Rotor Machines

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

December 3, 1999—English version November 9, 2013—last change
January 19, 2021

1 One-Rotor Ciphers

See the web page `http://www.staff.uni-mainz.de/pommeren/Cryptology/Classic/5_Rotor/OneRotor.html`

2 Mathematical Description of Rotors

Identify the alphabet Σ with $\mathbb{Z}/n\mathbb{Z}$, the integers mod n. Let ρ be the monoalphabetic substitution that the rotor performs in its initial position. Moving the rotor by one position forward the new substitution is

$$\rho^{(1)}(a) = \rho(a-1) + 1$$

Denote by τ the shift by 1 of the alphabet $\Sigma = \mathbb{Z}/n\mathbb{Z}$, that is $\tau(a) = a + 1$. Then the formula looks like this:

$$\rho^{(1)}(a) = \tau\rho\tau^{-1}(a)$$

By induction we immediately get part (i) of the following theorem:

Theorem 1 (The secondary alphabets of a rotor)

(i) *If a rotor in its initial position performs the substitution with the primary alphabet ρ, then after rotation by t positions forward it performs the substitution with the conjugate alphabet $\rho^{(t)} = \tau^t\rho\tau^{-t}$. In particular all secondary alphabets have the same cycle type.*

(ii) *The diagonals of the corresponding alphabet table each contain the standard alphabet (cyclically wrapped around).*

Proof. Assertion (i) is proved above. Assertion (ii) follows immediately by interpreting it as a formula:

$$\rho^{(i)}(j) = \tau^i\rho\tau^{-i}(j) = \rho(j-i) + i = \rho^{(i-1)}(j-1) + 1$$

◇

The definition of "cycle type" was given in Appendix A.

The formula makes it obvious why—in contrast with the cipher disk—for a rotor the (unpermuted) standard alphabet is completely useless: It corresponds to the identity permutation, therefore all its conjugates are identical.

In general the conjugate alphabet $\rho^{(t)}$ is identical with the primary alphabet ρ if and only if ρ is in the centralizer of the shift τ^t. The designer of a rotor might wish to avoid such wirings.

Examples.

1. If n is a prime number, then all the shifts τ^t for $t = 1, \ldots, n-1$ are cycles of length n. Therefore all their centralizers are identical to the cyclic group $< \tau >$ spanned by τ. If the designer avoids these n trivial wirings, then all the n conjugated alphabets are distinct.

2. If $\gcd(t, n) = d > 1$, then τ^t splits into d cycles of length $\frac{n}{d}$, $\tau^t = \pi_1 \cdots \pi_d$, and centralizes all permutations of the type $\pi_1^{s_1} \cdots \pi_d^{s_d}$. These are not in the cyclic group $< \tau >$ unless all exponents s_i are congruent mod $\frac{n}{d}$.
3. In the case $n = 26$ the shifts τ^t are cycles, if t is coprime with 26. However τ^t splits into two cycles of length 13, if t is even. All the powers τ^t, t even, $2 \leq t \leq 24$, span the same cyclic group because 13 is prime. The permutation τ^{13} splits into 13 transpositions. For example τ^2 centralizes the permutation $(ACE \ldots Y)$, and τ^{13} centralizes the transposition (AB), where we denoted the alphabet elements by the usual letters A, ..., Z. Therefore in wiring the rotors the designer should avoid the centralizers of τ^2 and of τ^{13}.

3 Cryptanalysis of One-Rotor Ciphers (with Unknown Alphabet)

See the web page `http://www.staff.uni-mainz.de/pommeren/Cryptology/Classic/5_Rotor/Anal1Rot.html`

4 Rotor Machines

General Description

Rotor machines are electromechanical devices that consist of several rotors in series connection. Figure 1 gives an impression of the electric flow through such a machine.

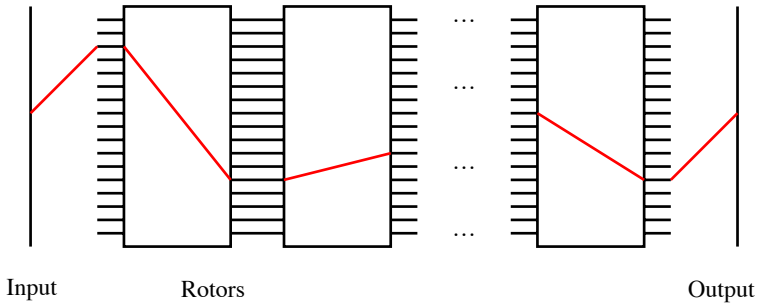

Figure 1: *Rotor machine circuit*

With each input letter the rotors move in individual ways, some by one position, some by several positions, some only after several steps. *The cryptographic security of a rotor machine depends* on the number of rotors, the multitude of different settings, and, in a crucial way, *on the complexity of the rotor movements.*

Operating a Rotor Machine

The operator hits a key on the keyboard that corresponds to the next plaintext letter. This action closes an electric circuit powering a light-bulb that corresponds to the ciphertext letter. Or it powers a type bar that prints the ciphertext letter. The rotors move according to their control logic, in general before the circuit is closed. See the FAQ at `http://www.staff.uni-mainz.de/pommeren/Cryptology/FAQ.html`.

Rotor machines are the state of the art in encryption during the period from 1920 until 1970. The mystic and irregularly rotating wheelwork that makes the desk tremble with each key hit looks very attractive and impresses the general or diplomat who wants to buy security.

Mathematical Description

The following abstract model describes an idealized rotor machine. Concrete historic machines each have their own peculiar details.

As before we identify the alphabet Σ with $\mathbb{Z}/n\mathbb{Z}$, the integers mod n. A rotor machine has the following characteristic parameters:

- A set $R \subseteq \mathcal{S}(\Sigma)$ of $p = \#R$ rotors. Each of these defines a primary alphabet, that is a permutation $\rho_i \in \mathcal{S}(\Sigma)$ that corresponds to the wiring of the rotor.

- A choice $\rho = (\rho_1, \ldots, \rho_q) \in \mathcal{S}(\Sigma)^q$ of q different rotors $\rho_i \in R$. There are $p \cdot (p-1) \cdots (p-q+1)$ choices if we assume that all rotors are differently wired ($q \leq p$). This choice serves as "primary key" and is usually fixed for several messages, say for an entire day.

- A state vector $z = (z_1, \ldots, z_q) \in (\mathbb{Z}/n\mathbb{Z})^q$ that describes the current rotor positions. The initial state $z^{(0)}$ serves as "secondary key" that usually changes with each message. The number of different initial states is n^q. Sometimes it is convenient to map the states to $\mathbb{Z}/n^q\mathbb{Z}$, the integers $\bmod n^q$, using the representation of integers in base n. The state vector $z = (z_1, \ldots, z_q) \in (\mathbb{Z}/n\mathbb{Z})^q$ then corresponds to the integer $\zeta = z_1 \cdot n^{q-1} + \cdots + z_q$.

- A state-transition function

  $$g \colon \mathbb{N} \times \Sigma^q \longrightarrow \Sigma^q$$

  that transforms the state at time i, $z^{(i)}$, to the state at time $i+1$, $z^{(i+1)} = g(i, z^{(i)})$, where "time" is discrete and simply counts the plaintext letters. This function g represents the control logic and is realized for example by more or less complex gear drives. In most rotor machines the state-transition function is independent of the time i.

- The substitution in state z:

  $$\sigma_z := \rho_q^{(z_q)} \circ \cdots \circ \rho_1^{(z_1)} \quad \text{where } \rho_j^{(z_j)} := \tau^{z_j} \circ \rho_j \circ \tau^{-z_j}$$

Ideally the map $\Sigma^q \longrightarrow \mathcal{S}(\Sigma)$, $z \mapsto \sigma_z$ would be injective, that is each state defines a different substitution. Unfortunately no useful general results seem to exist beyond the case $q = 1$ treated in Subsection 2.

Perl programs for encryption and decryption by rotor machines are in the web directory `http://www.staff.uni-mainz.de/pommeren/Cryptology/Classic/Perl/` as `rotmach.pl` and `rotdecr.pl`.

The Key Space

By the description above a key of our idealized rotor machine consists of

- a choice of rotors

- an initial state

Therefore the key space K has

$$\#K = n^q \cdot \frac{p!}{(p-q)!}$$

elements. In a typical example (HEBERN machine) we have $p = q = 5$, $n = 26$, $\#K = 120 \cdot 26^5 = 712882560$, and the effective key length is $d(F) \approx 29.4$. That was good enough in 1920. Today, against an enemy with a computer, this is much too little.

> In fact the HEBERN machine was not good enough even in 1920 because it allows attacks far more efficient than exhaustion.

Encryption and Decryption

The plaintext $a = (a_1, \ldots, a_r) \in \Sigma^r$ is encrypted by the formula

$$c_i = \sigma_{z^{(i)}}(a_i)$$

At full length this formula reads

$$c_i = \tau^{z_q^{(i)}} \circ \rho_q \circ \tau^{z_{q-1}^{(i)} - z_q^{(i)}} \circ \cdots \circ \tau^{z_1^{(i)} - z_2^{(i)}} \circ \rho_1 \circ \tau^{-z_1^{(i)}}(a_i)$$

Decryption follows the formula

$$a_i = \tau^{z_1^{(i)}} \circ \rho_1^{(-1)} \circ \tau^{z_2^{(i)} - z_1^{(i)}} \circ \cdots \circ \tau^{z_q^{(i)} - z_{q-1}^{(i)}} \circ \rho_q^{(-1)} \circ \tau^{-z_q^{(i)}}(c_i)$$

Technically for decryption we simply have to route the current through the machine in the reverse direction, of course interchanging the keyboard and lightbulbs. The sequence of states is identical for encryption and decryption.

The Rotor Machine as a Finite-State Automaton

Figure 2 shows an abstract model of a rotor machine.

Usually the state-transition function is independent of the step i. Then it has the simpler form

$$g \colon \Sigma^q \longrightarrow \Sigma^q$$

This makes the states periodic as shown in the next subsection.

Periods of State Changes

Let M be a finite set with $m = \#M$. We may think of the elements of M as "states". Consider a map ("state transition")

$$g : M \longrightarrow M.$$

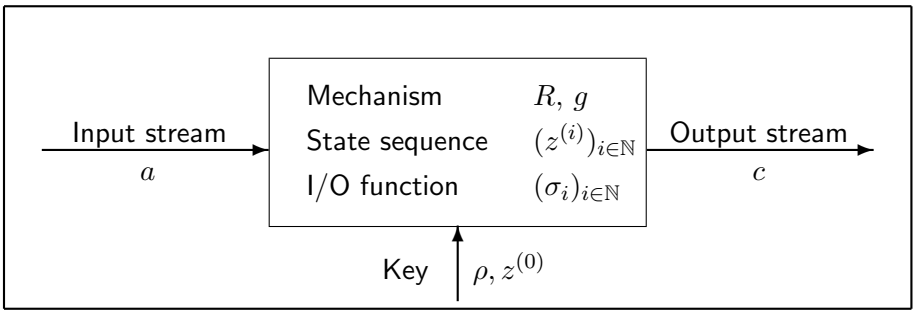

Figure 2: *Rotor machine as finite-state automaton*

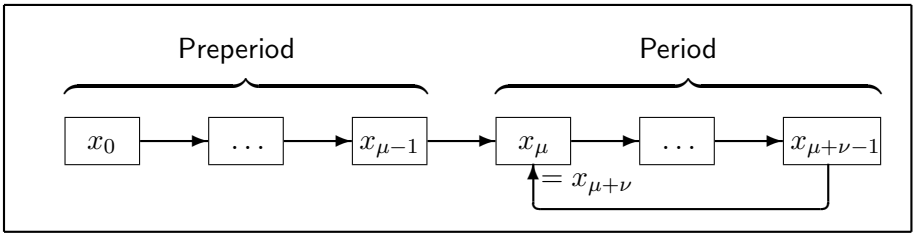

Figure 3: *Period and preperiod*

For each element ("initial state") $x_0 \in M$ we define a sequence $(x_i)_{i\in\mathbb{N}}$ in M by the recursion formula $x_i = g(x_{i-1})$ for $i \geq 1$. After a preperiod of length μ this sequence becomes periodic with a period of ν, see Figure 3, an explanation follows below.

Since M is finite there are smallest integers $\mu \geq 0$ and $\nu \geq 1$ such that $x_{\mu+\nu} = x_\mu$: Take for μ the smallest index such that the element x_μ reappears somewhere in the sequence, and for $\mu+\nu$ the index where the first repetition occurs. Then also

$$x_{i+\nu} = x_i \quad \text{for } i \geq \mu.$$

Obviously $0 \leq \mu \leq m-1$, $1 \leq \nu \leq m$, $\mu+\nu \leq m$. The values $x_0, \ldots, x_{\mu+\nu-1}$ are all distinct, and the values $x_0, \ldots, x_{\mu-1}$ never reappear in the sequence.

Definition: μ is called (length of the) **preperiod**, ν is called (length of the) **period**.

5 The Control Logic of a Rotor Machine

We treat several approaches to rotor stepping. The first three are streamlined versions of real control mechanisms that in practice are implemented in a more complex way: the odometer, the gear drive with gaps, the gear drive with different number of cogs. We also treat the ultimate mechanism: the pseudorandom stepping, and a historical one: the HEBERN mechanism. For the stepping of the Enigma we refer to Chapter 6.

The insight that an irregular movement is the essential ingredient for a secure rotor machine is apparently due to FRIEDMAN after he broke he HEBERN machine. He himself, together with his collaborator ROWLETT, then in several steps developed the top-level rotor machine, the SIGABA.

Example 1: The Odometer Logic

The rotors step like in a mechanical counter or electricity meter. Assume the rotors are mounted as in Figure 4. The rightmost rotor moves by one position for each input letter. Each rotor, after completing one revolution, by some kind of protrusion makes its left neighbor move by one position.

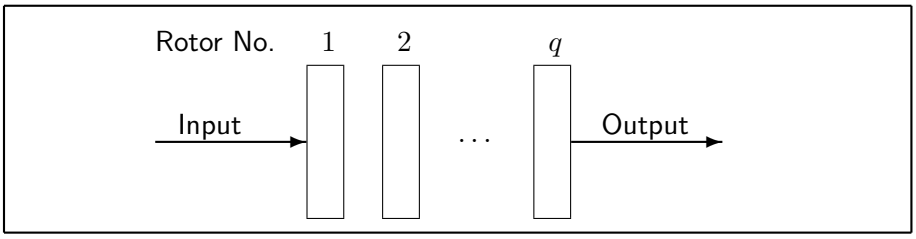

Figure 4: *Odometer logic*

Using the identification of the states with the integers mod n^q the sequence of states simply corresponds to the natural sequence of numbers beginning with the initial state.

Remarks

1. In this example the rightmost rotor, rotor number q, is a "fast" rotor, it moves with every step.
2. The leftmost rotor, number 1, is a "slow" rotor. It moves only after n^{q-1} steps, that is almost never, or only for very long messages. For this reason it makes little sense to use more then three rotors with odometer stepping. The effect of all additional rotors together only amounts to a fixed substitution. In the best case they could move once during encryption, effecting two different fixed substitutions.

3. Of course we could also implement the converse stepping where rotor 1 is fast and rotor q is slow.
4. The sequence of states has period n^q.

Example 2: Gaps

Figure 5 shows the principle of this control logic. For an implementation we have several mechanical options, for example a pin wheel.

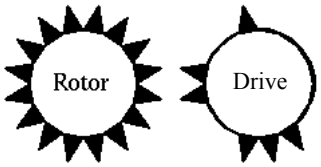

Figure 5: *Gear drive with tooth gaps*

A single wheel is characterized by a binary vector

$$u_{(j)} = (u_{j0}, \ldots, u_{j,t-1}) \in \mathbb{F}_2^t \quad \text{for } j = 0, \ldots, t-1$$

where t is the circumference of the wheel, not necessarily $t = n$. A 1 denotes a cog, a 0, a gap. We may describe all the wheels together by a binary matrix

$$u = \begin{pmatrix} u_{10} & \cdots & u_{1,t-1} \\ \vdots & \ddots & \vdots \\ u_{q0} & \cdots & u_{q,t-1} \end{pmatrix} \in M_{qt}(\mathbb{F}_2)$$

The column vectors

$$u^{(i)} = (u_{1i}, \ldots, u_{qi}) \in \mathbb{F}_2^q \quad \text{for } i = 0, \ldots, q-1$$

apply one after the other from left to right, cyclically repeated. This gives a sequence of period t for the states of the gear drive. The states of the rotors generally have a much larger period.

In the simplest case this logic steps the rotor j

- by one position, if $u_{ji} = 1$,
- not at all, if $u_{ji} = 0$,

for the i-th plaintext letter. This gives the formula

$$z^{(i+1)} = z^{(i)} + u^{(i)}$$

where addition is vector addition in $(\mathbb{Z}/n\mathbb{Z})^q$.

Another way to use gap wheels is turning them around a full turn in each step. Then the each of the rotors moves a number of steps given by the corresponding row sum in the matrix. This logic is equivalent with Example 3 below.

Example 3: Different Gear Wheels

Each rotor is driven by its own gear wheel. These share a common axis and make a full turn in each step. If wheel i has n_i cogs, then rotor i moves by n_i positions. The states occur with a period of $\operatorname{lcm}(n_1, \ldots, n_q)$.

The first models of Enigma (A and B) had a control like this.

Example 4: Pseudorandom Stepping

The rotor stepping is controlled by a (pseudo-) random generator, that is a mechanism or an algorithm that generates numbers indinguishable from pure random such as generated with the help of dice. This is easy for a computer simulation. For an (electro-) mechanical rotor machine one can use a key generating mechanism such as in one of the (later) HAGELIN machines.

FRIEDMAN was the first to detect the weaknesses of a regular rotor stepping when he analyzed the then current rotor machines in the 1920's. He came up with the idea of an irregular stepping by a pseudorandom mechanism. First he tried a punched paper tape, but this proved not robust enough. Then ROWLETT had the idea of realizing the stepping control by another set of rotors. Thus the American super rotor machine SIGABA was invented.

For details see the book

> Stephen J. Kelly: *Big Machines.* Aegean Park Press, Walnut Creek 2001, ISBN 0-89412-290-8.

Example 5: The HEBERN Machine

The HEBERN machine has $q = 5$ rotors and uses the standard alphabet with $n = 26$. The stepping follows an odometer logic, but with a complex mechanism that doesn't affect the neighboring rotor but another one, in more detail:

- Rotors 2 and 4 don't rotate at all. They are "stators".
- Rotor 5 moves by 1 position with every step, it is a fast rotor.
- Rotor 1 moves by 1 position with each complete turn of rotor 5. It is a "semi-fast" rotor.
- Rotor 3 moves by 1 position with each complete turn of rotor 1. It is a slow rotor.

Moreover the rotors move in the other direction compared with the description in Section 2.

The equation for the state change—not yet the correct one!—is

$$g(z_1, z_2, z_3, z_4, z_5) = (z_1 + \lambda(z_5), z_2, z_3 + \lambda(z_1)\lambda(z_5), z_4, z_5 + 1)$$

where $\lambda(x) = \delta_{x,25}$ is the KRONECKER symbol. The states occur with period $26^3 = 17576$.

Characteristic features:

- That the rotors 2 and 4 are static doesn't harm the security of the machine. By the odometer logic they would move only after 26^3 or 26^4 steps, that is only for extremely long messages.
- The stepping of rotor 1 (resp. 3) is induced by rotor 5 (resp. 1) moving from position "N" to position "O". The correct equation for the state change is left as an **exercise** to the reader.
- The wiring between the keyboard and rotor 1 as well as from rotor 5 to the light bulbs is irregular but static. It therefore is assumed as known to the enemy. We may interpret this wiring as two additional stators, one at each end of the rotor pack.
- For decryption there is a switch "direct/reverse" that interchanges input contacts and output contacts.
- The HEBERN rotors are symmetric: they may be mounted with their sides interchanged. This makes the number of possible primary keys larger by a factor of 2^5.

6 Historical Rotor Machines

See the web page `http://www.staff.uni-mainz.de/pommeren/Cryptology/Classic/5_Rotor/HistRot.html`

7 Historical Data on Cryptanalysis

See the web page `http://www.staff.uni-mainz.de/pommeren/Cryptology/Classic/5_Rotor/AnalHist.html`

8 Cryptanalysis of Rotor Machines

The cryptanalysis of rotor machines is complex and depends on the details of the machine under examination. The book by Deavours and Kruh [2] is a standard work and contains many elaborate examples. Here we only depict some general ideas:

- Superimposition
- Meet-in-the-middle
- Isomorphs

Superimposition

Assume that the cryptanalyst got hold of several ciphertexts that are encrypted with the same key, then he may align them in such a way that he gets monoalphabetically encrypted columns. Note that this is a ciphertext-only attack. However it needs lots of messages.

Note that operators impede this attack by changing the key (or initial position) for each message. Nevertheless in some scenarios they have to send many messages, think of war. Then with high probability the cryptanalyst will observe many ciphertexts that are produced by the same rotor positions, not necessarily at the same position in the text. She identifies these concordances by extensive calculation of coincidence indices.

Identification of a Fast Rotor

Assume that the set of rotors is known but not their actual choice. Assume that the last rotor at the output side steps by one position with each letter, and that the other rotors move infrequently. The attacker has no known plaintext.

Enumerate the rotors from 1 (= input rotor, slow) to q (= output rotor, fast), and assume the current flows from left to right as in Figure 6.

Now assume we have a ciphertext section of length m where only rotor q moved, and for simplicity use the indices 1 to m for this sequence of ciphertext letters. The rotors 1 to $q-1$ together effect a constant substitution μ.

Therefore this part of the encryption follows the schema

$$\begin{array}{ccccccc}
a_1 & \mapsto & b_1 := \mu(a_1) & \mapsto & \rho_q^{(z_1)}\mu(a_1) & = c_1 \\
a_2 & \mapsto & b_2 := \mu(a_2) & \mapsto & \rho_q^{(z_1+1)}\mu(a_2) & = c_2 \\
\vdots & & \vdots & & & \vdots \\
a_m & \mapsto & b_m := \mu(a_m) & \mapsto & \rho_q^{(z_1+m-1)}\mu(a_m) & = c_m
\end{array}$$

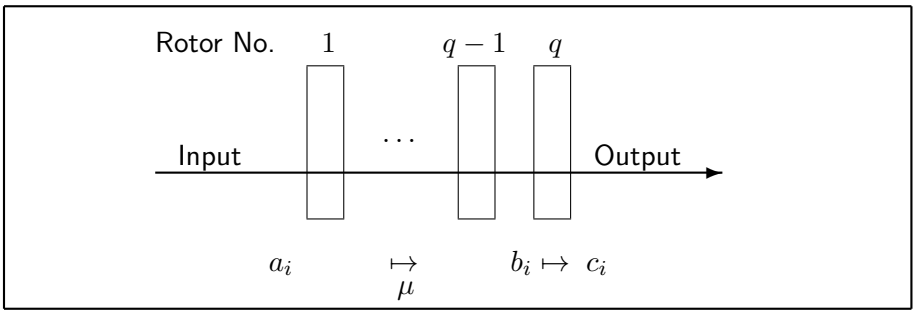

Figure 6: *Identifying the fast rotor*

Here $b = (b_1, \ldots, b_m) \in \Sigma^m$ is a monoalphabetic image of $a = (a_1, \ldots, a_m)$. We can also look at b "from the other side":

$$\begin{aligned} b_1 &= \left[\rho_q^{(z_1)}\right]^{-1}(c_1) \\ b_2 &= \left[\rho_q^{(z_1+1)}\right]^{-1}(c_2) \\ \vdots &\quad \vdots \\ b_m &= \left[\rho_q^{(z_1+m-1)}\right]^{-1}(c_m) \end{aligned}$$

These formulas enable an exhaustion of the p choices for rotor q and of the n choices for its initial position z_1.

- A wrong choice of the rotor or its initial position makes b look as a random text having coincidence index $\varphi(b) \approx \frac{1}{n}$.
- For the correct choice b is a monoalphabetically encrypted meaningful text having coincidence index $\varphi(b) \approx \kappa_M$, the coincidence index of the plaintext language.

This observation may lead to the identification of the fast rotor and its state for this section of the text at the price of $n \cdot p$ calculations of coincidence indices of texts of length m. But note that the coincidence test for $m = 26$ has little power, it will miss most positive events.

Remarks

1. In principle the method works at each position of the text. Therefore the very beginning of the text is worth a try.
2. In the unfavourable case one of the other rotors moved during the encryption of the m letters. Then the intermediate ciphertext b consists of two different monoalphabetic pieces. With a bit of luck this also leads to a somewhat conspicuous coincidence index.

Continuation of the Attack

As soon as the fast rotor is identified we can strip its effect off like a superencryption. In this way the intermediate ciphertext $(b_1, \ldots, b_m)$ extends to a ciphertext $c' \in \Sigma^r$ that is the result of encrypting the plaintext a by a much simpler machine.

If for example the rotors move like an odometer, and if the ciphertext is long enough ($\approx n^2$), then in a similar way we can identify the next rotor and strip its effect off.

Or we try to cryptanalyze the monoalphabetic parts of c' that we expect $\lfloor \frac{r}{n} \rfloor$ in number of length n plus one or two fragments of total length r mod n.

We also might first try to find the locations were the second rotor moves.

Known Plaintext Attack

Assume we know or guess a piece of plaintext $a = (a_1, \ldots, a_m)$, say a probable word. An essential step is finding text chunks with identical numerical patterns, also called **isomorphs**. Therefore this attack is known as **Method of Isomorphs**. More generally looking at an intermediate step of an encryption algorithm from both sides, is called **Meet-in-the-Middle**.

Identification of a Fast Output Rotor

If we have a piece of known plaintext we may identify a fast rotor by simple pattern comparisons without calculating coincidence indices: Check if the intermediate text $(b_1, \ldots, b_m)$ shows the same numerical pattern as $(a_1, \ldots, a_m)$.

Identification of a Fast Input Rotor

Known plaintext $a = (a_1, \ldots, a_m)$ also allows the identification of the fast rotor for a *reverse* odometer control where the left rotor is the fast one. In this case we consider the situation of Figure 7.

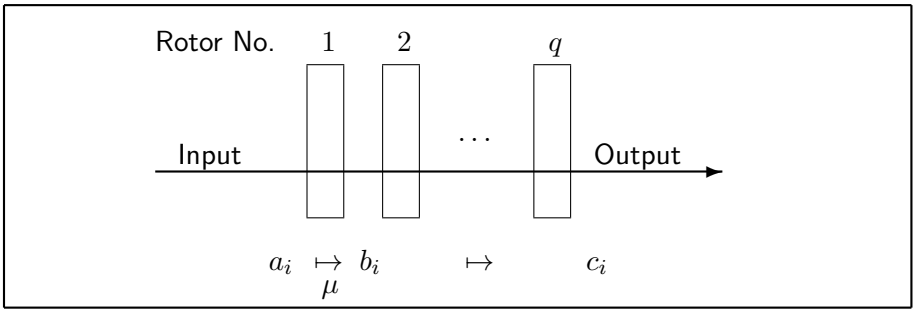

Figure 7: *Identifying a fast input rotor*

This part of the encryption follows the schema

$$\begin{array}{cccccc}
a_1 & \mapsto & b_1 := \rho_1^{(z_1)}(a_1) & \mapsto & \mu(b_1) & = c_1 \\
a_2 & \mapsto & b_2 := \rho_1^{(z_1+1)}(a_2) & \mapsto & \mu(b_2) & = c_2 \\
\vdots & & \vdots & & & \vdots \\
a_m & \mapsto & b_m := \rho_1^{(z_1+m-1)}(a_m) & \mapsto & \mu(b_m) & = c_m
\end{array}$$

Here $b = (b_1, \ldots, b_m)$ is a monoalphabetic image of $c = (c_1, \ldots, c_m)$. We try all p rotors in all their n initial positions until the numerical patterns of b and c coincide.

References

[1] Bauer, F. L. *Decrypted Secrets; Methods and Maxims of Cryptology.* Berlin: Springer 1997.

[2] Deavours, C. A., Kruh, L. *Machine Cryptography and Modern Cryptanalysis.* Norwood: Artech House 1985.

[3] Sinkov, A. *Elementary Cryptanalysis.* Washington: The Mathematical Association of America 1966.