Tabelleninhalte verschlüsseln

Der Schutz sensibler Daten in Access-Datenbanken ist keine triviale Angelegenheit. Die Tabellen liegen offen in nur einer Datei, die problemlos auf einen USB-Stick kopiert werden kann, um deren Inhalte anderenorts in aller Ruhe zu inspizieren. Frühere Access-Versionen bis Office 2003 erlaubten die Absicherung über ein Berechtigungsmodell, das später leider ersatzlos gestrichen wurde. Wer auf Nummer Sicher gehen will, der sollte sich Gedanken über eine Datenverschlüsselung machen.

Beispieldatenbank

Die Beispiele dieses Artikels finden Sie in der Datenbank **1701_Cryto.accdb**.

Strings verschlüsseln per VBA

Zu den Daten, die in einer Tabelle nicht unbedingt auf den ersten Blick einsehbar sein müssen, gehören vor allem Texte. In einer Adressenliste etwa beträfe dies die Namensfelder, die Telefon- und E-Mail-Daten, sowie andere Adressfelder. Verschlüsselt man diese, so kann ein Unbefugter mit ihnen nicht allzu viel anfangen.

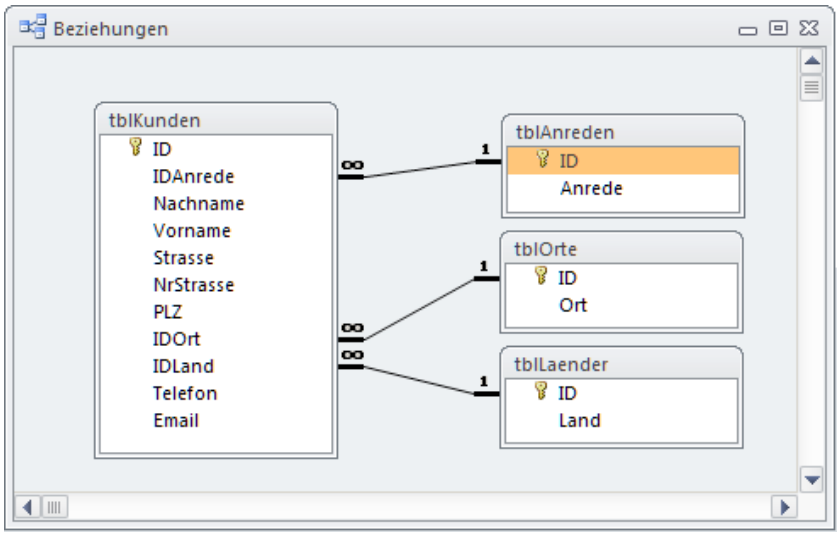


Bild 1: Das Datenmodell der Beispieldatenbank zu Kundenadressen

In der Beispieldatenbank bekommen wir es wieder einmal mit der bekannten Kundentabelle zu tun, deren Datenmodell Bild 1 wiedergibt. Neben den Nachschlagetabellen zu Anreden, Orten und Ländern enthält die Haupttabelle **tblKunden** die üblichen Adressfelder. Wir machen uns zur Aufgabe, die folgenden Felder zu verschlüsseln: **Nachname**, **Vorname**, **Strasse**, **Telefon** und **Email**. Das sollte ausreichen, um die Datensätze so zu verunstalten, dass aus ihnen keine Rückschlüsse auf den ursprünglichen Inhalt mehr möglich sind.

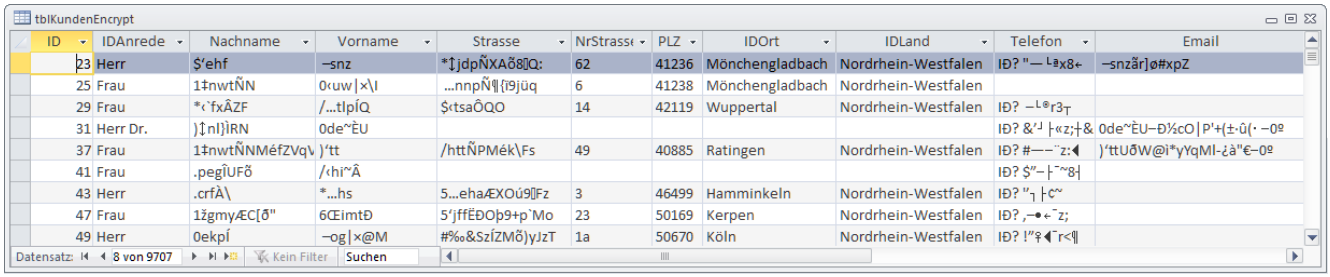
Der Ausgangszustand zeigt sich in Bild 2. Hier sind alle Adressdaten in konventioneller Weise abgespeichert. Jeder kann sie einsehen und gegebenenfalls stehlen. Dazu benötigt man noch nicht einmal eine **Office**-Installation, denn die Datenzugriffsbibliothe-

tblKunden

ID	IDAnrede	Nachname	Vorname	Strasse	NrStrasse	PLZ	IDOrt	IDLand	Telefon	Email
23	Herr	Fuchs	Bruno	Hölderlinstr.	62	41236	Mönchengladbach	Nordrhein-Westfalen	+49 747413384	Bruno@Fuchs.de
25	Frau	Schwarz	Roswitha	Bahner Straße	6	41238	Mönchengladbach	Nordrhein-Westfalen		
29	Frau	Hoffmann	Marlene	Forstweg	14	42119	Wuppertal	Nordrhein-Westfalen	+49 55755985	
31	Herr Dr.	Köhlhoff	Rebekka						+49 310101032	Rebekka.Köhlhoff@yahoo.de
37	Frau	Schwarzer-Queise	Kurt	Mintarder Weg	49	40885	Ratingen	Nordrhein-Westfalen	+49 64263112	Kurt@Schwarzer-Queiser.de
41	Frau	Levermann	Monika						+49 17214534	
43	Herr	Liersch	Hans	Wachtelgarten	3	46499	Hamminkeln	Nordrhein-Westfalen	+49 216195	
47	Frau	Szamlewski	Thomas	Wulfshäger Str.	23	50169	Kerpen	Nordrhein-Westfalen	+49 9533410	
49	Herr	Ricken	Brigitte	Am Sonnenbrink	1a	50670	Köln	Nordrhein-Westfalen	+49 47894977	

Datensatz: 8 von 9706 | Kein Filter | Suchen

Bild 2: Einige Datensätze der normalen Kundentabelle in der Datenblattansicht



tblKundenEncrypt

ID	IDAnrede	Nachname	Vorname	Strasse	NrStrasse	PLZ	IDOrt	IDLand	Telefon	Email
23	Herr	$'ehf	–snz	*↕jdpÑXAð8▯Q:	62	41236	Mönchengladbach	Nordrhein-Westfalen	IÐ? "— ᴸª x8←	–snzãr]ø#xpZ
25	Frau	1‡nwtÑN	0‹uw|x\l	…nnpÑ¶{i9jüq	6	41238	Mönchengladbach	Nordrhein-Westfalen		
29	Frau	*‹`fxÂZF	/…tlpíQ	$‹tsaÔQO	14	42119	Wuppertal	Nordrhein-Westfalen	IÐ? –ᴸ⁶r3ᴛ	
31	Herr Dr.	}‡nl}iRN	0de~ÈU						IÐ? &'ᴶ ├«z;┼&	0de~ÈU–Ð½cO|P'+(±·û(· –0º
37	Frau	1‡nwtÑNMéfZVq\	)'tt	/httÑPMék\Fs	49	40885	Ratingen	Nordrhein-Westfalen	IÐ? #—–¨z:◀	)'ttUðW@i*yYqMl-¿à"€–0º
41	Frau	.pegÎUFõ	/‹hi~Â						IÐ? $"–├¯~8┤	
43	Herr	.crfÀ\	*…hs	5…ehaÆXOú9▯Fz	3	46499	Hamminkeln	Nordrhein-Westfalen	IÐ? "┐├¢˜	
47	Frau	1žgmyÆC[ð"	6ŒimtÐ	5'jffÉDOþ9+p`Mo	23	50169	Kerpen	Nordrhein-Westfalen	IÐ? ,–●←˜z;	
49	Herr	0ekpí	–og|×@M	#‰&SzÍZMõ)yJzT	1a	50670	Köln	Nordrhein-Westfalen	IÐ? !"♀◀¯r<¶	

Datensatz: 8 von 9707 | Kein Filter | Suchen

Bild 3: Dieselben Datensätze der verschlüsselten Kundentabelle in der Datenblattansicht

ken **DAO** und **ADO** bringt **Windows** bereits von Haus aus mit. Selbst unter **Linux/Android** und etwa das Zugriffssystem **JDBC** käme man an die Tabelle heran. Das wäre zwar auch nach Verschlüsselung noch der Fall, doch dann nützt der Einblick wenig, wie Bild 3 beweist. Hier sind die erwähnten Textfelder über eine Aktualisierungsabfrage einer **RC4**-Verschlüsselung unterzogen worden, auf die wir noch zu sprechen kommen.

Verschlüsselungsalgorithmen

Access und VBA selbst bringen direkt keine Verschlüsselungsmethoden mit. Sie sind deshalb auf eigene VBA-Implementierungen in Form von Funktionen angewiesen.

Listing 1 zeigt eine extrem einfache Methode, die sich die VBA-Funktion **StrReverse** zunutze macht. Sie kehrt einen übergebenen String schlicht um:

```
CryptReverse "Access Basics"
->  scisaB sseccA /  Access Basics
```

Das Ergebnis stellt einen Hacker vor keinerlei Herausforderungen. Der Algorithmus ist ganz offensichtlich – von Verschlüsselung kann man hier wohl kaum sprechen. Anders sieht es bei der Routine zum zweiten Beispiel aus (siehe Listing 2), das alle Zeichen des übergebenen Strings einer binären **XOR**-Operation unterzieht:

```
CryptXOR "Access Basics"
->  ¾œœšŒŒƁ½ŽŒ-œŒ  / Access Basics
```

In der ersten **For-Next**-Schleife werden die einzelnen Zeichen des Strings zunächst über die **Mid**-Funktion ermittelt und deren **ASCII**-Ordinalzahl in der Variablen **nOrd** (**Long**) gespeichert. Diese wird binär mit **Xor 255** behandelt und anschließend per **Chr**-Funktion wieder in ein Zeichen verwandelt. Die Stringvariable **sCrypt** nimmt diese Zeichen nacheinander auf. Die zweite Schleife macht die Probe und wiederholt den

```
Sub CryptReverse(S As String)
    Dim sCrypt As String

    sCrypt = StrReverse(S)
    S = StrReverse(sCrypt)
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 1: Textverschlüsselung über eine String-Umkehr

```
Sub CryptXOR(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim nOrd As Long
    n = Len(S)
    sCrypt = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(S, i, 1))
        Mid(sCrypt, i) = Chr((nOrd Xor 255))
    Next i
    S = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(sCrypt, i, 1))
        Mid(S, i) = Chr((nOrd Xor 255))
    Next i
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 2: Textverschlüsselung über ein Byte-weises XOR

Vorgang auf genau die gleiche Weise. Verschlüsseln und Entschlüsseln geschehen also über den identischen Algorithmus, so, wie dies auch bei **StrReverse** der Fall ist. Tatsächlich handelt es sich bei Anwendung von **Xor 255** um eine ähnliche Angelegenheit, da hier die Bits eines Bytes schlicht umgekehrt werden:

```
00110101 -> 11001010
```

Der Ergebnis-String ist nun ziemlich kryptisch, doch auch hier hat ein Hacker sicher schnell die Lösung gefunden. Also versuchen wir es ihm noch schwerer zu machen.

Die Routine **CryptXORReverse** (Listing 3) kombiniert die beiden beschriebenen Prozeduren. Erst greift wieder der **XOR**-Algorithmus, doch anschließend kommt noch **StrReverse** zum Einsatz. Ergebnis:

```
CryptXORReverse "Access Basics"
->  Œœ-ŒŽ½ßŒŒŠœœ¾  /  Access Basics
```

```
Sub CryptXORReverse(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim nOrd As Long
    n = Len(S)
    sCrypt = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(S, i, 1))
        Mid(sCrypt, i) = Chr((nOrd Xor 255))
    Next i
    sCrypt = StrReverse(sCrypt)
    S = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(sCrypt, i, 1))
        Mid(S, i) = Chr((nOrd Xor 255))
    Next i
    S = StrReverse(S)
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 3: Textverschlüsselung über Byte-weises XOR plus String-Umkehr

Deutlich schwerer nachvollziehbar sind Algorithmen, die sich etwa die **Mod**-Funktion von VBA zunutze machen, welche die Ordinalzahl eines Zeichens quasi rotieren lässt. Listing 4 demonstriert so eine Lösung.

Statt **Xor** wird der Ordinalzahl in **nOrd** hier die willkürliche Zahl **144** hinzuaddiert und über **Mod** sichergestellt, dass dabei keine Zahl **>255** herauskommt.

Ansonsten gleicht der Algorithmus dem aus Listing 2. Zum Entschlüsseln können Sie allerdings nicht die identische Funktion verwenden. Der zweite Teil der Routine zeigt, dass hier zunächst vom Byte-Wert die Zahl **144** subtrahiert werden muss, um daraus wieder das korrekte Zeichen zu machen. Das Ergebnis hier:

```
CryptAddMod "Access Basics"
->  Ñóóõ°Òñùó  /  Access Basics
```

Die bisherigen Algorithmen haben den Nachteil, dass der Ergebnis-String die gleiche Länge hat, wie der Eingabe-String.

Die Routine **CryptDist** in Listing 5 ändert das. Hier bekommt man einen String variabler Länge heraus, in

```
Sub CryptAddMod(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim nOrd As Long
    n = Len(S)
    sCrypt = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(S, i, 1))
        Mid(sCrypt, i) = Chr(((nOrd + 144) Mod 256))
    Next i
    S = String(n, 0)
    For i = 1 To n
        nOrd = Asc(Mid(sCrypt, i, 1)) - 144
        Mid(S, i) = Chr((nOrd + 256) Mod 256)
    Next i
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 4: Textverschlüsselung über die Ordinalzahl der Zeichen und Mod

```
Sub CryptDist(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim nOrd As Long
    Dim arr() As String
    For i = 1 To Len(S)
        nOrd = Asc(Mid(S, i, 1))
        sCrypt = sCrypt & StrReverse(CStr(nOrd)) & Chr$(224)
    Next i
    S = String(Len(S), 0)
    arr = Split(sCrypt, Chr$(224))
    For i = 0 To UBound(arr) - 1
        nOrd = Val(StrReverse(arr(i)))
        Mid(S, i + 1) = Chr(nOrd)
    Next i
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 5: Textverschlüsselung über mehrere kompinierte Methoden

```
Sub CryptHex(S As String)
    Dim sCrypt As String
    Dim i As Long, n As Long
    Dim sOrd As String, nOrd As Long
    For i = 1 To Len(S)
        sOrd = "00"
        Mid(sOrd, 1) = StrReverse(Hex(Asc(Mid(S, i, 1))))
        sCrypt = sCrypt & sOrd
    Next i
    S = ""
    For i = 1 To Len(sCrypt) Step 2
        sOrd = Mid(sCrypt, i, 2)
        nOrd = Val("&H" & StrReverse(sOrd))
        S = S & Chr(nOrd)
    Next i
    Debug.Print sCrypt, "/", S
End Sub
```

Listing 6: Textverschlüsselung unter Einsatz der Hex-Funktion

dem nur noch sehr schwer das Original auszumachen ist:

```
CryptDist "Access Basics"
->  56à99à99à101à511à511à23à66à79à511à501à99à511à  /
Access Basics
```

Eine Beschreibung der Funktionsweise lassen wir an dieser Stelle außen vor. Trotz der Kürze der Prozedur fällt sie doch recht komplex aus.

Und schließlich ein letztes Beispiel (siehe Listing 6), das die **Hex**-Funktion von VBA heranzieht:

```
CryptHex "Access Basics"
-> 14363656373702241637963637 / Access Basics
```

Auch hier lässt das Ergebnis sicher kaum Rückschlüsse auf den Ausgangs-String zu.

Sie finden die beschriebenen Prozeduren alle im Modul **mdlCrypt** der Beispieldatenbank. Sie demonstrieren im Prinzip lediglich die Vorgehensweise bei der Verschlüsselung von Texten. Sie müssen sich jedoch keine Gedanken um eine eigene wasserdichte Implementierung machen, da derartige Routinen zuhauf im Netz zu finden sind.

Als Standardverfahren hat sich etwa der **RC4**-Algorithmus bewährt. Die Kodierung benötigt neben dem Eingabe-String noch einen Schlüssel, der dann bei der Dekodierung ebenso verwendet wird. Die Funktion **RC4** im Modul **mdlRC4**, welche wir hier nicht abbilden, erzeugt über dieses Paar einen Ergebnis-String:

```
globkey = "akrakadabra"
GenerateKey
S = RC4("Access Basics")
Debug.Print S, "/", RC4(S)
->  üÁ6-´ÑÚ_¥Ó  / Access Basics
```

Die globale Variable **globKey** nimmt den Schlüssel auf. Dieser kann von der Prozedur **RC4** allerdings nicht direkt verwendet werden, sondern muss erst über einen Aufruf der Hilfsroutine **GenerateKey** in ein Byte-Array überführt werden.

Einmal erzeugt, arbeitet die Routine forthin mit diesem Schlüssel-Array. Das Schöne an der Prozedur

ist, dass sie reversibel arbeitet und Kodierung, wie Dekodierung, gleichermaßen erledigt. Sie arbeitet hinreichend schnell, so dass sie auch für größere Datenmengen einzusetzen ist.

Wer an maximaler Performance interessiert ist, der kommt um **API**-Funktionen nicht herum. Windows enthält solche Verschlüsselungsfunktionen in der Bibliothek **advapi32.dll**. Das aufwändige Klassenmodul **clsCrypto** der Beispieldatenbank, Ergebnis mehrerer **Visual Basic 6**-Autoren, macht sie sich zunutze. Eine Einsatzmöglichkeit gibt Listing 7 wieder.

Die Funktion **RC42E** dient dem Verschlüsseln des übergebenen Strings, die Funktion **RC42D** der Entschlüsselung. Beide testen zunächst per **Len()**, ob der übergebene String überhaupt gefüllt ist – nur dann müssen die Prozeduren in Aktion treten.

Zweitens checken sie ab, ob die Klassenobjektvariable **cCrypt**, deklariert im Kopf des Moduls, schon initialisiert ist. Falls nicht, so erledigt ein Aufruf der Hilfsprozedur **InitCrypto** dies.

Sie erzeugt eine Instanz der Klasse, setzt das Verschlüsselungspasswort auf **akrakadabra** und weist die Klasse schließlich an, den **RC4**-Algorithmus zu verwenden. Die Methode **EncryptData** gibt dann den verschlüsselten String zurück, die Methode **DecryptData** den entschlüsselten.

Die Routinen schaffen auf einem aktuelleren Rechner problemlos die Ver- und Entschlüsselung von etwa Hunderttausend Strings pro Sekunde. Das hängt natürlich auch von der Länge der übergebenen Texte ab. Wir verwenden Sie deshalb für die weiteren Schritte unserer Aufgabe, die Kundentabelle zu kodieren.

Sicherheitshinweis: Das Passwort **akrakadabra** ist im Modul hartkodiert, was dem Hacker das Leben leicht macht. Im produktiven Einsatz sollte dies verschleiert werden. Sie könnten etwa zu so einem Trick greifen:

```
cCrypt.Password = StrReverse("karba") & _
   StrReverse("arbada")
```

```
Private cCrypt As clsCrypto

Public Function RC42E(S As Variant) As String
    If Len(Nz(S)) = 0 Then Exit Function
    If cCrypt Is Nothing Then InitCrypto
    RC42E = cCrypt.EncryptData(S)
End Function

Public Function RC42D(S As Variant) As String
    If Len(Nz(S)) = 0 Then Exit Function
    If cCrypt Is Nothing Then InitCrypto
    RC42D = cCrypt.DecryptData(S)
End Function

Private Sub InitCrypto()
    Set cCrypt = New clsCrypto
    cCrypt.Password = "abrakadabra"
    cCrypt.EncryptionAlgorithm = EC_CRYPT_ALGO_ID.RC4
End Sub
```

Listing 7: Ver- und Entschlüsselung mithilfe der Klasse **clsCrypto**

Sinnvoll ist es natürlich überdies, das VBA-Projekt mit einem Passwort zu sichern, damit es nicht eingesehen werden kann, oder die Datenbank gleich in das MDE- oder ACCDE-Format zu überführen.

Eine Tabelle RC4-verschlüsseln

Erzeugen Sie zunächst eine Kopie der Tabelle **tblKunden** inklusive der enthaltenen Datensätze und benennen sie **tblKundenEncrypt**. Damit stellen Sie sicher, dass die Daten nicht hinüber sind, falls beim Verschlüsseln etwas schief laufen sollte.

Die duplizierte Tabelle ist nun die Basis für die folgenden Schritte. Legen Sie eine neue Abfrage an und fügen Sie ihr die neue Tabelle ein. Stellen Sie als Modus eine **Aktualisierungsabfrage** ein. Fügen Sie nun dem Entwurf alle Felder hinzu, die verschlüsselt werden sollen.

In die Zeile **Aktualisieren** geben Sie dann jeweils den **RC42E**-bearbeiteten Feldinhalt an (siehe Bild 4). Nach

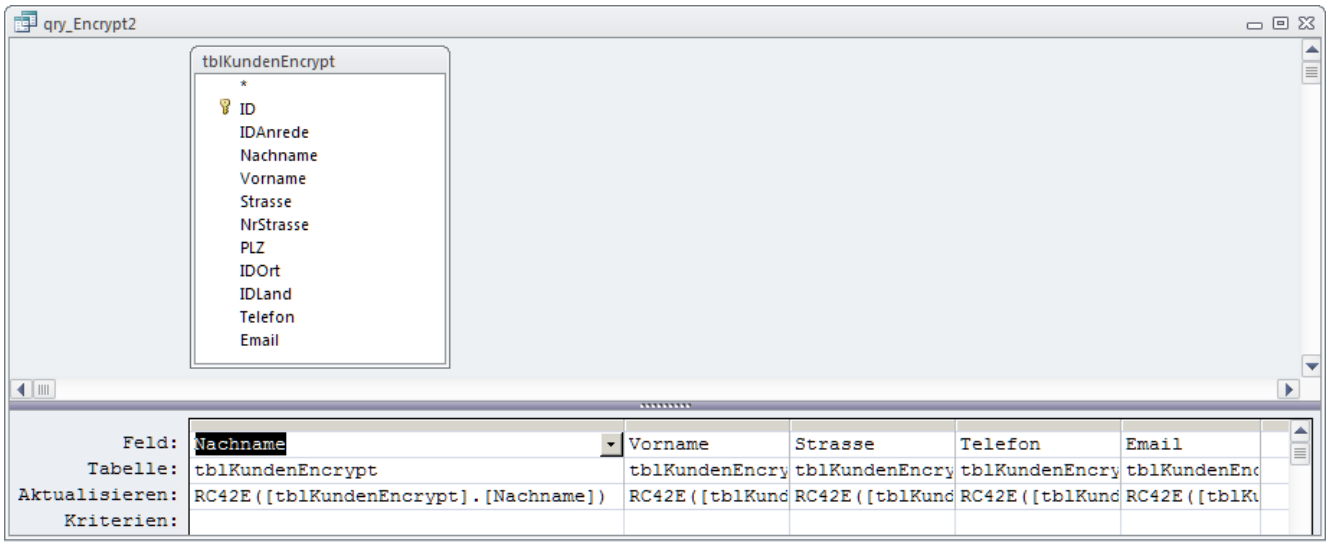


Bild 4: Diese Aktualisierungsabfrage verschlüsselt die Textinhalte der Kundentabelle **tblKundenEncrypt** über den RC4-Algorithmus

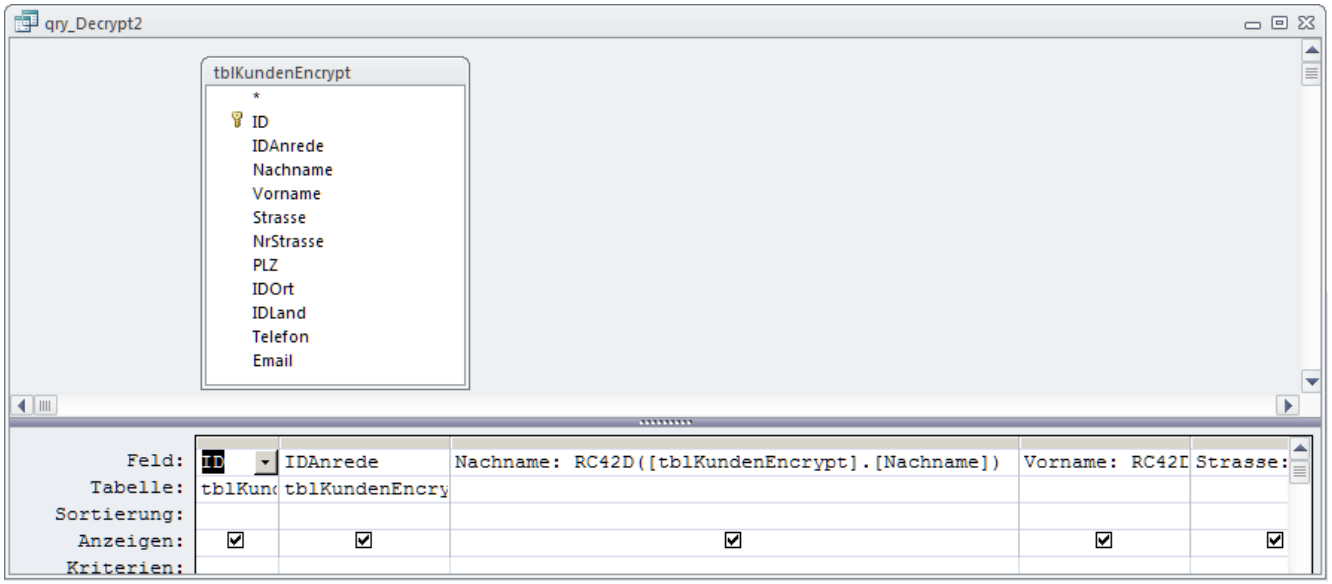


Bild 5: Und diese Auswahlabfrage entschlüsselt die Textinhalte der Tabelle **tblKundenEncrypt** wieder über RC4

dem Ausführen der Abfrage sind die Textinhalte dann so verschlüsselt, wie schon weiter oben gezeigt.

Damit aus diesen Daten wieder sinnvolle Angaben entstehen, benötigen Sie eine weitere Abfrage, die die Verschlüsselung rückgängig macht. Die Auswahlabfrage in Bild 5 unterzieht etwa den Nachnamen der Tabelle der Funktion **RC42D** und gibt ihn wiederum über den Feld-**Alias Nachname** aus. Dies ist auch für alle weiteren verschlüsselten Felder vorzunehmen. Die restlichen Tabellenfelder können Sie dem Ergebnis hingegen auf die übliche Weise hinzufügen.

Nach dem Start der Abfrage präsentieren sich alle Datensätze wieder in lesbarer Form. Allerdings können in dieser Datenblattansicht die Datensätze nicht aktualisiert werden, weil die RC42-Funktionen dies verhindern. Sie ist damit schreibgeschützt.

Verschlüsselte Daten im Formular

Verschlüsselte Tabellen lassen sich in einem Formular leider nicht so leicht bearbeiten, wie unverschlüsselte. Wenn Sie lediglich eine Ausgabe der Daten benötigen, so können Sie die vorige Abfrage direkt als Datenquelle für das Formular verwenden.

Sobald sie jedoch bearbeitbar sein sollen, ist ein anderes Vorgehen unumgänglich.

Das Formular **frmAdressen** der Beispieldatenbank (siehe Bild 6) zeigt die Feldinhalte korrekt an.

Klicken Sie auf den Button rechts oben, so zeigen sich die tatsächlich darunterliegenden Daten, wie in Bild 7. Die verschlüsselten Felder können dennoch aktualisiert werden. Das allerdings funktioniert nicht ohne die Zuhilfenahme von VBA.

Sehen Sie sich dazu den Entwurf des Formulars in Bild 8 an. Ein Teil der Felder ist an die dem Formular zugewiesene Datenherkunft (Tabelle **tblKundenEncrypt**) gebunden, die verschlüsselten Felder jedoch nicht.

Sie zeigen hier den Schriftzug **Ungebunden**. Folglich müssen deren Inhalte über Programmierung gesetzt werden. Die geeignete Ereignisprozedur ist für diesen Zweck die **Beim Anzeigen** (**Form_Current**).

Sie wird automatisch aufgerufen, sobald Sie zu einem beliebigen Datensatz navigieren. Listing 8 zeigt die Prozedur. Das Datenherkunftsfeld **Nachname** wird per **RC42D**-Funktion dechiffriert und dem Textfeld **txtNachname** zugewiesen. Dasselbe geschieht für die weiteren verschlüsselten Textfelder.

Wie aber können diese Felder dann aktualisiert werden? Beim Ändern eines Feldinhalts tritt etwa die Ereignisprozedur **Nach Aktualisierung** (**After_Update**)

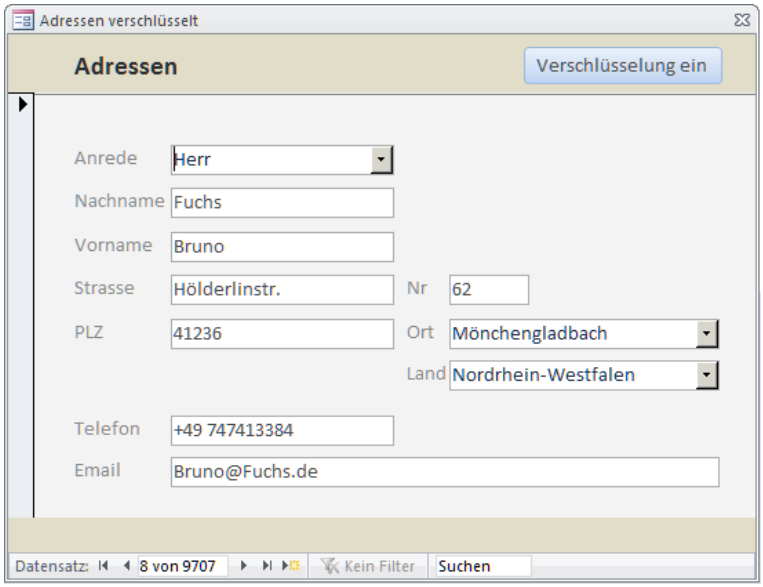


Bild 6: Das Adressformular baut auf der verschlüsselten Kundentabelle auf

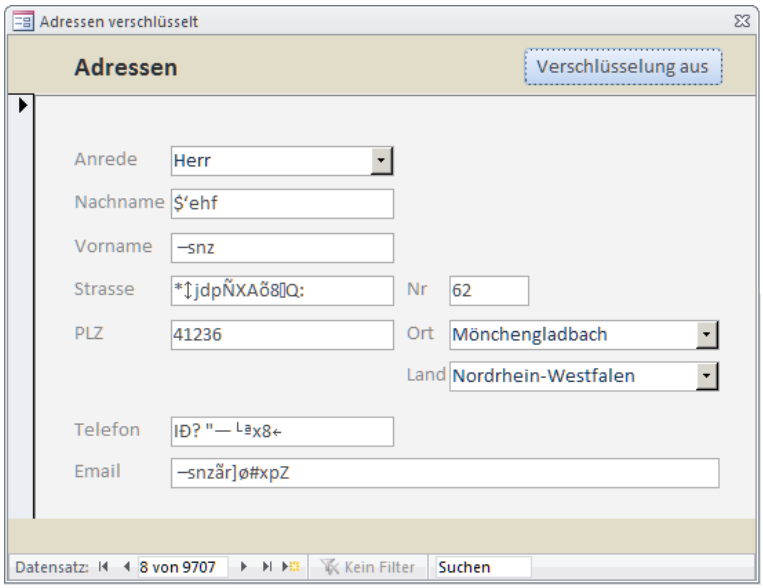


Bild 7: Bei ausgeschalteter Entschlüsselung zeigt sich der wahre Inhalt der Felder

```
Private Sub Form_Current()
    Me!txtNachname = RC42D(Me!Nachname)
    Me!txtVorname = RC42D(Me!Vorname)
    Me!txtStrasse = RC42D(Me!Strasse)
    Me!txtTelefon = RC42D(Me!Telefon)
    Me!txtEmail = RC42D(Me!Email)
End Sub
```

Listing 8: Die Ereignisprozedur **Beim Anzeigen** des Formulars setzt die entschlüsselten Felddaten ein.

ein. Das gilt auch für ungebundene Felder. Sie kann nun benutzt werden, um den eingegebenen Text zu kodieren:

```
Private Sub txtNachname_AfterUpdate()
    Me!Nachname = RC42E(Me!txtNachname)
End Sub

Private Sub txtVorname_AfterUpdate()
    Me!Vorname = RC42E(Me!txtVorname)
End Sub

Private Sub txtStrasse_AfterUpdate()
    Me!Strasse = RC42E(Me!txtStrasse)
End Sub

...
```

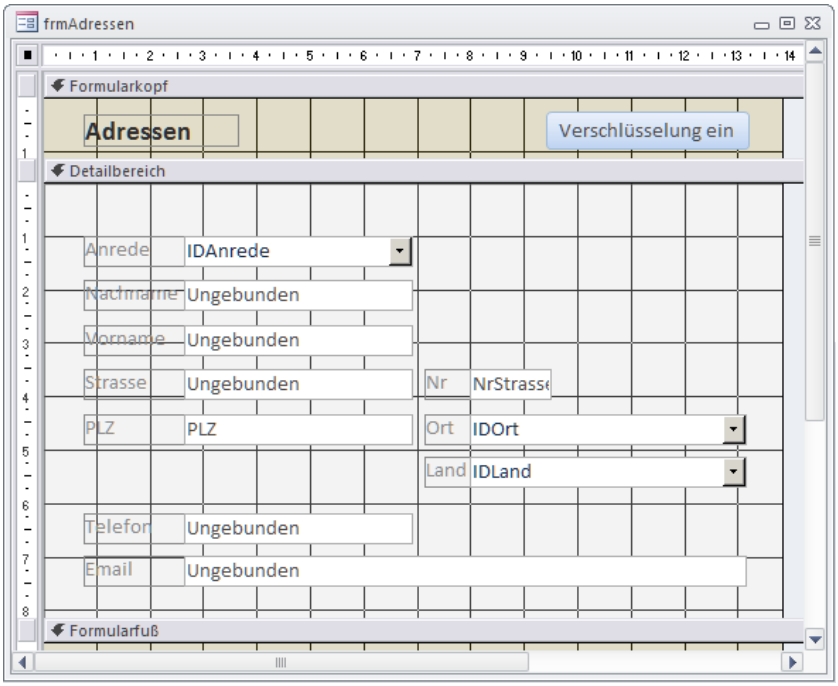


Bild 8: Die Entwurfsansicht des Adressformulars mit seinen ungebundenen Feldern

So landen die verschlüsselten Inhalte wieder in den Tabellenfeldern.

Bliebe noch die Routine, die beim Klick auf den Button **cmdAusEin** hervorgerufen wird (siehe Listing 9). Sie demonstriert nochmal einmal deutlicher, wie die Felder besetzt werden.

Hier wird auch klar, warum Textboxen nie mit dem gleichen Namen versehen werden sollten, wie deren Datenfelder. Bei **Me!Nachname** wüsste man ja sonst nicht, ob das Datenfeld oder die Textbox **Nachname** angesprochen würde. Leider macht Access dies beim automatischen Formularentwurf generell, weshalb dann Nacharbeit angesagt ist.

AllowEdits ist übrigens eine Formulareigenschaft, die einstellt, ob das Bearbeiten von Datensätzen erlaubt sei.

Steht sie auf **False**, so sind alle Eingabesteuerelemente gesperrt. Im unentschlüsselten Modus des Formulars, der hier nur zu Demonstrationszwecken eingebaut wurde, sollte das besser der Fall sein. Setzen von **True** hingegen bewirkt das augenblickliche Freigeben der Bearbeitbarkeit.

```
Private Sub cmdAusEin_Click()
    If Me!cmdAusEin.Caption = "Verschlüsselung ein" Then
        Me!cmdAusEin.Caption = "Verschlüsselung aus"
        Me!txtNachname = Me!Nachname
        Me!txtVorname = Me!Vorname
        Me!txtStrasse = Me!Strasse
        Me!txtTelefon = Me!Telefon
        Me!txtEmail = Me!Email
        Me.AllowEdits = False
    Else
        Me!cmdAusEin.Caption = "Verschlüsselung ein"
        Me!txtNachname = RC42D(Me!Nachname)
        Me!txtVorname = RC42D(Me!Vorname)
        Me!txtStrasse = RC42D(Me!Strasse)
        Me!txtTelefon = RC42D(Me!Telefon)
        Me!txtEmail = RC42D(Me!Email)
        Me.AllowEdits = True
    End If
End Sub
```

Listing 9: Umschalten zwischen ver- und entschlüsselter Feldansicht über Button