HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Frau
Kristina Martin

Extraktion von Passworthashes und Ermittlung von Passwörtern aus Browserapplikationen im Rahmen der Post-Mortem-Analyse

2018

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Extraktion von Passworthashes und Ermittlung von Passwörtern aus Browserapplikationen im Rahmen der Post-Mortem-Analyse

Autorin:
Kristina Martin

Studiengang:
Allgemeine und digitale Forensik

Seminargruppe:
FO15w3-B

Betreuer der Hochschule::
Prof. Dr. rer. nat. Christian Hummert

Betreuer am Landeskriminalamt Thüringen:
Dipl.-Inf. Andreas Sommer

Mittweida, August 2018

Faculty **Applied Computer Sciences & Biosciences**

BACHELOR THESIS

Extraction of Password Hashes and Decryption of Passwords stored by Browser Applications on the Basis of a Post-Mortem-Analysis

Authorin:
Kristina Martin

Study Programme:
General and Digital Forensic Science

Seminar Group:
FO15w3-B

First Referee::
Prof. Dr. rer. nat. Christian Hummert

Second Referee:
Dipl.-Inf. Andreas Sommer

Mittweida, August 2018

Bibliografische Angaben

Martin, Kristina: Extraktion von Passworthashes und Ermittlung von Passwörtern aus Browserapplikationen im Rahmen der Post-Mortem-Analyse, 95 Seiten, 49 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Bachelorarbeit, 2018

Referat

Die vorliegende Arbeit erläutert, wo Browser gespeicherte Nutzerpasswörter ablegen und wie diese zu entschlüsseln sind. Die Untersuchungen beziehen sich auf die vier derzeit meistgenutzten Browser Deutschlands, namentlich Google Chrome, Mozilla Firefox, Internet Explorer und Microsoft Edge. Dabei wird auf diverse Verschlüsselungs-, Hashing- und Kodierungsverfahren eingegangen, die bei der Ver- und Entschlüsselung von Bedeutung sind. Insbesondere die Windows-eigene Data Protection API spielt eine übergeordnete Rolle bei der sicheren Speicherung von Passwörtern.
Weiterführend wird die Entwicklung eines Programmes beschrieben, das die Browser-Passwörter per Knopfdruck aus Dateien bzw. Datenbanken von einem forensischen Abbild extrahiert und entschlüsselt. Die Anwendung zeichnet sich durch eine einfache Bedienung und eine geringe Laufzeit aus und kann so besonders bei polizeilichen Ermittlungen von Nutzen sein.

Bibliographic Information

Martin, Kristina: Extraction of Password Hashes and Decryption of Passwords stored by Browser Applications on the Basis of a Post-Mortem-Analysis, 95 pages, 49 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer Sciences & Biosciences

bachelor thesis, 2018

Abstract

This bachelor thesis presents where browsers store user passwords and how they can be decrypted. The analysis refers to the four most popular browsers in Germany, namely Google Chrome, Mozilla Firefox, Internet Explorer and Microsoft Edge. In the course of this some cryptographical processes like encryption, hashing and coding will be explained, which are used to encrypt or decrypt the given passwords. Especially the windows-owned Data Protection API plays a superordinate role in secure storage of login data.
Furthermore a program will be introduced, which is able to extract and decrypt browser passwords from files and databases inside a forensic image by touching a button. The application is characterized by a simple operation and a short run time, so it can be useful for police investigation.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

Listings

IV. Abkürzungsverzeichnis

3DES	Triple Data Encryption Standard
AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange, auch US-ASCII
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
BLOB	Binary Large Object
CBC	Chipher-block Chaining
CER	Canonical Encoding Rules
CFB	Cipher Feedback
DB	Datenbank
DEA	Data Encryption Algorithm
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DPAPI	Data Protection Application Progrmming Interface
ECB	Electronic Codebook
FF	Mozilla Firefox
GUID	Global Unique Identifier
HMAC	Keyed-Hash Message Authentication Code
HTTP	Hypertext Transfer Protocol
IE	Internet Explorer
ipad	inner pad (bei HMAC)
IV	Initialisierungsvektor
kB	Kilobyte
LSA	Local Security Authority
MAC	Message Authentication Code
MB	Megabyte
MD5	Message-Digest Algorithm 5
MD6	Message-Digest Algorithm 6
OFB	Output Feedback
opad	outer pad (bei HMAC)
PBKDF2	Password Based Key Derivation Function 2
PRD	Password Reset Disk

RPC ..... Remote Procedure Call
RSA ..... Rivest, Shamir, Adleman (Erfinder der RSA-Verschlüsselung)
S-Box ..... Substitutions-Box
SHA ..... Secure Hash Algorithm
SID ..... Security Identifier
SQL ..... Structured Query Language
SSL ..... Secure Sockets Layer
URL ..... Uniform Resource Locator
UTF ..... Unicode Transformation Format
VHD ..... Virtual Hard Disk
WPR ..... Windows Password Recovery

1 Einleitung

1.1 Motivation

Online-Dienste spielen in der heutigen Zeit eine große Rolle, vor allem E-Mail-Anbieter, Soziale Netzwerke, Clouds und Shopping-Seiten sind beliebt und fast jeder legt bereitwillig einen eigenen Account an, wenn er dazu aufgefordert wird. Nicht selten enthalten diese Konten sensible Daten, wie Geburtstage, Anschriften, private Bilder und Nachrichten. Aus diesem Grund sind die Accounts auch in der Forensik von großem Nutzen und können bei Ermittlungen entscheidende Hinweise liefern. Leider geraten gerade hier herkömmliche Datensicherungen an ihre Grenzen, denn die Daten liegen irgendwo auf riesigen Servern, auf die man nicht ohne weiteres zugreifen kann. Gibt es Hinweise auf Online-Dienste, die möglicherweise weitere Spuren enthalten, ist es also unumgänglich, die Zugangsdaten in Erfahrung zu bringen. Befinden sich diese ausschließlich im Kopf eines Beschuldigten, ist man auf dessen Kooperation angewiesen. Zum Glück ist das aber nur noch selten der Fall. Laut einer aktuellen Umfrage nutzen mehr als ein Drittel der Deutschen für jeden Online Dienst ein anderes Passwort [Bil18], wie aus Abbildung 1.1 hervorgeht.

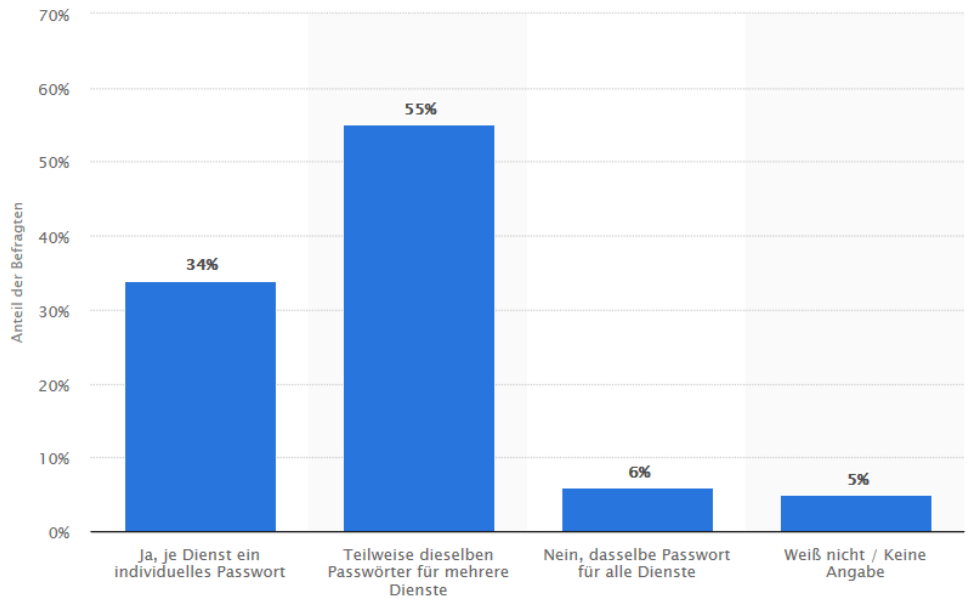


Abbildung 1.1: Umfrageergebnisse: Nutzen Sie unterschiedliche Passwörter für unterschiedliche Dienste? [Bil18]

Bei zwei oder drei verschiedenen Accounts mag es kein Problem sein, sich die Zugangsdaten zu merken, aber in der heutigen Zeit ist es schlichtweg normal, mehrere E-Mail-

Konten von unterschiedlichen Anbietern zu haben. Dazu kommen eine Hand voll sozialer Netzwerke, Online-Versandhäuser, Jobportale, Streaming-Dienste, Partnerbörsen, Online-Speicher, Ebay, Amazon und viele mehr. Wer jetzt tatsächlich für jedes seiner Konten ein anderes Passwort verwendet, kann unmöglich alle Nutzernamen und Kennwörter im Kopf haben. Will man nun nicht zu Zettel und Stift greifen, bieten alle geläufigen Browser bei jeder neuen Anmeldung an, die Login-Daten zu speichern, um das Login-Fenster beim nächsten Aufruf automatisch zu füllen.
Was für den Benutzer nun eine willkommene Unterstützung darstellt, ist für einen Forensiker eine wahre Goldgrube, denn wenn ein Browser etwas speichert, bedeutet das gleichzeitig, dass die Daten zwar verschlüsselt, aber dennoch lokal auf der Festplatte abgelegt und somit bei einer Datensicherung automatisch mit erfasst werden. Weiß man nun, wie die Informationen zu entschlüsseln sind, ist das die Tür zu vielen weiteren Spuren. So können beispielsweise Mails aus einem G-Mail-Konto oder ein Bild aus Google Drive unter Umständen helfen, Straftaten aufzuklären.

1.2 Zielstellung

Es existieren bereits diverse Tools, um die gespeicherten Browser-Passwörter zu entschlüsseln. Oft sind diese jedoch darauf konzipiert, am Originalsystem zu arbeiten und nicht an einem Image. Einige Programme können zwar auch Passwörter von anderen Nutzerprofilen oder Rechnern auswerten, jedoch müssen dafür meist für jeden Browser einzeln erst diverse Pfade angegeben werden, was relativ umständlich ist. Allgemein sind viele Passwort-Recovery-Programme, wie beispielsweise die Tools von Nirsoft, eine Black-Box, das heißt, man kann nicht nachvollziehen, welche Prozesse intern ablaufen.
Ziel dieser Arbeit ist es deshalb, die Verschlüsselungsprozesse im Detail zu untersuchen, um schließlich eine Anwendung zu entwerfen, die Browser-Passwörter auf Knopfdruck aus einem forensischen Image extrahiert und entschlüsselt.

2 Grundlagen

2.1 Kodierung, Verschlüsselung und Hashverfahren

Ein Browser oder auch jedes andere etablierte Programm legt keines seiner gespeicherten Passwörter als Klartext ab. Um die Daten hinreichend zu schützen, werden Kodierungs-, Verschlüsselungs- und Hashing-Verfahren angewendet bzw. häufig eine Kombination dieser.

Kodierung leistet in diesem Sinne keinen wirklichen Beitrag zur Sicherheit, sondern dient lediglich dazu, dass beispielsweise Binärdaten in eine standardisierte Form gebracht werden, so dass Applikationen diese verarbeiten können.

Hash-Funktionen hingegen sind mathematische Einweg-Funktionen, „die als Eingabe eine Zeichenfolge nahezu beliebiger Länge Z entgegennehmen und daraus einen Ausgabewert fester Länge, den Hashwert $H(Z)$, berechnen“ [Paw17]. Solche Funktionen zeichnen sich dadurch aus, dass die Ableitung des Hashwertes aus der eingegebenen Zeichenfolge sehr schnell abläuft, aber umgekehrt, die Ermittlung des Eingabewertes aus dem Hash nicht effizient bestimmbar ist. Neben dieser Einweg-Eigenschaft ist eine gute Hashing-Funktion außerdem durch eine Kollisionsresistenz gekennzeichnet. Das bedeutet, dass für die Zeichenkette Z effektiv keine andere Zeichenkette Z‘ erzeugt werden kann, für die gilt $H(Z) = H(Z‘)$ [Paw17]. Beispiele für bekannte Hash-Verfahren sind unter anderem MD5, MD6, SHA, oder Whirlpool.

So sind nun weder Kodierung noch Hashing allein für die Sicherung von Passwörtern sinnvoll, denn Kodierung ist ohne Probleme wieder zu dekodieren und Hashing letztlich nicht effektiv rückgängig zu machen. Als Vor- und Nachbereitungsschritte spielen sie aber eine wichtige Rolle. Den eigentlichen Kern der sicheren Passwortspeicherung bilden Verschlüsselungsverfahren, wie zum Beispiel DES, AES oder Blowfish.

Verschlüsselung fordert als Eingabe einen Klartext und einige Parameter, die sich meist von Verfahren zu Verfahren unterscheiden. Auf jeden Fall im Parametersatz enthalten ist jedoch ein Schlüssel bzw. auch mehrere Schlüssel. Werden Daten mit diesem Schlüssel chiffriert, sind sie für Außenstehende nicht lesbar. Nur Eingeweihte, also diejenigen, die über den richtigen Schlüssel verfügen, sind in der Lage, die Daten wieder zu entschlüsseln.

Bei allen Browsern spielt die Verschlüsselung mit Triple DES (3DES) oder AES 256 und das Hashing mit SHA1 eine zentrale Rolle. Ergänzt werden die Algorithmen zur sicheren Speicherung der Logindaten mit Base64- und ASN.1-Kodierung für Firefox und dem Hash-Verfahren HMAC und PBKDF2 für Chrome, Internet Explorer und Edge.

2.1.1 Base64

Die Base Kodierung wandelt beliebige Daten in einen ausschließlich aus druckbaren Zeichen bestehenden String um. Diese Form der Darstellung wird vor allem dann gebraucht, wenn Anwendungen auf US-ASCII beschränkt sind oder die Daten mittels einfachem Texteditor bearbeitet werden sollen. Welche Zeichen verwendet werden, legt das zugrunde liegende Alphabet fest. Base64 verwendet als Alphabet alle Groß- und Kleinbuchstaben von A bis Z, die Zahlen 0 bis 9 sowie „+“ und „/“, also insgesamt 64 Zeichen, wie Tabelle 2.1 zeigt [Jos06]:

Wert	Kodierung	Wert	Kodierung	Wert	Kodierung	Wert	Kodierung
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Tabelle 2.1: Base 64 Alphabet [Jos06]

Es existiert auch ein „URL and Filename safe“ Alphabet, das statt „+“ und „/“ Minus (-) und Unterstrich (_) benutzt [Jos06].

Bei der Konvertierung werden die Eingabedaten in 24-bit Gruppen (3-mal 8 Bit) eingeteilt. Jede Gruppe wird nun in 4 Untergruppen à 6 Bit aufgegliedert. Für jede Untergruppe bestimmt Base64 dann das entsprechende Zeichen aus dem Alphabet [Jos06]. Die folgende Abbildung veranschaulicht das Verfahren.

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	...
Bit-Wert	1	1	0	1	0	0	0	1	0	0	1	1	1	0	1	0	1	1	1	0	0	1	0	1	0	0	1	1	...
	erste 8-Bit Gruppe								zweite 8-Bit Gruppe								dritte 8-Bit Gruppe								...				
	erste 6-Bit Gruppe						zweite 6-Bit Gruppe						dritte 6-Bit Gruppe						vierte 6-Bit Gruppe						...				
neuer Wert	52						19						43				37								...				
neues Zeichen	0						T						r				l								...				

Abbildung 2.1: Konvertierung nach Base 64 (eigene Quelle)

Es kann vorkommen, dass am Ende der Eingabedaten keine vollständigen 24-Bit mehr verfügbar sind, sondern nur 16 oder 8. In diesem Falle werden Bits mit dem Wert 0 angehängt, bis die nächste 6-Bit Gruppe vollständig ist. Bei 16 Bit wird zusätzlich ein „=“ am Ende der Ausgabe angefügt, bei 8 Bit zwei „=“ [Jos06].

2.1.2 ASN.1-DER

Abstract Syntax Notation One (ASN.1) definiert eine Notation, um abstrakte Datenstrukturen in eine für Applikationen interpretierbare Form zu bringen. ASN.1 generiert eine Transfer-Syntax, in der vor allem Datentypen definiert werden können. Dabei wendet ASN.1 verschiedene Kodierungsregeln an. Dazu gehören Basic Encoding Rules (BER), Canonical Encoding Rules (CER) und Distinguished Encoding Rules (DER). DER findet insbesondere Anwendung, wenn kleine Werte kodiert werden sollen.
ASN.1 speichert zu einer Datenstruktur einen Identifier, der den Datentyp definiert, die Länge der Daten und schließlich die Daten selbst. Falls es eine Datenstruktur erfordert, wird noch eine Kennzeichnung angehängt, die das Ende der Daten markiert. ASN.1 erlaubt auch eine Schachtelung von Datentypen. So kann zum Beispiel ein Typ „Set“ eine String-Struktur und eine Integer-Struktur enthalten [ITU03]. Eine Tabelle mit Simple und Structured Types befindet sich im Anhang A.

2.1.3 Data Encryption Standard (DES)

DES (Data Encryption Standard) ist eine Block-Chiffre, die in den 1970er Jahren als DEA (Data Encryption Algorithm) entwickelt wurde. Es handelt sich um einen symmetrischen Algorithmus, der Daten in 64-Bit-Blöcken verschlüsselt. [Micd]
DES verwendet einen 64-Bit-Schlüssel, von dem aber nur 56 Bit effektiv für die Verschlüsselung genutzt werden, die anderen 8 Bit bilden eine Parity. Bei der Verschlüsselung werden aus dem 56-Bit-Key 16 Rundenschlüssel à 48 Bit permutiert. Die zu verschlüsselnden Daten durchlaufen dementsprechend 16 Runden. Zu Beginn wird jeder Block nochmals in zwei 32-Bit-Blöcke geteilt, in einen linken und einen rechten Block. [AZZ+10] Einer der beiden 32-Bit-Blöcke, zum Beispiel der rechte, durchläuft

anschließend eine Rundenfunktion. Dabei wird er auf 48 Bit Länge erweitert und mit einem der 16 Rundenschlüssel xor verknüpft. Der entstandene 48-Bit-Block wird nun wiederum in 8 mal 6 Bit aufgeteilt. Für jeden 6-Bit-Block steht eine eigene Substitutions-Box (S-Box) bereit, in der die Eingabe-Bits mit definierten Ausgabewerten ersetzt werden, so dass am Ende nur noch 4 Bit als Ergebnis der Substitution resultieren. Jede der 8 S-Boxen liefert eine Ausgabe von 4 Bit, so dass insgesamt wieder ein Block von 32 Bit zusammengesetzt werden kann. Zum Abschluss der Runde durchläuft der Block noch eine Permutation, also eine Art Durchmischung innerhalb des Blocks und wird mit dem anderen 32-bit-Block xor verknüpft. Für die nächste Runde wird das Ergebnis der xor Verknüpfung als neuer rechter Block angenommen und der alte rechte 32-Bit-Block wird zum neuen linken Block. Dieser Austausch erfolgt nach jeder Runde, ausgenommen der letzten. Zum besseren Verständnis sind in den Abbildungen 2.2 und 2.3 die Rundenfunktion und die Abfolge der Runden schematisch dargestellt. [MOV96]

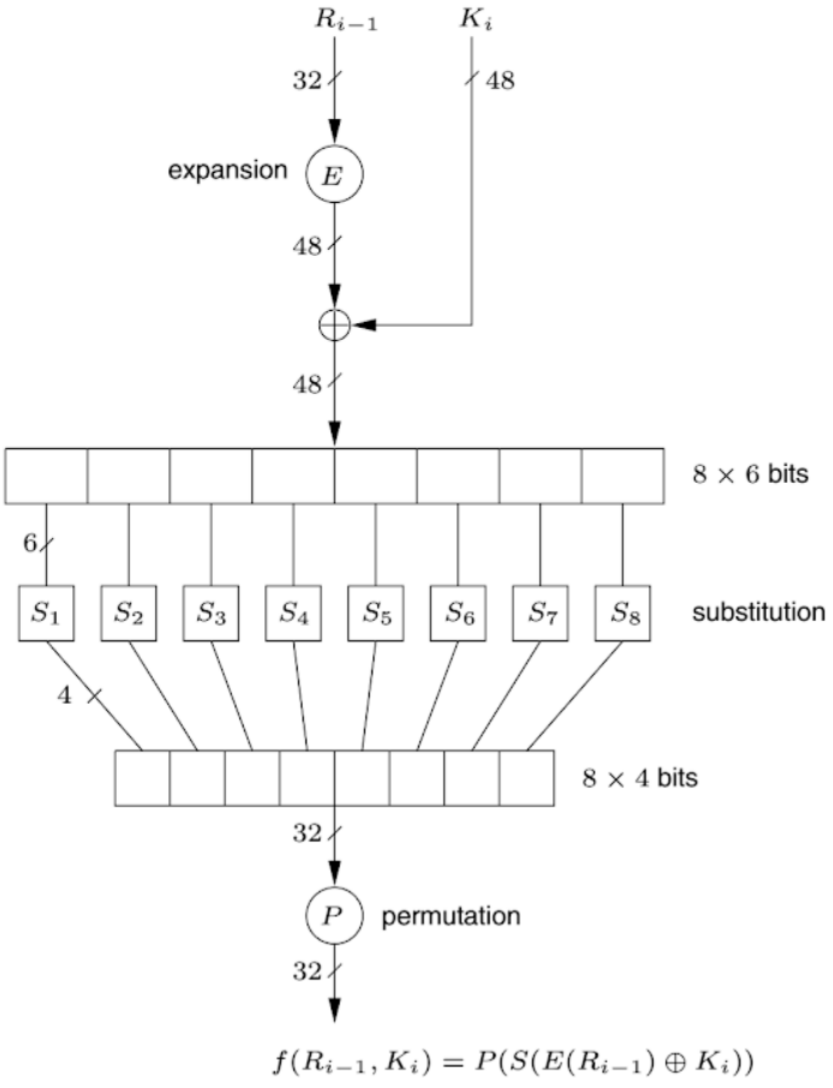


Abbildung 2.2: DES Rundenfunktion [MOV96]

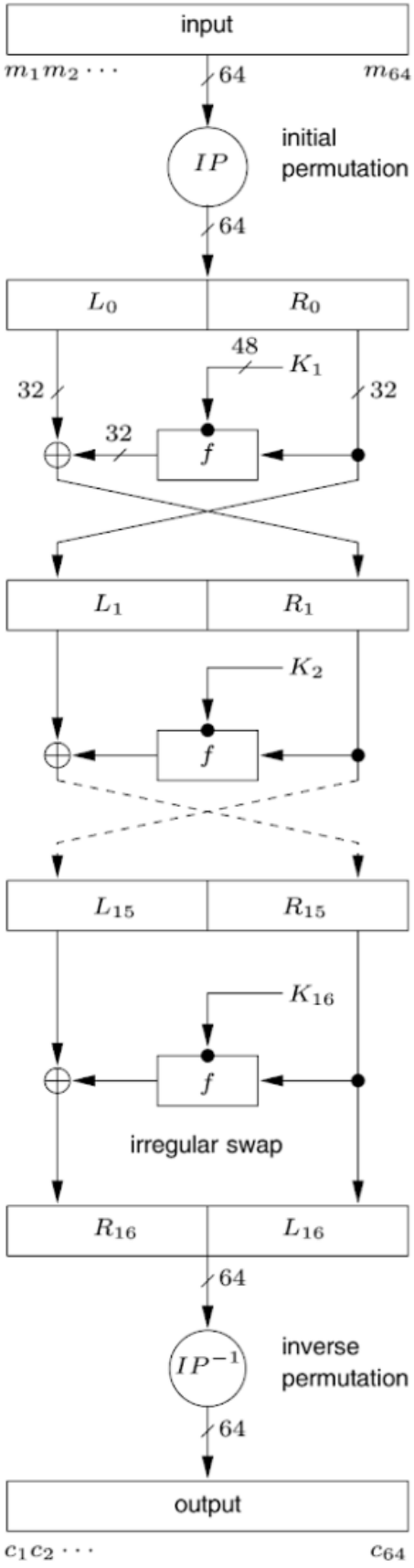


Abbildung 2.3: DES Algorithmus [MOV96]

Im Unterschied zu DES benutzt **Triple DES (3DES)** einen Schlüssel der Länge 168 Bit (3-mal 56 Bit). Grund dafür ist, dass der komplette DES-Algorithmus 3-mal durchlaufen wird, jedes Mal mit einem anderen Hauptschlüssel. [AZZ+10]

Eine weitere wichtige Eigenschaft von Blockchiffren im Allgemeinen sind die verschiedenen Betriebsmodi. Dazu zählen unter anderem Electronic Codebook (ECB) oder Chipherblock Chaining (CBC) [Wob01].
Bei ECB führen gleiche Klartextblöcke zu identischen Chiffreblöcken, da die Verschlüsselung der Blöcke völlig unabhängig voneinander durchgeführt wird [Wob01].
Ein besseres Vorgehen bietet der CBC Modus. Dabei erfolgt für jeden Klartextblock vor der Verschlüsselung eine xor-Vernüpfung mit dem zuvor erzeugten Chiffreblock. Für den ersten Block wird ein Initialisierungsvektor (IV) für die xor Verknüpfung verwendet [Wob01].
Beide Modi sind in den Abbildungen 2.4 und 2.5 vergleichend dargestellt.
Neben diesen beiden Blockchiffrierungen gibt es außerdem den CFB- und den OFB-Modus, die „den Blockalgorithmus nur [nutzen], um eine Stromchiffrierung zu definieren“ [Wob01].

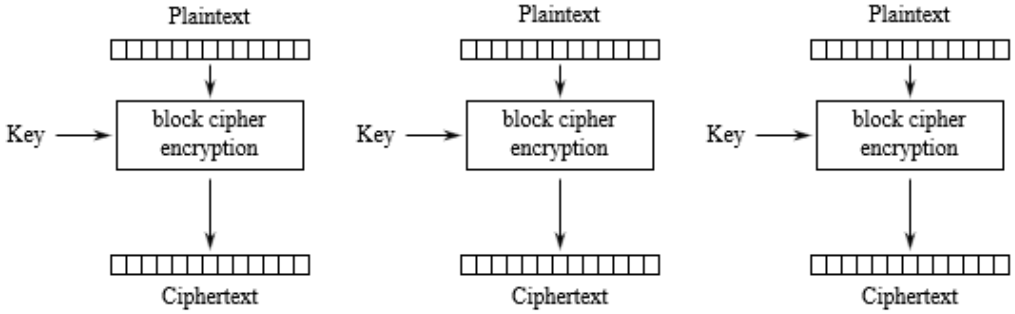


Abbildung 2.4: Blockchiffre im ECB Modus [Tim13]

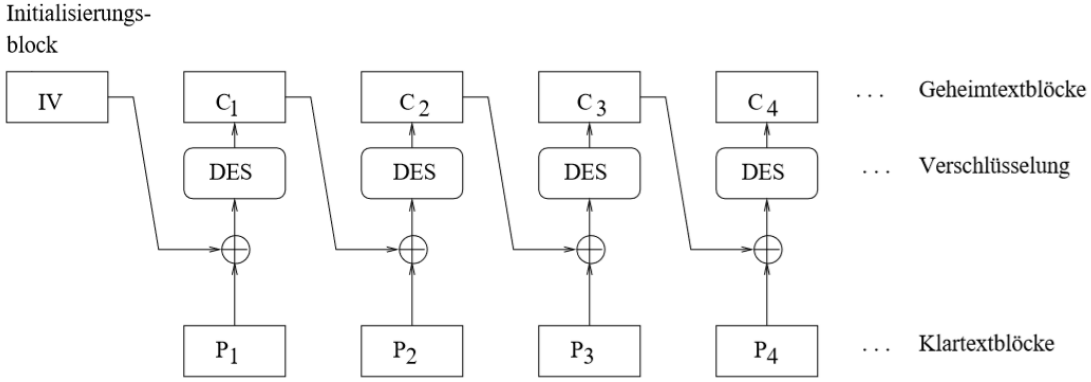


Abbildung 2.5: Blockchiffre im CBC Modus [Wob01]

2.1.4 Advanced Encryption Standard (AES)

Der Advanced Encryption Standard (AES) ist eine Spezifikation des Rijndael Algorithmus‘ und zählt somit zu den symmetrischen Blockchiffren. AES verwendet eine Blockgröße von 128 Bit und kann mit drei verschiedenen Schlüssellängen genutzt werden: 128, 192 oder 256 Bit. Zu Beginn des Algorithmus werden die Bytes eines Blockes in ein sogenanntes State-Array (s) der Größe 4 x 4 Byte übernommen. Für die Verschlüsselung benutzt AES eine Rundenfunktion, die aus 4 Transformationsschritten besteht. Wie oft die Rundenfunktion auf einen Block angewendet wird, entscheidet die Rundenzahl, die sich aus der Schlüssellänge ableitet [ST01]:

	Schlüssellänge	Rundenzahl
AES-128	128 Bit (4 words)	10
AES-192	192 Bit (6 words)	12
AES-256	256 Bit (8 words)	14

Tabelle 2.2: Beziehung zwischen Schlüssellänge und Rundenzahl bei AES [ST01]

Die erste Transformation innerhalb der Rundenfunktion heißt SubBytes() und wird durch S-Boxen realisiert, ähnlich wie sie bei DES zum Einsatz kommen. Dabei erhält jedes Byte aus dem State-Array nach einem bestimmten Schema einen anderen Wert zugewiesen. Sei der Wert eines Bytes im State-Array beispielsweise 0x3A, so wird innerhalb der S-Box Zeile 3, Spalte A gesucht und der dort gefundene Wert an die aktuelle Stelle im State-Array geschrieben [ST01]. Eine solche S-Box könnte wie folgt aussehen:

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Abbildung 2.6: Substitutions-Box (S-Box) für AES [ST01]

Im zweiten Schritt der Rundenfunktion, genannt ShiftRows(), werden die Bytes innerhalb einer Zeile nach links verschoben. Die erste Zeile bleibt unverändert, die zweite Zeile wird um ein Byte nach links verschoben, die dritte um 2 Byte und die letzte Zeile schließlich um 3, wie in Abbildung 2.8 dargestellt [ST01].
Die dritte Transformation nennt sich MixColums(). Dabei wird jede Spalte als Vektor betrachtet, mit einer fest definierten Matrix multipliziert und das Ergebnis dann als neue Spalte in das State-Array übernommen. Die Matrix beinhaltet folgende Werte [ST01]:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Als letzter Schritt der Rundenfunktion folgt nun AddRoundKey(). Um diese Transformation zu erklären, sei zuvor noch erwähnt, dass der Schlüssel zu Beginn expandiert wird, um eine Key Schedule aus 4-Byte-words zu erzeugen, also eine Art Rundenschlüssel. AddRoundKey() verküpft nun jede Spalte aus dem State-Array mittels xor mit einem solchen Rundenschlüssel aus der Key Schedule und ersetzt die alten Spaltenwerte mit dem Ergebnis dieser Operation [ST01].
Zusammenfassend sind die Transformationen in folgenden Abbildungen dargestellt.

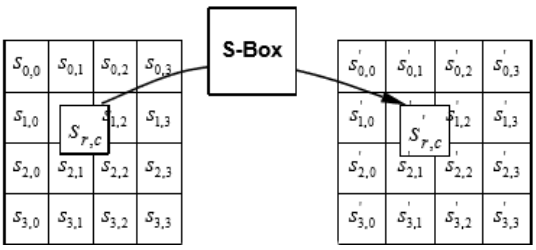


Abbildung 2.7: SubBytes() [ST01]

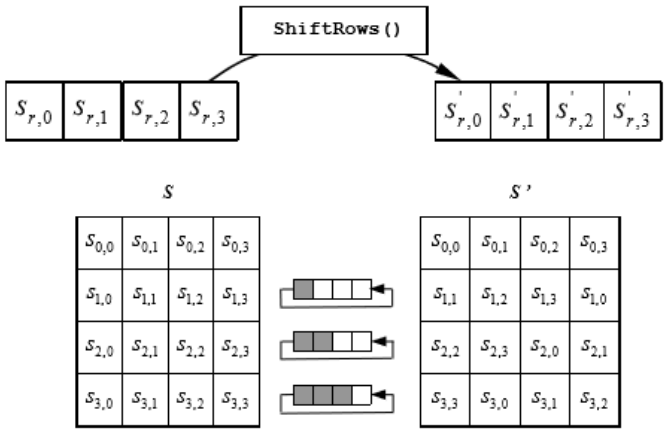


Abbildung 2.8: ShiftRows() [ST01]

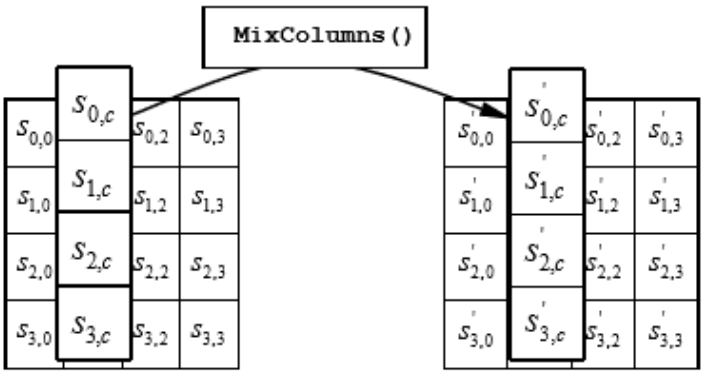


Abbildung 2.9: MixColumns() [ST01]

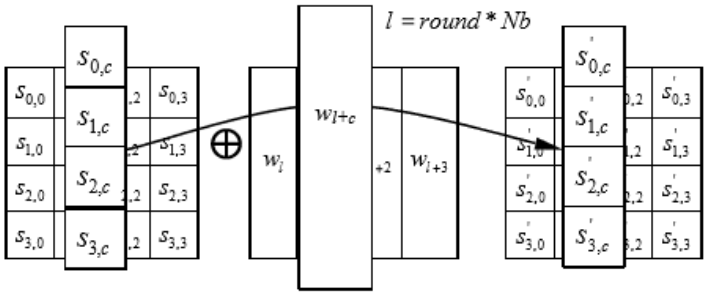


Abbildung 2.10: AddRoundKey() [ST01]

Zur Entschlüsselung werden die einzelnen Transformationen einfach invers durchgeführt. Das einfachste Beispiel hierfür ist die ShiftRows()-Funktion, bei der die Bytes während der Entschlüsselung nach rechts verschoben werden statt nach links [ST01].

2.1.5 SHA

Der Secure Hash Algorithm (SHA) ist ein iterativer Einweg Hash-Algorithmus, der aus einer Nachricht variabler Länge eine Nachricht fester Länge berechnet, welche message digest genannt wird. Die Bestimmung des Hashwertes geschieht in mehreren Schritten. Zunächst wird die Nachricht gepadded, so dass die Länge der Nachricht ein Vielfaches der Blockgröße ist. Danach erfolgt eine Unterteilung in Blöcke fester Größe (Blockgröße) und anschließend werden diese Blöcke in weitere Subblöcke bestimmter Länge gegliedert, sogenannte Words. Die Block- und Word-Länge unterscheidet sich zwischen den verschiedenen SHA-Algorithmen, wie Abbildung 2.11 verdeutlicht [Bry15].

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Abbildung 2.11: Eigenschaften der verschiedenen SHA-Algorithmen [Bry15]

Der letzte Schritt der Vorverarbeitung ist es, Initialisierungs-Werte zu setzen. Bei der eigentlichen Berechnung wird aus der gepaddeten Ausgangsnachricht eine sogenannte message schedule erzeugt. Diese message schedule wird zusammen mit diversen Funktionen, Konstanten und Word-Operationen genutzt, um iterativ mehrere Hashwerte zu berechnen. Der Wert aus der letzten Berechnung dient schließlich dazu, die message digest zu bestimmen [Bry15].

2.1.6 HMAC

Ein Message Authentication Code (MAC) ist nötig, um die Integrität einer übertragenen oder gespeicherten Information zu überprüfen. Ein MAC, der auf einer kryptografischen Hashfunktion wie SHA-1 oder MD5 basiert, wird als HMAC, bzw. HMAC-SHA1 oder HMAC-MD5 bezeichnet. Neben der Hashfunktion erfordert HMAC noch einen geheimen Schlüssel. Dieser kann eine maximale Länge von 64 Byte haben, was der Blockgröße B von SHA-1 bzw. MD5 entspricht. Sollte der Schlüssel doch länger sein, wird dieser zunächst der zugrundeliegenden Hashfunktion übergeben und der resultierende Hash als Schlüssel verwendet. Es empfiehlt sich, den Schlüssel nicht kürzer zu wählen als die Ausgabe der Hashfunktion, das heißt, für HMAC-SHA1 nicht kürzer als 20 Byte und für HMAC-MD5 nicht kürzer als 16 Byte.
Für die Bestimmung der HMAC werden zunächst die zwei Strings ipad und opad definiert. Der ipad-String besteht aus B 0x36-Bytes, während der opad-String als eine Folge von B 0x5C-Bytes festgelegt ist. Danach wird der Key K bis auf eine Länge von 64 Byte mit 0x00 Bytes aufgefüllt und mit ipad xor verknüpft. An das Ergebnis werden dann die eigentlichen Daten angehängt und über alles der Hash bestimmt. Dieser Hashwert wird dann an den mit opad xor verknüpften Key K angehängt und wieder über alles der Hash berechnet. Die entsprechende Formel lautet also wie folgt [KBC97]:

$$H(K \; xor \; opad, \; H(K \; xor \; ipad, text))$$

2.1.7 PBKDF2

Allgemein erzeugt eine Key Derivation Function aus einem Basis-Schlüssel einen daraus abgeleiteten (derived) Schlüssel. Bei einer Password Based Key Derivation Function ist dieser Basis-Schlüssel ein Passwort. Zusätzlich werden für den Derivation-Prozess weitere Parameter benötigt. Dazu gehört ein Salt, ein Iteration Count c und die gewünschte Länge l des resultierenden Schlüssels. [Kal00]
PBKDF2 liegt eine Pseudozufallsfunktion zugrunde. Diese gibt eine bestimmte Anzahl an Bytes aus, was gleichzeitig die Blockgröße vorgibt. Das zu verarbeitende Passwort muss in Blöcke dieser Größe unterteilt werden. Jeder Block durchläuft eine Routine. Dabei wird die Pseudozufallsfunktion auf den Block mit angehängtem Salt c-mal angewandt und die Ergebnisse jeder der c Iterationen xor verknüpft. Die Blöcke, die die Routine durchlaufen haben, werden schließlich aneinander gehängt. Je nachdem was als gewünschte Länge l des derived keys angegeben wurde, werden die ersten l Bytes als Resultat zurückgegeben [Kal00].

2.2 Data Protection API (DPAPI)

Seit Windows 2000 stellt Microsoft ein sogenanntes Data Protection Application Programming Interface, kurz DPAPI, zur Verfügung. Das Ziel dieses Services besteht darin, durch Verschlüsselung die Vertraulichkeit sensibler Nutzer- und Systemdaten zu gewährleisten. DPAPI bildet einen Teil der Crypt32.dll, ist also standardmäßig in jedem Windows-Betriebssystem enthalten. [GHBC01]
DPAPI stellt im Wesentlichen zwei Funktionen zur Verfügung: CryptProtectData() und CryptUnprotectData().

CryptProtectData. Vereinfacht ausgedrückt verlangt CryptProtectData Klartext-Daten und liefert einen sogenannten „opaque Data BLOB“[1]. Beim Funktionsaufruf können weitere Parameter definiert werden, um die Verschlüsselung durch DPAPI zu konfigurieren. Die entsprechende Syntax stellt sich wie folgt dar (C++):

[1] „opaque“, zu Deutsch „undurchsichtig“, bedeutet, dass die Datenstruktur bewusst nicht von Microsoft dokumentiert ist, da darin auch Informationen zur Entschlüsselung für die DPAPI enthalten sind. [GHBC01] Ein BLOB oder auch Binary Large Object ist eine Binärdatei mit variabler Länge von bis zu 2 GB [Ora]

```
BOOL WINAPI CryptProtectData(
  _In_       DATA_BLOB                 *pDataIn,
  _In_opt_   LPCWSTR                   szDataDescr,
  _In_opt_   DATA_BLOB                 *pOptionalEntropy,
  _Reserved_ PVOID                     pvReserved,
  _In_opt_   CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
  _In_       DWORD                     dwFlags,
  _Out_      DATA_BLOB                 *pDataOut
);
```

Abbildung 2.12: CryptProtectData() mit Parametern [Micb]

`pDataIn`
: verweist auf einen BLOB mit zu verschlüsselnden Klartext-Daten.

`szDataDescr`
: ist eine Beschreibung der verschlüsselten Daten und eine optionale Angabe. Ist keine Beschreibung erwünscht, wird der Parameter auf NULL gesetzt.

`pOptionalEntropy`
: beschreibt einen BLOB, in dem ein Passwort bzw. eine zusätzliche Entropie gespeichert ist. Auch dieser Parameter ist optional und darf NULL sein.

`pvReserved`
: muss immer NULL sein, da diese Option für spätere Nutzung reserviert ist.

`pPromtStruct`
: zeigt auf eine CRYPTOPROTECT_PROMTSTRUCT Struktur, die definiert, wann und wo der Benutzer zu Eingaben aufgefordert wird. Auch dieser Parameter kann auf NULL gesetzt werden, wenn diese Option nicht benötigt wird.

`dwFlags` enthält eine der folgenden Flags:

CRYPTPROTECT_LOCAL_MACHINE
: Die Daten werden nicht mit einem einzelnen Benutzer verknüpft, sondern mit dem kompletten System, so dass jeder Benutzer die Daten entschlüsseln kann.

CRYPTPROTECT_UI_FORBIDDEN
: Die Funktion darf nicht über eine Benutzeroberfläche aufgerufen werden, ansonsten wird ein Fehler zurückgegeben.

CRYPTPROTECT_AUDIT
: Es wird ein Audit zu der Funktion erzeugt.

CRYPTPROTECT_CRED_SYNC
: Hier werden keine Daten verschlüsselt, sondern nur die Masterkeys von der Festplatte abgerufen, wodurch eine Neuverschlüsselung durch ein anderes Passwort erfolgt.

CRYPTPROTECT_SYSTEM
Daten, die unter Benutzung dieser Flag verschlüsselt wurden, können auch nur unter Verwendung dieser Flag wieder entschlüsselt werden.

`pDataOut`
definiert schließlich den Data BLOB, in dem die verschlüsselten Daten abgelegt werden. [Micb]

CryptUnprotectData stellt die Umkehrfunktion zu CryptProtectData() dar und verlangt folglich einen BLOB mit verschlüsselten Daten und gibt den Klartext an die aufrufende Applikation zurück. [GHBC01] Der syntaktische Aufbau des Funktionsaufrufes ist grundsätzlich der gleiche wie bei CryptProtectData:

```
BOOL WINAPI CryptUnprotectData(
  _In_       DATA_BLOB                 *pDataIn,
  _Out_opt_  LPWSTR                    *ppszDataDescr,
  _In_opt_   DATA_BLOB                 *pOptionalEntropy,
  _Reserved_ PVOID                     pvReserved,
  _In_opt_   CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
  _In_       DWORD                     dwFlags,
  _Out_      DATA_BLOB                 *pDataOut
);
```

Abbildung 2.13: CryptUnprotectData() mit Parametern [Micc]

Beide Funktionen besitzen einen boolschen Rückgabewert. Bei erfolgreicher Ausführung wird TRUE zurückgegeben und falls die Ausführung scheitert FALSE. [GHBC01]
DPAPI bietet darüber hinaus keine Möglichkeit der Speicherung, sondern vollzieht lediglich die Verarbeitung der Daten. Wo und wie die BLOBs letztlich gespeichert werden, obliegt der Applikation, die die Dienste von DPAPI nutzt. [GHBC01]

Um die sensiblen Daten zu schützen, benötigt DPAPI ein an den Nutzer gekoppeltes Passwort. Es ist daher naheliegend, dass das Windows-Anmeldepasswort verwendet wird. Genauer gesagt, nutzt DPAPI das Logon Credential des Benutzers. Dieses unterscheidet sich jedoch entsprechend der Form der Anmeldung, beispielsweise wäre auch ein Login über einen Bildcode oder eine Smartcard möglich. Im einfachsten Fall, bei dem sich der Nutzer über ein Passwort authentifiziert, ist das Logon Credential einfach der Hash des Benutzerpasswortes. [GHBC01]
Die beiden zuvor genannten Funktionen greifen automatisch auf das entsprechende Credential zurück. Daten die mit CryptProtectData() unter einem bestimmten Nutzerkonto verschlüsselt wurden, können folglich auch nur unter Verwendung von CryptUnprotectData() unter diesem einen Konto wieder entschlüsselt werden. [GHBC01]

Das Logon Credential, in den folgenden Ausführungen als gehashtes Benutzerpasswort bezeichnet, verwendet die DPAPI nicht direkt zur Verschlüsselung geheimer Daten. Zunächst wird aus diesem Passwort gemeinsam mit einem Iteration Count und einem zufälligen Salt ein Key mittels PBKDF2 erzeugt. Mit diesem Key erfolgt die Verschlüsselung des sogenannten Masterkeys per 3DES und dessen Ablage im Nutzerverzeichnis unter Profile und der userspezifischen SID. Dieser Masterkey wird wiederum genutzt, um zusammen mit einem zufälligen Salt und einer optionalen Entropie einen symmetrischen Session Key zu generieren, mit dem die Daten schließlich verschlüsselt werden. Diese Zusammenhänge sind in Abbildung 2.14 vereinfacht dargestellt. Die einzelnen Komponenten werden im Anschluss genauer beschrieben. [GHBC01]

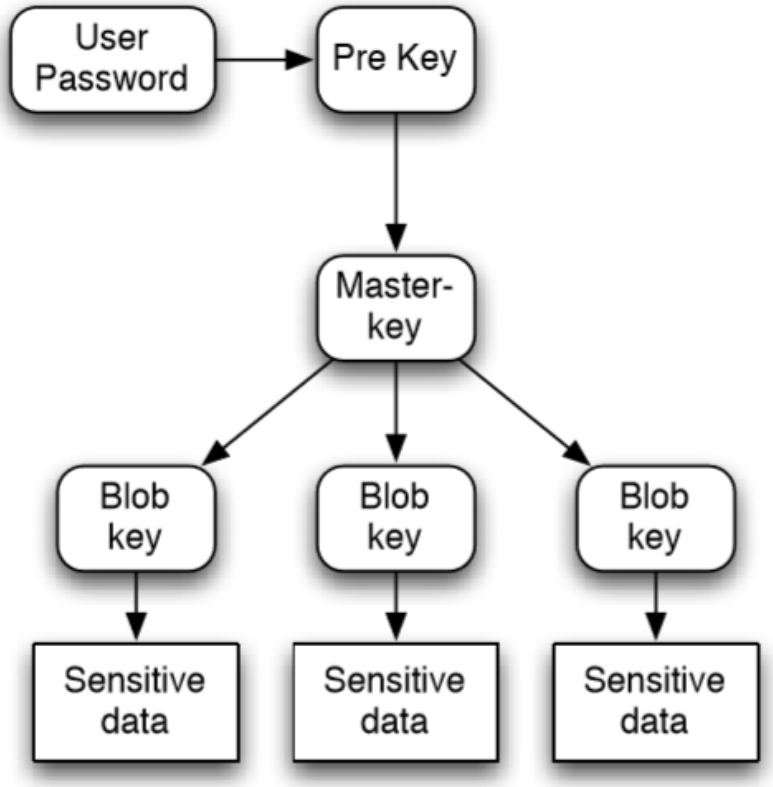


Abbildung 2.14: Beziehung der Schlüssel bei DPAPI [BP10]

Opaque Data BLOB. Der opaque Data BLOB enthält, wie bereits erwähnt, die verschlüsselten Daten. Weiterhin beinhaltet er aber auch Informationen, die der Entschlüsselung dienen. Zum einen sei hier natürlich die GUID des Masterkeys genannt, mit dem die Daten verschlüsselt wurden. Außerdem sind Angaben zum Verschlüsselungsalgorithmus, zur zugehörigen Schlüssellänge und zum verwendeten Hashalgorithmus enthalten. Um die Unverändertheit der Daten sicherzustellen, wird der gesamte BLOB mit HMAC gehasht. [GHBC01] Die gesamte Struktur kann wie folgt interpretiert werden:

0x00	4 Byte	Version der Datenstruktur/ Anz. Providers
0x04	16 Byte	GUID des Data Encryption Provieders (stellt Kompatibilität sicher)
0x14	4 Byte	Version/Anzahl Masterkeys
0x18	16 Byte	GUID des Masterkeys mit dem der BLOB verschlüsselt wurde
0x28	4 Byte	Flags (z.B. Bit 3: Entschlüsselung muss unter dem SYSTEM Account stattfinden)
0x2C	4 Byte	Länge der optionalen Beschreibung (len)
0x30	len	optionale Beschreibung (durch szDataDescr-Parameter in der CryptProtectData Funktion definiert; für NULL speichert DPAPI einen leeren UTF-16LE kodierten String mit len = 2)
0x30 + len	4 Byte	ID des Verschlüsselungsalgorithmus[2]
0x34 + len	4 Byte	Schlüssellänge des Verschlüsselungsalgorithmus
0x38 + len	4 Byte	Länge Salt (lenS)
0x3C + len	lenS	Salt
0x3C + len + lenS	4 Byte	Länge Strong Password (lenSP)
0x40 + len + lenS	4 Byte	Hashalgorithmus (SHA-1 = 0x800E)
0x44 + len + lenS	4 Byte	Länge des Hashs
0x48 + len + lenS	4 Byte	Länge HMAC (lenH)
0x4C + len + lenS	lenH	HMAC
0x4C + len + lenS + lenH	4 Byte	Länge der verschlüsselten Daten (lenD)
0x50 + len + lenS + lenH	lenD	VERSCHLÜSSELTE DATEN
0x50 + len + lenS + lenH + lenD	4 Byte	Länge Signatur (lenSig)
0x54 + len + lenS + lenH + lenD	lenSig	digitale Signatur (Integritätssicherung, HMAC)

Tabelle 2.3: Struktur des Opaque Data Blob [Pas12] [Jan17] [PB10] [Del17]

Masterkey. Ein Masterkey besteht aus 512 zufälligen Bits. Er ist in verschlüsselter Form in einer Masterkey-Datei, zusammen mit anderen Angaben, die der Entschlüsselung und Integritätssicherung dienen, abgelegt [GHBC01]. Eine Masterkey-Datei befindet sich für Windows 7 und höher standardmäßig unter C:\Benutzer\ <Username>\AppData\Roaming\Microsoft\Protect\<SID> [Pic14].

Oft sind dort mehrere Dateien mit eindeutigen GUIDs als Namen hinterlegt. Grund dafür ist, dass der Masterkey aller drei Monate durch einen anderen zufälligen Key ersetzt wird. Um aber dennoch auf Daten zugreifen zu können, die mit alten Keys verschlüsselt wurden, werden alle jemals verwendeten Schlüssel im genannten Verzeichnis gespeichert.
Zusätzlich beinhaltet der Data BLOB mit den chiffrierten Daten die GUID des entsprechenden Masterkeys, mit dem die Daten verschlüsselt wurden [GHBC01].
Eine Masterkey Datei besitzt zusammenfassend folgenden Aufbau:

Header

0x00	4 Byte	Datei-Version des Masterkeys
0x04	8 Byte	0
0x0C	72 Byte	textuelle GUID
0x54	8 Byte	0
0x5C	4 Byte	Flags (Bit 3 gesetzt: Hashing beim Entschlüsseln des User Passwortes mit SHA-1, sonst MD4; Bit 2 gesetzt: Backup für Masterkey wird benötigt)

Masterkey

0x60	8 Byte	Master Key Size (len)
0x68	8 Byte	Local Encryption Key Size (len2)
0x70	8 Byte	HMAC Size/CREDHIST Size/Local Backup Key Size? (len3)
0x78	8 Byte	Domain Backup Key Size (len4)
0x80	4 Byte	Version der Datenstruktur (Ver 1 immer mit Salt)
0x84	16 Byte	Salt
0x94	4 Byte	Iteration Count für PBKDF2
0x98	4 Byte	HMAC Hash Algorithmus ID
0x9C	4 Byte	ID des Verschlüsselungsalgorithmus
0xA0	16 Byte	Initialisierungsvektor (IV) evtl. auch berechnet
0xB0	64 Byte	verschlüsselter Masterkey
0xF0	24 Byte	verschlüsselter HMAC

Local Encryption Key (um local Backup Key zu entschlüsseln (Win 2000))

0xA0 + len – 0x20	4 Byte	Version der Datenstruktur
0xA4 + len – 0x20	16 Byte	Salt
0xB4 + len – 0x20	4 Byte	Iteration Count für PBKDF2
0xB8 + len – 0x20	4 Byte	HMAC Hash Algorithmus ID

0xBC + len – 0x20	4 Byte	ID des Verschlüsselungsalgorithmus
0xC0 + len – 0x20	len2 – 32 Byte	verschlüsselter Local Encryption Key
CREDHIST (Win XP und höher)		
0xC0 + len + len2 – 0x40	4 Byte	Version der Datenstruktur
0xC4 + len + len2 – 0x40	16 Byte	GUID
Local Backup Key (Win 2000)		
0xC0 + len + len2 – 0x40	4 Byte	Version der Datenstruktur
0xC4 + len + len2 – 0x40	16 Byte	Salt
0xD4 + len + len2 – 0x40	len3 – 20 Byte	verschlüsselter local Backup Key
Domain Backup Key		
0xC4 + len + len2 + len3 – 0x60	4 Byte	Version
0xC8 + len + len2 + len3 – 0x60	16 Byte	Salt
0xD8 + len + len2 + len3 – 0x60	4 Byte	IterationCount
0xDC + len + len2 + len3 – 0x60	4 Byte	HMAC Hash Algorithmus ID
0xE0 + len + len2 + len3 – 0x60	4 Byte	Verschlüsselungsalg ID
0xE4 + len + len2 + len3 – 0x60	len4 – 32 Byte	verschlüsselter Domain Backup Key[3]

Tabelle 2.4: Struktur des DPAPI Masterkeys [Pasa] [PB10] [Pic14]

CREDHIST. Wie bereits erwähnt, wird der Masterkey mithilfe des Benutzerpasswortes verschlüsselt. Ändert der Benutzer sein Kennwort, werden alle Masterkeys neu verschlüsselt. Zusätzlich muss das alte Passwort in einer Datei namens CREDHIST gespeichert werden, die ebenfalls im Protect-Verzeichnis liegt. Diese sogenannte „Credential History“ enthält alle bisherigen Benutzerpasswörter. Eine Passwortänderung bewirkt, dass das alte Kennwort an den Beginn der Liste gesetzt und die gesamte Datei wird mit dem neuen Passwort verschlüsselt wird [BP10]. Es entsteht also eine Art Schachtelung:

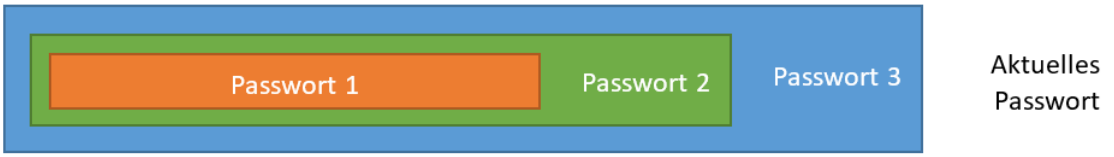


Abbildung 2.15: Schachtelung innerhalb der CREDHIST-Datei (eigene Quelle)

[3] die Entschlüsselung erfordert den Domain Controller RSA private key, der in der Active Directory Database gespeichert ist

Um nun einen bestimmten BLOB zu entschlüsseln, wird zunächst die GUID des zugehörigen Masterkeys aus dem BLOB extrahiert und geprüft, ob dieser Key mit dem aktuellen Passwort entschlüsselt werden kann. Ist dies nicht der Fall, erfolgt die Entschlüsselung der CREDHIST mit Hilfe des aktuellen Passworts, um so das vorherige Anmelde-Passwort zu erhalten. Mit diesem älteren Kennwort startet ein neuer Versuch, den Masterkey zu entschlüsseln. Dieser Vorgang muss nun solange wiederholt werden, bis das richtige Passwort ermittelt wurde [Pas12].
Natürlich ist die CREDHIST Datei keine einfache Liste aus verschlüsselten Passwörtern, sondern eine Binärdatei mit einem speziellen Aufbau. Zu jedem Passwort werden weitere Angaben gespeichert, wie die ID des Hash Algorithmus, ein Iteration Count und die ID des Verschlüsselungsalgorithmus [Pas12].
Eine Ausnahme bildet der zeitlich erste Eintrag, also der letzte innerhalb der CREDHIST Datei. Dieser sogenannte Footer besteht lediglich aus der Version, einem current link unique identifier und der Größe des folgenden Eintrags. Eine Größe von 0 bedeutet, dass kein weiterer Eintrag folgt. Ein Block innerhalb der CREDHIST gestaltet sich nach folgendem Schema:

0x00	4 Byte	Version
0x04	16 Byte	current link unique identifier (GUID Cred Hist)
0x14	4 Byte	next Cred Size
0x18	4 Byte	Cred Link Type
0x1C	4 Byte	Hash Algorithmus
0x20	4 Byte	IterationCount
0x24	4 Byte	SID Size (size)
0x28	4 Byte	Verschlüsselungsalg ID
0x2C	4 Byte	ShaHashSize (size1)
0x30	4 Byte	NT Hash Size (size2)
0x34	16 Byte	Salt
0x44	size	SID
0x44 + size	size1	verschlüsselter SHA-1 Hash
0x44 + size + size1	size2	verschlüsselter NTLM Hash

Tabelle 2.5: Struktur der CREDHIST [Pas12]

Local Backup Key und Domain Backup Key. Die Masterkey Datei kann neben dem Masterkey auch noch weitere Schlüssel enthalten, sogenannte Backup Keys. Der Local Backup Key wurde nur für Windows 2000 verwendet. Bei aktuellen Windows Versionen nutzt man den Domain Backup Key, vorausgesetzt der Computer

ist einer Domain zugehörig. In diesem Fall, fordert der lokale DPAPI Client immer dann, wenn ein Masterkey generiert wird, einen public key vom Domain Controller an. Der Domain Controller hat extra für DPAPI ein domainweit einheitliches public/private-Key-Paar. Hat der Client Computer den public key über einen authentifizierten und gesicherten RPC Call erhalten, wird der Masterkey damit verschlüsselt und der resultierende Backup Masterkey als Domain Backup Key mit in der Masterkey Datei abgelegt. Falls der originale Masterkey nicht verwendet werden kann, besteht die Möglichkeit, den Backup Key über RPC an den Domain Controller zu senden. Dort wird der Key mit dem privaten Schlüssel dechiffriert und wieder gesichert über RPC an den Client zurückgeschickt [GHBC01].

Password Reset Disk und Recovery Key. Computer die nicht zu einer Domain gehören, haben die Möglichkeit, eine Passwort Reset Disk (PRD) zu erstellen. Diese dient dann als Backup, falls das Passwort in Vergessenheit gerät. Bei der Erstellung wird ein 2048-bit RSA public/private Key Paar erzeugt. Im Anschluss erfolgt die Verschlüsselung des aktuellen Passwortes mit dem public key und die Ablage im Nuzterverzeichnis. Der private Key wird auf der PRD gespeichert. Um das Passwort zurückzusetzen, muss zunächst das alte Kennwort mit dem private key von der Disk entschlüsselt werden, um es schließlich durch ein neues zu ersetzen. Erfolgt die Rücksetzung des Passwortes über diese Disk, werden alle Masterkeys durch das neue Passwort verschlüsselt [GHBC01].

2.3 Webbrowser

Der Begriff Browser lässt sich auf das englische Verb „browse“ zurückführen, was so viel bedeutet wie blättern, stöbern, grasen oder umsehen. [Pon] Webbrowser sind Anwendungsprogramme, die sowohl Webseiten als auch andere Dokumente darstellen können.
Browser speichern eine große Menge an Daten, aus denen sich die Aktivitäten des Nutzers oft leicht rekonstruieren lassen. Dazu gehören unter anderem der Browserverlauf, die Download History, Einträge aus Formularen, Suchanfragen, temporäre Daten (Cache), Seiteneinstellungen, Cookies und eben auch Usernamen und Passwörter aus Login-Vorgängen. [AjW$^+$] Zu finden sind diese Daten meist im Profilordner des Nutzers in Form von Text-Dateien oder Datenbanken. Einige Browser legen die Informationen aber auch in der Registry ab. Wenn von gespeicherten Passwörtern bzw. Usernamen die Rede ist, sind im engeren Sinne die Daten gemeint, für die der „Passwort speichern“-Dialog mit „Ja“ beantwortet wurde.

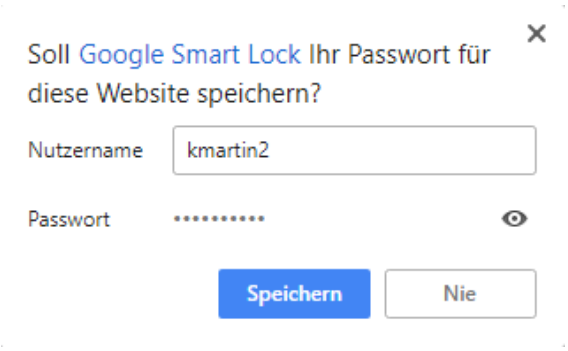


Abbildung 2.16: 'Passwort speichern'-Dialog in Google Chrome (eigene Quelle)

Laut aktueller Statistik ist Google Chrome mit fast 40 Prozent Marktanteil der am häufigsten verwendete Webbrowser in Deutschland, gefolgt von Mozilla Firefox mit 26,5 Prozent. Safari wird häufig auf Apple-Geräten eingesetzt, befindet sich mit 13 Prozent aber weit hinter Chrome und Firefox. Im Abstand von je 3 Prozent zueinander folgen schließlich noch Internet Explorer, Microsoft Edge und Opera. Letzterer Browser hielt in den letzten Jahren seinen Marktanteil sehr konstant zwischen 2 und 4 Prozent. [Sta18b]

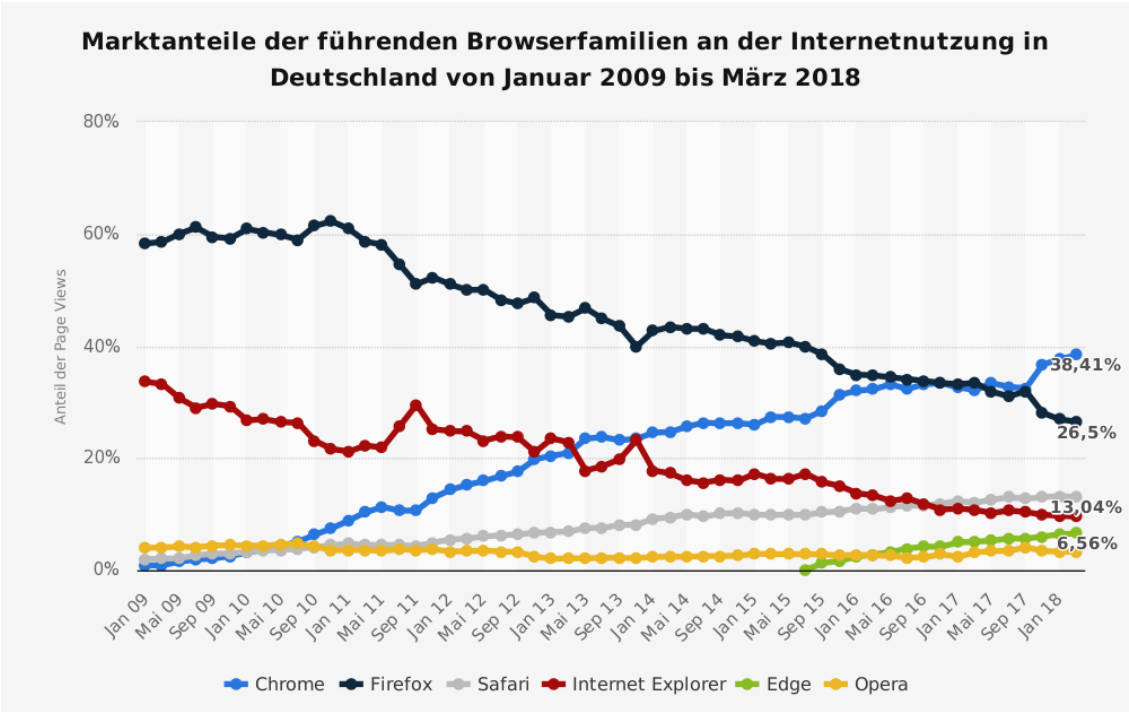


Abbildung 2.17: Browsernutzung in Deutschland [Sta18b]

Auf der Grundlage dieser Statistik sollen in dieser Arbeit insbesondere die unter Windows am häufigsten verwendeten Browser untersucht werden. Dazu zählen Chrome, Firefox, Internet Explorer und Edge.

2.3.1 Mozilla Firefox

Mozilla Firefox oder kurz Firefox ist ein kostenloser Webbrowser, der aus dem 1998 ins Leben gerufenen Mozilla-Projekt hervorging. Die erste vollständige Version „Mozilla 1.0“ wurde 2002 veröffentlicht und beinhaltete sowohl einen Browser als auch ein E-Mail Programm und andere Anwendungen. Nachdem Mozilla 2003 die Mozilla Foundation gründete, entwickelte man die einzelnen Komponenten zunehmend unabhängig voneinander weiter. Die Veröffentlichung von Firefox 1.0 erfolgte schließlich 2004 und verzeichnete bereits im ersten Jahr über 100 Millionen Downloads. Seitdem folgen regelmäßig neue Versionen und seit 2013 gibt es Firefox OS auch für Smartphones. [Moz]

logins.json

Firefox legt seine gespeicherten Login-Daten gegenwärtig (Version 60.0.1) in einer Datei namens logins.json unter C:\Benutzer\<Username>\AppData\Roaming\Mozilla\Firefox\Profiles\<ID>.<Nutzername> ab [dru15]. Im Profiles-Verzeichnis gibt es meist nur einen Ordner der mit <ID>.default benannt ist, wobei <ID> eine 8 Stellen lange Folge aus Zahlen und Kleinbuchstaben repräsentiert. Falls mehrere Nutzerprofile in Firefox registriert sind, können neben dem default-Ordner auch weitere Verzeichnisse existieren, die ebenfalls mit einer 8-stelligen ID und dem Nutzernamen benannt sind.
In älteren Firefox-Versionen nannte sich die logins.json signons.txt, ab Firefox 1.5 wurde sie in signongs2.txt umbenannt und ab Version 3.0 fand man sie unter dem Namen signons3.txt. Bevor logins.json eingeführt wurde, gab es noch eine völlig andere Variante. Die Daten waren in einer SQLite-Datenbank unter der Bezeichnung ßignons.sqlite“ abgelegt [Gun09]. Bei der aktuellen logins.json Datei handelt es sich wieder um eine einfache Textdatei, die zu jedem gespeicherten Passwort folgende Angaben in Klartext enthält:

id	Nummer des Eintrages
hostname	Adresse der „Hauptwebseite“
httpRealm	wenn mehrere Seiten zu einem Realm gehören, sind die Anmeldedaten für alle diese Seiten gültig
formSubmitURL	Adresse der Webseite, wo die Anmeldung stattfindet
usernameField	Name des Username-Feldes im Formular
passwordField	Name des Passwort-Feldes im Formular
encType	Typ der Verschlüsselung
timeCreated	Zeitpunkt der ersten Anmeldung auf dieser Webseite
timeLastUsed	Zeitpunkt der letzten Anmeldung auf dieser Webseite
timePasswordChanged	Zeitpunkt der letzten Passwortänderung
timesUsed	Anzahl der Anmeldungen

Tabelle 2.6: Inhalt der logins.json [Cle13] [dru15]

Neben diesen Einträgen finden sich auch die beiden Felder „encryptedUsername“ und „encryptedPassword“. Diese beiden Werte sind zunächst einmal Base64 kodiert. Nach der Dekodierung erhält man eine Zeichenkette, die wiederum ASN.1-DER kodiert ist [dru15]:

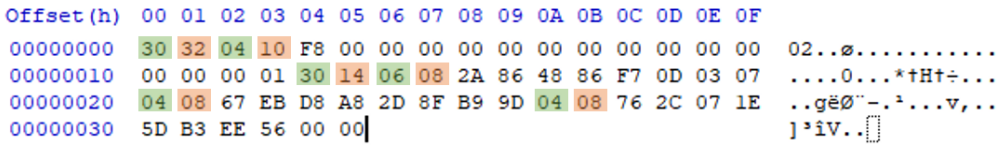


Abbildung 2.18: Beispiel für ein ASN.1-DER kodiertes Passwort aus der logins.json (eigene Quelle)

Die grün markierten Felder geben den Datentyp der nachfolgenden Daten an, die orangefarbenen Werte stehen für die Länge der Daten in Byte. Der Typ 30 an Offset 0x00 entspricht einer SEQUENZ, die weitere Typen beinhaltet. Zunächst eine Folge aus hexadezimalen Werten (OCTET STRING) der Länge 10, eingeleitet durch 0x04. Dieser enthält einen 16 Byte langen konstanten Wert. Es folgt eine weitere SEQUENZ an Offset 0x14. Sie beinhaltet einerseits einen OBJECT IDENTIFIER (Typ 0x06), der den Verschlüsselungsalgorithmus angibt, in diesem Falle des-EDE3-CBC, und zum anderen einen weiteren OCTET STRING, der den Initialisierungsvektor repräsentiert. Im Anschluss an diese SEQUENZ folgt nun das eigentliche verschlüsselte Passwort ebenfalls als OCTET STRING [dru15].

SEQUENZ
- OCTET STRING (16 Byte) Konstante 0xF800000000000000000000000000001
- SEQUENZ
  - OBJECT IDENTIFIER (8 Byte) Verschlüsselungsalgorithmus des-EDE3-CBC
  - OCTET STRING (8 Byte) Initialisierungsvektor
- OCTET STRING (x Byte) verschlüsseltes Passwort

key3.db

Um die gespeicherten Passwörter ermitteln zu können, wird nun noch der Schlüssel benötigt. Dieser befindet sich ebenfalls verschlüsselt in einer Datei namens key3.db, die im gleichen Verzeichnis liegt wie logins.json [dru15].
Der Aufbau dieser Binär-Datei ist recht komplex. Am Anfang befindet sich ein Header, der mit einem 4 Byte Magic eingeleitet wird [dru15]. So ist es möglich nach der key3.db zu suchen, falls sie aus irgendwelchen Gründen nicht im Standard Verzeichnis liegen sollte. Alle weiteren Informationen zum Aufbau des Headers sind der Übersicht 2.7 zu entnehmen:

0x00	4 Byte	Magic 0x00061561
0x04	4 Byte	Version ID
0x08	4 Byte	ByteOrder
0x0C	4 Byte	Bucket/Page Size
0x10	4 Byte	Bucket Shift
0x14	4 Byte	Directory Size
0x18	4 Byte	Segment Size
0x1C	4 Byte	Segment Shift
0x20	4 Byte	Location of overflow pages
0x24	4 Byte	Last overflow page freed
0x28	4 Byte	ID of Maximum Bucket in use
0x2C	4 Byte	Mask to modulo into entire table
0x30	4 Byte	Mask to modulo into lower half of table
0x34	4 Byte	Fill factor
0x38	4 Byte	Number of keys in Hash table
0x3C	4 Byte	size of table header
0x40	4 Byte	value of hash (CHARKEY)
0x44	4 Byte	number of bit maps and spare
0x48	4 Byte	spare pages for overflow
0x4C	4 Byte	address of overflow page bitmaps

Tabelle 2.7: Header der Binärdatei key3.db [Cle13] [dru15]

Nach dem Header folgen die sogenannten Pages, deren Größe im Header angegeben ist. Die Pages enthalten Schlüssel-Wert-Paare. Als Schlüssel dient hier ganz einfach der Name des Wertes. Die Offsets zu diesen Namen und zugehörigen Werten sind am Beginn der Page notiert [dru15]. Dabei ist zu beachten, dass die Offsets relativ zum Page-Anfang angegeben sind.
Page 1 beginnt mit 0x0600 und den Offsets zu Version, password-check und global-salt (jeweils zuerst der Offset zum Schlüssel, danach der Offset zum Wert selbst).
Page 2 wird mit 0x0200 eingeleitet, gefolgt von den Offsets zu Schlüssel und Wert von 0xF8000000000000000000000000000001.

Um das weitere Vorgehen zu verstehen, muss zunächst noch die Rolle des Masterpasswortes geklärt werden. Das Masterpasswort kann vom Benutzer als ein zusätzliches

Kennwort festgelegt werden, um die gespeicherten Login-Daten besser zu schützen. Standardmäßig ist die Masterpasswort-Option bei der Einrichtung von Firefox deaktiviert. In diesem Fall kann ein Angreifer, der im Besitz der key3.db und der logins.json ist, ganz leicht alle gespeicherten Login-Daten entschlüsseln, wie in den folgenden Ausführungen deutlich werden wird. Es wird daher empfohlen, ein Masterpasswort einzurichten, da dies im Grunde genommen den einzigen effektiven Schutz darstellt [Gun09].

Um zu überprüfen, ob das richtige Masterpasswort eingegeben wurde, nutzt Firefox den password-check von Page 1. Dieser Wert beinhaltet genau genommen noch eine eigene Struktur, die folgendermaßen interpretiert werden kann [dru15]:

0x00	1 Byte	Header
0x01	1 Byte	entry-salt size (size)
0x02	1 Byte	number of entry-salt
0x03	size	entry-salt (ES)
0x03+ size	2 Byte	seperator 0x000B
0x05+size	11 Byte	oid
0x0F+size	16 Byte	verschlüsselter password-check

Tabelle 2.8: Struktur password-check [dru15]

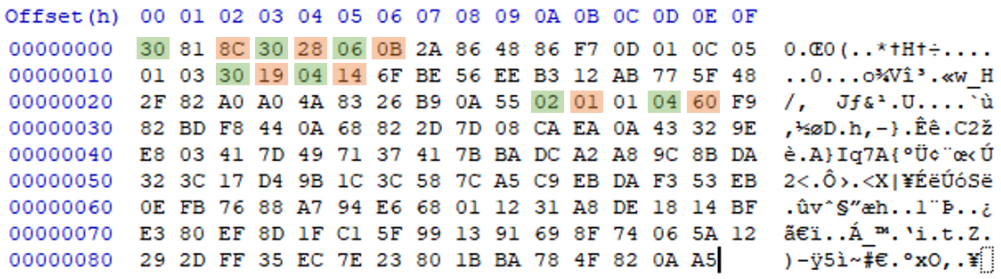
Soll nun überprüft werden, ob das Masterpasswort korrekt war, wird dieses zunächst an den global-salt von Page 1 angehängt und von beiden der SHA1-Hash bestimmt. An das Ergebnis wird dann der entry-salt aus dem password-check angehängt und beides wieder mit SHA1 gehasht. Im nächsten Schritt wird ein HMAC-SHA1 berechnet. Dabei bildet das Resultat des zweiten Hashings den Key und die Nachricht setzt sich aus dem auf 20 Byte mit 0x00 gepaddeten entry-salt und dem normalen entry-salt zusammen. Das Ergebnis dieser Berechnung ist der erste Teil des Schlüssels k. Um den zweiten Teil zu ermitteln, bedarf es der Bestimmung eines weiteren HMAC-SHA1. Der Key bleibt unverändert, die Nachricht bildet jedoch allein der gepaddete entry-salt. Es folgt eine dritte HMAC-SHA1 Berechnung, wieder mit gleichbleibendem Schlüssel. Die Nachricht setzt sich nun aus dem Ergebnis der vorherigen HMAC und dem entry-salt zusammen. Der daraus entstandene zweite Teil des Schlüssels k kann jetzt an den ersten Teil angehängt werden. k ist nun 40 Byte lang, wobei die ersten 24 Byte die drei Schlüssel für den DES-Algorithmus darstellen und die letzten 8 Byte den Initialisierungsvektor. Zum besseren Verständnis können die einzelnen Berechnungsschritte auch als Formeln dargestellt werden [dru15]:

hashed password $HP = SHA1(global\_salt|Masterpasswort)$
padded entry-salt PES = entry-salt(ES) durch anhängen von 0x00-Bytes
auf 20 Byte auffüllen

$CHP = SHA(HP|ES)$
$k1 = HMAC\_SHA1(key = CHP, msg = PES|ES)$
$tk = HMAC\_SHA1(key = CHP, msg = PES)$
$k2 = HMAC\_SHA1(key = CHP, msg = tk|ES)$
$k = k1|k2$
$keyDES = k[0..23]$
$iv = k[32..39]$

Mithilfe dieser Angaben kann nun der verschlüsselte password-check entschlüsselt werden. Lautet das Resultat der Entschlüsselung „password-check . .“, wobei die Punkte zweimal den Hexwert 0x02 repräsentieren, so ist der verwendete Masterkey korrekt [dru15].

Um nun schließlich die Login-Daten aus der logins.json Datei zu entschlüsseln, bedarf es einem sogenannten private key. Dieser befindet sich im Bereich von Page 2. Wie schon erwähnt existiert dort der Key 0xF8000000000000000000000000000001. Der zugehörige Wert ist ASN.1-DER kodiert [dru15]:



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	30	81	8C	30	28	06	0B	2A	86	48	86	F7	0D	01	0C	05	0.Œ0(..*†H†÷....
00000010	01	03	30	19	04	14	6F	BE	56	EE	B3	12	AB	77	5F	48	..0...o¾Vî³.«w_H
00000020	2F	82	A0	A0	4A	83	26	B9	0A	55	02	01	01	04	60	F9	/, Jƒ&¹.U....`ù
00000030	82	BD	F8	44	0A	68	82	2D	7D	08	CA	EA	0A	43	32	9E	,½øD.h,-}.Êê.C2ž
00000040	E8	03	41	7D	49	71	37	41	7B	BA	DC	A2	A8	9C	8B	DA	è.A}Iq7A{ºÜ¢¨œ‹Ú
00000050	32	3C	17	D4	9B	1C	3C	58	7C	A5	C9	EB	DA	F3	53	EB	2<.Ô›.<X|¥ÉëÚóSë
00000060	0E	FB	76	88	A7	94	E6	68	01	12	31	A8	DE	18	14	BF	.ûvˆ§”æh..1¨Þ..¿
00000070	E3	80	EF	8D	1F	C1	5F	99	13	91	69	8F	74	06	5A	12	ã€ï..Á_™.‘i.t.Z.
00000080	29	2D	FF	35	EC	7E	23	80	1B	BA	78	4F	82	0A	A5		)-ÿ5ì~#€.ºxO‚.¥

Abbildung 2.19: ASN.1-DER kodierter String mit enthaltenem private key (eigene Quelle)

Daraus ergibt sich folgende Struktur [dru15]:

- SEQUENZ
  - SEQUENZ
    - OBJECT IDENTIFIER (11 Byte)
    - SEQUENZ
      - OCTET STRING (20 Byte) entry-salt
      - INTEGER (1 Byte)
    - OCTET STRING (96 Byte) 3DES-verschlüsselter Wert

Mithilfe des neuen entry-salts kann der Schlüssel und der IV für 3DES auf dem gleichen Weg bestimmt und wie beim Passwort-Check der letzte OCTET STRING entschlüsselt werden [dru15]. Das Ergebnis der Entschlüsselung ist wiederum ASN.1-DER kodiert:

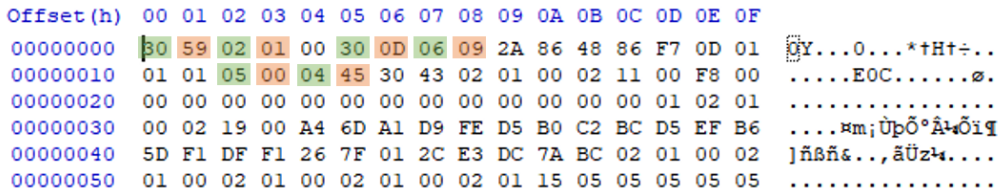


Abbildung 2.20: ASN.1-DER kodierter String als Resultat der vorangegangenen Entschlüsselung (eigene Quelle)

Man erhält also [dru15]:

SEQUENZ
- INTEGER (1 Byte)
- SEQUENZ
  - OBJECT IDENTIFIER (9 Byte)
  - NULL
- OCTET STRING (69 Byte) ASN.1-kodierter Wert

Der letzte OCTET STRING ist wiederum ASN.1-DER kodiert und wird wie folgt dekodiert [dru15]:

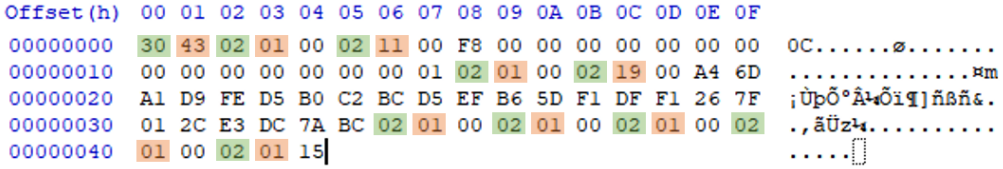


Abbildung 2.21: ASN.1-DER kodierter String aus dem OCTET STRING der vorhergehenden Dekodierung (eigene Quelle)

SEQUENZ
- INTEGER (1 Byte) 0
- INTEGER (17 Byte) 0xF80000000000000000000000000000001
- INTEGER (1 Byte) 0
- INTEGER (25 Byte) finaler Schlüssel, um die Login-Daten zu entschlüsseln[4]
- INTEGER (1 Byte) 0
- INTEGER (1 Byte) 0
- INTEGER (1 Byte) 0
- INTEGER (1 Byte) 0
- INTEGER (1 Byte) 15

[4] führendes Byte 0x00 muss in diesem Beispiel weggelassen werden um einen 24-Byte-Schlüssel zu erhalten

Damit sind alle notwendigen Informationen vorhanden, um die Login-Daten zu entschlüsseln: Aus der logins.json Datei werden die Art der Verschlüsselung, der IV und das verschlüsselte Passwort bzw. der verschlüsselte Username bezogen und aus der key3.db Datei der passende Schlüssel.

key4.db

Seit der Anfang 2018 veröffentlichten Version 58 wurden einige Dateien neu strukturiert, so auch die key3.db. Sie wird in aktuellen Versionen von einer key4.db Datei abgelöst [Moz18]. Obwohl der Name recht ähnlich klingt, handelt es sich hierbei nicht mehr um eine Binärdatei, sondern um eine SQLite Datenbank.
Dennoch finden sich alle Elemente aus der alten key3.db leicht wieder. Die Datenbank beinhaltet 2 Tabellen, ***metaData*** und ***nssPrivate***. Die erste besteht lediglich aus einer Zeile und 3 Spalten, wovon die erste mit der ID „password“ belegt ist.
Die anderen beiden Spalten sind überschrieben mit „item1“ und „item2“. Hinter diesen Bezeichnungen verbirgt sich jedoch nichts anderes als der global-salt und der password-check. Die Tabelle metaData stellt somit das Pendant zu Page 1 aus der key3.db dar.
Die Tabelle nssPrivate ist etwas umfangreicher, doch im Grunde genommen ist hier nur der Eintrag in der Spalte „a11“ für die Entschlüsselung der Login Daten relevant. Dort verbirgt sich der private-key inklusive entry-salt, dem verschlüsselten Key und dem IV, ebenso wie auf Page 2 der key3.db [LaZ].

Zusammenfassend kann gesagt werden, dass Firefox seine Passwörter nicht besonders sicher ablegt. Sobald die Abfolge der einzelnen Schritte zur Dekodierung und Entschlüsselung bekannt ist, kann jedes Login-Passwort ganz leicht entschlüsselt werden.
Eine deutlich bessere Sicherheit bietet das Masterpasswort. Ist dieses zusätzliche Passwort gesetzt, muss durch den Angreifer eine weitere Hürde überwunden werden, bevor er sein Ziel erreichen kann. Durch den Password-Check lässt sich dieses Masterpasswort aber zum Beispiel mit Brute-Force oder Wörterbuch-Angriffen herausfinden. Wie viel Schutz das Masterpasswort also letztlich bietet, hängt stark von seiner Länge und den verwendeten Zeichen ab.

2.3.2 Google Chrome

Google veröffentlichte im September 2008 seine Beta Version des Chrome-Browsers und schon im Dezember folgte die erste stabile Version 1.0 [Lar].
Chrome nutzt zur Verschlüsselung der User-Passwörter die Windows DPAPI, das heißt, die Daten werden durch die Funktion CryptProtectData() geschützt. Der daraus resultierende Data BLOB wird von Chrome in einer SQLite Datenbank namens „Login Data“, in frühren Versionen als „Web Data“ bezeichnet, gespeichert. Man findet diese Datenbank im User-Verzeichnis unter C:\Benutzer\<Username>\AppData\Locale\Google\Chrome\ User Data\Default [Boj11]. Insgesamt sind drei Tabellen in der Datenbank enthalten:

- logins
- meta
- stats

In Bezug auf geheime Daten ist die logins-Tabelle am interessantesten. Dort befinden sich ganz ähnliche Angaben wie in der logins.json Datei bei Firefox, wie zum Beispiel die URL der Webseite, ein signon_realm (bei Firefox httpRealm), der Zeitpunkt der Erstellung, die Anzahl der Anmeldungen. Im Unterschied zu Firefox wird der Username selbst in Klartext gespeichert und nur das Passwort verschlüsselt. Alle weiteren Bestandteile der logins-Tabelle sind der Abbildung 2.22 zu entnehmen.

Tabellen (3)		
logins		CREATE TABLE "logins" (origin_url VARCHAR NOT NULL, action_url VARCHAR, usernam
origin_url	VARCHAR	`origin_url` VARCHAR NOT NULL
action_url	VARCHAR	`action_url` VARCHAR
username_element	VARCHAR	`username_element` VARCHAR
username_value	VARCHAR	`username_value` VARCHAR
password_element	VARCHAR	`password_element` VARCHAR
password_value	BLOB	`password_value` BLOB
submit_element	VARCHAR	`submit_element` VARCHAR
signon_realm	VARCHAR	`signon_realm` VARCHAR NOT NULL
preferred	INTEGER	`preferred` INTEGER NOT NULL
date_created	INTEGER	`date_created` INTEGER NOT NULL
blacklisted_by_user	INTEGER	`blacklisted_by_user` INTEGER NOT NULL
scheme	INTEGER	`scheme` INTEGER NOT NULL
password_type	INTEGER	`password_type` INTEGER
times_used	INTEGER	`times_used` INTEGER
form_data	BLOB	`form_data` BLOB
date_synced	INTEGER	`date_synced` INTEGER
display_name	VARCHAR	`display_name` VARCHAR
icon_url	VARCHAR	`icon_url` VARCHAR
federation_url	VARCHAR	`federation_url` VARCHAR
skip_zero_click	INTEGER	`skip_zero_click` INTEGER
generation_upload_status	INTEGER	`generation_upload_status` INTEGER
possible_username_pairs	BLOB	`possible_username_pairs` BLOB
meta		CREATE TABLE meta(key LONGVARCHAR NOT NULL UNIQUE PRIMARY KEY, value LON
stats		CREATE TABLE stats (origin_domain VARCHAR NOT NULL, username_value VARCHAR, c

Abbildung 2.22: Aufbau der SQLite Datenbank Login Data (eigene Quelle)

2.3.3 Internet Explorer

Der Internet Explorer (IE) ist der älteste der vier Browser, wird heute aber immer noch oft verwendet. Erschienen ist der von Microsoft entwickelte Browser erstmals im August 1995 [Hei10]. Zu diesem Zeitpunkt war er jedoch nur im 50 Euro teureren Plus-Paket von Windows 95 enthalten und wurde deshalb nur von sehr wenigen Nutzern verwendet. Erst ein Jahr später mit Version 3.0 sollte der Internet Explorer standardmäßig auf allen zukünftigen Windows-Versionen vorinstalliert und kostenlos sein. Mit der Einführung von Windows 98 ging Microsoft sogar soweit, dass der fest installierte IE auch nicht mehr ohne weiteres zu deinstallieren war, um der Konkurrenz im Bereich der Webbrowser zuvor zu kommen [Kar].

Da der Internet Explorer so eng mit dem Windows Betriebssystem verwoben ist, liegt es nahe, dass auch dieser Browser die Windows DPAPI zur Sicherung der gespeicherten Login-Passwörter nutzt. Im Unterschied zu Chrome macht der IE jedoch Gebrauch von der optionalen Entropie, die der CryptProtectData() Funktion übergeben werden kann. Die Entropie wird dabei aus der URL der Webseite, auf der man sich anmeldet, bestimmt, indem diese URL als UTF16LE-kodierter String mit SHA1 gehasht wird [Jor13].
Eine weitere Besonderheit beim Internet Explorer sind die Speicherorte für die über DPAPI erzeugten Data BLOBs. Je nach Browser Version werden die Daten an unterschiedlichen Stellen und in unterschiedlichen Dateistrukturen abgelegt.

Version 4.0 bis 6.0 Hier liegen die Nutzerdaten im „Protected Storage“ in der Registry. Sofern man Zugriff auf den Protected Storage hat, findet man die Daten unter „HKEY_CURRENT_USER\Software\Microsoft\Protected Storage System Provider“ [Boj11].
Diese Versionen werden ab Windows 7 nicht mehr unterstützt.

Version 7.0 bis 9.0 Bei diesen Versionen wird unterschieden, ob es sich um Autocomplete Passwörter handelt oder um HTTP-Authentication-Passwörter.
Im ersten Fall werden die Daten in der Registry unter dem Key „HKCU\Software\Microsoft\Internet Explorer\IntelliForms\Storage2“ gespeichert. Dort ist für jedes Passwort ein Key-Value-Paar zu finden, wobei als Key der SHA1-Hash der zugehörigen URL verwendet wird. Der Wert entspricht einem DPAPI BLOB und kann demnach ganauso entschlüsselt werden wie der Password-BLOB in Chromes ’Login Data’ mit dem Unterschied, dass der SHA1-Hash der UTF16LE-kodierten URL als Entropie verwendet wird.
Die HTTP Authentication Passwörter hingegen liegen im Nutzerverzeichnis unter C:\Benutzer\<Username>\AppData\Local\Microsoft\Credentials zusammen mit anderen Passwörtern [Jor13]. Ein solches Credentialfile ist im Grunde genommen ein normaler DPAPI BLOB, der mit einem 12 Byte Header eingeleitet wird. Innerhalb des Headers findet sich die Größe des BLOBs an Offset 0x04, wie aus Abbildung 2.23 ersichtlich ist. Die verschlüsselten Daten innerhalb des Blobs repräsentieren

nach der Entschlüsselung jedoch noch nicht das gesuchte Login Passwort, sondern einen weiteren Blob, dessen Daten ebenfalls mit einer zusätzlichen Entropie verschlüsselt sind.

Abbildung 2.23: Aufbau eines Credentialfiles unter %APPDATA%\Local\Microsoft\Credentials [Pic16]

Version 10.0 bis 11.0 Seit Windows 8 werden alle Autocomplete Passwörter im Windows Vault gespeichert. Die neuen IE-Versionen laufen zwar ebenfalls unter Windows 7, jedoch werden die Passwörter hier weiterhin in der Registry abgelegt [Nir]. Die Vault Dateien liegen im Verzeichnis C:\Benutzer\<Username>\AppData\Local\Microsoft\Vault. Dort befinden sich ein oder mehrere Verzeichnisse, die mit einer GUID benannt sind. Sie enthalten wiederum 3 Typen von Dateien [Pic16]:

- Policy.vpol
- <GUID>.vsch
- <GUID>.vcrd

Die Datei ***Policy.vpol*** ist immer nur einmal vorhanden und beinhaltet die Schlüssel, um alle vcrd-Dateien zu entschlüsseln. Die Schlüssel sind mit DPAPI geschützt und liegen demnach als BLOB vor. Den Beginn der Datei bildet ein Header mit der GUID des Vaults und einer UTF16-kodierten Beschreibung. Die genaue Struktur ist in Abbildung 2.24 ersichtlich.

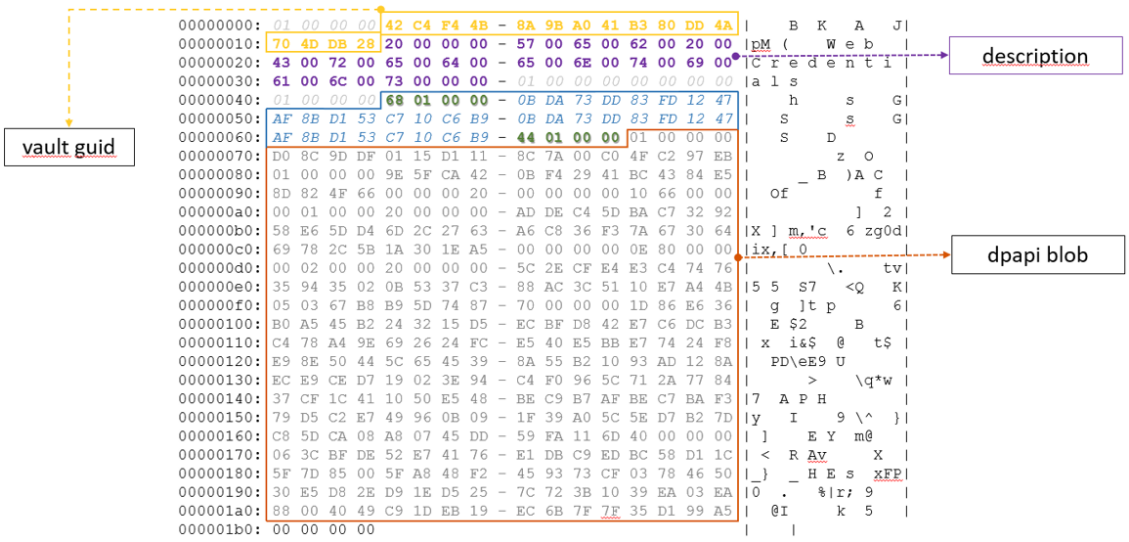


Abbildung 2.24: Struktur der Policy.vpol [Pic16]

Wurde der DPAPI BLOB erfolgreich entschlüsselt, erhält man 96 Byte. Die ersten 40 beinhalten einen 16 Byte langen Key für AES-128, während die übrigen 56 Byte einen 32-Byte-Key für AES-256 enthalten, wie aus Abbildung 2.25 hervorgeht.

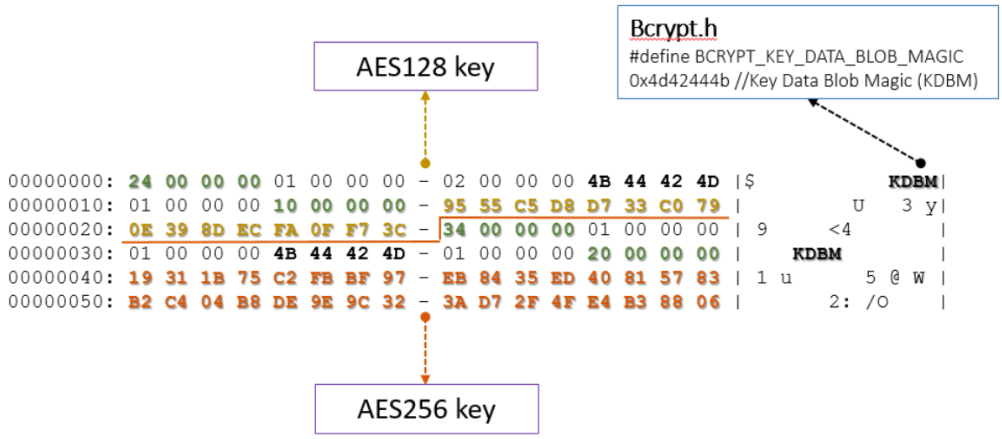


Abbildung 2.25: Entschlüsselter DPAPI BLOB aus Policy.vpol mit AES Keys [Pic16]

Die gewonnenen Informationen können nun genutzt werden, um die ***<GUID>.vcrd***-Dateien zu entschlüsseln. Diese haben einen etwas komplexeren Aufbau mit verschlüsselten Attributen variabler Länge. Den Anfang der Datei bildet eine 16 Byte lange GUID, die kennzeichnet, welcher vsch-Datei sie zuzuordnen ist. Es folgt ein Zeitstempel der letzten Aktualisierung und eine kurze Beschreibung. Es schließt sich eine Art Inhaltsverzeichnis für die im Anschluss aufgeführten Attribute an. Ein Eintrag in diesem Inhaltsverzeichnis ist immer 12 Byte lang und hat folgenden Aufbau:

0x00	4 Byte	ID des Attributs
0x04	4 Byte	Offset zum Beginn des Attributs
0x08	4 Byte	unbekannt (meist 0x00000000)

Tabelle 2.9: Struktur eines Eintrags im 'Attribut-Inhaltsverzeichnis' [Pic16]

Anhand der Offsets lassen sich nun die Attribute lokalisieren. Jedes Attribut beginnt mit seiner ID. Danach folgen 12 Byte unbekannter Bedeutung und die Länge der eigentlichen Daten. Es ist zu beachten, dass zu dieser Länge ein zusätzliches Byte zählt, das den verschlüsselten Daten vorangeht [Pic16].
Die Attribute sind alle einzeln per AES-128 verschlüsselt und können nun mithilfe des AES-128-Keys aus der Policy.vpol entschlüsselt werden. Eine Ausnahme bildet das letzte Attribut, welches das Passwort beinhaltet. Dieses Attribut ist mit AES-256 im CBC Modus verschlüsselt. Deshalb ist den verschlüsselten Daten noch ein IV vorangestellt, der nun zusammen mit dem AES-256-Key aus der Policy.vpol zur Entschlüsselung des Passwortes verwendet wird [Pic16].
Zusammenfassend sind die Zusammenhänge anhand eines Beispiels in Abbildung 2.26 dargestellt.

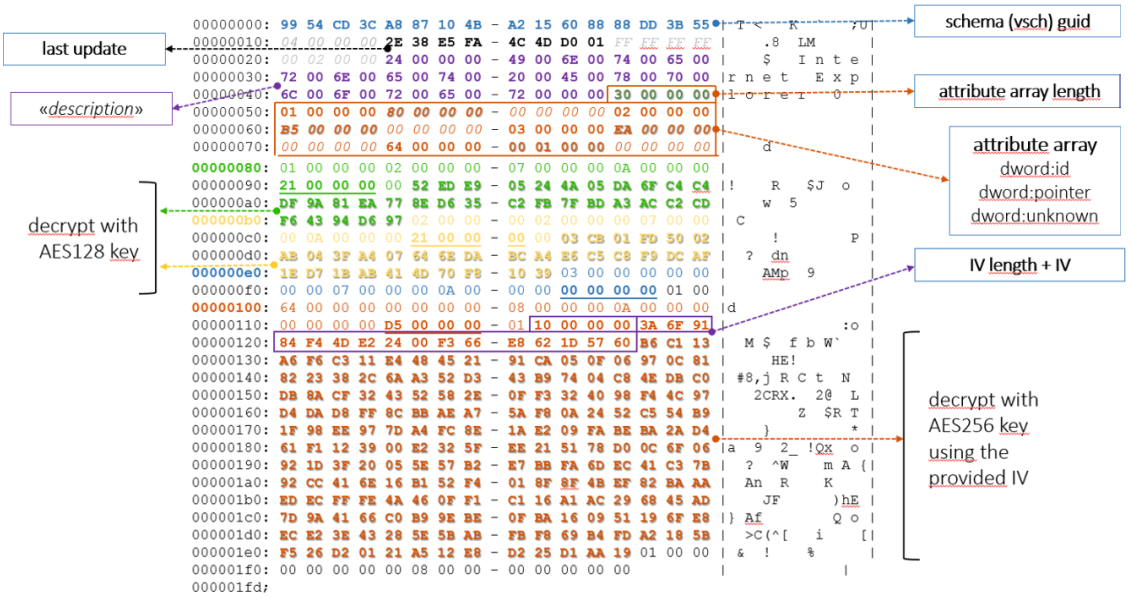


Abbildung 2.26: Struktur einer vcrd-Datei [Pic16]

Neben der vpol- und den vcrd-Dateien existieren noch weitere Dateien, benannt mit ***<GUID>.vsch***. Diese repräsentieren jeweils ein sogenanntes Vault-Schema, unter dem eine oder mehrere vcrd-Dateien zusammengefasst werden. Ein solches Schema beinhaltet eine Beschreibung der Daten, diverse Flags und weitere Systeminformationen [Pasb]. Welche vcrd-Dateien zu einem Vault-Schema gehören, ist

in den vcrd-Dateien selbst über die GUID des Schemas definiert, die sich auch in der entsprechenden vsch-Datei wiederfindet. Die Aufgabe eines Vault-Schemas besteht lediglich darin, die entschlüsselten Daten korrekt darzustellen [Pic16]. Einen Windows Vault kann man sich somit als eine Art Baum vorstellen, wie Abbildung 2.27 verdeutlicht.

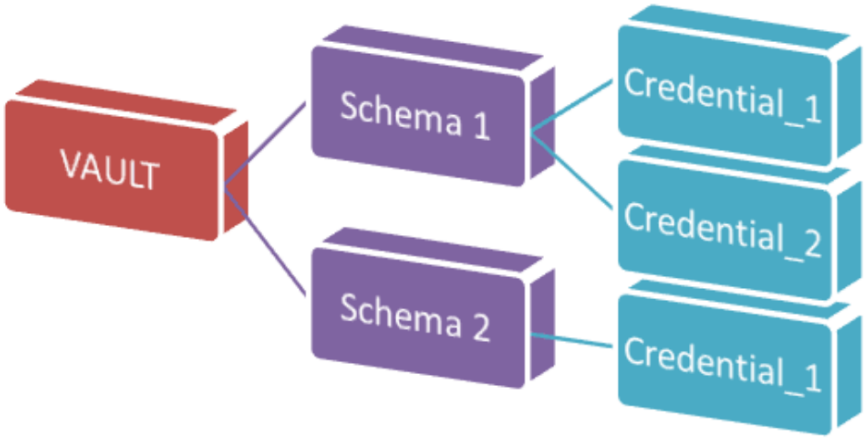


Abbildung 2.27: Struktur des Windows Vaults [Pasb]

2.3.4 Microsoft Edge

Unter dem Codenamen „Spartan“ gab Microsoft im Januar 2015 bekannt, einen völlig neuen Webbrowser zu entwickeln, der den Internet Explorer ablösen sollte. Veröffentlicht wurde der neue Browser schließlich mit dem Betriebssystem Windows 10 unter dem Namen „Edge“ [War15]. Der Internet Explorer bleibt dennoch auch weiterhin verfügbar, seine Rolle als Default-Browser übernimmt jedoch nun Edge.
Obwohl sich der neue Webbrowser weitestgehend von seinem Vorgänger abgrenzen sollte, wurde die Art der Passwortspeicherung beibehalten. Somit legt auch Edge seine Passwörter mit DPAPI verschlüsselt im Windows Vault unter C:\Benutzer\<Username>\AppData\Local\Microsoft\Vault ab.

3 Methoden

Dieses Kapitel beschäftigt sich mit der Konzipierung eines Tools, das mit Unterstützung bereits existenter Programme den Prozess der Extraktion und Entschlüsselung von Browser-Anmeldeinformationen aus Windowssystemen automatisiert und vereinfacht. Vorausgesetzt wird in Bezug auf Chrome und Internet Explorer bzw. Edge die Kenntnis des Nutzerpasswortes. Um das Tool eindeutig zu benennen, wird es in den weiteren Ausführungen als „PasswordXTract“ bezeichnet.

Wie die Statistik 3.1 belegt, ist derzeit Windows 10 deutschlandweit das am häufigsten verwendete Betriebssystem. Besonders auffällig ist eine Achsensymmetrie zwischen den Graphen von Windows 7 und 10, was bedeutet, dass sich viele Windows 7 Nutzer zugunsten von Windows 10 von ihrem alten Betriebssystem getrennt haben, was nicht zuletzt auch durch das kostenlose Upgrade bedingt ist. Zudem wird Microsoft den Support von Windows 7 am 14. Januar 2020 einstellen und empfiehlt deshalb ausdrücklich den Wechsel auf Windows 10 [Mic18]. Aus diesem Grund fokussiert sich PasswordXTract zunächst auf die Entschlüsselung der Passwörter aus Windows-Versionen 8 bis 10.

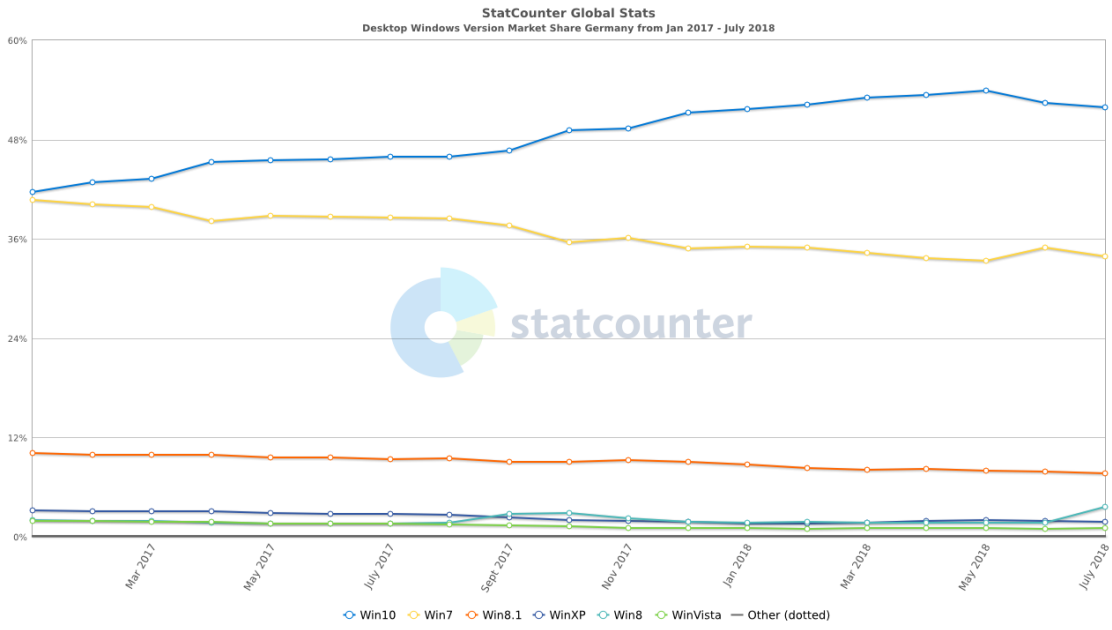


Abbildung 3.1: Marktanteil der Windows Versionen in Deutschland [Sta18a]

3.1 Vorbetrachtungen

In den folgenden Abschnitten sollen zum einen die Phasen der Extraktion von verschlüsselten Passwortdaten und zum anderen die Dechiffrierung derselben näher betrachtet werden. Es wird davon ausgegangen, dass eine forensische Datensicherung in Form eines Images auf einem Datenträger vorliegt. Die Extraktion bezieht sich vor allem auf das Mounten dieses Images und auf die verschiedenen Speicherorte relevanter Dateien. Die Phase der Entschlüsselung meint hier vorrangig die Dechiffrierung von DPAPI, was sowohl für den Internet Explorer bzw. Edge als auch für Chrome von wesentlicher Bedeutung ist. Ein Schwerpunkt liegt hierbei auf der Bewertung bereits vorhandener Software in Bezug auf deren Integrationsfähigkeit in PasswordXTract. Die Einschätzung erfolgt über die nachfolgend genannten drei Kriterien. Die Entschlüsselung von Firefox Passwörtern wurde ohne Unterstützung eines externen Programms implementiert, weshalb eine nähere Beschreibung in Abschnitt 3.4.3 erfolgt.

Automatisierung. Das zentrale Ziel dieser Arbeit ist es, die Prozesse der Passwortextraktion für verschiedene Browser in einem einzigen Programm zusammenzufassen und die Verarbeitung soweit wie möglich zu automatisieren, so dass im Idealfall nur ein Knopfdruck nötig ist, um eine übersichtliche Liste mit allen gespeicherten Passwörtern, Nutzernamen und URLs zu erhalten.
Aus diesem Grund wurden alle Methoden, Skripte oder Programme, die im Folgenden vorgestellt werden, besonders auf den Gesichtspunkt der Automatisierung hin überprüft und ausgewählt. Das bedeutet insbesondere, dass Programme mit graphischen Oberflächen für diese Zwecke eher ungeeignet sind und Kommandozeilen-Anwendungen bevorzugt wurden.

Nachvollziehbarkeit. Neben der Automatisierung spielt der Aspekt der Nachvollziehbarkeit eine wichtige Rolle. Ziel dieser Arbeit soll also keine weitere „Black Box“ sein, sondern vielmehr die Erstellung eines quelloffenen Programmes, bei dem vor allem die zentralen Schritte der Entschlüsselung nachvollziehbar sind.

Post-Mortem-Analyse. Das letzte wichtige Kriterium ist es, dass die Bearbeitung nicht am originalen System erfolgt, sondern als Post-Mortem-Analyse auf einer forensischen Datensicherung, um eine Unverändertheit der originalen Daten zu garantieren.

3.2 Extraktion von Schlüsseln und Passwörtern

3.2.1 Mounten des forensischen Images

Ein Image ist ein Abbild eines Datenträgers. Um die enthaltenen Dateien lesen und analysieren zu können, kann das Image entweder in eine Forensik-Suite wie X-Ways eingelesen oder direkt als Laufwerk im Auswertesystem eingebunden (gemountet) werden.

Die erste Methode ist für eine automatisierte Extraktion eher ungeeignet. Zwar unterstützen einige Forensik-Suites Scripting bzw. selbst geschriebene Erweiterungen, doch da man auch mit eigenen Skripten größtenteils nur innerhalb der Suite agieren kann, sind die Möglichkeiten hier als begrenzt einzuschätzen.
Beim Mounten wird das Image ein Teil des Dateisystems, der wie eine USB-Stick oder eine normale Festplatte zugreifbar ist. Der Zugriff auf die einzelnen Dateien wird so ohne ein zusätzliches Hilfsmittel möglich und in einer beliebigen Programmiersprache realisierbar.

Unter Windows stehen einige Programme zum Download zur Verfügung, die ein Image als Laufwerk einbinden können.
Unter dem Gesichtspunkt der Automatisierung ist OSFMount von PassMark Software ein geeignetes Werkzeug. Es kann über die Kommandozeile angesteuert werden und unterstützt zudem eine Vielzahl an geläufigen Image-Formaten, wie IMG, DD, ISO, VHD und E01 [Sof18]. Weiterhin ist OSFMount ein freies und eigenständiges Tool und beinhaltet ausschließlich die benötigten Funktionalitäten zum Mounten, wodurch sich die Größe auf etwa 7 MB beschränkt.
Das Einbinden des Images geschieht über eine einzelne Kommandozeilen-Operation:

```
osfmount.com -a -t file -m <drive> -f <path_to_image> -o ro
```

`osfmount.com` Soll OSFMount über die Kommandozeile ausgeführt werden, erfolgt der Aufruf nicht über „osfmount.exe“, sondern mittels „osfmount.com“.

Option `-a` Diese Option bewirkt, dass das Image als virtuelles Laufwerk gemountet wird.

Option `-t` Die `-t`-Option legt fest, in welcher Form die virtuelle Festplatte gespeichert wird; `file` gibt in diesem Fall an, dass es sich um eine Datei handelt.

Option `-m` Über `-m` wird der Mountpoint festgelegt, also beispielsweise ein Laufwerksbuchstabe.

Option `-f` Dieser Angabe folgt der Pfad zu einer Image-Datei.

Option `-o` Diese Option regelt den Lese- bzw. Schreibzugriff auf die virtuelle Festplatte. `ro` steht in diesem Falle für read only.

3.2.2 Sammlung relevanter Dateien

Die Abspeicherung der Passwortdaten erfolgt durch den Browser selbst, deshalb sollten alle benötigten Dateien an ihrem standardmäßigen Speicherort liegen. Prinzipiell kann man 3 verschiedene Ablagestrukturen unterscheiden, wie Tabelle 3.1 veranschaulicht. Während Chrome und Firefox für alle Versionen ähnliche Dateien und Strukturen verwenden, unterscheiden sich die Ablagestrategien des Internet Explorers sehr stark von Version zu Version. Aktuell speichert IE seine Passwörter im Verzeichnis „Vault“, ebenso wie Edge.
Weiterhin greifen Chrome, Internet Explorer und Edge auf DPAPI zurück. Somit verwenden diese drei Browser auch die selben Masterkeys, die im Verzeichnis „Protect“ liegen.

Browser	**relevante Dateien**	**Speicherort**
Chrome	Login Data	%userprofile%\AppData\Local\ Google\Chrome\UserData\Default
	Protect\	%userprofile%\AppData\Roaming\ Microsoft
Firefox	key3.db/key4.db	%userprofile%\AppData\Roaming\ Mozilla\Firefox\Profiles\<ID>.<username>
	logins.json	%userprofile%\AppData\Roaming\ Mozilla\Firefox\Profiles\<ID>.<username>
IE Vault	Vault\	%userprofile%\AppData\Local\ Microsoft\Vault
	Protect\	%userprofile%\AppData\Roaming\ Microsoft

Tabelle 3.1: Übersicht über alle benötigten Dateien und deren Speicherorte

Die Gesamtgröße aller relevanten Dateien beläuft sich auf wenige kB bis MB, abhängig von den installierten Browsern auf dem Zielsystem und der Anzahl der gespeicherten Passwörter. Dadurch können die Dateien auch auf die Festplatte des Auswertesystems kopiert werden, um bei der weiteren Bearbeitung leichter darauf zugreifen zu können. So werden auch eventuelle Leerzeichen in Pfadangaben vermieden, die sonst bei bestimmten Kommandozeilen-Aufrufen zu Problemen führen würden.

3.3 Überwinden der Data Protection API

Die Abläufe innerhalb der DPAPI sind im Vergleich zu Firefox überaus komplex und nur sehr spärlich dokumentiert. Dennoch existieren bereits einige Programme, die die Entschlüsselung von DPAPI beherrschen:

Data Protection Decryptor (DPD) von NirSoft. Diese Applikation ermöglicht sowohl eine Entschlüsselung von DPAPI-Blobs auf dem laufenden System als auch von externen Blobs anderer Benutzer bzw. PCs. In beiden Fällen müssen das Windows-Anmeldepasswort und der Pfad zu der zu entschlüsselnden Blob-Datei oder wahlweise der Blob selbst in hexadezimaler Form angegeben werden. Wenn es sich um einen Blob von einem externen PC oder anderen Benutzer handelt, muss zusätzlich der Speicherort des „Protect"-Verzeichnisses eingetragen werden. Der Data Protection Decryptor lässt sich ausschließlich über eine grafische Benutzeroberfläche bedienen und ist nicht als Open Source verfügbar.

dpapick von Jean-Micael Picod. Dpapick wird als das erste Open-Source-Tool bezeichnet, das in der Lage ist, DPAPI-Blobs offline und plattformunabhängig zu entschlüsseln [Pic]. Es wurde in Python 2.7 implementiert und erschien im Oktober 2014 in der dritten Version. Seitdem wurde dpapick leider nicht verändert bzw. aktualisiert [Pic]. Die Bedienung erfolgt über die Kommandozeile, in der die Pfade zu Masterkey(s), CREDHIST und Blob sowie die SID und das Benutzerpasswort angegeben werden müssen, um den gegebenen Blob zu entschlüsseln.

Windows Password Recovery (WPR) von Passcape Software. Dieses Toolkit ist eine Sammlung vieler Werkzeuge für Windows rund um das Thema Passwörter. In Bezug auf DPAPI besteht die Möglichkeit, den Rechner nach Blobs zu durchsuchen und deren Struktur aufzuschlüsseln. Weiterhin bietet WPR auch die Entschlüsselung von DPAPI-Blobs an. Ähnlich wie bei NirSofts Data Protection Decryptor erfolgen auch hier alle Pfadangaben und die Eingabe des Benutzerpassortes allein über eine grafische Oberfläche. Dabei wird für jede Funktionalität ein eigener Wizard eröffnet, in dem die benötigten Daten in mehreren Schritten eingetragen werden. Weiterhin ist WPR ein kommerzielles Produkt und kostet derzeit $65. Die frei verfügbare Testversion hat einige funktionelle Einschränkungen, so werden beispielsweise nur die ersten 3 Buchstaben der entschlüsselten Passwörter angezeigt.

Mimikatz von Benjamin Delpy. Mimikatz ist ein in C implementiertes Tool, das von Benjamin Delpy, auch unter dem Pseudonym „Gentil Kiwi" bekannt, entwickelt wurde. Delpy befasste sich eingehend mit der Sicherheit von Windows und hielt seine Erkenntnisse schließlich in Mimikatz fest. Sein Programm besteht dementsprechend aus mehreren sogenannten Modulen, die sich mit unterschiedlichen Sicherheitsmechanismen unter Windows auseinandersetzen, wie zum Beispiel Kerberos, LSA und auch DPAPI. Mimikatz ist für die Bedienung über die Kommandozeile konzipiert und arbeitet mit einer fest vorgeschriebenen Syntax, die nur zum Teil ausführlich dokumentiert ist.

Im Hinblick auf die Vorbetrachtungen in Abschnitt 3.1 erfüllen nur zwei der bisher vorgestellten Programme die Kriterien Automatisierung, Nachvollziehbarkeit und Post-Mortem-Analyse. Besonders ungeeignet sind die Closed-Source-Programme mit einer grafischen Benutzeroberfläche, die nicht automatisierbar und zudem undurchsichtig für den Benutzer sind. Die beste Funktionalität bieten dpapick und Mimikatz, wobei erste-

res augenscheinlich in letzter Zeit kaum noch gepflegt wurde. Zusammenfassend ist die Beurteilung der Programme in folgender Tabelle dargestellt:

Programm	**Automatisierung**	**Nachvollziehbarkeit**	**Post-Mortem-Analyse**	**Aktualität**
DPD	-	-	+	+
dpapick	+	+	+	-
WPR	-	-	+	+
Mimikatz	+	+	+	+

Tabelle 3.2: Beurteilung bereits existenter Programme zur Entschlüsselung von DPAPI, unter anderem in Bezug auf die Kriterien aus Abschnitt 3.1 (- = nicht erfüllt, + = erfüllt)

Damit erfüllt Mimikatz als einziges alle gegebenen Anforderungen. Durch die Bedienung über die Kommandozeile können alle Prozesse in einem Skript automatisiert werden, der Quellcode ist offen verfügbar, so dass man nachvollziehen kann, was im Hintergrund passiert und Mimikatz ist nicht an das originale System gebunden, so dass Daten fremder PCs und Nutzer ebenfalls analysiert werden können. Im Unterschied zu dpapick unterliegt Mimikatz ständiger Bearbeitung und stellt im Abstand weniger Monate neue, verbesserte bzw. erweiterte Versionen zur Verfügung. Der einzige Nachteil besteht darin, dass Mimikatz ausschließlich für Windows konzipiert ist und eine Auswertung auf Linux oder MAC demnach nur über Umwege möglich ist.

Wie bereits erwähnt bietet Mimikatz eine Vielzahl an Modulen, die wiederum eigene Kommandos bereitstellen. Allgemein ist ein Mimikatz Aufruf wie folgt aufgebaut:

```
mimikatz.exe modul::kommando [parameter]
```

Für die Entschlüsselung von DPAPI stellt Mimikatz ein separates DPAPI-Modul zur Verfügung. Einige ausgewählte Kommandos dieses Moduls werden nachfolgend genauer beschrieben:

protect. Mimikatz bietet die Möglichkeit, Daten mittels DPAPI zu verschlüsseln. Der Nutzer kann dabei optionale Angaben zu Beschreibung, Flags, Entropie und Ausgabedatei machen. Der Masterkey wird vom ausführenden Benutzer bezogen. Das Kommando zur Verschlüsselung der Zeichenkette „Test“ ohne Entropie lautet:

```
mimikatz.exe dpapi::protect /data:,,Test‘‘
```

masterkey. Der erste Schritt jeder Entschlüsselung ist die Bestimmung des Masterkeys. Mimikatz benötigt dafür mindestens den Pfad zu der entsprechenden Masterkey-Datei und das korrekte Benutzerpasswort bzw. dessen Hash. Eine optionale Angabe ist unter anderem die SID. Sie wird für die Entschlüsselung zwingend benötigt und ist meist im Masterkey-Pfad enthalten. Sollte dies nicht der Fall sein, muss sie explizit angegeben werden. Nach erfolgreicher Entschlüsselung gibt Mimikatz sowohl den Masterkey als auch dessen Hash zurück. Die Befehlssyntax gestaltet sich wie folgt:

```
mimikatz.exe dpapi::masterkey /in:<path_to_masterkey>
/password:<userpassword> [/sid:<SID>]
```

cache. Mimikatz verwaltet bereits entschlüsselte Daten, wie z.B. Masterkeys in einem eigenen Cache. So können andere Kommandos im Hintergrund auf bereits vorhandene Informationen zugreifen, ohne dass der Benutzer Zwischenergebnisse selbst speichern muss. Der Cache bleibt solange erhalten bis Mimikatz mit `exit` beendet wird. Soll der momentane Inhalt des Caches geprüft werden, geschieht das mittels:

```
mimikatz.exe dpapi::cache
```

blob. Bei der Entschlüsselung eines Blobs greift Mimikatz selbstständig auf den Cache zu und überprüft, ob der benötigte Masterkey dort bereits enthalten ist. Sollte dies nicht der Fall sein, kann er auch als Parameter übergeben werden. Mimikatz benötigt außerdem den Pfad zu dem chiffrierten Blob und die Option „/unprotect“, um die Entschlüsselung einzuleiten. Falls der Blob zusätzlich mit einer Entropie geschützt wurde, kann diese ebenfalls angegeben werden. Für einen Blob ohne Entropie, dessen Masterkey bereits im Cache liegt, lautet das Mimikatz-Kommando:

```
mimikatz.exe dpapi::blob /in:<path_to_blob> /unprotect
```

credhist. In der Credential History werden alle bisher verwendeten Benutzerpasswörter, verschlüsselt mit dem aktuellen Passwort, abgelegt. Übergibt man Mimikatz den Pfad zur CREDHIST-Datei, die SID des Benutzers und das aktuelle Nutzerpasswort, bestimmt Mimikatz aus diesen Angaben den NTLM- und den SHA1-Hash zu jedem enthaltenen Passwort. Alternativ kann statt des Passwortes in Textform auch dessen Hash angegeben werden, was durch die Eingabe „/hash:<userpassword_hash> anstelle von „/password:<userpassword>“ veranlasst wird. Das korrespondierende Kommando ist wie folgt aufgebaut:

```
mimikatz.exe dpapi::credhist /in:<path_to_credhist>
/sid:<SID> /password:<userpassword>
```

cred. Da die Credentials, die beispielsweise für Internet Explorer-HTTP-Authentifizierungspasswörter angelegt werden, keine normalen Blobs sind, bietet Mimikatz hierfür ein extra Kommando zur Entschlüsselung an. Sofern die benötigten Masterkeys bereits im Cache liegen, fordert Mimikatz lediglich den Pfad zu einer verschlüsselten Credential-Datei in folgender Form:

```
mimikatz.exe dpapi::cred /in:<path_to_credential-file>
```

vault. Ähnlich wie bei den Credentials werden auch die Dateien im Vault-Verzeichnis gesondert behandelt. Hier ist ebenfalls der Pfad zu einer gewünschten .vcrd-Datei aus dem Vault-Verzeichnis anzugeben. Dazu kommt ein weiterer Pfad, der auf die zugehörige Policy-Datei verweist. Die Übergabe dieser Parameter erfolgt nach folgender Syntax:

```
mimikatz.exe dpapi::vault /cred:<path_to_vcrd-file>
/policy:<path_to_vpol-file>
```

chrome. Chrome speichert in seiner „Login Data“-Datei mehrere Blobs innerhalb einer SQLite Datenbank. Mimikatz ist in diesem speziellen Fall in der Lage, alle Blobs selbstständig und nacheinander zu entschlüsseln. Als Ergebnis wird je Blob ein Datensatz ausgegeben, der die URL der aufgerufenen Webseite, den Benutzernamen und das zugehörige Passwort enthält. Unter der Voraussetzung, dass alle benötigten Masterkeys im Cache liegen, kann Mimikatz die „Login Data“-Datei mittels folgendem Befehl verarbeiten:

```
mimikatz.exe dpapi::chrome /in:<path_to_login_data>
/unprotect
```

3.4 Aufbau und Ablauf von PasswordXTract

PasswordXTract besteht aus mehreren einzelnen Skripten, die fast alle von einem zentralen Skript aufgerufen werden. Dadurch wird gewährleistet, dass Teile des Programmes auch separat aufgerufen werden können und das Programm auch dann im Hauptskript fortgesetzt wird, wenn in einem untergeordneten Programmteil ein Fehler auftritt. Die Kommunikation zwischen den einzelnen Skripten erfolgt durch Kommandozeilenparameter, die beim Aufruf übergeben werden.
Der Programmablauf gestaltet sich fünfstufig. Folglich können auch die Skripte in fünf Kategorien eingeteilt werden:

- Skripte zur Installation benötigter Software
- Skript zum Kopieren verschlüsselter Login-Daten vom Image auf die lokale Festplatte
- Skripte zur Entschlüsselung der Login-Daten
- Skripte zur Aufbereitung und übersichtlichen Ausgabe der Ergebnisse
- Skript zum Speichern der Ergebnisse und Löschen nicht mehr benötigter Daten und Programme

In PasswordXTract kommen zwei verschiedene Programmiersprachen zur Anwendung: Batch und Ruby. Für die meisten Programmteile wird Batch verwendet, womit Operationen wie der Download und die Installation von Software oder das Kopieren von Dateien leicht zu realisieren sind. Da sich auch Mimikatz über die Kommandozeile steuern lässt, ist auch die automatische Abarbeitung mehrerer Kommandos mit Batch einfach umzusetzen. Ruby hingegen findet dort Anwendung, wo Dateien an bestimmten Offsets gelesen werden müssen und Operationen mit Strings auszuführen sind. Auch reguläre Ausdrücke lassen sich mit Ruby um Vieles einfacher umsetzen als in einem Batch-Skript. Somit findet Ruby vor allem bei der Aufbereitung der entschlüsselten Login-Daten zu einer übersichtlichen Ergebnisliste und bei der Dechiffrierung von Firefox-Passwörtern Verwendung. Die Zusammenhänge zwischen den einzelnen Skripten sind in Übersicht 3.2 dargestellt.

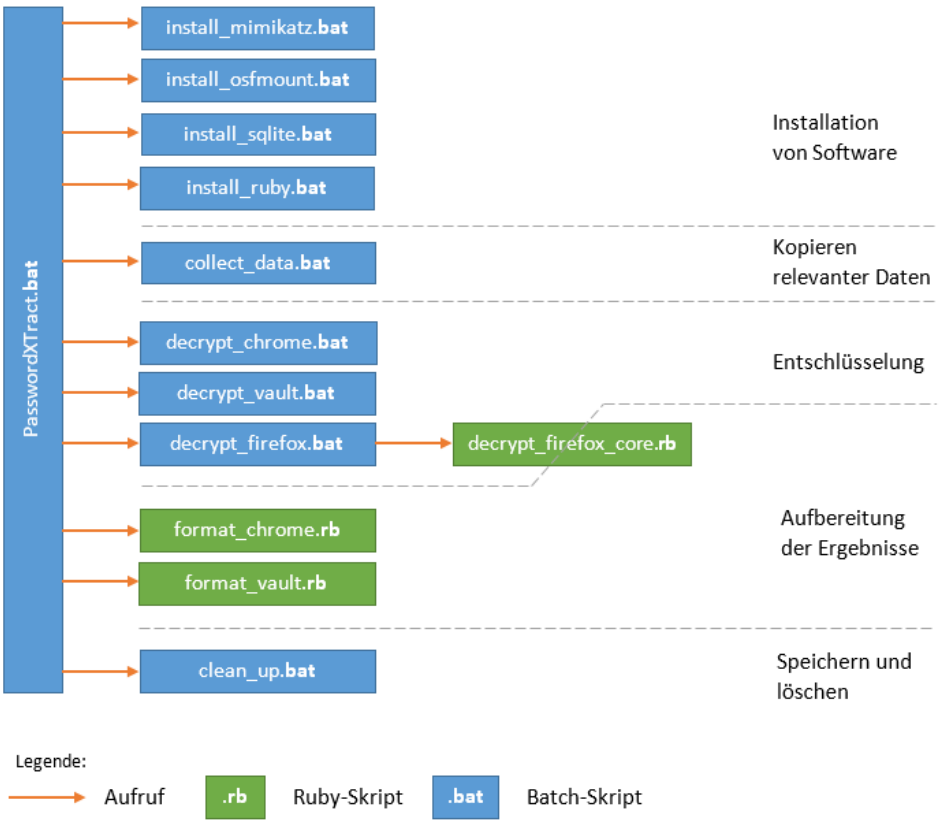


Abbildung 3.2: Zusammenhang zwischen den Skripten von PasswordXTract (eigene Quelle)

3.4.1 Installation benötigter Software

Um alle Funktionen bereitstellen zu können, benötigt PasswordXTract einige Programme von Drittanbietern, deren Größe sich insgesamt auf knapp 790 MB beläuft.

OSFMount kommt beim Mounten des Images zum Einsatz. Das Tool der Firma PassMark Software wurde als Ergänzung zu deren kommerziellen OSForensics-Toolkit konzipiert, steht aber als eigenständiges Programm kostenlos zur Verfügung. PasswordXTract nutzt Version 2.0.1001 vom 21. März 2018.

Mimikatz wird für die Entschlüsselung von Chrome und IE bzw. Edge benötigt. Vor der Installation empfiehlt es sich, den Virenscanner abzuschalten, da Mimikatz als Malware erkannt werden könnte und deshalb unter Umständen direkt nach dem Download wieder gelöscht wird. In PasswordXTract findet Mimikatz Version 2.1.0 20180616 vom 16. Juni 2018 Verwendung.

Ruby ist erforderlich, um die Skripte zur Aufbereitung der Entschlüsselungsergebnisse auszuführen und um die Passwörter des Firefox Browsers zu dechiffrieren. PasswordXTract nutzt Ruby in der Version 2.5.1-2 vom 24. Juni 2018.

SQLite Tools findet Verwendung, um Firefox-Daten aus der key4.db Datenbank zu extrahieren. SQLite Tools arbeitet ausschließlich über die Kommandozeile und kann so unter anderem SELECTs und andere SQL-Kommandos interpretieren. Für PasswordXTract wird SQLite Tools in der Version 3.24.0 vom 4. Juni 2018 genutzt.

Von diesen Programmen kommen OSFMount und Ruby bei allen Browsern zum Einsatz, womit beide obligatorisch sind. Mimikatz hingegen muss nur installiert sein, wenn Chrome- oder IE- bzw. Edge-Passwörter untersucht werden sollen und SQLite Tools nur, wenn die Entschlüsselung von Firefox-Passwörtern beabsichtigt wird.

PasswordXTract verwendet für den Download und die Installation jedes Programmes ein gesondertes Skript. Dazu zählen „install_mimikatz.bat“, „install_osfmount.bat“, „install_sqlite.bat“ und „install_ruby.bat“. Diese vier Skripte werden nacheinander vom zentralen Skript „PasswordXTract.bat“ aufgerufen.
Der Download läuft über das `certutil`-Kommando ab, nachdem der Nutzer diesem Vorgang mit „y“ zugestimmt hat. Mimikatz und SQLite Tool werden mit `jar` entpackt, während die Installation von Ruby und OSFMount über ein eigenes Setup erfolgt.
PasswordXTract installiert alle benötigten Programme unter „C:\PasswordXTract“. Dort wird für jedes Programm ein eigener Ordner angelegt. Alle PasswordXTract-Skripte müssen ebenfalls in diesem Verzeichnis in einem Order namens „bin“ liegen, um die korrekte Funktion des Programmes zu gewährleisten.

3.4.2 Benutzereingaben

Bevor PasswordXTract mit der Extraktion und Entschlüsselung beginnen kann, benötigt es noch einige Information, die durch den Benutzer eingegeben werden müssen:

- der Pfad zu dem Image, von dem Logindaten extrahiert werden sollen
- der Laufwerksbuchstabe, an dem das Image gemountet werden soll
- der Benutzername, der im Image genutzt wird
- die Browser, deren Passwörter entschlüsselt werden sollen

Abhängig von der Auswahl der Browser, gibt es noch zwei weitere Angaben:

- das Anmeldepasswort des Nutzers (nur bei Chrome und IE/Edge)
- das Masterpasswort für Firefox (optional)

Die Benutzereingaben werden in Variablen gespeichert, so dass sie beim Aufruf der anderen Skripte als Parameter übergeben werden können. Diese Abfragen erfolgen Schritt für Schritt zusammen mit informativen Ausgaben zum Ablauf, wie in Abbildung 3.3 dargestellt. Die gelb hinterlegten Begriffe sind beispielhafte Benutzereingaben, von denen die ersten drei durch PasswordXTract auf Plausibilität geprüft werden, das heißt, der Pfad zum Image sollte gültig sein, der Laufwerksbuchstabe noch nicht vergeben und der Benutzer muss existieren. Ist eine Angabe nicht korrekt, wird der Nutzer zur wiederholten Eingabe aufgefordert.

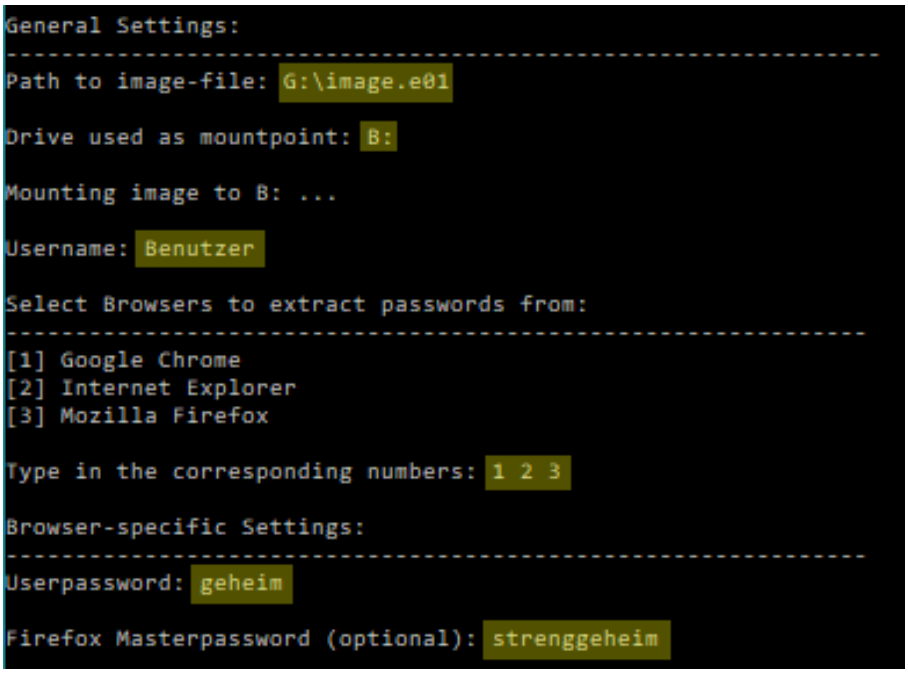


Abbildung 3.3: Interaktiver Programmteil von PasswordXTract (eigene Quelle)

3.4.3 Lokale Ablage relevanter Dateien

PasswordXTract kopiert alle Dateien, die für die Entschlüsselung von Login Daten benötigt werden, vom Image nach „C:\PasswordXTract\Files“. Dort befinden sich nach dem Kopieren bis zu vier Ordner, namentlich „DPAPI“, „Chrome“, „FF“ und „IE“, abhängig davon, welche Browser untersucht werden sollen. Nach jedem Programmstart wird zunächst der gesamte „Files“-Ordner gelöscht und beim ersten Kopiervorgang neu angelegt.
Das Kopieren der Dateien erfolgt in „collect_data.bat“. Beim Aufruf dieses Skriptes in „PasswordXTract.bat“ werden zusätzlich 5 Parameter übergeben. Die ersten drei legen die auszuwertenden Browser fest, der vierte Parameter repräsentiert den Laufwerksbuchstaben, unter dem das Image gemountet wurde, und der fünfte übergibt den Benutzernamen. Nicht immer bedarf es der Auswertung aller Browser. In diesem Fall können bis zu zwei der ersten drei Parameter leer sein. Die übergebenen Parameter werden in „collect_data.bat“ zu Beginn ausgewertet, wie folgender Quellcode-Ausschnitt zeigt:

```
:: 4. Parameter = Laufwerksbuchstabe
set drive=%4
:: 5. Parameter = Benutzername
set username=%5
:: Parameter >5 gehoeren zu mehrteilige Benutzernamen
IF NOT "%6"=="" set username=%5 %6
IF NOT "%7"=="" set username=%5 %6 %7
IF NOT "%8"=="" set username=%5 %6 %7 %8
IF NOT "%9"=="" set username=%5 %6 %7 %8 %9

:get_chrome_param
:: 1. Parameter = chrome
IF "%1"=="chrome" goto :get_chrome_pwds

:get_ie_param
:: 2. Parameter = ie
IF "%2"=="ie" goto :get_ie_pwds

:get_ff_param
:: 3. Parameter = ff
IF "%3"=="ff" goto :get_ff_pwds
```

Programmcode 3.1: Auswertung der Übergabeparameter; collect_data.bat (Batch)

Für jeden nicht-leeren Browser-Parameter kommt es zum Aufruf einzelner Programmteile, in denen die benötigten Dateien kopiert werden.
Im Fall von Chrome wird lediglich die Datei „Login Data“ benötigt und über das folgende Kommando nach „C:\PasswordXTract\Files\Chrome“ kopiert.

```
copy /Y "%drive%\Users\%username%\AppData\Local\Google\Chrome\User
    Data\Default\Login Data" "C:\PasswordXTract\Files\Chrome\
   LoginData" > nul
```

Programmcode 3.2: Kopieren der Datei „Login Data“; collect_data.bat (Batch)

Der gleiche Befehl mit angepassten Pfaden findet auch beim Kopieren der Firefox-Dateien „key3.db“, „key4.db“ und „logins.json“ Verwendung.

Für Internet Explorer Passwörter sind die Speicherorte und vor allem die Speicherformen von der jeweiligen Version abhängig. Da in dieser Arbeit nur die Windows-Versionen 8 bis 10 untersucht werden, kann auch der Internet Explorer nur in Version 10 oder höher auf dem Zielsystem installiert sein. Das schließt den Protected Storage und den Storage2 in der Registry als mögliche Speicherorte für IE Passwörter aus.
Weiterhin kommen HTTP-Authentifizierungspasswörter derzeit eher selten zur Anwendung, weshalb deren Extraktion und Entschlüsselung momentan noch nicht in PasswordXTract implementiert sind.

Die aktuellen IE Versionen 10 und 11 speichern ihre Autocomplete-Passwörter ab Windows 8 im Vault-Verzeichnis, wie auch Microsoft Edge. Da es sich hier um einen einfachen Ordner handelt, kann der Windows Vault mittels xcopy kopiert werden:

```
49 xcopy /E /H /Y /Q "%drive%\Users\%username%"\AppData\Local\
       Microsoft\Vault C:\PasswordXTract\Files\IE\Vault\ > nul
```

Programmcode 3.3: Kopieren des Verzeichnisses „Vault“; collect_data.bat (Batch)

Den Abschluss der Datensammlung bildet die Überprüfung, ob auch das Protect-Verzeichnis mit den DPAPI-Masterkeys benötigt wird. Dies ist nur bei der Entschlüsselung von Chrome- oder IE- bzw. Edge-Passwörtern der Fall. Es handelt sich auch hierbei um einen einfachen Ordner, so dass das Kopieren wieder über das xcopy-Kommando erfolgen kann.

3.4.4 Entschlüsselung von Firefox Passwörtern

Wie in Abschnitt 2.3.1 erläutert, benutzt Firefox einen eigenen Algorithmus zur Verschlüsselung der gespeicherten Logindaten, der lückenlos nachvollziehbar ist. Darauf aufbauend kann auch die Entschlüsselung leicht selbst implementiert werden. Die eigentliche Dechiffrierung findet im Ruby-Skript „decrypt_firefox_core.rb“ statt.
Zu Zwecken der Vorverarbeitung wird am Anfang das Batch-Skript „decrypt_firefox.bat“ ausgeführt, wie in Abbildung 3.2 ersichtlich. Darin werden für jedes Profilverzeichnis unter „C:\Users\<username>\AppData\Roaming\Mozilla\Firefox\Profiles“ zunächst alle wichtigen Einträge aus der „key4.db“-Datenbank mittels SQL extrahiert und in einzelne Dateien gespeichert, wie der untenstehende Quellcode zeigt. Zu diesen Einträgen zählen der global salt, der password-check und der verschlüsselte privatekey. Falls die „key4.db“ nicht existiert, werden dennoch die drei Dateien erzeugt, jedoch mit einer Größe von 0 Byte. Gleiches gilt bei der fehlender „key3.db“ oder „logins.json2“, was durch die Zeilen 24 und 26 im Programmcode 3.4 zum Ausdruck kommt.

```
for /f "delims=" %%i in ('dir /B C:\PasswordXTract\Files\FF') do (
  :: falls key4.db existiert...
  IF EXIST "C:\PasswordXTract\Files\FF\%%i\key4.db" (
    :: global salt auslesen
    "C:\PasswordXTract\SQLITE_cmd\sqlite-tools-win32-x86-3240000\
       sqlite3" "C:\PasswordXTract\Files\FF\%%i\key4.db" "select
       item1 from metaData;" ".exit" > "C:\PasswordXTract\Files\FF
       \%%i\globalsalt"
    :: password-check auslesen
    "C:\PasswordXTract\SQLITE_cmd\sqlite-tools-win32-x86-3240000\
       sqlite3" "C:\PasswordXTract\Files\FF\%%i\key4.db" "select
       hex(item2) from metaData;" ".exit" > "C:\PasswordXTract\
       Files\FF\%%i\password_check"
    :: private key auslesen
    "C:\PasswordXTract\SQLITE_cmd\sqlite-tools-win32-x86-3240000\
       sqlite3" "C:\PasswordXTract\Files\FF\%%i\key4.db" "select
       hex(a11) from nssPrivate;" ".exit" > "C:\PasswordXTract\
       Files\FF\%%i\privatekey"
  ) ELSE (
    :: falls key4.db nicht existiert, leere Dateien erzeugen
    copy NUL "C:\PasswordXTract\Files\FF\%%i\globalsalt" > nul
    copy NUL "C:\PasswordXTract\Files\FF\%%i\password_check" > nul
    copy NUL "C:\PasswordXTract\Files\FF\%%i\privatekey" > nul
    copy NUL "C:\PasswordXTract\Files\FF\%%i\key4.db" > nul
  )
  :: falls es key3.db nicht gibt, leere Datei erzeugen
  IF NOT EXIST "C:\PasswordXTract\Files\FF\%%i\key3.db" copy NUL "C
     :\PasswordXTract\Files\FF\%%i\key3.db" > nul
  :: falls es logins.json nicht gibt, leere Datei erzeugen
  IF NOT EXIST "C:\PasswordXTract\Files\FF\%%i\logins.json" copy
     NUL "C:\PasswordXTract\Files\FF\%%i\logins.json" > nul
  for /f %%a in ('more C:\PasswordXTract\temp\passwordCount') do
     set passCount=%%a
  C:\PasswordXTract\Ruby\bin\ruby.exe C:\PasswordXTract\bin\
     decrypt_firefox_core.rb "%%i" %1 %2 !passCount!
)
```

Programmcode 3.4: Kopieren der „key3.db“ und der „key4.db“; decrypt_firefox.bat (Batch)

Erst wenn alle Dateien erzeugt worden sind, wird „decrypt_firefox_core.rb“ aufgerufen. Beim Aufruf werden 3 Parameter übergeben:

- der vollständige Name des Profilverzeichnisses (<ID>.<Nutzername>)
- der Pfad zur Ergebnisdatei „RESULT“
- das Masterpasswort, falls bekannt

Während der Dechiffrierung von Firefox-Passwörtern spielt der global salt eine wichtige Rolle bei der Erzeugung der Schlüssel für password-check und private key. Um den global salt aus der key3.db bzw. der zuvor erzeugten Datei „globalsalt“ zu extrahieren, besitzt „decrypt_firefox_core.rb“ die Methode `get_global_salt`. Deren Rückgabe-

wert enthält den global salt als ASCII-String.
Die zweite wichtige Komponente bei der Schlüsselerzeugung ist der entry salt. Hier wird, wie in Abschnitt 2.3.1 beschrieben, zwischen dem password-check entry salt und dem private key entry salt unterschieden. Bei der key3.db befinden sich diese entry salts in verschiedenen Abschnitten der Datei, den sogenannten Pages. Page 1 enthält alle nötigen Informationen zum password-check, wie eben auch den entry salt, Page 2 übernimmt diese Aufgabe für den private key. In Bezug auf die key4.db entsprechen die erzeugten Dateien „password-check“ und „privatekey“ den Pages 1 und 2.
Die in Programmcode 3.5 dargestellte Methode `get_entry_salt` kann alle 4 entry salts extrahieren. Durch die Parameter `ver` und `page` wird definiert, welcher entry salt ausgelesen werden soll. `ver` kann die Werte 3 oder 4 annehmen und gibt damit an, ob die key3.db oder die key4.db zum Einsatz kommt. `page` enthält die Zahl 1 oder 2, wobei 1 den entry salt des password-checks repräsentiert und 2 den des private keys.

```
76 def get_entry_salt(ver, page)
77   if ver == 3
78     if page == 1
79       # entry salt fuer password-check
80       offset = "0x1" + read_ascii($key3, 4105, 1).ord.to_s(16) +
             read_ascii($key3, 4104, 1).ord.to_s(16)
81       offset = offset.to_i(16)
82       es_size = read_ascii($key3, offset + 1, 1).ord.to_s(16).to_i
             (16)
83       entry_salt = read_ascii($key3, offset + 3, es_size)
84     else
85       # entry salt fuer private key
86       offset = "0x2" + read_ascii($key3, 8197, 1).ord.to_s(16) +
             read_ascii($key3, 8196, 1).ord.to_s(16)
87       offset = offset.to_i(16)
88       es_size = read_ascii($key3, offset + 0x19, 1).ord
89       entry_salt = read_ascii($key3, offset + 0x1A, es_size)
90     end
91   else
92     if ver == 4
93       if page == 1
94         if $password_check.size > 0
95           # entry salt fuer password-check
96           es_size = read_ascii($password_check, 0x14, 1).ord
97           entry_salt = read_ascii($password_check, 0x15, es_size)
98         else
99           puts "ERROR: Unable to extract Firefox passwords...
                 missing key3.db/key4.db"
100          exit(1)
101        end
102      else
103        if $privatekey.size > 0
104          # entry salt fuer private key
105          es_size = read_ascii($privatekey, 0x14, 1).ord
106          entry_salt = read_ascii($privatekey, 0x15, es_size)
```

```
107         else
108           puts "ERROR: Unable to extract Firefox passwords...
                missing key3.db/key4.db"
109           exit(1)
110         end
111       end
112     end
113   end
114   return entry_salt
115 end
```

Programmcode 3.5: Methode get_entry_salt; decrypt_firefox_core.rb (Ruby)

Aus dem global salt und dem richtigen entry salt kann der Schlüssel für den password-check oder private key bestimmt werden. Dies geschieht mithilfe der Methode `get_key`, die ebenfalls mit den Parametern `ver` und `page` aufgerufen wird. Diese haben jedoch keinen direkten Einfluss auf die Methode selbst, sondern werden lediglich methoden-intern an `get_global_salt` und `get_entry_salt` weitergegeben. Die Berechnung des Schlüssels erfolgt über mehrere Hashing-Operationen, wie folgender Quellcode zeigt:

```
124 def get_key(ver, page)
125   global_salt = get_global_salt(ver)
126   entry_salt = get_entry_salt(ver, page)
127   # padded entry salt erzeugen
128   if entry_salt.size < 20
129     times = (20 - entry_salt.size)
130     i = 1
131     while i <= times
132       padded_entry_salt = entry_salt + 0.chr
133       i = i + 1
134     end
135   else
136     padded_entry_salt = entry_salt
137   end
138
139   sha1 = OpenSSL::Digest::SHA1.new
140
141   # SHA1 von global salt und masterpassword falls letzteres gesetzt
        ist
142   hp = sha1.digest(global_salt + $masterpassword)
143   # an eben bestimmten SHA1-Hash entry salt anhaengen und wiederum
        hashen
144   chp = sha1.digest(hp + entry_salt)
145   # HMAC-SHA1 mit chp als key und dem padded entry salt mit
        angehaengtem entry salt als message
146   k1 = OpenSSL::HMAC.digest("SHA1", chp, padded_entry_salt +
        entry_salt)
147   # HMAC-SHA1 mit chp als key und dem padded entry salt als message
148   tk = OpenSSL::HMAC.digest("SHA1", chp, padded_entry_salt)
149   # HMAC-SHA1 mit chp als key und dem eben bestimmten HMAC mit
        angehaengtem entry salt als message
```

```
150   k2 = OpenSSL::HMAC.digest("SHA1", chp, tk + entry_salt)
151   # k2 an k1 anhaengen, um den entgueltigen Schluessel k zu
          erhalten
152   k = k1 + k2
153   return k
154 end
```

Programmcode 3.6: Methode get_key; decrypt_firefox_core.rb (Ruby)

Zurückgegeben wird ein 40 Byte langer Schlüssel. Dabei bilden aber nur die ersten 24 Byte den tatsächlichen 3DES-Key und die letzten 8 Byte stellen den Initialisierungsvektor für den CBC-Modus dar. Die Entschlüsselung wird mit der Methode `decipher` realisiert, der sowohl der Key als auch der IV und die zu entschlüsselnden Daten übergeben werden. Intern verwendet die Methode die vom OpenSSL-Paket bereitgestellten Funktionen:

```
235 def decipher(key, iv, data)
236   des = OpenSSL::Cipher.new('DES-EDE3-CBC')
237   des.decrypt
238   des.padding = 0
239   des.key = key
240   des.iv = iv
241
242   result = des.update(data)
243   result << des.final
244   return result
245 end
```

Programmcode 3.7: Methode decipher; decrypt_firefox_core.rb (Ruby)

Diese Methode ermöglicht, alle 3DES-verschlüsselten Daten zu dechiffrieren. Dazu zählt unter anderem der password-check. Dieser hat die Aufgabe, das Masterpasswort zu überprüfen, wie in Abschnitt 2.3.1 erläutert. Der entsprechende Ablauf ist in der Methode `check_masterkey` implementiert. Auch hier wird `var` als Parameter übergeben, um der Methode mitzuteilen, aus welcher Datei der password-check ausgelesen werden soll. Zurückgegeben wird entweder der ASCII-String „password-check“ oder ein anderer ASCII-String, je nach Wahl des Masterpasswortes.
Für den Fall, dass das Masterpasswort zwar gesetzt, aber nicht von vornherein bekannt ist, wird ein Wörterbuch-Angriff auf Grundlage der Passwortliste „rockyou.txt“ ausgeführt. Diese Datei enthält über 32,5 Millionen Passwörter von Webseiten, sortiert nach der Häufigkeit ihres Vorkommens, wodurch eine hohe Wahrscheinlichkeit besteht, dass sich das gesuchte Masterpasswort in dieser Datei befindet. Um das korrekte Masterpasswort zu ermitteln, wird die rockyou.txt Zeile für Zeile durchlaufen, die jeweils aktuelle Zeile als Masterpasswort gesetzt, anschließend die Methode `check_masterkey` ausgeführt und das Ergebnis auf den „password-check“-String untersucht. Der entsprechende Quellcodeausschnitt für `ver` =3 ist nachfolgend dargestellt:

```
340     if not check_masterkey(3) =~ /password-check/
341       $masterpassword = ""
342       if not check_masterkey(3) =~ /password-check/
343         puts "  > Decrypting masterpassword...this could take a
                while..."
344       end
345     end
346     # rockyou.txt Zeile fuer Zeile durchgehen und nach dem
            richtigen Masterpasswort suchen
347     File.readlines('C:\\PasswordXTract\\rockyou.txt').each do |line
            |
348       # Suche stoppen sobald check_masterkey(3) "password-check"
            zurueckgibt
349       if check_masterkey(3) =~ /password-check/
350         break
351       end
352       $masterpassword = line.force_encoding("ASCII-8BIT")
353       $masterpassword = $masterpassword[0..$masterpassword.size -
            2]
354     end
355     if check_masterkey(3) =~ /password-check/
356       # Ausgabe des korekten Masterpasswortes
357       puts "  > Masterpassword for #{ARGV[0]} is #{$masterpassword}
            "
358       final_key = get_logins_key(3)
359     else
360       # falls rockyou.txt das korrekte Masterpasswort nicht
            enthaelt, koennen die Login Daten nicht entschluesselt
            werden
361       puts "Unable to decrypt masterpassword."
362       exit(1)
363     end
```

Programmcode 3.8: Wörterbuch-Angriff auf Masterpasswort; decrypt_firefox_core.rb (Ruby)

Wenn das gesuchte Passwort nicht in der „rockyou.txt“ enthalten ist, wird die Meldung „Unable to decrypt masterpassword“ ausgegeben und das Ruby-Skript mit exit-Code 1 beendet. In diesem Fall, kann die „rockyou.txt“-Datei mit einer beliebigen anderen Wortliste, deren Speicherung unter dem selben Namen in „C:\PasswordXTract“ erfolgt, ausgetauscht und PasswordXTract erneut gestartet werden.

Ist das richtige Masterpasswort gefunden, kann schließlich auch der private key zur finalen Dechiffrierung der Logindaten entschlüsselt werden. Dies geschieht in der Methode `get_logins_key`. Der übergebene `ver`-Parameter gibt einerseits wie gewohnt an, aus welcher Datei der verschlüsselte private key gelesen werden soll, jedoch weichen die Abläufe der Entschlüsselung von key3.db und key4.db leicht voneinander ab. Während der entschlüsselte private key aus der key4.db direkt dem finalen Schlüssel zur Dechiffrierung der Login Daten entspricht, ist der private key aus der key3.db nach der Entschlüsselung noch doppelt ASN.1 kodiert. Dieser Unterschied ist im Code 3.9 ersichtlich.

```
159 def get_logins_key(ver)
160  key = get_key(ver, 2)
161  if ver == 3
162   # Start-Offset des verschluesselten private key
163   offset = "0x2" + read_ascii($key3, 8197, 1).ord.to_s(16) +
         read_ascii($key3, 8196, 1).ord.to_s(16)
164   offset = offset.to_i(16)
165   es_size = read_ascii($key3, offset + 0x19, 1).ord
166   # Laenge des verschluesselten private key
167   data_size = read_ascii($key3, offset + 0x19 + es_size + 0x05, 1).
         ord
168   # verschluesselter private key
169   data = read_ascii($key3, offset + 0x19 + es_size + 0x06,
         data_size)
170   # padding falls noetig
171   while not data.length % 8 == 0
172    data = data + 0.chr
173   end
174   # entschluesseln
175   key_data = decipher(key[0..23], key[32..39], data)
176   # key_data ist ASN.1 kodiert --> octet string:
177   length1 = key_data[21].ord
178   key_data1 = key_data[22..22 + length1]
179   # key_data1 ist wiederum ASN.1 kodiert --> octet string
180   length2 = key_data1[0x1C].ord
181   dif = length2 - 24
182   # entschluesselter und dekodierter private key
183   final_key = key_data1[0x1D + dif..0x1D + length2 - 1]
184   return final_key
185  else
186   if ver == 4
187    es_size = read_ascii($privatekey, 0x14, 1).ord
188    # Laenge des verschluesselten private key
189    data_size = read_ascii($privatekey, 0x15 + es_size + 4, 1).ord
190    # verschluesselter private key
191    data = read_ascii($privatekey, 0x15 + es_size + 5, data_size)
192    # padding
193    while not data.length % 8 == 0
194     data = data + 0.chr
195    end
196    # entschluesseln
197    key_data = decipher(key[0..23], key[32..39], data)
198    # entschluesselter private key
199    final_key = key_data[0..23]
200    return final_key
201   end
202  end
203 end
```

Programmcode 3.9: Methode get_logins_key; decrypt_firefox_core.rb (Ruby)

Nachdem der endgültige Schlüssel feststeht, müssen nun noch die verschlüsselten und die unverschlüsselten Logindaten aus der „logins.json“ ausgelesen werden. Dazu zählen

- das verschlüsselte Passwort,
- der verschlüsselte Username,
- die Host URL (Adresse der Hauptwebseite) und
- die Action URL (Adresse, von der der Anmeldevorgang ausging).

Jeder Eintrag in der „logins.json“ repräsentiert die Login-Daten des Nutzers für einen bestimmten Web-Service, wie z.B. Facebook. Ein solcher Eintrag besteht aus mehreren Name-Wert-Paaren, die voneinander mit Kommas abgetrennt sind. Der Name wird immer von Anführungsstrichen umgeben, der Wert nur dann, wenn es sich um einen String handelt. Diese Formatierung begünstigt die Anwendung von regulären Ausdrücken, um die Start-Indizes relevanter Werte zu finden. Ab diesen Indizes werden solange Bytes gelesen, bis zu einem Ausführungszeichen, das das Ende des Wertes markiert. Das verschlüsselte Passwort und der Username sind zusätzlich Base64-kodiert.
Die beschriebenen Schritte sind für das verschlüsselte Passwort exemplarisch im nachfolgenden Quellcode ersichtlich.

```
265    while found == true
266       if log =~ /"encryptedPassword":"/
267          $password_count = $password_count + 1
268          # nach String "encryptedPassword" suchen
269          start = (log =~ /"encryptedPassword":"/) + 21
270          pass = ""
271          i = 0
272          # Byte fuer Byte lesen bis zum naechsten "
273          while not pass =~ /"/ and i <= log.size do
274             pass = pass + log[start + i]
275             i = i + 1
276          end
277          pass64 = pass[0..pass.length - 2]
278          # Base64 kodiertes Passwort dekodieren
279          dec64 = Base64.decode64(pass64)
```

Programmcode 3.10: Methode get_data; decrypt_firefox_core.rb (Ruby)

Die Extraktion der drei anderen Werte erfolgt weitestgehend analog. Ein regulärer Ausdruck in Ruby findet immer nur den ersten Treffer und gibt dessen Index zurück, das heißt, es würde auch nur der erste Eintrag in der logins.json berücksichtigt werden. Um auch die anderen Einträge auszuwerten, erfolgt die Speicherung der extrahierten Werte in einem Array. Danach wird der erste Eintrag, der mit dem Schlüsselwort „timesUsed“ endet, gelöscht und der Ablauf beginnt von vorn. Das Sammeln der Login Daten bricht erst ab, wenn der reguläre Ausdruck „encryptedPassword“ keinen Treffer mehr liefert.
Die Werte „encryptedPassword“ und „encryptedUsername“ enthalten neben den verschlüsselten Passwort- bzw. Username-Daten auch den jeweiligen Initialisierungsvektor an Offset 34 bis 41 und die Länge des verschlüsselten Strings an Offset 43. Die

`get_data`-Methode gibt schließlich ein Array mit mehreren Datensätzen zurück, bestehend aus

- IV für die Entschlüsselung des Passwortes,
- verschlüsseltem Passwort,
- IV für die Entschlüsselung des Usernames,
- verschlüsseltem Username,
- Host URL und
- Action URL.

Der letzte Schritt ist schließlich die Entschlüsselung der gesammelten Daten mit den zuvor bestimmten Schlüsseln. Dazu wird aus jedem Datensatz, der von `getData` erzeugt wurde, der IV und das verschlüsselte Passwort bzw. der verschlüsselte Username zusammen mit dem Key aus `get_login_key` an die `decipher`-Methode übergeben. Die so entschlüsselten Passwörter und Benutzernamen werden am Ende gemeinsam mit den beiden URLs in die Ergebnis-Datei eingetragen und gespeichert. Dieser letzte Programmteil ist im folgenden Quellcode dargestellt:

```
426   for ivkey in get_data($logins).each do
427      iv = ivkey[0]
428      data = ivkey[1]
429      username_iv = ivkey[2]
430      username_data = ivkey[3]
431      hostname = ivkey[4]
432      url = ivkey[5]
433      padd = "#{8.chr}#{6.chr}#{1.chr}#{5.chr}#{3.chr}#{7.chr}#{4.
            chr}#{2.chr}"
434      username_dec = decipher(final_key, username_iv, username_data
            ).gsub(/[#{padd}]/, '')
435      password_dec = decipher(final_key, iv, data).gsub(/[#{padd
            }]/, '')
436      File.open(ARGV[1], "a") do |output|
437         output.puts "Website : " + hostname + " ( " + url + " )"
438         output.puts "Username: " + username_dec
439         output.puts "Password: " + password_dec
440         output.puts ""
441      end
442   end
```

Programmcode 3.11: Entschlüsselung der Login Daten aus „logins.json“ und Speicherung in der „RESULT“-Datei; decrypt_firefox_core.rb (Ruby)

3.4.5 Entschlüsselung von Chrome Passwörtern

Bei der Entschlüsselung von Chrome-Passwörtern greift PasswordXTract auf das Tool Mimikatz zurück, das in Abschnitt 3.3 genauer erklärt wurde. Aus dem Mimikatz-Modul

„DPAPI“ werden hierfür die Kommandos `masterkey` und `chrome` verwendet. Letzteres entschlüsselt alle Login Daten innerhalb der „Login Data“-Datei automatisch, ohne für jeden Datensatz ein extra Kommando zu verwenden. Voraussetzung hierfür ist jedoch, dass alle Masterkeys aus dem „Protect“-Verzeichnis entschlüsselt im Cache von Mimikatz liegen. Um dies zu erreichen, sind zunächst die GUIDs aller Masterkeys in der Datei „masterkeys“ zwischenzuspeichern. Anschließend wird mithilfe einer Schleife eine Kommando-Kette erzeugt, die mehrere Kommandos der Form

```
dpapi::masterkey /in:C:\PasswordXTract\Files\DPAPI\Protect\%sid%\
%guid% /sid:%sid% /password:%password%
```

enthält. Die Variable `guid` nimmt dabei der Reihe nach die Werte aus der erzeugten „masterkeys“-Datei an.
Da Batch-Variablen nur eine begrenzte Speicherkapazität haben, wird die so generierte Kommando-Kette in die Datei „command_queue“ ausgelagert. Um alle Kommandos in eine Zeile zu schreiben, findet anstelle des Windows-internen `echo`-Befehls das von Linux bereitgestellte `echo` aus dem GnuWin32-Paket mit der Option `-n` Verwendung. Der beschriebene Vorgang ist in den Zeilen 15 bis 19 im Programmcode 3.12 implementiert.
Die Kommando-Kette wird abschließend noch um die Kommandos

```
dpapi::chrome /in:C:\PasswordXTract\Files\Chrome\Login Data
```

und

```
exit
```

ergänzt, um die Login Daten letztlich zu entschlüsseln und mimikatz zu beenden.

```
22 :: Dateinamen aller Masterkeys in der Datei "masterkeys" speichern
23 for /f "tokens=4" %%i in ('dir /A:S C:\PasswordXTract\Files\DPAPI\
      Protect\%sid%') do echo %%i >> C:\PasswordXTract\temp\masterkeys
24 for /f %%i in ('more C:\PasswordXTract\temp\masterkeys ^| C:\
      PasswordXTract\bin\wc.exe -l') do set /A "lines=%%i-4"
25 :: Mimikatz-Kommando zur Entschluesselung aller Masterkeys erzeugen
       und in Datei "command_queue" speichern
26 for /f %%i in ('more +1 C:\PasswordXTract\temp\masterkeys ^| C:\
      PasswordXTract\bin\head.exe -n %lines%') do "C:\PasswordXTract\
      bin\echo.exe" -n """dpapi::masterkey /in:C:\PasswordXTract\Files
      \DPAPI\Protect\%sid%\%%i /sid:%sid% /password:%1""" " >> C:\
      PasswordXTract\temp\command_queue
27 :: Mimikatz-Kommando zur Entschluesselung von "Login Data" zur "
      command_queue" hinzufuegen
28 "C:\PasswordXTract\bin\echo.exe" -n """dpapi::chrome /in:C:\
      PasswordXTract\Files\Chrome\LoginData /unprotect""" " >> C:\
      PasswordXTract\temp\command_queue
29 "C:\PasswordXTract\bin\echo.exe" -n "exit" >> C:\PasswordXTract\
      temp\command_queue
```

Programmcode 3.12: Entschlüsselung der Logindaten aus der Datenbank „Login Data“ des Chrome Browsers; decrypt_chrome.bat (Batch)

Abschließend erfolgt die Übergabe des Dateiinhalts von „command_queue“ an Mimikatz. Die Ausgabe von Mimikatz wird im Anschluss in die Datei „result_chrome“ geschrieben, wie die nachfolgenden Codezeilen verdeutlichen.

```
31 :: "command_queue" an Mimikatz uebergeben und Ausgabe in der Datei
      "result_chrome" speichern
32 for /f "delims=" %%i in ('more C:\PasswordXTract\temp\command_queue
      ') do %mimikatz% %%i > C:\PasswordXTract\temp\result_chrome
```

Programmcode 3.13: Übergabe der Kommando-Kette an Mimikatz; decrypt_chrome.bat (Batch)

3.4.6 Entschlüsselung von Internet Explorer- bzw. Edge-Passwörtern (Vaults)

Wie in Abschnitt 3.3 erwähnt, bietet Mimikatz ein eigenes Kommando für die Entschlüsselung von Vaults, jedoch muss, anders als bei Chrome, für jede einzelne vcrd-Datei ein eigenes Kommando ausgeführt werden. Analog zur Entschlüsselung der Chrome-Passwörter, wird auch hier eine Kommando-Kette erzeugt, die zunächst alle Kommandos zur Dechiffrierung der Masterkeys enthält. In einer weiteren Schleife werden die Namen der vcrd-Dateien mittels `findstr` gesucht und jeder Name extra als Laufvariable `%%i` in ein Kommando der Form

```
dpapi::vault /cred:C\PasswordXTract\Files\IE\Vault\%guid%\%%i
/policy:C:\PasswordXTract\Files\IE\Vault\%guid%\Policy.vpol
```

eingebettet. Mimikatz kann jedoch maximal 40 Kommandos innerhalb einer Kommando-Kette verarbeiten. Da die Entschlüsselung jedes Masterkeys je ein Kommando benötigt und die Kommando-Kette mit `exit` abgeschlossen wird, können maximal $40 - n_{MK} - 1$ vcrd-Dateien auf einmal entschlüsselt werden, wobei n_{MK} der Anzahl an Masterkeys entspricht. Genügen die 40 Kommandos zur Entschlüsselung aller vcrd-Dateien nicht, werden mehrere Kommando-Ketten erzeugt, die zunächst immer alle Masterkeys und danach die größtmögliche Anzahl an vcrd-Dateien verarbeiten, abgeschlossen durch ein `exit`. Das bedeutet gleichzeitig, dass PasswordXTract maximal 38 Masterkeys verarbeiten kann. Da jeder Masterkey drei Monate gültig ist, wird diese Anzahl frühstens 9,5 Jahre nach der Einrichtung des Betriebssystems erreicht.
Der daraus resultierende Quellcode unterscheidet sich erheblich von dem, der zur Entschlüsselung von Chrome Passwörtern verwendet wurde, wie folgender Ausschnitt verdeutlicht:

```
26 for /f "tokens=4" %%i in ('dir "C:\PasswordXTract\Files\DPAPI\
      Protect" ^| findstr /R /C:S-1-') do set sid=%%i
27
28 :: Dateinamen aller Masterkeys in der Datei "masterkeys" speichern
29 for /f "tokens=4" %%i in ('dir /A:S C:\PasswordXTract\Files\DPAPI\
      Protect\%sid%') do echo %%i >> C:\PasswordXTract\temp\masterkeys
30 :: Anzahl Masterkeys
```

```
for /f %%i in ('more C:\PasswordXTract\temp\masterkeys ^| C:\PasswordXTract\bin\wc.exe -l') do set /A "lines=%%i-4"
:: GUID bestimmen
for /f "tokens=4" %%i in ('dir C:\PasswordXTract\Files\IE\Vault ^| findstr /R /C:........-....-....-....-............') do set guid=%%i
:: Anzahl vcrd
for /f %%i in ('dir C:\PasswordXTract\Files\IE\Vault\%guid% ^| findstr /R /C:.....................................vcrd ^| C:\PasswordXTract\bin\wc.exe -l') do set /a "max=%%i"
:: Mimikatz-Kommandos zur Entschluesselung aller Masterkeys erzeugen und in Datei "command_queue" speichern
for /f %%i in ('more +1 C:\PasswordXTract\temp\masterkeys ^| C:\PasswordXTract\bin\head.exe -n %lines%') do "C:\PasswordXTract\bin\echo.exe" -n """dpapi::masterkey /in:C:\PasswordXTract\Files\DPAPI\Protect\%sid%\%%i /sid:%sid% /password:%1""" " >> C:\PasswordXTract\temp\command_queue
:: Dateinamen aller vcrd-Dateien in der Datei "vcrdFiles" speichern
for /f "tokens=4" %%i in ('dir C:\PasswordXTract\Files\IE\Vault\%guid% ^| findstr /R /C:.....................................vcrd') do echo %%i >> C:\PasswordXTract\temp\vcrdFiles
:: maximale Anzahl von Kommandos zur Entschluesselung von vcrd-Dateien innerhalb einer Kommando-Kette
set /A "mimi_max=39-%lines%"
:vcrd_to_command_queue
set /A "vcrd_count=0"
for /f %%i in ('more C:\PasswordXTract\temp\vcrdFiles') do (
   :: solange die maximale Anzahl an vcrd-Kommandos nicht ueberschritten ist: Mimikatz-Kommandos zur Entschluesselung aller vcrd-Dateien erzeugen und an "command_queue" anhaengen
   IF !vcrd_count! LSS %mimi_max% (
      "C:\PasswordXTract\bin\echo.exe" -n """dpapi::vault /cred:C:\PasswordXTract\Files\IE\Vault\%guid%\%%i /policy:C:\PasswordXTract\Files\IE\Vault\%guid%\Policy.vpol""" " >> C:\PasswordXTract\temp\command_queue
      set /A "password_count+=1"
      set /A "vcrd_count+=1"
      :: Abbruch sobald alle vcrd-Dateien verarbeitet wurden
      IF !password_count! GEQ %max% goto exit_mimi:
   ) ELSE (
      :exit_mimi
      :: die ersten mimi_max Namen von vcrd-Dateien aus der Datei vcrdFiles loeschen
      more +%mimi_max% C:\PasswordXTract\temp\vcrdFiles > C:\PasswordXTract\temp\vcrdFiles_temp
      copy C:\PasswordXTract\temp\vcrdFiles_temp C:\PasswordXTract\temp\vcrdFiles > nul
      :: "exit"-Kommando zur Kommando-Kette hinzufuegen
      echo "exit" >> C:\PasswordXTract\temp\command_queue
      :: "command_queue" an Mimikatz uebergeben und Ausgabe in der Datei "result_vault" speichern
```

```
60      for /f "delims=" %%a in ('more C:\PasswordXTract\temp\
            command_queue') do %mimikatz% %%a >> C:\PasswordXTract\
            temp\result_vault
61      :: alte Kommando-Kette loeschen
62      del C:\PasswordXTract\temp\command_queue
63      :: erneut die Masterkeys zur Kommando-Kette hinzufuegen
64      for /f %%m in ('more +1 C:\PasswordXTract\temp\masterkeys ^|
            C:\PasswordXTract\bin\head.exe -n %lines%') do "C:\
            PasswordXTract\bin\echo.exe" -n """dpapi::masterkey /in:C
            :\PasswordXTract\Files\DPAPI\Protect\%sid%\%%m /sid:%sid%
            /password:%1""" " >> C:\PasswordXTract\temp\command_queue
65      :: weiter mit der Entschluesselung der naechsten mimi_max
            vcrd-Dateien aus der vcrdFiles Datei
66      goto vcrd_to_command_queue:
67  )
```

Programmcode 3.14: Erstellen der Kommando-Kette(n) zur Entschlüsselung der vcrd-Dateien im Vault-Verzeichnis; decrypt_vault.bat (Batch)

3.4.7 Aufbereitung der Ergebnisse

Ziel ist eine Liste, die für jeden gespeicherten Login die Angaben Host bzw. Action URL, Benutzername und Passwort enthält. Während diese Formatierung für Firefox direkt im Skript „decrypt_firefox_core.rb“ erfolgt, müssen die Logindaten für Chrome und IE bzw. Edge (Vault) erst aus den Dateien „result_chrome“ und „result_vault“, die bei der Entschlüsselung von Chrome bzw. Vaults angelegt wurden, herausgefiltert werden. Diese beiden Dateien enthalten die vollständigen Ausgaben von Mimikatz, die aus der Ausführung der jeweiligen Kommando-Kette(n) hervorgegangen sind. Die Abbildungen 3.4 und 3.5 stellen die entscheidenden Abschnitte der beiden Dateien dar.

```
URL      : https://www.gmx.net/ ( https://www.gmx.net/ )
Username: praktikum1234@gmx.de
 * using CryptUnprotectData API
 * volatile cache: GUID:{b4eafef2-842d-4bb6-b5bc-6bd64e0e...
Password: TollesPasswort
```

Abbildung 3.4: Ausschnitt aus der Datei „result_chrome“ (eigene Quelle)

```
 * identity      : praktikum1234@gmx.de
 * ressource     : https://www.gmx.net/
 * authenticator : TollesPasswort
 * property 100  : d5 b6 3c 4e 56 25 d8 4c a4 8d c7 55 c7 37 cb a6
```

Abbildung 3.5: Ausschnitt aus der Datei „result_vault“ (eigene Quelle)

Um ein einheitliches Erscheinungsbild zu erhalten, werden die relevanten Informationen mittels regulärer Ausdrücke aus den beiden Dateien extrahiert und ebenso in die Datei „RESULT“ geschrieben, wie bereits die entschlüsselten Firefox-Logins.
Der Inhalt dieser Datei wird im Anschluss an die abgeschlossenen Dechiffrierungen im Terminal angezeigt.

3.4.8 Programmabschluss

Der Benutzer erhält am Ende die Aufforderung, einen Speicherort für die Ergebnisse anzugeben. Falls keine Eingabe erfolgt, werden die Login Daten nicht explizit gespeichert, eine Einsicht ist aber weiterhin unter C:\PasswordXTract\RESULT möglich.
Anschließend kann das als Laufwerk gemountete Image wieder entfernt werden, was aber nicht die Löschung des Images selbst bewirkt, sondern lediglich die Kopplung an den Laufwerksbuchstaben auflöst.
Außerdem besteht die Möglichkeit, alle Daten aus dem „temp“- und dem „Files“-Ordner, zusammen mit der Datei „RESULT“, zu löschen.
Weiterhin kann der Nutzer noch entscheiden, ob alle installierten Programme wieder deinstalliert werden sollen, um Speicherplatz freizugeben.

4 Ergebnisse

Zunächst ist festzuhalten, dass es im Rahmen dieser Arbeit gelungen ist, ein Programm zu entwickeln, das Browserpasswörter auf nutzerfreundliche und nachvollziehbare Weise aus einer forensischen Datensicherung extrahiert und entschlüsselt. In den weiteren Ausführungen, soll PasswordXTract besonders in Bezug auf Entschlüsselungsergebnisse und Laufzeit untersucht werden.

4.1 Ausgabe von PasswordXTract

Die entschlüsselten Passwörter, Usernamen und URLs werden nach der Dechiffrierung direkt im Terminal angezeigt. Außerdem ist das Ergebnis in der Datei „RESULT“ unter C:\PasswordXTract abgespeichert, die jedoch überschrieben wird, sobald ein Neustart von PasswordXTract erfolgt. Eine Kopie der Datei liegt an einem durch den Nutzer definierten Speicherort, sofern dieser zum Programmabschluss benannt wurde. In allen Fällen werden die Resultate in folgendem Format gespeichert bzw. angezeigt:

```
__________Mozilla Firefox:__________

Website : https://www.facebook.com ( https://www.facebook.com )
Username: firefox@mail.de
Password: firefoxFBPasswort

Website : https://www.facebook.com ( https://www.facebook.com )
Username: test@firefox.de
Password: keykeykey

Website : https://www.web.de ( https://www.web.de )
Username: smile@me.de
Password: hihi

__________Google Chrome:__________

Website : https://www.gmx.net/ ( https://www.gmx.net/ )
Username: praktikum1234@gmx.de
Password: TollesPasswort

__________Internet Explorer und Edge (Vault):__________

Website : https://www.gmx.net/
Username: praktikum1234@gmx.de
Password: TollesPasswort

Website : https://www.gmx.net/
Username: test98765@gmx.de
Password: geheim
```

Abbildung 4.1: beispielhafter Inhalt der „RESULT“-Datei (eigene Quelle)

4.2 Laufzeit

Um zu messen, wie viel Zeit PasswordXTract für die Entschlüsselung benötigt, wird programmatisch die Differenz zwischen Start- und Endzeitpunkt bestimmt. Bei Chrome und IE bzw. Edge beeinflussen mit der Anzahl der Masterkeys und der Anzahl der zu entschlüsselnden Passwörter zwei Faktoren die Laufzeit, während bei Firefox entscheidend ist, ob das Masterpasswort vorliegt oder erst noch entschlüsselt werden muss. Im letzteren Fall kann dies unter Umständen sehr viel Zeit in Anspruch nehmen.
Natürlich ist die Laufzeit von Computer zu Computer verschieden. Die unter 4.2.1 und 4.2.2 beschriebenen Messungen wurden an einem HP Split 13 x2[5] mit Windows 10 durchgeführt. Die genauen Messreihen sind im Anhang B ersichtlich.

4.2.1 Zusammenhang zwischen der Anzahl an Masterkeys und der Laufzeit

Um die gewünschte Anzahl an Masterkeys zu erzeugen, wurde ein einzelner Masterkey kopiert, mit einer neuen GUID benannt und im „Protect“-Verzeichnis gespeichert. Für die Messung wurde PasswordXTract für jede Zahl an Masterkeys dreimal gestartet und der Durchschnitt der ermittelten Laufzeiten bestimmt. Bei der Browserauswahl wurde „Google Chrome“ gewählt und in der „Login Data“ befand sich stets nur ein Eintrag. Die Ergebnisse sind im folgenden Diagramm dargestellt:

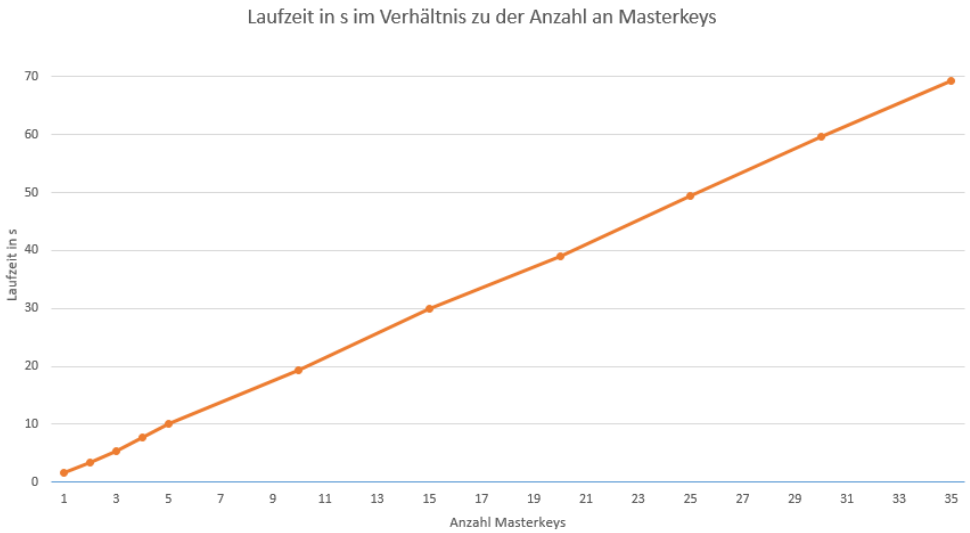


Abbildung 4.2: Abhängigkeit der Laufzeit von der Anzahl an Masterkeys (eigene Quelle)

Es besteht ein linearer Zusammenhang, der für das vorliegende Testsytem (HP Split 13 x2) annähernd durch die Formel $y(x) = 2x$ beschrieben werden kann.

[5] Prozessor: Intel(R) Core(TM) i5-4200Y CPU @ 1,40 GHz, 2 Kerne, 4 logische Prozessoren; Arbeitsspeicher: 4 GB

4.2.2 Zusammenhang zwischen der Anzahl der zu entschlüsselnden Passworte und der Laufzeit

Für die Messungen bei **Google Chrome** befanden sich stets zwei Masterkeys im „Protect“-Verzeichnis. Um genügend Einträge in der „Login Data“-Datenbank zu erzeugen, wurde eine vorhandene Datenreihe mehrmals dupliziert. Auch hier erfolgten jeweils drei Zeitmessungen, um eine gegebene Anzahl an Einträgen zu entschlüsseln. Mit den durchschnittlichen Laufzeiten wurde folgendes Diagramm erstellt:

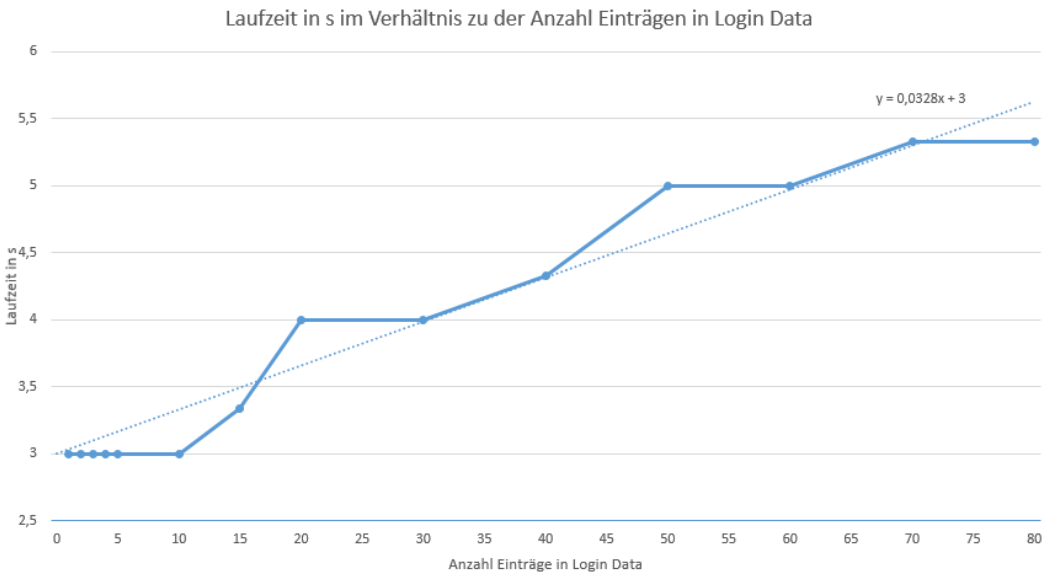


Abbildung 4.3: Abhängigkeit der Laufzeit von der Anzahl der Datensätze in der Datenbank „Login Data“ (eigene Quelle)

Auch hier ist ein linearer Zusammenhang erkennbar, so dass sich für den vorliegenden HP-Laptop die lineare Gleichung $y(x) = 0,0328x + 3$ ergibt. Damit ist der Anstieg um ein Vielfaches kleiner als im Diagramm 4.2, was bedeutet, dass die Anzahl der zu entschlüsselnden Passwörter einen deutlich kleineren Einfluss auf die Laufzeit hat als die Zahl der Masterkeys.

Ein ähnliches Ergebnis ergab die Messung für **Internet Explorer bzw. Edge (Vault)**. Um eine Vielzahl an vcrd-Dateien zu erzeugen, wurde eine bereits vorhandene Datei dupliziert und wie schon bei den Masterkeys unter einem neuen Namen abgespeichert. Für die Messungen befanden sich im „Protect“-Verzeichnis ebenfalls zwei Masterkeys. Das daraus resultierende Diagramm unterscheidet sich jedoch von dem vorangegangenen:

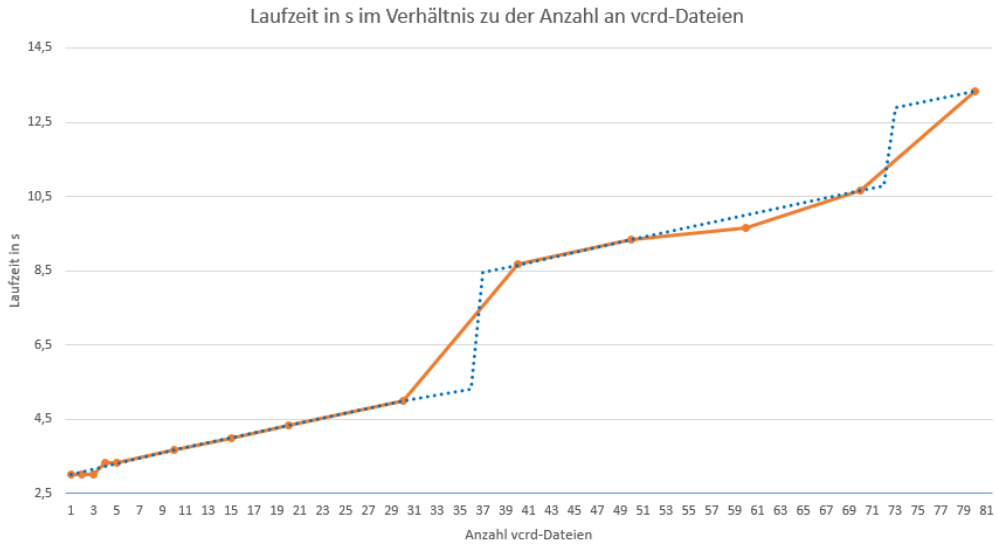


Abbildung 4.4: Abhängigkeit der Laufzeit von der Anzahl der vcrd-Dateien (eigene Quelle)

Während zu Beginn ein gleichmäßiger linearer Anstieg zu verzeichnen ist, kommt es zwischen x=30 und x=40 zu einem Sprung. Dieses Muster wiederholt sich anschließend ein zweites mal mit Sprung zwischen x=70 und x=80.
Dieses Verhalten leitet sich direkt aus dem Programmablauf ab. Da Mimikatz nur 40 Kommandos pro Aufruf übergeben werden können, erfolgt die Entschlüsselung bei sehr vielen Passwörtern stückweise.
In diesem Testfall wurden 2 Masterkeys entschlüsselt, das heißt es können 37 vcrd-Dateien pro Aufruf entschlüsselt werden (37+2=39, hinzu kommt das abschließende „exit“ Kommando zum Verlassen von Mimikatz). Nachdem die 37 Passwörter entschlüsselt sind, müssen einige Zwischenschritte durchlaufen werden, wie in Abschnitt 3.4.6 beschrieben. Diese Zwischenschritte nehmen mehr Zeit in Anspruch als die Entschlüsselung selbst, wodurch die beobachteten Sprünge entstehen.
In welchem Abstand diese Sprünge auftreten, lässt sich anhand folgender Formel bestimmen: $\Delta x = 40 - n_{MK} - 1$ wobei n_{MK} der Anzahl der Masterkeys entspricht. Bei diesem Beispiel treten die Sprünge deshalb zwischen x=37 und x=38 sowie zwischen x=74 und x=75 auf.
Die Entschlüsselung selbst verläuft wie schon bei Chrome linear mit einem ähnlich geringen Anstieg.

Die Entschlüsselung von **Mozilla Firefox** unterscheidet sich von Chrome, IE oder Edge und besitzt deshalb auch in Bezug auf die Laufzeit eine besondere Charakteristik. Für die Messungen wurde das Masterpasswort als gegeben angenommen. Die Einträge in der „logins.json“ sind nach Bedarf dupliziert worden, um die benötigte Anzahl an Datensätzen zu erzeugen.

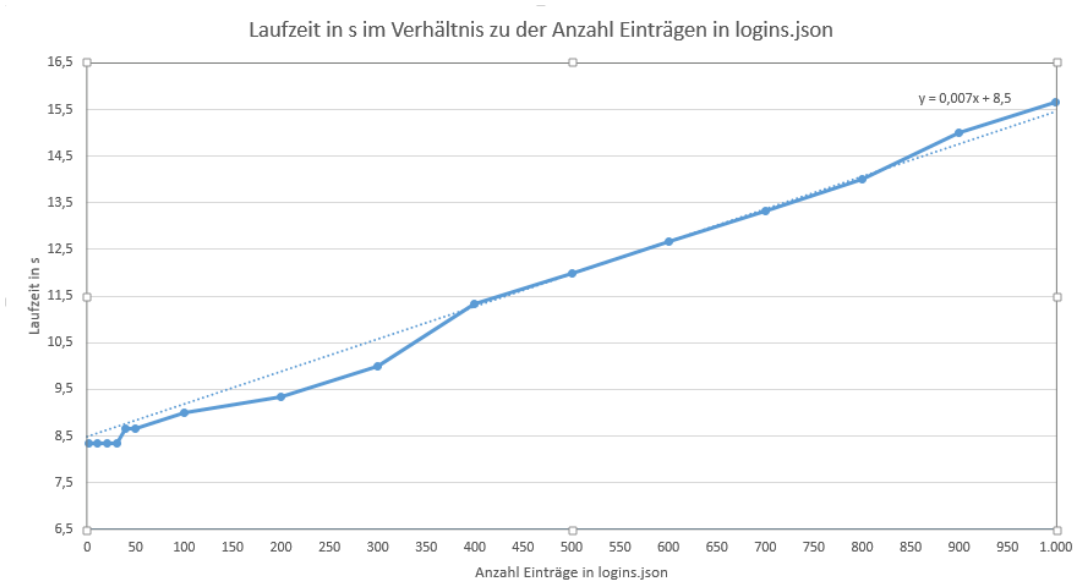


Abbildung 4.5: Abhängigkeit der Laufzeit von der Anzahl der Einträge in „logins.json“ (eigene Quelle)

Wie man im Diagramm 4.5 erkennen kann, zeichnet sich die Laufzeit durch eine hohe Konstanz aus. Selbst bei einer Anzahl von 1000 Passwörtern benötigt PasswordXTract auf dem Testgerät nur knapp 16 Sekunden, um alle zu entschlüsseln. Der Zusammenhang kann dennoch als linear bezeichnet werden, wobei der Anstieg entsprechend geringer ist als bei Chrome oder IE bzw. Edge.
Die Laufzeit verlängert sich jedoch, wenn neben dem „<ID>.default“-Verzeichnis noch weitere Ordner unter „C:\Users\<username>\AppData\Roaming\Mozilla\Firefox\Profiles“ zu finden sind. In diesem Fall werden die Laufzeiten der einzelnen Profile addiert. Bestehen unter dem Profil „<ID>.default“ beispielsweise 10 verschlüsselte Passwörter und unter dem Profil „<ID>.Nutzer1“ 100 Passwörter beträgt die Laufzeit auf dem Testrechner circa $9s + 8,333s = 17,333s$. Die 9s entsprechen der aus dem Diagramm entnommenen Laufzeit für 100 Passwörter und die 8,333s wurden für 10 Passwörter aus dem Diagramm abgelesen. Grundsätzlich gilt also: Je mehr Profile, desto höher die Laufzeit.

Allgemein verhalten sich alle Laufzeiten linear, so dass für die Entschlüsselung ein Aufwand von $O(n)$ angenommen werden kann.

5 Diskussion

5.1 Zusammenfassung

Diese Arbeit behandelte einige fundamentale Verfahren der Cryptographie sowie die Abläufe innerhalb der Windows-eigenen Data Protection API. Auf dieser Grundlage konnte für jeden der vier meist verbreiteten Browser aufgezeigt werden, wie die Verschlüsselung und Speicherung der Passwörter erfolgt.
Es wurde bereits existierende Software untersucht und kombiniert, um schließlich ein neues Programm zu entwerfen, das in der Lage ist, die Passwörter aller analysierten Browser zu extrahieren und zu entschlüsseln.

PasswordXTract ist das Ergebnis dieser Recherche- und Programmierarbeiten. Es vereint die Passwort-Entschlüsselung aller genannten Browser in einer Anwendung und erfüllt diese Aufgabe zudem in einer vertretbaren Zeitspanne.

5.2 Ergebnisevaluation und mögliche Fehlerquellen

Ziel dieser Arbeit war eine Extraktion und Entschlüsselung von Browser-Passwörtern per Knopfdruck auf der Basis eines Images. Auch wenn PasswordXTract maximal sechs Knopfdrücke in Form von Benutzereingaben benötigt, wird das Programm diesen Anforderungen gerecht. Nachdem alle benötigten Angaben von PasswordXTract erfragt wurden, läuft die Extraktion und Entschlüsselung vollkommen automatisch ab. Für die Entschlüsselungen selbst finden ausschließlich quelltextoffene Skripte Verwendung, so dass alles bei Bedarf leicht nachvollzogen werden kann. Da PasswordXTract selbst das Mounten des Images übernimmt, ist auch die Möglichkeit der Post-Mortem-Analyse gegeben bzw. bindend.
Obwohl PasswordXTract derzeit ausschließlich über die Kommandozeile arbeitet, gestaltet sich die Bedienung trotzdem recht intuitiv. Die benötigten Angaben werden Schritt für Schritt vom Benutzer erfragt, wobei es sich lediglich um einzelne Wörter oder Zahlen handelt. Auf einen komplizierten Aufruf mit Kommandozeilenparametern und einer festen Syntax wird somit verzichtet.
Dennoch gibt es bei der Arbeit mit PasswordXTract einige Dinge zu beachten:

Bindung an Windows. Das in PasswordXTract genutzte Programm „Mimikatz“ ist explizit für die Nutzung unter Windows konzipiert, wodurch auch PasswordXTract selbst derzeit nur auf Windows-Rechnern arbeitet.

Administratorrechte. PasswordXTract benötigt für eine korrekte Funktionalität Administratorrechte. Um dies zu erreichen, wird „cmd.exe“ mit „Rechtsklick > Als Ad-

ministrator ausführen“ gestartet und PasswordXTract über die sich öffnende Konsole aufgerufen.

Speicherort. PasswordXTract muss direkt unter „C:\PasswordXTract“ entpackt werden. Dadurch ist garantiert, dass die Pfade zu den Dateien keine Leerzeichen enthalten, um eine fehlerfreie Interpretation durch Mimikatz zu gewährleisten.

Virenscanner abschalten. Viele Anti-Viren-Programme stufen Mimikatz als Malware ein, obwohl die Verwendung unbedenklich ist. Es empfiehlt sich daher immer den Virenscanner solange zu deaktivieren, bis PasswordXTract beendet wird. Sind nur Firefox Passwörter zu entschlüsseln, muss keine Deaktivierung erfolgen, da Mimikatz hier nicht zum Einsatz kommt. Sollte zur Laufzeit ein Fehler auftreten, der nicht in Tabelle 5.1 benannt ist, empfiehlt es sich, zu überprüfen, ob „mimikatz.exe“ sich noch unter „C:\PasswordXTract\Mimikatz\x64“ befindet und auch nicht durch Anti-Virus-Software blockiert wird.

Weitere Fehler, die zur Laufzeit auftreten können, und deren Ursache sind in folgender Tabelle zusammengefasst:

Fehler/Warnung	**Ursache**
`Image not found! Insert a valid path!`	Der durch den Benutzer eingegebene Pfad zum Image ist ungültig.
`Choose a currently unused drive!`	Der angegebene Laufwerksbuchstabe kann nicht als Mountpoint verwendet werden, da dieses Laufwerk schon existiert.
`User named ... does not exist!`	Der angegebene Benutzername wird im Image nicht verwendet.
`Please enter valid answer y or n!`	Als Eingabe sind ausschließlich „y“ für „ja“ oder „n“ für „nein“ möglich.
`Please install Java jdk first!`	Auf dem Auswertesystem ist Java nicht im Standard-Verzeichnis unter „C:\Program Files\Java“ installiert.
`Unable to extract Firefox passwords...missing key3.db/key4.db`	Weder key3.db noch key4.db ist vorhanden.
`Unable to decrypt masterpassword.`	Das Firefox-Masterpasswort ist nicht in „rockyou.txt“ enthalten.
`Unable to extract Firefox passwords...missing logins.json`	Die „logins.json“ ist nicht vorhanden.

Tabelle 5.1: Mögliche Fehler und deren Ursachen

5.3 Vergleich mit ähnlichen Programmen

PasswordXTract ist mit folgenden Programmen vergleichbar:

- WebBrowserPassView von Nirsoft
- OSForensics von PassMark Software
- Browser Password Decryptor bzw. Browser Password Recovery Pro von XenArmor

5.3.1 Beschreibung der Testdaten als Grundlage für den Vergleich

Um die genannten Programme vergleichend mit PasswordXTract zu testen, wurde eine virtuelle Festplatte erstellt, als Laufwerk A: eingebunden und der nachfolgend abgebildete Verzeichnisbaum darauf angelegt. Dieser enthält ausschließlich diejenigen Dateien und Ordner, die für die Entschlüsselung aller Browserpasswörter nötig sind.

```
A:
└── Users\
    └── Testnutzer\
        └── AppData\
            ├── Local\
            │   ├── Google\
            │   │   └── Chrome\
            │   │       └── User Data\
            │   │           └── Default\
            │   │               └── Login Data
            │   ├── Microsoft\
            │   │   └── Vault\
            │   │       └── 4BF4C442-9B8A-41A0-B380-DD4A704DDB28\
            │   │           ├── 3CCD5499-87A8-4B10-A215-608888DD3B55.vsch
            │   │           ├── 56EEC2B103C16D10FDEC794F9B36D39362D33BC8.vcrd
            │   │           ├── 4FC04C8515F808342EA26073D33257A5C0F31CE6.vcrd
            │   │           └── Policy.vpol
            │   └── Roaming\
            │       ├── Microsoft\
            │       │   └── Protect\
            │       │       └── S-1-5-21-2555599271-1124747379-1021516555-1001\
            │       │           ├── 1c251876-872d-4e69-9028-4d9c6d0f78f7
            │       │           ├── 298383bf-cf79-4e05-a8d3-0d23363829e6
            │       │           ├── ...
            │       │           ├── b4eafef2-842d-4bb6-b5bc-6bd64e0eb465
            │       │           └── Preferred
            │       └── Mozilla\
            │           └── Firefox\
            │               └── Profiles\
            │                   ├── P8OZ1GTC.default\
            │                   │   ├── logins.json
            │                   │   ├── key3.db
            │                   │   └── key4.db
            │                   └── P8OZ1GTC.default1\
            │                       ├── logins.json
            │                       ├── key3.db
            │                       └── key4.db
```

Im Verzeichnis „Protect“ liegen insgesamt 16 verschiedene Masterkeys. Für Chrome befinden sich drei gespeicherte Passwörter in der Datenbank „Login Data“ und im Vault-Verzeichnis sind zwei vcrd-Dateien enthalten, wobei eine durch IE und eine durch Edge erzeugt wurde. Unter Firefox finden sich zwei Profilverzeichnisse, P80Z1GTC.default und P80Z1GTC.default1. In beiden Ordnern verfügt die logins.json über drei Passwörter, wobei diese beim zuerst genannten Profilordner zusätzlich durch ein Masterpasswort geschützt werden.

Die nach diesem Schema arrangierte virtuelle Festplatte wurde anschließend im vhd-Format gespeichert, so dass sie im weiteren Verlauf als Image mittels OSFMount gemountet werden kann. Während PasswordXTract den Mount-Vorgang selbstständig durchführt, muss das Einbinden des Images als Laufwerk für die anderen Programme manuell mit Hilfe von OSFMount erfolgen. So kann die Leistung aller aufgeführten Applikationen schließlich unter den gleichen Bedingungen getestet und evaluiert werden.

5.3.2 WebBrowserPassView

WebBrowserPassView von Nirsoft kann sowohl über eine grafische Oberfläche als auch über die Kommandozeile bedient werden. Es ist in der Lage, Browserpasswörter am laufenden System zu entschlüsseln. Zu den unterstützten Browsern zählen Internet Explorer, Chrome, Firefox, Safari, Opera, SeaMonkey und Yandex. Für Firefox und Chrome besteht unter den erweiterten Einstellungen die Möglichkeit, den Pfad zum „Profile“-Verzeichnis von Firefox bzw. zum „User Data“-Verzeichnis von Chrome anzugeben, um auch hier ein externes Dateisystem auswerten zu können:

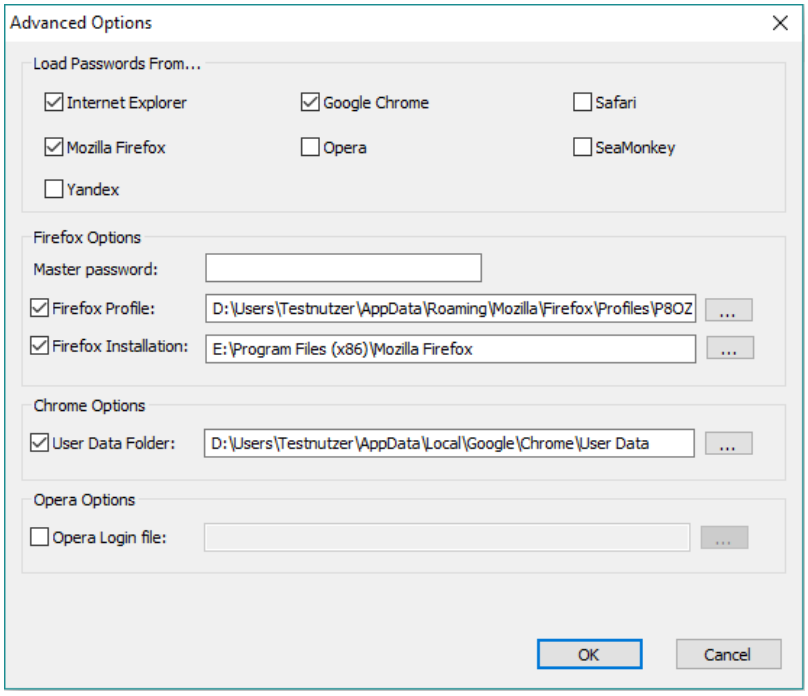


Abbildung 5.1: Erweiterte Optionen von WebbrowserPassView (eigene Quelle)

Leider unterstützt WebBrowserPassView für Chrome nach eigenen Angaben keine von IE 7.0 - 9.0 importierten Passwörter. Auch für Firefox gilt die Beschränkung, dass nur die Entschlüsselung der Passwörter erfolgen kann, die mit einer 32bit-Version von Firefox gespeichert wurden [Nir17]. Weiterhin ist es offenbar nicht möglich Internet Explorer-Passwörter von einem externen System zu entschlüsseln, da hierfür keine alternativen Pfade angegeben werden können.

Beim Test mit dem unter Abschnitt 5.3.1 beschriebenen Image sind zunächst in den erweiterten Optionen die erforderlichen Pfade für Chrome und Firefox sowie das Masterpasswort eingetragen worden. Als Ergebnis führt WebBrowserPassView die drei Einträge von Chrome und eine Vielzahl an IE-Datensätzen auf, wie in Abbildung 5.2 dargestellt. Bei genauerer Betrachtung handelt es sich bei letzteren ausschließlich um die gespeicherten Passwörter des laufenden Systems. Dies war zu erwarten, da für den Internet Explorer kein optionaler Pfad angegeben werden konnte.
Die Chrome Daten wurden zwar korrekt extrahiert, die Passwörter jedoch nicht entschlüsselt. Eine mögliche Begründung hierfür ist, dass WebBrowserPassView keine Eingabe eines Benutzerpasswortes für das fremde Nutzerprofil zulässt, was aber die Voraussetzung für eine erfolgreiche Entschlüsselung darstellt.
Firefox-Passwörter wurden weder gefunden noch entschlüsselt. Begründet wird dies damit, dass die Erzeugung der Passwörter mit einer 64bit-Version von Firefox erfolgt ist, die durch WebBrowserPassView nicht unterstützt wird.

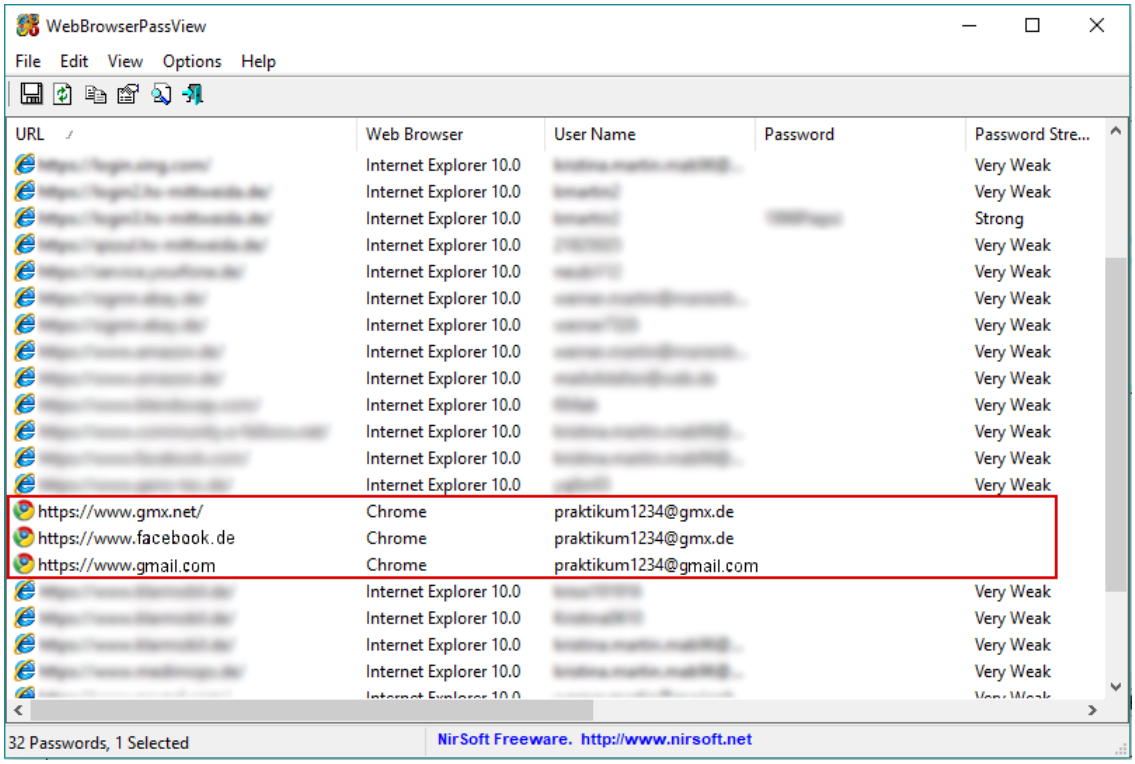


Abbildung 5.2: Ergebnis von WebbrowserPassView für die oben dargestellten Einstellungen (eigene Quelle)

Neben diesem All-in-one-Tool bietet Nirsoft für Firefox, Chrome und IE bzw. Edge auch separate Werkzeuge an.

ChromePass arbeitet am Live-System ohne Probleme und zeigt auch die Webseiten an, für die das Passwort nie gespeichert werden soll. Diese erkennt man durch ein leeres Feld in der „Password“-Spalte. In den erweiterten Optionen kann ein alternatives Nutzerprofil sowie ein Benutzerpasswort eingetragen werden.
Beim Test mit dem vorbereiteten Image ließen sich auf diese Art und Weise alle drei Chrome-Passwörter entschlüsseln, wie in Abbildung 5.3 zu sehen.

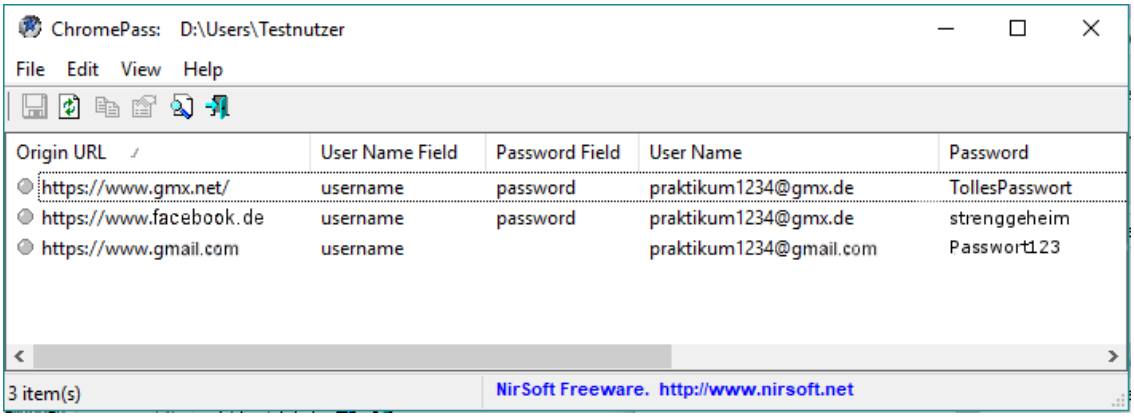


Abbildung 5.3: Ergebnis von ChromePass (eigene Quelle)

IE Pass View soll laut Nirsoft den Internet Explorer in den Versionen 4.0 bis 11.0 und Edge unterstützen [Nir16]. Wie schon bei Chrome Pass bietet auch IE Pass View in den erweiterten Optionen die Möglichkeit, einen Pfad zu einem externen Nutzerprofil anzugeben. Dadurch konnten beide Passwörter aus dem Vault-Verzeichnis des Testlaufwerks extrahiert und erfolgreich entschlüsselt werden, wie Abbildung 5.4 zeigt.

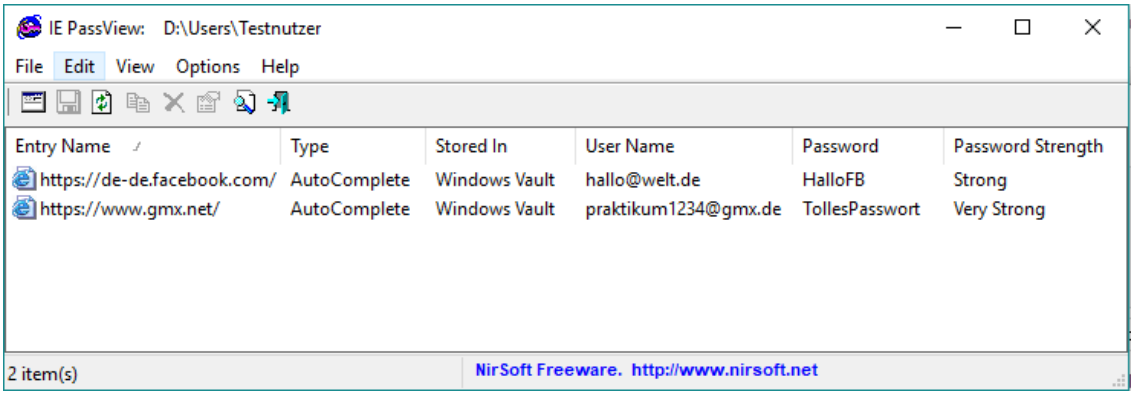


Abbildung 5.4: Ergebnis von IE Pass View (eigene Quelle)

PasswordFox steht sowohl in einer 32bit- als auch in einer 64bit-Version zur Verfügung. Welche Version verwendet werden muss, hängt von der installierten Firefox-Version ab. In beiden Fällen bedarf es der Angabe eines Nutzerverzeichnisses und des Installationsordners von Firefox. Falls ein Masterpasswort gesetzt ist, muss dieses explizit angegeben werden, um eine erfolgreiche Entschlüsselung zu ermöglichen.
Weiterhin kann auch nur ein einzelnes Nutzerprofil aus dem „Profiles“-Ordner verarbeitet werden.

Beim Test wurde die 64bit-Version von PasswordFox genutzt und zunächst der Pfad des ersten Profilordners zusammen mit dem korrekten Masterpasswort unter „File > Select Folders“ angegeben. Auf diese Weise konnten alle drei Passwörter erfolgreich entschlüsselt werden. Nach dem gleichen Vorgehen war es auch möglich, die Login-Daten aus dem zweiten Profil zu entschlüsseln, ohne ein Masterpasswort eingeben zu müssen. Beide Ergebnisse sind in den Abbildungen 5.5 und 5.6 dargestellt.

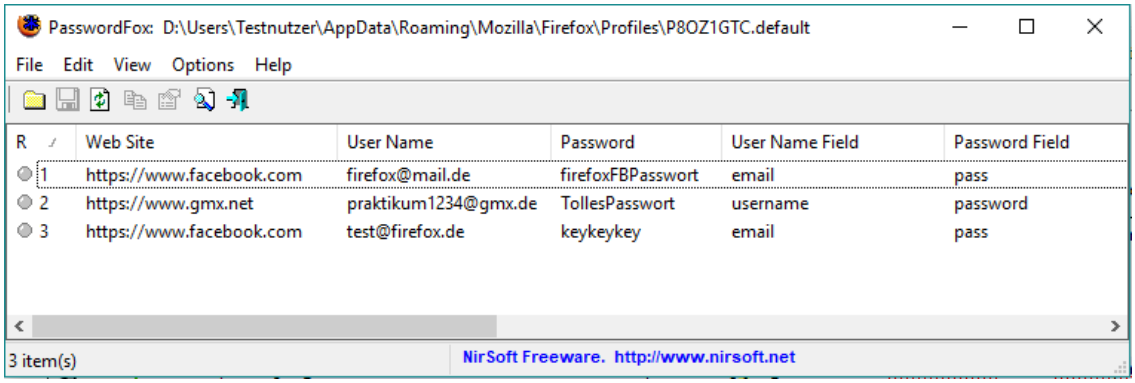


Abbildung 5.5: Ergebnis von PasswordFox mit Masterpasswort (eigene Quelle)

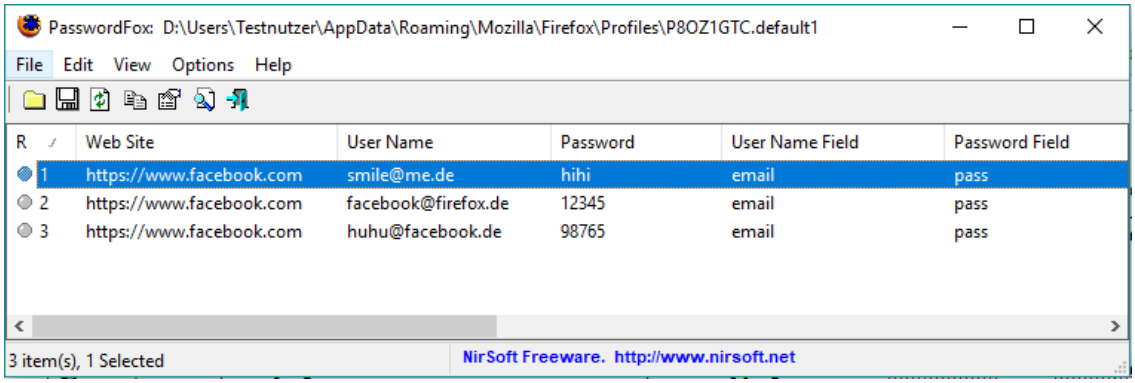


Abbildung 5.6: Ergebnis von PasswordFox ohne Masterpasswort (eigene Quelle)

5.3.3 OSForensics

OSForensics von PassMark Software ist ein sehr umfangreiches Tool. Die Entschlüsselung von Passwörtern nimmt nur einen kleinen Teil des Programmes ein und umfasst neben Firefox, Chrome, IE und Edge auch die Dechiffrierung von Microsoft Product Keys, Wifi-, Autologon-, Outlook-, Windows Live Mail-, Opera- und Safari-Passwörtern. OSForensics beherrscht sowohl die Entschlüsselung am Live-System als auch am Offline-System. Eine besonders nützliche Funktion ist die Möglichkeit, im Zuge der Dechiffrierung einen Wörterbuchangriff auf ein nicht bekanntes Benutzerpasswort durchzuführen. Bei OSForensics handelt es sich jedoch um ein kommerzielles und mit $995 plus $345 jährlich für Support und Wartung sehr kostenintensives Programm. Eine kostenlose Testversion ist für 30 Tage verfügbar, sie besitzt aber nur eine eingeschränkte Funktionalität, beispielsweise werden nur 5 Passwörter je Browser entschlüsselt bzw. angezeigt.

Der Test mit dem vorbereiteten Image wurde unter folgender Konfiguration durchgeführt:

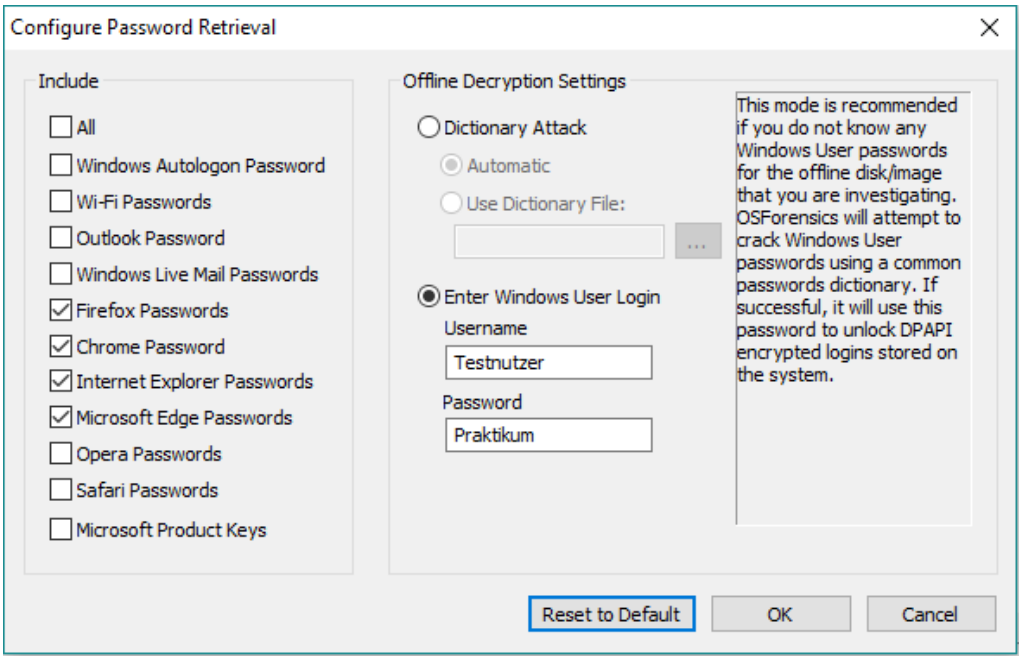


Abbildung 5.7: Einstellungen für die Passwort-Extraktion durch OSForensics (eigene Quelle)

Als Ergebnis lieferte OSForensics insgesamt 8 Datensätze, wie aus Abbildung 5.8 ersichtlich. Davon enthalten drei die korrekt entschlüsselten Chrome-Passwörter. Die übrigen fünf wurden aus den zwei Profilordnern von Firefox extrahiert, konnten jedoch nicht entschlüsselt werden. Das fehlende sechste Firefox-Passwort zeigte die Testversion nicht an, da für jeden Browser maximal 5 Datensätze einsehbar sind. Die zwei Passwörter aus dem IE- bzw. Edge-Browser wurden nicht gefunden.

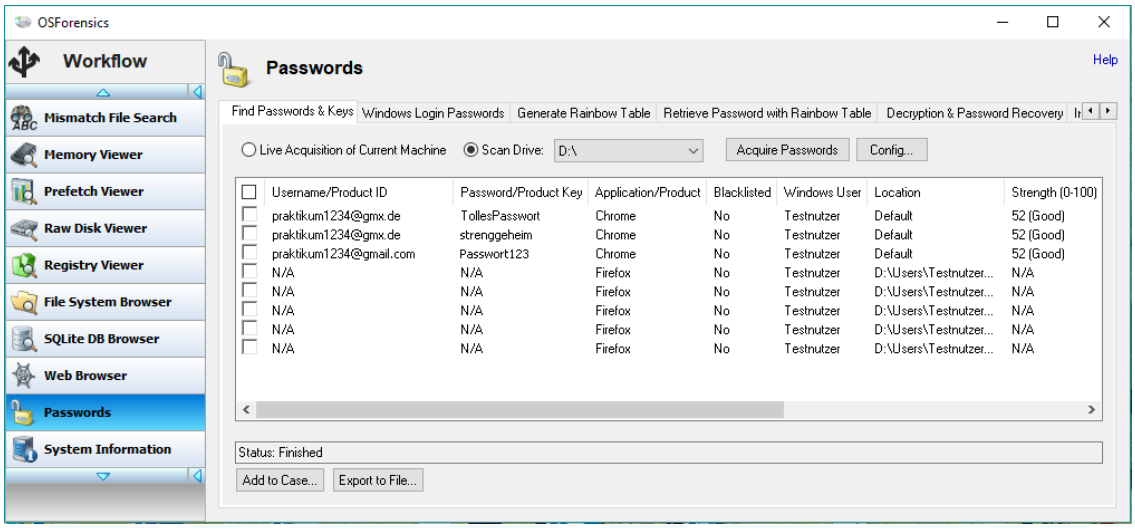


Abbildung 5.8: Ergebnisse aus OSForensics (eigene Quelle)

5.3.4 Browser Password Decryptor und Browser Password Recovery Pro

Browser Password Decryptor von XenArmor ist ein Tool zur Entschlüsselung von Passwörtern, das tatsächlich mit nur einem Knopfdruck auskommt. Allerdings können hier ausschließlich die Passwörter des laufenden Systems ausgewertet werden. Das Programm unterstützt Chrome, IE, Firefox, Edge, Opera und Safari sowie weitere eher unbekannte Browser wie Comodo Dragon, SeaMonkey, SRWare Iron und Flock.
Genau wie bei Nirsoft erfolgt auch hier die Anzeige der Webseiten, deren Passwörter nie gespeichert werden sollen, einschließlich der dazugehörigen Benutzernamen.

Browser Password Recovery Pro ist die kommerzielle Version zum Browser Password Decryptor. Die Pro Edition für den persönlichen Gebrauch kostet $99 und die Kosten für die Enterprise Edition belaufen sich auf $199. Eine freie Testversion steht 7 Tage lang zur Verfügung, jedoch werden nur die ersten beiden Zeichen der entschlüsselten Passwörter angezeigt.
Ergänzend zu den bereits genannten Browsern unterstützt Browser Password Recovery Pro auch den Sleipni, Tor und Vivaldi Browser. Anders als bei der freien Basis-Version können für jeden Browser einzeln die Pfade zu den Nutzerprofilen definiert werden. Eine Ausnahme bildet der Internet Explorer bzw. Edge, für den weiterhin nur die Möglichkeit der Entschlüsselung am Live-System besteht.

Für den Test mit dem vorbereiteten Image wurden die Pfade für Chrome und Firefox in den Einstellungen hinterlegt. Als Ergebnis lieferte Browser Passwort Recovery Pro eine Liste mit 29 Einträgen für IE bzw. Edge, die alle vom laufenden System extrahiert wurden. Davon sind zwei korrekt entschlüsselt worden, die übrigen repräsentieren die Webseiten, für die kein Passwort gespeichert werden soll.

Weiterhin enthält die Ergebnisliste acht Passwörter aus Chrome, von denen fünf ebenfalls vom lokalen System stammen. Die restlichen drei wurden aus dem eingebundenen Test-Image extrahiert, aber aufgrund der nicht geforderten Angabe eines Benutzerpasswortes falsch entschlüsselt, wie Abbildung 5.9 zeigt.
Für Firefox ergaben sich für keines der beiden Profile Ergebnisse, auch die auf dem laufenden System befindlichen FF-Passwörter lieferten keine Treffer.

Abbildung 5.9: Ergebnisse von Browser Password Recovery Pro (eigene Quelle)

5.3.5 Gegenüberstellung mit PasswordXTract

Die einzelnen Leistungen der vorgestellten Programme sind in Bezug auf die Testdaten in folgender Übersicht zusammengefasst. Ergänzt wird der tabellarische Vergleich durch die Ergebnisse von PasswordXTract.

Programm	**entschlüsselte Chrome-Passwörter**	**entschlüsselte FF-Passwörter**	**entschlüsselte IE-/Edge-Passwörter**
WebBrowserPassView	0 / 3	0 / 6	0 / 2
> ChromePass	3 / 3	0 / 6	0 / 2
> IE Pass View	0 / 3	0 / 3	2 / 2
> PasswordFox	0 / 3	6 / 6	0 / 2
OSForensics	3 / 3	0 / 6	0 / 2
Browser Password Recovery Pro	0 / 3	0 / 6	0 / 2
PasswordXTract	**3 / 3**	**6 / 6**	**2 / 2**

Tabelle 5.2: Vergleich der Testergebnisse

Allgemein haben viele der vorgestellten Programme Schwierigkeiten bei der Entschlüsselung von Passwörtern, die nicht auf dem laufenden System erzeugt wurden.
Zudem ist keines der Programme quelloffen, wodurch schwer nachzuvollziehen ist, auf welcher Basis sie arbeiten. Es lässt sich lediglich vermuten, dass oft nur die von Windows selbst bereitgestellte CryptUnprotectData()-Funktion genutzt wird, um Chrome- und IE- bzw. Edge-Passwörter zu entschlüsseln. Diese Funktion greift jedoch selbstständig auf das Benutzerpasswort des laufenden Systems zurück, so dass fremde Passwörter nicht entschlüsselt werden können.
Außerdem benötigen alle Programme detaillierte Pfadangaben zu den einzelnen Profilverzeichnissen der unterschiedlichen Browser. Die Handhabung ist deshalb für den Benutzer umständlich und setzt oft Hintergrundwissen zu den Speicherorten voraus.

Mit PasswordXTract wurde eine Lösung geschaffen, bei der diese Probleme aufgegriffen und verbessert wurden, so dass eine nutzerfreundliche Bedienung mit einer zuverlässigen Entschlüsselung kombiniert werden konnte. Im Hinblick auf die Testdaten ist PasswordXTract unter den aufgeführten Programmen das einzige, das alle Passwörter erfolgreich entschlüsseln konnte. Es lässt sich jedoch anhand dieses Tests nicht mit Sicherheit nachweisen, dass die anderen Programme tatsächlich nicht in der Lage sind, alle Passwörter zu entschlüsseln. Es wäre besonders für die kommerziellen Softwarepakete denkbar, dass in der Testversion einige Funktionen gesperrt sind, wodurch eine umfassende Dechiffrierung verhindert wird.

Neben der Entschlüsselung übernimmt PasswordXTract zusätzlich auch die Installation benötigter Software und das Mounten des Images. Durch die Anzeige verschiedener Meldungen, kann der Nutzer jederzeit nachvollziehen, was im Hintergrund abläuft und in welcher Phase sich PasswordXTract momentan befindet.

Dennoch gibt es auch Punkte, in denen PasswordXTract noch Defizite im Vergleich zu den anderen Programmen aufweist. Zum einen besitzt es keine grafische Oberfläche, was voraussetzt, dass der Nutzer Kenntnis darüber hat, wie die Kommandozeile unter Administratorrechten ausgeführt und wie ein Batch-Skript damit aufgerufen werden kann.
Außerdem unterstützt PasswordXTract die Untersuchung von Live-Systemen nicht. Eine derartige Funktionalität war jedoch auch nicht Ziel dieser Arbeit und damit nicht als Nachteil zu werten.
Die meisten anderen Programme listen neben den gespeicherten Passworten auch die URL und den Benutzernamen der Webseiten auf, für die der „Passwort speichern“-Dialog mit „nie“ o.ä. beantwortet wurde. PasswordXTract hingegen schließt nur Webseiten mit gespeicherten Passwörtern ein. Dadurch wird das Ergebnis einerseits übersichtlicher, andererseits könnten die zusätzlichen Informationen aber auch hilfreich sein, um weitere Webseiten ausfindig zu machen, auf denen der Nutzer aktiv ist.
Weiterhin beschränkt sich PasswordXTract auf Firefox, Chrome sowie IE bzw. Edge und die Windows-Versionen 8 bis 10, während andere Programme auch unbekannte Browser und ältere Windows- bzw. Browser-Versionen auswerten können.
OSForensics bietet zudem die Möglichkeit, im Vorfeld der Entschlüsselung einen Wörterbuchangriff auf das Benutzerpasswort durchzuführen, während PasswordXTract voraussetzt, dass dieses bekannt ist. Hierbei gilt zudem die Beschränkung, dass der Nutzer, dessen Passwörter entschlüsselt werden sollen, ein normales Windows-Anmelde-Passwort benutzt. Wird ein Picture Password oder eine PIN verwendet ist PasswordXTract nicht in der Lage, Chrome- und IE- bzw. Edge-Passwörter zu dechiffrieren.
Einen weiteren Faktor, bei dem PasswordXTract den anderen Programmen unterlegen ist, bildet die Laufzeit. PasswordXTract benötigt für die Entschlüsselung länger, doch in Anbetracht der Ergebnisse erscheint auch eine Dauer von wenigen Sekunden bis Minuten akzeptabel.
PasswordXTract ist zudem abhängig von Drittanbieter-Pogrammen, wie OSFMount oder Mimikatz, wodurch ein fehlerfreier Ablauf die korrekte Funktion dieser Software voraussetzt. Dieser Aspekt bindet PasswordXTract auch an Windows als Auswertesytem. Für macOS oder Linux existieren zwar Alternativen zu OSFMount, wie zum Beispiel „xmount“. Auch Ruby und SQL ist auf anderen Betriebssystemen verfügbar, doch für Mimikatz ist derzeit keine gleichwertige Alternative bekannt.

5.4 Fazit

Abschließend kann gesagt werden, dass das Ziel, ein Programm zu entwickeln, das Browserpasswörter im Rahmen einer Post-Mortem-Analyse per Knopfdruck ausliest, mit PasswordXTract erfüllt wurde.
Insbesondere zeichnet sich PasswordXTract durch einen linearen Aufwand $O(n)$ und eine unkomplizierte Schritt-für-Schritt-Bedienung aus.
Im Vergleich zu anderen Programmen mit ähnlichen Funktionen hebt sich PasswordXTract durch seine Fähigkeit zur Verarbeitung von Passwörtern aus forensischen Images von bereits angebotener Software ab. Auch sind keine Pfadangaben zu den Browserspezifischen Speicherorten nötig, da diese aus dem angegeben Laufwerksbuchstaben und dem Benutzernamen automatisch abgeleitet werden. Erforderlich ist jedoch die Angabe eines Benutzerpasswortes, um die Entschlüsselung von Chrome und IE bzw. Edge zu gewährleisten.

5.5 Ausblick

PasswordXTract bietet noch viele Möglichkeiten zur Verbesserung und Weiterentwicklung. Zum einen kann das Programm dahingehend erweitert werden, dass auch die HTTP-Authentifizierungspasswörter aus dem Credential-Verzeichnis und IE-Passwörter aus der Registry von Windows7-Systemen entschlüsselt werden können.
Ebenso ist ein zusätzlicher Programmteil möglich, der das Windows-Anmelde-Passwort entschlüsselt, das dann zur Dechiffrierung von Chrome und IE bzw. Edge verwendet werden kann. In Ergänzung dazu bleibt zu analysieren, wie DPAPI bei Nutzerprofilen mit Autologin, Picture Password oder PIN arbeitet, um auch diese Varianten in PasswordXTract implementieren zu können.
Um die Bedienung noch nutzerfreundlicher zu gestalten, ist auch eine grafische Oberfläche für PasswordXTract denkbar, in die weitere Funktionen, wie z.B. ein Fortschrittsbalken integriert werden können, um die Abläufe durchsichtiger zu gestalten.
Weiterhin ist zu überlegen, ob der Source-Code von Mimikatz auf Linux oder macOS kompiliert werden kann, um PasswordXTract auch für die Auswertung auf einem Linux- bzw. Apple-Rechner nutzen zu können.
Um noch etwas weiter zu denken, sollte es möglich sein, die Entschlüsselung der verschiedenen Browser auf mehrere Prozessorkerne zu verteilen, um die Laufzeit zu verkürzen. Eine Parallelisierung sollte prinzipiell leicht zu realisieren sein, da für jeden Browser ein eigenes Skript zur Dechiffrierung verwendet wird. Diese Skripte, namentlich „decrypt_chrome.bat“, „decrypt_firefox.bat“ und „decrypt_vault.bat“, sind zudem unabhängig voneinander und werden vom zenralen Skript „PasswordXTract.bat“ aufgerufen.
PasswordXTract ist besonders für den Einsatz durch Strafverfolgungsbehörden konzipiert, um Ermittlern oder Sachverständigen einen Zugang zu Online-Profilen o.ä. und damit zu wichtigen Spuren zu gewähren.

Anhang A: ASN.1-DER

A.1 Simple Types

Simple Types	Tag	Typical Use
BOOLEAN	1	Model logical, two-state variable values
INTEGER	2	Model integer variable values
BIT STRING	3	Model binary data of arbitrary length
OCTET STRING	4	Model binary data whose length is a multiple of eight
NULL	5	Indicate effective absence of a sequence element
OBJECT IDENTIFIER	6	Name information objects
REAL	9	Model real variable values
ENUMERATED	10	Model values of variables with at least three states
CHARACTER STRING	*	Models values that are strings of characters from a specified characterset

Abbildung A.1: ASN.1-DER Simple Types [CAa]

A.2 Sructured Types

Structured Types	Tag	Typical Use
SEQUENCE	16	Models an ordered collection of variables of different type
SEQUENCE OF	16	Models an ordered collection of variables of the same type
SET	17	Model an unordered collection of variables of different types
SET OF	17	Model an unordered collection of variables of the same type
CHOICE	*	Specify a collection of distinct types from which to choose one type
SELECTION	*	Select a component type from a specified CHOICE type
ANY	*	Enable an application to specify the type **Note:** ANY is a deprecated ASN.1 Structured Type. It has been replaced with X.680 Open Type.

Abbildung A.2: ASN.1-DER Structured Types [CAb]

Anhang B: Messreihen Laufzeit

B.1 Auswirkung der Anzahl der Masterkeys auf die Laufzeit

Anzahl Masterkeys	t_1 **in s**	t_2 **in s**	t_3 **in s**	$t_{average}$ **in s**
1	2	2	1	1,666
2	4	3	3	3,333
3	5	5	6	5,333
4	8	8	7	7,666
5	10	10	10	10,0
10	19	19	20	19,333
15	30	30	30	30,0
20	39	39	39	39,0
25	50	49	49	49,333
30	60	60	59	59,666
35	70	69	69	69,333

Tabelle B.1: Messreihe Anzahl Masterkeys - Laufzeit

B.2 Verhältnis zwischen der Anzahl an Einträgen in der „Login Data“-Datenbank und Laufzeit

Anzahl Einträge	t_1 **in s**	t_2 **in s**	t_3 **in s**	$t_{average}$ **in s**
1	3	3	3	3,0
2	3	3	3	3,0
3	3	3	3	3,0
4	3	3	3	3,0
5	3	3	3	3,0
10	3	3	3	3,0
15	4	3	3	3,333
20	4	4	4	4,0
30	4	4	4	4,0
40	4	4	5	4,333
50	5	5	5	5,0
60	5	5	5	5,0
70	6	5	5	5,333
80	5	5	6	5,333

Tabelle B.2: Messreihe Anzahl Einträge in „Login Data“ - Laufzeit

B.3 Verhältnis zwischen der Anzahl an vcrd-Dateien im Vault-Verzeichnis und Laufzeit

Anzahl vcrd-Dateien	t_1 **in s**	t_2 **in s**	t_3 **in s**	$t_{average}$ **in s**
1	3	3	3	3,0
2	3	3	3	3,0
3	3	3	3	3,0
4	3	4	3	3,333
5	4	3	3	3,333
10	3	4	4	3,666
15	4	4	4	4,0
20	4	5	4	4,333
30	5	5	5	5,0
40	9	8	9	8,666
50	10	9	9	9,333
60	10	10	9	9,666
70	11	11	10	10,666
80	14	13	13	13,333

Tabelle B.3: Messreihe Anzahl vcrd-Dateien - Laufzeit

B.4 Verhältnis zwischen der Anzahl an Einträgen in der „logins.json“ und Laufzeit

Anzahl Einträge	t_1 **in s**	t_2 **in s**	t_3 **in s**	$t_{average}$ **in s**
1	8	8	9	8,333
10	9	8	8	8,333
20	8	9	8	8,333
30	8	8	9	8,333
40	9	8	9	8,666
50	9	9	8	8,666
100	9	9	9	9,0
200	9	9	10	9,333
300	10	10	10	10,0
400	11	12	11	11,333
500	12	12	12	12,0
600	13	13	12	12,666
700	14	13	13	13,333
800	14	14	14	14,0
900	15	15	15	15,0
1000	15	16	16	15,666

Tabelle B.4: Messreihe Anzahl Einträge in „logins.json“ - Laufzeit

Literaturverzeichnis

[AjW+] AliceWyman ; jscher2000 ; Wang, Cheng ; Underpass ; novica ; Tonnes ; Rodaro, Michele ; Sharp, Gavin ; Verdi, Michael ; scoobidiver ; rampuse ; Sengupta, Swarnava ; Matt_G ; adampeebleswrites ; user832823 ; scootergrisen ; Joni: *Delete browsing, search and download history on Firefox*. – https://support.mozilla.org/en-US/kb/delete-browsing-search-download-history-firefox, letzter Zugriff: 28.05.2018

[AZZ+10] Alanazi, Hamdan ; Zaidan, B. B. ; Zaidan, A. A. ; Jalab, Hamid A. ; Shabbir, M. ; Al-Nabhani, Yahya: New comparative study between DES, 3DES and AES within nine factors. In: *Journal of Computing* 2 (2010). – ISSN 2151–9617

[Bil18] Bilendi: *Nutzen Sie unterschiedliche Passwörter für unterschiedliche Dienste?* 2018. – https://de.statista.com/statistik/daten/studie/818713/umfrage/nutzung-von-unterschiedlichen-passwoertern-fuer-unterschiedliche-dienste-in-deutschland/, letzter Zugriff: 30.05.2018

[Boj11] Boja, Catalin: Security Survey of Internet Browsers Data Managers. In: *Journal of Mobile, Embedded and Distributed Systems* 3 (2011), Nr. 3. – ISSN 2067–4074

[BP10] Burzstein, Elie ; Picod, Jean M.: *Recovering Windows Secrets and EFS Certificates Offline*. 2010. – https://elie.net/static/files/reversing-dpapi-and-stealing-windows-secrets-offline/reversing-dpapi-and-stealing-windows-secrets-offline-paper.pdf, letzter Zugriff: 28.05.2018

[Bry15] Bryson, John: FIPS 180-4 - Secure Hash Standard (SHS) / Information Technology Laboratory National Institute of Standards and Technology Gaithersburg. 2015. – Forschungsbericht

[CAa] Cassel, Lillian N. ; Austing, Richard H.: *ASN.1 Simple Types*. – https://www.obj-sys.com/asn1tutorial/node10.html, letzter Zugriff: 27.05.2018

[CAb] Cassel, Lillian N. ; Austing, Richard H.: *ASN.1 Structured Types*. – https://www.obj-sys.com/asn1tutorial/node11.html, letzter Zugriff: 27.05.2018

[Cle13] CLEVY, Laurent: *Protection des mots de passe par Firefox et Thunderbird : analyse par la pratique*. 2013. – https://connect.ed-diamond.com/MISC/MISC-069/Protection-des-mots-de-passe-par-Firefox-et-Thunderbird-analyse-par-la-pratique, letzter Zugriff: 27.05.2018

[Del17] DELAUNAY, Jean-Christophe: *DPAPI exploitation during pentest and password cracking*. 2017. – https://www.synacktiv.com/ressources/univershell_2017_dpapi.pdf, letzter Zugriff: 28.05.2018

[dru15] DRUIDE: *Firefox et son trousseau de mot de passes*. 2015. – https://www.druid.es/content/firefox-son-trousseau-mot-passes, letzter Zugriff: 27.05.2018

[GHBC01] GRIFFIN, Wesley ; HEYMAN, Michael ; BALENSON, David ; CARMAN, David: *Windows Data Protection*. 2001. – https://msdn.microsoft.com/en-us/library/ms995355.aspx, letzter Zugriff: 28.05.2018

[Gun09] GUNEYDAS, Ismail: *How FF store your passwords? Is it secure?* 2009. – http://realinfosec.com/?p=111, letzter Zugriff: 28.05.2018

[Hei10] HEISE: *Internet-Timeline @ iX*. 2010. – https://web.archive.org/web/20101025071338/http://www.heise.de/ix/timeline/?rubrik=1626, letzter Zugriff: 27.05.2018

[ITU03] ITU-T STUDY GROUP 17: ITU-T Recommendation X.690 - Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) / INTERNATIONAL TELECOMMUNICATION UNION. 2003. – Forschungsbericht

[Jan17] JANUSZKIEWICZ, Paula: *DPAPI and DPAPI-NG: Decryption aka. Hacking Toolkit*. 2017. – https://de.slideshare.net/paulajanuszkiewicz/black-hat-europe-2017-dpapi-and-dpaping-decryption-toolkit, letzter Zugriff: 28.05.2018

[Jor13] JORDAN: *How Browsers Store Your Passwords (and Why You Shouldn't Let Them)*. 2013. – http://raidersec.blogspot.de/2013/06/how-browsers-store-your-passwords-and.html, letzter Zugriff: 28.05.2018

[Jos06] JOSEFSSON, S.: *The Base16, Base32, and Base64 Data Encodings*. 2006. – https://tools.ietf.org/html/rfc4648#section-4, letzter Zugriff: 28.05.2018

[Kal00] KALISKI, B.: PKCS #5: Password-Based Cryptography Specification Version 2.0 / RSA Laboratories. 2000. – Forschungsbericht

[Kar] KARADENIZ, Besim: *Die Geschichte des Webbrowsers*. – http://www.netplanet.org/www/browser.shtml, letzter Zugriff: 27.05.2018

[KBC97] KRAWCZYK, H. ; BELLARE, M. ; CANETTI, R.: HMAC: Keyed-Hashing for Message Authentication / IBM and UCSD. 1997. – Forschungsbericht

[Lar] LARSON, Mark: *Chrome Releases*. – https://chromereleases.googleblog.com/2008/09/, letzter Zugriff: 27.05.2018

[LaZ] LAZAGNE: *firepwd.py, an open source tool to decrypt Mozilla protected passwords*. – https://jacknorthrup.com/jupyter-notebooks/retrieve-firefox-passwords-enter-sqlite-database.html, letzter Zugriff: 04.07.2018

[Mica] MICROSOFT: *ALG_ID*. – https://msdn.microsoft.com/en-us/library/windows/desktop/aa375549(v=vs.85).aspx, letzter Zugriff: 28.05.2018

[Micb] MICROSOFT: *CryptProtectData Function*. – https://msdn.microsoft.com/en-us/library/aa380261(vs.85).aspx, letzter Zugriff: 28.05.2018

[Micc] MICROSOFT: *CryptUnprotectData Function*. – https://msdn.microsoft.com/en-us/library/aa380882(v=vs.85).aspx, letzter Zugriff: 28.05.2018

[Micd] MICROSOFT: *Security Glossary*. – https://msdn.microsoft.com/en-us/library/windows/desktop/ms721573(v=vs.85).aspx, letzter Zugriff: 28.05.2018

[Mic18] MICROSOFT: *Der Support für Windows 7 endet am 14. Januar 2020*. 2018. – https://support.microsoft.com/de-de/help/4057281/windows-7-support-will-end-on-january-14-2020, letzter Zugriff: 26.07.2018

[MOV96] MENEZES, Alfred J. ; OORSCHOT, Paul C. ; VANSTONE, Scott A.: *Handbook of Applied Cryptography*. 1996

[Moz] MOZILLA: *History of the Mozilla Project*. – https://www.mozilla.org/en-US/about/history/details/, letzter Zugriff: 27.05.2018

[Moz18] MOZILLAZINE: *Transferring data to a new profile - Firefox*.

2018. – http://kb.mozillazine.org/Transferring_data_to_a_new_profile_-_Firefox, letzter Zugriff: 04.07.2018

[Nir] NIRSOFT: *VaultPasswordView v1.08*. – http://www.nirsoft.net/utils/vault_password_view.html, letzter Zugriff: 27.05.2018

[Nir16] NIRSOFT: *IE PassView v1.40 - Recover lost passwords stored by Internet Explorer*. 2016. – http://www.nirsoft.net/utils/internet_explorer_password.html

[Nir17] NIRSOFT: *WebBrowserPassView v1.86*. 2017. – http://www.nirsoft.net/utils/web_browser_password.html, letzter Zugriff: 27.07.2018

[Ora] ORACLE: *BLOB data type*. – https://docs.oracle.com/javadb/10.10.1.2/ref/rrefblob.html, letzter Zugriff: 28.05.2018

[Pasa] PASSCAPE: *Windows Password Recovery - DPAPI Master Key analysis*. – https://www.passcape.com/windows_password_recovery_dpapi_master_key, letzter Zugriff: 28.05.2018

[Pasb] PASSCAPE: *Windows Password Recovery - Vault Explorer and Decoder*. – https://www.passcape.com/windows_password_recovery_vault_explorer#v2, letzter Zugriff: 27.05.2018

[Pas12] PASSCAPE_ADMIN: *DPAPI Secrets. Security analysis and data recovery in DPAPI*. 2012. – https://www.passcape.com/index.php?section=blog&cmd=details&id=20, letzter Zugriff: 28.05.2018

[Paw17] PAWLASZCZYK, Dirk: *Digitaler Tatort, Sicherung und Verfolgung digitaler Spuren*. Springer, 2017. – ISBN 978–3–662–53800–5

[PB10] PICOD, Jean-Michel ; BURSZTEIN, Elie: *Decrypting DPAPI data*. 2010. – https://elie.net/static/files/reversing-dpapi-and-stealing-windows-secrets-offline/reversing-dpapi-and-stealing-windows-secrets-offline-slides.pdf, letzter Zugriff: 28.05.2018

[Pic] PICOD, Jean-Michel: *DPAPI offline decryption utility*. – https://github.com/jordanbtunker/dpapick, letzter Zugriff: 05.07.2018

[Pic14] PICASSO, Francesco: *give me the password and I'll rule the world - dpapi, what else?* 2014. – https://digital-forensics.sans.org/summit-archives/dfirprague14/Give_Me_the_Password_and_Ill_Rule_the_World_Francesco_Picasso.pdf, letzter Zugriff:28.05.2018

[Pic16] PICASSO, Francesco: *Windows ReVaulting*. 2016. – http://blog.digital-forensics.it/2016/01/windows-revaulting.html, letzter Zugriff: 27.05.2018

[Pon] PONS: *browse*. – https://de.pons.com/Ãijbersetzung/englisch-deutsch/browse, letzter Zugriff: 28.05.2018

[Sof18] SOFTWARE, PassMark: *OSFMount*. 2018. – https://www.osforensics.com/tools/mount-disk-images.html, letzter Zugriff: 03.07.2018

[ST01] STANDARDS, National I. ; TECHNOLOGY: ADVANCED ENCRYPTION STANDARD (AES) / National Institute of Standards and Technology. 2001. – Forschungsbericht

[Sta18a] STATCOUNTER: *Desktop Windows Version Market Share Germany*. 2018. – http://gs.statcounter.com/windows-version-market-share/desktop/germany/#monthly-201701-201807

[Sta18b] STATCOUNTER: *Marktanteile der führenden Browserfamilien an der Internetnutzung in Deutschland von Januar 2009 bis März 2018*. 2018. – https://de.statista.com/statistik/daten/studie/13007/umfrage/marktanteile-der-browser-bei-der-internetnutzung-in-deutschland-seit-2009/, letzter Zugriff: 28.05.2018

[Tim13] TIMBERWOLF, White: *Electronic Codebook (ECB) mode encryption*. 2013. – https://upload.wikimedia.org/wikipedia/commons/d/d6/ECB_encryption.svg, letzter Zugriff: 27.05.2018

[War15] WARREN, Tom: *This is Microsoft Edge, the replacement for Internet Explorer*. 2015. – https://www.theverge.com/2015/4/29/8511169/microsoft-edge-official-name-internet-explorer-upgrade, letzter Zugriff: 26.07.2018

[Wob01] WOBST, Reinhard: *Abenteuer Kryptologie*. 3. Addison-Wesley Verlag, 2001

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, August 2018