# Euler's totient and RSA

Application of discrete models

## 1 Euler's totient function

Earlier (see Congruences) we have seen that $a \in \mathbb{Z}$ is invertible modulo $m$ if and only if $a$ and $m$ are coprime, i.e. $\gcd(a, m) = 1$. For example, modulo 10, the invertible remainders are 1, 3, 7 and 9 (because everything else is divisible by either 2 or 5).

An important property about the invertible numbers is that their product is also invertible:

**Theorem 1.1.** *If $a$ and $b$ are invertible modulo $m$, then $ab$ is also invertible modulo $m$.*

**Example.** For $m = 10$, $3 \cdot 9 \equiv 7\ (10)$, $3 \cdot 3 \equiv 9\ (10)$, $3 \cdot 7 \equiv 1\ (10)$ etc.

We have seen a special case of this in the previous document (Discrete logarithm), where the modulus was a prime number, because then all nonzero remainders are invertible, and we have seen that the product of two nonzero remainders is also nonzero. But we have also seen that it is *not* true if the modulus is not prime (e.g. $4 \cdot 5 \equiv 0\ (10)$), and now we can see that it is because the proper way to generalize this is to change "nonzero" to "invertible".

We have also introduced the notion of *complete residue systems*, e.g. $\{0, 1, 2, \ldots, m-1\}$ (see Congruences). Now let's see a similar notion, which only contains the invertible numbers:

**Definition 1.2.** If $S$ is a complete residue system (modulo $m$), then $\{a \in S \mid \gcd(a, m) = 1\}$ is a **reduced residue system** (modulo $m$).

**Example.** All of the following sets are reduced residue systems modulo 10:
$\{1, 3, 7, 9\}$, $\{11, 13, 17, 19\}$, $\{1, 53, 77, -21\}$ etc.

Similarly to complete residue systems, there are infinitely many reduced residue systems for any $m$ (but usually we are only interested in the one that contains the remainders, e.g. $\{1, 3, 7, 9\}$). Also, for any given $m$, they have the same number of elements, e.g. for $m = 10$, they have always 4 elements.

But how many elements do these sets have in general, i.e. how many invertible remainders are there for a given $m$? This is measured by the following function:

**Definition 1.3.** For $n \in \mathbb{N}^+$, let $\boldsymbol{\varphi(n)}$ be the number of integers between 0 and $n$–1 that are coprime to $n$, i.e. $\varphi(n) := |\{k \in \mathbb{Z} \mid 0 \leq k \leq n-1 \ \wedge\ \gcd(n, k) = 1\}|$. It is called **Euler's totient function** (or *Euler's $\varphi$ function*). ($\varphi$ is pronounced "phi".)

**Example.** $\varphi(10) = |\{1, 3, 7, 9\}| = 4$ and $\varphi(7) = |\{1, 2, 3, 4, 5, 6\}| = 6$.

In Sage, $\varphi(n)$ is called `euler_phi(n)`. Let's print its first few values:

```
sage: matrix([[n, euler_phi(n)] for n in range(1, 26)]).transpose()
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]
[ 1  1  2  2  4  2  6  4  6  4 10  4 12  6  8  8 16  6 18  8 12 10 22  8 20]
```

We can also plot them:

```
sage: plot(euler_phi, 1, 100)
```

The most important properties of $\varphi(n)$:

1. $\varphi(1) = 1$ (since $\gcd(1, 0) = 1$).
2. If $n \geq 2$, then $\varphi(n) \leq n-1$ (since $\gcd(n, 0) = n \neq 1$).
3. If (and only if) $p$ is prime, then $\varphi(p) = p-1$, because all numbers between 1 and $p-1$ are coprime to $p$. (So the highest possible values of $\varphi(n)$ are for prime numbers.)
4. If $p$ is prime, then $\varphi(p^k) = p^k - p^{k-1} = (p-1)p^{k-1}$, e.g. $\varphi(16) = \varphi(2^4) = 2^4 - 2^3 = 8$. (Proof: from the total $p^k$ numbers, we need to remove the multiples of $p$, whose number is $p^k/p = p^{k-1}$.)
5. If $a$ and $b$ are coprime, then $\varphi(ab) = \varphi(a) \cdot \varphi(b)$. For example, $\varphi(10) = \varphi(2) \cdot \varphi(5) = 1 \cdot 4 = 4$. (The proof is complicated, so it is omitted.) Important: this is only true if $a$ and $b$ are coprime! Otherwise for example $\varphi(2) \cdot \varphi(4) = 1 \cdot 2 = 2$, but $\varphi(2 \cdot 4) = \varphi(8) = 2^3 - 2^2 = 4 \neq 2$.

Using the above properties, we can calculate $\varphi(n)$ for any $n$ whose prime factorization is known.

**Example.** If $n = 24 = 2^3 \cdot 3$, then, since $2^3$ and 3 are coprime:

$$\varphi(24) = \varphi(2^3) \cdot \varphi(3) = (2^3 - 2^2) \cdot (3 - 1) = 4 \cdot 2 = 8.$$

In general, since powers of different prime numbers are always coprime, we can use the prime factorization of $n$ to split $\varphi(n)$ into $\varphi()$'s of prime powers, and then use the formula for $\varphi(p^k)$.

**Example.**

$$\varphi(4200) = \varphi(2^3 \cdot 3 \cdot 5^2 \cdot 7) = \varphi(2^3) \cdot \varphi(3) \cdot \varphi(5^2) \cdot \varphi(7) = (2^3 - 2^2) \cdot 2 \cdot (5^2 - 5) \cdot 6 = 960.$$

The general formula is the following:

**Theorem 1.4** (calculation of $\varphi$)**.** *If the canonical representation of $n$ is $n = p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m}$, then:*

$$\varphi(n) = \left(p_1^{k_1} - p_1^{k_1-1}\right)\left(p_2^{k_2} - p_2^{k_2-1}\right) \cdots \left(p_m^{k_m} - p_m^{k_m-1}\right).$$

As a special case, if $n$ is a product of two prime numbers, $n = p \cdot q$, then $\varphi(n) = (p-1)(q-1)$.

Note: we can see that it is very easy to calculate $\varphi(n)$ if we know the prime factorization of $n$. But if we don't know that – and as we will see, factoring a large $n$ is very difficult –, then we cannot calculate $\varphi(n)$ efficiently either. This will be of crucial importance later.

# 2 Modular exponentiation II

In the previous document (Discrete logarithm), we have discussed modular exponentiation, but mainly for prime modulus. In this section, we generalize it for arbitrary modulus.

For example, let's calculate the first few powers of all remainders modulo 10:

```
sage: matrix([[power_mod(a, k, 10) for k in range(30)] for a in range(10)])
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
[1 2 4 8 6 2 4 8 6 2 4 8 6 2 4 8 6 2 4 8 6 2 4 8 6 2 4 8 6 2]
[1 3 9 7 1 3 9 7 1 3 9 7 1 3 9 7 1 3 9 7 1 3 9 7 1 3 9 7 1 3]
[1 4 6 4 6 4 6 4 6 4 6 4 6 4 6 4 6 4 6 4 6 4 6 4 6 4 6 4 6 4]
[1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5]
[1 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6]
[1 7 9 3 1 7 9 3 1 7 9 3 1 7 9 3 1 7 9 3 1 7 9 3 1 7 9 3 1 7]
[1 8 4 2 6 8 4 2 6 8 4 2 6 8 4 2 6 8 4 2 6 8 4 2 6 8 4 2 6 8]
[1 9 1 9 1 9 1 9 1 9 1 9 1 9 1 9 1 9 1 9 1 9 1 9 1 9 1 9 1 9]
```

Comparing this with the case of prime modulus from eariler, we can observe the following:

- Here 1 doesn't always reappear in all rows. For some rows, it appears periodically, but for others, $a^0 = 1$ is the only 1.
- Eventually, each row repeats periodically, but the first reappearing power is not always 1.
- For which bases does 1 repeat? For 1, 3, 7 and 9, i.e. exactly for the invertible numbers.
- The *order* of these numbers (i.e. the first positive exponent for which the power is 1) can be 1, 2 or 4 (e.g. $\mathrm{ord}_{10}(7) = 4$). Notice that they are all divisors of 4.
- Just like for prime modulus, it is also true here that $\mathrm{ord}_{10}(1) = 1$ and $\mathrm{ord}_{10}(10-1) = 2$.

In order to generalize most of the above statements, we need a general theorem about modular exponentiation, similar to Fermat's little theorem, which said for a prime modulus that $a^{p-1} \equiv 1 \pmod{p}$ if $a \not\equiv 0 \pmod{p}$. The general version for arbitirary modulus is the following:

**Theorem 2.1** (Euler's totient theorem, a.k.a. Fermat–Euler theorem, or simply Euler's theorem)**.** *If* $m \in \mathbb{N}^+$, $a \in \mathbb{Z}$ *and* $\gcd(a, m) = 1$, *then*

$$a^{\varphi(m)} \equiv 1 \pmod{m}.$$

**Example.** For $m = 10$, $\varphi(10) = 4$, so $3^4 \equiv 1 \pmod{10}$, $7^4 \equiv 1 \pmod{10}$ etc.

So in general, it is not the $p-1$'st power that is 1, but the $\varphi(m)$'th power (for prime numbers, they are the same: $\varphi(p) = p-1$).

Very important: this theorem works only if $a$ and $m$ are coprime. If they are not, then not only the $\varphi(m)$'th power will not be 1, but no positive power at all. For example, for $m = 10$, all powers of even numbers are even, and all powers of 5 are 5. This is true in general: it is easy to see that any $a^n \bmod m$ is divisible by $\gcd(a, m)$. So if $\gcd(a, m) \neq 1$, i.e. they are not coprime, then 1 cannot appear among the powers. But if they *are* coprime, then Euler's theorem guarantees that 1 appears.

A consequence of Euler's totient theorem is that the powers repeat periodically after $\varphi(m)$ steps (if $\gcd(a, m) = 1$):

$$a^{n+\varphi(m)} \equiv a^n a^{\varphi(m)} \equiv a^n 1 \equiv a^n \pmod{m}.$$

The repetition may have a shorter period than $\varphi(m)$, but it is always true that the order ($\mathrm{ord}_m(a)$) divides $\varphi(m)$. For example, for $m = 10$, the order is either 1, 2 or 4 (see above), and indeed, they all divide $\varphi(10) = 4$. (This is a generalization of Lagrange's theorem, which was stated only for prime modulus in the previous document.)

Note: although Euler's theorem only holds for invertible bases, so complete repetition is only possible for them (since for the others, not even 1 repeats), but the powers of other bases will also repeat *eventually*. It can be proven that the length of this eventual period is also at most $\varphi(m)$, more precisely a divisor of $\varphi(m)$.

An important application of Euler's theorem is to simplify the exponents of modular powers:

**Example.** Let's calculate the modular power $137^{75} \bmod 10$. First, of course, we can take the base modulo 10:

$$137^{75} \equiv (137 \bmod 10)^{75} \equiv 7^{75} \pmod{10}.$$

For the exponent, the same thing doesn't work ($7^{75} \not\equiv 7^5 \pmod{10}$). Instead, we can use Euler's theorem, and realize that we can take the remainder of 75, not modulo 10, but modulo $\varphi(10) = 4$:

$$7^{75} \equiv 7^{75 \bmod 4} \equiv 7^3 \pmod{10}.$$

This works because Euler's theorem states that $7^{\varphi(10)} \equiv 1 \pmod{10}$, i.e.:

$$7^{75} \equiv 7^{18 \cdot 4 + 3} \equiv \left(7^4\right)^{18} 7^3 \equiv 1^{18} \cdot 7^3 \equiv 7^3 \pmod{10}.$$

In this way, we have simplified $137^{75}$ to $7^3$, which is much easier to calculate:

$$7^3 \equiv 7^2 \cdot 7 \equiv 49 \cdot 7 \equiv 9 \cdot 7 \equiv 63 \equiv 3 \pmod{10}.$$

The above simplification has a theoretical and a practical limitation: first, Euler's theorem requires that the base must be coprime to the modulus (which was true here: $\gcd(7, 10) = 1$), and second, we need to be able to calculate $\varphi(m)$ for the modulus $m$. The latter requires the prime factorization of $m$, i.e. it works only if $m$ is not too big, or if it has a special form.

Summary: if we can calculate $\varphi(m)$, and $\gcd(a, m) = 1$, then the modular power $a^n \bmod m$ can be calculated by taking the base modulo $m$, and the exponent modulo $\varphi(m)$:

$$a^n \equiv (a \bmod m)^{n \bmod \varphi(m)} \pmod{m},$$

which is an easier power to calculate (and we can e.g. use fast exponentiation, see the previous document).

But what if the base is not coprime to the modulus? Consider for example $138^{75} \bmod 10$ (where $\gcd(138, 10) = 2$). The first step still works:

$$138^{75} \equiv (138 \bmod 10)^{75} \equiv 8^{75} \pmod{10}.$$

But now we can't use Euler's theorem directly, because $\gcd(8, 10) \neq 1$. Instead, the idea is to split the modulus into two parts: $10 = 5 \cdot 2$, and split the problem along with it: instead of $8^{75} \bmod 10$, calculate $8^{75} \bmod 5$ and $8^{75} \bmod 2$. The point of this is that for the first part, $\gcd(8, 5) = 1$, so we can use Euler's theorem to simplify it:

$$8^{75} \equiv 3^{75} \equiv 3^{75 \bmod 4} \equiv 3^3 \equiv 2 \pmod{5},$$

and the second part is trivial:

$$8^{75} \equiv 0^{75} \equiv 0 \pmod{2}.$$

These two results can be combined into the final result using the Chinese remainder theorem (CRT, see Congruences):

$$\left.\begin{aligned} x &\equiv 2 \pmod{5} \\ x &\equiv 0 \pmod{2} \end{aligned}\right\} \quad \longrightarrow \quad x \equiv 2 \cdot (-2) \cdot 2 + 0 \cdot 1 \cdot 5 \equiv -8 \equiv 2 \pmod{10}.$$

# 3 The RSA algorithm

RSA is a commonly used cryptosystem, i.e. a method to encrypt and decrypt messages. Its name comes from the initials of the creators (Rivest, Shamir and Adleman). It is used in various places such as internet communication (e.g. HTTPS, SSH) and bank security.

Warning: the point of the description below is merely to illustrate the main points of RSA, so it contains certain simplifications (this variant is often called "textbook RSA"). Therefore, this information is not sufficient for a practical, secure implementation of RSA, so it should not be used in this form to protect any real-life data. (In general, implementing cryptographic algorithms – other than toy examples – should be left to experts in the field, because even the smallest mistakes can cause serious security flaws.)

## 3.1 Asymmetric encryption

RSA is an *asymmetric* encryption algorithm (also known as *public-key* encryption), which means that it uses two different keys: one for encryption, and one for decryption (i.e. to restore the encrypted message). This is in contrast to *symmetric-key* encryptions (such as the commonly used AES, which will be discussed in a later document), which use the same key for both encryption and decryption. Symmetric-key encryptions are usually simpler and much faster than the asymmetric ones, but their drawback is that both parties must know the same secret key. So they either have to meet in private to agree on this shared key, or use an algorithm such as the Diffie–Hellman key exchange to create one (see the previous document: Discrete logarithm).

Asymmetric encryptions however, such as RSA, have two different, but related keys:

- **Public key**: it can be shared publicly, and it is used to encrypt messages.
- **Private key**: it must be kept secret by its owner, and it is used to decrypt the messages that were encrypted by the corresponding public key.

This means that the two parties, say Alice and Bob, don't have to agree on a secret key, instead, Alice publishes her public key, and Bob uses it to encrypt a message for her, which can only be decrypted by Alice's private key. Therefore, anyone can send encrypted messages to Alice, but only she can decrypt them.

In order for an asymmetric encryption to work, the two keys must be related, because one must decrypt what the other has encrypted; but also, they must be different enough so that one cannot be calculated from the other (or at least the private key from the public key), otherwise the whole system would be pointless. These two requirements seemingly contradict with each other. Also, if one key cannot be calculated from the other, then how are they created in the first place?

## 3.2 How RSA works

The RSA algorithm is based on the fact that prime factorization is an extremely difficult task for very large numbers. There is no known general algorithm that can efficiently calculate the prime factors of a sufficiently large integer. That is, if we have two big prime numbers, $p$ and $q$, we can easily multiply them: $n := p \cdot q$, but nobody will be able to calculate $p$ and $q$ back from $n$. (Assuming that they are large enough, and they satisfy certain other mathematical requirements, which are not detailed here.)

Note: this is similar to the discrete logarithm (see the previous document) in that it also had an operation, the modular exponentiation, which is easy to perform, but its inverse, the discrete logarithm is hard. Such an operation is called a *one-way function*.

The main operation of RSA, just like for the Diffie–Hellman key exchange, is modular exponentiation, but in this case, the modulus is not a prime number, but the product of two primes: $n = p \cdot q$. Both the encryption and the decryption are exponentiations modulo $n$, but to different exponents:

- for a message $m$ (the *plaintext*), the encryption calculates $c \equiv m^e \pmod{n}$, where $c$ is the encrypted message (the *ciphertext*) and $e$ is the *public exponent*;
- then, the decryption restores $m$ from the ciphertext $c$ by calculating $m \equiv c^d \pmod{n}$, where $d$ is the *private exponent*.

The question is of course how to create $e$ and $d$ so that the above calculation works.

In order for the decryption to restore the same $m$ that was encrypted, we need

$$m \equiv c^d \equiv (m^e)^d \equiv m^{ed} \pmod{n}.$$

We know from Euler's totient theorem (see above) that the exponent can be taken modulo $\varphi(n)$, i.e. $m^{ed} \equiv m^{ed \bmod \varphi(n)} \pmod{n}$. So, if $e$ and $d$ satisfy $ed \equiv 1 \pmod{\varphi(n)}$ – i.e. $d$ is the inverse of $e$ modulo $\varphi(n)$ –, then we get $m^{ed} \equiv m \pmod{n}$ as desired.

(Note: if $\gcd(m, n) \neq 1$, then we can't use Euler's totient theorem, but it can still be proven in other ways that $m^{ed} \equiv m \pmod{n}$.)

The private exponent ($d$) can only be calculated from the public exponent ($e$) if we know $\varphi(n)$. But in order to calculate $\varphi(n)$, we need the prime factorization of $n$, i.e. $p$ and $q$. So the security of RSA is indeed based on the fact that prime factorization is difficult.

The first step of RSA is **key generation**:

1. Generate two different random secret prime numbers: $p$ and $q$.
2. Calculate their product: $n := p \cdot q$.
3. Calculate $\varphi(n)$: $\varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$.
4. Choose a public exponent $e$ for which $2 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$.
5. Solve the congruence $ed \equiv 1 \pmod{\varphi(n)}$ for $d$, i.e. calculate the inverse of $e$ modulo $\varphi(n)$.
6. The public key is the pair $(n, e)$, which can be published.
7. The private key is the pair $(n, d)$, which must be kept secret.

(Note: not only $d$, but also $p$, $q$ and $\varphi(n)$ are secret values, though they are not needed anymore, so they can be deleted.)

Then, RSA performs **encryption** and **decryption** as follows:

1. Write the plaintext message as an integer $m$ between 0 and $n-1$.
2. Encryption: calculate $c \equiv m^e \pmod{n}$ using the public key $(n, e)$.
3. Decryption: calculate $m \equiv c^d \pmod{n}$ using the private key $(n, d)$.

(Note: converting the message to an integer $m$ is not trivial. First, a too long message must be split into smaller chunks. Second, the trivial, bitwise conversion does not result in a secure encryption; instead, we need to add some random bits to the message (a.k.a. *padding*). But this is far beyond the scope of this document.)

Just like for the Diffie–Hellman key exchange, the two prime numbers $p$ and $q$ must be generated using a cryptographically secure pseudorandom number generator (CSPRNG, see the previous document), so that an outside listener cannot possibly predict them. It is also important that the prime numbers must be large enough (and appropriately chosen), so that $n$ cannot be factored into $p$ and $q$. Nowadays, it is common to use 2048-bit or 4096-bit RSA, which means that the binary representation of $n$ has 2048 or 4096 bits (so that $p$ and $q$ each have 1024 or 2048 bits, respectively).

And how do we generate $e$, the public exponent? Since it is public, it doesn't need to be chosen randomly, but it can be chosen to make the calculation more efficient. In practice, it is common to use the value $e = 2^{16} + 1 = 65537$, because it is simple enough to make modular exponentiation efficient (remember that fast exponentiation depends on the length and the number of 1 bits in the binary representation), but it is also complex enough that it doesn't risk the security of RSA.

## 3.3 Digital signature

As we have seen, RSA encryption works by encrypting the message using the public key, and decrypting it using the private key. In this way, anyone can send an encrypted message to a recipient, but only that person, i.e. the owner of the private key can decrypt it.

But what happens if we reverse that? What happens if we encrypt a message using the private key, and decrypt it using the public key? Then only the owner of the private key can encrypt messages, but anyone can decrypt them. Does it make any sense?

This is called *digital signature*, which is another important application of RSA besides encryption. In this case, the message $m$ is "encrypted" using the private key, i.e. the value $s \equiv m^d \pmod{n}$ is calculated, and this $s$ – the *signature* – is attached to the message $m$. Then the recipient calculates $m \equiv s^e \pmod{n}$ from $s$, and if it matches the received $m$, then the signature verification is successful, i.e. the recipient can be certain that the message was indeed written by the person who possesses the private key.

### 3.4 Improvement using CRT

We have seen that the public exponent, $e$ can be chosen to make encryption as efficient as possible. But the private exponent, $d$ cannot be influenced directly, so it can be very big – which is of course desirable from the security point of view, but it means that decryption, i.e. raising to the exponent $d$ is much slower than encryption, i.e. raising to $e$.

But we can improve the speed of decryption as follows. Instead of calculating $m \equiv c^d \pmod{n}$ directly, we do it in two parts: since $n = p \cdot q$, we can calculate this exponentiation separately for $p$ and $q$:

$$m \equiv c^d \pmod{n} \quad \longrightarrow \quad \begin{cases} m_p \equiv c^d \pmod{p}, \\ m_q \equiv c^d \pmod{q}. \end{cases}$$

The two results can be combined using the Chinese remainder theorem (see Congruences). The point of this is that not only can we calculate with smaller numbers (because of the smaller modulus), but we can also reduce the exponents using Fermat's little theorem (see Discrete logarithm): the actual exponents we need to raise $c$ to are $d_p := d \bmod (p-1)$ and $d_q := d \bmod (q-1)$, respectively. So although we perform two modular exponentiations instead of one, each one is more than four times faster than the original one, because the exponents are half as long, so we only need half as many multiplications, and the multiplied numbers are also half as long.

So the improved version calculates $m \equiv c^d \pmod{n}$ as follows:

1. Calculate the following values beforehand:
   - $d_p := d \bmod (p-1)$,
   - $d_q := d \bmod (q-1)$,
   - the inverse of $q$ (i.e. $q^{-1}$) modulo $p$.
2. $m_p := c^{d_p} \bmod p$.
3. $m_q := c^{d_q} \bmod q$.
4. Calculate $m$ from $m_p$ and $m_q$ using the Chinese remainder theorem (CRT):
   - $m_d := q^{-1}(m_p - m_q) \bmod p$.
   - $m := m_q + m_d q$.

Because of this, the private key is often not only $(n, d)$, but the tuple $(p, q, d_p, d_q, q^{-1})$.

# 4 Author

This document was written by Uray M. János, partly using the following sources:


[1] https://compalg.elte.gitlab-pages.hu/dimoa-web/tematika/dimoa (Hungarian)

[2] https://en.wikipedia.org/wiki/Euler%27s_totient_function

[3] https://en.wikipedia.org/wiki/RSA_cryptosystem

[4] Rivest, Shamir, Adleman (1977): A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. https://publications.csail.mit.edu/lcs/pubs/pdf/MIT-LCS-TM-082.pdf



Errors and suggestions can be reported here: uray.janos@inf.elte.hu.