Fortgeschrittene Mensch-Computer Interaktion

Usable security in Security Mechanisms

vorgelegt von

Maik Alexander Ruth (Matrikelnummer: 77869)

Yusuf Enes Tamer (Matrikelnummer: 78159)

Studiengang MIN

Hochschule Aalen

Hochschule für Technik und Wirtschaft

Betreut durch Dr. Marc Hermann

Contents

1 Abstract **2**

2 Problemstellung **2**

3 Problemlösung **2**
3.1 UI 2
3.2 Code 4
3.2.1 Drücken der Pintaste 5
3.2.2 Drücken eines ASCII-Zeichens 5
3.2.3 Übertragung eines ASCII-Zeichens in den Backdoor 6
3.2.4 Realisierung der Zeiterfassung 6
3.2.5 Erstellung des Unicodes 7
3.2.6 Entschlüsselung gemäß einer Rekursion 7
3.2.7 Drücken der Stoptaste 8
3.2.8 Drücken der Resettaste 9
3.2.9 Auswählen eines Passworts 9

4 Implementierung einer Symbian-Tastatur **10**

5 Programmausführung - Person ist trainiert für die Passworteingabe **13**

1 Abstract

Bei der Passworteingabe spielen viele verschiedene Faktoren meist eine größere Rolle als gedacht. Eines der größten Faktoren ist die Eingabeperformanz und die Genauigkeit der Eingabe. Die Performanz fügt sich durch die Eingabezeit und die eingegebenen Daten zusammen, womit dann die Performanz durch die Komplexität des Informationsgehalts ermessen lässt. Die Komplexität ist relativ auch an das Taktverhältnis gebunden, sodass die Kombination der eingegebenen Tasten die Komplexität erhöht oder senkt.
Um dies zu ermessen, befasst sich dieses Projekt mit einer virtuellen Passworteingabe unter Verwendung einer Software-Implementierten Tastatur. Mithilfe einer vorhandenen Systemuhr, welche zugreifbar aus einer c++ Bibliothek ist, kann die Zeiterfassung realisiert werden. Auf der linken Hälfte des UIs, findet die Passwort-Authentifizierung statt und auf der rechten Hälfte des UIs sind die Entwickleroptionen von den Passwort-Performanz Tools verfügbar. Damit kann der Benutzer seine Mensch-Comptuer-Interaktion messen und trainieren auf zugrundeliegenden Zeitdaten.

2 Problemstellung

Da bei einem Rechnersystem mehrere Programme aktiv sind, wird bei der Bildschirmausgabe davon mehrere Objekte erstellt. Das Kurzzeitgedächnis von dem Computerbenutzer wird aufgrund der Menge an Informationen überflutet. Damit wird die Eingabeperformanz sowie auch die Genauigkeit beeinträchtigt, sodass die Paswortauthentifizierung fehlschlägt. Aus diesem Grund muss die Passworteingabe trainiert werden. Durch eine schlechte Eingabeperformanz kann ein Angreifer die eingegebenen Daten miterfahren.

3 Problemlösung

3.1 UI

Um dieses Training zu bewerkstelligen soll auf der linken Seite des UIs der Software eine Tastatur bereit gestellt werden, welche für mehrere simple Eingaben verwendet werden kann. Die Eingaben des Benutzers sollen wie bei einer regulären Passworteingabe in Unicode-Kodierung in einem Textfeld über der Tastatur angegeben werden. Bei der Unicode-Kodierung wird ein Symbol verwendet und nicht der wirklich getätigte Buchstabe aus der ASCII-Tabelle, welcher vom Benutzer ausgewählt wurde.
Die Informationen, welche aus der Eingabe hervorgehen, werden auf der rechten Seite des UIs dargestellt und sind damit folgende GUI-Objekte:

- Die Eingabezeit, welche zwischen jeder Eingabe eines neuen Zeichens gemessen wird
- Zwei Buttons, welche die Messung für eine Passworteingabe starten und zu stoppen
- Einen Button, welcher die Eingabezeiten und das Ausgabefeld leert
- Ein Radiobutton, welcher die Unicode-Kodierung im Ausgabefeld rückgängig macht und damit den Klartext anzeigt
- Eine Auswahlliste, aus welcher der Benutzer ein Passwort auswählen kann, welches dann trainiert wird
- Die Angabe, ob das eingegebene Passwort korrekt ist oder nicht

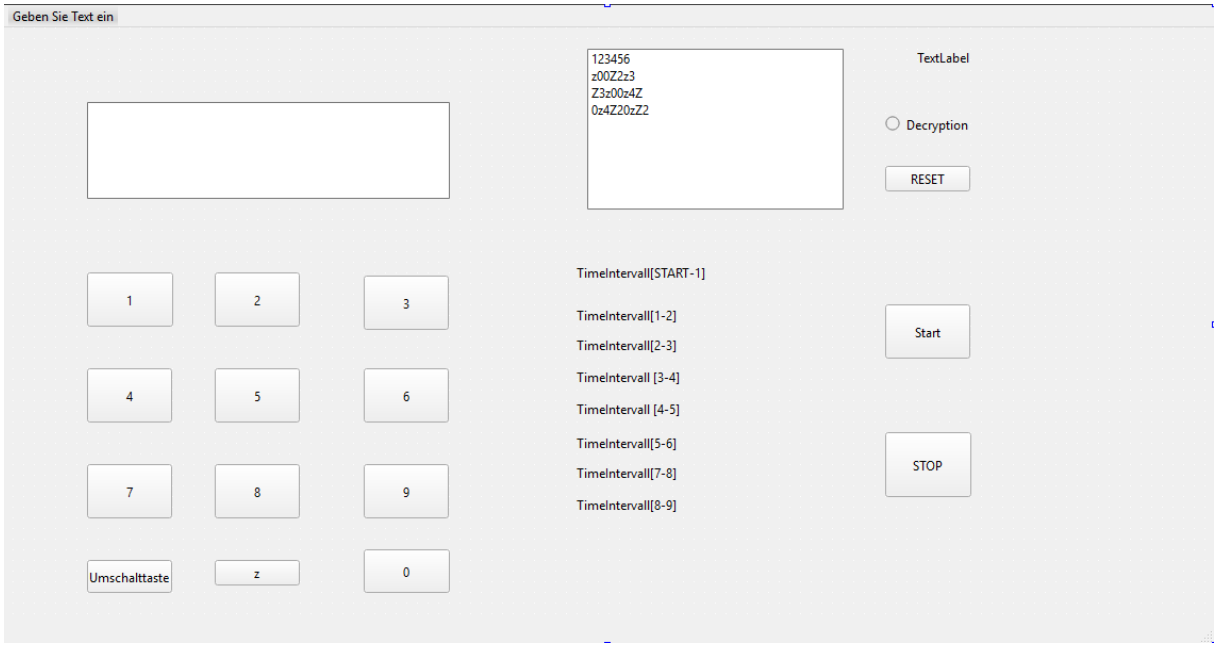

Figure 1: UI der Anwendung

Durch ein gutes Trainingsergebnis wird der Benutzer darauf vorbereitet die Passwörter mit einer hohen Eingabeperformanz einzugeben. Diese Eingaben werden mithilfe der Systemuhr gemessen und können dann vom Benutzer ausgewertet werden. Damit wird bei anderen Personen, welche sich im Umfeld bei der Passworteingabe befinden, die Sicht auf das Eingabefeld blockiert, da die Eingabedaten von dem Benutzer in einer schnellen Taktrelation übertragen werden.

3.2 Code

Für den Code des Programms wurde zunächst ein Klassendiagramm angelegt. Mithilfe dieses Diagramms wurde der Code für das Programm erstellt und entwickelt.

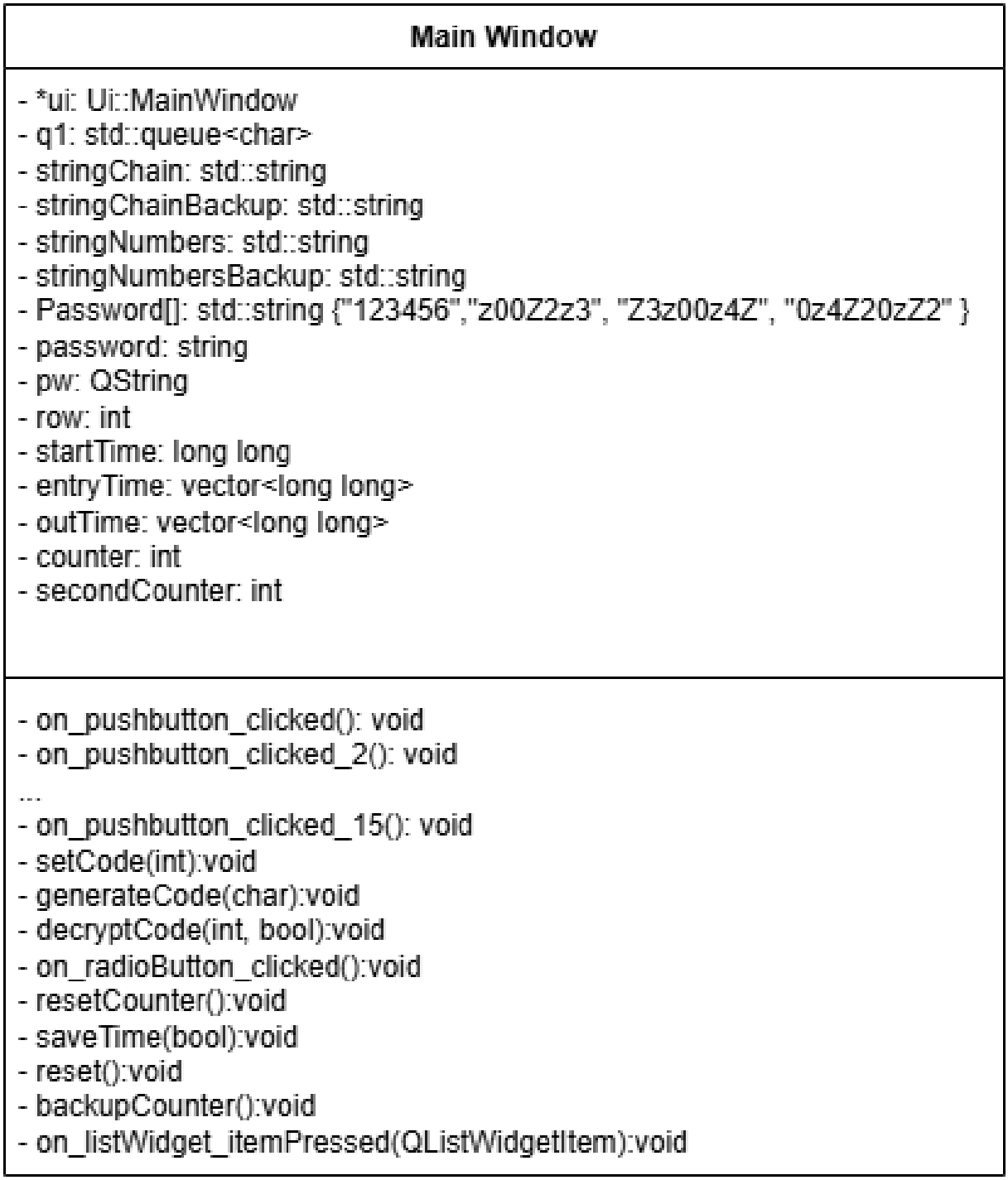

Figure 2: UML

3.2.1 Drücken der Pintaste

Die Funktion der Eingabebuttons ist von der Funktionsweise für jeden Button gleich, es wird lediglich ein anderes ASCII-Zeichen für jeden Button ausgegeben. Das Ausgabefeld des Buttons ist bei der Codegenerierung gekoppelt, weshalb dieser als Parameter verwendet wird, und nach einer Char-Umwandlug der **generate(..)** Funktion übergeben. Der Code für die Zeichen 0-9 ist im folgenden Listing zu sehen:

```
void MainWindow::on_pushButton_clicked()
{
 this->generateCode(ui->pushButton->text().at(0).toLatin1());
}
```

3.2.2 Drücken eines ASCII-Zeichens

Da es sich bei dem Zeichen "Z" um einen Buchstaben handelt, muss dieser die Möglichkeit besitzen sowohl groß als auch klein angebbar zu sein. Deswegen muss neben dem Button für den Buchstaben auch eine Umschalttaste eingefügt und berücksichtigt werden. Die Umschalttaste verändert den Text auf dem "Z" Button, damit der Benutzer auch den Unterschied zwischen gedrückter Umschalttaste und nicht gedrückter Umschalttaste sieht. Der Code für die Umschalttaste ist im folgenden Listing zu sehen:

```
void MainWindow::on_pushButton_11_clicked()
{

    if( "z" == ui->pushButton_12->text())
    {
        ui->pushButton_12->setText("Z");

    }
    else if ("Z" == ui->pushButton_12->text() )
    {
        ui->pushButton_12->setText("z");

    }

}
```

Aufgrund der Umschalttaste ist der Code für den "Z" Buchstaben modifiziert mit einer Abfrage. Dieser Code überprüft den Text, welcher sich gerade auf dem Button selber befindet und abhängig davon ob die Umschalttaste betätigt wurde gibt der Button entweder ein kleines oder ein großes "Z" aus. Dies ist im folgenden Listing zu sehen:

```
void MainWindow::on_pushButton_12_clicked()
{

    if( "z" == ui->pushButton_12->text())
    {
        this->generateCode('z');

    }
    else if ("Z" == ui->pushButton_12->text() )
    {
        this->generateCode('Z');

    }
}
```

3.2.3 Übertragung eines ASCII-Zeichens in den Backdoor

Die generateCode Funktion erfüllt folgende Funktionen:

- Doppelte Realisierung der Zeiterfassung sowie auch die Auswertung
- Speicherung der eingegeben Daten in einer Backdoor bzw. im Hintergrund in einer Q-Datenstruktur
- Bereitstellen eines Zählervariablen, welche die Passwortlänge festlegt
- Aufruf der Funktion setCode, welche den Unicode erzeugt und den eigentlichen Klartext verbirgt

```
void MainWindow::generateCode(char a)
{
    this->saveTime(true);
    this->q1.push(a);
    this->counter++;
    int localCounter=0;
    this->setCode(localCounter);
    this->saveTime(false);
}
```

3.2.4 Realisierung der Zeiterfassung

In der saveTime Funktion wird die Systemuhr aufgerufen und versucht die aktuelle Uhrzeit einzulesen. Bei der Uhrzeit handelt es sich um eine UNIX-Zeituhr. Dieser Zeitstempel wird in Millisekunden angegeben über das Casting-Verfahren. Diese werden in einem Vektor gespeichert. Falls der Parameterwert wahr ist, wird die Eingabezeit in einem Vektor gespeichert. Anderseits wird die Ausgangszeit in einem anderen Vektor geladen.

```
void MainWindow::saveTime(bool v1)
{

    auto now = chrono::system_clock::now();

    // Convert the current time to time since epoch
    auto duration = now.time_since_epoch();

    // Convert duration to milliseconds
    auto milliseconds
        = chrono::duration_cast<chrono::milliseconds>(
              duration)
              .count();
    if(v1)
        this->entryTime.push_back(milliseconds);
    else
        this->outTime.push_back(milliseconds);
}
```

3.2.5 Erstellung des Unicodes

In der setCode Funktion wird der Klartext mit einem Unicode-Zeichen, in diesem Fall dem Zeichen "*", ersetzt und dadurch verborgen. Der Ablauf des Algorithmus ist gemäß einer Rekursion implementiert worden. Die Zeichenkette wird in dem *else*-Zweig verlängert und der Zählerwert wird um den Wert 1 erhöht. Die Funktion ruft sich dann erneut mit dem neuen Zählerstand auf.

```
void MainWindow::setCode(int c)
{

    if(c == this->counter)
    {
        ui->textEdit->setText(this->stringChain.c_str());
        this->stringChainBackUp.assign(this->stringChain);
        this->stringChain = "";
        return;
    }
    else{
        this->stringChain += "*";
        setCode(++c);

    }
    return;
}
```

3.2.6 Entschlüsselung gemäß einer Rekursion

Um den Klartext wieder anzuzeigen kann der Radiobutton in der oberen rechten Ecke des UIs verwendet werden. Wird dieser aktiviert wird der Text, welcher sich im Eingabefeld befindet wieder in Klartext umgewandelt mithilfe der decryptCode Funktion. Der Ablauf des Algorithmus basiert wie die setCode Funktion auf einer Rekursion. Das erste Abfragekonstrukt ist für die Rekursion zuständig, wo beim *else*-Zweig die Stringkette gebildet wird und damit die Daten aus dem Backdoor in den Vordergrund übertragen werden. Sind weitere Zeichen im Backdoor vorhanden, muss die Funktion sich erneut aufrufen. Hierbei werden zwei Parameter übergeben. Bei dem ersten Parameter handelt es sich um einen Zählerwert, welcher für den Ablauf der Rekursion benötigt wird und der zweite Parameter wird in einer verschachtelten Abfrage überprüft. Wenn die Abfrage erfüllt ist, wird nur die Ausgabe erzeugt. Anderenfalls findet eine Zuweisungs-Operation der Stringkette statt, sodass nach dem letzten Zeichen die Stringkette leer wird. Der Zähler und weitere Komponenten innerhalb dieses Frames werden zurückgesetzt.

```
void MainWindow::decryptCode(int c, bool visible)
{

    if(c == this->counter)
    {
        if(visible == true)
            ui->textEdit->setText(this->stringNumbersBackUp.c_str());
        else
            ui->textEdit->setText(this->stringChainBackUp.c_str());
        this->stringNumbersBackUp.assign(this->stringNumbers);
        this->stringNumbers = "";
        this->resetCounter();
        return;
    }
    else
    {
        this->stringNumbers += this->q1.front();
        this->q1.pop();
        decryptCode(++c, visible = false);
    }
}
```

3.2.7 Drücken der Stoptaste

Die Funktion des Stop Buttons hat mehrere Aufgaben. Zuerst wird die Passwortlänge mittels der *backupCounter* abgefragt. Beim Ablauf dieses Algorithmus wird folglich ein Loop implementiert, wobei innerhalb dieses Loops eine Switch Case Auswertung stattfindet. Bei dieser wird der Eingangs- und Ausgangsvektor die Zeiterfassung ausgelesen. Der Index des Eingangvektors ist um 1 höher als beim Ausgangsverktor, da der Übergang zwischen der vorherigen Tasteneingabe und der aktuellen Tasteneingabe ausgegeben wird. Nach der Ausführung soll die *decryptCode*-Funktion aufgerufen werden, um die Backdoordaten bereitzustellen in einer Stringkette. Folglich wird davon die Länge abgefragt sowie auch von dem vorhandenen Passwort auf der Entwicklerseite. Diese beiden Daten werden in einer weiteren Schleife durch zwei Indexierungen verglichen. Der Vergleich basiert auf einer Inkrementierung. Falls der richtige Zählerstand erreicht wurde, wird die Phrase "PASSWORD IS VALID" ausgegeben. Dagegen, wenn ein falscher Zählerstand vorliegt, wird die Phrase "PASSWORD IS NOT VALID" ausgegeben.

```
void MainWindow::on_pushButton_13_clicked()
{
    this->backupCounter();
    for(int i = 1; i<this->secondCounter; i++)
    {
        switch(i)
        {
        case 1:
            ui->label_8->setText(QString::number(this->entryTime[0] - this->
   startTime)); // 1 - 2 [1] - [0]
            break;
        case 2:
            ui->label_9->setText(QString::number(this->entryTime[1] - this->
   outTime[0])); // 2 - 3 [2] - [1]
            break;
        case 3:
            ui->label_10->setText(QString::number(this->entryTime[2] - this->
   outTime[1])); // 3 - 4 [3] - [2]
            break;
        case 4:
        ...
        case 8:
            ui->label_21->setText(QString::number(this->entryTime[7] - this->
   outTime[6]));
            break;
            continue;
        }
    }
    int localCounter = 0;

    this->decryptCode(localCounter,false);
    int n1 = this->stringNumbersBackUp.length();
    int n2 = this->Password[this->row].length();
    int counter2 = 0;
    if(n1 != n2)
        return;
    for(int i = 0; i<n1; i++)
    {
        if(this->stringNumbersBackUp.at(i) == this->Password[this->row].at(i))
            counter2++;
    }
    if(counter2 == n1 && counter2 == n1)
        ui->label_6->setText(QString::fromStdString("PASSWORD IS VALID"));
    else
        ui->label_6->setText(QString::fromStdString("PASSWORD IS NOT VALID"));
    return;
}
```

3.2.8 Drücken der Resettaste

Die *reset*-Funktion hat die Aufgabe sämtliche Eingaben des Benutzers auf der rechten Seite des UIs zu löschen und das Textfeld vollständig zu leeren. Zudem sollen die Zählerstände der Entwicklerausgaben auf der rechten Seite des UIs auf Null gesetzt werden, damit eine weitere Messung der Eingabezeiten durchgeführt werden kann.

```
void MainWindow::reset()
{
    this->resetCounter();
    this->entryTime.clear();
    this->outTime.clear();
    ui->label_8->clear();
    ui->label_9->clear();
    ui->label_10->clear();
    ui->label_11->clear();
    ui->label_12->clear();
    ui->label_18->clear();
    ui->label_19->clear();
    ui->label_21->clear();
   // ui->label_22->clear();
    ui->textEdit->clear();
    ui->label_6->clear();
    this->stringNumbersBackUp = "";
}
```

Bei der *resetCounter*-Funktion wird dem Zähler, welcher für die Rekursionen der *setCode*- und der *decryptCode*-Funktion zuständig ist, zurück auf Null gesetzt. Zudem wird die Angabe, welche aussagt ob das Passwort valide ist oder nicht geleert um ein vollständiges Zurücksetzen zu ermöglichen.

```
void MainWindow::resetCounter()
{
    this->counter = 0;
     ui->label_6->setText(QString::fromStdString(this->password));
}
```

3.2.9 Auswählen eines Passworts

Das *ListWidget* gibt dem Benutzer die Möglichkeit eines von vier angegeben Passwörtern auszuwählen und zu trainieren. Dafür wird der ausgewählte Zeilenwert eingescannt, um auf das Passwortfeld mit dem Attribut *row* zuzugreifen: *Passwort[this −> row]*

```
void MainWindow::on_listWidget_itemPressed(QListWidgetItem *item)
{
    this->row = item->listWidget()->currentRow();
    ui->label_13->setText(QString::fromStdString(to_string(this->row)));
}
```

4 Implementierung einer Symbian-Tastatur

Es wurde eine Symbian-Tastatur implementiert, bei der 26 Großbuchstaben erzeugt werden, sowie auch mit einer Umschalttaste in Kleinbuchstaben umgewandelt. Das Auswählen eines Buchstabens erfolgt über die Taste **[A-z]**. Dafür werden die ConvertButton_X-Funktionen ausgeführt.

```
void MainWindow::on_pushButton_16_clicked()
{
    this->ConvertButton1();
    this->ConvertButton2();
    this->ConvertButton3();
    this->ConvertButton4();
    this->ConvertButton5();
    this->ConvertButton6();
    this->ConvertButton7();
    this->ConvertButton8();
    this->ConvertButton9();
}
```

Innerhalb der ConvertButton-Funktion gibt es eine If/Else Strukturanweisung, die allerdings **nicht** verschachtelt ist.

```
        if( "8" == ui->pushButton_8->text())
        {
            ui->pushButton_8->setText("W");
            return;
        }
        if( "W" == ui->pushButton_8->text())
        {
            ui->pushButton_8->setText("V");
            return;
        }
        if( "V" == ui->pushButton_8->text())
        {
            ui->pushButton_8->setText("X");
            return;
        }
        if( "X" == ui->pushButton_8->text())
        {
            ui->pushButton_8->setText("8");
            return;
        }
```

Um einen Kleinbuchstaben zu generieren, wird die Umschalttaste gedrückt und der folgende Code ausgeführt:

```
void MainWindow::on_pushButton_11_clicked()
{

    if("A" ==ui->pushButton->text())
    {
        ui->pushButton->setText("a");
        this->lowerCase = true;
    }
    else if("a" == ui->pushButton->text())
    {
        ui->pushButton->setText("A");
        this->lowerCase = false;
    }

    if( "B" ==ui->pushButton->text())
    {
        ui->pushButton->setText("b");
        this->lowerCase = true;
    }
    else if("b" == ui->pushButton->text())
    {
        ui->pushButton->setText("B");
        this->lowerCase = false;
    }
    if("C" == ui->pushButton->text())
    {
        ui->pushButton->setText("c");
        this->lowerCase = true;

    }
    else if("c" == ui->pushButton->text())
    {
        ui->pushButton->setText("C");
        this->lowerCase = false;
    }
}
```

Dabei kann bei dem Tastenfeld nicht auf die nächste Sequenz umgeschaltet werden, sodass die Taste **[A-z]** blockiert wird. Diese Blockierung wird innerhalb dieser Funktion umgesetzt mit dem folgenden Code:

```
if(this->lowerCase)
    {
        ui->pushButton_16->setEnabled(false);
    }
    else
    {
        ui->pushButton_16->setEnabled(true);
    }
```

Die Erstellung eines dynamischen Passworts erfolgt auf das Betätigen eines **GENERATE**-Buttons. Hierbei wird die folgende Funktion ausgeführt:

```
void MainWindow::on_pushButton_17_clicked()
{
    if(this->generateStatic == true)
    {
        ui->listWidget->addItem(QString::fromStdString(this->Password[0]));
        ui->listWidget->addItem(QString::fromStdString(this->Password[1]));
        ui->listWidget->addItem(QString::fromStdString(this->Password[2]));
        ui->listWidget->addItem(QString::fromStdString(this->Password[3]));
        this->generateStatic = false;
    }
    QRandomGenerator64 * PRNG = new QRandomGenerator64(this->seed);
    QByteArray * data = new QByteArray();
    QByteArray * output = new QByteArray();
    int x = PRNG->generate();
    QString tmp  = QString::number(x);

    QString tmp3 = QString(QCryptographicHash::hash((tmp.toStdString()),
    QCryptographicHash::Md5).toHex());
    string passwords = tmp3.toStdString();
    string wordLen;
    for(int i = 0; i<2; i++)
    {
        for(int j = 9 * i; j< 9 * i + 9; j++)
        {
            wordLen += passwords.at(j);
        }
        ui->listWidget->addItem(QString::fromStdString(wordLen));
        this->dynamicPassword.push_back(wordLen);
        wordLen.clear();
    }
    this->seed = this->seed +1;
}
```

Für die Passwortgenerierung wird eine Md5-Hashfunktion benutzt. Der Eingabeparameter wird für diese Hash-Funktion von einem PseudoRandomGenerator bestimmt. Die Passwortlänge ist 9 und pro Hashwert resultieren sich zwei Passwörter. Diese Passwörter werden über die **addItem**-Methode in die ListWidget eingetragen. Sowie auch werden diese zwei erzeugten Passwörter im Hintergrund (Backdoor) gespeichert, um später die eigentliche Passworteingabe zu validieren.
Die statischen Passwörter werden beim ersten Anklicken des **GENERATE**-Buttons erzeugt. Beim ersten Anklicken wird zusätzlich zwei dynamische Passwörter aus der MD5-Hashfunktion abgeleitet. Daraufhin werden ab dem zweiten Anklicken nur noch dynamische Passwörter generiert. Pro Klicken eines **GENERATE**-Buttons werden der Liste zwei Passwörter angefügt, sodass die Liste damit um zwei Items erweitert wird.

5 Programmausführung - Person ist trainiert für die Passworteingabe

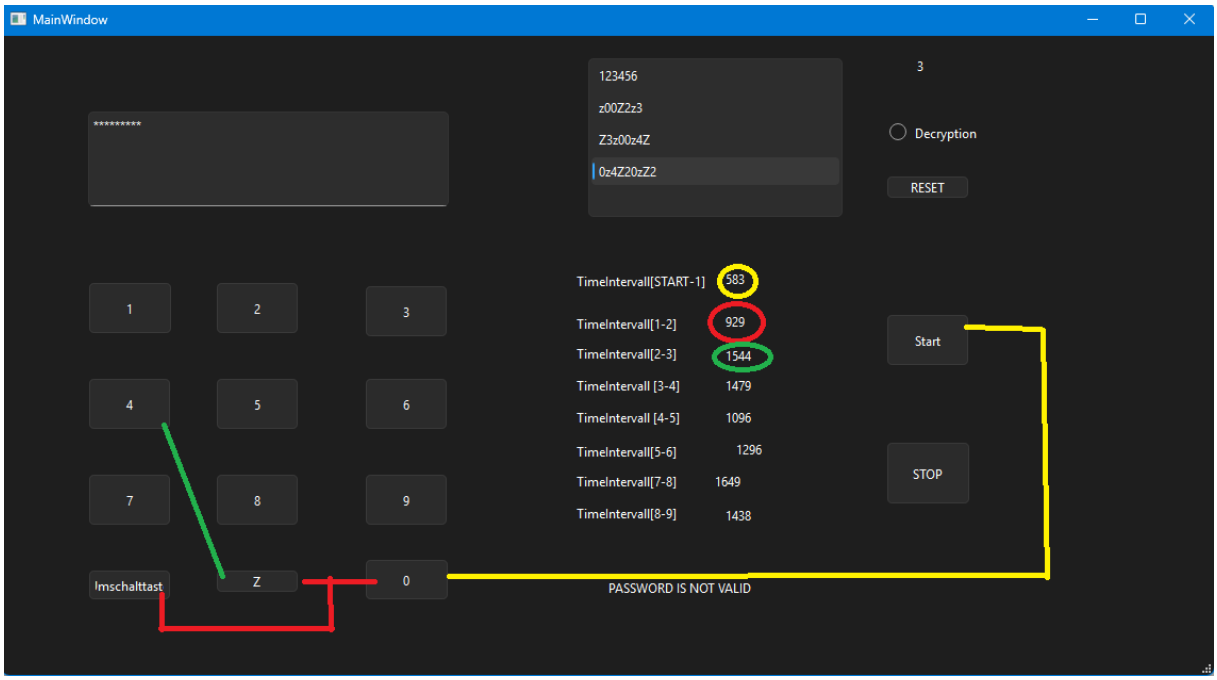

Figure 3: Layout

Auf der vorliegenden Grafik ist zu erkennen, dass die Tasteneingabe hier bereits trainiert wurde und dem Benutzer eine bessere Reaktionszeit zugrunde liegt. Zwischen dem Startbutton und der Taste 0 ist die Distanz am weitesten. Jedoch durch das Training wurde hier eine sehr gute Zeit erwischt, die kürzer ist, als die von der nächsten Eingabe. Zwischen der Taste 0 und z beträgt das Zeitintervall unter einer Sekunde, sowie auch die restlichen Zeiten, die unter zwei Sekunden sind. Damit ist zu erkennen, dass während des Programmstartes der Benutzer unter einer starken Konzentration war und diesen Test erfolgreich absolviert hat.