How was life before the computer

- Memory was something that you lost with age
- An application was for employment
- A program was a TV show
- A cursor used profanity
- The web was the home of the spider
- A virus was something like the flu
- A hard drive was a long trip on the road
- And if you had a 3 ½" floppy you just hoped that nobody found out

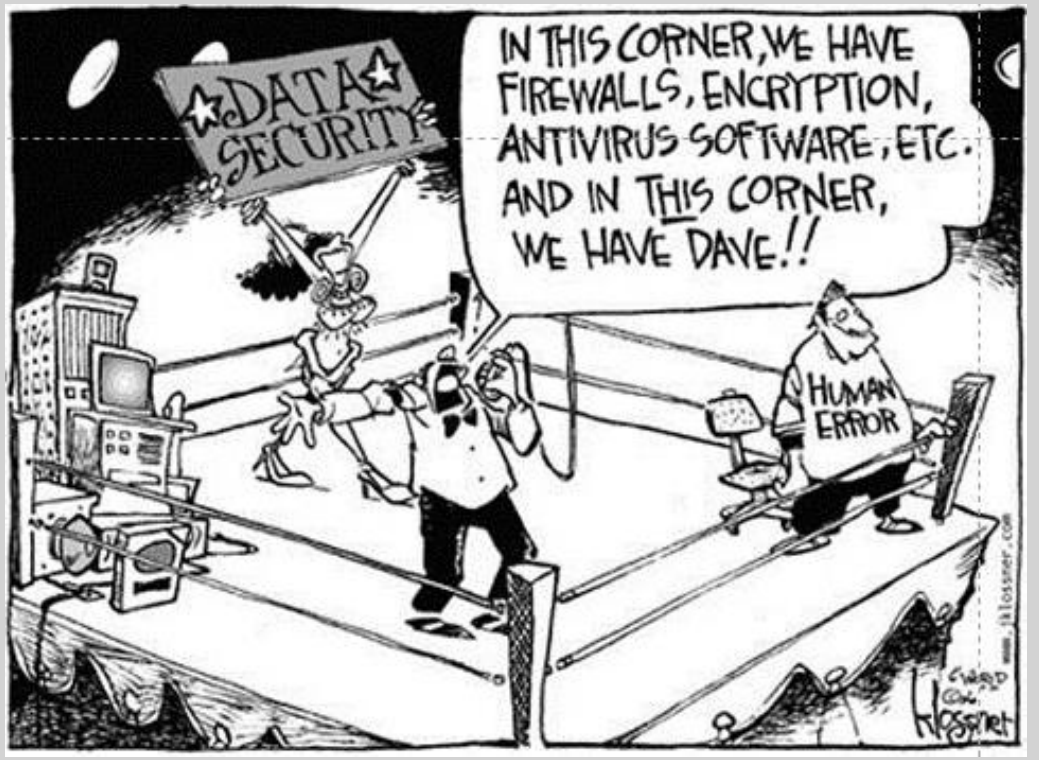
The enigma, a riddle from the past

ELECTRO-MECHANICAL ENCRYPTION

ALMOST AS GOOD AS MODERN IT TECHNOLOGY?!

AGENDA

- Encryption
- Poly alphabetic encryption: Vigenère
- History ENIGMA
- ENIGMA Design, Rotor in Detail , Schematic view
- Configuring Enigma, Daily operation and Demo
- The Simulator: design goals
- Qt, Qt Demo, how to boost Qt compilation time
- The Enigma Library and GUI
- Qt GUI design and Take-aways
- The amount of Enigma keys
- Weaknesses and code breaking



Encryption

The primary goal of encryption is to protect sensitive information
(to achieve or maintain a competitive advantage)

Encryption is almost as old as literacy

Many methods developed, f.e.

Caesar encryption:
mono alphabetic,
very weak method

Vigenère:
poly alphabetic,
rather strong for short messages

……..

POLY-Alphabetic encryption: Vigenère

Messages are encrypted using multiple alphabets. The used alphabet depends on the position of the characters in the message. The encryption alphabets are derived from a key sentence only known to sender and receiver.

	0	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
0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

History will be kind to me for I intend to write it.
Churchill(?)

key	S	T	A	R	R	Y	S	T	A	R	R	Y	N	I	G	H	T	S	T	A	R	R	Y	S	T	A	R	R	Y	N	I	G	H	T	S	T	A	R	R	Y
text	H	I	S	T	O	R	Y	W	I	L	L	B	E	K	I	N	D	T	O	M	E	F	O	R	I	I	N	T	E	N	D	T	O	W	R	I	T	E	I	T
encrypted	Z	B	S	K	F	P	Q	P	I	C	C	Z	R	S	O	U	W	L	H	M	V	W	M	J	B	I	E	K	C	A	L	Z	V	P	J	B	T	V	Z	R

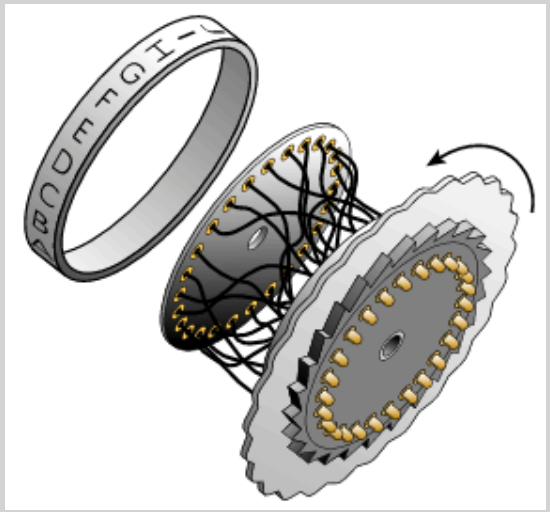
History ENIGMA

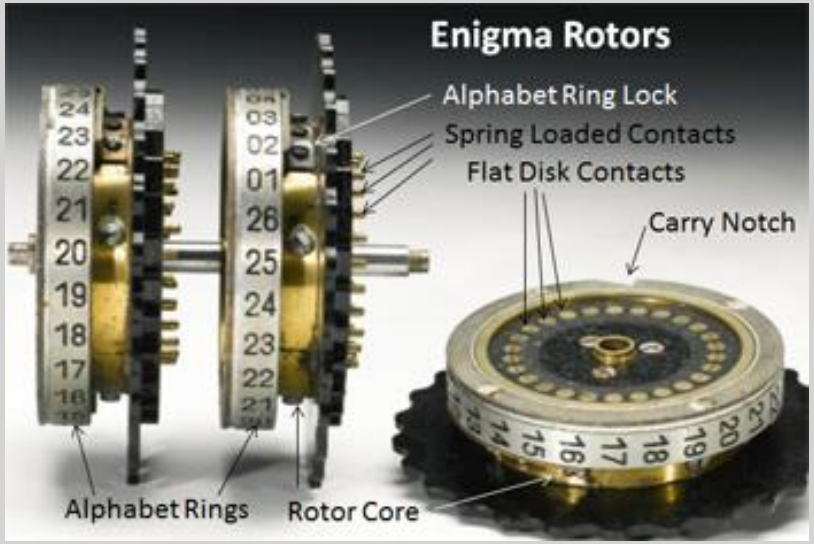
- Developed at the end of WWI
- Based on poly-alphabetic substitution driven by a mechanical machine
- Almost simultaneously invented in several countries:
  - The very first: Netherlands (1915) by two Naval officers
  - Germany (1918), Sweden and USA
  - German tool was called Enigma, initially aimed at the commercial market
  - The method how the encryption machine was designed is therefore not a secret!
  - Enigma uses far more alphabets than Vigenère, the char sequence looks “randomized”
- In 1927 the German army (Wehrmacht) ordered the Enigma with some modifications
  - Usable in the field
  - Encryption/decryption with same machine and configuration
  - Much cheaper than the commercial version
- Kriegsmarine (Navy) and Luftwaffe (Airforce) adopted the Enigma after 1932

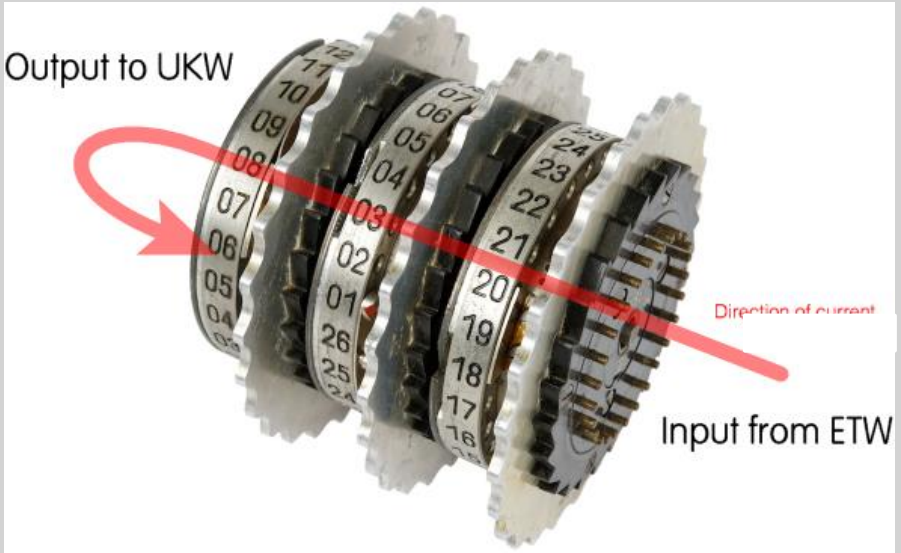
ENIGMA Design

- The alphabet contains only 26 characters
- 3 rotors out of a set of up to 8 rotors (Walzen)
- Each rotor generates a permutation
- The internal wiring of a rotor can be turned relative to the exterior frame
- Every time a key is pressed, the R or the M + R or the L + M + R rotor(s) turn(s) one step
- A non rotating part called Reflector or UKW, also generating a permutation, reflects the incoming signal back into the rotors. This ensures encryption-decryption with the same configuration
- A plugboard or SB, swaps pairs of keyed characters
- A lamp lights at the encrypted output character

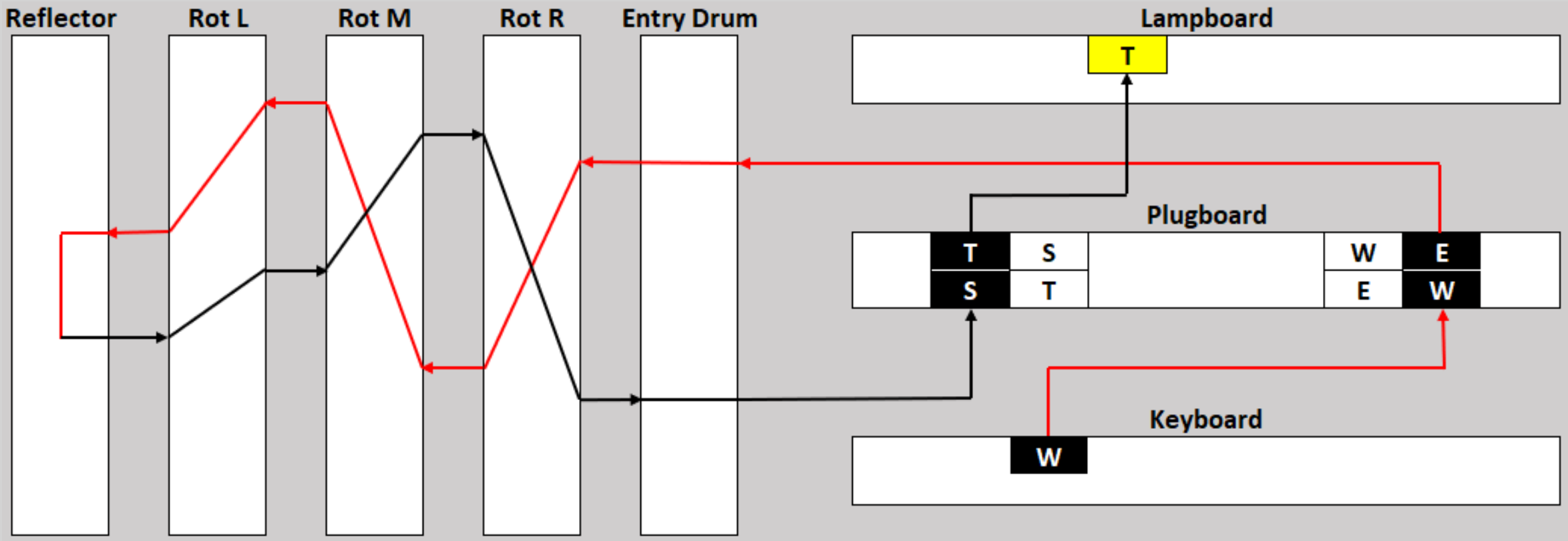
Rotor In detail



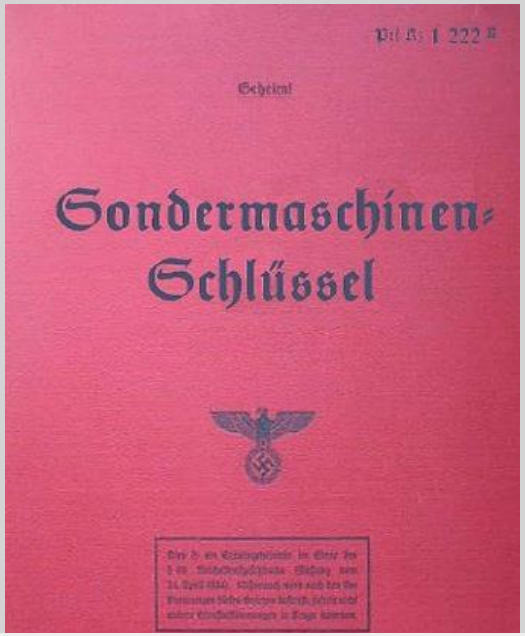


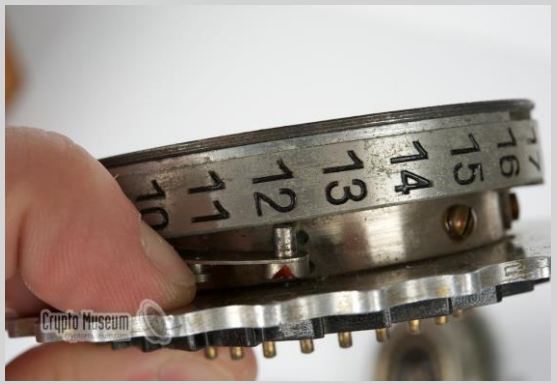


Schematic View



CONFIGURING ENIGMA





Geheim! Sonder - Maschinenschlüssel BGT

Datum	Walzenlage	Ringstellung	Steckerverbindungen	Grundstellung
31.	IV II I	F T R	HR AT IW SK UY DF GV LJ BO MX	vyj
30.	III V II	Y V P	OR KI JV OT ZW MU BF YC DS GP	cqr
29.	V IV I	O H R	UX JC PB BM TA ED ST DS LU FI	vnf

Daily operations

- Every day at midnight CET the setup of the Enigma was altered according to the *daily key book (Tagesschlüsselbuch)*
- Using that identical setup by all army units would ease the task of code breaking due to the volume of encrypted messages having the same key
- Therefore each message was encrypted with an operator choosen 3 char key
  - First Enigma is set to its configuration of the day
  - The operator chooses his 3- character key and encodes this twice using the configuration of the day
  - Then he turns the rotors to his chosen key setting and encrypts the message with its own key
  - The message sent consists of twice the encrypted new key followed by the message encrypted by that new key
  - The receiver decrypts the private key first, configures its Enigma with the new/private key and decodes the remaining part of the message
- Demo….

Demo

- According to the code book:
  - Walzenlage III, VI, VIII
  - UKW B
  - Steckerverbindungen AN EZ HK IJ LR MQ OT PV SW UX
  - Ringstellung AHM
  - Kenngruppe GSI
- The (original) intercepted message, received at 26th of December 1943:

```
YBK MXJ YKAE NZAP MSCH ZBFO CUVM RMDP YCOF HADZ IZME FXTH FLOL PZLF GGBO
TGOX GRET DWTJ IQHL MXVJ WKZU ASTR
```

The tragic content of this historic message

German: Steuere Tanafjord an. Standort Quadrat AC4992, fahrt 20sm. Scharnhorst.

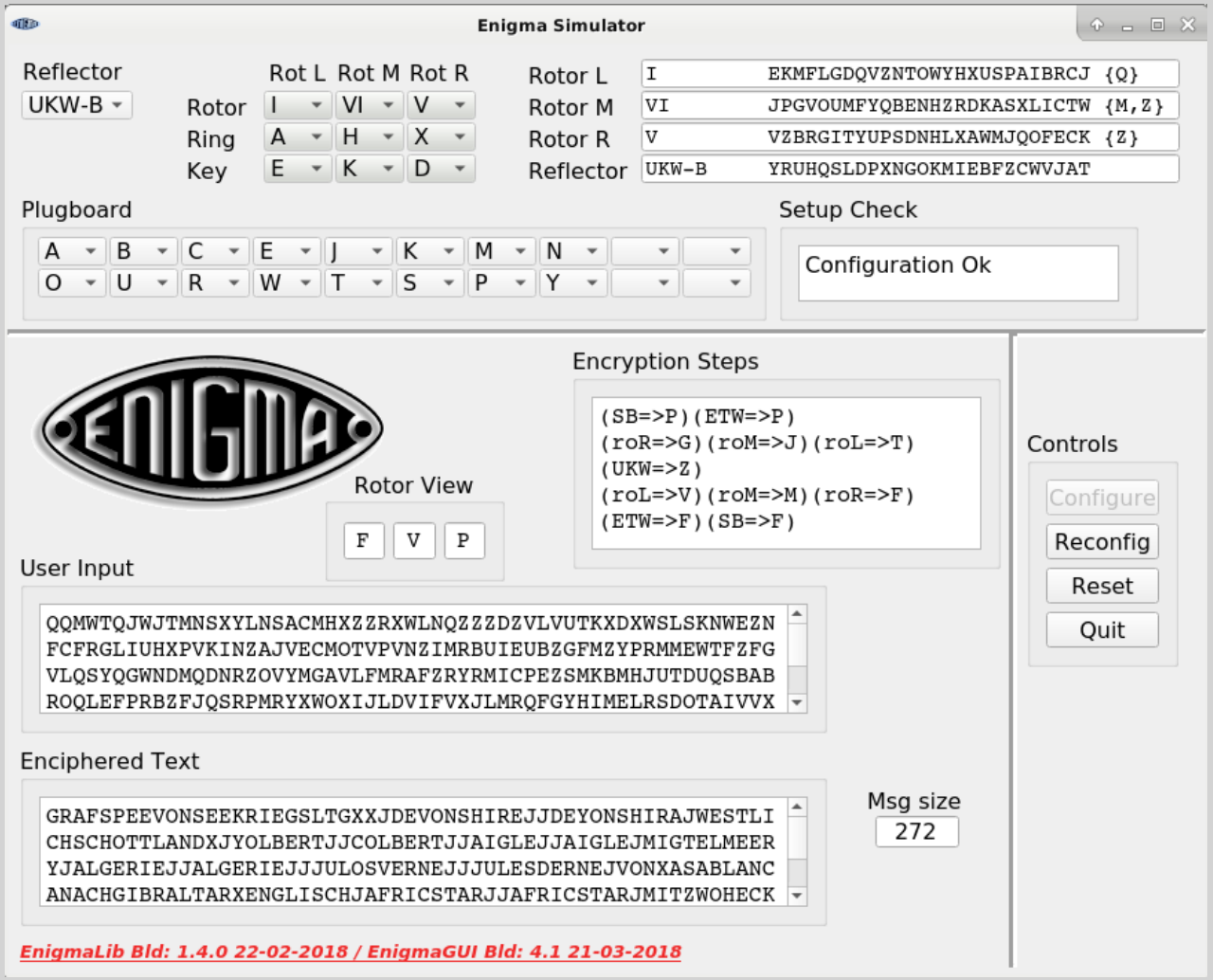
English: Heading for Tanafjord. Position Square AC4992, speed 20 knots. Scharnhorst.

Crew: 1968 man
Commissioned: 7-1-1939

Sunk by the British Navy: 26-12-1943 en route to Tanafjord, 36 survivors, shortly after sending the above message

Breaking the Enigma coding was a major effort and it probably shortened the war by 1-2 years and prevented a lot of allied casualties and maybe even German casualties.

The Enigma SIMULATOR: Design goals

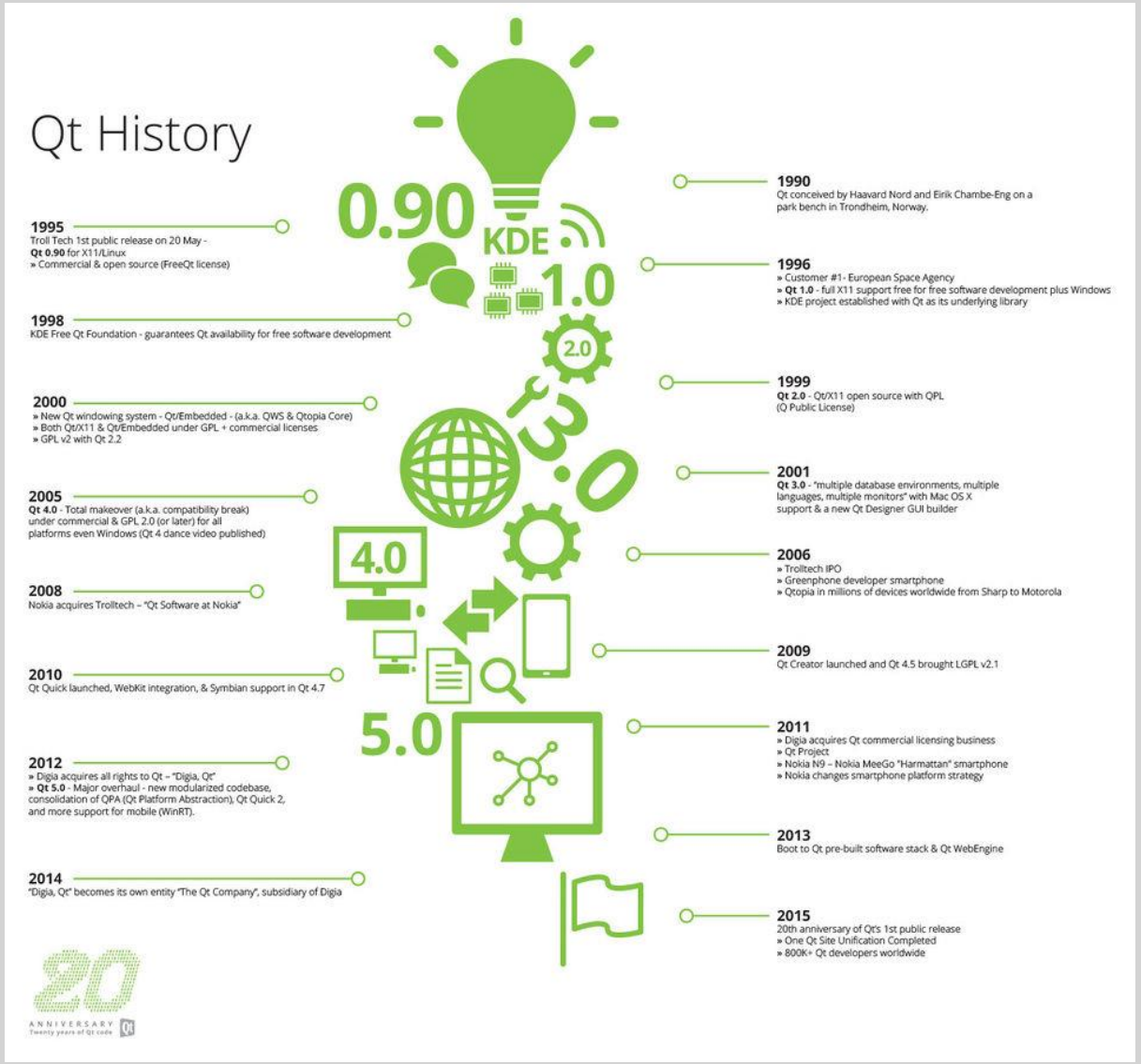


- Compliant with Enigma
  - Decrypt real war messages
- An intuitive graphical user interface
  - GUI developed with Qt
  - Takes care of in-output & setup validation
  - Manageable GUI code
  - Easy to use, e.g. cut & paste
- A “stand alone” Enigma Lib (QT free)
  - Library offers rotors, UKWs and handles the encryption
  - PIMPL for Enigma Lib to avoid ABI breaches
  - Manage Enigma Lib with ICM* tools

Qt

- Qt's conception: Norway, 1990
- Open source, Free Qt licence
- Company TrollTech
- First release 1995 for Linux/X11
- First customer ESA in 1996
- Nokia acquires Qt in 2008

- Qt is now a cross-platform IDE, supporting Linux, Windows, MAC, many devices, …
- C, C#, C++, D, Java, Python, …
- Some customers:
  - Spotify, Adobe, Volvo, Walt Disney, Lucas Film, US Army Research Lab, Los Alamos Research Lab, BitCoin Core



Qt Demo

Qt core:

- Very Nice IDE
- Very Good graphical development tool
- Very Good documentation system

Demo

Boost Qt

Performance is a little struggle:

- Build time increases linearly with the amount of files to be compiled and linked

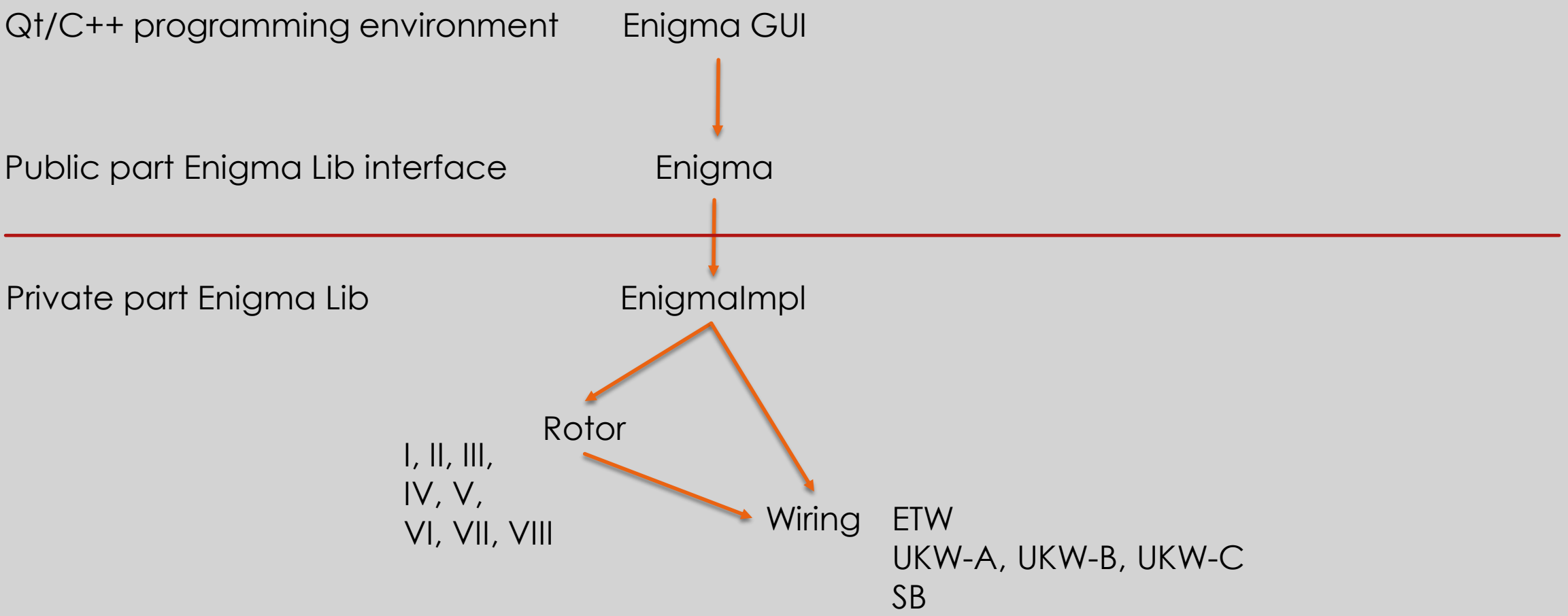
Main reasons for this:

- the amount of included (Qt) headers that gets parsed for every file (1000's of lines)
- no parallel processing/threads, only 1 CPU core utilized

Solution & result:

- Use pre-compiled headers which is easy in Qt IDE
- Set make opt –j<n> with n set to about 2 + * CPU-cores (experiment for best value)
- Result:
- Old laptop i3, 3 Gb, 2.2 Ghz, SSD: 1m:50s, -j8 0m:50s, pch 0m:20s
- Less old desktop, i7, 16 Gb, 3.4 Ghz, Vbox 4 CPU: 0m:56s, -j16 0m:16s, pch 0m:07s

Enigma Lib & GUI



Enigma Library high level interface

Enigma in the box

- Reflectors and rotors to choose from
- Enigma alphabet & size
- Count of rotors to be inserted

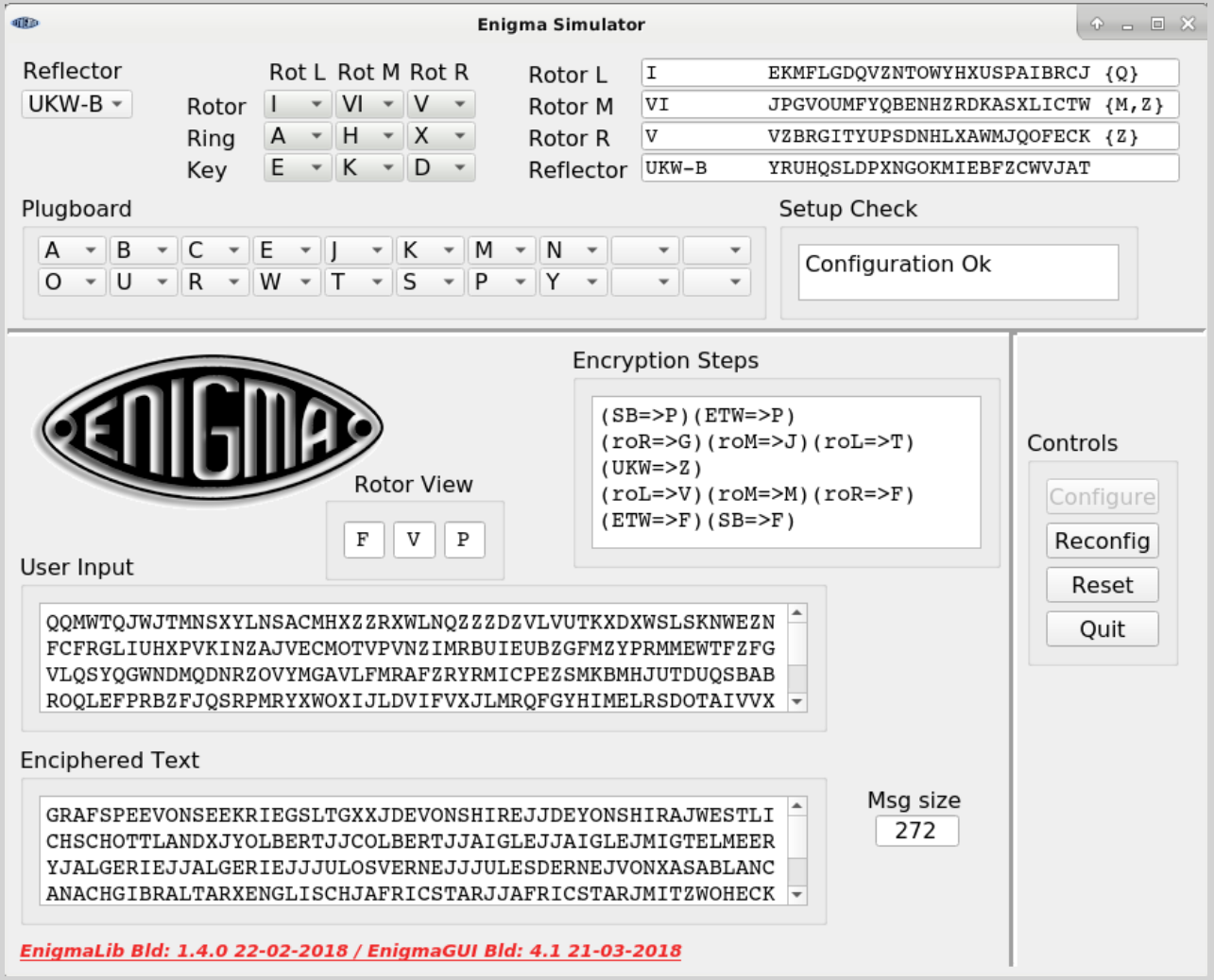
Setup Enigma

- Set key and ring values per rotor
- “Insert” reflector and rotors
- Check and set plugboard
- Reset Enigma to starting position

Encryption

- Encrypt a char
- View of rot position after an encryption step

Debug information



Public interface EnigmaLib

```
class Enigma {

public:
    Enigma();

    //Enigma's content: alphabet, reflectors and rotors
    static std::string alphabet();           //returns the Enigma alphabet that can be used
    static size_t rotorCnt();                //number of rotors to be installed
    std::vector<std::string>*rflLst() const; //list of available reflectors
    std::vector<std::string>*rotLst() const; //list of available rotors

    //setup Enigma and reset Enigma
    void setRfl(std::string const &rflId);                  //install reflector with rflId
    void setRot(size_t rPos, std::string const &rotId, size_t key, size_t ring);
                                                            //install rotor with rotId at position rPos
    bool chkPlgBd(std::vector<size_t> const &vecSB) const; //check if plugboard setting is ok
    void cfgPlgBd(std::vector<size_t> const &vecSB);       //configure plugboard
    void reset();                                          //reset = initialize (0 chars typed)

    //encryption and view on (turned) rotors
    char encrypt(char ch);                   //encrypt one char
    size_t rotView(size_t rPos) const;       //returns visible key on rotor

    //debug . . . .
};
```

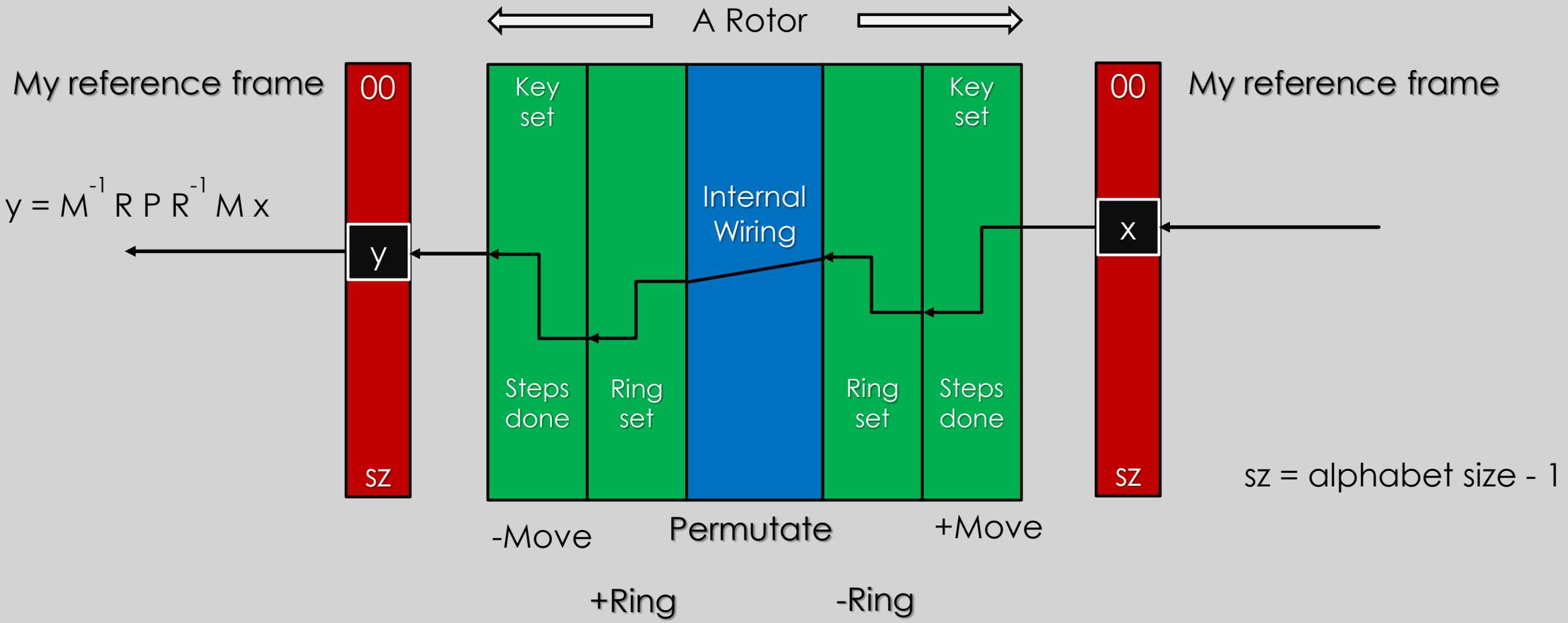
char EnigmaImpl::encrypt(char ch)

```
char EnigmaImpl::encrypt(char ch)
{
    turnRots();    //prior to encryption turn the rotor(s)
    ch = ch - 'A'; //bring input in range 0 .. wireSze() - 1

    ch = fwdWrEncypher(d_SB, ch);
    ch = fwdWrEncypher(d_ETW, ch);

    ch = fwdRtEncypher(roR, ch);
    ch = fwdRtEncypher(roM, ch);
    ch = fwdRtEncypher(roL, ch);

    ch = fwdWrEncypher(*d_UKW, ch);

    ch = bckRtEncypher(roL, ch);
    ch = bckRtEncypher(roM, ch);
    ch = bckRtEncypher(roR, ch);

    ch = bckWrEncypher(d_ETW, ch);
    ch = bckWrEncypher(d_SB, ch);

    return ch + 'A';
}
```

Rotor Movements

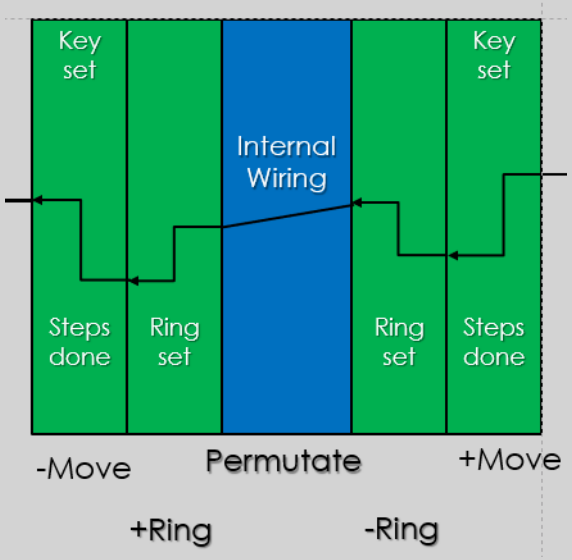
- Every time a key is pressed 1, 2 or 3 of the rotors will turn prior to encryption
- Each rotor has one or more turnover points (mechanical: pawl and notches)
- The rightmost rotor (R) always turns
- Rotors R or M will turn its left-hand neighbour iff that rotor is at one of its turnover points
- If rotor R has pushed rotor M to its turnover point then, at the next key press, rotor M will turn rotor L (both turn). So the middle rotor will turn twice in two consecutive steps, this is called double stepping. It is caused by the mechanical design of Enigma.

From input to output through a rotor



size_t EnigmaImpl::fwdRtEncypher()

```
//the turning is considered in a fixed frame 0 .. alphabetSze()
//so input has to be mapped on the turned rotor position (due to
//steps and key) and the turned internal wiring (ring)
//output has to be mapped back on the fixed frame

size_t EnigmaImpl::fwdRtEncypher(size_t rPos, size_t idx) const
{
   Rotor *rot = d_rots.Rot.at(rPos);
   size_t ring = d_rots.Ring.at(rPos);

   idx = rot->rotWr().permute((idx  + rotView(rPos) + alphabetSze() - ring) % alphabetSze());

   return (idx + ring + alphabetSze() - rotView(rPos)) % alphabetSze();
}
```

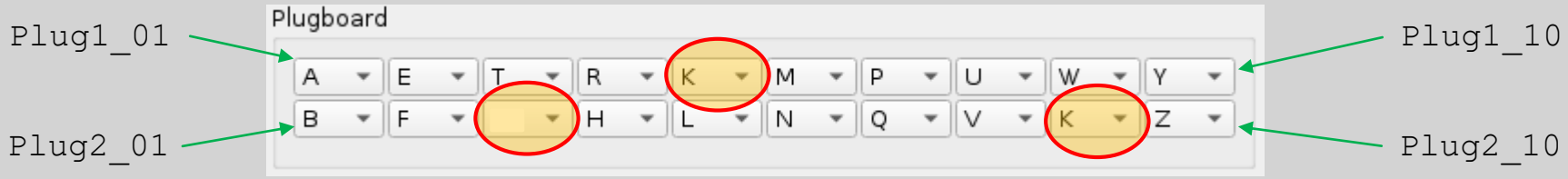


Enigma GUI

```
class Gui : public QMainWindow {        //derived class

    Q_OBJECT                            //calls Qt pre-processor

public:
    explicit Gui(QWidget *parent = 0); //copy and implicit initialization not allowed
    ~Gui();

private slots:                          //not C++ slang
    void on_Configure_clicked();        //setup selected rotors/rings/keys/reflector/plugboard
    void on_Reconfig_clicked();         //do setup again
    void on_Reset_clicked();            //clear input, reset rotors to starting cfg
    void on_UserInput_textChanged();    //analyze input, repair input or encrypt if input ok

private:
    bool plugboardOk() const;           //check plugboard for ambiguous plugs like (AB) en (BK)
    bool loadPlugBoard();               //copies plugboard settings from Gui to a vector
    . . . . . .
```

GUI DESIGN / Qt (1)

Naming of widgets on GUI doesn't allow for arrays or lists or any ordering that you could walk through using for() or while(). Consider f.e. checking the plugboard settings:




```
// check for plugboard errors
bool MachineWindow::plugboardOk() const
{
    . . . . .
    //inspect plugboard settings, both plugs are set or both plugs are blank
    if(ui->Plug1_01->currentText().length() != ui->Plug2_01->currentText().length())
       return false;
    if(ui->Plug1_02->currentText().length() != ui->Plug2_02->currentText().length())
       return false;
    . . . .
    if(ui->Plug1_10->currentText().length() != ui->Plug2_10->currentText().length())
       return false;
```

GUI DESIGN / Qt (2)

- You can put the widgets in a list and then process it

  (I would prefer a vector but that is not implemented)

- findChildren<QType*>(QRegEx) puts all the widgets of

  QType in a list if their names match a regular expression

- So: choose the names of the widgets carefully

- Put the group in a list and sort it if needed

- Then process it

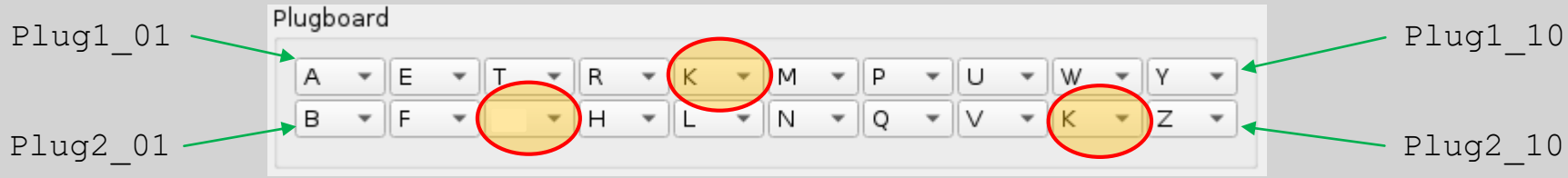
Iterator	Widget
List.begin()	Plug1_01
List.end()	Plug1_10

```
QRegularExpression("Plug1_\\d{2}$"));
```

GUI DESIGN / Qt (3)

```
//return a sorted list of QComboBox* matching a reg expr
QList<QComboBox*> Gui::aGuiLst(QString const regex)
{
    QList<QComboBox*> lst;
    lst = findChildren<QComboBox*>(QRegularExpression(regex));

    auto order = [](QComboBox* const p1, QComboBox* const p2)
                 {return p1->objectName() < p2->objectName();};

    std::sort(lst.begin(), lst.end(), order);
    return lst;
}
```

```
void Gui::bldGuiLsts()
{
   d_SB1 = aGuiLst("Plug1_\\d{2}$");
   d_SB2 = aGuiLst("Plug2_\\d{2}$");
   . . . .
}
```




```
//pairwise checking "Plug1_01" vs "Plug2_01", "Plug1_02" vs "Plug2_02"
for (auto idx = 0; idx != d_SB1.size(); ++idx)
{
      if (d_SB1.at(idx)->currentText().length() !=
          d_SB2.at(idx)->currentText().length())
          return false; //one char choosen one empty
};
```

GUI DESIGN TAKE-AWAYS

- The design of a Gui is far different from the programming task
  - As Industrial Design differs from Mechanical Engineering
- A graphical design interface like Qt has is a must; you need to see what you get
- Two aspects are important for the design of Gui:
  - The interface from a user perspective: intuitive && attractive => easy to use
  - The interface from a programmer perspective: transparent && maintainable coding

    Be smart in choosing the names of the widgets provided that the programming interface offers some clever functionality to make use of that

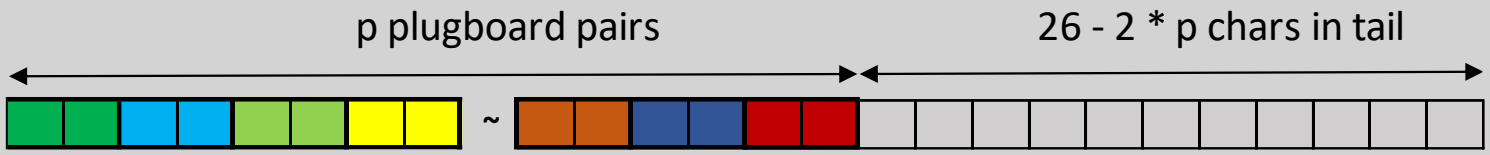
Amount of Enigma keys

- Assume an Enigma with 8 rotors, 3 reflectors and 10 pairs of plugs
- The number of **ordered** selections of three rotors is 8 * 7 * 6 = 336 (A)
- Key settings for 3 rotors 26 * 25 * 26 = 16.900 (B) , due to double stepping the mid term is 25
- Ring settings 26 * 26 = 676 (C), the ring and the key of 3rd rotor is just equal to an other key set
- Number of plugboard permutations with p plugs:

  $26!/((26\text{-}2p)! * p! * 2^p)$ (D)

- Total amount of keys is (A)*(B)*(C)*(D) = 1.7 * 10 ^ 24 = 2 ^ 80
  (the mass of planet Mercury is 3.3 * 10 ^ 23 kg)
- If we allow 0 <= p <= 10 then the amount is 2.4 * 10 ^ 24 = 2 ^ 81
  - not bad for an old mechanical encryption tool

How to count the plugboard combs



There are 26! ways to arrange an alphabet of 26 chars
Only the first 2 * p chars of the alphabets are used for the plugs
The arrangement of the remaining 26 – 2 * p chars doesn't matter

(1) Combs = 26! / (26 – 2 * p)!

There are p pairs of characters, the arrangement of the pairs doesn't matter

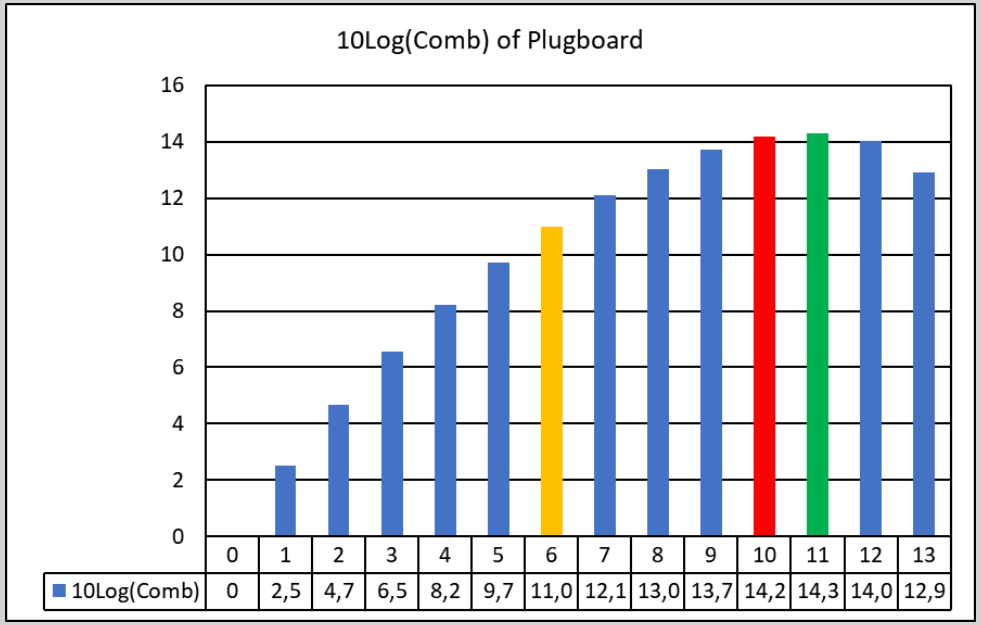
(2) Combs = 26! / ((26 – 2 * p)! * p!)

And the order of chars in each of the p plug pairs doesn't matter either

(3) Combs = 26! / ((26 – 2 * p)! * p! * 2 ^ p)

Number of SB configurations

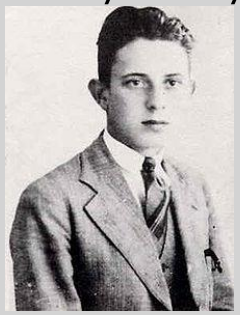
- The amount of permutations depends of course on the number of plugs set but it maximizes at 11, not at 10
  - C(11)/C(10) = 1,4
- Most probable explanation: a calculation error!!
- Most significant factor in the number of permutations is the plugboard BUT: without rotors it is just a mono alphabetic substitution



Plugs	Permutations	C(p)/C(p - 1)	10Log(Comb)
0	1		0
1	325	325,0	2,5
2	44.850	138,0	4,7
3	3.453.450	77,0	6,5
4	164.038.875	47,5	8,2
5	5.019.589.575	30,6	9,7
6	100.391.791.500	20,0	11,0
7	1.305.093.289.500	13,0	12,1
8	10.767.019.638.375	8,3	13,0
9	53.835.098.191.875	5,0	13,7
10	150.738.274.937.250	2,8	14,2
11	205.552.193.096.250	1,4	14,3
12	102.776.096.548.125	0,5	14,0
13	7.905.853.580.625	0,1	12,9
Sum	532.985.208.200.575		14,7

Weaknesses and code breaking (1)

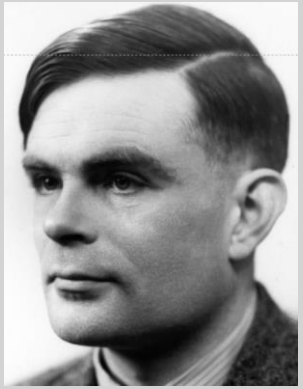
- Initially the operators did send twice a private 3-letter key, that allowed for the breakdown of the first rotors (continued to be used throughout the war)
- The wiring of the first 3-5 rotors was broken by a taskforce of the Polish intelligence. Three smart mathematicians were hired in 1932:
  Marian Rejewski (1905-1980), Jerzy Różycki (1909-1942) and Henryk Zygalski (1908-1978).


  - They used pure mathematics to decode the rotors using group theory and the theory of permutation groups
  - In January 1938 the Polish could break 75% of the messages until the Germans increased plugboard connections and rotors
  - 5 weeks prior to Germanies invasion of Poland (September 1st 1939) the Polish shared their knowledge with France and England and this was a crucial step as the Allies weren't making little progress in decrypting Enigma

Weaknesses and code breaking (2)

- A major weakness is the fact that no character can be encrypted to itself
  - This allows for the search of the position of expected texts in an encrypted message f.e. “Wetterbericht” which can be used to derive plugboard settings
- Operators in the field were identifiable by the rhythm of their Morse transmissions
  - used often the same keys like girlfriends part of name LIS or ANN or just even ABC
  - the preferred keys of an operator and its Morse rhythm were known reducing search time
- It was not encouraged at all by “das Oberkommando” to doubt Enigma
- There was no smart German group to challenge the own encryption methods
- Allen Turing (1912-1954) used probability theory & automation to break the Naval code



Wrap-up

We discussed:

- The construction of the mechanical Enigma and its operational use
- The programming of an Enigma simulator using Qt for GUI and a separate EnigmaLib
- Some interesting issues using Qt
    - Smart processing of the GUI using lists of widgets
    - Boosting Qt compilations
- Some details of the encryption implementation
- The complexity of Enigma compared to modern encryption
- Weaknesses and code breaking

The End

Thank you for your attention!!