BackupAes256

Benutzerhandbuch

Ein Programm für

1. unverschlüsselte Sicherungskopien,
2. symmetrisch verschlüsselte Sicherungskopien und
3. hybrid verschlüsselte Datenübertragung.

Diese Software soll die Grundlage für eine dezentrale und quelloffene Telematikinfrastruktur im deutschen Gesundheitswesen sein.

Oder ein Hilfsmittel, um im Praxisalltag Daten zu sichern.

Das ist das Handbuch zu Programmversion 1.0.0 vom 27.11.2019.

Noch sind nicht alle Funktionen eingerichtet, beispielsweise wird die Fortschrittsanzeige erst mit der nächsten Programmaktualisierung funktionieren. Die Verschlüsselung ist schon getestet und kann verwendet werden. Garantie oder Haftung übernehme ich allerdings nicht.

Aktualisierungen sind unter der Adresse https://github.com/dasSubjekt/BackupAes256 in ca. wöchentlichen Abständen geplant.

Kontakt: praxis@matthias-gloeckner.de
https://twitter.com/dasSubjekt

Inhaltsverzeichnis

1. Einleitung

»Einen Brunnen graben just an der Stelle, wo man gerade steht.«

Mit diesem Satz bezieht der Schriftsteller Theodor Fontane (1819 bis 1898) sich in seinem Roman »Der Stechlin« auf die zu jener Zeit populäre »Wasserkur« des Pfarrers Sebastian Kneipp. Seine Romanfigur des Pastor Lorenzen lässt er anfügen: »Nicht so ganz unbedingt mit dem Neuen. Lieber mit dem Alten, soweit es irgend geht, und mit dem Neuen nur, soweit es muss.« Die Kneipp-Kur steht heute im bundesweiten Verzeichnis des immateriellen Kulturerbes. Populär sind dagegen andere, vor allem neuere, Heilmethoden. Also doch Altes zum Kulturerbe und dann lieber mit dem Neuen?

Das »soweit es muss« spricht Fontane auch mit fragendem Blick auf die zu jener Zeit sich entwickelnde Sozialdemokratie aus. Heute ist »soweit es muss« als ein Standpunkt in Diskussionen über die Digitalisierung zu hören. Aus »Wasser auf die Mühlen der Sozialdemokratie« am Ende des 19. Jahrhunderts scheint »Wasser auf die Mühlen der Digitalisierung« am Beginn des 21. Jahrhunderts geworden zu sein.

Wasser-Metaphern funktionieren noch immer. So haben symmetrische kryptographische Schlüssel weniger Bits als asymmetrische kryptographische Schlüssel. Der Grund dafür ist, dass erstere dicht gepackt sind wie Wasser bei 4 °C, letztere sind der Pulverschnee[1] der Primzahlen in der Luft aller teilbaren Zahlen. Bit kann wie Liter entweder die Maßeinheit für eine konkrete Menge sein oder das »Hohlmaß« für die maximal mögliche Menge.

Mit meiner Software BackupAes256 will ich der Diskussion um die Digitalisierung im deutschen Gesundheitswesen und speziell um die Telematikinfrastruktur[2] ein Maß an neuen Informationen hinzufügen. Falls es stimmt, dass Patientin und Patient Eigentümer ihrer Daten bleiben sollen, weshalb ist dann die Speicherung kryptographischer Schlüssel und der mit ihnen kodierten Daten in einer zentralisierten Telematikinfrastruktur besser als die dezentrale Lösung, die ich hier vorstelle?

Bereits im Jahr 2007 schlug der Hausarzt Wilfried Deiß im Deutschen Ärzteblatt[3] eine Punkt-zu-Punkt-Kommunikation aller Beteiligten des Gesundheitswesens über ein sicheres Netz vor. Mit meinem Projekt biete ich eine konkrete, praxistaugliche und, wie ebenfalls von ihm gefordert, »schlanke Lösung« zu dieser Idee an.

1 https://de.wikipedia.org/wiki/Ulam-Spirale

2 https://de.wikipedia.org/wiki/Telematikinfrastruktur

3 https://www.aerzteblatt.de/archiv/57836/Projekt-Elektronische-Gesundheitskarte-Fuchs-statt-Monster

Zugleich möchte ich angesichts intensiver Forschung an Quantencomputern die Diskussion darüber fördern, über welchen Zeitraum hinweg ein »sicheres Netz« noch als sicher gelten kann. Vor 122 Jahren wurde »Der Stechlin« geschrieben. Vermutlich wesentlich früher als in 122 Jahren werden Quantencomputer alles, was wir mit den heutigen asymmetrischen bzw. hybriden Verfahren verschlüsseln und durch das Internet schicken, auch für Unbefugte lesbar machen können.[4]

Dabei darf Datensicherheit nicht vom Besitz der größeren technischen oder intellektuellen Fähigkeiten abhängen, nach dem Motto: wer das Rätsel löst, dem gehören die Daten. Datensicherheit soll auf dem Besitz eines Geheimnisses beruhen, das auch mit großen intellektuellen und technischen Möglichkeiten praktisch nicht erraten werden kann.

So halte ich es für unabdingbar, dass Patientin und Patient die kryptographischen Schlüssel zum sicheren Speichern ihrer Daten selbst erzeugen und selbst verwalten können. Höchstens öffentliche kryptographische Schlüssel gehören in die Hände einer zentralen Stelle, die ihre Veröffentlichung erleichtert.

Wer dann immer noch die Hoheit über seine Gesundheitsdaten in die gesetzliche Betreuung der Telematikinfrastruktur übergeben wollte, könnte es durch freiwilliges Übermitteln seiner selbst erzeugten privaten kryptographischen Schlüssel an den »großen Bruder« tun.

Anders als die Telematikinfrastruktur ist meine Software quelloffen, so dass alle Interessierten ihre Funktionen nachvollziehen, auf Fehler hinweisen, Verbesserungen vorschlagen und eigene Versionen davon ableiten können. Manche der hier vorgestellten Ideen werden sich als alt und vielleicht auch als bewährt erweisen.

BackupAes256 fertigt Kopien von Verzeichnissen und Dateien auf drei verschiedene Arten an. Die Software

1. erstellt und aktualisiert unverschlüsselte Sicherungskopien,

2. erstellt und aktualisiert mit einem geheimen Schlüssel symmetrisch verschlüsselte Sicherungskopien und

3. erstellt hybrid verschlüsselte Kopien, das heißt symmetrisch verschlüsselte Daten mit asymmetrisch verschlüsseltem Schlüssel, zur Übermittlung an ausgewählte Empfänger über das Internet.

Ich hoffe, dass die Punkte 1. und 2. den Praxisalltag ein wenig erleichtern können. Mit Punkt 3. will ich mehr ein Nachdenken über das beginnende Zeitalter der Unsicherheit asymmetrischer und hybrider Verschlüsselungsverfahren anregen, als dessen tatsächliche Verwendung.

---

4 https://www.aerzteblatt.de/nachrichten/99034/Langfristige-sichere-Speicherung-von-Gesundheitsdaten-laut-IT-Experten-im-Augenblick-nicht-gewaehrleistet

So habe ich den ursprünglichen Namen meines Programms beibehalten und es nicht in »Andere Elektronische Gesundheitsakte (AEG)« oder in etwas mit »verschlüsselte Datenübertragung« und »sicheres Netz« umbenannt. Man soll meiner Software weiterhin ansehen, dass ich vor der Psychologie zwei Jahre lang Informatik studiert habe, dass ich dieses Projekt zur Vereinfachung der Datensicherung in meiner psychotherapeutischen Praxis begonnen habe und dass ich ein Befürworter der Digitalisierung nur insoweit bin, wie sie auf naturwissenschaftlicher Bildung und Einsicht beruht.

Ich würde mich freuen, wenn mein Programm zu den Zwecken der Datensicherung und der Einsicht in die Methoden der Verschlüsselung nun auch von Kolleginnen und Kollegen verwendet würde.

Was die Software nicht kann, ist

1. Daten und kryptographische Schlüssel gegen Viren, Trojaner und Hintertüren des Betriebssystems auf den ver- und entschlüsselnden Computern schützen,

2. Garantien oder Haftungsansprüche gegen mich begründen und

3. selbst Viren, Trojaner oder Hintertüren enthalten.

Verschlüsselung bedeutet nicht, dass die vertraulichen Schlüssel nicht aus dem Computer oder aus einem unsicheren Aufbewahrungsort entwendet werden können. Das muss nicht durch Trojaner geschehen. Auch von Windows weiß niemand, was es tut oder in Zukunft tun wird.

BackupAes256 enthält die Ver- und Entschlüsselungsalgorithmen nicht selbst, sondern nutzt die Kryptographie-Schnittstelle »Next Generation« (CNG) von Windows. Das ist eine Weiterentwicklung derjenigen Schnittstelle, die vor 20 Jahren im Verdacht stand, einen Zweitschlüssel der National Security Agency der Vereinigten Staaten (NSA) zu enthalten[5]. Ob der NSA-Schlüssel mit weiterentwickelt wurde, ist nicht bekannt.

Mein Programm selbst kann keine Viren enthalten, da ich nur die Textdateien mit dem Quellkode zur Verfügung stelle. Wie daraus ein ausführbares Programm wird, steht in Kapitel 2 dieser Anleitung.

5 https://de.wikipedia.org/wiki/NSAKEY

Beim Schreiben der Software und dieser Anleitung ist mir manches deutlicher geworden. Die wichtigsten Punkte sind:

1. Die Digitalisierung verlangt von Praxisinhaber*innen ein großes Wissen über Informationstechnik und Datenschutz, das sie sich teils selbst aneignen, teils von externen Berater*innen einkaufen müssen. Unter dieser Voraussetzung kann ihnen auch die Bedienung des vorliegenden Programms zugetraut werden.

2. Digitales Grundwissen gehört heute zur Allgemeinbildung. Wer irgendeine Art gesellschaftlicher Verantwortung trägt, sollte mit Kerckhoffs' Prinzip[6] und mit den Unterschieden zwischen symmetrischer, asymmetrischer und hybrider Verschlüsselung vertraut sein.

3. Asymmetrische Verschlüsselung ist schon jetzt leichter zu brechen als symmetrische Verschlüsselung. Mit fortschreitender Entwicklung von Quantencomputern wird sich dieser Trend beschleunigen. Das macht es immer weniger angemessen, beide Verfahren in demselben Begriff von »Datensicherheit« zusammenzufassen.

4. Wer Gesundheitsdaten nicht in die Hände US-amerikanischer Konzerne geraten lassen will, sollte ein quelloffenes und freies Betriebssystem wie Linux nutzen.

Im Verlauf von »Der Stechlin« verändert sich gemäß dessen langsamer, aber fließender Natur die Einstellung von Pastor Lorenzen gegenüber Alt und Neu. In dem zentralen Dialog bei zwei Dritteln des Buchs sagt er:

»Ich habe mich dagegen gewehrt, als das Eis aufgeschlagen werden sollte, denn alles Eingreifen oder auch nur Einblicken in das, was sich verbirgt, erschreckt mich. Ich respektiere das Gegebene. Daneben aber freilich auch das Werdende, denn eben dieses Werdende wird über kurz oder lang abermals ein Gegebenes sein. Alles Alte, soweit es Anspruch darauf hat, sollen wir lieben, aber für das Neue sollen wir recht eigentlich leben.«

6 https://de.wikipedia.org/wiki/Kerckhoffs%E2%80%99_Prinzip

2. Das Programm zum Laufen bringen

2.1. Warum so kompliziert?

Als Benutzer des Betriebssystems Windows sind wir es gewohnt, eine Installationsdatei herunter zu laden, sie mit Doppelklick zu starten und die Nachfrage, ob wir wirklich wissen, was wir tun, mit dem gewohnten Klick auf »ja« zu beantworten. Für das Programm BackupAes256 gelten andere Regeln.

Wollte man wirklich nachvollziehen, was so eine Installationsdatei und das von ihr installierte Programm auf unserem Computer tun, müsste man großen technischen Aufwand betreiben. Funktioniert das Programm tatsächlich so, wie seine Dokumentation es beschreibt? Schickt es Daten irgendwo hin, ohne mich zu informieren? Enthält es einen Computervirus, den der Virenscanner nicht erkennt? Für diese Zwecke gibt es spezialisierte Computerprogramme, sogenannte Disassembler und Decompiler. Jedoch braucht man zu ihrer Benutzung eine fortgeschrittene Programmiererausbildung. Außerdem hat die Anwendung solcher Programme auf kommerzielle Software rechtliche Hürden.

Um Transparenz und Sicherheit den Vorrang zu geben, habe ich für mein Programm eine andere Entscheidung getroffen. Das macht es wesentlich einfacher, die eben genannten Fragen zu beantworten, aber etwas komplizierter und langwieriger, das Programm zum ersten Mal auf dem eigenen Computer zum Laufen zu bringen. Als willkommenen Nebeneffekt dieser Umstände können auch Benutzer*innen ohne Programmierkenntnisse einige Programmfunktionen selbst anpassen. Beispielsweise, ob der Wert symmetrischer kryptographischer Schlüssel durch Benutzer nachträglich geändert werden kann oder ob die Benutzeroberfläche auf Deutsch oder Englisch beschriftet ist.

Aus den Begriffen Disassembler und Decompiler lässt sich erahnen, dass es sich um die Gegenteile von Assemblern und Compilern handelt. Assembler sind sehr nah an der Maschinensprache des Computerprozessors angesiedelt und wandeln eine so genannte Assemblersprache in Prozessorbefehle um. Mit jedem Jahrzehnt der Computer- und Softwareentwicklung grenzte sich ihre Verwendung immer weiter auf Spezialbereiche ein und sie wurden durch Compiler für höhere Programmiersprachen ersetzt. Wer BackupAes256 nutzen möchte, braucht also einen Compiler, und zwar einen für die Programmiersprache C# (gesprochen: see sharp). Anstelle einer Installationsdatei muss der Quelltext des Programms aus dem Internet heruntergeladen werden. Der Compiler übersetzt diesen dann in Prozessorbefehle.

Bis vor einigen Jahren war das alles ungefähr so kompliziert, wie es klingt. Dank Microsoft (das muss hier doch einmal lobend gesagt werden) ist es einfacher geworden.

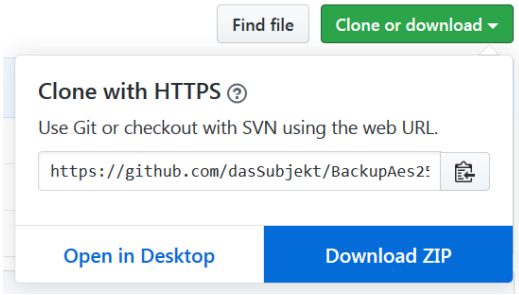
2.2. Herunterladen des Quellkode

Linus Torvalds ist der Ideen- und Namensgeber des Betriebssystems Linux. Er ist jedoch auch Ideen- und Namensgeber einer Versionsverwaltung für Software namens Git. Zumindest scherzte er, dass Git (Englisch für Depp) nach ihm benannt sei.

Ein Unternehmen in San Francisco entwickelte Git zu einem Speicherdienst für Softwareprojekte namens Github weiter. Versuchsweise, aber vielleicht auch dauerhaft, steht BackupAes256 dort zum Herunterladen bereit. Die Adresse ist

https://github.com/dasSubjekt/BackupAes256

Mit einem Klick auf »Clone or download« und einem weiteren auf »Download ZIP« können alle benötigten Dateien in Form eines Zip-Archivs heruntergeladen werden. Anschließend lässt sich dieses mit der rechten Maustaste anklicken und auf der Festplatte entpacken.

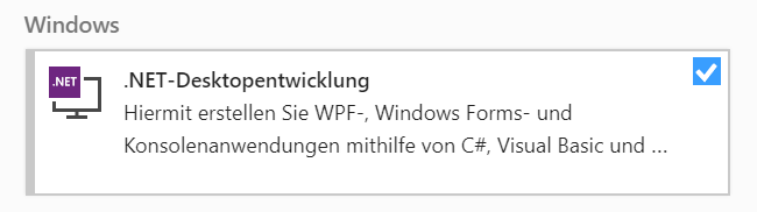


2.3. Herunterladen von Visual Studio Community

Um den Quellkode auf übersichtliche Weise anzuzeigen, bei Bedarf darin Änderungen vorzunehmen und schließlich ein lauffähiges Programm daraus zu erzeugen, wird eine sogenannte Integrierte Entwicklungsumgebung benötigt. Die Abkürzung dafür ist IDE, von englisch Integrated Development Environment.

In diesem Fall ist das Visual Studio Community. Diese Entwicklungsumgebung kann von der Adresse https://visualstudio.microsoft.com/de/vs/community/ heruntergeladen werden. Da es sich tatsächlich um eine »Umgebung« handelt, nicht nur um ein Programm, kann die Installation einige Stunden dauern. Manchmal kommt es dabei zu kleineren Fehlern, die ignoriert werden können.

Nicht alle der zahlreichen Bestandteile müssen installiert werden. Benötigt wird die ».NET-Desktopentwicklung«.



Soll die voraussichtlich ab Januar 2020 verfügbare Linux-Version von BackupAes256 Verwendung finden, kann schon jetzt zusätzlich »Plattformübergreifende .NET Core-Entwicklung« angekreuzt werden. Dieser Schritt lässt sich jedoch auch später noch nachholen.

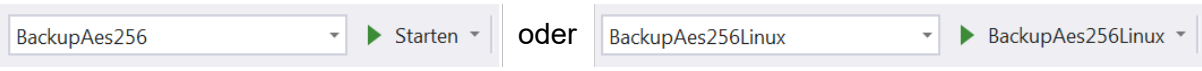
Wer Visual Studio Community länger als 30 Tage nutzen möchte, muss sich mit einer E-Mail-Adresse bei Microsoft registrieren. Microsoft hält sich bedeckt, welche Nutzerdaten es außerdem aus Visual Studio ausliest und in den Vereinigten Daten oder an dritten Orten speichert. Aus beiden Gründen empfiehlt es sich, zeitnah die ausführbare Datei zu erzeugen.

Wenn nicht nur das Programm ausprobiert, sondern mit vertraulichen Daten gearbeitet werden soll, ist es sicherer, die ausführbare Datei BackupAes256.exe immer direkt zu starten, nicht aus Visual Studio heraus.

2.4. Ein lauffähiges Programm erzeugen

Um das Projekt in Visual Studio Community zu öffnen, kann entweder in dem Verzeichnis mit dem entpackten Quellkode die Datei BackupAes256.sln gesucht und mit einem Doppelklick geöffnet werden. Oder man startet Visual Studio zuerst und wählt darin »Projekt oder Projektmappe öffnen«, dann BackupAes256.sln.

Anschließend wählt man die zu startende Version und klickt auf den grünen Pfeil.



In derselben Zeile sollte »Release« ausgewählt sein, um ein von Visual Studio unabhängiges Programm zu erzeugen. »Debug« dient dem Testen beim Programmieren und erzeugt eine größere, nicht optimierte Programmdatei.

Unter Projekt => Build => Ausgabe => Ausgabepfad ist der relative Speicherort innerhalb des Quellkode-Verzeichnisses zu finden, an dem jetzt die ausführbare Datei BackupAes256.exe erzeugt wurde. Innerhalb des Quellkode-Verzeichnis ist das in der Regel der Pfad »\BackupAes256\BackupAes256\bin\x64\Release«. Es genügt, diese ausführbare Datei und das Unterverzeichnis »Assets« mit seinem Inhalt an einen beliebigen Ort zu kopieren. Dort kann das Programm nun immer direkt gestartet werden, Visual Studio wird nicht mehr benötigt.

Zur Ausführung unter Windows benötigt BackupAes256 das NetFramework 4.7.2 oder höher. Falls es nicht schon auf dem Computer vorhanden war, wird es automatisch als ein Teil von Visual Studio Community installiert. Um BackupAes256 auf weiteren Windows-Computern ausführen zu können, muss zuvor gegebenenfalls das NetFramework 4.7.2 von der entsprechenden Microsoft-Seite aus der Spalte »Runtime«[7] heruntergeladen und auf dem Computer installiert werden. Fehlt es, erscheint beim Versuch des Programmstarts der entsprechende Hinweis.

Wer ein anderes Programmsymbol als das voreingestellte wünscht, kann eine Symboldatei z.B. von http://www.iconarchive.com/tag/password herunterladen und in Visual Studio Community unter Projekt => BackupAes256-Eigenschaften => Anwendung => Symbol festlegen. Die Festlegung für das Programmfenster erfolgt in der Datei »MainWindow.xaml«.

7 https://dotnet.microsoft.com/download/visual-studio-sdks

Als Sprache der Benutzeroberfläche kann entweder Deutsch oder Englisch gewählt werden. Dazu muss unter Projekt => BackupAes256-Eigenschaften => Build => Symbole für bedingte Kompilierung als Schlüsselwort entweder DEUTSCH oder ENGLISH eingetragen werden.

Die Linux-Version in derselben Projektmappe wie die Windows-Version zu verwalten, hat sich nicht bewährt. Da die Kryptographieplattform CGN unter Linux nicht zur Verfügung steht, wird zum Teil anderer Programmkode benötigt. Außerdem führte das Verwenden gemeinsamen Quellkodes dazu, dass das Erstellen der unter Linux ausführbaren Datei immer erst beim zweiten Versuch fehlerfrei lief. Ab Januar 2020 wird es deshalb für den Linux-Teil eine eigene Projektmappe zum Herunterladen geben. Dort wird der komplette Inhalt des Verzeichnisses »publish« benötigt, der nach Linux kopiert werden muss. Unter Linux muss zudem die .NET Core Laufzeitumgebung[8] installiert werden. Anders als in der Anleitung angegeben, wird die Version 2.1 benötigt. In dem entsprechenden Befehl muss also diese Versionsnummer verwendet werden. Getestet habe ich das Programm unter Linux Mint[9] 19.2 Xfce 64-bit. Das ist zugleich die Version, die ich für einen Einstieg in die Linux-Welt empfehle.

Ein Nachteil für die Nutzung unter Linux ist, dass dazu der Quelltext zuvor auf einem Windows-Rechner übersetzt werden muss. Vielleicht schafft hier jemand Abhilfe, der oder die sich besser mit Linux-Programmierung auskennt als ich?

2.5. Lizenzen

Die Lizenz für Visual Studio Community[10] räumt Nutzern weitreichende Rechte ein. Dafür »zahlen« diese möglicherweise mit nicht genau bezeichneten Nutzerdaten.

Die Linux-Version von BackupAes256 benötigt die Software-Plattform .NET Core[11] und die Benutzeroberfläche AvaloniaUI[12]. Beide stehen unter der MIT-Lizenz[13], die das Nutzen, Verändern, Weiterverbreiten und Verkaufen der so lizenzierten Software erlaubt, Garantie und Haftung jedoch ausschließt.

Ein Teil der in BackupAes256 verwendeten Bilder und Symbole stammt aus der Sammlung »My Secret Icons«[14] der Designerin https://artdesigner.me, der andere Teil aus der Sammlung »Dock Icons«[15] des Designers cemagraphics. Beide Autoren erlauben die nichtkommerzielle und kommerzielle Nutzung ihrer Werke unter der Bedingung einer legalen Verwendung und der Verlinkung auf sie als Urheber.

---

8 https://dotnet.microsoft.com/download/linux-package-manager/ubuntu18-04/runtime-current

9 https://linuxmint.com/download.php

10 https://visualstudio.microsoft.com/de/license-terms/mlt031819/

11 https://github.com/dotnet/core

12 https://github.com/AvaloniaUI/Avalonia

13 https://de.wikipedia.org/wiki/MIT-Lizenz

14 http://www.iconarchive.com/show/my-secret-icons-by-artdesigner.html

15 https://www.deviantart.com/cemagraphics/gallery/7888035/dock-icons

Sowohl der Advanced Encryption Standard als auch das RSA-Kryptosystem sind lizenzfrei. Das Patent für den RSA-Algorithmus erlosch am 21. September 2000. Beim AES-Algorithmus war die Freiheit von patentrechtlichen Ansprüchen eine der Bedingungen des Ausschreibungsverfahrens.

Programmkode habe ich für die Darstellung kryptographischer Schlüssel in Dezimalschreibweise aus dem Projekt InfInt[16] übernommen und für grundlegende Funktionen des ViewModel aus dem Projekt mvvmlight[17]. Auch letzteres Projekt verwendet die MIT-Lizenz.

Diesem Muster bin ich gefolgt und habe BackupAes256 unter die MIT-Lizenz gestellt.

3. Das Programm bedienen

3.1. Aufbau der Benutzeroberfläche

Am linken Rand des Programmfensters sind unter einander vier Register angeordnet: »Aufgabe«, »Fortschritt«, »Nachrichten« und »Schlüssel«. Jedes dieser Register kann per Mausklick auf das entsprechende Symbol in den Vordergrund geholt werden.

Unter »Aufgabe« lassen sich Quelle und Ziel der Synchronisierung bzw. Verschlüsselung einstellen, die zu verwendenden Schlüssel auswählen und anschließend die Verarbeitung starten. Je nach Umfang der zu kopierenden und zu verschlüsselnden Daten kann die Laufzeit des Programms variieren. Nur die Bearbeitung ganzer Verzeichnisse ist möglich. Die Auswahl einzelner Dateien innerhalb eines Verzeichnisses ist nicht vorgesehen.

Quell- und Zielverzeichnis können entweder über die Schaltflächen »auswählen« ausgewählt oder mit Ziehen und Ablegen aus dem Windows-Explorer übernommen werden. Sollte beim Ziehen und Ablegen das Register »Aufgabe« nicht im Vordergrund sein, kann es aktiviert werden, indem man mit der Maus über das entsprechende Symbol zieht.

Vorbehaltlich weiterer Testung wird es nicht möglich sein, in BackupAes256 einzelne Dateien zu verarbeiten, die größer als 2 Gigabyte sind. Diese Größe erreichen beispielsweise längere oder hochauflösende Videos und Filme. Das ist eine Begrenzung des Microsoft .NET Framework.

Unter »Fortschritt« wird (ab der nächsten Programmaktualisierung immer) angezeigt (werden), welche Dateien bisher verarbeitet wurden und für welche Dateien die Synchronisierung bzw. Verschlüsselung noch aussteht.

Das Register »Nachrichten« enthält die Versionsnummer des Programms und wird in nachfolgenden Programmversionen ausführlichere Status- und Fehlermeldungen anzeigen.

16 https://github.com/sercantutar/infint

17 https://github.com/lbugnion/mvvmlight

Unter »Schlüssel« werden alle verfügbaren kryptographischen Schlüssel angezeigt. Dieses Register wird in Punkt »3.5. Einlesen, Erzeugen und Speichern von Schlüsseln« beschrieben.

3.2. Unverschlüsselte Synchronisierung

Hierzu muss ein Quellverzeichnis angegeben werden, das als Vorlage für die Synchronisierung dient. Außerdem wird ein Quellverzeichnis benötigt, dessen Inhalt am Ende des Synchronisierungsvorgangs mit dem des Quellverzeichnisses identisch sein soll. Der häufigste Anwendungsfall ist das Aktualisieren einer Sicherungskopie auf einem externen Datenträger. Dazu ermittelt BackupAes256 zunächst die im Quellverzeichnis neu dazu gekommenen Dateien sowie die Dateien mit einem neueren Bearbeitungsdatum und kopiert diese anschließend in das Zielverzeichnis.

Anschließend startet ein Klick auf die Schaltfläche »vergleichen« den Vergleich beider Verzeichnisse. Dieser Schritt ändert noch keine Daten. Im Register »Fortschritt« ist zu sehen, welche Dateien synchronisiert werden sollen und ein Klick auf die Schaltfläche »synchronisieren« startet anschließend den Schreibvorgang.

3.3. Symmetrisch verschlüsselte Synchronisierung mit AES-256

Der Inhalt eines Verzeichnisses und seiner Unterverzeichnisse kann symmetrisch verschlüsselt werden, entweder in eine einzelne Datei oder in ein verschlüsseltes Dateisystem.

Im Moment existiert nur die Verschlüsselung in eine Datei, die als Vorgabe mit der Erweiterung .aes gespeichert wird. Diese Datei lässt sich als verschlüsselte Sicherungskopie für wenige Daten nutzen. Zugleich kann sie als »elektronische Patientenakte auf einem USB-Stick« verstanden werden. In dem Fall müssen zwischen den Beteiligten je ein für die Authentifizierung und ein für die Verschlüsselung zu verwendender geheimer 256-Bit-Schlüssel vereinbart werden. Das sollte bei einem persönlichen Kontakt geschehen, z.B. bei einem Arztbesuch zwischen Versichertem und Arzt / Ärztin. Die damit verschlüsselte Datei kann anschließend über das Internet versandt werden oder, z.B. durch Verlust des Datenträgers, in die Hände Dritter geraten. Nur die beiden Schlüssel müssen geheim bleiben.

Das verschlüsselte Dateisystem wird im Verlauf des Dezember 2019 zu Verfügung stehen. Es wird viele Gigabyte an Daten aufnehmen können, da nur neue und veränderte Daten neu geschrieben werden. Es wird aus mehreren Dateien bestehen und ist nur für das Anlegen von verschlüsselten Sicherungskopien gedacht, nicht für die Kommunikation mit Dritten.

3.4. Hybride Verschlüsselung mit AES-256 und RSA-4096 bis RSA-16384

Der Inhalt eines Verzeichnisses und seiner Unterverzeichnisse kann in einer einzelnen Datei hybrid verschlüsselt werden. Als Vorgabe wird diese Datei mit der Erweiterung .hyb gespeichert. Sie ist ausschließlich als »elektronische Patientenakte« gedacht, nicht für Sicherungskopien. Wie in Abschnitt »4.7. Eine kurze Geschichte der hybriden Verschlüsselung«

beschrieben, können bei hybrider Verschlüsselung die Schlüssel auf elektronischem Weg über das Internet vereinbart werden, ohne persönlichen Kontakt der Beteiligten.

Bei hybrider Verschlüsselung der einfachsten Art kann nur der Empfänger die verschlüsselten Daten lesen, nicht einmal mehr deren Absender. Um es dem Absender zu ermöglichen, den Inhalt einer hybrid verschlüsselten Datei nachträglich zu überprüfen, verschlüsselt BackupAes256 einmal mit dem Schlüssel des Empfängers und einmal mit dem Schlüssel des Absenders. Das macht die verschlüsselte Datei doppelt so groß wie die Ursprungsdaten. In künftigen Programmversionen wird die Wahl zwischen beiden Möglichkeiten bestehen.

Wer asymmetrische Schlüssel mit anderen Längen als den vorgegebenen erzeugen möchte, kann mittels Visual Studio Community im Quellkode der Datei »MainViewModel_methods.cs« nach »_blNewKeyTypes = new BindingList<Property>« suchen, die dortige Liste ergänzen und das ausführbare Programm anschließend neu erstellen. Empfänger derart verschlüsselter Daten müssen nichts ändern und können sie in gleicher Weise entschlüsseln wie bei üblichen Schlüssellängen.

3.5. Einlesen, Erzeugen und Speichern von Schlüsseln

BackupAes256 kann symmetrische 256-Bit-Schlüssel zweier anderer Programme einlesen und verwenden. Das sind zum einen Startschlüssel der Windows-Laufwerksverschlüsselung BitLocker mit der Dateiendung .BEK und zum anderen auf .key endende Schlüsseldateien, die mit dem Passwortverwaltungsprogramm KeePass2[18] erstellt wurden.

Innerhalb des Programms BackupAes256 können sowohl symmetrische 256-Bit-Schlüssel erzeugt werden als auch asymmetrische Schlüssel für das RSA-Verfahren mit Längen von 4096, 7680, 8192, 15360 und 16384 Bit.

Zum Speichern symmetrischer Schlüssel wird das KeePass2-Format mit der Endung .key verwendet. So lassen sich mit BackupAes256 erzeugte symmetrische 256-Bit-Schlüssel auch als Schlüsseldateien in KeePass2 verwenden. Dieses Format beruht auf der »Extensible Markup Language«[19] (XML, deutsch »Erweiterbare Auszeichnungssprache«).

Auch die Umsetzung des RSA-Algorithmus innerhalb des Microsoft .NET Framework legt dieses Format zum Speichern von Schlüsseln nahe, indem es über eine Funktion »ToXmlString« verfügt. Hier und bei KeePass2 erfolgt die Übersetzung des Binärformats des Schlüssels in XML-Text mittels der Base64-Kodierung[20].

Auch asymmetrische private Schlüssel speichert BackupAes256 als Dateien mit der Endung .key, diese können vom Programm KeePass2 jedoch nicht gelesen werden. Öffentliche Schlüssel werden gleichfalls im XML-Format gespeichert, hier habe ich aber als Endung des Dateinamens .txt gewählt, um deutlich zu machen, dass diese Dateien wie ein gewöhnlicher,

18 https://keepass.info/download.html

19 https://de.wikipedia.org/wiki/Extensible_Markup_Language

20 https://de.wikipedia.org/wiki/Base64

öffentlicher Text verteilt werden können. Im Vergleich dazu soll die Endung .key zwar vertraulicher wirken, aber auch diese Dateien können mit einem Texteditor geöffnet und betrachtet werden. Wer andere Endungen verwenden will, kann das am Beginn der Datei »CryptoKey.cs« einstellen und das ausführbare Programm anschließend neu erstellen.

Aus Gründen der Einfachheit durchsucht BackupAes256 auf jedem Laufwerk nur das Wurzelverzeichnis nach Schlüsseln, beispielsweise »E:\«. Nur wenn es sich um das Betriebssystem-Laufwerk von Windows handelt, wird anstelle des Wurzelverzeichnisses das Verzeichnis »AppData\Local\BackupAes256« des aktuellen Benutzers als Schlüsselverzeichnis verwendet, also z.B. »C:\Users\Praxis\AppData\Local\BackupAes256«. In der Voreinstellung verbirgt Windows das AppData-Verzeichnis. Wer hineinschauen möchte, es aber nicht findet, kann das wie folgt im Windows-Explorer ändern: Organisieren => Ordner- und Suchoptionen => Ansicht => Ausgeblendete Dateien, Ordner und Laufwerke anzeigen.

Zum Erzeugen neuer Schlüssel muss am unteren Rand des Registers »Schlüssel« in dem Kombinationsfeld das gewünschte Format gewählt und anschließend auf »erzeugen« geklickt werden. Symmetrische und kürzere asymmetrische Schlüssel sind rasch erstellt, das Erzeugen eines asymmetrischen Schlüssels der längsten Variante dauert einige Minuten. Anschließend kann der Name des neuen Schlüssels geändert, ihm ein Laufwerk zugewiesen und er dort gespeichert werden.

Während ein Schlüssel erzeugt wird, kann die Bearbeitung mit »abbrechen« beendet werden. Dann ist es sofort möglich, mit BackupAes256 weiter zu arbeiten und z.B. andere Schlüssel zu erzeugen oder mit einem anderen Schlüssel eine Verschlüsselung zu starten. Allerdings lässt sich die tatsächliche Schlüsselerzeugung im Betriebssystem nicht stoppen, so dass die Prozessorauslastung hoch bleibt, bis der Schlüssel erstellt ist. Erst dann wird dieser Schlüssel verworfen.

Eingelesene und erzeugte, symmetrische und asymmetrische Schlüssel werden gemeinsam in einer Liste angezeigt und sind darin nach ihren Schlüsselnamen sortiert. Wurde ein externer Datenträger angeschlossen oder entfernt, kann die Liste mit der Taste F5 oder einem Klick auf die Schaltfläche »Laufwerke neu lesen« aktualisiert werden. Wählt man einen privaten asymmetrischen Schlüssel aus, zu dem kein gleichnamiger öffentlicher Schlüssel gefunden wurde, ist die Schaltfläche »exportieren« aktiv und der öffentliche Schlüssel kann auf das gewählte Laufwerk geschrieben werden.

Während symmetrische Schlüssel nur einen Wert besitzen, bestehen asymmetrische Schlüssel für das RSA-Verfahren aus mehreren Werten. Im Einzelnen sind das:

RSA-Wert	**Beschreibung**	**Geheimhaltung**
p	eine Primzahl	geheim
q	eine andere Primzahl	geheim

RSA-Wert	Beschreibung	Geheimhaltung
p * q	Das Produkt aus p und q. Diese Zahl wird auch mit dem Buchstaben N bezeichnet oder, mit Bezug auf die Division mit Rest, als Modul[21].	öffentlich
e	Der öffentliche Exponent oder Verschlüsselungs-exponent. Er muss eine zu dem Wert von (p - 1) * (q - 1) teilerfremde Zahl sein, die weder so klein (z.B. 3) ist, dass das Verfahren angreifbar wird, noch so groß, dass dieser Exponent während der Berechnung zu unnötig riesigen Zahlen führt. Für e wird üblicherweise die Zahl 65537 verwendet.	öffentlich
d	Der Entschlüsselungsexponent ist das multiplikativ Inverse von e bezüglich des Moduls (p - 1) * (q - 1) und kann über den Euklid-Algorithmus für den größten gemeinsamen Teiler bestimmt werden.	geheim
dp	d mod (p – 1), ein Hilfswert.	geheim
dq	d mod (q – 1), ein Hilfswert.	geheim
invq	(1 / q) mod p, ein Hilfswert.	geheim
Besitzer	Dieser Wert hat nichts mit dem eigentlichen Verfahren zu tun, sondern wurde von mir eingeführt, um den Besitzer des öffentlichen Schlüssels namentlich zu kennzeichnen.	öffentlich
E-Mail-Adresse	Dieser Wert hat nichts mit dem eigentlichen Verfahren zu tun, sondern wurde von mir eingeführt, um eine E-Mail-Adresse angeben zu können, unter der der Besitzer des öffentlichen Schlüssels zu erreichen ist, um verschlüsselte Daten zu übermitteln.	öffentlich
Homepage	Dieser Wert hat nichts mit dem eigentlichen Verfahren zu tun, sondern wurde von mir eingeführt, um auf die Homepage des Besitzers des öffentlichen Schlüssels verweisen zu können.	öffentlich

Die Hilfswerte dp, dq, und invq beziehen sich auf ein Rechenverfahren mit dem Namen Chinesischer Restsatz[22]. Sie zu speichern und nicht jedes Mal neu zu berechnen steigert die Effizienz des Verfahrens. Die Speicherung aller Schlüsselwerte erfolgt in Base64-Kodierung, auch die zu Besitzer, E-Mail-Adresse und Homepage. Wird der öffentliche Schlüssel ins Internet gestellt, können so die Daten nicht automatisiert von Spammern eingesammelt werden[23]. Darüber hinaus ermöglicht Base64 das problemlose Speichern und Übertragen beliebiger Sonderzeichen.

21 https://de.wikipedia.org/wiki/Division_mit_Rest#Modulo

22 https://de.wikipedia.org/wiki/RSA-Kryptosystem#RSA_mit_dem_Chinesischen_Restsatz

23 https://de.wikipedia.org/wiki/Spam#Verschleierung_der_E-Mail-Adresse

Die Schlüsselwerte können in binärer, dezimaler oder hexadezimaler Schreibweise angezeigt werden sowie in Base64-Kodierung. Die Auswahl der Schreibweise und des anzuzeigenden Wertes erfolgt in Kombinationsfeldern. Die Anzeige geheimer Werte lässt sich dort auch verbergen. Die Werte symmetrischer Schlüssel lassen sich ändern, mit der Ausnahme eingelesener BitLocker-Schlüssel. Bei asymmetrischen Schlüsseln können nur die Zusatzeigenschaften Besitzer, E-Mail-Adresse und Homepage geändert werden.

3.6. Geplante Programmfunktionen

Es wird möglich sein, Synchronisierungsaufgaben anzulegen, um wiederkehrende Einstellungen zu speichern.

Die noch weitgehend leere Registerkarte »Nachrichten« wird den Verlauf von Status- und Fehlermeldungen anzeigen.

Wenn als Laufwerk eines Schlüssels »ungespeichert« gewählt wird, kann dieser Schlüssel anschließend von dort gelöscht werden.

Ich werde testen, in wieweit es im Rahmen dieses Programms sinnvoll ist, eine Datei für mehrere Empfänger hybrid zu verschlüsseln, so wie beispielsweise OpenPGP es erlaubt. Diese Möglichkeit ist im jetzigen Format hybrider Dateien bereits angelegt.

Zum Aufbewahren und Übermitteln kryptographischer Schlüssel könnten QR-Codes[24] eingesetzt werden. So könnte ein vertraulicher symmetrischer Schlüssel vom Versicherten zunächst erstellt und in Form eines QR-Code auf ein Blatt Papier gedruckt werden. Arzt oder Ärztin des Vertrauens könnten diesen Schlüssel auf einem Computer ohne Netzwerkverbindung einscannen und das Papier anschließend vernichten. Anstelle einer Speicherung auf Festplatte, USB-Stick oder CD könnten alle Typen von Schlüsseln als QR-Code ausgedruckt und an einem sicheren Ort aufbewahrt werden. So wären sie gegen digitale Angriffe geschützt. In den maximalen Informationsgehalt eines QR-Code von 23.648 Bit würde auch der innerhalb des Programms BackupAes256 mit 16.384 Bit längstmögliche asymmetrische Schlüssel noch hinein passen.

3.7. Nächste Aktualisierungstermine

Eine Programmaktualisierung mit verbessertem Abfangen und Rückmelden von Fehlern bei der Ver- und Entschlüsselung sowie mit dann funktionierender Fortschrittsanzeige ist für Freitag, den 29.11.2019, geplant.

Im Verlauf des Dezember 2019 werde ich ein symmetrisch verschlüsseltes Dateisystem für umfangreiche Sicherungskopien ergänzen.

Eine unter dem Betriebssystem Linux lauffähige Programmversion ist für Januar 2020 geplant.

---

24 https://de.wikipedia.org/wiki/QR-Code

4. Kodieren und Verschlüsseln

4.1. Vertraulichkeit, Integrität, Verfügbarkeit

Informationssicherheit umfasst drei Schutzziele, die von einander weitgehend unabhängig sind: die Vertraulichkeit, die Integrität und die Verfügbarkeit von Daten.

Vertraulichkeit wird durch Rechtsnormen wie § 203 StGB geschützt. Durch technische Mittel kann sie gefördert oder erzwungen werden. Verfügbarkeit ist selten gesetzlich geregelt, stützt sich jedoch gleichfalls auf technisch-organisatorische Maßnahmen wie das regelmäßige Anlegen von Sicherungskopien und das Vorhalten von Ersatz-Hardware.

Wesentlich weniger bekannt als die Tatsache, dass die Vertraulichkeit von Daten unter anderem mittels Verschlüsselung und ihre Verfügbarkeit unter anderem durch Redundanz[25] sichergestellt werden kann, ist die Bedeutung von Datenintegrität.

Datenintegrität ist unabhängig vom Inhalt, d.h. der Bedeutung von Daten und versucht zu verhindern, dass Daten unbemerkt beschädigt oder verändert werden. Die Beschädigung kann durch technische und menschliche Fehler oder mit Absicht geschehen. Datenintegrität hat mit den anderen beiden Schutzzielen gemeinsam, dass sie mit technischen Mitteln gefördert oder erzwungen werden kann. Das wichtigste technische Verfahren, um Datenintegrität sicherzustellen, sind Nachrichtenauthentifizierungskodes[26].

Auch sicher verschlüsselte Daten können beschädigt werden, weshalb Verschlüsselung immer mit Authentifizierung kombiniert werden soll. Auf der nächsten Seite folgt ein Beispiel für bewahrte Vertraulichkeit ohne Schutz der Datenintegrität. Einen Text aus Goethes »Faust«, zweiter Teil, habe ich mittels AES-256 verschlüsselt, nach dem Verschlüsseln die Daten in vier Blöcke geteilt und die Blöcke in der Reihenfolge 1–3–2–4 wieder zusammengefügt. Das Ergebnis war ohne technische Probleme zu entschlüsseln und ergab den folgenden Text. Die von mir rot hervorgehobenen Fehler haben jeweils die Länge eines AES-Blocks von 16 Byte. Erst die, hier fehlende, Authentifizierung hätte zu einer Fehlermeldung geführt.

25 https://de.wikipedia.org/wiki/Redundanz_(Technik)

26 https://de.wikipedia.org/wiki/Message_Authentication_Code

Faust
Wohin der Weg? –

Mephistopheles
Kein Weg! Ins Unbetretene,
Nicht zu Betretende; ein Weg ans Unerbetene,
Nicht zu Erbittende. Bist du bereit? –
Nicht Schlösser sind, nicht Riegel wegzuschieben,
Von Einsamkeiten wirst umhergetrieben.
Hast du Begriff von öd' und Einsamkeit?

Faust
Du spartest, dächt' ich, solche Sprüche;
Hier wittert's nach der Hexenküche,
Nach einer längst vergangnen Zeit.
Mußt' ich nicht mit der Welt verkehren?
Das Leere lernen, Leeres lehren? –
Sprach ich vernünftig, wie ich's angeschaut,
Erklang der Widerspruch gedoppelt laut;
Mußt' ich sogar vor widerwärtigen Streichen
Zur Einsamkeit, zur Wildernis entweichen
Und, um nicht ganz versäumt, allein zu leben,
Mich doch zuletzt dem Teufel übergeben.

Mephistopheles
Und hättest du den Ozean durchschwommen,
Das Grenzenlose dort geschaut,
So sähst du dort doch Well' auf Welle kommen,
Selbst wenn es dir vorm Untergange **Ñ¼øp¢Ä#U£1[0ì“@R**tern.

Faust
Den Müttern! Trifft's mich immer wie ein Schlag!
Was ist das Wort, das ich nicht hören mag?

Mephistopheles
Bist du beschränkt, daß neues Wort dich stört?
Willst du nur hören, was du schon gehört?
Dich störe nichts, wie es auch weiter klinge,
Schon längst gewohnt der wunderbarsten Dinge.

Faust
Doch im Erstarren such' ich nicht mein Heil,
Das Schaudern ist der Menschheit bestes Teil;
Wie auch die Welt ihm das Gefühl verteure,
Ergriffen, fühlt er tief das Ungeheure.

Mephistopheles
Versinke denn! Ich könnt' auch sagen: steige!
's ist einerlei. Entfliehe dem Entstandnen
In der Gebilde losgebundne Reiche!
Ergetze dich am längst nicht mehr Vorhandnen;
Wie Wolkenzüge schlingt sich das Getreibe,
Den Schlüssel schwinge, halte sie vom Leibe!

Faust
Wohl! fest ihn fassend fühl' ich neue Stärke,
Die Brust erweitert, hin zum großen Werke.

Mephistopheles
Ein glühnder Dreifuß tut dir endlich kund,
Du seist im tiefsten, allertiefsten Grund.
Bei seinem Schein wi***‰6È6;°Ë]å#i'ûâþ** doch etwas. Sähst wohl in der Grüne
Gestillter Meere streichende Delphine;
Sähst Wolken ziehen, Sonne, Mond und Sterne –
Nichts wirst du sehn in ewig leerer Ferne,
Den Schritt nicht hören, den du tust,
Nichts Festes finden, wo du ruhst.

Faust
Du sprichst als erster aller Mystagogen,
Die treue Neophyten je betrogen;
Nur umgekehrt. Du sendest mich ins Leere,
Damit ich dort so Kunst als Kraft vermehre;
Behandelst mich, daß ich, wie jene Katze,
Dir die Kastanien aus den Gluten kratze.
Nur immer zu! wir wollen es ergründen,
In deinem Nichts hoff' ich das All zu finden.

Mephistopheles
Ich rühme dich, eh' du dich von mir trennst,
Und sehe wohl, daß du den Teufel kennst;
Hier diesen Schlüssel nimm. –

Faust
Das kleine Ding!

Mephistopheles
Erst faß ihn an und schätz ihn nicht gering.

Faust
Er wächst in meiner Hand! er leuchtet, blitzt!

Mephistopheles
Merkst du nun bald, was man an ihm besitzt?
Der Schlüssel wird die rechte Stelle wittern,
Folg ihm hinab, er führt**ÎP²Ži„ì…ó: ó9YÉ**r sehn,
Die einen sitzen, andre stehn und gehn,
Wie's eben kommt. Gestaltung, Umgestaltung,
Des ewigen Sinnes ewige Unterhaltung.
Umschwebt von Bildern aller Kreatur;
Sie sehn dich nicht, denn Schemen sehn sie nur.
Da faß ein Herz, denn die Gefahr ist groß,
Und gehe grad' auf jenen Dreifuß los,
Berühr ihn mit dem Schlüssel! –

Mephistopheles
So ist's recht!
Er schließt sich an, er folgt als treuer Knecht;
Gelassen steigst du, dich erhebt das Glück,
Und eh' sie's merken, bist mit ihm zurück.
Und hast du ihn einmal hierher gebracht,
So rufst du Held und Heldin aus der Nacht,
Der erste, der sich jener Tat erdreistet;
Sie ist getan, und du hast es geleistet.
Dann muß fortan, nach magischem Behandeln,
Der Weihrauchsnebel sich in Götter wandeln.

Faust
Und nun was jetzt? –

Mephistopheles
Dein Wesen strebe nieder;
Versinke stampfend, stampfend steigst du wieder.

Mephistopheles
Wenn ihm der Schlüssel nur zum besten frommt!
Neugierig bin ich, ob er wiederkommt.

4.2. Was ist ein kryptographischer Schlüssel?

Ein kryptographischer Schlüssel ist eine Zahl. Diese Zahl ist immer eine positive ganze Zahl, also ohne Minus und ohne Kommastellen. Um die Vertraulichkeit der mit ihr verschlüsselten Daten zu garantieren, muss diese positive ganze Zahl viel größer sein als alle Zahlen, die uns sonst im Alltag begegnen.

Der Advanced Encryption Standard beispielsweise verwendet als Schlüssel Zahlen zwischen 0 und 115.792.089.237.316.195.423.570.985.008.687.907.853.269.984.665.640.564.039.457.584.007.913.129.639.935.

4.3. Bits, Bytes und das Stellenwertsystem mit der Basis 16

Um zu wissen, woher die »krummen« Zahlen beim Umgang mit Computern kommen, ist es hilfreich, Zahlensysteme zu verstehen. Eines nicht mehr fernen Tages werden neuronale Netze und Quantencomputer flexibler mit Zahlen umgehen können. Aber vorerst kennen Computer nur die beiden Zustände »Strom fließt« und »Strom fließt nicht«. Jenseits der mittlerweile sprichwörtlichen 0 und 1 sehen Zahlen in einem Computer deshalb anders aus als die Zehner-, Hunderter- und Tausenderstellen in unserem Alltag.

Man könnte es bei der 0 und 1 des binären Zahlensystems belassen und anstelle 256-Bit-Verschlüsselung von 100000000-Bit-Verschlüsselung sprechen. Nur wäre das platzraubend und unübersichtlich. So hat es sich bewährt, dass das menschliche Gehirn und die 0 und 1 des Computers sich im Stellenwertsystem mit der Basis 16 treffen. Um nicht durcheinander zu geraten, stellt man solchen Hexadezimalzahlen eine 0x voran. Also 0x100-Bit-Verschlüsselung.

Ein wenig Mathematik: 256 = 10 * 10 * 2 + 10 * 5 + 6 = 16 * 16 * 1 + 16 * 0 + 0. Verwendet man Sechzehner-, Zweihundertsechsundfünfziger- und Viertausendsechsundneunziger-stellen anstelle der Zehner-, Hunderter- und Tausenderstellen, lässt sich daraus in kleinen »Happen« auf die interne Darstellung im Computer schließen und »krumme« Dezimalzahlen stehen als »schöne« Hexadezimal- oder Binärzahlen da.

Weil es für 10, 11, 12, 13, 14 und 15 keine einzelnen Ziffern gibt, verwendet man in der Hexadezimalschreibweise dafür die Buchstaben, A, B, C, D, E und F, wahlweise groß oder klein geschrieben. Ein unschönes Detail ist, dass »Byte« so klingt wie englisch »bite« = deutsch »Bissen« und auch davon abgeleitet ist. Allerdings hat es sich so entwickelt, dass man zwei Happen = Stellen in der hexadezimalen Schreibweise als ein Byte bezeichnet. Das Byte 10010110 im Computer entspricht also der Zahl 2 * 2 * 2 * 2 * 2 * 2 * 2 * 1 + 2 * 2 * 2 * 2 * 2 * 2 * 0 + 2 * 2 * 2 * 2 * 2 * 0 + 2 * 2 * 2 * 2 * 1 + 2 * 2 * 2 * 0 + 2 * 2 * 1 + 2 * 1 + 0 = 150 = 16 * 9 + 6 = hexadezimal 0x96. Eine Stelle hexadezimal fasst vier Nullen oder Einsen, d.h. vier Bits, der internen Computerschreibweise zusammen und kodiert so ein halbes Byte. Acht Bits gleich ein Byte. Ein Byte kodiert die gleiche Informationsmenge wie log(256) / log(10) ≈ 2,40824 Dezimalziffern. Und scherzhaft, deshalb wenig gebräuchlich, besteht ein Byte aus zwei Nibble. Ein Nibble ist gleich einer Hexadezimalziffer.

Zeit zur Auflockerung. Zur Erinnerung an den Science-Fiction-Schriftsteller Douglas Adams wird am 25. Mai jeden Jahres der »Handtuchtag«[27] begangen. In seinem Roman »Per Anhalter durch die Galaxis« wird ein Handtuch als »so ziemlich das Nützlichste« bezeichnet, was man auf Reisen durch die Galaxis mit sich führen kann. Die Antwort auf die Frage aller Fragen, die Frage »nach dem Leben, dem Universum und dem ganzen Rest« beantwortet der Roman mit »42«[28]. Zu Lebzeiten von Douglas Adams hatte der 25. Mai keine Verbindung zu ihm oder seinen Büchern. Erst nach Adams‘ Tod kam jemand auf die Idee, man könne doch hexadezimal rechnen 0x25 + 0x5 = 0x2a = 16 * 2 + 10 = 42.

Im Computer muss auch alles Nichtmathematische wie Texte, Bilder und Musik auf Nullen und Einsen zurückgeführt werden. So wird der Text »i love you« als 0x69 0x20 0x6c 0x6f 0x76 0x65 0x20 0x79 0x6f 0x75 gespeichert[29]. Für die Binärschreibweise dessen höre man das Lied »Analogpunk«[30] von Judith Holofernes.

4.4. Eine kurze Geschichte der symmetrischen Verschlüsselung

Schon der römische Feldherr Gaius Julius Caesar kannte Verschlüsselung. In den Nachrichten an seine Truppen, die nicht in die Hand des Feindes fallen durften, ersetzte er jeden Buchstaben durch den an 3. Stelle nachfolgenden Buchstaben. Anstelle »Telematikinfrastruktur« hätte Caesar »Whohpdwlnlqiudvwuxnwxu« geschrieben. Dieser sehr einfache und trotzdem einschüchternd wirkende Kode ist als Caesar-Verschlüsselung[31] bekannt.

Die im Verlauf der Jahrhunderte häufig unsichere, unsystematische und im Vergleich zu heute seltene Anwendung von Verschlüsselung erfuhr mit der Verbreitung der Telegraphie eine Systematisierung. Im Jahr 1883 veröffentlichte der niederländische Linguist Auguste Kerckhoffs (1835 bis 1903) seine Schrift »La cryptographie militaire« (deutsch »Die militärische Kryptographie«). Darin formulierte er einen Grundsatz für sichere Kommunikation, der bis heute gilt: Die Sicherheit des Verfahrens darf sich allein auf seine mathematische Stärke und auf die Geheimhaltung des Schlüssels stützen. Falls der Rest des Systems bekannt wird, insbesondere der Verschlüsselungsalgorithmus und die verwendete Hardware, darf kein Schaden entstehen. Im Gegenteil zu dem naiven Impuls, auch das Verschlüsselungsverfahren geheim zu halten, soll es veröffentlicht werden, so dass möglichst viele Experten es auf Schwächen untersuchen können.

Im Ersten Weltkrieg, 15 Jahre nach Kerckhoffs‘ Tod, trug die Entzifferung deutscher Funksprüche ab Frühjahr 1918 nach Ansicht einiger Historiker und Kryptologen dazu bei, dass es den Deutschen nicht gelang, Paris einzunehmen[32].

---

27 http://towelday.org/faq/#whydate

28 https://de.wikipedia.org/wiki/42_(Antwort)

29 https://de.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange

30 https://www.youtube.com/watch?v=LDbJ2Mx-I6o&t=112

31 https://de.wikipedia.org/wiki/Caesar-Verschl%C3%BCsselung

32 https://de.wikipedia.org/wiki/ADFGX#Entzifferung

Im zweiten Weltkrieg hatten die Deutschen aus dieser Niederlage gelernt und verwendeten die wesentlich komplexere, maschinelle Ver- und Entschlüsselung durch die Enigma-Maschine. Jedoch führte die Geheimhaltung ihres Aufbaus und damit die Nichtbeachtung von Kerckhoffs' Prinzip dazu, dass die volle mathematische Stärke dieser Konstruktion nicht genutzt wurde. Ab Januar 1940 und nahezu während des gesamten Zweiten Weltkriegs konnte insbesondere das britische Militär weite Teile des deutschen Nachrichtenverkehrs auf allen Ebenen mitlesen. Es wird vermutet, dass dieser Informationsvorsprung den Zweiten Weltkrieg um mindestens zwei Jahre verkürzte oder ihn überhaupt entschied. Heute ist die öffentliche Analyse der Stärken und Schwächen im Aufbau der Enigma Teil eines als exzellent ausgezeichneten Wikipedia-Artikels[33].

4.5. Der Advanced Encryption Standard (AES)

Am Beginn dieses Verfahrens stand eine Ausschreibung im Jahr 1997, bei der ein Advanced Encryption Standard[34] (deutsch »fortschrittlicher Verschlüsselungsstandard«) gesucht wurde. Aus fünfzehn eingereichten Vorschlägen wurde in mehreren öffentlichen Konferenzen und Diskussionen ein Algorithmus aus Sieger bestimmt, der überdurchschnittlich schnell ist und, soweit erkennbar, mit realistischem physikalisch-rechnerischem Aufwand nicht gebrochen werden kann. Dieser Algorithmus wurde im Oktober 2000 vom Nationalen Institut für Standards und Technologie der Vereinigten Staaten (NIST) als Standard bekanntgegeben. Der Ablauf dieses Wettbewerbs gilt bis heute als vorbildlich.

Auch bei AES handelt es sich um ein symmetrisches Verschlüsselungsverfahren, das heißt die Schlüssel zum Ver- und Entschlüsseln sind identisch. Der Algorithmus ist eine Blockchiffre, bei der Blöcke von 128 Bit = 16 Byte gemeinsam ver- und entschlüsselt werden. Die möglichen Schlüssellängen sind auf 128, 192 oder 256 Bit festgelegt. Durch zahlreiche Vertauschungen und Verknüpfungen in aufeinander folgenden »Runden« wird die Wahrscheinlichkeit mathematischer Angriffe auf das Verfahren minimiert.

Da zu verschlüsselnde Dokumente praktisch immer länger sind als die Blocklänge von 16 Byte, wird zusätzlich ein Betriebsmodus[35] benötigt, um aufeinander folgende Datenblöcke so miteinander zu verknüpfen, dass daraus nicht Teile des verschlüsselten Inhalts oder der geheime Schlüssel abgeleitet werden können. Sicherer als die Daten erst zu authentifizieren und anschließend zu verschlüsseln ist es, erst zu verschlüsseln und die verschlüsselten Daten anschließend zu authentifizieren.[36] Beide Schlüssel dürfen nicht identisch sein.

4.6. Die Festplattenverschlüsselung BitLocker als ein Beispiel für AES

Das Aktivieren der Festplattenverschlüsselung mit BitLocker unter Windows ist einfacher, als es klingt. Doch ist es für Nutzer von Windows Home mit 145 € nicht ganz billig. Getestet habe ich BitLocker unter Windows 10, es funktioniert aber auch unter Windows 8 und Windows 7. BitLocker jetzt noch unter Windows 7 zu installieren, lohnt sich nicht mehr, denn

33 https://de.wikipedia.org/wiki/Enigma_(Maschine)

34 https://de.wikipedia.org/wiki/Advanced_Encryption_Standard

35 https://de.wikipedia.org/wiki/Betriebsmodus_(Kryptographie)

36 https://crypto.stackexchange.com/questions/202/should-we-mac-then-encrypt-or-encrypt-then-mac

ab Januar 2020 werden für dieses Betriebssystem keine aktuellen Sicherheitsupdates mehr ausgeliefert, so dass Windows 7 auf einem Praxisrechner sich spätestens dann verbietet.[37]

Ratsam ist zunächst, alle Daten vom Rechner auf externer Festplatte, SD-Karte oder USB-Stick zu sichern. Bei der Installation von Windows Pro und BitLocker geht selten etwas schief. Aber falls doch, sind dann so keine Daten verloren. Um herauszufinden, welche Edition von Windows auf dem Rechner läuft, kann man zum Beispiel die Windows-Logo-Taste + Pause (neben F12) drücken oder die Windows-Logo-Taste + R drücken und msinfo32 öffnen. Sollte dort Windows Home stehen, muss als Voraussetzung für den Einsatz von BitLocker zunächst das kostenpflichtige Upgrade auf Windows Pro durchgeführt werden.

Für das Upgrade zu Windows 10 Pro wählt man Einstellungen => Update und Sicherheit => Aktivierung => Store aufrufen. Dort klickt man auf die Schaltfläche »Kaufen« und muss sich dann zunächst in ein bestehendes Microsoft-Konto einloggen oder ein neues Konto erstellen. Darin kann zwischen zwei Bezahlmethoden gewählt werden: Sofortüberweisung[38] und Giropay[39]. Den besseren Datenschutz bietet Giropay, jedoch wird diese Methode nicht von allen Banken angeboten. Wer das Praxiskonto beispielsweise bei der apoBank hat, muss auf die Abbuchung von einem Konto bei einer anderen Bank ausweichen oder die Sofortüberweisung nutzen, welche die Zugangsdaten zum Onlinebanking an Dritte weitergibt. Dann sollte man seine Zugangsdaten zum Onlinebanking im Anschluss ändern. Unter »Neue Zahlungsmethode hinzufügen« werden Kreditkarte, PayPal und Klarna angeboten. Sie habe ich an dieser Stelle nicht getestet. Nach erfolgtem Kauf muss der Rechner neu gestartet werden und nach wenigen Minuten ist Windows Pro installiert.

Vor der Aktivierung von BitLocker stellen sich mehrere Glaubens- und Geschmacksfragen. Die erste Frage ist jene für oder wider den in den meisten aktuellen Rechnern verbauten TPM-Chip[40] (Trusted Platform Module). Ob der Rechner über einen TPM-Chip verfügt, findet man heraus, indem man in einem Benutzerkonto mit Administratorrechten die Windows-Logo-Taste + R drückt und tpm.msc öffnet. Dort steht unter »Status«, ob das TPM vorhanden und einsatzbereit ist.

Für dessen Verwendung spricht, dass mit dem Chip die Festplatte nur dann wieder entschlüsselt werden kann, wenn die Hardware des Rechners unverändert ist. Würde man die Festplatte in einen anderen Rechner einbauen, um dort die Verschlüsselung zu brechen, ließe der TPM-Chip den Versuch scheitern. Gegen den TPM-Chip spricht, dass dessen Manipulation durch die NSA oder andere staatliche Stellen nicht auszuschließen ist. Diese könnten dann die Festplatte selbst entschlüsseln oder deren Entschlüsselung durch den Besitzer verhindern.

Nicht zu empfehlen ist die Authentifizierung von BitLocker allein über das TPM, weil einige Angriffe auf die verschlüsselten Daten dann trotzdem möglich wären. So lautet die zweite

37 https://support.microsoft.com/de-de/help/4057281/windows-7-support-will-end-on-january-14-2020

38 https://de.wikipedia.org/wiki/Sofort%C3%BCberweisung

39 https://de.wikipedia.org/wiki/Giropay

40 https://de.wikipedia.org/wiki/Trusted_Platform_Module

Entscheidung, was entweder als alleinige Authentifizierung ohne TPM oder als zweiter Authentifizierungsfaktor zusätzlich zum TPM-Chip gewählt werden soll: Passwort oder Schlüsseldatei?

Vor- und zugleich Nachteil des Passwortes ist, dass es nicht aufgeschrieben werden muss. Was man im Gedächtnis hat, kann zwar nicht ausgespäht, aber vergessen werden. Das Passwort ist weniger komplex als die Schlüsseldatei und deshalb leichter durch Probieren zu finden. Bei jedem Start des Rechners muss es eingegeben werden.

Vor- und zugleich Nachteil der Schlüsseldatei ist deren Länge. Um sich diese Information zu merken, müsste man ein Gedächtniskünstler sein. Deshalb wird die Schlüsseldatei auf einem USB-Stick oder einer SD-Karte abgelegt (die jeweils beliebige weitere, nicht mit BitLocker verschlüsselte Daten enthalten dürfen). Bei jedem Einschalten muss der Datenträger mit der Schlüsseldatei mit dem Rechner verbunden sein und kann dann ggf. bis zum nächsten Einschalten wieder entfernt werden. Auch von der Schlüsseldatei sollte man sich eine oder besser zwei Sicherungskopien anfertigen und sie gut verwahren. Sucht man die Schlüsseldatei auf dem USB-Stick, sieht man zunächst nichts, da sie als Systemdatei gekennzeichnet ist und Windows Systemdateien standardmäßig nicht anzeigt. Das kann man ändern, indem man im Windows-Explorer auf Datei => Ordner- und Suchoptionen ändern => Ansicht geht und das Häkchen bei »Geschützte Systemdateien ausblenden (empfohlen)« entfernt. Jetzt zeigt Windows im Windows-Explorer die Schlüsseldatei mit an, deren Name auf *.BEK endet. Nicht passieren darf es, dass USB-Stick bzw. SD-Karte und damit die Schlüsseldatei gemeinsam mit dem verschlüsselten Rechner in die Hände Unbefugter geraten.

Eine Entscheidungshilfe zu der Frage TPM-Chip, Passwort und/oder Schlüsseldatei bietet dieser Artikel[41] (auf Englisch). Microsoft empfiehlt die Kombination aus TPM-Chip und Passwort. Ich bevorzuge die Kombination aus TPM-Chip und USB-Stick, um mir das Eintippen eines weiteren Passwortes vor dem eigentlichen Windows-Login zu sparen. So macht sich BitLocker beim Benutzer nicht bemerkbar, solange beim Anschalten des Rechners der USB-Stick mit der Schlüsseldatei steckt.

Vor dem Einrichten von BitLocker kann eine dritte Entscheidung getroffen werden. Die Verschlüsselungsstärke von BitLocker ist auf 128 Bit voreingestellt, kann aber auf 256 Bit erhöht werden, ohne dass es die Schnelligkeit der meisten Rechner wesentlich bremst.

Zusätzliche Informationen zu BitLocker findet man im Internet beispielsweise bei bei Microsoft, Wikipedia und YouTube. Um die Installation durchzuführen, müssen wir Windows mitteilen, wie wir uns hinsichtlich TPM-Chip, Password oder Schlüsseldatei und Verschlüsselungsstärke entschieden haben. Das geschieht in den sogenannten Gruppenrichtlinien. Um dorthin zu gelangen, drückt man die Windows-Logo-Taste + R, öffnet gpedit.msc und klickt sich durch bis zum Unterordner Richtlinien für Lokaler Computer => Computerkonfiguration => Administrative Vorlagen => Windows-Komponenten => BitLocker-Laufwerkverschlüsselung. (Bitte nicht in die Benutzerkonfiguration verrutschen, wo es auch Administrative Vorlagen gibt.)

41 https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-7/ee706531(v=ws.10)

Im Unterordner BitLocker-Laufwerkverschlüsselung angekommen, stehen auf der rechten Seite mehrere Einträge mit dem Namen »Verschlüsselungsmethode und Verschlüsselungsstärke für Laufwerk auswählen«. Der passende Eintrag ist der für Windows-Version 1511 und höher. Ist man sich nicht sicher, können auch alle Einträge bearbeitet werden. Nach Doppelklick auf den Eintrag muss die Option »Aktiviert« und darunter die gewünschte Kombination aus Verschlüsselungsmethode und Verschlüsselungsstärke gewählt werden, beispielsweise XTS-AES 256-Bit.

Zurück auf der linken Seite wählt man den Unterordner »Betriebssystemlaufwerke« und doppelklickt auf der rechten Seite auf den Eintrag »Zusätzliche Authentifizierung beim Start anfordern«. Auch hier muss wieder die Option »Aktiviert« gewählt werden. Möchte man den TPM-Chip nicht nutzen, muss das Häkchen bei »BitLocker ohne kompatibles TPM zulassen« gesetzt sein. Auch die nachfolgenden Einstellungen zu Passwort (PIN) und Schlüsseldatei können entsprechend den eigenen Wünschen gesetzt werden.

Dieses Fenster kann jetzt wieder geschlossen werden und es geht weiter im Windows-Explorer, wo man mit der rechten Maustaste auf das Betriebssystem-Laufwerk (meist C:) klickt, »Bitlocker aktivieren« wählt und die Laufwerksverschlüsselung startet. Je nach Größe des Laufwerks dauert der Verschlüsselungsvorgang zwischen einer und mehreren Stunden. Dabei ist es ratsam, den ungenutzten Platz des Laufwerks mit zu verschlüsseln, da sich dort Reste gelöschter Daten befinden können, die sonst für einen Angreifer lesbar bleiben. Die Abläufe beim Verschlüsseln der Festplatte zeigt dieses Video[42] (auf Englisch). Am Ende darf nicht vergessen werden, den Wiederherstellungsschlüssel zu sichern, am besten sowohl als Datei auf einem externen Laufwerk als auch als Ausdruck auf Papier.

4.7. Eine kurze Geschichte der hybriden Verschlüsselung

Bis in die 1970er Jahre gab es nur symmetrische Verschlüsselung, bei der Sender und Empfänger denselben Schlüssel besitzen mussten. Das wirft das Problem auf, dass es keinen sicheren und einfachen Weg gibt, auf dem der Schlüssel unabhängig von den mit ihm verschlüsselten Informationen vom Sender zum Empfänger gelangen kann. Am sichersten ist es, wenn Sender und Empfänger sich zunächst einmal persönlich treffen, um den gemeinsamen symmetrischen Schlüssel zu vereinbaren.

Nach entweder geheim gehaltenen oder erfolglosen Versuchen am Anfang der 1970er Jahre wurde 1977 von Rivest, Shamir und Adleman das erste sichere asymmetrische Verschlüsselungsverfahren entwickelt und nach ihnen RSA-Verfahren genannt. Bei entsprechend großen Schlüssellängen gilt es bis heute als sicher und wird häufig verwendet. Wenn eine anschauliche Metapher für symmetrische Verschlüsselung die Haustür ist, ist die Metapher für asymmetrische Verschlüsselung der Briefkasten.

Es ist ebenso üblich wie irreführend, von »öffentlichen« und »privaten« asymmetrischen Schlüsseln zu sprechen. Niemand käme auf die Idee, den Briefkastenschlitz als »öffentlichen Briefkastenschlüssel« zu bezeichnen. Während die Funktion eines Briefkastens durch Gesetze der Physik sichergestellt ist, stellen Gesetze der Mathematik sicher, dass es

42 https://www.youtube.com/watch?v=5o9zGAOOg4c

einfach ist, aus einem »privaten Schlüssel« den zugehörigen »öffentlichen Schlüssel« zu berechnen, jedoch äußerst unwahrscheinlich, aus einem »öffentlichen Schlüssel« zurück auf den zugehörigen »privaten Schlüssel« zu schließen. Der »öffentliche Schlüssel« dient dem doppelten Zweck, Daten für den Besitzer des zugehörigen »privaten Schlüssels« zu verschlüsseln und von diesem empfangene Daten auf Integrität zu prüfen. Der »private Schlüssel« dient dem doppelten Zweck, Daten vor dem Versand zu authentisieren und Daten, die mit dem zugehörigen »öffentlichen Schlüssel« verschlüsselt wurden, wieder zu entschlüsseln. Die Prüfung, ob Daten unbeschädigt und deshalb mit hoher Wahrscheinlich authentisch sind, ist unabhängig von Verschlüsselung und funktioniert auch mit im Klartext versandten oder teils verschlüsselten, teils unverschlüsselten Daten.

Sollen Daten verschlüsselt werden, die länger sind, als der asymmetrische Schlüssel selbst, wird asymmetrische Verschlüsselung in der Praxis immer in einer Kombination mit symmetrischer Verschlüsselung verwendet. Die Bezeichnung hierfür ist hybride Verschlüsselung. Asymmetrische Verfahren arbeiten wegen ihres Konstruktionsprinzips wesentlich langsamer als symmetrische Verfahren. Durch diesen Nachteil, dem kein Vorteil gegenüber steht, wurde die theoretische Möglichkeit, aufeinander folgende Blöcke von Daten jeweils mit demselben asymmetrischen Schlüssel zu verschlüsseln, nie standardisiert oder hinreichend auf Schwachstellen analysiert. Gerade bei der Datensicherheit soll man nicht »das Rad neu erfinden«, sondern auf bewährte, schon von vielen Experten durchdachte Konzepte setzen. Reine asymmetrische Verschlüsselung bleibt für Alltagszwecke deshalb bedeutungslos.

Hybride Verschlüsselung verbindet die Schnelligkeit symmetrischer mit dem leichteren Schlüsselaustausch asymmetrischer Verfahren. Zugleich hat sie gegenüber rein symmetrischer Verschlüsselung mehrere Nachteile. Ein Nachteil hybrider Verschlüsselung besteht darin, dass sie gebrochen ist, sobald einer ihrer beiden Teile gebrochen ist. Gelingt es, aus einem öffentlichen Schlüssel den zugehörigen privaten Schlüssel zu berechnen, wird der mit den Daten gemeinsam übertragene symmetrische Schlüssel lesbar. Gelingt es, den symmetrischen Schlüssel durch Ausprobieren zu finden, hat man den asymmetrischen Teil des Verfahrens umgangen.

Ein weiterer Nachteil asymmetrischer und damit auch hybrider Verschlüsselung besteht darin, dass auch sie das eingangs geschilderte Problem nicht vollständig löst, einen sicheren und einfachen Weg zu finden, auf dem ein kryptographischer Schlüssel unabhängig von den mit ihm verschlüsselten Informationen vom Sender zum Empfänger gelangen kann. Da hier Sender und Empfänger sich nicht persönlich treffen, um ihre Schlüssel zu vereinbaren, verbleibt ein Risiko, dass Identitäten gefälscht sind. Während private Schlüssel ihrerseits verschlüsselt oder sicher verwahrt werden müssen, benötigen öffentliche Schlüssel zusätzliche Identitätsmerkmale, um Fälschungen vorzubeugen. Das kann die schon länger bestehende Praxis-Homepage sein, die den öffentlichen Schlüssel der Praxis zum Herunterladen anbietet, die telefonische Nachfrage beim (vermeintlichen) Absender einer E-Mail, der ein öffentlicher Schlüssel angehängt ist, vielleicht in Zukunft auch der Überweisungsschein mit Praxisstempel, dem ein öffentlicher Schlüssel in Form eines QR-Code aufgedruckt ist. Für öffentliche Schlüssel muss ein »Netz des Vertrauens«[43] aufgebaut werden.

43 https://de.wikipedia.org/wiki/Web_of_Trust

Alle öffentlichen Schlüssel durch eine zentrale Stelle nicht nur (mit-)speichern, sondern auch beglaubigen zu lassen, würde dagegen in blindes Vertrauen zurück führen. Wer zertifiziert die Zertifizierungsstelle, die das Zertifikat der Zertifizierungsstelle zertifiziert, das...?

Selbst dann noch, wenn keine öffentlichen Schlüssel unter falscher Identität in Umlauf sind, gibt es Angriffsmöglichkeiten, um wahre Identitäten gegen einander auszutauschen. Geht man von der Denkweise symmetrischer Verschlüsselung aus, bei der nur die Identität des einen Schlüssels existiert, ist das zunächst nicht ganz einfach zu verstehen. Wird eine Nachricht erst verschlüsselt und dann signiert, könnte jeder, der diese Nachricht abfängt, die ursprüngliche Signatur entfernen, durch seine eigene Signatur ersetzen und sich so als der wahre Absender ausgeben. Deshalb sollte die Identität des Absenders immer mit verschlüsselt und nicht allein über die Signatur bekannt gegeben werden.

Wird eine Nachricht hingegen erst signiert und dann verschlüsselt, sind die Angriffsmöglichkeiten geringer. Das steht im Gegensatz zu symmetrischer Verschlüsselung, bei der es sicherer ist, erst zu verschlüsseln und dann zu authentifizieren. Als Angreifer kommt hier nur der legitime Empfänger als in Betracht, der die Nachricht entschlüsseln, ihre Signatur beibehalten und beides mit dem öffentlichen Schlüssel eines Dritten neu verschlüsseln könnte. Dieser Angriff wird als »heimliches Weiterleiten« bezeichnet.[44] Deshalb sollte die Identität des Empfängers immer mit signiert und nicht allein über Verschlüsselung sichergestellt werden.

4.8. Das RSA-Kryptosystem nach Rivest, Shamir und Adleman

Das RSA-Verfahren nach Rivest, Shamir und Adleman aus dem Jahr 1977 gilt bis heute als sicher und ist ein sehr häufig verwendetes asymmetrisches Verfahren innerhalb hybrider Verschlüsselung. Es beruht auf der Mathematik der Primzahlen. Eine Primzahl ist eine ganze Zahl, die größer ist als 1 und die ausschließlich durch sich selbst und durch 1 teilbar ist. Die Folge der Primzahlen beginnt mit 2, 3, 5, 7, 11, 13 und ist unendlich. Es ist wesentlich leichter, zwei Primzahlen miteinander zu multiplizieren als umgekehrt für eine gegebene Zahl auszuprobieren, aus welchen Primfaktoren sie besteht.

Beispielsweise könnte man von der Zahl 143 auf den ersten Blick vermuten, dass sie das Dreifache einer anderen ganzen Zahl sei. Bis man auf die einzig mögliche Primzahlzerlegung in 11 * 13 kommt, dauert es etwas länger. Umgekehrt gibt es eine einfache Rechenvorschrift dafür, die Zahlen 11 und 13 miteinander zu multiplizieren und das Ergebnis 143 zu erhalten.

Auf dieser Grundlage sind zwei geheim gehaltene Primzahlen der »private Schlüssel« des RSA-Verfahrens, der zum Entschlüsseln von Nachrichten dient. Ihr Produkt ist der »öffentliche Schlüssel«, der zum Verschlüsseln verwendet wird und dessen Länge die Schlüssellänge des Verfahrens definiert. Der »öffentliche Schlüssel« darf und soll bekannt gemacht werden, so dass jede(r) dem Besitzer oder der Besitzerin des zugehörigen »privaten Schlüssels« verschlüsselte Nachrichten schreiben kann. Entschlüsseln kann diese Nachricht jedoch nur, wer die beiden Primzahlen kennt.

---

44 http://world.std.com/~dtd/sign_encrypt/sign_encrypt7.html

Auch hier müssen die für eine sichere Verschlüsselung verwendeten Primzahlen um ein Vielfaches größer sein als die Zahlen im obigen Beispiel und größer als alle anderen Zahlen in unserem Alltag. Anders als bei symmetrischer Blockverschlüsselung ist die Länge dieser Zahlen frei wählbar. Trotz einiger Gegenargumente[45] nimmt man auch hier meist Längen, die in der binären Darstellung des Computers »runde« Zahlen ergeben.

Grenzstein mit der Zahl 2^{12}, Südufer des Großen Stechlinsee, Brandenburg

Die voraussichtlich noch für einige Jahre sichere Mindestlänge eines RSA-Schlüssels sind 4096 Bit[46] = 512 Byte. Mit den log(256) / log(10) ≈ 2,40824 Dezimalstellen pro Byte multipliziert und dann aufgerundet entspricht das einer Dezimalzahl mit 1234 Stellen als öffentlichem Schlüssel. Da die Rechenleistung von Supercomputern auch ohne Quantenkryptographie weiter steigt, müsste man für langfristig sicher aufzubewahrende Daten, wie Gesundheitsdaten es sind, »schon heute Schlüssel mit 15360 Bit nutzen.«[47] Denn es nützt nichts, wenn wir die Daten mit heute sicheren Schlüsseln von 2048 oder 4096 Bit Länge übertragen, sie dabei aber möglicherweise gespeichert und so lange aufgehoben werden, bis Schlüssel dieser Länge gebrochen werden können. Unabhängig vom jeweiligen Stand der Technik herkömmlicher Computer gilt laut einer Analyse des Nationalen Instituts für Standards und Technologie der Vereinigten Staaten (NIST)[48], dass außerhalb der Quantenkryptographie erst bei 15360 Bit Schlüssellänge ein ähnlich großer Rechenaufwand zum Brechen einer RSA-Verschlüsselung geleistet werden muss, wie er zum Brechen einer AES-256-Verschlüsselung erforderlich wäre.

4.9. Das Kryptomodul der KBV als ein Beispiel für hybride Verschlüsselung

Eine Anwendung, die das Prinzip hybrider Verschlüsselung verdeutlicht, ist das Kryptomodul der Kassenärztlichen Bundesvereinigung[49]. Zunächst mag es wie ein ausschließlich asymmetrisches Verfahren erscheinen, da Benutzer die symmetrischen Schlüssel nie zu Gesicht bekommen. Wer mit der Kassenärztlichen Vereinigung abrechnet, schickt dorthin am Ende eines Quartals eine Datei mit den Abrechnungsdaten. Der Name dieser Datei endet auf .CON.XKM und man kann sie entweder auf eine CD brennen und per Einschreiben an seine Geschäftsstelle der Kassenärztlichen Vereinigung schicken oder über

45 https://blog.josefsson.org/2016/11/03/why-i-dont-use-2048-or-4096-rsa-key-sizes/

46 https://www.golem.de/news/verschluesselung-was-noch-sicher-ist-1309-101457-5.html

47 https://www.heise.de/security/artikel/Kryptographie-in-der-IT-Empfehlungen-zu-Verschluesselung-und-Verfahren-3221002.html?seite=4

48 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

49 ftp://ftp.kbv.de/ita-update/KBV-Software/Kryptomodul/

ein sogenanntes sicheres Netz wie die Telematikinfrastruktur und das KV-SafeNet elektronisch einreichen.

Öffnet man diese Datei mit einem Texteditor, entdeckt man überwiegend unleserliche Zeichenfolgen. Das sind die Abrechnungsdaten, die symmetrisch per AES und einem 128 Bit langen Schlüssel verschlüsselt wurden. Sie stammen aus einer anderen Datei mit der Endung .CON, welche die Quartalsabrechnung im Klartext enthält und nicht mit eingereicht werden darf.

Den verschlüsselten Daten vorangestellt sind einige Verwaltungsdaten wie Betriebsstättennummer und Erstellungsdatum im Klartext, ein SHA-256-Hashwert[50] über die unverschlüsselten Daten zur Prüfung auf nachträgliche Beschädigungen und der einmalige, zufällig erzeugte symmetrische 128-Bit-Schlüssel. Da es zwecklos wäre, Daten zu verschlüsseln, wenn man ihren Schlüssel offen lesbar mitliefert, ist dieser per RSA-Verschlüsselung und dem 2048 Bit langen öffentlichen Schlüssel der Kassenärztlichen Bundesvereinigung verschlüsselt. Dadurch nimmt er in der Datei nicht mehr 128 Bit, sondern 2048 Bit = 256 Byte ein. Die verschlüsselte Datei .CON.XKM ist also stets länger als die unverschlüsselte Datei .CON.

Da diese Datei nicht vom Absender signiert ist, sondern nur gegen Beschädigungen ab dem Zeitpunkt ihrer Verschlüsselung gesichert, wäre es theoretisch möglich, dass ein Angreifer die Datei Datei .CON.XKM bei ihrer elektronischen Übertragung oder auf dem Postweg abfängt. Dann wären die originalen Daten zwar nicht zu entschlüsseln, jedoch könnten an ihrer Stelle komplett andere Daten mit dem öffentlichen Schlüssel der Kassenärztlichen Bundesvereinigung verschlüsselt, ihrerseits gegen nachträgliche Beschädigungen gesichert und bei der Kassenärztlichen Vereinigung zur Abrechnung im Namen der absendenden Praxis eingereicht werden.

4.10. Quantencomputer

Auch schon in heutiger Elektronik werden quantenmechanische Zustände genutzt, jedoch nur zum Speichen von Informationen. So funktionieren die in USB-Sticks verbauten Flash-Speicher über den quantenmechanischen Tunneleffekt[51]. Will man die Gesetze der Quantenmechanik zur Informationsverarbeitung nutzen, erhält man anstelle von Bits, die als Zustand entweder 0 oder 1 annehmen können, sogenannte Qubits[52], in denen die beiden Zustände sich zeitgleich überlagern.

Prinzipiell können Quantencomputer keine neuen Probleme lösen, einige Probleme werden sie allerdings deutlich schneller lösen können als herkömmliche Rechentechnik. Durch die Überlagerung von Verschränkung von Quantenzuständen können Suchprozesse effizienter gestaltet werden. Und das Brechen kryptographischer Schlüssel besteht gerade in der Suche nach dem einen Schlüssel unter vielen, der zu interpretierbaren Daten führt. Auch

50 https://de.wikipedia.org/wiki/SHA-256

51 https://de.wikipedia.org/wiki/Tunneleffekt#Flash-Speicher

52 https://de.wikipedia.org/wiki/Qubit

das Ermitteln von Primzahlen, deren Multiplikation einen bekannten öffentlichen Schlüssel ergibt, ähnelt einem Suchprozess.

Der gegenwärtige Stand der Technik ist offenbar der Quantenprozessor »Sycamore«[53] von Google, der mit 53 Qubits gleichzeitig rechnen kann. Jener Artikel der FAZ macht die folgende Rechnung auf: Um einen heute üblichen asymmetrischen Schlüssel mit 2048 Bits in seine Primfaktoren zu zerlegen, bräuchte ein Quantencomputer aus 10 Millionen Qubits 16 Stunden und würde dabei eine Leistung ähnlich der Antriebsleistung eines Hochgeschwindigkeitszugs benötigen. Leider fehlt eine Quelle für diese Rechnung und ein Kommentator unter dem Artikel bemerkt richtig die physikalische Unmöglichkeit, dass sich das angeblich auf der Fläche eines Quadratzentimeters realisieren ließe.

Naturwissenschaftlich besser fundiert erscheint mir ein Artikel[54], in dem berechnet wird, dass 2953 Qubits erforderlich wären, um einen AES-Schlüssel mit einer Länge von 128 Bit zu brechen. Zum Durchprobieren aller AES-Schlüssel mit einer Länge von 256 Bit würden 6681 Qubits benötigt. Leider ist dort der dafür benötigte Zeit- und Energieaufwand nicht unmittelbar ersichtlich.

Ein weiteres Puzzlestück liefert der Artikel »Post-quantum RSA«[55]. Quantenkryptographische Faktorisierungsverfahren wie der Shor-Algorithmus arbeiten ungleich schneller als quantenkryptographische Suchverfahren wie der Grover-Algorithmus. Das ist an der sogenannten O-Notation für deren Laufzeit zu erkennen. An der Stelle, wo beim Shor-Algorithmus[56] der natürliche Logarithmus steht, steht beim Gover-Algorithmus[57] die Quadratwurzel. Große Zahlen werden durch Logarithmieren wesentlich »kleiner gemacht« als durch Wurzelziehen. So hat die Quadratwurzel aus einhundert Milliarden sechs Stellen vor dem Komma, aber der natürliche Logarithmus aus einhundert Milliarden ist rund 25. Dass in der O-Notation des Shor-Algorithmus außerdem noch die dritte Potenz steht, ändert daran kaum etwas. Auch 25^3=15625 verfehlt sechs Stellen vor dem Komma.

Konsens scheint zu sein, dass die unteren bis mittleren Größen heute noch sicherer kryptographischer Schlüssel, darunter AES-128 und RSA-2048, in den nächsten Jahren oder Jahrzehnten durch Fortschritte herkömmlicher Rechentechnik und durch Fortschritte in der Quantenkryptographie unsicher gemacht werden könnten. Tritt diese Unsicherheit ein, wird durch jeden weiteren technischen Fortschritt der Quantencomputer die Sicherheit der Schlüssel bei asymmetrischen Verfahren wie RSA logarithmisch zur Anzahl möglicher Werte »abgeschmolzen«, bei symmetrischen Verfahren wie AES nur mit der Quadratwurzel. Für sichere asymmetrische Schlüssel käme man irgendwann bei einer Länge von Giga- und Terabyte an, die länger wäre als alle Daten, die jemals damit geschützt werden.

---

53 https://www.faz.net/aktuell/wissen/computer-mathematik/google-das-neue-zeitalter-mit-dem-quantencomputer-16452509.html

54 https://arxiv.org/pdf/1512.04965.pdf

55 https://cr.yp.to/papers/pqrsa-20170419.pdf

56 https://de.wikipedia.org/wiki/Shor-Algorithmus#Eigenschaften

57 https://de.wikipedia.org/wiki/Grover-Algorithmus

Auch dann noch werden sichere symmetrische Schlüssel in einen QR-Code passen. Ähnlich unproblematisch ist Quantenkryptographie für die zur Nachrichtenauthentifizierung verwendeten Hash-Funktionen. Zudem wird an kryptographischen Verfahren geforscht, die voraussichtlich in der Lage sein werden, auch den Rechenleistungen der Quantencomputer zu widerstehen.

Aus diesen Gründen wäre es eine unzulässige Verallgemeinerung zu behaupten, dass sich künftig die Zustände »Vertraulichkeit bewahrt« und »Vertraulichkeit gebrochen« mit immer schwerer zu bestimmenden Wahrscheinlichkeiten wechselseitig überlagern werden. Kein denkbarer technischer Fortschritt ist bekannt, der ein solches Szenario von »Schrödingers Schweigepflicht«[58] zwangsläufig auslöst. Kerckhoffs' Prinzip und die Gesetze der Mathematik stehen dem entgegen, soweit man sie anwendet.

5. Dokumentation für Programmierer

5.1. Model-View-ViewModel

BackupAes256 folgt dem Software-Entwurfsmuster Model-View-ViewModel (MVVM)[59]. Das spiegelt sich im Aufbau der Projektmappe wider, die die Unterordner »Model«, »View« und »ViewModel« enthält. Zusätzlich liegen in einem Ordner »Assets« die vom View benötigten Bilder und Symboldateien.

Unter »Model« versteht man die eigentliche Programmlogik, die so weit wie möglich unabhängig vom konkreten Benutzer sein soll. In diesem Projekt gehören zum Model unter anderem zu vergleichende Paare von Dateien, kryptographische Schlüssel und das Ansteuern der Kryptographieschnittstelle des Betriebssystem. Nicht zum Model gehört, ob der Nutzer den Typ eines kryptographischen Schlüssels auf Deutsch als »symmetrisch« oder auf Englisch als »symmetric« angezeigt bekommt, ob er mit einer Maus auf eine graphische Benutzeroberfläche klickt oder am anderen Ende eines Netzwerks vor einer Tastatur sitzt. Das Model ist in der Programmiersprache C# geschrieben.

Im Gegensatz dazu ist »View« die graphische Benutzeroberfläche mit ihren Bildern, Textfeldern und Schaltflächen. Der View soll so weit wie möglich passiv, ohne eigene Programmlogik sein und ist in der Beschreibungssprache Extensible Application Markup Language (XAML) geschrieben.

Zwischen »View« und »Model« vermittelt das »ViewModel«, das Daten aus der Programmlogik an die Benutzeroberfläche liefert, aber weder die tieferen Abläufe im Model noch den genauen Aufbau des View kennen soll. Außerdem prüft und verknüpft das ViewModel Eingabedaten und schreibt sie zurück in das Model. Es ist in der Programmiersprache C# geschrieben.

58 https://de.wikipedia.org/wiki/Schr%C3%B6dingers_Katze

59 https://de.wikipedia.org/wiki/Model_View_ViewModel

5.2. Namenskonventionen

Bislang bin ich nicht davon abzubringen, dass ich mit meiner Groß- bzw. Kleinschreibung für Bezeichner von den Microsoft-Richtlinien[60] abweiche. Die zugehörige Warnung »IDE1006« habe ich deshalb in den Projekteigenschaften unterdrückt. Unverändert gilt, dass Namen von Methoden (= Teilprogrammen ohne Rückgabewert), Funktionen (= Teilprogrammen mit Rückgabewert) und Klassen (= Abbildung realer Objekttypen in Software) mit einem Großbuchstaben beginnen.

Variablen stelle ich ihren Datentyp in Form eines Kürzels aus einem oder zwei Kleinbuchstaben voran. Das hat den Vorteil, dass ich für gleiche Sachverhalte nicht verschiedene Variablennamen erfinden muss, sondern z.B. *iBits* schreiben kann für die interne Zahlendarstellung und *sBits* für die Darstellung derselben Information als Zeichenkette auf der Benutzeroberfläche. *ltKeys* bezeichnet die programminterne Liste aller Schlüssel, *blKeys* die an die Benutzeroberfläche gebundene *BindingList* aller zum gewählten Verschlüsselungsverfahren passenden Schlüssel. Kann ich mich nicht an den Namen eines von mir vergebenen Bezeichners erinnern, ist dessen Datentyp dennoch meist aus dem Kontext ersichtlich, so dass der entsprechende Anfangsbuchstabe meine Suche eingrenzt. Die von mir gewählten Kürzel habe ich im Folgenden zusammengestellt.

Datentyp	**Typname**	**Kürzel**
Konstante eines unten aufgeführten Typs, z.B. double	const double	cdBezeichner
hardwarenah		
Wahrheitswert	bool	isBezeichner
Byte	byte	bBezeichner
Zahlen		
selbst definierte Aufzählung von Zuständen	enum nBezeichner	eBezeichner
16-Bit-Ganzzahl mit Vorzeichen	short	hBezeichner
32-Bit-Ganzzahl mit Vorzeichen	int	iBezeichner
64-Bit-Ganzzahl mit Vorzeichen	long	kBezeichner

60 https://docs.microsoft.com/de-de/dotnet/standard/design-guidelines/capitalization-conventions

Datentyp	Typname	Kürzel
16-Bit-Ganzzahl ohne Vorzeichen	ushort	tBezeichner
32-Bit-Ganzzahl ohne Vorzeichen	uint	uBezeichner
64-Bit-Ganzzahl ohne Vorzeichen	ulong	vBezeichner
32-Bit-Fließkommazahl	float	fBezeichner
64-Bit-Fließkommazahl	double	dBezeichner
Text		
einzelnes Zeichen	char	cBezeichner
Zeichenkette	string	sBezeichner
komplexe Objekte		
Array = »Feld« eines oben aufgeführten Typs, z.B. double	double[]	adBezeichner
Delegate Command	ICommand	dcBezeichner
Datum und Zeit	DateTime	dtBezeichner
Ergebnis einer LINQ-Abfrage	IEnumerable<>	qyBezeichner
Fehlerzustand	Exception	exBezeichner
Liste	List<>	ltBezeichner
an die Benutzeroberfläche gebundene Liste	BindingList<>	blBezeichner
Warteschlange	Queue<>	quBezeichner
Wörterbuch	Dictionary<>	dyBezeichner

5.3. Algorithmen

Um mich zunächst wieder dem Programmieren zu widmen, bevor ich weiter darüber schreibe, erwähne ich hier vorerst nur einen exzellenten Übersichtsartikel[61] zum Umgang mit kryptographischen Schlüsseln.

61 https://stephenroughley.com/2019/06/09/concepts-of-compliant-data-encryption/