


```

72:     Label5: TLabel;
73:     Label6: TLabel;
74:     MASCI: TMemo;           // Memo Konsole (ASCII-Zeichensatz).
75:     MHEX: TMemo;           // Memo Konsole (Hexadezimal).
76:     Label3: TLabel;
77:     LDecryptFortschritt: TLabel; // Label über der Fortschrittsanzeige TGauge1.
78:     Ansicht1: TMenuItem;
79:     bersicht1: TMenuItem;    // Übersichtsfenster aufrufen/verstecken.
80:     Gauge1: TGauge;         // Fortschrittsanzeige.
81:     RbSchnelltest: TRadioButton; // Schnelltestverfahren für Decrypt.
82:     RbSinW: TRadioButton;    // weiteres Verfahren für Decrypt.
83:     RbWinS: TRadioButton;    // weiteres Verfahren für Decrypt.
84:     procedure bersicht1Click(Sender: TObject);
85:     procedure FormClose(Sender: TObject; var Action: TCloseAction);
86:     procedure ffneWordlistClick(Sender: TObject);
87:     procedure BtClearWordlistClick(Sender: TObject);
88:     procedure BtLadeStdWordlistClick(Sender: TObject);
89:     procedure Exit1Click(Sender: TObject);
90:     procedure Info2Click(Sender: TObject);
91:     procedure BtDecryptClick(Sender: TObject);
92:     procedure BtEncryptClick(Sender: TObject);
93:     procedure FormCreate(Sender: TObject);
94:
95:
96:
97:     private
98:     { Private-Deklarationen }
99:     public
100:     x: String;               // Übergabevariable; Verschlüsselung -> Entschlüsselung.
101:     w: TStringList;        // Speicherort der Wortliste.
102:     Function gettime:String; // Funktion zum Auslesen der Zeit.
103:     Function StrToHex(s: String): String; //String nach Hexadezimal wandeln.
104: { Public-Deklarationen }
105: end;
106:
107: type
108: TDecrypt = class(TThread)
109:
110:     private
111:     Fx : string;
112:     Fw : TStringList;
113:     Fa, Fb: Integer;
114:     zaehl, zaehl2: integer;
115:     time: extended;
116:     trys: integer;
117:     Ergebnis: string;
118:     procedure Decrypt;
119:     procedure UpdateMemo;
120:     protected
121:     procedure Execute; override;
122:
123:     //procedure UpdateMemo;
124:     public
125:     constructor Create(var a,b: integer; x: string; w: TStringList);
126:     end;
127:
128: var
129:     Form1: TForm1;
130:     MyThread1, MyThread2, MyThread3, MyThread4 : TDecrypt;
131:
132: implementation
133:
134: uses Unit2; //Übersichtsfenster einbinden.
135:
136: {$R *.dfm}
137:
138:
139:
140: function OpenThread(dwDesiredAccess: DWord;
141:     bInheritHandle: Bool;
142:     dwThreadId: DWord): DWord; stdcall; external 'kernel32.dll';

```

```
143:
144: function GetCurrentThreadId: Cardinal; stdcall; external 'kernel32.dll';
145:
146: function SetThreadAffinityMaskByID(ID, AffinityMask: Cardinal): Boolean;
147: var
148:     Handle: THandle;
149:     const
150:     THREAD_TERMINATE           = $0001;
151:     THREAD_SUSPEND_RESUME     = $0002;
152:     THREAD_GET_CONTEXT        = $0008;
153:     THREAD_SET_CONTEXT        = $0010;
154:     THREAD_SET_INFORMATION     = $0020;
155:     THREAD_QUERY_INFORMATION  = $0040;
156:     THREAD_SET_THREAD_TOKEN   = $0080;
157:     THREAD_IMPERSONATE        = $0100;
158:     THREAD_DIRECT_IMPERSONATION = $0200;
159:     THREAD_SET_LIMITED_INFORMATION = $0400;
160:     THREAD_QUERY_LIMITED_INFORMATION = $0800;
161:     THREAD_ALL_ACCESS         = STANDARD_RIGHTS_REQUIRED or SYNCHRONIZE or $03FF;
162: begin
163:     Result:=False;
164:     Handle:=OpenThread(THREAD_SET_INFORMATION or THREAD_QUERY_INFORMATION, False, ID);
165:     if Handle<>0 then begin
166:         Result:=SetThreadAffinityMask(Handle, AffinityMask)<>0;
167:         CloseHandle(Handle);
168:     end;
169: end;
170:
171: constructor TDecrypt.Create;
172: begin
173:     inherited Create(False);
174:     Fx := x;
175:     Fw := w;
176:     Fa := a;
177:     Fb := b;
178:     FreeOnTerminate := true;
179:     Priority          := tpNormal;
180: end;
181:
182: procedure TDecrypt.UpdateMemo;
183: begin
184:     Form1.LZeit.Caption := floattostr(strtfloat(Form1.LZeit.Caption)+ time);
185:     Form1.LDecrypt1.Caption := inttostr(strtoint(Form1.LDecrypt1.Caption) + trys);
186:     Form1.LDecrypt2.Caption := inttostr(strtoint(Form1.LDecrypt2.Caption) + zaehl);
187:     Form1.MASCI.Lines.Add(Ergebnis);
188: end;
189:
190: procedure TDecrypt.Execute;
191: begin
192:     Decrypt;
193:     Synchronize(UpdateMemo);
194: end;
195:
196: procedure TDecrypt.Decrypt;
197: var i, j, l, m: Integer; // Lokale Zählvariablen für Schleifen.
198:     s, t: String;
199:     d: Integer; // Zählvariable für Stringlist.
200:     g, h: Boolean; // g: Fund (ja/nein); h: Teilfund (ja/nein).
201:     p, q: Extended;
202: begin
203:     Form1.Gaugel.Progress := 0;
204:     g := false;
205:     h := false;
206:     zaehl := 0;
207:     zaehl2 := 0;
208:     p := 0;
209:     q := 0;
210:     trys := 0;
211:     time := 0;
212:     if (Form1.ComboBoxEx1.Text = 'Brute-Force') and (Form1.BtLadeStdWordlist.Enabled = false) and
d (length(Fx) > 0) then //ComboBoxEx und Wortliste prüfen.
```

```
213: begin
214:
215:   if (length(Fx) > 0) then           // s ungleich 0 Zeichen.
216:   begin
217:     for i := Fa to Fb do // Testen aller möglichen Schlüssel (0-255).
218:     begin
219:       s := Fx;           // Übergebenen String in Lokale Variable übergeben. // Anzeige für die
Anzahl getesteteter
220:       inc(zaehl);
221:       inc(zaehl2);
222:
223:       for l := 1 to length(s) do //Entschlüsseln mit aktuellem Key-i.
224:       begin
225:         s[l] := char(i Xor ord(s[l]));
226:         inc(zaehl);
227:       end;
228:
229:       // In Konsole ausgeben, welche Schlüsselmöglichkeit getestet wird.
230:
231:
232: // Drei Methoden, um Entschlüsselungsprodukt auf Fund zu überprüfen.
233: if Form1.RbSchnelltest.Checked = true then //Standard Methode.
234: begin
235:   for d := 0 to Fw.count-1 do // Wort 0 - letztes Wort.
236:   begin
237:     inc(zaehl);
238:     if s = Fw.Strings[d] then // Wort d mit s Vergleichen.
239:     begin
240:
241:       g := true;           // Booleanvariable für Fund -> break;.
242:       //Zeitnahme.
243:     end;
244:   end;
245: end;
246:
247: if Form1.RbSinW.Checked = true then //Methode: Wort in der Wordlist suchen.
248: begin
249:   inc(zaehl);
250:   if Fw.IndexOf(s) > 0 then
251:   begin
252:     g := true;
253:   end
254:   else
255:     for j := length(s)-1 downto 3 do
256:     begin
257:       inc(zaehl);
258:       t := Copy(s, 1, j);
259:       if Fw.IndexOf(t) > 0 then
260:       begin
261:         h := true;
262:       end;
263:     end;
264:   end;
265:
266: if Form1.RbWinS.Checked = true then // Methode: Wort der Wortliste in
267: begin // entschlüsseltem Wort suchen.
268:   if Fw.IndexOf(s) > 0 then
269:   begin
270:     inc(zaehl);
271:     g := true;
272:   end
273:   else
274:   begin
275:     for m := 0 to Fw.Count - 1 do
276:     begin
277:       inc(zaehl);
278:       if AnsiPos(Fw[m],s) > 0 then
279:       begin
280:         h := true;
281:       end;
282:     end;
```

```

283:         end;
284:     end;
285:
286: //Überprüfung auf Funde.
287:     if g = true then           //Fund.
288:     begin
289:         Ergebnis := '<' + Form1.gettime + '> ' + inttostr(ThreadID) + ' Final: ' + s + #13#1
0 + ' Entschlüsselung erfolgreich.';
290:         trys := zaehl2;
291:         q := GetTickCount;
292:         break;
293:     end
294: else if h = true then        //Teilfund.
295: begin
296:     Ergebnis := '<' + Form1.gettime + '> ' + inttostr(ThreadID) + ' Teilfund in: ' + s;
297:     trys := zaehl2;
298:     q := GetTickCount;
299:     break;
300: end;
301: Form1.Gaugel.Progress := Form1.Gaugel.Progress + 1;
302: end;
303: if (i = Fb+1) and (g = false) then //Fehlschlag.
304: begin
305:     Ergebnis := '<' + Form1.gettime + '> ' + inttostr(ThreadID) + ' Fehlschlag: ' + s;
306:     trys := zaehl2;
307:     q := GetTickCount;
308: end;
309: end;
310: end;
311:
312: //Entschlüsseln mit Schlüsseleingabe.
313: if Form1.ComboBoxEx1.Text = 'Schlüssel' then //Direkte Entschlüsselung.
314: begin
315:     s := Fx;
316:     for l := 1 to length(s) do
317:     begin
318:         s[l] := char(strtoint(Form1.ESchluessel.Text) Xor ord(s[l]));
319:     end;
320:     Ergebnis := '<' + Form1.gettime + '> ' + inttostr(ThreadID) + ' Entschlüsselung mit Schlüss
el: ' + s;
321:     trys := 1;
322: end;
323: time := ((q-p)/1000);
324: end;
325:
326:
327: //=====Functions=====
328: Function TForm1.StrToHex(s: String): String;
329: var i: Integer;
330: begin
331:     result:='';
332:     if length(s) > 0 then
333:         for i := 1 to length(s) do
334:             result := result + IntToHex(Ord(s[i]),2);
335:         end;
336:
337: Function TForm1.GetTime: String;
338: var a: TDateTime;
339: begin
340:     a := Time(); // Systemzeit auslesen.
341:     result := FormatDateTime('hh:nn:ss:zzz',a); // Zeit formatieren und ausgeben.
342: end;
343:
344: Function GetFileVersion(Path: string): string; // vorgefertigte Function
345: var // zum Auslesen der Programm-
346: lpVerInfo: pointer; // version.
347: rVerValue: PVSFixedFileInfo;
348: dwInfoSize: cardinal;
349: dwValueSize: cardinal;
350: dwDummy: cardinal;
351: lpstrPath: pchar;

```

```
352: begin
353:   if Trim(Path) = EmptyStr then
354:     lpstrPath := pchar(ParamStr(0))
355:   else lpstrPath := pchar(Path);
356:   dwInfoSize := GetFileVersionInfoSize(lpstrPath, dwDummy);
357:   if dwInfoSize = 0 then
358:     begin
359:       Result := 'No version specification';
360:       Exit;
361:     end;
362:   GetMem(lpVerInfo, dwInfoSize);
363:   GetFileVersionInfo(lpstrPath, 0, dwInfoSize, lpVerInfo);
364:   VerQueryValue(lpVerInfo, '\\', pointer(rVerValue), dwValueSize);
365:
366:   with rVerValue^ do
367:     begin
368:       Result := IntToStr(dwFileVersionMS shr 16);
369:       Result := Result + '.' + IntToStr(dwFileVersionMS and $FFFF);
370:       Result := Result + '.' + IntToStr(dwFileVersionLS shr 16);
371:       Result := Result + '.' + IntToStr(dwFileVersionLS and $FFFF);
372:     end;
373:   FreeMem(lpVerInfo, dwInfoSize);
374: end;
375:
376: //=====
377: //=====Hauptprogramm=====
378: //=====
379:
380: //=====Verschlüsselung=====
381: Procedure TForm1.BtEncryptClick(Sender: TObject);
382: var s: String; // Lokale Variablen: s - String der verschlüsselt werden soll.
383:     i: Integer; // i - Zählvariable für Schleife.
384:     key: Integer; // Schlüssel.
385: begin
386:   Gaugel.Progress := 0;
387:   s := MKLartext.text;
388:
389:   if MKLartext.text = '' then
390:     exit
391:   else if (length(s) > 0) then
392:     begin
393:       Form2.Label15.Caption := s;
394:       Form2.Label6.Caption := ESchluessel.Text;
395:       Form2.Image1.Visible := true;
396:       Form2.Image2.Visible := true;
397:       LDecrypt2.caption := '0';
398:       LDecrypt1.caption := '0';
399:       x := s;
400:       LEncrypt2.Caption := '0';
401:       key := StrToInt(ESchluessel.Text);
402:       for i := 1 to length(s) do
403:         begin
404:           LEncrypt2.Caption := inttostr(strtoint(LEncrypt2.caption) + 1);
405:           s[i] := char(key xor Ord(s[i]));
406:         end;
407:       MASCI.Text := '';
408:       MHEX.Text := '';
409:       MASCI.Lines.Add('<' + gettime + '> ' + 'Klartext: ' + x);
410:       MHEX.Lines.Add('<' + gettime + '> ' + 'Klartext: ' + strtohex(x));
411:       MASCI.Lines.Add('<' + gettime + '> ' + 'Verschlüsselter Text: ' + s);
412:       MHEX.Lines.Add('<' + gettime + '> ' + 'Verschlüsselter Text: ' + strtohex(s));
413:       LDecryptFortschritt.Caption := 'Klartext verschlüsselt.';
414:       Form2.Label8.Caption := s;
415:       x := s; // Übergabe des Endprodukts.
416:     end
417: end;
418:
419: //=====Entschlüsselung=====
420: Procedure TForm1.BtDecryptClick(Sender: TObject);
421: var a,b,c,d,e,f,g,h: Integer;
422: begin
```

```
423: a := 0;
424: b := 64;
425: c := 65;
426: d := 128;
427: e := 129;
428: f := 192;
429: g := 193;
430: h := 255;
431: SetThreadAffinityMaskByID(GetCurrentThreadId, 0); // Bitset für cpu: 4 cpus -> 0000 bit
432: MyThread1 := TDecrypt.Create(a, b, x, w); // bsp: 0100 entspricht 4, bedeutung
433: SetThreadAffinityMaskByID(MyThread1.ThreadID, 1); // CPU 3 aktiviert.
434: MyThread2 := TDecrypt.Create(c, d, x, w);
435: SetThreadAffinityMaskByID(MyThread2.ThreadID, 2);
436: MyThread3 := TDecrypt.Create(e, f, x, w);
437: SetThreadAffinityMaskByID(MyThread3.ThreadID, 4);
438: MyThread4 := TDecrypt.Create(g, h, x, w);
439: SetThreadAffinityMaskByID(MyThread3.ThreadID, 8);
440: end;
441:
442:
443:
444: //=====
445: //=====Diverses=====
446: Procedure TForm1.FormCreate(Sender: TObject); //Initialisierung des Programms.
447: begin
448: Form1.Caption := 'MTXorCryptBeta' + ' v.' + GetFileVersion(Application.Exename);
449: Form1.Height := 739; //Maße bestimmen.
450: Form1.Width := 688;
451: Form1.LZeit.Caption := '0';
452: RbSchnelltest.Checked := true;
453: RbWinS.Checked := false;
454: RbSinW.Checked := false;
455: MASCI.Text := ''; //Inhalt Memofenster bei Anwendungsstart löschen.
456: MKlartext.Text := '';
457: MHEX.Text := '';
458: LDecrypt1.Caption := '';
459: end;
460:
461: Procedure TForm1.Info2Click(Sender: TObject);
462: begin
463: ShowMessage(' Xor Ver- und Entschlüssler' + #13#10 + '
(c) Benedikt Welp' + #13#10 + 'erstellt mit BORLAND® DELPHI® 2006 PROFESSIONAL ');
464: end;
465:
466: Procedure TForm1.Exit1Click(Sender: TObject);
467: begin
468: Form1.close; //Anwendung beenden; Speicher der Stringlist
469: end; //wird automatisch in "OnClose" freigeräumt.
470:
471: Procedure TForm1.BtLadeStdWordlistClick(Sender: TObject);
472: begin
473: w := TStringList.create; //Stringlist erzeugen.
474: w.LoadFromFile('germanwordlist.txt'); //Wordlist aus Programmverzeichnis laden.
475: LWordlistStatus.Caption := inttostr(w.count) + ' Worte Geladen!'; //angeben ob Liste geladen
.
476: BtLadeStdWordlist.enabled := false; //neuladen verhindern.
477: ffneWordlist1.Enabled := false;
478: Gaugel.Progress := 0;
479: Form2.Label2.Caption := inttostr(w.count) + ' Worte in germanwordlist.txt';
480: end;
481:
482:
483:
484: Procedure TForm1.BtClearWordlistClick(Sender: TObject);
485: begin
486: BtLadeStdWordlist.enabled := true; //neuladen aktivieren.
487: ffneWordlist1.Enabled := true;
488: w.Free; //Speicher der alten Liste freigeben.
489: w := nil; //w "entknüpfen".
490: LWordlistStatus.Caption := ''; //angeben, dass nichts geladen ist.
491: Gaugel.Progress := 0;
```

```
492:   Form2.Label2.Caption := '';
493: end;
494:
495: Procedure TForm1.ffneWordlist1Click(Sender: TObject);
496: begin
497:   if opendirlog1.execute then           //ermöglicht laden beliebiger Wordlists.
498:   begin
499:     if Assigned(w) then w.Free;
500:     w := TStringList.create;
501:     w.LoadFromFile(opendirlog1.FileName);
502:     LWordlistStatus.Caption := inttostr(w.count) + ' Worte Geladen!';
503:     BtLadeStdWordlist.enabled := false;           //neuladen verhindern.
504:     ffneWordlist1.Enabled := false;
505:     Gage1.Progress := 0;
506:     Form2.Label2.Caption := inttostr(w.count) + ' Worte in ' + ''' + opendirlog1.FileName + '''
;
507:   end;
508: end;
509:
510: procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
511: begin
512:   if assigned(w) then w.free;
513: end;
514:
515: procedure TForm1.bersicht1Click(Sender: TObject);
516: begin
517:   if bersicht1.Checked = false then // Prüfen ob Übersichtsfenster aktiviert
518:   begin // ist, und es dann aktivieren oder
519:     Form2.Visible := true;           // deaktivieren.
520:     bersicht1.Checked := true;
521:   end
522:   else begin
523:     bersicht1.Checked := false;
524:     Form2.Visible := false;
525:   end;
526: end;
527:
528: end.
```