb{N}$*

$$(X+a)^{n^i} \equiv X^{n^i} + a \pmod{\mathfrak{q}}.$$

Proof. We reason by induction over i. In

$$(X+a)^n = X^n + a + n \cdot f(X) + (X^r - 1) \cdot g(X)$$

we substitute $X \mapsto X^{n^i}$ in $\mathbb{Z}[X]$:

$$\begin{aligned}(X+a)^{n^{i+1}} \equiv (X^{n^i}+a)^n &= X^{n^i\cdot n} + a + n \cdot f(X^{n^i}) + (X^{n^i\cdot r} - 1) \cdot g(X^{n^i}) \\ &\equiv X^{n^{i+1}} + a \pmod{\mathfrak{q}},\end{aligned}$$

since $X^{n^i r} - 1 = (X^r)^{n^i} - 1 = (X^r - 1)(X^{r\cdot(n^i-1)} + \cdots + X^r + 1)$ is a multiple of $X^r - 1$. $\diamond$

Now let $p|n$ be a prime divisor. *Claim*: n is a power of p.

We enlarge the ideal $\mathfrak{q} = (n, X^r - 1) \trianglelefteq \mathbb{Z}[X]$ to $\hat{\mathfrak{q}} := (p, X^r - 1) \trianglelefteq \mathbb{Z}[X]$. Then the identity from Lemma 5 holds also $\bmod\ \hat{\mathfrak{q}}$, and since we now calculate $\bmod\, p$, we even have:

Corollary 2 *For all $a = 1, \dots s$ and all $i, j \in \mathbb{N}$*

$$(X+a)^{n^i p^j} \equiv X^{n^i p^j} + a \pmod{\hat{\mathfrak{q}}}.$$

Let $H := \langle n, p \rangle \leq \mathbb{M}_r$ be the subgroup generated by the residue classes $n \bmod r$ and $p \bmod r$. Let

$$d := \#(\mathbb{M}_r / H) = \frac{\varphi(r)}{\# H}.$$

From $q = \operatorname{ord}_r n \mid \# H$ we have $d \mid \frac{\varphi(r)}{q}$. Hence d satisfies the precondition of Proposition 14. For the remainder of the proof we fix a complete system of representants $\{m_1, \dots, m_d\} \subseteq \mathbb{M}_r$ of $\mathbb{M}_r / H$. Corollary 2 then extends to

Corollary 3 *For all $a = 1, \dots s$, all $k = 1, \dots, d$, and all $i, j \in \mathbb{N}$*

$$(X^{m_k} + a)^{n^i p^j} \equiv X^{m_k n^i p^j} + a \pmod{\hat{\mathfrak{q}}}.$$

Proof. We use the same trick as in Lemma 5 and substitute $X \mapsto X^{m_k}$ in $\mathbb{Z}[X]$:

$$(X+a)^{n^i p^j} = X^{n^i p^j} + a + p \cdot f(X) + (X^r - 1) \cdot g(X) \text{ in } \mathbb{Z}[X],$$

$$(X^{m_k}+a)^{n^i p^j} = X^{m_k n^i p^j} + a + p \cdot f(X^{m_k}) + (X^{m_k \cdot r} - 1) \cdot g(X^{m_k}),$$

and from this the proof is immediate. $\diamond$

The products $n^i p^j \in \mathbb{N}$ with $0 \leq i, j \leq \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor$ are bounded by

$$1 \leq n^i p^j \leq n^{2 \cdot \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor}.$$

The number of such pairs $(i, j) \in \mathbb{N}^2$ is $(\lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor + 1)^2 > \frac{\varphi(r)}{d}$, and all $n^i p^j \bmod r$ are contained in the subgroup H with $\# H = \frac{\varphi(r)}{d}$. Hence there are different $(i, j) \neq (h, l)$ with

$$n^i p^j \equiv n^h p^l \pmod{r}.$$

We even have $i \neq h$—otherwise $p^j \equiv p^l \pmod{r}$, hence $p | r$. Thus we have shown the first part of the following lemma:

Lemma 6 *There exist i, j, h, l with $0 \leq i, j, h, l \leq \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor$ and $i \neq h$ such that for $t := n^i p^j$, $u := n^h p^l$, the congruence $t \equiv u \pmod{r}$ is satisfied, and $|t - u| \leq n^{2 \cdot \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor} - 1$, as well as*

$$(X^{m_k} + a)^t \equiv (X^{m_k} + a)^u \pmod{\hat{\mathfrak{q}}}$$

for all $a = 1, \dots, s$ and all $k = 1, \dots d$.

Proof. The latter congruence follows from $X^t = X^{u+cr} \equiv X^u \pmod{X^r - 1}$, hence

$$(X^{m_k} + a)^t \equiv X^{m_k t} + a \equiv X^{m_k u} + a \equiv (X^{m_k} + a)^u \pmod{\hat{\mathfrak{q}}},$$

for all a and k. $\diamond$

Now r and n are coprime, and p is a prime divisor of n, thus $X^r - 1$ has no multiple zeroes in an algebraic closure of $\mathbb{F}_p$. Hence it has r distinct zeroes, and these are the r-th roots of unity $\bmod p$. They form a cyclic group by Proposition 2. Let ζ be a generating element, that is a primitive r-th root of unity. For one of the irreducible divisors $h \in \mathbb{F}_p[X]$ of $X^r - 1$ we have $h(\zeta) = 0$. Let

$$K = \mathbb{F}_p[\zeta] \cong \mathbb{F}_p[X]/h\mathbb{F}_p[X] \cong \mathbb{Z}[X]/\hat{\hat{\mathfrak{q}}}$$

with the ideal $\hat{\hat{\mathfrak{q}}} = (p, h) \trianglelefteq \mathbb{Z}[X]$. Thus we have an ascending chain of ideals

$$\mathfrak{q} = (n, X^r - 1) \hookrightarrow \hat{\mathfrak{q}} = (p, X^r - 1) \hookrightarrow \hat{\hat{\mathfrak{q}}} = (p, h) \trianglelefteq \mathbb{Z}[X]$$

and a corresponding chain of surjections

$$\mathbb{Z}[X] \longrightarrow \mathbb{Z}[X]/\mathfrak{q} \longrightarrow \mathbb{F}_p[X]/(X^r - 1) \longrightarrow K = \mathbb{F}_p[\zeta] \cong \mathbb{F}_p[X]/h\mathbb{F}_p[X].$$

Lemma 7 *With the notations of Lemma 6 we have in K:*

(i) $(\zeta^{m_k} + a)^t = (\zeta^{m_k} + a)^u$ *for all $a = 1, \ldots, s$ and all $k = 1, \ldots d$.*

(ii) *If $G \leq K^\times$ is the subgroup generated by the $\zeta^{m_k} + a \neq 0$, then $g^t = g^u$ for all $g \in \bar{G} := G \cup \{0\}$.*

Proof. (i) follows from Lemma 6 using the homomorphism $\mathbb{Z}[X] \longrightarrow K$, $X \mapsto \zeta$ with kernel $\hat{\hat{\mathfrak{q}}} \supseteq \hat{\mathfrak{q}}$.

(ii) is a direct consequence from (i). $\diamond$

The $X + a \in \mathbb{F}_p[X]$ for $a = 1, \ldots s$ are pairwise distinct irreducible polynomials since $p > s$ by the premises of Proposition 14. Thus also all products

$$f_e := \prod_{a=1}^{s} (X + a)^{e_a} \quad \text{for } e = (e_1, \ldots, e_s) \in \mathbb{N}^s$$

are distinct in $\mathbb{F}_p[X]$. We consider their images under the map

$$\begin{array}{rcl} \Phi \colon \mathbb{F}_p[X] & \longrightarrow & K^d, \\ f & \mapsto & (f(\zeta^{m_1}), \ldots, f(\zeta^{m_d})) \,. \end{array}$$

Lemma 8 *The images $\Phi(f_e) \in K^d$ of the f_e with $\deg f_e = \sum_{a=1}^{s} e_a \leq \varphi(r) - 1$ are pairwise distinct.*

Proof. Assume $\Phi(f_c) = \Phi(f_e)$. By Corollary 3 for $k = 1, \ldots, d$

$$f_c(X^{m_k})^{n^i p^j} = \prod_{a=1}^{s} (X^{m_k} + a)^{n^i p^j c_a} \equiv \prod_{a=1}^{s} (X^{m_k n^i p^j} + a)^{c_a}$$

$$= f_c(X^{m_k n^i p^j}) \pmod{\hat{\mathfrak{q}}}$$

and likewise

$$f_e(X^{m_k})^{n^i p^j} \equiv f_e(X^{m_k n^i p^j}) \pmod{\hat{\mathfrak{q}}},$$

a forteriori mod $\hat{\hat{\mathfrak{q}}}$. Applying Φ to the left-hand sides yields

$$f_c(X^{m_k n^i p^j}) \equiv f_e(X^{m_k n^i p^j}) \pmod{\hat{\hat{\mathfrak{q}}}}.$$

Thus for the difference $g := f_c - f_e \in \mathbb{F}_p[X]$ we have $g(X^{m_k n^i p^j}) \in h\mathbb{F}_p[X]$ for all $k = 1, \ldots, d$. Let $b \in [1 \ldots r-1]$ be coprime with r, hence represent an element of $\mathbb{M}_r$. Then b is contained in one of the cosets $m_k H$ of $\mathbb{M}_r/H$. Thus there exist k, i, and j with $b \equiv m_k n^i p^j \pmod{r}$. Hence

$$g(X^b) - g(X^{m_k n^i p^j}) \in (X^r - 1)\mathbb{F}_p[X] \subseteq h\mathbb{F}_p[X],$$

hence $g(X^b) \in h\mathbb{F}_p[X]$, and $g(\zeta^b) = 0$. Thus g has the $\varphi(r)$ different zeroes ζ^b in K. But the degree of g is $< \varphi(r)$. Hence $g = 0$, and $f_c = f_e$. ◇

Corollary 4

$$\#\bar{G} \geq \binom{\varphi(r) + s - 1}{s}^{1/d} \geq |t - u| + 1.$$

Proof. There are $\binom{\varphi(r)+s-1}{s}$ options for choosing the exponents $(e_1, \ldots, e_s)$ as in Lemma 8. Since all $\Phi(f_e) \in \bar{G}^d$, we conclude

$$\#\bar{G}^d \geq \binom{\varphi(r) + s - 1}{s} \geq n^{2d \cdot \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor}$$

by the premises of Proposition 14, hence

$$\#\bar{G} \geq n^{2 \cdot \lfloor \sqrt{\frac{\varphi(r)}{d}} \rfloor} \geq |t - u| + 1$$

by Lemma 6. ◇

Now we can complete the proof of Proposition 14: Since $g^t = g^u$ for all $g \in \bar{G} \subseteq K$, the polynomial $X^{|t-u|}$ has more than $|t - u|$ zeroes in K. This is possible only if $t = u$. By the definition of t and u (in Lemma 6) n is a power of p.

This proves Proposition 14. ◇

3.8 The AKS Algorithm

We describe the algorithm in the version given by LENSTRA/BERNSTEIN. It is not trimmed to uttermost efficiency but aims at a transparent proof of polynomiality.

Input

An integer $n \geq 2$.

We measure the length of the input by the number ℓ of bits in the representation of n to base 2,

$$\ell = \begin{cases} \lceil \log_2 n \rceil, & \text{if } n \text{ is not a power of 2,} \\ k+1, & \text{if } n = 2^k. \end{cases}$$

Output

A Boolean value, coded as “COMPOSITE” or “PRIME”.

Step 1

Catch powers of 2:

- If $n = 2$: output “PRIME”, **end**.
- (Else) if n is a power of 2: output “COMPOSITE”, **end**.

  We recognize this case by $\log_2 n$ being an integer.

From now on we may assume that n is not a power of 2, and $\ell = \lceil \log_2 n \rceil$.

Step 2

We precompute a big number $N \in \mathbb{N}$ as

$$N = 2n \cdot (n-1)(n^2-1)(n^3-1)\cdots(n^{4\ell^2}-1) = 2n \cdot \prod_{i=1}^{4\ell^2}(n^i - 1).$$

This number is huge, but more importantly:

- The number $4\ell^2$ of multiplications is polynomial in ℓ.
- From

  $$N \leq 2n \cdot n^{\sum_{i=1}^{4\ell^2} i} = 2n \cdot n^{\frac{4\ell^2(4\ell^2+1)}{2}} \leq 2n \cdot n^{16\ell^4},$$

  we conclude that

  $$k := \lceil \log_2 N \rceil \leq 1 + (16\ell^4 + 1) \cdot \ell$$

  is polynomial in ℓ.

We repeatedly use this integer k in the following. We have $N < 2^k$, and k is the smallest positive integer with this property.

Requirements

We have to find positive integers r and s that satisfy the following requirements:

1. r and n are coprime.
2. The integer interval $[1, \ldots, s]$ contains no prime divisor of n.
3. For each divisor $d \mid \frac{\varphi(r)}{q}$, where $q = \operatorname{ord}_r n$,
$$\binom{\varphi(r)+s-1}{s} \geq n^{2d \cdot \lfloor \frac{\varphi(r)}{d} \rfloor}.$$
4. The primality criterion: For all $a = 1, \ldots, s$
$$(X+a)^n \equiv X^n + a \pmod{(n, X^r - 1)}.$$

Step 3

We choose r as the smallest prime that doesn't divide N. Then r also doesn't divide n. In particular requirement 1 is satisfied.

Why can we find r with polynomial cost?

By one of the extensions of the prime number theorem, equation (2), we have
$$\prod_{p \leq 2k,\, p \text{ prime}} p = e^{\vartheta(2k)} > 2^k > N.$$
Thus not all primes $< 2k$ divide N.

With costs that are at most quadratic in $2k$, and thus polynomial in ℓ, we get the list of all primes $\leq 2k$ (using Eratosthenes' sieve).

Step 4

Set $s := r$. Then requirement 2 is not necessarily satisfied. Hence we run through the list of primes $p < r$ that is known from step 3:

- If $p = n$: Output "PRIME", **end**.

  [This can happen only for "small" n since n grows exponentially with ℓ but r only polynomially.]

- (Else) If $p|n$: Output "COMPOSITE", **end**.

If we reach this point in the algorithm, then s satisfies requirement 2.

Requirement 3

To prove requirement 3 we start with the observation that $q := \operatorname{ord}_r n > 4\ell^2$.

> Otherwise $n^i \equiv 1 \pmod r$ for some i with $1 \le i \le 4\ell^2$, hence $r \mid n^i - 1 \mid N$, contradiction.

Now assume d divides $\frac{\varphi(r)}{q}$. Then

$$\begin{aligned} d &\le \frac{\varphi(r)}{q} < \frac{\varphi(r)}{4\ell^2}, \\ 2d \cdot \lfloor\sqrt{\frac{\varphi(r)}{d}}\rfloor &\le 2d \cdot \sqrt{\frac{\varphi(r)}{d}} = \sqrt{4d\varphi(r)} < \frac{\varphi(r)}{\ell} < \frac{\varphi(r)}{{}^2\log n}, \\ n^{2d\cdot\lfloor\sqrt{\frac{\varphi(r)}{d}}\rfloor} &< n^{\frac{\varphi(r)}{{}^2\log n}} = 2^{\varphi(r)}. \end{aligned}$$

On the other hand $\varphi(r) \ge 2$, so

$$\binom{\varphi(r)+s-1}{s} = \binom{\varphi(r)+r-1}{r} = \binom{2\varphi(r)}{\varphi(r)+1} \ge 2^{\varphi(r)}.$$

Hence requirement 3 is satisfied.

Step 5

Next we check requirement 4,

$$(X+a)^n \equiv X^n + a \pmod{(n, X^r - 1)}$$

in a loop for $a = 1, \ldots, r$. The number of iterations is at most r, thus $\le 2k$, hence polynomial in ℓ. During each iteration we have two binary power computations, hence a total of at most 4ℓ multiplications, the factors being polynomials of degree $< r$—polynomial in ℓ—with coefficients of size $< n$, hence of bitlength polynomial in ℓ.

- If an a violates requirement 4, then output "COMPOSITE", **end**.

Otherwise all a satisfy requirement 4, therefore n is a prime power by the AKS criterion.

Step 6

Finally we must decide whether n is a proper prime power. Since the primes $\le r$ don't divide n, we only have to check in a loop for t with $1 < t < \log_r n$:

- If $\sqrt[t]{n}$ is integer: Output "COMPOSITE", **end**.

The number of iterations is $\le \ell$, and the test in each single iteration also takes polynomial cost, if we compute $\lfloor\sqrt[t]{n}\rfloor$ by a binary search in the interval $[1 \ldots n-1]$.

- If the algorithm reaches this point, output “PRIME”, **end**.

This completes the proof of:

Theorem 1 *The AKS algorithm decides the primality of* n *with costs that depend polynomially on* $\log n$.

Chapter 4

The Discrete Logarithm with Cryptographic Applications

Computing discrete logarithms is believed—like factoring large integers—to be a hard problem. This serves as basis of many cryptographic procedures.

A useful aspect of most of these procedures is that they rely only on the group property of the multiplicative groups of the residue class rings of integers. Therefore they often have an immediate translation to other groups such as elliptic curves. Should discrete logarithms for residue class rings happen to be efficiently computable there remains a chance that the procedures remain secure for other groups.

4.1 The Discrete Logarithm

Let G be a group (multiplicatively written) and $a \in G$ be an element of order s (maybe ∞). Then the **exponential function** to base a in G

$$\exp_a \colon \mathbb{Z} \longrightarrow G, \quad x \mapsto a^x,$$

is a group homomorphism (since $a^{x+y} = a^x a^y$) and has period s (since $a^{x+s} = a^x a^s = a^x$ if $s < \infty$). By the homomorphy theorem the induced homomorphism h

$$\begin{tikzcd}
\mathbb{Z} \arrow[r, "\exp_a"] \arrow[d] & \langle a \rangle \subseteq G \arrow[dl, shift left, "\log_a"] \\
\mathbb{Z}/s\mathbb{Z} \arrow[ur, shift left, "h"] &
\end{tikzcd}$$

is an isomorphism, hence has an inverse map

$$\log_a \colon \langle a \rangle \longrightarrow \mathbb{Z}/s\mathbb{Z}$$

defined on the cyclic subgroup $\langle a \rangle \subseteq G$, the **discrete logarithm** to base a that is an isomorphism of groups. [The case $s = \infty$ fits into this scenario for $s\mathbb{Z} = 0$ and $\mathbb{Z}/s\mathbb{Z} = \mathbb{Z}$.]

We apply this to the multiplicative group $\mathbb{M}_n$: For an integer $a \in \mathbb{Z}$ with $\gcd(a, n) = 1$ the exponential function $\bmod n$ to base a,

$$\exp_a \colon \mathbb{Z} \longrightarrow \mathbb{M}_n, \quad x \mapsto a^x \bmod n,$$

has period $s = \operatorname{ord} a | \lambda(n) | \varphi(n)$. The inverse function

$$\log_a \colon \langle a \rangle \longrightarrow \mathbb{Z}/s\mathbb{Z}$$

is the discrete logarithm $\bmod n$ to base a.

We know of no efficient algorithm that computes the discrete logarithm $\log_a$ for large $s = \operatorname{ord} a$, or to invert the exponential function—not even a probabilistic one.

> **Informal definition:** A function $f : M \longrightarrow N$ is called **one-way function** if for "almost all" images $y \in N$ there is no efficient way to compute a pre-image $x \in M$ with $f(x) = y$.
>
> This definition can be given a mathematically precise (although not completely satisfying) formulation in terms of complexity theory, see Appendix B.

Discrete logarithm assumption: The exponential function $\exp_a \bmod n$ is a one-way function for “almost all” bases a.

Note that this is an unproven conjecture.

The most important special case is a prime module $p \geq 3$, and a primitive element $a \in [2, \ldots, p-2]$, i. e., $\operatorname{ord} a = p - 1$.

$$\begin{tikzcd}
\mathbb{Z} \arrow[r, "\exp_a"] \arrow[d] & \mathbb{F}_p^\times \arrow[dl, shift left, "\log_a"] \\
\mathbb{Z}/(p-1)\mathbb{Z} \arrow[ur, shift left, "\text{bij}"] &
\end{tikzcd}$$

To make the computation of discrete logarithms hard in practice we have to choose a prime module p of about the same size as an RSA module. Thus according to the state of the art 1024-bit primes are completely obsolete, 2048-bit primes are safe for short-time applications only.

The book by Shparlinski (see the references for these lecture notes) contains some lower bounds for the complexity of discrete logarithm computations in various computational models.

4.2 Diffie-Hellman Key Exchange

We treat some exemplary applications that provide astonishingly elegant solutions for seemingly unsolvable problems under the discrete logarithm assumption.

Imagine A (Alice) and B (Bob) want to exchange a key for a symmetric cipher. In 1976 Diffie and Hellman proposed the following protocol whose security relies on the dicrete logarithm assumption:

1. A and B (publicly) agree on a prime p and a primitive element a mod p.
2. A generates a random integer x, computes $u = a^x \bmod p$, and sends u to B.
3. B generates a random integer y, computes $v = a^y \bmod p$, and sends v to A.
4. A computes $k = v^x \bmod p$, and B computes $k = u^y \bmod p$.

Now A and B share a secret k that may be used as key. The fact that A and B compute the same key k lies in the equation

$$v^x \equiv a^{xy} \equiv u^y \pmod{p}.$$

An eavesdropper can intercept the values p, a, u, and v. But this doesn't enable her to efficiently compute k, or x, or y.

This protocol realizes a kind of hybrid encryption. A difference with a "proper" asymmetric cipher concerns the need for synchronization between A and B, preventing spontaneous messages (for example by e-mail that follows an asynchroneous protocol).

An attacker who is able to efficiently compute discrete logarithms is also able to efficiently break the Diffie-Hellman protocol. It is unknown whther the converse also holds.

The British Secret Service CESC knew the procedure already in 1974 but of course kept it secret.

Here is a mathematical model for a somewhat more abstract protocol:

1. A and B (publicly) agree on a set X, an element $a \in X$, and a commutative subsemigroup $H \subseteq \mathrm{Map}(X, X)$.
2. A chooses a random map $\varphi_A \in H$, computes $u = \varphi_A(a)$, and sends u to B.
3. B chooses a random map $\varphi_B \in H$, computes $v = \varphi_B(a)$, and sends v to A.
4. A computes $\varphi_A(v)$, and B computes $\varphi_B(u)$.

Then A and B share the secret value

$$k = \varphi_A(v) = \varphi_A(\varphi_B(a)) = \varphi_B(\varphi_A(a)) = \varphi_B(u)$$

and may use it as key for their secret communication—at least if an attacker has no method to derive φ_A, φ_B, or k from the entities X, a, u, and v she knows or intercepts.

For the adaption of this protocol to elliptic curves an even more abstract scenario is useful that is visualized by a commutative diagram as follows:

$$\begin{tikzcd}
X \arrow[r, "\varphi_A"] \arrow[d, "\varphi_B"'] & Y \arrow[d, "\tilde{\varphi}_B"] \\
Y \arrow[r, "\tilde{\varphi}_A"'] & Z
\end{tikzcd}$$

4.3 The Man in the Middle

In this section we consider a communication protocol with asymmetric encryption, and note that the same attack works against the Diffie-Hellman key exchange. The basic problem is that an attacker can plant his own key into the procedure. In some more detail:

Suppose A = Alice and B = Bob want to exchange messages. First A sends her public key E_A to B, and B sends his public key E_B to A.

The attacker E = Eve who only listens cannot use these public data for eavesdropping. However the attacker M = Mallory, the "man in the middle" who actively forges messages, intercepts the key exchange, and each time replaces the intercepted public key by his own key E_M. From now on M is able to monitoring and even counterfeiting the complete communication of A and B. Figure 4.1 illustrates the attack.

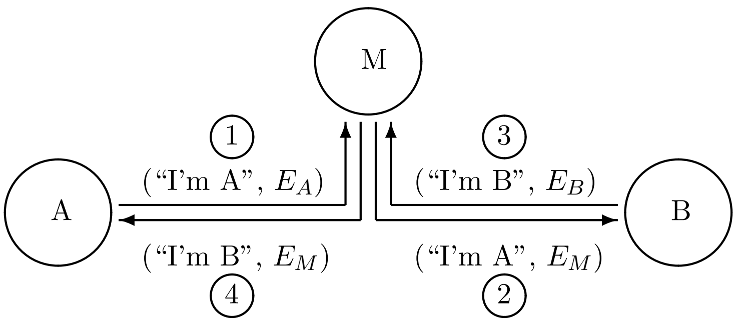


Figure 4.1: The man in the middle

There are different ways to prevent this attack. But all of them make asymmetric encryption more complex. The usual way is the use of certificates: The public keys of all participants of a communication network get a digital signature by a "trusted third party".

Definition. A certificate is a public key signed by a trusted third party.

Mnemonic. *A key exchange can be secure from the man in the middle only if the partners are mutually authenticated.*

Exercise. What information in the Diffie-Hellman protocol is suited to be used in a certificate?

4.4 Secret Communication without Key Exchange

Even without exchanging keys in advance a confidential conversation is possible. (Note that this protocol also is not secure from the man in the middle.)

An analogy from veryday life illustrates the idea:

- Alice puts her message in a box and locks it with a padlock whose key is hers and not available to anyone else.
- Of course Bob is unable to open the box. Instead he locks it with another padlock of his own. He returns the doubly locked box to Alice.
- Alice removes her padlock and returns the box that is locked with Bob's padlock only.
- Bob removes his padlock, opens the box, and reads the message.

This cryptographic protocol is called the Massey-Omura scheme or Shamir's no-key algorithm. It may be implemented with the discrete exponential function. Its security relies on the discrete logarithm assumption:

The procedure uses a public large prime number p. Alice and Bob each choose a pair of exponents d and e with $ed \equiv 1 \pmod{p-1}$, hence $a^{de} \equiv a \pmod{p}$ for all integers $a \in \mathbb{Z}$. Each one keeps *both* of their exponents secret.

Then Alice sends a message a to Bob according to the following protocol:

$$\begin{tikzcd}
\textcircled{\textbf{A}} & & \textcircled{\textbf{B}} \\
\boxed{a^{e_A} \bmod p} \arrow[rr] & & {} \\
{} & & \boxed{a^{e_A e_B} \bmod p} \arrow[ll] \\
\boxed{a^{e_A e_B d_A} = a^{e_B} \bmod p} \arrow[rr] & & {} \\
{} & & \boxed{a^{e_B d_B} = a \bmod p}
\end{tikzcd}$$

An attacker who is able to compute discrete logarithms is also able to compute the exponent e_B from the intercepted ciphertexts $a^{e_A} \bmod p$ and $a^{e_A e_B} \bmod p$. From this she computes d_B by congruence division and solves for a.

This is the only known attack. Hence the protocol is secure from Eve as long as the discrete logarithm assumption holds. To be secure from Mallory the protocol must be supplemented by an authentication phase.

4.5 ElGamal Cipher—Idea

The ElGamal cipher is an asymmetric cipher—or more precisely a hybrid cipher—that also relies on the complexity of the discrete logarithm.

The basic public parameters are a prime p and an element $g \in [2 \ldots p-2]$. The order of g in $\mathbb{F}_p^\times$ should be high, preferably g should be a primitive element $\bmod p$.

> p and g may be shared by all participants but also may be individually chosen.

Each participant chooses a random integer

$$d \in [2 \ldots p-2]$$

as private key, und computes

$$e = g^d \bmod p$$

as corresponding public key. Computing d from e is computing a discrete logarithm, hence presumably hard.

The definitioon of the cipher needs one more idea: How to transform a message a in such a way that it can be reconstructed only with knowledge of d?

> The naive idea of sending $e^a = g^{da} \bmod p$ is useless—knowing d doesn't help with decrypting a. Also sending $r = g^a \bmod p$ is useless—the receiver can compute $r^d = e^a \bmod p$ but not a.

The idea is to first generate a message key to be used with a hybrid procedure:

- Alice chooses a random $k \in [2 \ldots p-2]$. As key she will use $K = e^k \bmod p$ where e is the Bob's public key, thus Alice can compute K.
- To share the key K with Bob Alice sends the *key information* $r = g^k \bmod p$ together with the encrypted message.
- Bob computes $r^d = g^{kd} = e^k = K \bmod p$ using his private key d.

As symmetric component of the hybrid encryption the shift cipher in $\mathbb{F}_p^\times$ is used with K as one-time key. So Alice has to generate a new key K for each plaintext block and to send the corresponding key information, doubling the length of the message.

Thus, after generating the key K and the key information r:

- the formula for encryption is $c = Ka \bmod p$,
- and the message to be sent is (c, r).

Bob computes the key K from r, and then decrypts

- $a = K^{-1}c \bmod p$ by congruence division.

4.6 Computing Discrete Logarithms

The classical algorithm for computing discrete logarithms is the **index calculus** by ADLEMAN—“index” was GAUSS’ denotation of the discrete logarithm.

Let $p \geq 3$ be a prime and a be a primitive element for p.

The naive algorithm for computing $\log_a y$ for $y \in \mathbb{F}_p^\times$ is punished by exponentially growing costs, as usual. It computes $a, a^2, a^3, \ldots$ in order until x with $a^x = y$ is found. In the mean it needs $\frac{p}{2} - 1$ trials, in the worst case, $p - 2$ (omitting the trivial value $y = 1$).

Preliminary Steps

For given p and a we need to execute this precomputation only once.

Let $p_1 = 2$, $p_2 = 3$, …, p_k be the first k primes.

If we randomly choose an exponent r, then it could happen that $a^r \bmod p$—considered as integer $\in \mathbb{Z}$—has only prime divisors in $\{p_1, \ldots, p_k\}$. After h strokes of luck we have a system of h equations:

$$\begin{aligned} a^{r_1} \bmod p &= p_1^{\alpha_{11}} \cdots p_k^{\alpha_{1k}}, \\ &\vdots \\ a^{r_h} \bmod p &= p_1^{\alpha_{h1}} \cdots p_k^{\alpha_{hk}}. \end{aligned}$$

in $\mathbb{Z}$ and a forteriori in $\mathbb{F}_p$. Taking logarithms results in a system of linear equations over the ring $\mathbb{Z}/(p-1)\mathbb{Z}$ for the k unknowns $\log_a p_i$:

$$\begin{aligned} r_1 &= \alpha_{11} \cdot \log_a p_1 + \cdots + \alpha_{1k} \cdot \log_a p_k, \\ &\vdots \\ r_h &= \alpha_{h1} \cdot \log_a p_1 + \cdots + \alpha_{hk} \cdot \log_a p_k. \end{aligned}$$

From Chapter I we know efficient algorithms for solving it. If h is sufficiently large—at least $h \geq k$—, then we can compute $\log_a p_1$, …, $\log_a p_k$.

The random search for “strokes of luck” makes the precomputation probabilistic.

Computation

Let $y \in \mathbb{F}_p^\times$ be given. We want to compute $\log_a y$.

For a randomly chosen exponent s it could happen that

$$y \cdot a^s \bmod p = p_1^{\beta_1} \cdots p_k^{\beta_k}$$

in $\mathbb{Z}$. Then we easily compute

$$\log_a y = \beta_1 \cdot \log_a p_1 + \cdots + \beta_k \cdot \log_a p_k - s\,.$$

This observation reduces the computation of the discrete logarithm of any element to the computation for the elements of the **factor basis** $(p_1, \ldots, p_k)$. This reduction is also probabilistic.

Variants

The presented approach has several variants that result in different running times. They vary in the choice of the factor basis—that might be adapted to y and need not consist of the first primes without gap—and in the strategy of choosing the exponents r and s.

The fastest known variant uses a number field sieve such as applied for factoring large integers and has expenses of

$$\approx e^{c \cdot \sqrt[3]{\log p \cdot (\log\log p)^2}},$$

the same order of magnitude as is needed for factoring an integer of the same size. By the state of the art 1024-bit primes are insecure, and 2048-bit primes secure only for short-term cryptographic applications.

As an oddity we mention that the "Secure NFS" protocol deployed by SUN used a 192-bit prime (58 decimal places) even in the 1990s.

Special Primes

There are reasons to choose p as a special prime of the form $p = 2p' + 1$ with p' prime:

1. Some algorithms are very fast if $p - 1$ has only small prime divisors. This argument is no longer considered as solid since the advantage of special algorithms over the current versions of the number field sieve is only small. Moreover the probability of choosing such a "bad" prime by accident is extremely small.

2. Finding a primitive element is easy, see Section A.9 in the appendix.

Chapter 5

Hard Number Theoretic Problems

The following table gives an overview over cryptologically relevant number theoretic computational problems. "Efficient" means "computable with polynomial cost", "ERH" means "if the extended RIEMANN hypothesis holds", "prob." means "by a probabilistic algorithm".

Computational problem	**efficient?**	**treated in**
Primality test	yes	3.1–3.8
For a prime number p		
Finding a primitive element	yes (ERH or prob.)	A.2, A.9
Finding a quadratic nonresidue	yes (ERH or prob.)	A.8
Quadratic residuosity	yes	A.6
Taking a square root	yes (ERH or prob.)	follows in 5.3
Discrete logarithm	? (probably no)	4.1, 4.6
For a composite integer n		
Factoring	? (probably no)	2.2, 2.4
RSA inversion (e-th root)	? (probably no)	2.2
Computation of EULER function	? (probably no)	2.2
Computation of CARMICHAEL f.	? (probably no)	2.2
Finding a primitive element	? (probably no)	A.4
Quadratic residuosity	? (probably no)	A.11
Taking a square root	? (probably no)	follows in 5.5
Discrete logarithm	? (probably no)	follows in 5.1

Figure 5.1 shows the connection between computational problems for a composite integer n. An arrow from A to B indicates that problem B reduces to Problem A by an efficient (maybe probabilistic) algorithm. For an unidirectional arrow the reverse direction is unknown. Reductions indicated by red arrows will be proved in this chapter (sometimes only in the case

where n is a product of two primes). [The task denoted by "Pol. fact." means factoring polynomials in one variable over the residue class ring $\mathbb{Z}/n\mathbb{Z}$. We'll not treat it in these lecture notes.]

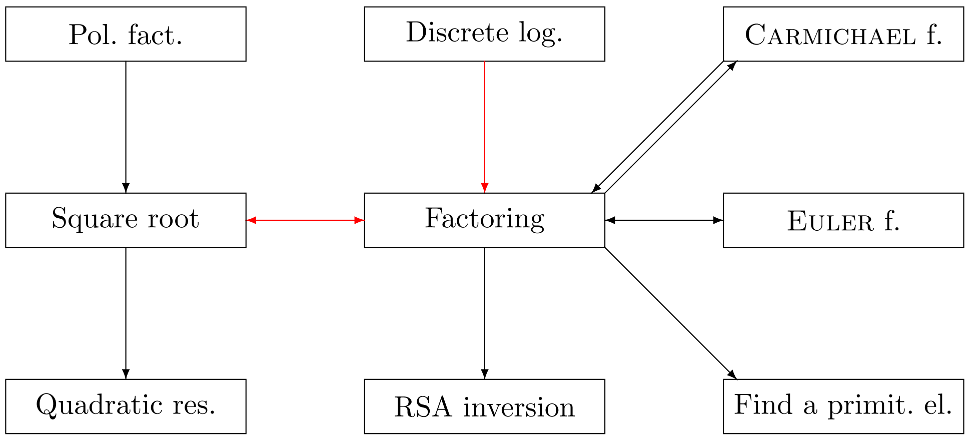


Figure 5.1: Connection between computational problems for a composite module

5.1 Discrete Logarithm and Factorization

Let $a \in \mathbb{M}_n$, $\operatorname{ord} a = s$, and consider the exponential function

$$\exp_a : \mathbb{Z} \longrightarrow \mathbb{M}_n$$

The problem of computing discrete logarithms $\bmod\ n$ is to find an algorithm that for each $y \in \mathbb{M}_n$

- outputs "no" if $y \notin \langle a \rangle$,
- else outputs an $r \in \mathbb{Z}$ with $0 \le r < s$ and $y = a^r \bmod n$.

Proposition 15 (E. Bach) *Let $n = pq$ with different primes $p, q \geq 3$. Then factoring n admits a probabilistic efficient reduction to the computation of discrete logarithms* $\bmod n$*.*

Proof. We have $\varphi(n) = (p-1)(q-1)$. For a randomly chosen $x \in \mathbb{M}_n$ always $x^{\varphi(n)} \equiv 1 \pmod{n}$. Let $y := x^n \bmod n$, thus

$$y \equiv x^n \equiv x^{n-\varphi(n)} = x^{pq-(p-1)(q-1)} = x^{p+q-1} \pmod{n}.$$

The discrete logarithm yields an r with $0 \leq r < \operatorname{ord} x \leq \lambda(n)$ and $y = x^r \bmod n$. Hence

$$x^{r-(p+q-1)} \equiv 1 \pmod{n}, \quad \operatorname{ord} x \mid r - (p+q-1).$$

Since $|r - (p+q-1)| < \lambda(n)$ the probability is high that $r = p+q-1$. This happens for example if $\operatorname{ord} x = \lambda(n)$. Otherwise choose another x.

From the two equations

$$\begin{aligned} p + q &= r + 1 \\ p \cdot q &= n \end{aligned}$$

we easily compute the factors p and q. $\diamond$

5.2 Square Roots and Factorization

Proposition 16 (M. Rabin) *Let $n = pq$ with different primes $p, q \geq 3$. Then factoring n admits a probabilistic efficient reduction to taking square roots* $\bmod n$.

Proof. $\mathbb{Z}/n\mathbb{Z}$ contains four different roots of unity, hence also four different square roots of each square in $\mathbb{M}_n$.

For a random choice of $x \in M_n$ the square root algorithm provides a root $y \in \mathbb{M}_n$ of x^2, thus

$$y^2 \equiv x^2 \pmod{n}.$$

The probability that $y \not\equiv \pm x \pmod{n}$ is $\frac{1}{2}$. Since

$$n \mid (x^2 - y^2) = (x+y)(x-y), \quad n \not| (x \pm y),$$

$\gcd(n, x+y)$ is a proper divisor of n. $\diamond$

Therefore an attacker who knows how to take square roots $\bmod n$ also can factorize n.

The reverse direction follows in Section 5.5.

5.3 Square Roots in Finite Prime Fields

In many cases taking square roots is a trivial task as the following simple consideration shows:

Lemma 9 *Let G be a finite group of odd order m. Then for each $a \in G$ there is exactly one $x \in G$ with $x^2 = a$, and it is given by $x = a^{\frac{m+1}{2}}$.*

Proof. Since $a^m = 1$ we have $x^2 = a^{m+1} = a$. We conclude that the squaring map $x \mapsto x^2$ is surjective, hence a bijection $G \longrightarrow G$. $\diamond$

We search methods for taking square roots in a finite prime field $\mathbb{F}_p$ as efficiently as possible. The case $p \equiv 3 \pmod 4$ is extremely simple by the foregoing consideration: If $p = 4k + 3$, then the group $\mathbb{M}_p^2$ of quadratic residues has odd order $\frac{p-1}{2} = 2k + 1$. Hence for a quadratic residue $z \in \mathbb{M}_p^2$ the unique square root is $x = z^{k+1} \bmod p$ [Lagrange 1769]. The cost of taking this square root is at most $2 \cdot \log_2(p)$ congruence multiplications.

Examples

1. For $p = 7 = 4 \cdot 1 + 3$ we have $k + 1 = 2$. By A.8 2 is a quadratic residue. A square root is $2^2 = 4$. Check: $4^2 = 16 \equiv 2$.

2. For $p = 23 = 4 \cdot 5 + 3$ we have $k + 1 = 6$. By A.8 again 2 is a quadratic residue. A square root is $2^6 = 64 \equiv 18$. Check: $18^2 \equiv (-5)^2 = 25 \equiv 2$.

Unfortunately for $p \equiv 1 \pmod 4$ we cannot hope for such a simple procedure. For example -1 is a quadratic residue, but no power of -1 can be a square root of -1 since always $[(-1)^m]^2 = (-1)^{2m} = 1 \neq -1$.

Fortunately there are general procedures, for example one that is baptized AMM after Adleman, Manders, and Miller, but was described already by Cipolla in 1903. It starts by decomposing $p - 1$ into $p - 1 = 2^e \cdot u$ with odd u. Furthermore we choose (once and for all) an arbitrary quadratic nonresidue $b \in \mathbb{F}_p^\times - \mathbb{M}_p^2$—this is the only nondeterministic step in the algorithm, see Section A.8. (Assuming ERH the procedure is even deterministic, as it is in the many cases where a quadratic nonresidue is known anyway.)

Now we consider a quadratic residue $z \in \mathbb{M}_p^2$ and want to find a square root of it. Since $z \in \mathbb{M}_p^2$, we have $\operatorname{ord}(z) \mid \frac{p-1}{2}$, hence the 2-order $r = \nu_2(\operatorname{ord}(z))$ of $\operatorname{ord}(z)$ is bounded by $\leq e - 1$, and r is minimal with $z^{u2^r} \equiv 1$.

We recursively define a sequence $z_1, z_2, \ldots$ beginning with

$$z_1 = z \quad \text{with } r_1 = \nu_2(\operatorname{ord}(z_1)).$$

If $z_i \in \mathbb{M}_p^2$ is chosen, and r_i is the 2-order of $\operatorname{ord}(z_i)$, then the sequence terminates if $r_i = 0$. Otherwise we set

$$z_{i+1} = z_i \cdot b^{2^{e-r_i}}.$$

Then $z_{i+1} \in \mathbb{M}_p^2$. Furthermore

$$z_{i+1}^{u \cdot 2^{r_i - 1}} \equiv z_i^{u \cdot 2^{r_i-1}} \cdot b^{u \cdot 2^{e-1}} \equiv 1,$$

since the first factor is $\equiv -1$ due to the minimality of r_i, and the second factor is $\equiv (\frac{b}{p}) = -1$, for $u \cdot 2^{e-1} = \frac{p-1}{2}$. Hence $r_{i+1} < r_i$. The terminating condition $r_n = 0$ is reached after at most e steps with $n \le e \le \log_2(p)$.

Then we compute reversely:

$$x_n = z_n^{\frac{u+1}{2}} \bmod p$$

with $x_n^2 \equiv z_n^{u+1} \equiv z_n$ (since $\operatorname{ord}(z_n) \mid u$ by its odd parity). Recursively

$$x_i = x_{i+1} / b^{2^{e-r_i-1}} \bmod p$$

that by induction satisfies

$$x_i^2 \equiv x_{i+1}^2 / b^{2^{e-r_i}} \equiv z_{i+1} / b^{2^{e-r_i}} \equiv z_i.$$

Hence $x = x_1$ is a square root of z.

In addition to the cost of finding b we count the following steps:

- Computing the powers $b^2, \ldots, b^{2^{e-1}}$, costing $(e-1)$ modular squares.
- Computing the powers $b^u, b^{2u}, \ldots, b^{2^{e-1}u}$, taking at most $2 \cdot \log_2(u) + e - 1$ congruence multiplications.
- Computing z^u, taking at most $2 \cdot \log_2(u)$ congruence multiplications.
- Furthermore we compute for each $i = 1, \ldots, n \le e$:
  - z_i by one congruence multiplication,
  - z_i^u from z_{i-1}^u by one congruence multiplication,
  - $z_i^{u2^r}$ from $z_{i-1}^{u2^r}$ by one congruence multiplication,
  - and then r_i.

  This makes a total of at most $3 \cdot (e-1)$ congruence multiplications.
- x_n as a power by at most $2 \cdot \log_2(u)$ congruence multiplications.
- x_i from x_{i+1} each by one congruence division with cost $\mathrm{O}(\log(p)^2)$.

Summing up we get costs of size about $\mathrm{O}(\log(p)^3)$ with a rather small constant coefficient.

Example Let $p = 29$ and $z = 5$. Then $p - 1 = 4 \cdot 7$, hence $e = 2$ and $u = 7$. By the remarks above $b = 2$ is a quadratic nonresidue. We compute the powers

$$b^2 = 4, \quad b^u \equiv 128 \equiv 12, \quad b^{2u} \equiv 144 \equiv -1,$$

$$z^2 \equiv 25 \equiv -4, \quad z^4 \equiv 16, \quad z^6 \equiv -64 \equiv -6, \quad z^7 \equiv -30 \equiv -1.$$

Now

$$z_1 = 5, \quad z_1^u \equiv -1, \quad z_1^{2u} \equiv 1, \quad r_1 = 1,$$

$$z_2 \equiv z_1 b^2 \equiv 5 \cdot 4 = 20, \quad z_2^u \equiv z_1^u b^{2u} \equiv (-1)(-1) = 1, \quad r_2 = 0.$$

Now we go backwards:

$$x_2 \equiv z_2^{\frac{u+1}{2}} = z_2^4 = (z_2^2)^2 \equiv 400^2 \equiv (-6)^2 = 36 \equiv 7,$$

$$x_1 = x_2/b \bmod p = 7/2 \bmod 29 = 18.$$

Hence $x = 18$ is the wanted root. Check: $18^2 = 324 \equiv 34 \equiv 5$.

Exercises Find deterministic algorithms (= simple formulas) for taking square roots in the fields

- $\mathbb{F}_p$ with $p \equiv 5 \pmod 8$
- $\mathbb{F}_{2^m}$ with $m \geq 2$ [Hints: 1. Consider the order of the radicand in the multiplicative group. 2. Invert the linear map $x \mapsto x^2$.]
- $\mathbb{F}_q$ for $q = p^m$

Alternative algorithms: Almost all known efficient algorithms that completely cover the case $p \equiv 1 \pmod 4$ are probabilistic and have a deterministic variant whose cost is polynomial assuming ERH. The book by FORSTER (*Algorithmische Zahlentheorie*) has a variant of the CIPOLLA/AMM algorithm that uses the quadratic extension $\mathbb{F}_{p^2} \supseteq \mathbb{F}_p$ and is conceptionally quite simple. The *Handbook of Applied Cryptography* (MENEZES/VAN OORSCHOT/VANSTONE) contains an algorithm by TONELLI 1891 that admits a concise formulation, but cost $\mathrm{O}(\log(p)^4)$. Another method is a special case of the CANTOR/ZASSENHAUS algorithm for factoring polynomials over finite fields, see VON ZUR GATHEN/GERHARD: *Modern Computer Algebra.* Yet another procedure by LEHMER uses the LUCAS sequence (a_n) with $a_1 = b$, $a_2 = b^2 - 2z$, where $b^2 - 4z$ is a quadratic nonresidue. The only known deterministic algorithm with proven polynomial cost was given by SCHOOF. It uses elliptic curves, and costs $\mathrm{O}(\log(p)^9)$, so it is of theoretical interest only.

For overviews see:

- E. Bach/ J. Shallit: *Algorithmic Number Theory.* MIT Press, Cambridge Mass. 1996.
- D. J. Bernstein: Faster square roots in annoying finite fields. Preprint (siehe die Homepage des Autors `http://cr.yp.to/`).

5.4 Square Roots for Prime Power Modules

A simple procedure (implicitly using HENSEL's lifting) allows to extend the square root algorithms from prime modules to prime powers. Let p be a prime $\neq 2$, and let $e \geq 2$. Let z be a quadratic residue $\bmod p^e$. We want to find a square root of z.

Of course z is also a quadratic residue $\bmod p^{e-1}$. Assume we already have found a root for it, that is a y with $y^2 \equiv z \pmod{p^{e-1}}$. Let

$$a = 1/(2y) \bmod p$$

and $y^2 - z = p^{e-1} \cdot u$. We set

$$x := y - a \cdot (y^2 - z) \bmod p^e.$$

Then we have

$$\begin{aligned} x^2 &\equiv y^2 - 2ay(y^2 - z) + a^2(y^2 - z)^2 \equiv y^2 - 2ayp^{e-1}u \\ &\equiv y^2 - p^{e-1}u = z \pmod{p^e}. \end{aligned}$$

Hence x is a square root of z mod p^e.

We wont explicit this algorithm but illustrate it with two examples:

Examples

1. $n = 25$, $z = 19$. We have $p = 5$, $19 \bmod 5 = 4$. Hence we can take $y = 2$ and $a = 1/4 \bmod 5 = 4$. Then $y^2 - z = -15$ and

   $$x = 2 + 15 \cdot 4 \bmod 25 = 62 \bmod 25 = 12.$$

   Check: $12^2 = 144 = 125 + 19$.

2. $n = 27$, $z = 19$. We have $p = 3$, $19 \bmod 3 = 1$. Hence in the first step we can take $y = 1$ and $a = 1/2 \bmod 3 = 2$. Then $y^2 - z = -18$ and

   $$x = 1 + 2 \cdot 18 \bmod 9 = 37 \bmod 9 = 1.$$

   For the second step (from 9 to 27) again $y = 1$, $y^2 - z = -18$, and

   $$x = 37 \bmod 27 = 10.$$

   Check: $10^2 = 100 = 81 + 19$.

The costs consist of two contributions:

1. One square root $\bmod p$ and one division. (The quotient a needs to be computed only once since $x \equiv y \pmod{p}$.)

2. Each time the exponent is incremented we execute two congruence multiplications and two subtractions.

Hence the total cost is $\mathrm{O}(\log(n)^3)$ for the module n.

Finally we have to consider the case where $n = 2^e$ is a power of two.

For $e \leq 3$ the only quadratic residue is 1, its square root is 1.

For larger exponents e we have again a recurrence to $e - 1$: Let z be an odd integer (all invertible elements are odd). Assume we already found a y with $y^2 \equiv z \pmod{2^{e-1}}$. Then $y^2 - z = 2^{e-1} \cdot t$. If t is even, then $y^2 \equiv z \pmod{2^e}$. Otherwise we set $x = y + 2^{e-2}$. Then

$$x^2 \equiv y^2 + 2^{e-1}y + 2^{2e-4} \equiv z + 2^{e-1} \cdot (t + y) \equiv z \pmod{2^e},$$

since $t + y$ is even. Hence $x = y$ or $y + 2^{e-2}$ is a square root of z. Here the cost is even smaller than $\mathrm{O}(\log(n)^3)$.

By the way we have shown that z is a quadratic residue $\bmod 2^e$ (for $e \geq 3$) if and only if $z \equiv 1 \pmod 8$.

5.5 Square Roots for Composite Modules

If we know the prime decomposition of the module n, then we can efficiently compute square roots in $\mathbb{M}_n$. The two tasks "factoring" and "computing square roots" are equivalent with respect to their complexity.

For an execution of the procedure we successively decompose n into coprime factors (down to the prime powers).

So let $n = rs$ with coprime factors r and s. First we compute coefficients a and b such that $ar + bs = 1$ using the extended Euclidean algorithm.

We want to find a square root of z. Let u be a square root $\bmod r$ and v be a square root $\bmod s$. Then $x := arv + bsu \bmod n$ satisfies the congruences:

$$\begin{aligned} x &\equiv bsu \equiv u \pmod{r}, & x &\equiv arv \equiv v \pmod{s}, \\ x^2 &\equiv u^2 \equiv z \pmod{r}, & x^2 &\equiv v^2 \equiv z \pmod{s}, \end{aligned}$$

hence $x^2 \equiv z \pmod{n}$.

The cost for this procedure is two square roots modulo the factors, one Euclidean algorithm, and four congruence multiplications (+ 1 congruence addition). Hence it is $\mathrm{O}(\log(n)^3)$.

For Blum integers (see Appendix A.11) we even have a simpler algorithm, namely an explicit formula:

Corollary 1 *Let $n = pq$ with primes $p, q \equiv 3 \pmod{4}$. Then*

(i) $d = \frac{(p-1)(q-1)+4}{8}$ *is an integer.*

(ii) *For each quadratic residue $x \in \mathbb{M}_n^2$ the power x^d is the (unique) square root of x in $\mathbb{M}_n^2$.*

Proof. (i) If $p = 4k+3$, $q = 4l+3$, then $(p-1)(q-1) = 16kl + 8k + 8l + 4$, hence $d = 2kl + k + l + 1$.

(ii) The exponent of the multiplicative group $\mathbb{M}_n$,

$$\lambda(n) = \mathrm{kgV}(p-1, q-1) = 2 \cdot \mathrm{kgV}(2k+1, 2l+1)$$

is a divisor of $2 \cdot (2k+1) \cdot (2l+1)$, The exponent of the subgroup $\mathbb{M}_n^2$ of squares is $\frac{\lambda(n)}{2}$, hence a divisor of $(2k+1)\cdot(2l+1) = 4kl+2k+2l+1 = 2d-1$. Thus $x^{2d} \equiv x \pmod{n}$ for all $x \in \mathbb{M}_n^2$, thus the square of x^d is x. $\diamond$

This simple formula has the effect that the Rabin cipher is especially easy to handle for Blum integer modules.

Chapter 6

Equivalences of Basic Cryptographic Functions

In real world applications the basic cryptographic functions

1. Symmetric ciphers:
   (a) bitblock ciphers
   (b) bitstream ciphers
2. Asymmetric ciphers
3. Keyless ciphers:
   (a) one-way functions
   (b) hash functions
4. Random generators:
   (a) physical random generators
   (b) (algorithmic) (pseudo-) random generators
5. Steganographic procedures

are used for the construction of cryptographic protocols. We'll see that the existence of most of them—1a, 1b, 3a, 3b, 4b in suitable variants—is equivalent with the basic problem of theoretic computer science $\mathbf{P} \overset{?}{\neq} \mathbf{NP}$, and thus lacks a proof.

Warning: Most parts of this section are *mathematically inexact*. Statements on complexity are formulated in the naive way and justified by heuristic arguments. Then we sketch the approach to formalizing them by TURING machines. However this model turns out as insufficient for cryptology. The mathematically sound versions of complexity results for cryptologic procedures are given in Appendix B.

6.1 One-Way Functions

We continue to use the informal definition from 4.1. An exact approach is given in Appendix B.2.

Application A natural application of one-way functions is one-way encryption. This means:

- *Everyone* can encrypt.
- *No one* can decrypt.

What is it good for if no one can decrypt? There are several meaningful applications for one-way functions, in particular for the special case of hash functions, see 6.2:

- Password management, for instance in Unix or MS Windows. No one must be able to read the password. But the operating system must be able to compare an entered password with the one it has in its data base in encrypted form. **("cryptographic matching")**
- A similar application is **pseudonymization**: Data of a person should be combined with data from the same person stored elsewhere or at other times without revealing the identity of this person.
- Another application is making **digital signatures** faster, see 6.2.
- The crucial property of asymmetric encryption is that nobody can derive the private key from the public one. However the direct naive application of one-way functions doesn't work, as we saw already for the ElGamal cipher in 4.5.

Examples of conjectured one-way functions:

1. The discrete exponential function, see 4.1.
2. Consider a bitblock cipher

$$F: M \times K \longrightarrow C$$

that resists an attack with known plaintext. A standard trick to get a one-way function $f: K \longrightarrow C$ from it works as follows:

$$f(x) := F(m_0, x).$$

In words: We take a fixed plaintext m_0—maybe the all-zero block—and encrypt it with a key that is exactly the block x to be one-way encrypted. Inverting this function amounts to an attack with known plaintext m_0 on the cipher F.

3. Let $n \in \mathbb{N}$ be a composite module. From 5.2 we know that—at least in the case where n is the product of two large prime numbers—computing square roots $\bmod n$ is probably hard. Hence the squaring map $x \mapsto x^2 \bmod n$ is a probable one-way function of the residue class ring $\mathbb{Z}/n\mathbb{Z}$. Note that calculating the inverse map is possible with additional information in form of the prime factors of n. Such an additional information is called a "trapdoor". The function is then called a "trapdoor one-way function". This is the crucial security feature of the Rabin cipher.
4. The same conclusion holds for the RSA function $x \mapsto x^e \bmod n$ with an exponent e that is coprime with $\lambda(n)$ (oder $\varphi(n)$).

6.2 Hash Functions

Hash functions are the most important special cases of one-way functions. They are also known as “message digests” or “cryptographic check sums”.

Definition 1 Let Σ be an alphabet and $n \in \mathbb{N}$ be a fixed integer ≥ 1. A one-way function

$$h \colon \Sigma^* \longrightarrow \Sigma^n$$

is called **weak hash function** over Σ.

It maps character strings of *arbitrary* lengths to character strings of a given *fixed* length. (Since Σ^* is infinite we interpret the one-way property as: the restriction of h to Σ^r is one-way for all sufficiently large r.

Definition 2 A one-way function $f \colon M \longrightarrow N$ is called **collision free** if there is no efficient way to find $x_1, x_2 \in M$ with $x_1 \neq x_2$, but $f(x_1) = f(x_2)$.

This is a kind of “virtual injectivity”. Needless to say that true injective one-way functions are collision free. If $\#M > \#N$, then f cannot be injective, but nevertheless could be collision free.

Definition 3 A **(strong) hash function** is a collision free weak hash function.

For practical applications (mostly with $\Sigma = \mathbb{F}_2$) the length n of the hash values should be as small as possible. On the other hand to exclude efficient invertibility, and thus to get cryptographic security, n must be sufficiently large. We want a weak hash function to deliver uniformly distributed values that look statistically random, and to be safe from an exhaustion attack as illustrated in Figure 6.1. Inserting m blanks at will we generate 2^m different—but optically indistinguishable—versions of a text document. If m is large enough, with high probability one of these versions will have the given hash value.

row 1	(add blank)	$\rightarrow$ 2 different versions
⋮	⋮	
row i	(add blank)	$\rightarrow$ 2 different versions
	⋮	⋮
row m	(add blank)	$\rightarrow$ 2 different versions

Figure 6.1: An exhaustion attack: How to fake a document to have a given hash value by generating 2^m different versions

As a consequence $n = 80$ is just too weak as a lower bound, we'd better use 128-bit hashes. This is the hash length of the well-known but outdated functions MD2, MD4, MD5.

But virtually all applications even need collision free hashes. Remember the birthday paradox, see I.2.6: To exclude collisions with sufficient certainty we need about twice the bitlength than for the one-way property. So hash values of 160 bits are just below the limit. The former standard hash functions SHA-1 and RIPEMD use exactly this length. Their use is strongly discouraged. In the context of AES the hash function SHA-2 with at least 256-bit values was specified, conveniently also denoted as SHA-256 etc. [see `http://csrc.nist.gov/publications/`]. The new standard SHA-3 is valid since 2015.

In fact for the MDx functions there is a systematic way to find collisions [Dobbertin 1996ff.], also SHA-1 collisions are known (2005).

document a		document b	
row 1	(add blank)	row 1	(add blank)
$\vdots$	$\vdots$	$\vdots$	$\vdots$
row m	(add blank)	row m	(add blank)

Figure 6.2: A collision attack: How to fake a document to have the same hash value as another document

Applications

(Strong) hash functions are in use for

- digital signatures: To sign a long message with the private key would take much time due to the slowness of asymmetric ciphers. The standard procedure is to sign a hash of the message.

  For security we need a collision free hash function. Otherwise the attacker could get a valid signature for an arbitrary document a without stealing Alice's private key: He produces an innocently looking document b that Alice is glad to sign. Then he fabricates $q = 2^m$ variants $a_1, \ldots, a_q$ and $b_1, \ldots, b_q$ of both documents, for example by inserting spaces at m different positions. If he finds a collision $h(a_i) = h(b_j)$, he lets Alice sign b_j, getting a valid signature for a_i too.

- transforming a long, but memorizable passphrase ("Never change a working % password 24 because you'll ? forget the nEW one+") into an n-bit key (`BA8C0C8C1C65364F` in hexadecimal notation) for a symmetric cipher.

6.3 Conversion Tricks

We give heuristic reasons that the following statements (A) to (D) are equivalent, and that each of them implies (E)—for a formal mathematical proof we don't have yet the exact definitions.

These implications also have practical relevance for constructing a basic function given another one. A coarse summary—for the discussion on regulations of cryptography that pop up from time to time—consists of the statements

- Who wants to prohibit encryption also must prohibit hash functions and pseudorandom generators.
- Who wants to make cryptography impossible must prove that $\mathbf{P} = \mathbf{NP}$.

(A) There is a one-way function $f\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$.

($\tilde{\mathbf{A}}$) There is a one-way function $\tilde{f}\colon \mathbb{F}_2^{2n} \longrightarrow \mathbb{F}_2^n$.

(B) There is a weak hash function $h\colon \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^n$.

(C) There is a strong symmetric cipher $F\colon \mathbb{F}_2^n \times \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^n$ (where "strong" means secure under a known-plaintext attack).

(D) There is a perfect pseudorandom generator $\sigma\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^{p(n)}$.

(E) $\mathbf{P} \neq \mathbf{NP}$.

Remark 1 Making the statements precise in terms of complexity theory we have to state (A) – (D) for families of functions that are parametrized by n.

Remark 2 A pseudorandom generator is perfect if for unknown $x \in \mathbb{F}_2^n$, given some bits of the output $\sigma(x)$, there is no efficient way to predict some more bits of the output, or to compute x. In the specification p is a polynomial with integer coefficients—from a "seed" of length n the generator produces $p(n)$ bits.

We omit reasoning about the implication "(D) $\Longrightarrow$ (E)".

"(C) $\Longrightarrow$ (D)": Set $\sigma(x) = (s_1, \ldots, s_{p(n)/n})$ with $s_0 := x$ and $s_i := F(s_{i-1}, z)$ for $i \geq 1$, where the key z is a secret constant parameter. Note the similarity with the OFB mode for bitblock ciphers. For no block s_i of the sequence the attacker is able to determine the previous block s_{i-1}—otherwise the cipher wouldn't be secure. It is not obvious that this property suffices to show perfectness, we'll show this in Chapter IV.

"(D) $\Longrightarrow$ (C)": Consider the bitstream cipher that uses $\sigma(x)$ as bitstream and x as key.

"(A) $\Longrightarrow$ (C)": There is a simple approach by E. Backus: Set $F(a,k) = a + f(k)$. Under a known-plaintext attack a and $c = F(a,k)$ are known. Hence also $f(k) = c - a$ is known. So the attack reduces to inverting f.

[Other approaches: MDC (= Message Digest Cryptography) by P. Gutmann, or the Feistel scheme.]

"(C) $\Longrightarrow$ (A)": See the example in Section 6.1.

"(A) $\Longrightarrow$ ($\tilde{\text{A}}$)": Define $\tilde{f}$ by $\tilde{f}(x,y) := f(x+y)$. Assume we can compute a pre-image (x,y) of c for $\tilde{f}$. Then this gives also the pre-image $x+y$ of c for f.

"($\tilde{\text{A}}$) $\Longrightarrow$ (B)": Pad $x \in \mathbb{F}_2^*$ with (at most $n-1$) zeroes, giving $(x_1, \ldots, x_r) \in (\mathbb{F}_2^n)^r$. Then set

$$\begin{aligned} c_0 &:= 0, \\ c_i &:= \tilde{f}(c_{i-1}, x_i) \quad \text{for } 1 \le i \le r, \\ h(x) &:= c_r. \end{aligned}$$

This defines $h\colon \mathbb{F}_2^* \longrightarrow \mathbb{F}_2^n$.

Let $y \in \mathbb{F}_2^n$ be given. Assume the attacker finds a pre-image $x \in (\mathbb{F}_2^n)^r$ with $h(x) = y$. Then she also finds a $z \in (\mathbb{F}_2^n)^2$ with $\tilde{f}(z) = y$, namely $z = (c_{r-1}, x_r)$ (where $y = c_r$ in the construction of h).

"(B) $\Longrightarrow$ (A)": Restricting h to $\mathbb{F}_2^n$ also gives a one-way function.

6.4 Physical Complexity

The obvious approach to assessing the complexity of an algorithm is counting the primitive operations that a customary processor executes, or, more exactly, counting the clock cycles. This would lead to concrete results like: "Computing ... costs at least (say) 10^{80} of the following steps: ...". For example we could count elementary arithmetical operations (additions, multiplications, ...), taking into account the word size of the processor (e. g. 32 bits) and the number of clock cycles for the considered operations. [Note that this number might not be uniquely defined on a modern CPU with pipeline architecture.]

For many concrete algorithms statements of this kind are possible, and often lead to interesting mathematical problems as abundantly demonstrated by D. Knuth in his books.

Unfortunately no flavour of complexity theory yields results on the *minimum* number of steps that *each* algorithm for solving a certain problem must execute, except for extremely simple problems like evaluating a polynomial for a certain argument. If we knew results of this kind, we could mathematically prove the security of cryptographic procedures without recurring to unproven conjectures or heuristic arguments.

This kind of reasoning could take into account physical bounds that limit the resources computers in this universe can dispose of. A known estimate of this kind was proposed by Louis K. Scheffer in `sci.crypt`:

- Our universe contains at most 10^{90} elementary particles. This is certainly an upper bound for the number of available CPUs.
- Passing an elementary particle with the speed of light takes at least 10^{-35} seconds. This is certainly a lower bound for the time required by a single operation.
- Our universe has a life span of at most 10^{18} seconds ($\approx 30 \times 10^9$ years). This is certainly an upper bound for the available time.

Multiplying these bounds together we conclude that at most $10^{143} \approx 2^{475}$ operations can be executed in our universe. In particular 500-bit keys are secure from exhaustion ...

> ... until such time as computers are built from something other than matter, and occupy something other than space. (Paul Ciszek)

Note that this security bound holds for the one algorithm "exhaustion". It has no relevance for the security of even a single cryptographic procedure! (As long as there is no proof that no attack is faster than exhaustion.)

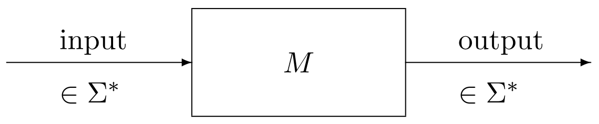
Needless to say that a realistic upper bound is smaller by several orders of magnitude.

For comparision we list some cryptologically relevant quantities:

seconds/year	3×10^7
CPU cycles/year (1 GHz CPU)	3.2×10^{16}
age of our universe (years)	10^{10}
CPU cycles since then (1 GHz)	3.2×10^{26}
atoms of the earth	10^{51}
electrons in our universe	8.37×10^{77}
ASCII strings of length 8 (95^8)	6.6×10^{15}
binary strings of length 56 (2^{56})	7.2×10^{16}
binary strings of length 80	1.2×10^{24}
binary strings of length 128	3.4×10^{38}
binary strings of length 256	1.2×10^{77}
primes with 75 decimal places (about 250 bits)	5.2×10^{72}

6.5 Turing Machines

The mathematical results of complexity theory consist almost exclusively of asymptotic cost estimates, and in almost all cases these estimates are upper bounds. Complexity theory in its various flavours relies on diverse models of computation. In this section we shortly sketch the common formalism by Turing machines.



Here Σ (as usual) denotes a finite alphabet. The input is a finite string on a tape that is infinite in both directions. The Turing machine M can assume states from a finite set that also contains a state "halt". Depending on the state the machine executes certain operations, for instance reads one character from the tape, changes its state, writes one character to the tape, moves the reading head by one position to the left or to the right. If M reaches the state "halt", then the current string on the tape is the output.

Let $L \subseteq \Sigma^*$ be a language. If M reaches the "halt" state after a finite number of steps for all inputs $x \in L$, then we say that M **accepts the language** L. If $f\colon L \longrightarrow \Sigma^*$ is a function, and M reaches "halt" after finitely many steps for each $x \in L$ with output $f(x)$, then we say that M **computes** f.

With some effort, and not too overwhelming elegance, we can describe all algorithms by Turing machines. Then by counting the steps we may express their complexities in the form: for input x the machine M takes τ_x steps until reaching "halt".

Usually we consider "worst case" complexity. Let $L_n := L \cap \Sigma^n$. Then the function

$$t_M\colon \mathbb{N} \longrightarrow \mathbb{N}, \quad t_M(n) := \max\{\tau_x \mid x \in L_n\},$$

is called **(time) complexity** of the Turing machine M (for L).

The subset **P** ("polynomial time") of the set of all functions from L to Σ^* consists of the functions $f\colon L \longrightarrow \Sigma^*$ for which there exists a Turing machine M and an integer $k \in \mathbb{N}$ such that

(i) M computes f,

(ii) $t_M(n) \le n^k$ for almost all $n \in \mathbb{N}$.

Remark Equivalent with (ii) is the statemant: There is a polynomial $p \in \mathbb{N}[X]$ with $t_M(n) \le p(n)$ for all $n \in \mathbb{N}$.

For if there is such a polynomial $p = a_r X^r + \cdots + a_0$ (with $a_r \neq 0$), then

$$\begin{array}{rcl} a_r n^r & \geq & a_{r-1} n^{r-1} + \cdots + a_0 \quad \text{for } n \geq n_0, \\ p(n) & \leq & 2a_r n^r \quad \text{for } n \geq n_0, \\ p(n) & \leq & n^{r+1} \quad \text{for } n \geq n_1 = \max\{2a_r, n_0\}. \end{array}$$

Conversely if $t_M(n) \leq n^k$ for $n \geq n_0$, then we choose $c \in \mathbb{N}$ with $t_M(n) \leq c$ for the finitely many $n = 0, \ldots, n_0 - 1$. Then $t_M(n) \leq p(n)$ for all $n \in \mathbb{N}$ with $p = X^k + c$.

Analogously we define the set **EXPTIME** ("exponential time"): f is in **EXPTIME** if there exist a TURING machine M, an integer $k \in \mathbb{N}$, and real numbers $a, b \in \mathbb{R}$ with

(i) M computes f,

(ii) $t_M(n) \leq a \cdot 2^{bn^k}$ for almost all $n \in \mathbb{N}$.

Obviously $\mathbf{P} \subseteq$ **EXPTIME**.

Examples with $\Sigma = \mathbb{F}_2$.

1. Assume
$$L := \{(p, z) \in \mathbb{N}^2 \mid p \text{ prime } \equiv 3 \pmod 4,\ z \in \mathbb{M}_p^2\}$$
is coded as a subset of Σ^* by a suitable binary representation. Let $f(p, z) =$ the square root of z mod p, likewise coded as an element of Σ^*. Then $f \in \mathbf{P}$ by 5.3.

2. Let $L = \mathbb{N}_2$ be the set of integers ≥ 2 (binary coded). Let $f(x) =$ be the smallest prime factor of x. Then $f \in$ **EXPTIME** since we can try all the integers $\leq \sqrt{x} \leq 2^{n/2}$.

   Presumably $f \notin \mathbf{P}$.

3. The **knapsack problem**. Here
$$L = \{(m, a_1, \ldots, a_m, N) \mid m, a_1, \ldots, a_m, N \in \mathbb{N}\}$$
with suitable binary encoding,
$$f(m, a_1, \ldots, a_m, N) = \begin{cases} 1, & \text{if there is } S \subseteq \{1, \ldots, m\} \\ & \text{with } \sum_{i \in S} a_i = N, \\ 0 & \text{otherwise.} \end{cases}$$
Then $f \in$ **EXPTIME** since we can try all of the 2^m subsets $S \subseteq \{1, \ldots, m\}$.

   Presumably $f \notin \mathbf{P}$.

6.6 The Class NP

We say that the TURING machine M computes $f\colon L \longrightarrow \Sigma^*$ **nondeterministically** if for each $x \in L$ there is a $y \in \Sigma^*$ such that M, given the concatenation xy of x and y as input, reaches “halt” after finitely many steps with output $f(x)$.

Example Let $\Sigma = \mathbb{F}_2$ and $L = \{(n, a, x) \in \mathbb{N}^3 \mid n \geq 2, a, x \in \mathbb{M}_n\}$. Let $f = \log_a \bmod n$ be the discrete logarithm.

For a given x let y be the logarithm of x—it doesn't matter in the definition from where we get the logarithm, in any case it exists. All the TURING machine M has to do is to check whether $a^y = x$. Then it writes y to the tape and halts.

General idea A candidate y for the solution is provided, M only does a check.

Alternative idea An unbounded number of *parallel* TURING machines each checks a different $y \in \Sigma^*$.

The set **NP** (“nondeteministic polynomial time”) is defined as the set of all functions for which there exists a TURING machine M and an integer $k \in \mathbb{N}$ with:

(i) M computes f nondeterministically,

(ii) $t_M(n) \leq n^k$ for almost all $n \in \mathbb{N}$.

We have the inclusions

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXPTIME}.$$

The first of these is trivial, the second is a theorem that we don't prove here.

The most important unsolved problem of theoretical computer science is the conjecture

$$\mathbf{P} \overset{?}{\neq} \mathbf{NP}.$$

Likewise unproven is the conjecture

$$\mathbf{NP} \overset{?}{\neq} \mathbf{EXPTIME}.$$

On the other hand the statement

$$\mathbf{P} \neq \mathbf{EXPTIME},$$

is proven, if only by constructing “artificial” problems. There is no known “natural” problem proven to be in the difference set.

By the way we cannot make cryptanalysis of a cipher more difficult than **NP**: Exhaustion—that is trying all keys with a known plaintext—is always possible, and the encryption function must be efficient, hence in **P**.

Examples

1. If f is the discrete logarithm as above, then $f \in \mathbf{NP}$.
2. Likewise factoring integers is in **NP**.
3. Also the knapsack problem is in **NP**.

We call the function f **NP-complete** if for each Turing machine M that computes f (deterministically!) and each function $g \in \mathbf{NP}$ there exists a Turing machine N that computes g and an integer $k \in \mathbb{N}$ such that

$$t_N(n) \leq t_M(n)^k \quad \text{for almost all } n \in \mathbb{N}.$$

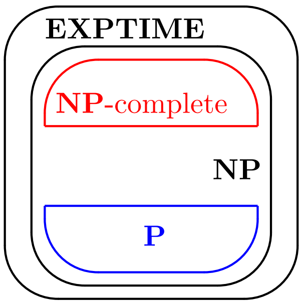
In other words the complexity of N is at most polynomial in the complexity of M.

Interpretation **NP**-complete problems are the maximally complex ones among those in **NP**.

It is known that **NP**-*complete problems exist.* We refrain from proving this theorem here.

For instance the knapsack problem is **NP**-complete, as is the determination of roots of (polynomial) functions $p\colon \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$. Factoring integers is presumably not **NP**-complete.

Should $\mathbf{P} = \mathbf{NP}$ hold—nobody believes it—, then all functions in **NP** would be **NP**-complete. If not, the following drawing illustrates the relative situation of the complexity classes:



Appendix A

Primitive Elements and Quadratic Residues

This mathematical appendix treats in a closed form some number theoretic subjects that play a major role for cryptology. They relate to the multiplicative group of a residue class ring.

As we saw in the main text several results on the security of cryptographic procedures depend on the non-existence of efficient algorithms for some tasks.

Relevant problems and their (incomplete) solutions are:

1. Find a primitive element.
    - The complexity of the general case is unknown.
    - Exhaustion is efficient if ERH holds.
    - There is a much more efficient probabilistic algorithm, that however doesn't even terminate in the worst case.
    - For many prime modules the solution is trivial.
    - Proving primitivity is efficient if the prime factors of the order of the multiplicative group are known. Otherwise the complexity is unknown.
    - For a composite module the problem reduces to its prime factors—if these are known.
2. Decide on quadratic residuosity.
    - For prime modules there is an efficient algorithm.
    - For a composite module the problem reduces to its prime factors—if these are known.
    - For composite modules with unknown prime factors the complexity is unknown. Presumably the problem is hard (as hard as prime decomposition).

3. Find a quadratic non-residue.
    - The complexity of the general case is unknown.
    - Exhaustion is efficient if ERH holds.
    - There is an efficient probabilistic algorithm, that however doesn't even terminate in the worst case.
    - For most primes the solution is trivial.
    - For a composite module the problem reduces to its prime factors—if these are known.

A related problem, finding square roots in residue class rings, is treated in Chapter 5.

A.1 Primitive Elements for Powers of 2

The cases $n = 2$ or 4 are trivial: $\mathbb{M}_2$ is the one-element group. $\mathbb{M}_4$ is cyclic of order 2, thus $3 \equiv -1 \pmod 4$ is primitive.

From now on we assume $n = 2^e$ with $e \geq 3$. Note that $\mathbb{M}_n$ consists of the residue classes of the odd integers, hence $\varphi(n) = 2^{e-1}$.

Lemma 10 *Let $n = 2^e$ with $e \geq 2$.*

(i) *If a is odd, then*

$$a^{2^s} \equiv 1 \pmod{2^{s+2}} \quad \textit{for all } s \geq 1.$$

(ii) *If $a \equiv 3 \pmod 4$, then $n \mid 1 + a + \cdots + a^{n/2-1}$.*

Proof. (i) First we prove the statement for $s = 1$. In the case $a = 4q + 1$ we have $a^2 = 16q^2 + 8q + 1$. In the case $a = 4q + 3$ we have $a^2 = 16q^2 + 24q + 9$, hence $a^2 \equiv 1 \pmod 8$.

The assertion for general s follows by induction:

$$a^{2^{s-1}} = 1 + t2^{s+1} \Longrightarrow a^{2^s} = (a^{2^{s-1}})^2 = 1 + 2t2^{s+1} + t^2 2^{2s+2}.$$

(ii) By (i) we have $2n = 2^{e+1} \mid a^{n/2} - 1$. Since only the first power of 2 divides $a - 1$ we conclude

$$n = 2^e \mid \frac{a^{n/2} - 1}{a - 1}$$

as claimed. $\diamond$

Lemma 11 *Let p a prime and e an integer with $p^e \geq 3$. Let p^e be the largest power of p that divides $x - 1$. Then p^{e+1} is the largest power of p that divides $x^p - 1$.*

Proof. We have $x = 1 + tp^e$ with an integer t that is not a multiple of p. The binomial theorem yields

$$x^p = 1 + \sum_{k=1}^{p} \binom{p}{k} t^k p^{ke}.$$

Since p divides all binomial coefficients $\binom{p}{k} = \frac{p!}{k!(p-k)!}$ for $k = 1, \ldots, p-1$ we can factor out p^{e+1} from the sum:

$$x^p = 1 + tp^{e+1}s$$

with some integer s. Hence p^{e+1} divides $x^p - 1$. It remains to show that s is not a multiple of p. We take a closer look at s:

$$\begin{aligned} s &= \sum_{k=1}^{p} \frac{1}{p}\binom{p}{k} \cdot t^{k-1} p^{e(k-1)} \\ &= 1 + \frac{1}{p}\binom{p}{2} \cdot tp^e + \cdots + \frac{1}{p} \cdot t^{p-1} p^{e(p-1)}. \end{aligned}$$

Since $p^e \geq 3$ we have $e(p-1) \geq 2$, hence $s \equiv 1 \pmod{p}$. ◇

Lemma 10 implies

$$a^{2^{e-2}} \equiv 1 \pmod{n} \quad \text{for all odd } a.$$

Hence the exponent $\lambda(n) \leq 2^{e-2}$, and $\mathbb{M}_n$ is not cyclic. More exactly:

Proposition 17 *Let $n = 2^e$ with $e \geq 3$. Then:*

(i) *The order of -1 in $G = \mathbb{M}_n$ is 2, the order of 5 is 2^{e-2}, and G is the direct product of the cyclic groups generated by -1 and 5.*

(ii) *If $e \geq 4$, then the primitive elements $\bmod\, n$ are the integers $a \equiv 3,\ 5 \pmod{8}$. Their number is $n/4$.*

Proof. (i) Since $\operatorname{ord} 5 \mid 2^e$ and $\operatorname{ord} 5 \leq 2^{e-2}$, we conclude that $\operatorname{ord} 5$ is a power of 2 and $\leq 2^{e-2}$.

Now 2^2 is the largest power of 2 in $5 - 1$, thus 2^3 is the largest power of 2 in $5^2 - 1$ (by Lemma 11). Successively we conclude that 2^{e-1} is the largest power of 2 in $5^{2^{e-3}} - 1$. Hence the 2^{e-2}-th power of 5 is the smallest one $\equiv 1 \pmod{2^e}$.

The product of the two subgroups is direct since -1 is not a power of 5—otherwise $5^k \equiv -1 \pmod{n}$, and, because of $e \geq 2$, also $5^k \equiv -1 \pmod{4}$, contradicting $5 \equiv 1 \pmod{4}$.

The direct product is all of G since its order is $2 \cdot 2^{e-2}$.

(ii) By (i) each element $a \in G$ has a unique expression of the form $a = (-1)^r 5^s$ with $r = 0$ or 1, and $0 \leq s < 2^{e-2}$. Hence a^k equals 1 in $\mathbb{Z}/n\mathbb{Z}$ if and only if kr is even and ks is a multiple of 2^{e-2}. In particular then k is even. If s is even, then the condition is satisfied for some $k < 2^{e-2}$. Thus a is primitive if and only if s is odd, or equivalently $a \equiv \pm 5 \pmod{8}$. ◇

As a corollary we have $\lambda(2^e) = 2^{e-2}$ for $e \geq 4$, and $\lambda(8) = 2$.

A.2 Primitive Elements for Prime Modules

More difficult (and mathematically more interesting) is the search for primitive elements for a prime module. Since the multiplicative group is cyclic it suffices to find *one* primitive element—all the other ones are powers of it with exponents coprime with $p-1$. In particular there are exactly $\varphi(p-1)$ primitive elements $\bmod p$. Usually the primitive elements for any module n where $\mathbb{M}_n$ is cyclic are also called **primitive roots** $\bmod n$.

The simplest, but not best, method is trying $x = 2, 3, 4, \ldots$, and testing if $x^d \neq 1$ for each proper divisor d of $p-1$. We need not to test all divisors:

Lemma 12 *Let p be a prime ≥ 5. An integer x is primitive* $\bmod p$*, if and only if $x^{(p-1)/q} \neq 1$ in $\mathbb{F}_p$ for each prime factor q of $p-1$.*

Proof. The order of x divides $p-1$, and each proper divisor of $p-1$ divides at least one of the quotients $\frac{p-1}{q}$. $\diamond$

To apply this criterion we need the prime decomposition of $p-1$. Then the test is efficient: The number of prime factors is $\leq \log_2(p-1)$, and for each of them we apply the binary power algorithm.

Example For $p = 41$ we have $p - 1 = 40 = 2^3 \cdot 5$. Hence x is primitive if and only if $x^{20} \neq 1$ and $x^8 \neq 1$. The test runs through the following steps in $\mathbb{F}_{41}$:

$$\begin{array}{llll}
x = 2: & x^2 = 4, & x^4 = 16, & \left\{\begin{array}{l} x^8 = 10, \\ x^{20} = x^8x^8x^4 = 1. \end{array}\right. \\
x = 3: & x^2 = 9, & x^4 = 81, & x^4 = -1,\ x^8 = 1. \\
x = 4: & x = 2^2, & \text{hence} & x^{20} = 1. \\
x = 5: & x^2 = 25, & x^4 = 10 & \left\{\begin{array}{l} x^8 = 18, \\ x^{20} = x^8x^8x^4 = 1. \end{array}\right. \\
x = 6: & x^2 = 36, & x^4 = 25 & \left\{\begin{array}{l} x^8 = 10, \\ x^{20} = x^8x^8x^4 = -1. \end{array}\right.
\end{array}$$

Hence 6 *is a primitive root for* $p = 41$.

The obvious question is how many integers must we try to find a primitive root? The quantity

$$\alpha(p) := \min\{x \in \mathbb{N} \mid x \text{ is primitive for } p\}$$

measures the complexity of complete search (but neglects the complexity of the proof of primitivity). It is known that the the function α is not bounded. In 1962 Burgess proved

$$\alpha(p) = \mathrm{O}(\sqrt[6]{p}).$$

Assuming ERH this exponential bound may be lessened to a polynomial one. The best known result is by Shoup 1990:

$$\alpha(p) = \mathrm{O}(\log(p)^6(1 + \log\log(p))^4).$$

Even completely simple questions are yet unanswered:

- Is 2 primitive for infinitely many primes?
- Is 10 primitive for infinitely many primes? (Gauss' conjecture)

Artin more generally conjectured: If $a \in \mathbb{N}$, and a is not an integer square (i. e. $a \neq 0, 1, 4, 9, \ldots$), then a is primitive for infinitely many primes.

Some relevant references:

- D. R. Heath-Brown: Artin's conjecture for primitive roots. Quart. J. Math. Oxford 37 (1986), 27–38.
- M. Ram Murty: Artin's conjecture for primitive roots. Math. Intelligencer 10 (1988), 59–67.
- V. Shoup: Searching for primitive roots in finite fields. Proc. 22nd STOC 1990, 546–554.
- Murata: On the magnitude of the least prime primitive root. J. Number Theory 37 (1991), 47–66.

A.3 Primitive Elements for Prime Powers

For prime powers we need one more lemma.

Lemma 13 *Let p be prime ≥ 3, k, an integer, and $d \geq 0$. Then*

$$(1+kp)^{p^d} \equiv 1 + kp^{d+1} \pmod{p^{d+2}}.$$

Proof. For $d = 0$ the statement is trivial. For $d \geq 1$ we reason by induction: Assume

$$(1+kp)^{p^{d-1}} = 1 + kp^d + rp^{d+1} = 1 + (k+rp)p^d.$$

Then

$$(1+kp)^{p^d} = (1+(k+rp)p^d)^p \equiv 1 + p\cdot(k+rp)\cdot p^d \equiv 1 + kp^{d+1} \pmod{p^{d+2}},$$

since $d+2 \leq 2d+1$ and $p \geq 3$. $\diamond$

Proposition 18 *Let p be prime ≥ 3, e, an exponent ≥ 2, and a be primitive* $\operatorname{mod} p$. *Then:*

(i) *a generates the group $\mathbb{M}_{p^e}$ if and only if $a^{p-1} \operatorname{mod} p^2 \neq 1$.*

(ii) *a or $a+p$ generates $\mathbb{M}_{p^e}$.*

(iii) *$\mathbb{M}_{p^e}$ is cyclic, and $\lambda(p^e) = \varphi(p^e) = p^{e-1}(p-1)$.*

Proof. (i) Let t be the multiplicative order of $a \operatorname{mod} p^e$, necessarily a multiple of the order of $a \operatorname{mod} p$, hence of $p-1$. On the other hand t divides $\varphi(p^e) = p^{e-1}(p-1)$. Hence $t = p^d(p-1)$ with $0 \leq d \leq e-1$.

Choose k such that $a^{p-1} = 1 + kp$. Then by Lemma 13

$$(a^{p-1})^{p^{e-2}} \equiv 1 + kp^{e-1} \equiv 1 \pmod{p^e} \iff p|k \iff a^{p-1} \equiv 1 \pmod{p^2}.$$

This is *not* the case if and only if $d = e-1$.

(ii) Assume a doesn't generate $\mathbb{M}_{p^e}$. Then $a^{p-1} \equiv 1 \pmod{p^2}$, hence

$$(a+p)^{p-1} \equiv a^{p-1} + (p-1)a^{p-2}p \equiv 1 - a^{p-2} \pmod{p^2},$$

and this is not $\equiv 1 \pmod{p^2}$.

(iii) follows immediately from (ii). $\diamond$

We immediately get an analogous result for modules that are twice a prime power:

Corollary 1 *Let $q = p^e$ be a power of a prime $p \geq 3$. Then:*

(i) *The multiplicative group* $\mathbb{M}_{2q}$ *is canonically isomorphic with* $\mathbb{M}_q$*, hence cyclic.*

(ii) *If* a *is a primitive element* $\bmod q$*, then* a *is primitive* $\bmod 2q$ *for odd* a*, and* $a+q$ *is primitive* $\bmod 2q$ *for even* a*.*

(iii) $\lambda(2p^e) = p^{e-1}(p-1)$.

Proof. (i) Since q and 2 are coprime, and $\mathbb{M}_2$ is the trivial group, by the chinese remainder theorem $\mathbb{M}_{2q} \cong \mathbb{M}_2 \times \mathbb{M}_q \cong \mathbb{M}_q$. This map is explicitely given by $a \bmod 2q \mapsto a \bmod q$.

(ii) Exactly one of a and $a+q$ is odd, hence coprime with $2q$. Thus the inverse isomorphism is

$$a \mapsto \begin{cases} a, & \text{if } a \text{ is odd,} \\ a+q, & \text{if } a \text{ is even.} \end{cases}$$

(iii) obvious. $\diamond$

A.4 The Structure of the Multiplicative Group

The previous results allow a complete characterization of the modules n for which the multiplicative group $\mathbb{M}_n$ is cyclic:

Corollary 2 (Gauss **1799**) *For $n \geq 2$ the multiplicative group $\mathbb{M}_n$ is cyclic if and only if n is one of the integers 2, 4, p^e, or $2p^e$ with an odd prime p.*

Proof. This follows from Proposition 18, Corollary 1, and the following Lemma 14. $\diamond$

Lemma 14 *If $m, n \geq 3$ are coprime, then $\mathbb{M}_{mn}$ is not cyclic, and $\lambda(mn) < \varphi(mn)$.*

Proof. If $n \geq 3$, then $\varphi(n)$ is even. For a prime power this follows from the explicit formula. In the general case we reason by the multiplicativity of the φ-function. We conclude

$$\mathrm{kgV}(\varphi(m), \varphi(n)) < \varphi(m)\,\varphi(n) = \varphi(mn),$$

$$\lambda(mn) = \mathrm{kgV}(\lambda(m), \lambda(n)) \leq \mathrm{kgV}(\varphi(m), \varphi(n)) < \varphi(mn).$$

Hence $\mathbb{M}_{mn}$ is not cyclic. $\diamond$

Now the structure of the multiplicative group is completely known also for a general module. Let us denote the cyclic group of order d by $\mathcal{Z}_d$.

Theorem 2 *Let $n = 2^e p_1^{e_1} \cdots p_r^{e_r}$ be the prime decomposition of the integer $n \geq 2$ with different odd primes $p_1, \ldots, p_r$, and $e \geq 0$, $r \geq 0$, $e_1, \ldots, e_r \geq 1$. Let $q_i = p_i^{e_i}$ and $q_i' = p_i^{e_i - 1}(p_i - 1)$ for $i = 1, \ldots, r$. Then*

$$\mathbb{M}_n \cong \begin{cases} \mathcal{Z}_{q_1'} \times \cdots \times \mathcal{Z}_{q_r'}, & \textit{if } e = 0 \textit{ or } 1, \\ \mathcal{Z}_2 \times \mathcal{Z}_{2^{e-2}} \times \mathcal{Z}_{q_1'} \times \cdots \times \mathcal{Z}_{q_r'}, & \textit{if } e \geq 2. \end{cases}$$

We find a primitive element $a \bmod n$ by choosing primitive elements $a_0 \bmod 2^e$ (if $e \geq 2$) and $a_i \bmod q_i$ and solving the simultaneous congruences $a \equiv a_i \pmod{q_i}$, and if applicable $a \equiv a_0 \pmod{2^e}$.

Proof. All this follows from the chinese remainder theorem. $\diamond$

Exercise Derive a general formula for $\lambda(n)$.

A.5 The Jacobi Symbol

Consider the multiplicative group $\mathbb{M}_n = (\mathbb{Z}/n\mathbb{Z})^\times$ for a module $n \geq 2$, and its squaring map

$$\mathbf{q}\colon \mathbb{M}_n \longrightarrow \mathbb{M}_n, \quad x \mapsto x^2 \bmod n\,.$$

$\mathbf{q}$ is a group homomorphism. The elements in the image of $\mathbf{q}$ are the **quadratic residues** mod n. An integer x is a quadratic residue mod n if $x \bmod n$ is invertible, and there exists an integer u with $u^2 \equiv x \pmod{n}$. Thus the set of quadratic residues is the subset $\mathbb{M}_n^2$ of the residue class ring $\mathbb{Z}/n\mathbb{Z}$. (This notation is not standard just as little as $\mathbb{M}_n$. But it spares writing $((\mathbb{Z}/n\mathbb{Z})^\times)^2$ over and over again.)

Remarks and Examples

1. For $n = 2$ we have $\mathbb{M}_n^2 = \mathbb{M}_n = \{1\}$.

2. For $n \geq 3$ we have $-1 \neq 1$ and $(-1)^2 = 1$. Hence $\mathbf{q}$ is not injective and thus also not surjective. Therefore quadratic non-residues exist.

3. Let $n = p \geq 3$ be prime. Then the kernel of $\mathbf{q}$ exactly consists of the roots of the polynomial $X^2 - 1$ in the field $\mathbb{F}_p$, hence of $\{\pm 1\}$. We conclude that the number of quadratic residues is $\frac{p-1}{2}$.

4. More generally let $n = q = p^e$ be a power of an odd prime p. Then $\mathbb{M}_n$ is cyclic of order $\varphi(q) = q \cdot (1 - \frac{1}{p})$ by Proposition 18. Thus 1 has exactly the square roots ± 1 in $\mathbb{M}_q$, and the number of quadratic residues is $\varphi(q)/2$.

5. Let n be a product of two different odd primes p and q. By the chinese remainder theorem the natural map $\mathbb{M}_n \longrightarrow \mathbb{M}_p \times \mathbb{M}_q$ is an isomorphism. Hence $\mathbb{M}_n$ contains exactly four square roots of 1, and $\mathbb{M}_n^2 \leq \mathbb{M}_n$ is a subgroup of index 4.

6. In the general case let $n = 2^e p_1^{e_1} \cdots p_r^{e_r}$ be the prime decomposition with different odd primes $p_1, \ldots, p_r$, and $r \geq 0$, $e \geq 0$, $e_1, \ldots, e_r \geq 1$. Proposition 2 tells us the number of square roots of 1 in $\mathbb{M}_n$:

$$\begin{array}{ll} 2^r, & \text{if } e = 0 \text{ or } 1, \\ 2^{r+1}, & \text{if } e = 2, \\ 2^{r+2}, & \text{if } e \geq 3. \end{array}$$

   This number is also the order of the kernel of $\mathbf{q}$, hence the index of $\mathbb{M}_n^2$ in $\mathbb{M}_n$.

The naive algorithm, exhaustion, for determing the quadratic residuosity of a mod n tries $1^2, 2^2, 3^2, \ldots$ until it hits a. A quadratic non-residue always takes $\lfloor \frac{n}{2} \rfloor$ steps, a quadratic residue $n/4$ steps in the average. Thus the costs grow exponentially with the number $\log n$ of places.

For the case where n is *prime* we'll see better algorithms.

The phenomen that there is no efficient algorithm for *composite* integers n is the basis of many cryptographic constructions, for instance the simplest perfect random generator (BBS, see Part IV).

For a prime module p the LEGENDRE **symbol** indicates quadratic residuosity:

$$(\frac{x}{p}) = \begin{cases} 1 & \text{if } x \text{ is a quadratic residue,} \\ 0 & \text{if } p|x, \\ -1 & \text{otherwise.} \end{cases}$$

The LEGENDRE symbol defines a homomorphism

$$(\frac{\bullet}{p}) : \mathbb{M}_p \longrightarrow \mathbb{M}_p/\mathbb{M}_p^2 \cong \{\pm 1\}.$$

In the special case $p = 2$

$$(\frac{x}{2}) = \begin{cases} 1 & \text{if } x \text{ is odd,} \\ 0 & \text{if } x \text{ is even.} \end{cases}$$

Proposition 19 (EULER's criterion) *Let p be an odd prime. then*

$$x^{\frac{p-1}{2}} \equiv (\frac{x}{p}) \pmod p \quad \textit{for all } x.$$

Proof. If $p|x$ both sides equal 0. Otherwise $(x^{\frac{p-1}{2}})^2 = x^{p-1} \equiv 1$, hence $x^{\frac{p-1}{2}} \equiv \pm 1$. Let a be primitive mod p. Then both sides equal -1, hence the assertion holds for $x = a$. Since both sides represent homomorphisms $\mathbb{F}_p^\times \longrightarrow \{\pm 1\}$ the assertion is true for all powers of a, hence for all x that are no multiples of p. $\diamond$

EULER's criterion yields an efficient algorithm for deciding quadratic residuosity: We have to take $\frac{p-1}{2}$-th powers in $\mathbb{F}_p^\times$, and this costs at most $2\lfloor \log_2(\frac{p-1}{2}) \rfloor$ multiplications mod p. Taking the cost of modular multiplication into account we get an order of magnitude of $\log_2(p)^3$.

By EULER's criterion -1 is a quadratic residue if and only if $\frac{p-1}{2}$ is even, hence $p \equiv 1 \pmod 4$. The decision on 2 or 3 is significantly more difficult. However there is an even faster algorithm. It is the subject of the following Section A.6.

The LEGENDRE symbol has a natural generalization by the JACOBI symbol (that uses the same notation): For $n > 0$ with prime decomposition

$n = p_1 \cdots p_r$ (the p_i not necessarily distinct)

$$(\frac{x}{n}) := (\frac{x}{p_1}) \cdots (\frac{x}{p_r}) \quad \text{for } x \in \mathbb{M}_n.$$

In particular $(\frac{x}{n}) = 0$ if x and n are not coprime. The supplementing definitions $(\frac{x}{1}) = 1$, $(\frac{x}{n}) = (\frac{x}{-n})$ for $n < 0$, and $(\frac{x}{0}) = 0$, make the JACOBI symbol a function

$$(\frac{\bullet}{\bullet}) : \mathbb{Z} \times \mathbb{Z} \longrightarrow \mathbb{Z}$$

with values in $\{0, \pm 1\}$, and multiplicative in numerator and denominator. In particular the JACOBI symbol defines a homomorphism $(\frac{\bullet}{n})$ from $\mathbb{M}_n$ to $\{\pm 1\}$. But it is *not* an indicator of quadratic residuosity. Denoting $\mathbb{M}_n^+ = \ker(\frac{\bullet}{n})$ and $\mathbb{M}_n^- = \mathbb{M}_n - \mathbb{M}_n^+$, in general $\mathbb{M}_n^2$ is a proper subgroup of $\mathbb{M}_n^+$. Its index is given by example 6 above: If the number of square roots of 1 is 2^k with $k \geq 1$, then $\mathbb{M}_n^2$ has index 2^{k-1} in $\mathbb{M}_n^+$.

In any case $(\frac{x}{n})$ depends on the residue class $x \bmod n$ only. Obviously

$$(\frac{x}{2^k}) = \begin{cases} 1, & \text{if } x \text{ is odd,} \\ 0, & \text{if } x \text{ is even.} \end{cases}$$

A.6 Quadratic Reciprocity

Quadratic reciprocity provides a very convenient method of computing the JACOBI (or LEGENDRE) symbol and thereby deciding quadratic residuosity. It relies on the following two propositions and a lemma that helps to reduce composite modules to prime modules.

Lemma 15 *Let $s, t \in \mathbb{Z}$ be odd. Then*

(i) $\frac{s-1}{2} + \frac{t-1}{2} \equiv \frac{st-1}{2} \pmod 2$,

(ii) $\frac{s^2-1}{8} + \frac{t^2-1}{8} \equiv \frac{s^2t^2-1}{8} \pmod 2$.

Proof. Assume $s = 2k+1$ and $t = 2l+1$. Then $st = 4kl + 2k + 2l + 1$,

$$\frac{st-1}{2} = 2kl + k + l \;\equiv\; k + l = \frac{s-1}{2} + \frac{t-1}{2}.$$

Moreover

$$s^2 = 4 \cdot (k^2 + k) + 1, \quad t^2 = 4 \cdot (l^2 + l) + 1,$$
$$s^2t^2 = 16 \cdot \ldots + 4 \cdot (k^2 + k + l^2 + l) + 1,$$
$$\frac{s^2t^2 - 1}{8} = 2 \cdot \ldots + \frac{k^2 + k + l^2 + l}{2},$$

and this proves the assertion. $\diamond$

Proposition 20 *Let n be odd. Then*

(i) $(\frac{-1}{n}) = (-1)^{\frac{n-1}{2}}$,

(ii) $(\frac{2}{n}) = (-1)^{\frac{n^2-1}{8}}$

Proof. The lemma reduces the assertions to the case $n = p$ prime.
(i) is a direct consequence of EULER's criterion, Proposition 19.
(ii) We have

$$(-1)^k \cdot k \equiv \begin{cases} k, & \text{if } k \text{ is even,} \\ p - k, & \text{if } k \text{ is odd,} \end{cases}$$

$$\prod_{k=1}^{\frac{p-1}{2}} (-1)^k \cdot k \equiv 2 \cdot 4 \cdots (p-1) = 2^{\frac{p-1}{2}} \cdot (\frac{p-1}{2})!.$$

Om the other hand

$$\prod_{k=1}^{\frac{p-1}{2}} (-1)^k \cdot k = (\frac{p-1}{2})! \cdot (-1)^{\frac{p^2-1}{8}}, \quad \text{since} \quad \sum_{k=1}^{\frac{p-1}{2}} k = \frac{(p-1)(p+1)}{2\cdot 2\cdot 2}.$$

Now $(\frac{p-1}{2})!$ is a product of positive integers $< p$, thus not a multiple of p. Hence we may divide by it. Then from the two equations and EULER's criterion we get

$$(-1)^{\frac{p^2-1}{8}} \equiv 2^{\frac{p-1}{2}} \equiv (\frac{2}{p}) \pmod{p}.$$

Since $p \geq 3$ this congruence implies equality. $\diamond$

In particular 2 is a quadratic residue modulo the prime p if and only if $(p^2-1)/8$ is even, or $p^2 \equiv 1 \pmod{16}$, or $p \equiv 1$ or $7 \pmod{8}$.

Theorem 3 (Law of Quadratic Reciprocity) *Let m and n be two different odd coprime positive integers. Then*

$$(\frac{m}{n})(\frac{n}{m}) = (-1)^{\frac{m-1}{2}\frac{n-1}{2}}.$$

Here is a somewhat more comprehensible formula:

$$(\frac{m}{n}) = \begin{cases} -(\frac{n}{m}) & \text{if } m \equiv n \equiv 3 \pmod{4}, \\ (\frac{n}{m}) & \text{else.} \end{cases}$$

The proof is in the next section. First we illustrate the computation with an example:

Is 7 a quadratic residue mod 107? *No*, as the following computation shows:

$$(\frac{7}{107}) = -(\frac{107}{7}) = -(\frac{2}{7}) = -1.$$

Likewise 7 is not a quadratic residue mod 11:

$$(\frac{7}{11}) = -(\frac{11}{7}) = -(\frac{4}{7}) = -(\frac{2}{7})(\frac{2}{7}) = -1.$$

Hence 7 is a quadratic non-residue also mod $1177 = 11 \cdot 107$. But $(\frac{7}{1177}) = 1$.

From the law of quadratic reciprocity we derive the following algorithm:

Procedure JacobiSymbol

```
Input parameters:
    $m$, $n$ = two integers.
Output parameter:
    jac = $(\frac{m}{n})$.
Instructions:
    If $n = 0$ output jac = 0 end
    If $m = 0$ output jac = 0 end
    If $\gcd(m, n) > 1$ output jac = 0 end
    [Now $m, n \neq 0$ are coprime, so jac $= \pm 1$.]
    jac = 1.
    If $n < 0$ replace $n$ by $-n$.
    If $n$ is even divide $n$ by the maximum possible power $2^k$.
    If $m < 0$
        replace $m$ by $-m$,
        if $n \equiv 3 \pmod 4$ replace jac by $-$jac.
    [From now on $m$ and $n$ are coprime, and $n$ is positive and odd.]
    [In the last step $m = 0$ and $n = 1$ may occur.]
    If $m > n$ replace $m$ by $m$ mod $n$.
    While $n > 1$:
        If $m$ is even:
            Divide $m$ by the maximum possible power $2^k$,
            if ($k$ is odd and $n \equiv \pm 3 \pmod 8$) replace jac by $-$jac.
        [Now $m$ and $n$ are odd and coprime, $0 < m < n$.]
        [The law of quadratic reciprocity applies.]
        If ($m \equiv 3 \pmod 4$ and $n \equiv 3 \pmod 4$)
            replace jac by $-$jac.
        Set $d = m$, $m = n$ mod $m$, $n = d$.
```

The analysis of this algorithm resembles the analysis of the Euclidean algorithm: We need at most $5 \cdot \log(m)$ steps, each one essentially consisting of one integer division. Since the size of the operands rapidly decreases, the total cost amounts to $O(\log_2(m)^2)$. This is significantly faster than applying Euler's criterion.

A.7 Proof of the Law of Quadratic Reciprocity

Now for the proof of the law of quadratic reciprocity. In the literature we find many different proofs. We adapt one that uses the theory of finite fields and follows ideas by ZOLOTAREV (Nouvelles Annales de Mathematiques 11 (1872), 354–362) and SWAN (Pacific J. Math. 12 (1962), 1099–1106).

Lemma 16 *Let p an odd prime, and a and p be coprime. Then the following statements are equivalent:*

(i) *a is a quadratic residue* $\bmod p$.

(ii) *Multiplication by a is an even permutation of $\mathbb{F}_p$.*

Proof. Denote the multiplication by $\mu_a : \mathbb{F}_p \longrightarrow \mathbb{F}_p$, $x \mapsto ax \bmod p$. Then $a \mapsto \mu_a$ is an injective group homomorphism $\mu : \mathbb{F}_p^\times \longrightarrow \mathfrak{S}_p$ to the full permutation group on p elements. If a is primitive, then μ_a has exactly two cycles: $\{0\}$ and $\mathbb{F}_p^\times$. Since p is odd, the sign of μ_a is $\sigma(\mu_a) = (-1)^{p-2} = -1$, hence μ_a is an odd permutation.

Since a generates the group $\mathbb{F}_p^\times$, the two homomorphisms

$$\left(\frac{\bullet}{p}\right) \quad \text{and} \quad \sigma \circ \mu : \mathbb{F}_p^\times \longrightarrow \{\pm 1\}$$

must be identical, and this was the assertion. $\diamond$

As another tool we use the **discriminant** of a polynomial $f = a_nT^n + \cdots + a_0 \in K[T]$. We can compute it in any extension field $L \supseteq K$ that contains all the zeroes $t_1, \ldots, t_n$ of f by the formula

$$D(f) = a_n^{2n-2} \cdot \prod_{1 \le i < j \le n} (t_i - t_j)^2.$$

> The discriminant is invariant under all permutations of the zeroes. Hence it is in K. In our case this will also follow from the explicit computation.

The ususal method of computing the discriminant from the coefficients consists in comparing it with the resultant of f and its derivative f'. For the cyclotomic polynomial $f = T^n - 1$ the computation is outstandingly simple:

Lemma 17 *Assume that* $\operatorname{char} K$ *doesn't divide n. Then the polynomial $f = T^n - 1 \in K[T]$ has discriminant*

$$D(f) = (-1)^{\frac{n(n-1)}{2}} \cdot n^n.$$

Proof. Let ζ be a primitive n-th root of unity (in some suitable extension field). Then

$$\begin{aligned} f &= \prod_{i=0}^{n-1}(T-\zeta^i), \\ D(f) &= \prod_{0\le i<j\le n-1}(\zeta^i-\zeta^j)^2 = (-1)^{\frac{n(n-1)}{2}}\cdot\prod_{i\ne j}(\zeta^i-\zeta^j) \\ &= (-1)^{\frac{n(n-1)}{2}}\cdot\prod_{i=0}^{n-1}\left[\zeta^i\cdot\prod_{k=1}^{n-1}(1-\zeta^k)\right]. \end{aligned}$$

The polynomial

$$g = T^{n-1}+\cdots+1 = \prod_{k=1}^{n-1}(T-\zeta^k) \in K[T]$$

satisfies $g(1) = n$. Hence

$$D(f) = (-1)^{\frac{n(n-1)}{2}}\cdot\prod_{i=0}^{n-1}[\zeta^i\cdot n] = (-1)^{\frac{n(n-1)}{2}}\cdot n^n,$$

as claimed. $\diamond$

Lemma 18 *Let p be an odd prime and n an odd integer, coprime with p. Then the following statements are equivalent:*

(i) *The discriminant of $T^n-1 \in \mathbb{F}_p[T]$ is a quadratic residue* $\bmod\, p$.

(ii) $l = (-1)^{(n-1)/2}\cdot n$ *is a quadratic residue* $\bmod\, p$.

Proof. By Lemma 17 the discriminant is $D(f) = l^n$. Let $n = 2k+1$. Then $D(f)$ is the product of l with the quadratic residue l^{2k}. $\diamond$

The discriminant of a polynomial $f \in K[T]$ is a square in an extension field $L \supseteq K$ that contains the zeroes of f:

$$D(f) = \Delta(f)^2 \quad\text{with}\quad \Delta(f) = a_n^{n-1}\cdot\prod_{i<j}(t_i-t_j).$$

But $\Delta(f)$ inherits the sign of a permutation of the zeroes. Thus is not invariant, and therefore in general is not contained in K.

Proof of the theorem. Because of Lemma 15 (i) it suffices to prove the quadratic reciprocity law for two different odd primes p and q.

Let $K = \mathbb{F}_p$, ζ be a primitive q-th root of unity, $L = K(\zeta)$, and $f = T^q - 1$. Then $\zeta \mapsto \zeta^p$ defines a permutation μ_p of the roots of unity, and an automorphism of L over K. Thus:

$$\sigma(\mu_p) \cdot \Delta(f) = \prod_{i<j} (\zeta^{pi} - \zeta^{pj}) = \Delta(f)^p.$$

This yields a chain of equivalent statements:

$$(-1)^{\frac{q-1}{2}} \cdot q \text{ quadratic residue } \bmod p \iff D(f) \text{ quadratic residue } \bmod p$$

$$\iff \Delta(f) \in \mathbb{F}_p \iff \Delta(f) = \Delta(f)^p \iff \sigma(\mu_p) = 1$$

$$\iff p \text{ quadratic residue } \bmod q.$$

From Proposition 20 (i) we get

$$(\frac{p}{q}) = (\frac{(-1)^{\frac{q-1}{2}} q}{p}) = (\frac{q}{p}) \cdot (\frac{-1}{p})^{\frac{q-1}{2}} = (\frac{q}{p}) \cdot (-1)^{\frac{p-1}{2}\frac{q-1}{2}},$$

as claimed. $\diamondsuit$

A.8 Quadratic Non-Residues

How to find a quadratic *non-residue* modulo a prime p? That is, an integer a with $p \nmid a$ that is not a quadratic residue mod a. The preferred solution is the smallest possible positive one. Nevertheless we start with -1:

Proposition 21 *Let $p \geq 3$ be prime.*

(i) -1 *is a quadratic non-residue* $\bmod p \iff p \equiv 3 \pmod 4$.

(ii) 2 *is a quadratic non-residue* $\bmod p \iff p \equiv 3$ *or* $5 \pmod 8$.

(iii) (*For* $p \geq 5$) 3 *is a quadratic non-residue* $\bmod p \iff p \equiv 5$ *or* $7 \pmod{12}$.

(iv) (*For* $p \geq 7$) 5 *is a quadratic non-residue* $\bmod p \iff p \equiv 2$ *or* $3 \pmod 5$.

Proof. (i) This follows from Proposition 20. However there is an even simpler proof:

$$\begin{aligned} -1 \in \mathbb{M}_p^2 &\iff \bigvee_{i\in\mathbb{Z}} i^2 \equiv -1 \pmod p \iff \bigvee_{i\in\mathbb{Z}} \operatorname{ord}_p i = 4 \\ &\iff 4 \mid \#\mathbb{F}_p^\times = p-1 \iff p \equiv 1 \pmod 4. \end{aligned}$$

(ii) This also follows from Proposition 20: By the adjacent remark $2 \in \mathbb{M}_p^2 \iff p \equiv 1$ or $7 \pmod 8$.

(iii) We use the law of quadratic reciprocity:

$$\begin{aligned} (\frac{3}{p}) = (-1)^{\frac{p-1}{2}}(\frac{p}{3}) &= \begin{cases} (-1)^{6k}(\frac{1}{3}) = 1 & \text{if } p = 12k+1, \\ (-1)^{6k+2}(\frac{2}{3}) = -1 & \text{if } p = 12k+5, \\ (-1)^{6k+3}(\frac{1}{3}) = -1 & \text{if } p = 12k+7, \\ (-1)^{6k+5}(\frac{2}{3}) = 1 & \text{if } p = 12k+11, \end{cases} \\ &= \begin{cases} 1 & \text{if } p \equiv 1 \text{ or } 11 \pmod{12}, \\ -1 & \text{if } p \equiv 5 \text{ or } 7 \pmod{12}. \end{cases} \end{aligned}$$

(iv) By quadratic reciprocity

$$(\frac{5}{p}) = (\frac{p}{5}) = \begin{cases} 1 & \text{if } p \equiv 1 \text{ or } 4 \pmod 5, \\ -1 & \text{if } p \equiv 2 \text{ or } 3 \pmod 5, \end{cases}$$

as claimed. $\diamond$

Corollary 1 241 *is the unique odd prime* < 400 *for which none of* -1, 2, 3, 5 *are quadratic non-residues.*

Corollary 2 *For each odd prime p at least one of -1, 2, 3, or 5 is a quadratic non-residue except for $p \equiv 1, 49 \pmod{120}$.*

For arbitrary, not necessarily prime, modules we have some analogous results:

Lemma 19 *Let $n \in \mathbb{N}$, $n \geq 2$. Assume that $(\frac{a}{n}) = -1$ for some $a \in \mathbb{Z}$. Then a is a quadratic non-residue in $\mathbb{Z}/n\mathbb{Z}$.*

Proof. Let $n = p_1^{e_1} \cdots p_r^{e_r}$ be the prime decomposition. Then

$$(\frac{a}{n}) = (\frac{a}{p_1})^{e_1} \cdots (\frac{a}{p_r})^{e_r}.$$

Hence for some k the exponent e_k is odd, and $(\frac{a}{p_k}) = -1$. Then a is a quadratic non-residue mod p_k. Since $\mathbb{F}_{p_k}$ is a homomorphic image of $\mathbb{Z}/n\mathbb{Z}$, a is a forteriori a quadratic non-residue mod n. $\diamond$

Corollary 3 *Let $n \in \mathbb{N}$, $n \geq 2$, and not a square in $\mathbb{Z}$.*

(i) *If $n \equiv 3 \pmod{4}$, then -1 is a quadratic non-residue in $\mathbb{Z}/n\mathbb{Z}$.*

(ii) *If $n \equiv 5 \pmod{8}$, then 2 is a quadratic non-residue in $\mathbb{Z}/n\mathbb{Z}$.*

And so on. Unfortunately this approach doesn't completely cover all cases, see the remark below. Nevertheless we note that an algorithm for finding a quadratic non-residue needs to address the cases $n \equiv 1 \pmod{8}$ only. Again there are two variants:

- A deterministic algorithm that tests $a = 2, 3, 5, \ldots$ in order. Assuming ERH—for the character $\chi = (\frac{\bullet}{n})$—it is polynomial in the number $\log(n)$ of places.
- A probabilistic algorithm that randomly chooses a and succeeds with probability $\frac{1}{2}$ each time, yielding $(\frac{a}{n}) = -1$. Computing the Jacobi symbol takes $O(\log(n)^2)$ steps. In the average we need two trials to hit a quadratic non-residue.

Exercise For which prime modules is 7, 11, or 13 a quadratic non-residue? What is the smallest prime module for which this approach (together with Proposition 21) doesn't provide a quadratic non-residue?

Remark A result by Chowla/Fridlender/Salié says that (with a constant $c > 0$) there are infinitely many primes such that all integers a with $1 \leq a \leq c \cdot \log(p)$ are quadratic residues mod p. Ringrose/Graham and—assuming ERH—Montgomery have somewhat stronger versions of this result.

Remark There is no global polynomial (in $\log(n)$) upper bound for the smallest quadratic non-residue that is valid for all modules n. A very weak but simple result is in the following proposition.

Proposition 22 *Let $p \geq 3$ be a prime. Then there is a quadratic non-residue $a < 1 + \sqrt{p}$.*

Proof. There are quadratic non-residues > 1 (and $< p$). Let a be the smallest of these. Let $m = \lceil \frac{p}{a} \rceil$. Thus $(m-1) \cdot a < p < m \cdot a$, or

$$0 < m \cdot a - p < a.$$

Hence $m \cdot a \equiv m \cdot a - p$ is a quadratic residue. This is possible only if m is a quadratic non-residue. Since a is minimal we have $a \leq m$. We conclude

$$(a-1)^2 < (m-1) \cdot a < p,$$

hence $a - 1 < \sqrt{p}$. $\diamond$

Relevant references

- V. R. Fridlender: On the least n-th power non-residue. Dokl. Akad. Nauk. SSSR 66 (1949), 351–352.
- H. Salié: Über den kleinsten positiven quadratischen Nichtrest nach einer Primzahl. Math. Nachr. 3 (1949), 7–8.
- N. C. Ankeney: The least quadratic nonresidue. Ann. of Math. 55 (1952), 65–72.
- H. L. Montgomery: *Topics in Multiplicative Number Theory.* Springer LNM 227 (1971).
- J. Buchmann/V. Shoup: Constructing nonresidues in finite fields and the extended Riemann hypothesis. Preprint 1990.
- S. W. Graham/C. Ringrose: Lower bounds for least quadratic non-residues. In: B. C. Berndt et al. (Eds): *Analytic Number Theory*, Birkhäuser, Boston 1990, 270–309.
- D. J. Bernstein: Faster algorithms to find non-squares modulo worst-case integers. Preprint 2002.

A.9 Primitive Elements for Special Primes

For many prime modules finding quadratic non-residues has turned out to be extremely easy. The same is true for finding primitive roots.

Proposition 23 *Let $p = 2p' + 1$ be a special prime. Then:*

(i) *$a \in [2 \ldots p-2]$ is a primitive root* mod p *if and only if it is a quadratic non-residue.*

(ii) *$(-1)^{\frac{p'-1}{2}} \cdot 2$ is a primitive root* $\bmod p$.

Proof. We have $p \equiv 3 \pmod 4$, thus -1 is a quadratic non-residue by Proposition 21.

(i) Since the order $\#\mathbb{F}_p^\times = p-1$ is even, moreover each primitive root is also a quadratic non-residue. There are $\varphi(p-1) = p'-1$ of them, thus we have found p' quadratic non-residues. Since $p' = \frac{p-1}{2}$, these must be all of them.

(ii) In the case $p' \equiv 1 \pmod 4$ we have $p \equiv 3 \pmod 8$, hence $2 = (-1)^{\frac{p'-1}{2}} \cdot 2$ is a quadratic non-residue by Proposition 21, hence also primitive.

In the case $p' \equiv 3 \pmod 4$ we have $p \equiv 7 \pmod 8$, hence 2 is a quadratic residue, and -1 is a quadratic non-residue again by Proposition 21. Therefore $-2 = (-1)^{\frac{p'-1}{2}} \cdot 2$ is a quadratic non-residue, hence also primitive. $\diamond$

The effortlessness of finding a primitive root is one of several reasons why cryptologists like special primes.

Corollary 1 *Let $p = 2p' + 1$ be a special prime. Then the order of 2 in $\mathbb{F}_p^\times$ is*

(i) $p - 1 = 2p'$ *if* $p' \equiv 1 \pmod 4$,

(ii) $(p-1)/2 = p'$ *if* $p' \equiv 3 \pmod 4$.

Proof. (i) 2 is a primitive root.

(ii) The divisors of $\#\mathbb{F}_p^\times$ are $\{1, 2, p', 2p'\}$. Since 2 is a quadratic residue, it is not primitive, hence the order is not $2p'$. The order cannot be 1 since $2 \neq 1$ in $\mathbb{F}_p$. And the order 3 would imply that $4 = 1$, hence $3 = 0$ in $\mathbb{F}_p$, hence $p = 3$ which ic not a special prime. $\diamond$

A.10 Some Group Theoretic Trivia

Here we collect some elementary results on finite groups. The exponent of a group G is the minimum positive integer e (or ∞) such that $x^e = \mathbf{1}$ for all $x \in G$. Denote the order of a group element x by $\operatorname{ord} x$ (positive integer or ∞).

Lemma 20 *Let G be a finite group with exponent e. Then $e \mid \#G$, and $e = t := \operatorname{lcm}(\{\operatorname{ord} x \mid x \in G\})$.*

Proof. By LAGRANGE's Theorem $\operatorname{ord} x \mid \#G$ for all $x \in G$, hence $e \mid \#G$. Moreover $x^e = \mathbf{1}$ by definition of e, hence $\operatorname{ord} x \mid e$ for all $x \in G$. Hence $t \mid e$. Sinc $x^t = \mathbf{1}$ for all x, even $t = e$. $\diamond$

Lemma 21 *Let G and H be groups, $g \in G$ with $\operatorname{ord} g = r$ and $h \in H$ with $\operatorname{ord} h = s$. Then $\operatorname{ord}(g, h) = \operatorname{lcm}(r, s)$ in the direct product $G \times H$.*

Proof.

$$(g^e, h^e) = (g, h)^e = \mathbf{1} \text{ in } G \times H \iff g^e = \mathbf{1} \text{ in } G \text{ and } h^e = \mathbf{1} \text{ in } H.$$

$\diamond$

Lemma 22 *Let G be a group with exponent r and H be a group with exponent s. Then the direct product $G \times H$ has exponent $t := \operatorname{lcm}(r, s)$.*

Proof. Since $r, s \mid t$ we have $(g, h)^t = (g^t, h^t) = (\mathbf{1}, \mathbf{1})$ for all $g \in G$ and $h \in H$. Thus the exponent e of $G \times H$ is $\leq t$.

Since $(\mathbf{1}, \mathbf{1}) = (g, h)^e = (g^e, h^e)$ for all g, h, we have $r \mid e$ and $s \mid e$, hence $t \mid e$. $\diamond$

Lemma 23 *Let G be a cyclic group of prime order r, and H, a cyclic group of prime order $s \neq r$. Then the direct product $G \times H$ is cyclic of order $r \cdot s$.*

Proof. Let $g \in G$ have order r, and $h \in H$ have order s. Then by Lemma 21 the element (g, h) has order $\operatorname{lcm}(r, s) = r \cdot s = \#(G \times H)$, hence generates $G \times H$. $\diamond$

Lemma 24 *Let G be an abelian group.*

(i) *Let $a, b \in G$, $\operatorname{ord} a = r$, $\operatorname{ord} b = s$, where r, s are finite and coprime. Then $\operatorname{ord}(ab) = rs$.*

(ii) *Let $a, b \in G$,* $\operatorname{ord} a = r$ *and* $\operatorname{ord} b = s$ *finite,* $t := \operatorname{lcm}(r, s)$*. Then* $\operatorname{ord}(ab) \mid t$*, and there is a* $c \in G$ *with* $\operatorname{ord} c = t$*.*

(iii) *Let* $m = \max\{\operatorname{ord} a \mid a \in G\}$ *be finite. Then* $\operatorname{ord} b \mid m$ *for all* $b \in G$*. In particular* m *is the exponent of* G*.*

Proof. (i) Let $k := \operatorname{ord}(ab)$. From $(ab)^{rs} = (a^r)^s \cdot (b^s)^r = \mathbf{1}$ we conclude that $k \mid rs$. Conversely, since $a^{ks} = a^{ks} \cdot (b^s)^k = (ab)^{ks} = \mathbf{1}$ we have $r \mid ks$, hence $r \mid k$, and likewise $s \mid k$, hence $rs \mid k$.

(ii) Let $k := \operatorname{ord}(ab)$. From $(ab)^t = a^t \cdot b^t = \mathbf{1}$ follows that $k \mid t$.

Now let p^e be a prime power with $p^e \mid t$, say $p^e \mid r$. Then a^{r/p^e} has order p^e. Let $t = p_1^{e_1} \cdots p_r^{e_r}$ be the prime decomposition with different primes p_i. Then there are $c_i \in G$ with $\operatorname{ord} c_i = p_i^{e_i}$. Since these orders are pairwise coprime, the element $c = c_1 \cdots c_r$ has order t by (i).

(iii) Let $\operatorname{ord} b = s$. Then by (ii) there is a $c \in G$ with $\operatorname{ord} c = \operatorname{lcm}(m, s)$. Hence $\operatorname{lcm}(m, s) \le m$, hence $= m$, thus $s \mid m$. $\diamond$

Remarks

1. For non-abelian groups all three statements (i)–(iii) may be false. As an example consider the symmetric group $\mathcal{S}_4$ of order $4! = 24$. The possible orders of its elements are 1 (for the trivial permutation), 2 for permutations consisting of one or two disjoint 2-cycles, 3 for all 3-cycles, and 4 for all 4-cycles. Thus the maximum order is 4, but the exponent = the lcm of all orders is 12 (by Lemma 20). The cycle $\sigma = (1\,2\,3)$ has order $r = 3$, the transposition $\tau = (3\,4)$ has order $s = 2$. Their product is the 4-cycle $(2\,3\,4\,1)$ of order $4 \ne \operatorname{lcm}(r, s) = 6$, and there doesn't exist any permutation of order 6.

2. In a nontrivial abelian group the order of a product ab in general differs from the lcm of the single orders: Take $a \ne \mathbf{1}$ and $b = a^{-1}$.

A.11 Blum Integers

Let $n = pq$ with different primes $p, q \geq 3$. Then

$$\mathbb{M}_n \cong \mathbb{M}_p \times \mathbb{M}_q, \quad \mathbb{M}_n^2 \cong \mathbb{M}_p^2 \times \mathbb{M}_q^2,$$

$$\mathbb{M}_n/\mathbb{M}_n^2 \cong \mathbb{M}_p/\mathbb{M}_p^2 \times \mathbb{M}_q/\mathbb{M}_q^2 \cong \mathcal{Z}_2 \times \mathcal{Z}_2,$$

in particular $\#(\mathbb{M}_n/\mathbb{M}_n^2) = 4$. The subgroups $\mathbb{M}_n^2 \leq \mathbb{M}_n^+$ and $\mathbb{M}_n^+ \leq \mathbb{M}_n$ are proper and hence of index 2. The ring $\mathbb{Z}/n\mathbb{Z}$ contains exactly 4 roots of unity: $1, -1, \tau, -\tau$, where

$$\tau \equiv -1 \pmod p, \quad \tau \equiv 1 \pmod q,$$

thus $(\frac{\tau}{n}) = -1$. In other words: The kernel of the squaring homomorphism $\mathbf{q} : \mathbb{M}_n \longrightarrow \mathbb{M}_n^2$ is $K = \{\pm 1, \pm\tau\}$, isomorphic with the Klein four-group.

An integer of the form $n = pq$ with different primes $p, q \equiv 3 \pmod 4$ is called Blum **integer**.

Examples

1. 1177 in A.6.
2. If p is a special prime, then $p \equiv 3 \pmod 4$. Therefore a product of two special primes is a Blum integer. Let us call such an integer a **special** Blum **integer**.

In general, if $n = pq$ with different odd prime numbers p and q, then $\mathbb{M}_n^2 \cong \mathbb{M}_p^2 \times \mathbb{M}_q^2$ has order $\frac{p-1}{2} \cdot \frac{q-1}{2}$, and this number is odd if and only if p and q both are $\equiv 3 \pmod 4$. Hence:

Lemma 25 *A product n of two odd prime numbers is a* Blum *integer if and only if the group $\mathbb{M}_n^2$ of quadratic residues has odd order.*

For a Blum integer -1 is a quadratic non-residue in $\mathbb{M}_p$ and $\mathbb{M}_q$, hence also in $\mathbb{M}_n$. But

$$(\frac{-1}{n}) = (\frac{-1}{p})(\frac{-1}{q}) = (-1)^2 = 1,$$

thus $-1 \in \mathbb{M}_n^+$. Hence

$$(\frac{-x}{n}) = (\frac{-1}{n})(\frac{x}{n}) = (\frac{x}{n})$$

for all x. Moreover $\mathbb{M}_n^2 \cap K = \{1\}$, thus the restriction of $\mathbf{q}$ to $\mathbb{M}_n^2$ is injective, hence bijective, and $\mathbb{M}_n$ is the direct product

$$\mathbb{M}_n = K \times \mathbb{M}_n^2, \quad \mathbb{M}_n^+ = \{\pm 1\} \times \mathbb{M}_n^2.$$

Each quadratic residue $a \in \mathbb{M}_n^2$ has exactly one square root in each of the four cosets of $\mathbb{M}_n/\mathbb{M}_n^2$. If $x \in \mathbb{M}_n^2$ is one of them, then the other ones are $-x$, τx, $-\tau x$. This shows:

Proposition 24 *Let n be a* Blum *integer. Then:*

(i) *If $x^2 \equiv y^2 \pmod{n}$ for $x, y \in \mathbb{M}_n$, and $x, -x, y, -y \bmod n$ are pairwise distinct, then $(\frac{x}{n}) = -(\frac{y}{n})$.*

(ii) *The squaring homorphism $\mathbf{q}$ is an automorphism of $\mathbb{M}_n^2$.*

(iii) *Each $a \in \mathbb{M}_n^2$ has has exactly two square roots in $\mathbb{M}_n^+$. If x is one of them, then $-x \bmod n$ is the other one, and exactly one of these two is itself a quadratic residue. Moreover a has exactly two more square roots, and these are contained in $\mathbb{M}_n^-$.*

Thus from the four square roots of a quadratic residue x exactly one is itself a quadratic residue. We consider this one as something special, and denote it by $\sqrt{x} \bmod n$. The least significant bit of x—also characterized as the parity of x, or as $x \bmod 2$—is denoted by $\operatorname{lsb}(x)$.

Corollary 1 *Let $x \in \mathbb{M}_n^+$. Then x is a quadratic residue if and only if*

$$\operatorname{lsb}(x) = \operatorname{lsb}(\sqrt{x^2} \bmod n).$$

Proof. If x is a quadratic residue, then $x = \sqrt{x^2} \bmod n$. Now assume x is a quadratic non-residue, and let $y = \sqrt{x^2} \bmod n$. By (iii) we have $y = -x \bmod n = n - x$. Since n is odd, x and y have different parities. $\diamond$

The problem of deciding quadratic residuosity $\bmod n$ remains hard. Only if the prime decomposition $n = pq$ is known there is an efficient solution:

$$x \in \mathbb{M}_n^2 \Longleftrightarrow (\frac{x}{p}) = (\frac{x}{q}) = 1.$$

We know of no efficient procedure that works without using the prime factors. *Presumably* deciding quadratic residuosity is equivalent with factoring in the sense of complexity theory. Generally believed to be true is the

Quadratic Residuosity Assumption: Deciding quadratic residuosity for Blum integers is hard.

A mathematical sound definition of "hard" is in Section B.4.

A.12 The Multiplicative Group Modulo Special Blum Integers

Let $p = 2p' + 1$ be a special prime. Then the multiplicative group $\mathbb{M}_p = \mathbb{F}_p^\times$ is cyclic of order $p - 1 = 2p'$. Its subgroup $\mathbb{M}_p^2 \leq \mathbb{M}_p$ of quadratic residues has index 2 and is itself cyclic, its order being the prime p'. Thus

$$\begin{aligned} \mathbb{M}_p &\cong \mathcal{Z}_{2p'}, & \#\mathbb{M}_p &= \varphi(p) = \lambda(p) = 2p', \\ \mathbb{M}_p^2 &\cong \mathcal{Z}_{p'}, & \#\mathbb{M}_p^2 &= p'. \end{aligned}$$

Let $n = pq$ be a special Blum integer, $p = 2p' + 1$ and $q = 2q' + 1$ being special primes. Then we know that

$$\begin{aligned} \mathbb{M}_n &\cong \mathbb{M}_p \times \mathbb{M}_q, & \#\mathbb{M}_n &= \varphi(n) = 4p'q', \\ \mathbb{M}_n^2 &\cong \mathbb{M}_p^2 \times \mathbb{M}_q^2, & \#\mathbb{M}_n^2 &= p'q'. \end{aligned}$$

Moreover $\lambda(n) = \operatorname{lcm}(2p', 2q') = 2p'q'$. Since $\mathbb{M}_n^2$ as a direct product of two cyclic groups of coprime orders is itself cyclic of order $p'q'$ we conclude:

Proposition 25 *Let n be a special* Blum *integer as above. Then the group $\mathbb{M}_n^2$ of quadratic residues* $\bmod n$ *is cyclic of order $p'q'$ and consists of*

(i) 1 *element of order* 1,

(ii) $p' - 1$ *elements x of order p', characterized by $x \bmod q = 1$,*

(iii) $q' - 1$ *elements x of order q', characterized by $x \bmod p = 1$,*

(iv) $(p' - 1)(q' - 1)$ *elements of order $p'q'$.*

Note that these numbers sum up to $p'q'$, the order of $\mathbb{M}_n^2$.

Corollary 1 *Let n be a special* Blum *integer with prime factors $p = 2p'+1$ and $q = 2q' + 1$. Then the probability $\eta = P\{x \in \mathbb{M}_n^2 \mid \operatorname{ord}(x) = p'q'\}$ that a randomly chosen quadratic residue* $\bmod n$ *has the maximum possible order $p'q'$ is*

$$\eta = 1 - \frac{p' + q' - 1}{p'q'}.$$

If we follow the common usage of choosing (RSA or) BBS modules n as products of two l-bit primes, or p' and q' as $(l - 1)$-bit primes, then

$$2^{l-1} < p' < 2^l, \quad 2^{l-1} < q' < 2^l,$$

$$2^l < p' + q' - 1 < 2^{l+1}, \quad 2^{2l-1} < p' \cdot q' < 2^{2l},$$

$$\frac{1}{2^l} = \frac{2^l}{2^{2l}} < \frac{p' + q' - 1}{p'q'} < \frac{2^{l+1}}{2^{2l-1}} = \frac{1}{2^{2l-3}} = \frac{8}{2^l}.$$

We resume

Corollary 2 *Let n be a special* BLUM *integer with prime factors $p = 2p' + 1$ and $q = 2q' + 1$ of bitlengths l. Then the probability η is bounded by*

$$1 - \frac{8}{2^l} < \eta < 1 - \frac{1}{2^l}.$$

The deviation of this probability from 1 is asymptotically negligible: If we choose a random quadratic residue x (say as the square of a random element of $\mathbb{M}_n$), then with overwhelming probability its order has the maximum possible value. However there is an easy test: Check that neither $x \bmod p$ nor $x \bmod q$ is 1.

Since $\mathbb{M}_n$ is the direct product of $\mathbb{M}_n^2$ with a KLEIN four-group we also know the orders of the elements of $\mathbb{M}_n$ and their numbers, in particular

Corollary 3 *Let n be a special* BLUM *integer with prime factors $p = 2p' + 1$ and $q = 2q' + 1$. Then $\mathbb{M}_n$ has exactly $(p' - 1)(q' - 1)$ elements of order $p'q'$, and exactly $3(p' - 1)(q' - 1)$ elements of order $2p'q'$.*

A.13 The BBS Sequence

Let n be a positive integer. Let x be invertible $\bmod n$, and let $s := \operatorname{ord}(x)$ be its order in the multiplicative group $\bmod n$.

Lemma 26 *For each integer r we have*

$$r \equiv 1 \pmod{s} \iff x^r \equiv x \pmod{n}.$$

Proof. "$\Longrightarrow$": Let $r = 1 + c \cdot s$. Then

$$x^r = x^{1+c\cdot s} \equiv x \cdot 1 = x \bmod n.$$

"$\Longleftarrow$": Dividing $\bmod n$ by the invertible element x gives

$$x^{r-1} \equiv 1 \pmod{n},$$

hence $s \mid r - 1$. ◇

Now let $x_0 := x$, and define the **BBS sequence** of integers x_i by the recursive formula $x_i = x_{i-1}^2$ for $i \geq 1$, or

$$(1) \qquad x_i = x^{2^i} \bmod n \quad \text{for } i = 0, 1, 2, 3, \ldots$$

Lemma 27 *The BBS sequence (x_i) is purely periodic if and only if $s = \operatorname{ord}(x)$ is odd. Then the period ν equals the multiplicative order of $2 \bmod s$.*

Proof. Assume the sequence is purely periodic with period ν. Then ν is minimal with $x_\nu \equiv x_0 \pmod{n}$. Hence

$$x_0^{2^\nu} \equiv x_0 \pmod{n}.$$

Thus $s \mid (2^\nu - 1)$ by Lemma 26, and ν is minimal with this property too, or with $2^\nu \equiv 1 \bmod s$. In particular s is odd, and ν is the order of $2 \bmod s$.

Conversely assume that s is odd. Then 2 is invertible $\bmod s$. Let μ be the multiplicative order of $2 \bmod s$. Then $2^\mu \equiv 1 \bmod s$, hence $x_\mu = x^{2^\mu} \equiv x_0 \bmod n$ by Lemma 26, thus the sequence is purely periodic. ◇

Proposition 26 *Let n be a* Blum *integer and x be a quadratic residue $\neq 1 \bmod n$. Then the BBS sequence x_i as defined in (1) is purely periodic of period $\nu = \operatorname{ord}_s(2)$.*

Proof. Assume $n = pq$ where p and q are two different odd primes $\equiv 3 \bmod 4$. Let $p = 4k + 3$ and $q = 4l + 3$ with integers k and l. Then the multiplicative group $\mathbb{M}_n$ has order $(p - 1)(q - 1) = (4k + 2)(4l + 2)$. The group $\mathbb{M}_n^2$ of quadratic residues has index 4 in $\mathbb{M}_n$, hence order $(2k + 1)(2l + 1)$, an odd integer. Thus every quadratic residue has odd order, and Lemma 27 applies for x. $\diamond$

Corollary 4 *Let n be a* Blum *integer and ν, the period of a BBS sequence. Then $\nu \mid \lambda(\lambda(n))$ where λ is the* Carmichael *function.*

Proof. By Proposition 26 we have $\nu = \operatorname{ord}_s(2) \mid \lambda(s)$. Moreover $s = \operatorname{ord}_n(x) \mid \lambda(n)$, hence $\lambda(s) \mid \lambda(\lambda(n))$. We conclude that $\nu \mid \lambda(\lambda(n))$. $\diamond$

A.14 The BBS Sequence for Superspecial Blum Integers

Again we get the most satisfying results in the superspecial case:

Definition A **superspecial** Blum **integer** is a product of two different superspecial primes.

Examples The two smallest superspecial primes are $p = 23$ (with $p' = 11$, $p'' = 5$) and $q = 47$ (with $q' = 23$, $q'' = 11$). Thus the smallest superspecial Blum integer is $n = 23 \cdot 47 = 1081$. By Section 2.1 we are confident (however don't know for sure) that there are very many superspecial Blum integers.

Now let $n = pq$ be a superspecial Blum integer with $p = 2p'+1 = 4p''+3$ and $q = 2q' + 1 = 4q'' + 3$. Form the BBS sequence (1) for an initial value $x \in \mathbb{M}_n^2 - \{1\}$. Then $s = \operatorname{ord}_n(x)$ takes one of the values p', q', or $p'q'$, the last on with extremely high probability, and the first two may be excluded by an easy check. The period of the BBS sequence is $\nu = \operatorname{ord}_s(2)$ by Proposition 26, and we may assume that $s = p'q'$. By the chinese remainder theorem and Lemma 21

$$\nu = \operatorname{lcm}(\operatorname{ord}_{p'}(2), \operatorname{ord}_{q'}(2))$$

By the Corollary of Proposition 23 in Section A.9

$$\begin{aligned}
\operatorname{ord}_{p'}(2) &= \begin{cases} 2p'' & \text{if } p'' \equiv 1 \pmod 4), \\ p'' & \text{if } p'' \equiv 3 \pmod 4), \end{cases} \\
\operatorname{ord}_{q'}(2) &= \begin{cases} 2q'' & \text{if } q'' \equiv 1 \pmod 4), \\ q'' & \text{if } q'' \equiv 3 \pmod 4), \end{cases}
\end{aligned}$$

Thus finally we have shown:

Proposition 27 *Let n be a superspecial* Blum *integer. Let x be a quadratic residue* $\bmod n$ *with $x \not\equiv 1 \pmod p$ and $x \not\equiv 1 \pmod q$. Then the BBS sequence* $\bmod n$ *for x has period*

$$\nu = \begin{cases} p''q'' & \textit{if } p'' \equiv q'' \equiv 3 \pmod 4, \\ 2p''q'' & \textit{otherwise.} \end{cases}$$

If p'' and q'' are $(l-2)$-bit primes (hence $> 2^{l-3}$, and n is an l-bit integer), then the period is $> 2^{l-2}$ or about $n/4$.

Appendix B

Complexity Theory for Cryptology

For (at least) three reasons "ordinary" complexity theory (using TURING machines) is insufficient for cryptologic needs:

1. Complexity theory primarily addresses the question whether the *worst case* of a problem is hard (i. e. a solution is not efficiently computable). However to preclude an efficient cryptanalysis we want the *normal case* to be hard. We saw that the existence of strong basic cryptographic functions implies $\mathbf{P} \neq \mathbf{NP}$, but conversely this inequality (if true) would *not* suffice to prove the existence of strong cryptography.

   From other parts of mathematics we are warned that a worst case scenario may not suffice to make a problem hard. For instance the NEWTON algorithm for determining roots of polynomials and the simplex method for linear optimization are hard in the worst case, but very efficient in the normal case.

2. The cryptanalyst is free to use *probabilistic* algorithms ("Monte Carlo" algorithms) that are very efficient but don't give a correct result in all cases. We saw several examples for number theoretic problems.

   The exact mathematical treatment uses concepts from probability theory: Parts of the input are taken from a probability space Ω. The results are statements on the distribution of the output.

3. Moreover the cryptanalyst is free to adapt her methods to the concrete problem at hand. She doesn't necessarily need a universal algorithm that is efficient for *all* instances of her problem. For example she could choose a different algorithm depending on the key length. Thus we have to consider *non-uniform* models of computation.

As a consequence the ordinary theory of TURING machines is insufficient for formalizing complexity theory as it is needed in cryptology. We could remedy this shortage by considering families of TURING machines—say a different one for each input length—, and also admitting probabilistic input.

However an alternative model of computation, using Boolean circuits, has a simpler, more intuitive description and a more direct and elegant application: families of probabilistic circuits (FPC for short). Realizing common algorithms by circuits is distinctly simpler and more intuitive than programming a TURING machine.

B.1 Probabilistic Boolean Circuits

A Boolean circuit describes an algorithm in the form of a flow chart that connects the single bit operations, see Appendix C.12 of Part II. It has two supplemental generalizations:

a probabilistic circuit formalizes probabilistic algorithms,

a family of circuits allows to express the complexity of an algorithm for increasing input sizes.

First we formalize the concept of a probabilistic algorithm for computing a map

$$f\colon A \longrightarrow \mathbb{F}_2^s$$

on a set A. To this end we consider maps (to be represented by circuits)

$$C\colon A \times \Omega \longrightarrow \mathbb{F}_2^s$$

where Ω is a probability space. We look at the probabilities that C "computes" $f(x)$ or f:

$$P(\{\omega \mid C(x,\omega) = f(x)\}) \quad (\text{"locally" at } x) \text{ and}$$

$$P(\{(x,\omega) \mid C(x,\omega) = f(x)\}) \quad (\text{"globally"})$$

that we want to be "significantly" $> \frac{1}{2^s}$, the probability of hitting a value in $\mathbb{F}_2^s$ by pure chance. In the local case we average over Ω for fixed x, in the global case we average also over $x \in A$. In general we assume that the probability spaces Ω and $A \times \Omega$ are finite and (in most cases) uniformly distributed.

In order to describe probabilistic algorithms we need circuits with *three* different types of input nodes:

- r **deterministic input nodes** that are seeded by an input tuple $x \in \mathbb{F}_2^r$, or x from a subset $A \subseteq \mathbb{F}_2^r$,
- some **constant input nodes**, each a priori set to 0 or 1,
- k **probabilistic input nodes** that are seeded by an element ("event") of the LAPLACEan probability space $\Omega = \mathbb{F}_2^k$ (corresponding to k "coin tosses"), or by an element of a subset $\Omega \subseteq \mathbb{F}_2^k$.—Sometimes also other probability distributions on Ω, different from the uniform distribution, might be taken into account.

The theory aims at statements on the probabilities of the output values $y \in \mathbb{F}_2^s$.

Examples

1. Searching a quadratic non-residue for an n bit prime module p. Here we choose a random $b \in [1 \dots p-1]$ and compute $(\frac{b}{p})$ (the Legendre symbol that is 1 for quadratic residues, -1 for quadratic non-residues). The success probability is $\frac{1}{2}$, the cost $\mathrm{O}(n^2)$ (see Appendix A.8).

   More generally we ask whether an h-tuple

   $$(b_1, \dots, b_h) \in \Omega = [1 \dots p-1]^h$$

   of independently choosen elements contains a quadratic non-residue. There is a probabilistic circuit (for the given p) without deterministic input nodes (but with some constant input nodes to input p),

   $$C : \mathbb{F}_2^{hn} \longrightarrow \mathbb{F}_2^n,$$

   $$C(\omega) = \begin{cases} b_i, & \text{the first } b_i \text{ that is a quadratic non-residue,} \\ 0 & \text{if none of the } b_i \text{ is a quadratic non-residue,} \end{cases}$$

   of size $\mathrm{O}(hn^2)$ that outputs a quadratic non-residue with probability $1 - \frac{1}{2^h}$. Note the deviation of this example from the definition above: Here C doesn't compute an explicitly given function f but provides output with a certain property.

2. The strong pseudoprime test: Here the input is taken from the set $A \subseteq [3 \dots 2^n - 1]$ of odd integers. We want to compute the primality indicator function

   $$f : A \longrightarrow \mathbb{F}_2, \quad f(m) = \begin{cases} 1 & \text{if } m \text{ is composite,} \\ 0 & \text{if } m \text{ is prime.} \end{cases}$$

   The probabilistic input consists of a base $a \in \Omega = [2 \dots 2^n - 1]$. The strong pseudoprime test is represented by a circuit

   $$C : \mathbb{F}_2^n \times \mathbb{F}_2^n \longrightarrow \mathbb{F}_2$$

   of size $\mathrm{O}(n^3)$, and yields the result 1 if m fails (then m is proven to be composite), 0 if m passes (then m is possibly prime). Thus C outputs the correct result only with a certain probability.

Now we formalize the property of a (probabilistic) circuit C of computing the correct value of $f(x) \in \mathbb{F}_2^s$ with a probability that “significantly” differs from a random guess: Given $\varepsilon \geq 0$, a circuit

$$C : \mathbb{F}_2^r \times \Omega \longrightarrow \mathbb{F}_2^s$$

(with r deterministic input nodes) has an **ε-advantage** for the computation of $f(x)$ or f if

$$P(\{\omega \in \Omega \mid C(x,\omega) = f(x)\}) \geq \frac{1}{2^s} + \varepsilon \quad (\text{“local case”}) \text{ or}$$

$$P(\{(x,\omega) \in A \times \Omega \mid C(x,\omega) = f(x)\}) \geq \frac{1}{2^s} + \varepsilon \quad (\text{“global case”}).$$

Thus in the global case the probability with respect to ω of getting a correct result is additionally averaged over $x \in A$. The advantage 0, or the probability $\frac{1}{2^s}$, corresponds to randomly guessing the result.

C has an **error probability** δ for computing $f(x)$ or f if

$$P(\{\omega \in \Omega \mid C(x,\omega) = f(x)\}) \geq 1 - \delta \ \text{ or}$$

$$P(\{(x,\omega) \in A \times \Omega \mid C(x,\omega) = f(x)\}) \geq 1 - \delta.$$

Examples

1. For searching a quadratic non-residue mod p we have

$$P(\{\omega \in \Omega \mid C(\omega) \text{ is a quadratic non-residue}\}) = 1 - \frac{1}{2^h}.$$

   Thus the circuit has an $(\frac{1}{2} - \frac{1}{2^h})$-advantage and an error probability of $\frac{1}{2^h}$.

2. For the strong pseudoprime test we have for fixed m

$$P(\{\omega \in \Omega \mid C(m,\omega) = f(m)\}) \begin{cases} \geq \frac{3}{4} & \text{if } m \text{ is composite,} \\ = 1 & \text{if } m \text{ is prime.} \end{cases}$$

   Averaging over m we get

$$P(\{(m,\omega) \in A \times \Omega \mid C(m,\omega) = f(m\}) \geq \frac{3}{4},$$

   hence an $\frac{1}{4}$-advantage and an error probability of $\frac{1}{4}$. (Since the number of composite integers is much larger than the number of primes, the value $\frac{1}{4}$ is not significantly changed by averaging over m.)

B.2 Polynomial Size Families of Circuits

A circuit has a fixed number of input nodes. Therefore it can process inputs of a fixed length only, in contrast with a Turing machine. However to assess the efficiency of an algorithm in general we have to estimate the increase of cost for increasing input sizes.

To this end we consider families $(C_n)_{n\in\mathbb{N}}$ of circuits with an increasing number of deterministic input nodes. Then the cost of a computation may be expressed as a function of the length of the input.

More exactly we define: A **family of probabilistic circuits (FPC)** is a family $C = (C_n)_{n\in\mathbb{N}}$,

$$(1) \qquad C_n \colon \mathbb{F}_2^{r(n)} \times \mathbb{F}_2^{k(n)} \longrightarrow \mathbb{F}_2^{s(n)},$$

where the circuit C_n has $r(n)$ deterministic input nodes, and $k(n)$ probabilistic ones. Of course, if all $k(n) = 0$, we speak of a family of deterministic circuits.

A **polynomial size family of probabilistic circuits (PPC)** is an FPC $C = (C_n)_{n\in\mathbb{N}}$, such that $\#C_n \le \alpha(n)$ for all $n \in \mathbb{N}$ with a polynomial $\alpha \in \mathbb{N}[X]$ (non-negative integer coefficients). In particular the number of input nodes of all kinds, as well as the number $s(n)$ of output nodes, is polynomially bounded. (We don't require that the functions r, k, s themselves are polynomials.)

Even in the deterministic case this model of computation might be able to compute more functions than the common model of Turing machines (and it is in fact), since it allows to choose a different algorithm for each input length. For this reason we also speak of a "non-uniform computational model". On first sight this feature seems not so pleasant. Nevertheless it is particularly realistic for cryptanalysis: Depending on the input size n the cryptanalyst may choose a suitable algorithm.

If a Turing machine computation in polynomial time is possible, then for the same problem there is a PPC also. The reverse statement is *not* true, although we only know "artificial" counterexamples.

Should any NP-complete problem be computable by a PPC, then so would be all the other ones. Virtually nobody believes in this possibility.

Non-uniform complexity may be modelled by Turing machines also, simply admitting a different Turing machine for each input length. Moreover we could also define probabilistic Turing machines. After all preferring the Shannon model of circuits over Turing machines is a matter of taste.

A computational problem is called **hard** if there is no PPC that solves it with a distinguished advantage; we'll make this definition more precise in the next sections. The "hard number theoretic problems" from Chapter 5, such as prime decomposition, are conjectured to be hard in this sense.

We know already that the basic operations on integers are computable by (even deterministic) PPC's. And therefore so are all algorithms on integers that are efficiently computable in the naive sense, using only “polynomially many” elementary arithmetic operations.

B.3 Efficient Algorithms

To generalize the results from Section B.1 we first define the concepts of advantage and error probability for PPCs.

Let $L \subseteq \mathbb{F}_2^*$ be a language over the binary alphabet $\mathbb{F}_2$, and set $L_n := L \cap \mathbb{F}_2^n$. Let f be a map

$$(2) \qquad f\colon L \longrightarrow \mathbb{F}_2^* \ \text{ with } f(L_{r(n)}) \subseteq \mathbb{F}_2^{s(n)}$$

where $r(n)$ is the monotonically increasing sequence of indices i with $L_i \neq \emptyset$. We want to compute this map by a PPC as in (1).

Examples

1. The function $f(x, y, z) := xy \bmod z$ for n-bit integers x, y, z is computable by a (deterministic) circuit

$$C_n\colon \mathbb{F}_2^{3n} \longrightarrow \mathbb{F}_2^n$$

   of size $\#C_n = \mathrm{O}(n^3)$ (with error probability 0). Here $r(n) = 3n$ and $s(n) = n$.

2. Let L be the set of (binary encoded) odd integers ≥ 3, and $f\colon L \longrightarrow \mathbb{F}_2$ be the primality indicator as in Section B.1. There we saw a PPC for the strong pseudoprime test of size $\mathrm{O}(n^3)$ with advantage $\frac{1}{4}$ and error probability $\frac{1}{4}$ (constant with respect to n). Using t bases we get a size of $\mathrm{O}(tn^3)$, and an error probability of $\frac{1}{4^t}$.

Definition 1 A function $\varphi\colon \mathbb{N} \longrightarrow \mathbb{R}_+$ is called **(asymptotically) negligible** if for each nonconstant polynomial $\eta \in \mathbb{N}[X]$

$$\varphi(n) \leq \frac{1}{\eta(n)} \quad \text{for almost all } n \in \mathbb{N}.$$

In other words, $\varphi(n)$ tends to 0 faster than the inverse of any polynomial.

Example An obvious example is $\varphi(n) = 2^{-n}$.

Definition 2 Sei $f\colon L \longrightarrow \mathbb{F}_2^*$ be as in (2). Let C be a PPC that computes f on $L_{r(n)}$ with an error probability of ε_n. Assume ε_n is a negligible function of n. Then C is called an **efficient probabilistic algorithm** for f.

f is called **(probabilistically) efficiently computable** if there is an efficient algorithm for f.

This definition substantiates the idea of an algorithm that is “efficient for almost all input tuples” (or input strings if the input is taken from a language L).

For RABIN’s primality test, that is the repeated execution of the strong pseudoprime test, we satisfy this requirement by letting the number t of bases grow with n. In order to get a polynomial family we upgrade t to a polynomial $\tau \in \mathbb{N}[X]$. Then C_n has n deterministic input nodes, and $n\tau(n)$ probabilistic ones. The size is $\mathrm{O}(n^3\tau(n))$, and the error probability, $\frac{1}{4^{\tau(n)}}$. Thus we have shown:

Proposition 28 RABIN*’s primality test is an efficient probabilistic algorithm for deciding primality.*

B.4 Hard Problems

Exactly defining what a hard problem is is somewhat more tricky. We want to characterize a problem that has *no efficient solution for almost all* input tuples (or strings). Simply negating the property "efficient" is clearly insufficient. Somewhat better is the requirement that the advantage of an algorithm approaches 0 with increasing n. But also this is not yet a suitable definition since the advantage describes a lower bound only.

A better requirement is the non-existence of an advantage that approaches 0 too slowly. "Too slowly" is

$$\frac{1}{\eta(n)} \quad \text{with an arbitrary polynomial } \eta \in \mathbb{N}[X].$$

"Slow enough" is for instance the inverse exponential finction $1/2^n$.

Moreover there should be "almost no" exceptions, or the set of exceptions should be "sparse". Now we try to translate these ideas into an exact definition.

For $x \in L_{r(n)}$ we consider the probability

$$p_x := P(\{\omega \in \Omega_{k(n)} \mid C_n(x,\omega) = f(x)\}),$$

and the set of input strings x for which C_n has an ε-advantage:

$$L_{r(n)}(\varepsilon) := \{x \in L_{r(n)} \mid p_x \geq \frac{1}{2^{s(n)}} + \varepsilon\}.$$

For a polynomial $\eta \in \mathbb{N}[X]$ the set $L_{r(n)}(\frac{1}{\eta(n)})$ consists of the input strings x for which C computes $f(x)$ with advantage $\frac{1}{\eta(n)}$. Thus the exceptional set for η is

$$L^{[f,C,\eta]} := \bigcup_{n\in\mathbb{N}} L_{r(n)}(\frac{1}{\eta(n)}).$$

We denote it as "**advantageous set for f, C, η**". Its components should become more and more marginal with increasing n. The definition is:

Definition 3 A subset $A \subseteq L$ is called **sparse** if

$$\frac{\#A_n}{\#L_n}$$

is negligible.

Remarks and Examples

1. If $\#A_n = c$ is constant, and $L_n = \mathbb{F}_2^n$, then A is sparse in L for

$$\frac{\#A_n}{\#L_n} = \frac{c}{2^n}.$$

2. If $\#A_n$ grows at most polynomially, but $\#L_n$ grows faster than any polynomial, then A is sparse in L.

3. If $\#A_n = c \cdot \#L_n$ is a fixed proportion, then A is not sparse in L.

4. If $L = \mathbb{N}$, and A is the set of primes (in binary coding), then by the prime number theorem
$$\#A_n \approx \frac{2^{n-1}}{n \cdot \ln(2)} = \frac{\#L_n}{n \cdot \ln(2)}.$$
Hence the set of primes is not sparse in $\mathbb{N}$.

5. No known efficient algorithm is able to factorize a non-sparse subset of the set M of all products of primes whose lengths differ by at most one bit.

Definition 4 Let f be as in (2). Then f is called **hard** if for each PPC as in (1) and for each polynomial $\eta \in \mathbb{N}[X]$ the advantageous set $L^{[f,C,\eta]}$ is a sparse subset of L.

Examples

1. The conjecture that prime decomposition of integers is hard makes sense by remark 5.

2. **Quadratic residuosity conjecture:** Let B be the set of Blum integers (products of two primes $\equiv 3 \pmod 4$),
$$L = \{(m,a) \mid m \in B, a \in \mathbb{M}_n^+\},$$
(for $\mathbb{M}_n^+$ see Appendix A.5) and let
$$f \colon L \longrightarrow \mathbb{F}_2$$
be the indicator function
$$f(m,a) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue mod } m, \\ 0 & \text{else.} \end{cases}$$
Then f is hard. (A forteriori when we more generally admit $a \in \mathbb{M}_n$.)

B.5 Basic Cryptographic Functions

Now the theoretic basis suffices for an exact definition of one-way functions and strong symmetric ciphers. Note that the "functions" or "maps" in these definitions are infinite families with growing input size. There is no mathematically sound definition of one-way or hash functions, or of strong symmetric ciphers, for a fixed input size, as we assumed in treating these concepts in a naive way in Section 4.1 and Chapters 5 and 6

Definition 5 Let $f : L \longrightarrow \mathbb{F}_2^*$ be as in (2). A right inverse of f is a map $g \colon f(L) \longrightarrow L \subseteq \mathbb{F}_2^*$ with $f(g(y)) = y$ for all $y \in f(L)$. In other words g finds pre-images of f. We call f a **one-way function** if each right inverse of f is hard.

Adapting this definition the conjecture that the discrete exponential function in finite prime fields is hard makes sense.

Now for the definition of a strong cipher. An "ordinary" block cipher is a map

$$F \colon \mathbb{F}_2^r \times \mathbb{F}_2^q \longrightarrow \mathbb{F}_2^r.$$

The corresponding decryption function is a map

$$G \colon \mathbb{F}_2^r \times \mathbb{F}_2^q \longrightarrow \mathbb{F}_2^r$$

with $G(F(x, k), k) = x$ for all $x \in F_2^r$ and $k \in F_2^q$.

An attack with known plaintext finds a key $k \in F_2^q$ with $F(x, k) = y$, given $x, y \in \mathbb{F}_2^r$. We formalize this by a map

$$H \colon \mathbb{F}_2^r \times \mathbb{F}_2^r \longrightarrow \mathbb{F}_2^q$$

with $F(x, H(x, y)) = y$ for all $x, y \in \mathbb{F}_2^r$ with $y \in F(x, \mathbb{F}_2^q)$ ("possible pairs" (x, y)).

Exercise Give an exact definition of a possible pair.

A more general attack uses several, say s, plaintext blocks. So it defines a map

$$H \colon \mathbb{F}_2^{rs} \times \mathbb{F}_2^{rs} \longrightarrow \mathbb{F}_2^q$$

with $F(x_i, H(x_i, y_i)) = y_i$ for $i = 1, \ldots s$ for all possible $x, y \in \mathbb{F}_2^{rs}$.

Now we give a definition in terms of complexity theory.

Definition 6 A **symmetric cipher** is a family $F = (F_n)_{n\in\mathbb{N}}$ of block ciphers

$$F_n \colon \mathbb{F}_2^{r(n)} \times \mathbb{F}_2^{q(n)} \longrightarrow \mathbb{F}_2^{r(n)}$$

with strictly monotonically increasing functions r and q, such that $F_n(\bullet, k)$ is bijective for each $k \in \mathbb{F}_2^{q(n)}$, and

- F is efficiently computable,
- there is an efficiently computable family $G = (G_n)_{n\in\mathbb{N}}$ of corresponding decryption functions.

Definition 7 An **known plaintext attack** on a symmetric cipher F is a family $H = (H_n)_{n\in\mathbb{N}}$ of maps

$$H_n \colon \mathbb{F}_2^{r(n)s(n)} \times \mathbb{F}_2^{r(n)s(n)} \longrightarrow \mathbb{F}_2^{q(n)}$$

with

$$F_n(x_i, H_n(x_i, y_i)) = y_i \quad \text{for } i = 1, \ldots, s(n)$$

for all possible pairs $x, y \in \mathbb{F}_2^{r(n)s(n)}$.

F is called a **strong symmetric cipher** if each known plaintext attack on F is hard.

Defining a hash function is even more tricky. We omit it.