Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Ernst-Günter Giessmann

Beschreibung des Verfahrens für ein

Elektronisches Gutachten auf Papier

Ist der Begriff „Elektronisches Gutachten auf Papier“ nicht ein Oxymoron? Widersprüchlich in sich selbst? Da es sich bei einem Gutachten um ein Dokument mit gewisser rechtlicher Bedeutung handelt, betrachten wir zuerst eine Definition. Ein *Elektronisches Dokument* ist nach der Europäischen eIDAS-Verordnung [SV] „jeder in elektronischer Form, insbesondere als Text-, Ton-, Bild- oder audiovisuelle Aufzeichnung gespeicherte Inhalt“. Das ist nun wirklich sehr allgemein und doch sind die wesentlichen Grundlagen schon vorhanden.

Beginnen wir mit der *elektronischen Form*. Alles, was sich in so einer Form darstellen lässt, muss sich in einzelne Bestandteile mit einer diskreten Beschreibung zerlegen lassen. Das ist charakteristisch für elektronische Dokumente. Egal ob in Bits oder Bytes, Strich- oder QR-Codes, für alle gilt, dass die „elektronische Form“ zerlegt werden kann und dass die „Atome“ dieser Zerlegung aus einer endlichen Menge bestehen. Aus technischer Sicht kann man diese Zerlegung auch *Digitalisierung* nennen, denn die Elemente einer endlichen Menge lassen sich durch Bit-Folgen fixierter Länge beschreiben.

Dann ist da als nächstes die *Speicherung durch Text-, Ton-, Bild- oder audiovisuelle Aufzeichnung*. Elektronische Dokumente können also auf verschiedene Art und Weise gespeichert werden. Es muss nur zweifelsfrei klar sein, wie sich aus den gespeicherten wieder die digitalen Daten des elektronischen Dokuments ergeben. Normalerweise ergibt sich das aus der Art der Speicherung und das geht auch über die aufgezählten Beispiele hinaus. Beispielsweise werden auf einer CD die digitalisierten Inhalte durch *Pits* und *Lands* dargestellt[1]. Diese standardisierte Kodierung ist in der Aufzählung nicht genannt, aber das waren ja auch nur Beispiele.

Und schließlich gibt es noch den Verweis auf die *gespeicherten Inhalte*. Mit dem elektronischen Dokument, genauer gesagt, den digitalisierten Daten, muss festgelegt werden, wie aus den eindeutigen digitalen Daten unverändert auf ein und die gleiche Art und Weise ein bestimmtes lesbares Textdokument, ein verständliches hörbares Ton-Dokument, ein bestimmtes erkennbares Bild oder ein entsprechendes audiovisuelles Dokument entsteht. Dies ist der bei der Digitalisierung eindeutig festgelegte *Dateityp*, der unabhängig von der Art der Speicherung ist.

Zu einem elektronischen Dokument gehören immer diese beiden Schnittstellen, die zu dem jeweiligen Speichermedium und die für die visuelle oder auch audiovisuelle Wahrnehmung. Während die zum Speichermedium in der Regel immer eindeutig und durch das konkrete Speichermedium selbst bestimmt sind, ist der Datentyp nicht so eindeutig. Betrachten wir dazu ein sehr einfaches Beispiel.

---

*2020-07-29 URL: `www.informatik.hu-berlin.de/~giessman/ElektronischeGutachten.pdf`

[1] Unterschiedliche lange Vertiefungen auf der CD-Spur. Dabei wird aber noch ein Datenbyte in 14 oder 16 Kanalbits transformiert (EFM). Außerdem wird die „1“ durch den Wechsel von *Pit* zu *Land* oder umgekehrt kodiert. Findet in einer Streckeneinheit kein Wechsel statt, ist der Wert „0“.

Die URL *https://informatik.hu-berlin.de* kann als Folge von ASCII-Zeichen durch die Folge der dreißig Bytes

68 74 74 70 73 3A 2F 2F 69 6E 66 6F 72 6D 61 74 69 6B 2E 68 75 2D 62 65 72 6C 69 6E 2E 64 65

dargestellt werden. Sie kann als diese Zeichenfolge ausgedruckt, angezeigt oder auch vorgelesen werden. Wenn man sie nicht als Textzeile, sondern als URL interpretiert, würde man darunter die aktuelle Web-Seite verstehen, sie ausdrucken, anzeigen oder vorlesen lassen. Das wären dann zwei unterschiedliche Formen menschlicher Wahrnehmung, einmal als ASCII-Text oder als URL, je nachdem, welcher Datei-Typ vorgegeben wurde.

Für die Speicherung auf einem Medium kann sie als die obige Byte-Folge

`https://informatik.hu-berlin.de`

aber auch base64-kodiert als Folge

`aHR0cHM6Ly9pbmZvcm1hdGlrLmh1LWJlcmxpbi5kZQo=`

als QR-Code

oder sogar als Tonfolge im Morse-Code

kodieren und dann als Ladezustände in Halbleiterzellen speichern, als *Pits* und *Lands* einer CD darstellen oder als helle und dunkle Flächen ausdrucken. Daraus kann man sie jederzeit wieder verlustfrei in die obige Folge der ASCII-Zeichen zurückverwandeln[2]. Das wären alles unterschiedliche Beispiele für Formen der Speicherung des elektronischen Dokuments.

Gerade beim QR-Code sieht man, dass das elektronische Dokument (hier die URL des Instituts) auch als Ausdruck auf Papier gespeichert sein kann. Bei der Text-Zeile oder base64-kodiertem Text ist es nicht ganz so einfach, weil es dafür eigentlich keine Lesegeräte gibt. Wenn man allerdings, wie in dem obigen Beispiel, einen standardisierten OCR-Font verwendet, dann hat man mit dem Ausdruck auch schon eine Speicherung der URL als elektronisches Dokument. Es kann bei entsprechender Auflösung in der Regel fehlerfrei von einem Scanner gelesen werden. Aus der elektronischen PDF-Datei kann man jedoch sowohl die Textzeile als auch den base64-kodierten Text direkt kopieren.

Der Anhang zu diesem Artikel enthält auf Seite 4 die elektronische Version dieses Artikels als LaTeX-Datei, aus der ein PDF-Dokument erzeugt wurde. Sicherheitshalber wurde dabei eine Fontgröße gewählt, die fehlerfrei beim Scannen erkannt wird. Bei guter Qualität des Ausdrucks könnte sie aber auch noch kleiner sein. In der PDF-Datei ist diese elektronische Version im OpenPGP-Format (siehe [PGP]) als ASCII *armor* gespeichert und kann von dort durch Kopieren und Einfügen in eine einfache Textdatei als `signedDocument.pgp` auch bei noch geringerer Fontgröße entnommen werden.

Die Abgabe von Gutachten im elektronischer Form, etwa als PDF, ist eigentlich nicht zulässig und nur unter besonderen Bedingungen, wie etwa im Fall der Corona-Krise, erlaubt. Authentisch ist nur das unterschriebene Dokument in Schriftform. Soll es als elektronisches Dokument die gleichen Anforderungen wie die Schriftform erfüllen, muss es elektronisch unterschrieben werden und der Unterzeichner muss eindeutig benannt sein. Dazu analysieren wir die angehängte elektronische Form und zeigen, dass es ein elektronisch signiertes Dokument ist, dessen Unterzeichner auch

[2] Das gilt auch fast für den Morse-Code, obwohl dessen Dekodierung nur `https//informatik.hu=berlin.de` ergibt.

zweifelsfrei festgestellt werden kann.

Hat man den gesamten entsprechenden Text als Datei `signedDocument.pgp` gesichert, erhält man bei Anwendung des Kommandos

```
gpg --output signedDocument.tex --verify signedDocument.pgp
  gpg: invalid armor header:
     owJ4nLV8zY8bSXZnz9oGxloIMGxg4TVgIFrTsqQWk2Qmv0tbY7OKLKmk0mdJKrVEzUwyMzKZY
     ...wIAvY8D+D7z73ovID7JYGmnRW91VJDMjI168z997L6g/uvwLn3z71358qv/nf/zjX/mX\n
  gpg: Signature made Wed Jul 29 12:30:42 2020 CEST
  gpg:                using ECDSA key 222E67EA2AA253A967408ABE9F7A3119DF9110EB
  gpg: Good signature from "Ernst G Giessmann <giessmann@informatik.hu-berlin.de>"
```

zwar eine Warnung, aber die Signatur vom `2020-07-29 10:30:42 UTC` ist gültig und der LaTeX-Quelltext `signedDocument.tex` wird korrekt erzeugt. Wenn der zugehörige öffentliche Schlüssel[3] im Schlüsselring vorhanden ist, werden der Name und die geprüfte E-Mail-Adresse angezeigt. Alle Bedingungen für ein elektronisches Dokument sind erfüllt.

Der Fehler entsteht durch das Kopieren. Zwar wird im PDF-Dokument scheinbar eine für die korrekte OpenPGP-Nachricht erforderliche Leerzeile nach den Kopfzeilen angezeigt, aber sie wird beim Kopieren nicht in die Datei `signedDocment.pgp` übernommen. Sie ist auch tatsächlich nicht vorhanden, und erst durch nachträgliches Einfügen dieser fehlenden Leerzeile wird die Signaturverifikation fehlerfrei.

Das LaTeX-Dokument importiert als Logo eine externe Datei, die jedoch nicht im elektronischen Dokument `signedDocument.pgp` gespeichert wird. Ist sie bei der Übersetzung des Quelltextes mit `pdflatex` nicht vorhanden, wird eine URL angezeigt[4], an der man entsprechende Bilddateien findet. Speichert man unter dem Dateinamen `Huberlin-logo.png` eine Version in passender Auflösung, ergibt das extrahierte Dokument bei der Übersetzung die gewünschte und zu diesem Papierdokument äquivalente elektronische Version mit gültiger Signatur.

Literatur

[SV] Verordnung (EU) Nr. 910/2014 des Europäischen Parlaments und des Rates vom 23. Juli 2014 über elektronische Identifizierung und Vertrauensdienste für elektronische Transaktionen im Binnenmarkt und zur Aufhebung der Richtlinie 1999/93/EG, 2014-07-23

[PGP] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, R. Thayer: OpenPGP Message Format, RFC 4880, IETF Standards Track, 2007-11

[3] `https://keys.openpgp.org/vks/v1/by-fingerprint/222E67EA2AA253A967408ABE9F7A3119DF9110EB`

[4] `https://de.wikipedia.org/wiki/Datei:Huberlin-logo.svg`

Im folgenden Text ist in elektronischer Form (OpenPGP-Format [PGP]) das Dokument gespeichert, aus dem mit LaTeX das hier ausgedruckte PDF-Dokument erzeugt wurde[5].

-----BEGIN PGP MESSAGE-----

[illegible]
=ilj3
-----END PGP MESSAGE-----

Das elektronische Dokument im Detail

Der Anhang ist eine OpenPGP-Nachricht, die im Allgemeinen aus mehreren (auch verschachtelten) so genannten Paketen bestehen kann. Wir zeigen hier, wie man den LaTeX-Quelltext aus der OpenPGP-Nachricht extrahieren und die Signatur verifizieren kann. Wir stellen dazu als erstes die Binärdaten der base64-kodierten Nachricht wieder her.

```
sed '/^[-=]/d;' signedDocument.pgp | base64 -d > signedDocument.bin
```

OpenPGP-Pakete sind wie in der ASN.1-Syntax als TLV-Strukturen (*Tag-Length-Value*) aufgebaut. Das erste Byte der Binärdaten `A3=2#1010 0011` enthält sowohl das Tag `8=2#1000` (Bit 3–6), als auch die Längen-Information `indeterminate=0x03` (Bit 7–8). Das vorliegende Paket vom Typ 8 ist ein *Compressed Data Packet* und besteht nur aus dem Algorithmus-Byte und den dann folgenden komprimierten Daten. Dabei entspricht das Byte `0x02` der Kompression mit `zlib` (vgl. `https://tools.ietf.org/html/rfc1950`). Der Vollständigkeit halber sei noch erwähnt, dass das Byte `0x01` der Kompression mit *deflate* (`https://tools.ietf.org/html/rfc1951`) und das Byte `0x03` der Kompression mit `bzip2` (`http://www.bzip.org`) entspricht.

Einen Dekompressionsalgorithmus `zpipe.c` (Byte `0x02`) findet man im Gesamt-Paket[6] der `zlib` im Unterverzeichnis `examples`.

Gibt man beim Signieren den Kompressionsalgorithmus nicht an[7], wird *deflate* nach RFC 1951 (Byte `0x01`) verwendet. Allerdings fehlen dann bei den komprimierten Daten die Header-Bytes und die Adler32-Prüfsumme. In diesem Fall kann man jedoch das einfache *inflate*-Programm `pufftest.c` nutzen, das man im Verzeichnis `contrib/puff` des genannten `zlib`-Pakets findet.

[5] Details findet man auf `https://www.informatik.hu-berlin.de/~giessman/ElektronischesGutachten`.

[6] `https://github.com/madler/zlib/releases/tag/v1.2.11`

[7] Entweder als Option `--compress-algo 2` oder in der Konfigurationsdatei `gpg.conf`.

Das Paket der Datei `signedDocument.bin` kann man jetzt mit folgendem Kommando auspacken[8]:

```
tail -c+3 signedDocument.bin | zpipe -d > signedDocument.dec
```

Die dekomprimierten Daten enthalten insgesamt drei Pakete. Das erste Paket beschreibt dabei in 13 Bytes alles das, was für die Signaturprüfung notwendig ist, insbesondere den Hash-Algorithmus SHA-256 (`0x08`). Das zweite (*literal data packet*) enthält den ursprünglichen Dateinamen und den gesamten LaTeX-Quelltext.

```
gpg -v --list-packets signedDocument.dec
  # off=0 ctb=90 tag=4 hlen=2 plen=13
  :onepass_sig packet: keyid 9F7A3119DF9110EB
        version 3, sigclass 0x00, digest 8, pubkey 19, last=1
  # off=15 ctb=ad tag=11 hlen=3 plen=20026
  :literal data packet:
        mode b (62), created 1596018642, name="ElektronischeGutachten.tex",
        raw data: 19994 bytes
  # off=20044 ctb=88 tag=2 hlen=2 plen=117
  :signature packet: algo 19, keyid 9F7A3119DF9110EB
        version 4, created 1596018642, md5len 0, sigclass 0x00
        digest algo 8, begin of digest 68 35
        hashed subpkt 33 len 21
          issuer fpr v4 222E67EA2AA253A967408ABE9F7A3119DF9110EB
        hashed subpkt 2 len 4 (sig created 2020-07-29)
        subpkt 16 len 8 (issuer key ID 9F7A3119DF9110EB)
        data: 5CE3EC00779E117EB08FCC6EA9AEA8B695AA82BF1189591A5B77DC211CB7459D
        data: 5025C0FB23E4E9308DDF3FFE31E99019F7CFC13B525972D7F0E97B8F86C50110
```

Das dritte, das Signaturpaket, besteht aus vier Informations-Bytes, den signierten und unsignierten Attributen, den ersten beiden Bytes `68 35` des Hash-Werts und einer ECDSA-Signatur dieses Hash-Wertes. Dieser wird über den LaTeX-Quelltext, die Informations-Bytes und die beiden signierten Attribute (*fingerprint* und *signatur creation time*)

```
  04 00 13 08      : version  sig-type sig-algo hash-algo
  001d             : signed attributes
   16 21           : issuer key fingerprint
    04 222e67ea2aa253a967408abe9f7a3119df9110eb
   05 02 5f214fd2  : signature creation time
```

sowie weiteren sechs Trailer-Bytes mit der Längeninformation gebildet:

```
  04 ff 00000023   : version 4 trailer : length 23
           : 0x04(info bytes) + 0x02(length bytes) + 0x1d (signed attributes)
```

Die beiden Bytes `68 35` des Hash-Werts dieser Daten

```
openssl sha256 -binary dtbs.bin | xxd -p -c32
  68353b221095706cf3ec4633fadcc5dab3b25f9062bea89f33777954aa682960
```

sind bereits im Signaturpaket angekündigt worden.

Für diesen Hash-Wert kann man mit dem ECDSA-Schlüssel

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE0PGk0fLdZspbiNQD9q7McjpAiR4F
4vkfxBvhv5gjqW1E+Wd6IZ7PJIt/B0KC64qA2/iCubmXp8gibuSmr0+jCw==
-----END PUBLIC KEY-----
```

dann leicht die Signatur prüfen. Die detaillierten Kommandos findet man im LaTeX-Quelltext.

[8] Für die *deflate*-komprimierte Nachricht entsprechend `tail -c+3 signedDocument.bin | pufftest`.