Symulacja w Badaniach i Rozwoju
Vol. 6, No. 1/2015

Anna BOROWSKA
Politechnika Białostocka, Wydział Informatyki
ul. Wiejska 45A, 15-351 Białystok
E-mail: a.borowska@pb.edu.pl

The Cryptanalysis of the Enigma Cipher. The Catalogue Method. Part II

1 Introduction

The M3 Enigma machine was an electro-mechanical encrypting device used during World War II, mainly by the German military and government services. The catalogue algorithm given in part I of this paper can be used to decode messages eavesdropped before September 15, 1938. Elements of the key, which we can obtain thanks to this algorithm, are not sufficient enough to read messages. Therefore, the author provides her own algorithm which returns the *ring settings* and the *initial drum settings* and additionally, the new plugboard algorithm which relies on Rejewski's idea, but its technical solution is the author's proposal.

The German service used different kinds of Enigma machines, but we are only interested in the M3 Enigma machine. For the reader's convenience, we described the construction of this device and the manner of generating messages transmitted until September 15, 1938 in [2] (in the Appendix). This section makes up a brief survey of well-known information taken from publications [6, 10, 7, 8, 3, 4, 9]. We suggest reading the Appendix first for better understanding the terms and facts that we use. These terms are denoted in this paper by *. Section 2 contains a mathematical analysis of the M3 Enigma machine. In sections 3 we present the new plugboard algorithm. We use Rejewski's idea to read plug connections on the basis of suitably signed permutations, but we give our own proposal how to get all the dissimilar notations for these permutations quickly. In section 4 we propose the ring settings algorithm designed according to our idea. By means of these three algorithms (described in parts I and II) we can generate the complete *daily key** and read the *message settings** on the basis of a given set of messages intercepted before September 15, 1938. This allows us to read these messages. We enclose an implementation of given algorithms in Cpp language.

2 Mathematical model

The cryptosystem of the M3 Enigma machine will be defined as a tuple (**P**, **C**, **K**, **E**, **D**), where **P** = {A, B, …, Z} is the *plaintext space*, **C** = **P** is the *ciphertext space* and **K** is a set of all possible *keys*.

$\mathbf{E} = \{\Lambda_k : k \in \mathbf{K}\}$ is a set of *encryption functions* $\Lambda_k : \mathbf{P} \to \mathbf{C}$.

$\mathbf{D} = \{\Delta_k : k \in \mathbf{K}\}$ is a set of *decryption functions* $\Delta_k : \mathbf{C} \to \mathbf{P}$.

Each letter $a \in \mathbf{P}$ is transformed according to the following permutation (cf. [10]).

$$\Lambda_k = \Lambda = SHQ^{z}RQ^{-z}Q^{y}MQ^{-y}Q^{x}LQ^{-x}BQ^{x}L^{-1}Q^{-x}Q^{y}M^{-1}Q^{-y}Q^{z}R^{-1}Q^{-z}H^{-1}S^{-1} \qquad (1)$$

S - is a permutation describing the *plugboard** transformation (S consists of transpositions and 1-cycles only), B - is a permutation describing the *reflector** transformation,

L, M, R - are permutations describing transformations of the three *cipher drums**,

H - is a transformation of the *entry wheel** (H is the identity permutation),

Q = (ABCDEFGHIJKLMNOPQRSTUVWXYZ) – a cycle of length 26,

Ds[i] ($i = l, m, r$) - positions of *drums** (left, middle and right) before pressing any key,

Rs[i] ($i = l, m, r$) - positions of *rings** (left, middle and right) (Ds[i], Rs[i]$\in$**P**),

x, y, z - positions of *rotors** before pressing any key (values from set **IP** = {0,1,…,25}),

x = (Ds[l]-Rs[l])%26 for the left rotor,

y = (Ds[m]-Rs[m])%26 for the middle rotor,

z = (Ds[r]-Rs[r])%26 for the right rotor (cf. [6]).

We denote by Λ_H a permutation which we obtain by substituting in the formula (1) the identity permutation H for a permutation S, i.e. $\Lambda_k = \Lambda = S\Lambda_H S^{-1}$. We treat the letters A, B, …, Z of the Latin alphabet as the numbers from the set **IP**.

3 Plugboard algorithm

The plugboard algorithm generates all possible permutations S, which satisfy the equation $A_M D_M = SA_H D_H S^{-1}$. By using permutations A_M and D_M the first and the fourth letters of each message were coded on a given day. The product $A_H D_H$ is a result of the catalogue algorithm (cf. [2], section 5). One of the obtained permutations S represents plug connections. The cryptologists found this permutation S by hand. They placed the product $A_H D_H$ under the product $A_M D_M$ so that cycles of the same length (in both permutations) were written down one under the other. But they did not always obtain a solution quickly. The algorithm presented below relies on this idea but the author proposed the quick manner of generating all the dissimilar notations for couples of permutations on the basis of which we obtain the permutation S.

Tab. 1. Zestaw szyfrogramów wykorzystanych w eksperymentach

Tab. 1. The set of messages which were used in experiments

(1)	LIO	IWN	BVSVIWKOUYZKHEHCCNBKWHTMMI	(14)	SXR	ECR	BWTIDPHQJYQMCQHKUEFSJNPEAI
(2)	YCJ	OFQ	IQAOVGEDVSAYLVTJCQGYRPXWQU	(15)	EJT	BAO	SJLGXJEGWODMEAAFFUMPHMKEMQ
(3)	MKG	DDU	ZIJTCVRCXQWYYGYJGIWRHBSAHT	(16)	FFF	JLY	NSPMPLANTTXKRITJHAQCFRLNNF
(4)	RMV	CJH	JAXLOYCKQKHBKSXHCILQNOLKVH	(17)	ZGZ	KSM	IYSZTRZQJHVQKSQRNAGOLBMYHH
(5)	OOH	NRL	DNHNDEKDDQSJAFVURZIWVJTGBK	(18)	DNK	GQX	VNGKDZVVPHRBAUSJJSKGQYHCCJ
(6)	PSD	LXS	UUSOFCFSYCUHZBSMEMOYWSBSNQ	(19)	NZC	TGK	YQBBRRKUFZCQSWCAUXIIGLLQIU
(7)	VRQ	RHJ	OHNRXYIDMUPOXWMZRZIGOQMYSD	(20)	XPS	SKA	WOWXIOIZFYTVVQDSCYVZPJOTTB
(8)	JHY	MIF	DEEGFUXRYRSBNGISWBXOYXFKKP	(21)	WQA	WUE	LVYNFGVSJZFUFRAGBWEZICVTGY
(9)	KUU	ATV	IJXULQTGTSHKOBZDETGAAQFLPQ	(22)	GYM	ZBT	QSXCNCFSMGBEHTERSJIDZKHVMF
(10)	HVP	POZ	WNIZUZOVEGOGTQZTROXURPLVQK	(23)	UEW	UNW	UYEJKXKOZRMMHPXHNNBBLGCDJU
(11)	IBB	QVC	IXQTPPJPTNFZPYMRVSAUTBHFTJ	(24)	TLI	YMD	HHRENCJMTUOYOSRXLDKZFDQNIR
(12)	AWN	FEB	BQKUHSAFAZGIVIHFCDVUHNEJQK	(25)	BAX	XPP	YHQPDLVPQGXOUKROILVXIWYPUC
(13)	CDL	HZI	DUYYSPVBTXHPWXUGJLJFADGFRW	(26)	QTE	VYG	TREUTGCDHREXSGDMVFTFDUGOHW

Example 3.1 We continue computations from [2] (cf. [2], example 5.1). Table 1 contains messages eavesdropped during the same day (i.e. received for the same daily key). All letters of the alphabet occur on each of the six headline positions. We reveal that these headlines were generated for ring settings RS = ZHL, for the order of drums I,

II, III, for settings IDS = YNC and for plug connections S = (BY)(CX)(EO)(HV)(KR)(PZ).

Let us consider permutations $A_M D_M$ and $A_H D_H$ = ZGN (cf. [2], example 5.1).

$A_M D_M$: AFJMDG**ZK**.**BX**S**E**.NT**YO**.**CHP**LIQ**VR**.U.W

ZGN: AFJMDGPR.BENT.CSOY.HKXVZLIQ.U.W

Let us write down ZGN as follows

ZGN: AFJMDG**PR**.**YC**S**O**.NT**BE**.**XVZ**LIQ**HK**.U.W

and read permutations

S_1 = (BY)(CX)(EO)(HV)(KR)(PZ), S_2 = (BY)(CX)(EO)(HV)(KR)(PZ)(UW)

Definition 3.1 Let us assume that permutations A, B and S (of an n-element set) satisfy the equation $A = SBS^{-1}$ and the permutation S consists of transpositions and 1-cycles only. Let us write down permutations A and B (in cycle notation) one under the other. Additionally, let us place cycles of both permutations so that any letter b_j will be written under the letter a_i, where $S(a_i)=b_j$ and $S(b_j)=a_i$ for $i, j = 1, 2, \ldots, n$. Then we shall say that A and B are *suitably signed* for the permutation S. We shall also say, that cycles $(a_l, a_{l+1}, \ldots, a_k)$ and $(b_l, b_{l+1}, \ldots, b_k)$ are *suitably signed* for S.

Lemma 3.1 Let us assume that permutations A, B and S (of an n-element set) satisfy the equation $A = SBS^{-1}$ and the permutation S consists of transpositions and 1-cycles only. Let us write down A and B in cycle notation. Let $A_1=(a_1, a_2, \ldots, a_k)$ and $B_1=(b_1, b_2, \ldots, b_k)$ be cycles of A and B accordingly. Let A_1, B_1 be suitably signed for S. If the i-th sign of A_1 belongs to a cycle B_i (of the permutation B) and the i-th sign of B_1 belongs to a cycle A_i (of the permutation A), then

(A) Lengths of cycles A_i and B_i are the same.

(B) We can write cycles A_i and B_i in such a way that they will be suitably signed.

Lemma 3.2 (cf. [6]) Let A and B be similar permutations. We can always distinguish in these permutations (expressed as products of disjoint cycles) cycles of the same length corresponding to each other.

Lemma 3.1 results from lemma 3.2 and from the assumption that the permutation S consists of transpositions and 1-cycles only.

Example 3.2 Let permutations A, B and S satisfy assumptions of lemma 3.1

A = (U)(W)(BXSE)(NTYO)(AFJMDGZK)(CHPLIQVR)
B = (U)(W)(YCSO)(BENT)(AFJMDGPR)(XVZLIQHK)
S = (BY)(CX)(EO)(HV)(KR)(PZ)

Let us consider cycles (**BX**SE) and (**YC**SO) which are suitably signed (for S). For signs **B** and **Y** we have cycles (NT**Y**O) and (**B**ENT), which have length 4 and can be suitably signed (the second cycle has to be signed as (NT**B**E)). For signs **X** and **C** we have cycles (**C**HPLIQVR) and (**X**VZLIQHK), which have length 8 and are suitably signed.

3.1 Schema of the plugboard algorithm

1. Enter permutations $A_M D_M$, $A_H D_H$ (parameters S1, S2) in the cycle notation.

2. For each of these permutations create a list of elements (of type `String`) which consist of cycles of the same length (separated by a single dot). Each list is ordered for the sake of cycle lengths. E.g., for permutations $A_M D_M$ and $A_H D_H$ (example 3.1) the algorithm will create 3-element lists `SL1` and `SL2` accordingly

`SL1: .U.W., .BXSE.NTYO., .AFJMDGZK.CHPLIQVR.`

`SL2: .U.W., .BENT.CSOY., .AFJMDGPR.HKXVZLIQ.`

3. For each couple of consecutive elements from lists `SL1` and `SL2` (i.e. for each length of cycles) find all possible couples of suitably signed cycles.

(a) Substitute a consecutive element of the list `SL1` (`SL2`) for variable `sl1` (`sl2`), e.g. `sl1`: `.BXSE.NTYO.`, `sl2`: `.BENT.CSOY.`

(b) For each consecutive cycle from `sl1` and for each representation of any cycle from `sl2` create (if it is possible) a couple of suitably signed cycles. That is, for a couple of cycles (e.g. `BXSE` and `CSOY`) create the following pairs [`BXSE`, `CSOY`], [`BXSE`, `SOYC`], [`BXSE`, `OYCS`], [`BXSE`, `YCSO`] and check which of them are suitably signed; e.g., for the last pair [`BXSE`, `YCSO`]:

- Substitute a cycle from `sl1` for variable `sc1` (e.g. `sc1=BXSE`) and a cycle from `sl2` for variable `sc2` (e.g. `sc2=YCSO`).
- In permutation `S1` find a cycle which contains the first letter of string `YCSO` (it will be cycle `NTYO`) and in permutation `S2` - a cycle which contains the first letter of string `BXSE` (it will be cycle `BENT`). Substitute `NTYO` for variable `sh1` and `NTBE` for variable `sh2`. If `sh1`[1]∈`sc1` and `sh2`[1]∈`sc2`, clear `sh1` and `sh2`.
- Check whether for any couple of letters [`sc1.sh1`][*i*] and [`sc2.sh2`][*i*] lengths of cycles (which contain these letters) in permutations `S1` and `S2` accordingly are identical. That is, e.g. for the couple [`BXSE.NTYO YCSO.NTBE`] compare lengths of the following couples of cycles [NT**<u>Y</u>**O, **<u>B</u>**ENT], [**<u>C</u>**HPLIQVR, HK**<u>X</u>**VZLIQ], [BX**<u>S</u>**E, C**<u>S</u>**OY], [NTY**<u>O</u>**, B**<u>E</u>**NT], [**<u>N</u>**TYO, BE**<u>N</u>**T], [N**<u>T</u>**YO, BEN**<u>T</u>**], [**<u>B</u>**XSE, CSO**<u>Y</u>**], [BXS**<u>E</u>**, CS**<u>O</u>**Y].
- Check whether cycles `sc1.sh1` and `sc2.sh2` can be suitably signed (i.e., if `Y` is signed under `B`, then `B` has to be signed under `Y`). If some letter occurs in one string only, the algorithm assumes that it is all right; e.g., couple [`BXSE.NTYO`, `YCSO.NTBE`] is correct.
- If cycle `sh1` precedes cycle `sc1` in string `S1`, clear variables `sh1` and `sh2`.

(c) Use (if it is possible) each correct pair [`sc1.sh1 sc2.sh1`] to expand strings which are in the vector `CSt` so as to obtain all possible couples of suitably signed permutations; i.e., for each object of the vector `CSt` (each object consists of two fields `s1` and `s2` of type `String`):

(i) Follow steps (ii)-(iv) if strings `s1`, `s2` do not contain cycles `sc1` and `sc2` accordingly.

(ii) Substitute string `s1` (`s2`) for variable `h1` (`h2`). Join `sh1` at the end of `h1` and `sh2` at the end of `h2` if `sc2`[1]∉`h1` or `sc1`[1]∉`h2`.

(iii) Join `sc1` (`sc2`) at the beginning of `h1` (`h2`).

(iv) Add the couple of strings [`h1 h2`] at the end of the vector `CSt`.

(d) From the vector `CSt` remove objects, which were created by joining shorter cycles (i.e., objects joined in previous iterations).

4. Move complete couples of strings which represent suitably signed permutations from the vector `CSt` to the vector `CS`.

5. For each couple of permutations from `CS` create a permutation *S*.

Executing the plugboard algorithm

(1).We enter permutations `S1` and `S2`

`S1`: `U.W.BXSE.NTYO.AFJMDGZK.CHPLIQVR`

`S2`: `U.W.BENT.CSOY.AFJMDGPR.HKXVZLIQ`

(2).The algorithm creates lists `SL1` and `SL2`

`SL1`: `.U.W.,.BXSE.NTYO.,.AFJMDGZK.CHPLIQVR.`

`SL2`: `.U.W.,.BENT.CSOY.,.AFJMDGPR.HKXVZLIQ.`

(3b). For each coupe of cycles `sc1` and `sc2` (of the same length), where `sc1`∈`S1` and `sc2`∈`S2` the algorithm checks whether these cycles are suitably signed and next it generates strings of the form [`sc1.sh1 sc2.sh2`]. For permutations `S1` and `S2` the algorithm created the couples (1)-(8) (see below).

(3cd). Each correct couple [`sc1.sh1 sc2.sh2`] is used to expand strings which are in the vector `CSt`. The algorithm joined the following strings to the vector `CSt`. The numbers in the brackets mean the numbers of consecutively joined couples [`sc1.sh1 sc2.sh2`].

```
U. U.                   (1)     (sc1=U, sh1="", sc2=U, sh2="")
U.W W.U                 (2)
W. U.                   (3)
W. W.                   (4)
BXSE.NTYO YCSO.NTBE     (5)
NTYO. NTBE.             (6)
AFJMDGZK. AFJMDGPR.     (7)
CHPLIQVR. XVZLIQHK.     (8)
//--------------------------
U. U.                    (1)    <--- Add 1-cycles
U.W W.U                  (2)
WU. WU.                 (1,4)   (We join string (4) to string (1))
BXSEU.NTYO YCSOU.NTBE   (1,5)   <--- Join 4-cycles
BXSEU.WNTYO YCSOW.UNTBE (2,5)
BXSEWU.NTYO YCSOWU.NTBE (1,4,5)
NTYOU. NTBEU.           (1,6)
NTYOU.W NTBEW.U         (2,6)
NTYOWU. NTBEWU.         (1,4,6)
AFJMDGZKBXSEU.NTYO AFJMDGPRYCSOU.NTBE        (1,5,7) <--- Join 8-cycles
AFJMDGZKBXSEU.WNTYO AFJMDGPRYCSOW.UNTBE      (2,5,7)
AFJMDGZKBXSEWU.NTYO AFJMDGPRYCSOWU.NTBE      (1,4,5,7)
AFJMDGZKNTYOU. AFJMDGPRNTBEU.                (1,6,7)
AFJMDGZKNTYOU.W AFJMDGPRNTBEW.U              (2,6,7)
AFJMDGZKNTYOWU. AFJMDGPRNTBEWU.              (1,4,6,7)
CHPLIQVRAFJMDGZKBXSEU.NTYO XVZLIQHKAFJMDGPRYCSOU.NTBE      (1,5,7,8)
```

CHPLIQVRAFJMDGZKBXSEU.WNTYO **XVZLIQHK**AFJMDGPRYCSOW.UNTBE	(2,5,7,8) (a)
CHPLIQVRAFJMDGZKBXSEWU.NTYO **XVZLIQHK**AFJMDGPRYCSOWU.NTBE	(1,4,5,7,8)(b)
CHPLIQVRAFJMDGZKNTYOU. **XVZLIQHK**AFJMDGPRNTBEU.	(1,6,7,8)
CHPLIQVRAFJMDGZKNTYOU.W **XVZLIQHK**AFJMDGPRNTBEW.U	(2,6,7,8)
CHPLIQVRAFJMDGZKNTYOWU. **XVZLIQHK**AFJMDGPRNTBEWU.	(1,4,6,7,8)

(4).The algorithm found two strings which represent suitably signed permutations.

(a)**CHP**LIQ**VR**.AFJMDG**ZK**.**BX**S**E**.**U**.**W**.NT**YO**

XVZLIQ**HK**.AFJMDG**PR**.**YC**S**O**.**W**.**U**.NT**BE**

(b)**CHP**LIQ**VR**.AFJMDG**ZK**.**BX**S**E**.W.U.NT**YO**

XVZLIQ**HK**.AFJMDG**PR**.**YC**S**O**.W.U.NT**BE**

(5).Finally, it generated the following permutations

S_1 = (BY)(CX)(EO)(HV)(KR)(PZ)(UW), S_2 = (BY)(CX)(EO)(HV)(KR)(PZ)

3.2 Implemention

The `PlugBoard()` method of the `Cycles` class determines all possible permutations S, which satisfy the equation $A_M D_M = S A_H D_H S^{-1}$. By using permutations A_M and D_M the first and the fourth letters of each message were coded on a given day. The product $A_H D_H$ (parameter `S2`) is a result of the catalogue algorithm (cf. [2], section 5).

```
void Cycles::PlugBoard(String S1, String S2){
String s1, s2, s2h, sc1, sc2, sh1, sh2, h1, h2;
std::vector<CSO*>CSt; int SCS=0, i=1, j, v;
TStringList *SL1=strList(S1); TStringList *SL2=strList(S2);
while(i<SL1->Count){
  s1=SL1->operator [](i); s2=SL2->operator [](i);
  while(s1.Length()>1){
    sc1=cycle(s1,s1[2]); s1=delCycle(s1,sc1); s2h=s2;
    while(s2h.Length()>1){
      sc2=cycle(s2h, s2h[2]); j=1;
      while(j<=sc2.Length()){
        if(compLength(sc1,sc2,SL1,SL2)){
          if((sc1.Pos(sc2[1])&&sc2.Pos(sc1[1])))sh1=sh2="";
          else{
            sh1=cycle(S1,sc2[1]);
            sh2=move(sh1,sc2[1],cycle(S2,sc1[1]),sc1[1]);}
          if(compLength(sh1,sh2,SL1,SL2)&&suitSign(sc1+sh1,sc2+sh2)){
            if(S1.Pos(sc1)>S1.Pos(sh1))sh1=sh2="";
            if(sc1[1]==S1[2])CSt.push_back(new CSO(sc1+sh1,sc2+sh2));
            else{v=0;
              while(v<CSt.size()){
                h1=CSt[v]->s1; h2=CSt[v]->s2;
                if(!(h1.Pos(sc1[1])||h2.Pos(sc2[1]))){
                  if(!(h1.Pos(sc2[1])&&h2.Pos(sc1[1]))){
                    h1+=sh1; h2+=sh2;}
                  h1=sc1+h1; h2=sc2+h2;
                  CSt.push_back(new CSO(h1,h2));}
                v++;}}}}
        sc2=sc2.SubString(2,sc2.Length()-1)+sc2[1]; j++;}
      s2h=delCycle(s2h,sc2);}}
  for(int l=0; l<SCS; l++)CSt.erase(&CSt[0]);}
  v=0;
  while(v<CSt.size()){
    if(CSt[v]->s1.Length()==26){
      CS.push_back(new CSO(CSt[v]->s1,CSt[v]->s2));
      CSt.erase(&CSt[v]); v--;}
    v++;}
```

```
(38)    SCS=CSt.size(); i++;}
(39) createS();}
```

The `strList()` method of the `Cycles` class creates (for a permutation provided by the parameter) the list, elements of which consist of cycles of the same length (separated by a single dot) (e.g. the lists `SL1`, `SL2`). Each list is ordered for the sake of cycle lengths. The `cycle()` method of the `Cycles` class returns a cycle (which contains an indicated sign) included in a given permutation (e.g. `cycle(S1,'Y')` returns `NTYO`). The `move()` method of the `Cycles` class returns another representation of a cycle (provided by the third parameter). That is, it moves signs of a given representation so that the letter pointed out by the fourth parameter was placed on the same position as the letter (indicated by the second parameter) in the cycle pointed out by the first parameter. For example, if we call `move("BONT",'B',"NTYE",'Y')`, we shall obtain `YENT`. The `compLength()` method of the `Cycles` class compares cycle lengths (cf. subsection 3.1, 3.(b), example [`BXSE.NTYO YCSO.NTBE`]). The `suitSign()` method of the `Cycles` class checks whether given (by parameters) cycles can be suitably signed. The `delCycle()` method of the `Cycles` class removes the indicated cycle from a given string. The `createS()` method of the `Cycles` class generates the list of permutations *S* on the basis of suitably signed permutations included in the vector `CS`. The vectors `CS` and `CSt` (an auxiliary vector) contain objects which consist of two fields of type `String`.

4 The ring setting algorithm

The first six letters (a headline) of each message were ciphered on a given day (until September 15, 1938) for the same *ring settings* (in short RS) and *initial drum settings* (in short IDS). That is, these 6 letters were coded consecutively for the same permutations *A*, *B*, *C*, *D*, *E* and *F* (determined for *rotor settings* (in short Rt) resulting from the formula Rt[i] = (Ds[i]-Rs[i])%26 (i = l, m, r)). We can obtain the same rotor settings for different couples of *relative ring settings* (in short RRS) and *relative drum settings* (in short RDS). The reader can observe this fact in Table 2.

The algorithm presented below generates settings RS and IDS for which in fact the messages were ciphered. RS are indispensable to read each message. The cryptologists reconstructed settings RS by means of the `ANX` method (cf. [6]). The proposed ring setting algorithm is the author's idea. We are still analyzing the set of messages (Table 1) eavesdropped during the same day.

Example 4.1 Given messages (Table 1) were coded for settings RS = `ZHL` and IDS = `YNC`. We executed the catalogue algorithm for different settings RRS. Table 2 contains settings RRS, corresponding to them settings RDS (received in the catalogue algorithm) and differences Rt[i] = (RDS[i]-RRS[i])%26 (i = l, m, r). Due to the regularities which result from these differences, we can obtain RS and IDS.

Tab. 2. Ustawienia RRS, RDS i Rt

Tab. 2. Settings RRS, RDS and Rt

XUA, -	EPH, DVY, [25,6,17]	RKO, QQF, [25,6,17]	QMV, PSM, [25,6,17]
TFB, -	GBI, FHZ, [25,6,17]	HRP, GXG, [25,6,17]	UGW, TMN, [25,6,17]
RLC, -	ADJ, ZJA, [25,6,17]	CZQ, BFH, [25,6,17]	PNX, OTO, [25,6,17]
MQD, -	JYK, IEB, [25,6,17]	IJR, HPI, [25,6,17]	DWY, CCP, [25,6,17]
VHE, UMV, [25,5,17]	FCL, EIC, [25,6,17]	YVS, XBJ, [25,6,17]	BCZ, -
SEF, RKW, [25,6,17]	ZOM, YUD, [25,6,17]	OST, NYK, [25,6,17]	
WAG, VGX, [25,6,17]	LTN, KZE, [25,6,17]	NIU, MOL, [25,6,17]	

4.1 Schema of the ring settings algorithm

Input: Data from a given day, i.e. any message, a couple of RRS and RDS (received in the catalogue algorithm), the proper order of drums and plug connections *S*.

Output: The algorithm returns ring settings and initial drum settings.

1. Set your machine in the following way:

- Set the drums to the proper order and the plugboard to the permutation *S*.
- Set rings to relative ring settings (parameter `rrs`) and drums to relative drum settings (parameter `rds`) which you obtained in the catalogue algorithm.
- Enter a headline (parameter `head`) and a content (parameter `text`) of a given message.

2. Determine rotor settings for given settings `rrs` and `rds`. For these rotor settings the first six letters of all headlines are coded / decoded on a given day.
3. Determine message settings by decoding the headline (of the message) for settings `rrs` and `rds`.
4. For each (of 26^3 possible) ring settings `rrs1` (see: the iteration loop, lines 7-13)

- Determine such drum settings `rds1` so that rotor settings will be the same as the ones determined in point 2, i.e. Rt[*i*]=`rds`[*i*]-`rrs`[*i*]=`rds1`[*i*]-`rrs1`[*i*] for *i*=*l*, *r*, *m*.
- Set rings and drums to `rrs1` and `rds1` accordingly.
- Encrypt message settings (of the given message) for the current couple of settings `rrs1` and `rds1`. If you obtained the headline `head`, write out relative ring settings `rrs1`, relative drum settings `rds1` and a decoded (for settings `rrs1` and message settings) text of the given message.

5. Look through decoded contents of the message. Ring settings, for which a decoded content is intelligible, make up actual ring settings for which all messages of a given day were coded.

Example 3.1 Continuation.

Input: RRS=`AAW`, RDS=`ZGN`,

message: `LIOIWN BVSVIWKOUYZKHEHCCNBKWHTMMI`

```
ZGY YMP SFEGNUTVALKUQVQHHDGZIQUCFL   <--- A fragment of a result
ZHF YNW QROMWFYAVLMCSSQURFZDPQVZFQ    of the ring settings algorithm
ZHG YNX JTGQHILWLLHNAMCIUFFUOZBWZS
ZHH YNY PEARUKZBDKJMPLSHIKVZHQSOWV
ZHI YNZ AHMTKJWSJKRXZXAXHSMOCWEYOO
```

ZHJ	YNA	ODWECZHSQPJYOAPKXLKWTJSXYJ
ZHK	YNB	VYRHPUISLPVYFMZQKOABKQDYXB
ZHL	**YNC**	**ABCDEFGHIJKLMNOPQRSTUVWXYZ**
ZHM	YND	HQTYESHNMRQZCGFGPKORIAXXXX
ZHN	YNE	GMTBUZCXWDEWMQMDGBWVUFEBXG

We can state that actual RS=`ZHL`. In order to read a text of any message we have to set rings to `ZHL` and drums to message settings (of this message) and type a text.

4.2 Implementation

The `findRS()` method of the `Cycles` class generates actual ring settings, initial drum settings and decodes a content of a given (by parameters `head` and `text`) message. The table `Rt`[] keeps rotor settings for given (by parameters `rrs` and `rds`) relative ring settings and relative drum settings. The `codeStr()` method of the `Enigma` class decodes a specific (by the parameter `s`) text for current settings of Enigma. During the coding process the double step on the middle drum is taken into account. The `moveRing()` method of the `Cycles` class shifts ring settings forward by *k* positions. The fields `Rs` and `Ds` of an object of the `Enigma` class represent current ring settings and drum settings accordingly. Current rotor settings are stored in the table `RT`[]. By `tpR` and `tpM` we signify the turnover positions for the right and the middle drums accordingly. The `codeLetter()` method ciphers a letter described by the first parameter for the rotor settings determined by the next three parameters.

```
void Cycles::findRS(String rrs, String rds, String head, String text){
int* Rt=new int[4];
for(int i=1; i<=3; i++) Rt[i]=(26+CTI(rds[i])-CTI(rrs[i]))%26;
CE->setEnigma(rrs,rds);
String key=CE->codeStr(head); key=key.SubString(1,3);
String rds1="AAA", rrs1="AAA";
for(int i=1; i<17576; i++){               // 26^3 possible ring settings
  for(int k=1; k<=3; k++) rds1[k]=ITC((Rt[k]+CTI(rrs1[k]))%26);
  CE->setEnigma(rrs1,rds1);
  if(CE->codeStr(key+key)==head){
    CE->setEnigma(rrs1,key);
    out >> rrs1+" "+rds1+" "+CE->codeStr(text);}
  rrs1=CE->moveRing(rrs1,1);}}
//------
String Enigma::codeStr(String s){         // with the double step
String PI=Rs, BE=Ds, BEH="", s1="";
int* RT=new int[4]; int j=s.Length(); s=s.UpperCase();
for(int i=1; i<=j; i++){
  BEH=BE; BE[3]=ITC((CTI(BE[3])+1)%26);
  if(BEH[3]==tpR) BE[2]=ITC((CTI(BE[2])+1)%26);
  if(BEH[2]==tpM){
    BE[1]=ITC((CTI(BE[1])+1)%26);
    BE[2]=ITC((CTI(BE[2])+1)%26);}
  for(int i=1; i<=3; i++) RT[i]=(26+CTI(BE[i])-CTI(PI[i]))%26;
  s1+=codeLetter(s[i],RT[1],RT[2],RT[3]);}
return s1;}
```

5 Computational complexity

The total running time of the `findRS()` method (by using a computer with an AMD Turion 64 X2 processor clocked at 1.9GHz) is about 6 seconds. To guess plug connections we call the `PlugBoard()` method for each of the listed (in the `catalogue()` algorithm) settings RDS. It produces a result immediately (in a fraction of a second). The cryptologists needed 1-2 hours to find ring settings (cf. [6]).

6 The implications of the work and conclusions

We can solve the Enigma cipher (by analyzing and completing historical information) because trained Polish and (later) British cryptologists did it earlier. The Enigma cipher is not trivial and its breaking on the basis of eavesdropped messages is practically impossible even nowadays. The three cryptologists used the help of spies, mistakes of operators and numerous favorable coincidences. The reader can find other decryption algorithms of the Enigma cipher in [1] (Zygalski's sheets method) and [3] (the cryptologic bomb method and the plugboard algorithm). All these methods are interesting exercises and encourage the study of current problems of cryptology.

References

1. Borowska A.: The Cryptanalysis of the Enigma Cipher, *Advances in Computer Science Research*, 10, 2013, pp. 19-38
2. Borowska A.: The Cryptanalysis of the Enigma Cipher. The Catalogue Method. Part I, (This paper will appear in *Symulacja w Badaniach i Rozwoju*)
3. Borowska A., Rzeszutko E.: The Cryptanalysis of the Enigma Cipher. The Plugboard and the Cryptologic Bomb, *Computer Science*, AGH University of Science and Technology Press, 15(4), Krakow, 2014, pp. 365-388
4. Christensen C.: Polish Mathematicians Finding Patterns in Enigma Messages, *Mathematics Magazine*, 2007, pp. 247-273
5. Garliński J.: *Enigma. Mystery of the Second World War*, University of Maria Curie-Sklodowska Publishing House, Lublin, 1989
6. Gaj K.: *The Enigma Cipher. The Method of Breaking*, Communication and Connection Publishing House, Warsaw, 1989
7. Grajek M.: *Enigma. Closer to the Truth*, REBIS Publishing House, Poznan, 2007
8. Gralewski L.: *Breaking of Enigma. History of Marian Rejewski*, Adam Marszalek Publishing House, Torun, 2005
9. Kozaczuk W.: *How the German Machine Cipher Was Broken and How It Was Read by the Allies in World War Two*, University Publications of America, 1984
10. Rejewski M.: *How did Polish Mathematicians Decipher the Enigma*, Polish Mathematics Association Yearbooks. Series 2nd: Mathematical News, (23), 1980

Summary

We study the problem of decoding secret messages encrypted by the German Army with the M3 Enigma machine. We focus on the algorithmization and programming of this problem. In part I of this paper we proposed a reconstruction and completion of the catalogue method. Here we complete this method with two author's algorithms, i.e. the plugboard algorithm and the ring settings algorithm. On the basis of these three methods we can obtain the complete daily key and any message settings, as well as read each message eavesdropped before September 15, 1938. We enclose an implementation of presented algorithms in Cpp language.

Keywords: Enigma M3, Rejewski, characteristic of a given day

Kryptoanaliza Szyfru Enigmy. Metoda Katalogu. Część II

Streszczenie

Tematem pracy jest kryptoanaliza szyfru niemieckiej Enigmy M3, używanej do kodowania tajnych depesz przez siły zbrojne oraz inne służby państwowe Niemiec podczas II wojny światowej. W części I zaproponowaliśmy algorytm będący rekonstrukcją metody katalogu. Tu uzupełniamy metodę katalogu o dwa brakujące algorytmy służące do wyznaczania odpowiednio połączeń łącznicy wtyczkowej i ustawień pierścieni. Wykonanie trzech wspomnianych algorytmów pozwala na odtworzenie pełnego dziennego klucza oraz klucza dowolnej depeszy. Dzięki temu możemy przeczytać dowolną depeszę przechwyconą przed 15 września 1938 r. Dodatkowo załączamy implementację przedstawionych algorytmów w języku Cpp.

Słowa kluczowe: Enigma M3, Rejewski, charakterystyka dnia

The research presented in this paper was founded by the BST S/WI/1/2014.