Technische Berichte in Digitaler Forensik

Herausgegeben vom Lehrstuhl für Informatik 1 der Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) in Kooperation mit dem Masterstudiengang Digitale Forensik (Hochschule Albstadt-Sigmaringen, FAU, Goethe-Universität Frankfurt am Main)

Quantenprogrammiersprache Q#: Forensische Spuren im Dateisystem

Michael Terörde

19.11.2018

Technischer Bericht Nr. 16

Zusammenfassung

Es ist davon auszugehen, dass sobald frei programmierbare Quantencomputer zur Verfügung stehen, diese zukünftig auch für kriminelle Zwecke verwendet werden. Verfügt jemand über einen skalierbaren universellen Quantencomputer, gehen sämtliche Sicherheitsgarantien der Public-Key-Kryptographie verloren. Damit wären verschlüsselte oder signierte E-Mail-Kommunikation oder sicheres Surfen im Internet mittels HTTPS und VPNs unmöglich. Grundsätzlich besteht jeder Quantencomputer auch aus einem herkömmlichen Rechner, der das quantenmechanische Experiment steuert, das Ergebnis ausliest und auswertet. In diesem Anteil des Quantencomputer-Systems sind herkömmliche Speicher wie Arbeitsspeicher und Festplatten enthalten. Die untersuchte Quantenprogrammiersprache Q# ist als .NET Sprache umgesetzt und erzeugt persistente Spuren im Dateisystem, in dem Windows-Prefetch-Ordner und in der Registry. Somit können im Rahmen einer Datenträgerforensik-Analyse Rückschlüsse auf die Verwendung von Q# und eine frühere Installation gezogen werden.

Der technische Bericht ist im Rahmen des Studiengangs Digitale Forensik unter der Anleitung von Prof. Martin Rieger und Prof. Holger Morgenstern entstanden.

1 Einführung

Es ist davon auszugehen, dass sobald frei programmierbare Quantencomputer zur Verfügung stehen, diese zukünftig auch für kriminelle Zwecke verwendet werden. Verfügt jemand über einen skalierbaren universellen Quantencomputer, gehen sämtliche Sicherheitsgarantien der Public-Key-Kryptographie verloren [Bsi16]. Alle derzeit gebräuchlichen Verschlüsselungsverfahren wie RSA, DSA, DH, SSH, TLS, IPSec, PGP und S/MINE wären unbrauchbar [Poh17]. Damit wären verschlüsselte oder signierte E-Mail-Kommunikation oder sicheres Surfen im Internet mittels HTTPS und VPNs unmöglich. Unter Zuhilfenahme eines Quantencomputers können Kriminelle Spionage betreiben, digitale Signaturen fälschen, Kopierschutzmechanismen umgehen und Wasserzeichen aus geschützten Werken entfernen. Ebenso können Kriminelle gespeicherte Daten nachträglich entschlüsseln und diese z.B. für Erpressungen missbrauchen. Quantencomputer gefährden auch Blockchain-Technologien wie die Kryptowährung Bitcoin, da diese auf dem SHA-256-Algorithmus und den ECDSA-Verfahren basieren, die beide durch einen Quantencomputer geknackt werden können [Btc16].

Grundsätzlich besteht jeder Quantencomputer auch aus einem herkömmlichen Rechner, der das quantenmechanische Experiment steuert, das Ergebnis ausliest und auswertet. In diesem Anteil des Quantencomputer-Systems sind herkömmliche Speicher wie Arbeitsspeicher und Festplatten enthalten. Um einen Quantencomputer einzusetzen, müssen Programme geschrieben werden. Diese werden durch einen Quanten-Compiler in physikalische Effekte (z.B. Laserpolarisation, Mikrowellenstrahlung) umgewandelt, die wiederum auf die Quanten-Register (eine Folge von Qubits) einwirken.

Die zur Programmierung eingesetzte Software soll die Verwendung von quantenmechanischen Eigenschaften wie der Superposition und der Verschränkung ermöglichen. Solch eine Quanten-Programmiersprache ist die Ende 2017 von Microsoft frei zur Verfügung gestellte Software Q#. Diese ist Teil des Quantum Development Kits und kann zusammen mit Microsoft Visual Studio verwendet werden. Mittels dieser Quantenprogrammiersprache sollen unabhängig von der jeweiligen Realisierung eines Quantencomputers zukünftig Quantenprogramme ausgeführt werden können. Dazu ist ein entsprechender Quantencompiler notwendig. Der eigentliche Quantencomputer wird durch den im Quantum Development Kit enthaltenen Simulator simuliert. Es ist realistisch anzunehmen, dass bei einer Programmierung eines Quanten-Programmes eine weitentwickelte Entwicklungsumgebung wie Visual Studio verwendet wird. Aufgrund der hohen Komplexität von Quantenalgorithmen erscheint eine Programmierung ohne Entwicklungsumgebung unwahrscheinlich.

Die forensische Relevanz des Themenfelds Quantencomputer ergibt sich nicht nur aus der Tatsache, dass diese böswillig und kriminell eingesetzt werden können, sondern auch daraus, dass diese auch für Forensiker zur Spurendetektion und -analyse und somit zum Lösen von Fällen eingesetzt werden können.

2 Quantencomputer

Ein Quantencomputer ist ein Computer, dessen Funktionsweise auf den Gesetzen der Quantenmechanik basiert. Es ist bereits bewiesen worden, dass bestimmte Probleme der Informatik mit Quantencomputern wesentlich effizienter gelöst werden können, als es mit klassischen Computern möglich ist.

2.1 Funktionsweise und Fähigkeiten

Die besonderen Eigenschaften erhält der Quantencomputer durch die Verwendung von Quantenbits, auch Qubits genannt. Ein Quantencomputer mit n Qubits kann sich in einer Überlagerung aus 2^n Basiszuständen befinden und somit mehr Zustände abbilden als herkömmliche Rechner, die nur n Zustände darstellen können.

Ein Quantencomputer mit 1.000 Qubits kann in der Superposition $2^{1.000}$ (etwa 10^{300}) Zustände abbilden und damit mehr als es Atome im Universum gibt. Dennoch wird bei jeder Messung nur einer dieser Zustände ausgegeben. Quantenalgorithmen nutzen nun die Eigenschaft, dass sich Amplituden destruktiv und konstruktiv verhalten können. Somit sollen Rechenwege, die zu einer falschen Lösung führen, destruktiv interagieren und Rechenwege, die zu einer korrekten Lösung führen, sollen sich konstruktiv verhalten. Dadurch wird die Wahrscheinlichkeit erhöht, bei der Messung ein korrektes Ergebnis zu erhalten [Spe12]. Es stellt sich somit die Aufgabe, für welche Rechenprobleme diese Interferenzen genutzt werden können und zwar derart, dass weniger Rechenschritte als auf einem klassischen Computer nötig sind.

2.2 Frei programmierbare Quantencomputer

Der frei programmierbare Quantencomputer arbeitet basierend auf den Prinzipien der Superposition und Verschränkung. Er wird auch als Universal Quantencomputer bezeichnet. Das System, das als Quantencomputer bezeichnet wird, besteht aus den folgenden Komponenten:

- Konventioneller Rechner
- Quantencompiler
- Ansteuerung
- Quantenbits
- Messeinrichtungen

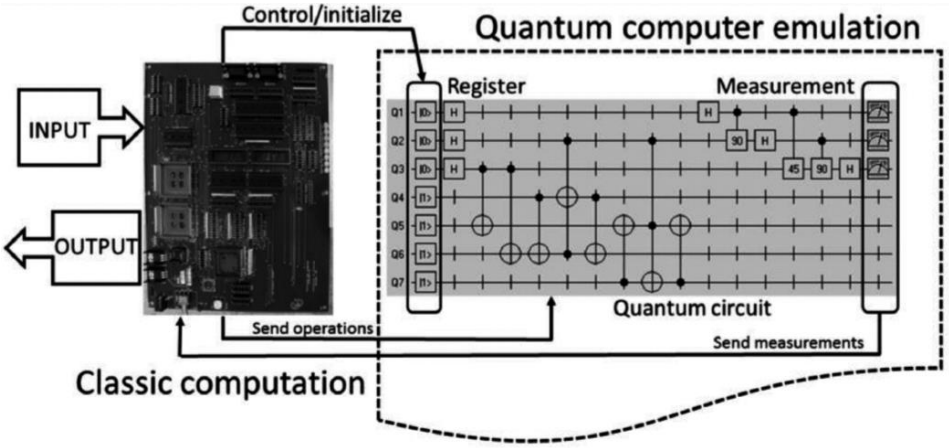


Bild 2.1 Quantum Random Access Modell eines frei programmierbaren Quantencomputers [Lah16]

Das am meisten verwendete Modell zur Beschreibung eines Quantencomputers ist das Quantum Random Access Modell (QRAM-Modell). Dabei steuert ein klassischer Rechner als Master einen Quanten-Prozessor als Slave (siehe **Bild 2.1**, wobei links der Master und rechts der Slave zu sehen ist). Der klassische Rechner enthält das eigentliche Programm, das als Bytecode an einen Quantencompiler geschickt wird. Der Quantencompiler übersetzt die Programmbefehle in die physikalische Manipulation der Qubits. Diese ist abhängig von der verwendeten Technik, z.B. ob Supraleiter, Photonen, topologische Isolatoren oder Atome verwendet werden und kann aus Laserimpulsen oder Mikrowellenimpulsen bestehen. Da die physikalische Form der zukünftigen einsatzbereiten Quantencomputer nicht absehbar ist, ist die Entwicklung eines Quantencompilers schwierig [Gol17]. Ein solcher Quantencompiler wandelt die Quantenprogramme in physikalische Manipulationen der Qubits um. Mittels Messgeräten, die wieder von der jeweiligen Realisierung der Qubits abhängen, wird das Messergebnis zurück an den konventionellen Rechner gesendet. Ebenso wird das Messergebnis, also der Output des Quantencomputers, in einen konventionell arbeitenden Computer eingelesen und weiter verarbeitet. Es wird konventionelle Software genutzt werden können, um ein Quantenprogramm zu schreiben.

Die Herausforderungen beim Bau eines Quantencomputers betreffen insbesondere die Instabilität der Qubits aufgrund der Dekohärenz.

2.3 Bewertung im Hinblick auf die digitale Forensik

Dieses Kapitel zeigt, dass Quantencomputer zukünftig das Potential haben, den Informations- und Kommunikationssektor zu revolutionieren. Ein solches System kann aber auch für böswillige Angriffe oder Straftaten verwendet werden. Denkbare Szenarien sind das entschlüsseln von geheimen Nachrichten, die böswillige Veränderung von Blockchains und somit auch Kryptowährungen und das Fälschen von digitalen Signaturen.

Das Themenfeld von Quantencomputer-Forensik, im Englischen auch Quantum Forensics genannt, erfährt derzeit keine Beachtung von Forschern [Ove11]. Da ein Quantencomputer immer von einem konventionellen Rechner programmiert und gesteuert wird, können bekannte

Ansätze der digitalen Forensik auf Quantencomputer übertragen werden. Eine Spurenmenge von Quantencomputern ist jedoch nicht vorhanden und diese Arbeit beschäftigt sich erstmals damit.

Beim Einsatz von Malware, z.B. zum Diebstahl geistigen Eigentums, verwendet der Angreifer häufig verschlüsselte Kommunikation, um die Malware zu steuern und die Daten hochzuladen. Somit bleibt die Absicht der Malware obfuskiert und die hochgeladenen Daten unentdeckt. Damit kein Reverse Engineering einen symmetrischen Schlüssel finden kann, nutzen Angreifer Public-Key Cryptography. Kann ein Incident Responder Pakete mit dem asymmetrischen Schlüsselaustausch abfangen, kann der symmetrische Schlüssel entschlüsselt werden und alle hochgeladenen Daten und Befehle die ein Angreifer gesendet hat [Jod14]. Ebenso können Forensiker mittels Quantencomputer Fälle lösen, die in der Vergangenheit liegen, aber aufgrund von Verschlüsselungstechnik noch ungelöst sind.

3 Quantenprogrammierung

Ein Quantencomputer könnte mittels klassischer Algorithmen programmiert werden. Da dann aber die besonderen quantenmechanischen Eigenschaften wie Superposition und Verschränkung nicht genutzt werden, bringt es keine Vorteile gegenüber herkömmlichen Computern. Die Programmierung eines Quantencomputers unter Verwendung der quantenmechanischen Phänomene wird als Quantenprogrammierung (Kurzform für Quantencomputer-Programmierung) bezeichnet.

Quantenalgorithmen nutzen quantenmechanische Effekte und sind somit probabilistisch. Das bedeutet, dass das Ergebnis mit einer gewissen Wahrscheinlichkeit korrekt ist. Unerlässliche Quantenoperationen sind das CNOT-Gatter und die Hadamard-Transformation zwischen verschiedenen Basisvektoren. Es gibt zwei bekannte Quantenalgorithmen, den Shor- und den Grover-Algorithmus, die Quantencomputer in den jeweiligen Anwendungsszenarien allen anderen Computern überlegen machen.

3.1 Quanten-Programmiersprachen

Die Möglichkeiten eines Quantencomputers können nur realisiert werden, wenn es passende Programmiersprachen gibt. Herkömmlichen Programmiersprachen fehlt es an passenden Datenstrukturen und an notwendigen Operatoren, die für eine einfache Repräsentation und Manipulation von Quanten-Daten nötig sind [Sof08]. Quantenprogrammiersprachen sind Programmiersprachen, die entwickelt wurden, um Programme für Quantencomputer zu schreiben. Dabei müssen Quantenprogrammiersprachen Phänomene berücksichtigen, die bei klassischen Berechnungen nicht auftauchen wie Quantenbits, Verschränkung, destruktive Messung und das No-Cloning-Theorem [Sof08]. Neben diesen Quantencomputer-spezifischen Sprachkonstrukten enthalten.

3.2 Q#

Microsoft hat im Dezember 2017 die neue Programmiersprache Q# der Öffentlichkeit vor- und zur Verfügung gestellt. Diese ist Teil des Quantum Development Kits, das auf Visual Studio 2017 läuft und neben Q# auch einen Quantencomputer-Simulator sowie ergänzende Ressourcen für die Programmierung bereitstellt [Par17]. Damit möchte Microsoft die Entwicklung von Software für Quantencomputer fördern. Es handelt sich um eine .NET Sprache eingebettet in Visual Studio 2017.

Das Programm .NET, auch Dotnet genannt, ist eine freie, Opensource Entwicklungs-Plattform für eine Vielzahl von Applikationen. Die .NET Apps können in C#, F# oder Visual Basic geschrieben werden. Um .NET zu nutzen wird das .NET SDK (Software Development Kit) installiert. .NET erzeugt eine sprachunabhängige Runtime, wodurch die verwendete Programmiersprache frei gewählt werden kann. Am häufigsten werden die Programmiersprachen C# und Visual Basic verwendet, um Software auf der .NET Plattform zu entwickeln. Am häufigsten wird von .NET-Entwickler als Entwicklungsumgebung Visual Studio verwendet. Wesentliche Bestandteile von .NET sind das .NET Framework, die auf .NET aufsetzende Programmiersprachen und diverse zusätzliche Klassenbibliotheken von Microsoft. .NET ist heute allgemein neben Java die am meisten verwendete Softwareentwicklungsplattform für neue Softwareentwicklungsprojekte. Diese große Verbreitung und Flexibilität lässt vermuten, dass auch zukünftig Quanten-Programmiersprachen für .NET entwickelt werden.

Der lokal ausführbare Quantensimulator ist zum Testen eigener Entwicklungen geeignet. Es können 30 logische Qubits auf herkömmlichen Rechnern mit einer 64-Bit Windowsvariante genutzt werden. Über einen Simulator in der Azure-Cloud können 40 logische Qubits simuliert werden [Par17]. Im verwendeten Model gibt es drei Ebenen [Mic17b]:

- Klassische Berechnungen zum Einlesen von Eingabedaten, zur Vorbereitung der Quantenberechnung, Auslösen der Quantenberechnung und zur Ausgabe der Ergebnisse
- Quantenberechnungen, die auf dem Quanten-Simulator oder direkt auf Quantenbits ausgeführt werden und somit den Quantenalgorithmus implementieren
- Klassische Berechnungen, die vom Quantenalgorithmus während der Laufzeit verwendet werden

Neben bekannten Datentypen wie Int, Double, Bool, Range und String enthält Q# quantenspezifische Datentypen. Dies sind u.a. Qubit, Pauli und Result. Der Datentyp Qubit repräsentiert ein Quantenbit. Der Dateityp Pauli repräsentiert ein Element einer Einzel-Qubit-Pauli-Gruppe und wird zur Bezeichnung der Basisoperation für Rotationen verwendet. Der Dateityp Result repräsentiert das Ergebnis einer Messung und kann entweder den Eigenwert -1 oder +1 aufweisen.

Die unterschiedlichen Komponenten des Quantum Development Kits sind in **Tabelle 1** zusammengefasst.

Tabelle 1 Komponenten des Microsoft Quantum Development Kit [Mic17]

Komponente	Beschreibung
Q# Sprache und Compiler	Eine Domain-spezifische Programmiersprache zum Ausdruck von Quanten-Algorithmen, die benutzt wird um Unterprogramme zu schreiben
Q# Standardbibliothek	Diese Bibliothek enthält Operationen und Funktionen zur Unterstützung Q# Quanten-Algorithmen
Lokaler Quantencomputer-Simulator	Ein full-state-vector-Simulator optimiert für genaue Vektorsimulation
Quantencomputer Trace Simulator	Soll nicht eine Quanten-Umgebung simulieren, sondern dient zur Abschätzung von Ressourcen, die zur Ausführung eines Quantenprogramms benötigt werden.
Visual Studio Erweiterung	Beinhaltet Vorlagen für Q# und Q#-Projekte sowie ein Syntax Highlighting

Zur Ausführung von Q# wird eine 64-Bit Installation von Windows, macOS oder Linux benötigt. Notwendig ist ebenfalls .NET Core SDK 2.0 oder eine höhere Version. Hardware-seitig wird eine Advance Vector Extension (AVX) empfohlen, die standardmäßig bei Intel Prozessoren ab 2011 enthalten ist.

Die in Visual Studio mit Q# programmierten Funktionen mit quantenmechanischen Eigenschaften haben die Dateiendungen „.qs“. Diese Q#-Funktionen werden durch eine Datei mit der Endung „.cs“, die z.B. in C# geschrieben ist, aufgerufen.

Microsoft verspricht, dass sich die Q#-Programme in Zukunft problemlos auf echte Quanten-Hardware transferieren lässt [Seb17]. Damit stünde bereits jetzt eine Software bereit, die über einen Quanten-Compiler einen Quantencomputer steuern und dessen Ergebnisse auswertet kann.

4 Untersuchungsmethoden

4.1 Ablauf der forensischen Untersuchungen

Das Vorgehen bei der forensischen Analyse der Quantenprogrammiersprache ist an der Abfolge einer typischen Verwendung orientiert. Dazu wird der Ablauf in unterschiedliche Phasen unterteilt, wie in **Tabelle 2** beschrieben.

Tabelle 2 Übersicht der zu untersuchenden Phasen

Lfd. Nr.	**Aktion/Phase**	**Kurz-Beschreibung**
1	Installation von Visual Studio 2017 Community und .NET Core SDK	Installation von Visual Studio 2017 Community mittels des Installationsprogramms *vs_community__ 1527698340. 1530114608*. Es wird als spezielle Workload *.NET Core cross-platform development* ausgewählt und ansonsten werden die Standardeinstellungen verwendet.
2	Installation des Quantum Development Kits	Die Installation des Quantum Development Kits erfolgt über das Installationsprogramm *QsharpVSIX*
3	Bibliotheken und Beispiele von GitHub laden	Bibliotheken und Beispiele des Microsoft Quantum Development Kit werden von GitHub[1] in Visual Studio integriert. Dazu wird die URL *https://github.com/Microsoft/Quantum.git* bei Local Git Repositories in Visual Studio eingetragen. Damit ist es auf der lokalen Festplatte geklont.
4	Ausführung eines Quantenprogrammes	Das Programm *Teleportation* wird als Startprojekt festgelegt und gestartet.
5	Deinstallation von Visual Studio	Deinstallation von Visual Studio 2017, Visual Studio Installer sowie Microsoft .NET Core SDK – 2.1.201 mittels der Windows-Funktion *Programme deinstallieren* in der Systemsteuerung

Vor jeder der Phasen wird ein Snapshot zur Sicherung des Status der virtuellen Maschine erstellt. So kann später zu den einzelnen Phasen zurückgekehrt werden. Jede Phase wird mehrfach untersucht: Die virtuelle Maschine (VM) wird für jede Phase auf den entsprechenden Anfangszustand gesetzt, damit alle Untersuchungen denselben Ausgangs-zustand nutzen.

Damit betriebssystembedingte Änderungen aus der Spurenmenge entfernt werden, wird ein weiteres Abbild $S_{Leerlauf}$ erzeugt, bei dem drei Stunden lang keine Aktion ausgeführt wird. Dadurch erhält man nur Spuren, die vom Betriebssystem selbst erzeugt werden. Dieses

[1] Vollständige Internetseite: https://github.com/Microsoft/Quantum

Leerlauf-Abbild wird verwendet, um Spuren als charakteristisch zu klassifizieren, indem geprüft wird, dass charakteristische Spuren nicht in der Leerlauf-Spurenmenge vorkommen.

4.2 Beschreibungen der Phasen

Im Folgenden werden die einzelnen Phasen auf denen die Zustandsmethoden angewendet werden ausführlich erläutert, um die Ergebnisse nachvollziehen und reproduzieren zu können. Eine Phase bezeichnet dabei festgelegte durchzuführende Aktionen.

Zwingende Voraussetzung für die Installation des QDK sind die Installationen von Windows als 64-Bit Variante, vom .NET Core SDK in der Version 2.0 oder einer späteren Version sowie von Visual Studio 2017.

Grundsätzlich handelt es sich bei Q# um eine Programmiersprache, beim QDK um die zugehörige Entwicklungsplattform mit den notwendigen Bibliotheken und Compilern. Das .NET Core SDK ist eine allgemeine Entwicklungsplattform, die eine Laufzeitumgebung zur Verfügung stellt. Visual Studio ist eine integrierte Entwicklungsumgebung, bei der sich ein Programmierer auf die Programmierarbeit konzentrieren kann und unabhängig vom verwendeten System ist. Eine Beschreibung der für die Nutzung der Quantenprogrammiersprache notwendigen Software ist in **Tabelle 3** zu finden.

Tabelle 3 Notwendige Software für die Quantenprogrammiersprache

Software	**Beschreibung**
Visual Studio 2017	Integrierte Entwicklungsumgebung für .NET Anwendungen
.NET Core Software Development Kit 2.0	Freie und offene Entwicklungsplattform für unterschiedliche Applikationen, die u.a. eine Laufzeitumgebung für die Ausführung von Programmen bereitstellt. Ermöglicht die Programmerstellung mittels Kommandozeile. Wichtige Komponenten sind die Laufzeitumgebung CoreCLR, die Klassenbibliotheken, und der Anwendungsstarter dotnet.
Quantum Development Kit	Entwicklungsplattform für Q# mit den notwendigen Bibliotheken, Werkzeugen und dem Compiler zur Entwicklung von Quanten-Software
Q#	Quanten-Programmiersprache, die von .NET unterstützt wird und von Microsoft entwickelt wurde

4.2.1 Phase 1: Visual Studio und .NET Core SDK installieren

Die Auswertung dieser Phase liefert Informationen über die Software-Voraussetzungen zur Quantenprogrammierung und nicht über deren Anwendung. Sowohl Visual Studio als auch das .NET Core SDK sind zwingende Voraussetzungen für die Quantenprogrammierung mittels Q#.

Über die Internetseite *www.visualstudio.com*[2] kann die Installationsdatei für Visual Studio Community 2017 heruntergeladen werden. Beim Starten der Installerdatei muss die *Plattformübergreifende .NET Core-Entwicklung* mitinstalliert werden[3] (siehe **Bild 4.1**).

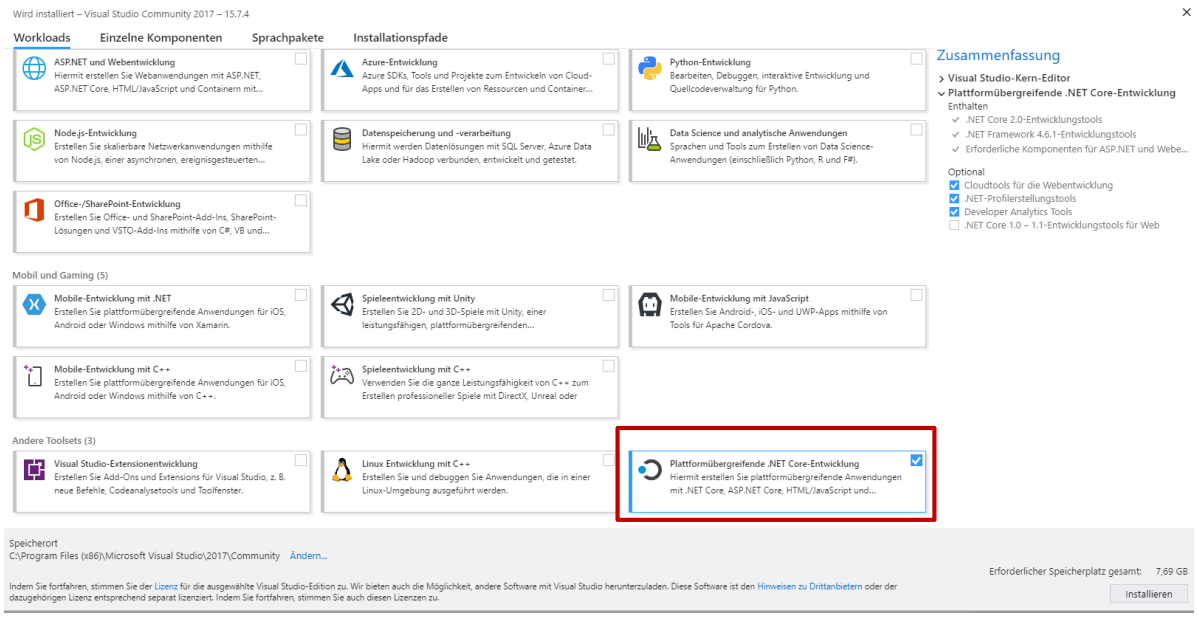


Bild 4.1 Oberfläche der Visual Studio Installerdatei

4.2.2 Phase 2: Quantum Development installieren

Nach Abschluss der Phase 1, wird über die Internetseite *https://marketplace. visualstudio.com/items?itemName=quantum.DevKit* das Microsoft Quantum Development Kit *QsharpVSIX* Version 0.2.1802.2202 (letztes Update 28.02.018) heruntergeladen. Diese 62 kB große Datei führt den Installationsprozess durch und installiert ebenso den Quantencomputer-Simulator. Dabei wird die Konfiguration von Visual Studio geändert (siehe **Bild 4.2**). Nach der QDK-Installation kann man Q# Applikationen und Libraries in Visual Studio erzeugen (siehe **Bild 4.3**). Der Installer liefert nach der Installation ein Protokoll.

[2] https://www.visualstudio.com/de/downloads/?rr=https%3A%2F%2Fdocs.microsoft.com %2Fde-de%2Fquantum%2Fquantum-installconfig%3Fview%3Dqsharp-preview%26tabs %3Dtabid-vs2017

[3] Eine nachträgliche Installation ist ebenfalls möglich.

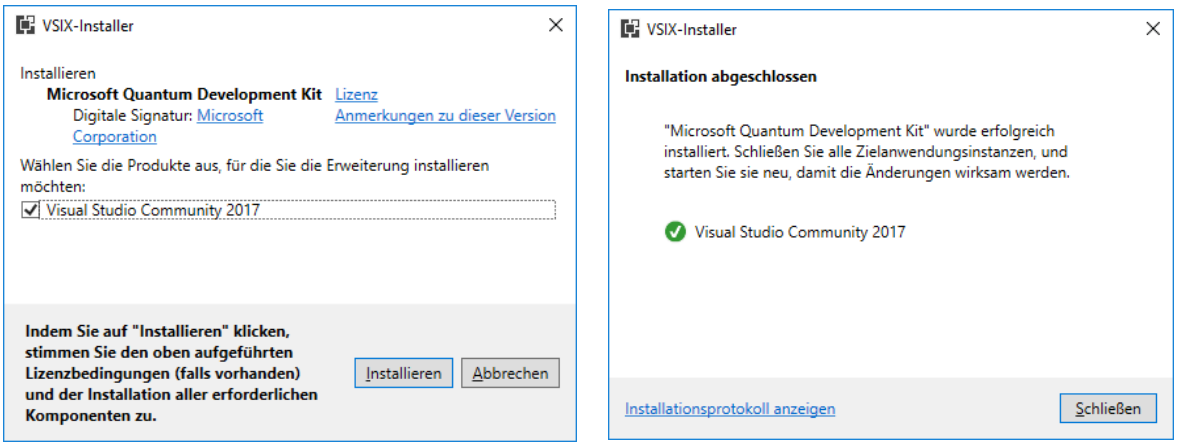


Bild 4.2 VSIX-Installer vor und nach der Installation

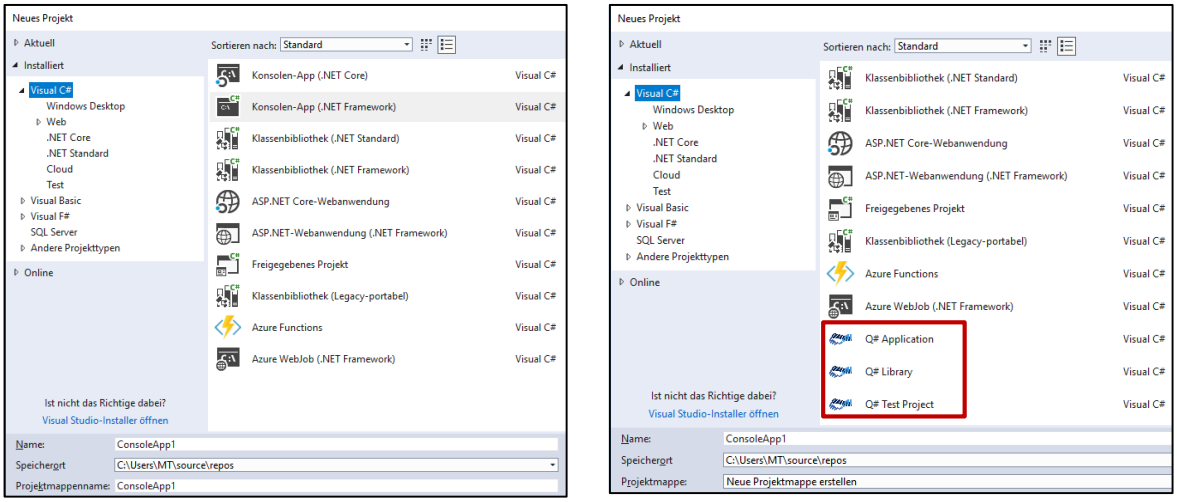


Bild 4.3 Nach der Visual Studio-Installation (links), nach der QDK-Installation (rechts)

Die Installerdatei vom QDK hat die Endung .vsix. Diese Dateiendung zeigt eine Erweiterung von Visual Studio an und kann deshalb nur mit Visual Studio ausgeführt werden. Eine VSIX-Datei ist eigentlich eine ZIP-Datei mit der Open Packaging Convention. Ändert man die Dateiendung in .zip sind die Inhalte im Explorer anzeigbar (siehe **Bild 4.4**). Entpackt sind die enthaltenen 32 Dateien 309 kB groß. Eine VSIX-Datei enthält die notwendige [Content_Types].xml-Datei, damit die API den Inhalt kennt. Die *extension.vsixmanifest*, ist das Manifest, welches die Erweiterung beschreibt[4]. Ein Ausschnitt dieser Datei ist in **Bild 4.5** zu sehen. Neben dem festgelegten anzuzeigenden Namen (DisplayName) „Microsoft Quantum Development Kit“ ist auch die Abfrage nach einer geeigneten Visual Studio Version als Installationsziel erkennbar.

[4] https://blogs.msdn.microsoft.com/quanto/2009/05/26/what-is-a-vsix/

Name	Typ	Komprimierte Größe	Größe
_rels	Dateiordner		
Grammars	Dateiordner		
ItemTemplates	Dateiordner		
package	Dateiordner		
ProjectTemplates	Dateiordner		
[Content_Types]	XML-Dokument	1 KB	2 KB
catalog.json	JSON-Datei	1 KB	1 KB
extension.vsixmanifest	VSIXMANIFEST-Datei	1 KB	2 KB
License	RTF-Dokument	31 KB	259 KB
manifest.json	JSON-Datei	2 KB	4 KB
mobius_strip_icon	PNG-Datei	1 KB	1 KB
mobius_strip_preview	PNG-Datei	7 KB	7 KB
syntax.pkgdef	PKGDEF-Datei	1 KB	1 KB

Bild 4.4 Inhalt der QsharpVSIX-Datei

extension.vsixmanifest - Editor

Datei Bearbeiten Format Ansicht ?

```
<?xml version="1.0" encoding="utf-8"?>
<PackageManifest Version="2.0.0" xmlns="http://schemas.microsoft.com/developer/vsx-schema/2011">
  <Metadata>
    <DisplayName>Microsoft Quantum Development Kit</DisplayName>
  <Installation InstalledByMsi="false">
    <InstallationTarget Id="Microsoft.VisualStudio.Pro" Version="[15.0,16.0)" />
    <InstallationTarget Id="Microsoft.VisualStudio.Community" Version="[15.0,16.0)" />
    <InstallationTarget Id="Microsoft.VisualStudio.Enterprise" Version="[15.0,16.0)" />
  </Installation>
  <Dependencies>
    <Dependency Id="Microsoft.Framework.NDP" DisplayName="Microsoft .NET Framework" Version="4.5" />
    <Dependency Id="Microsoft.VisualStudio.MPF.14.0" DisplayName="Visual Studio MPF 14.0" Version="[14.0,15.0)" />
  </Dependencies>
</PackageManifest>
```

Bild 4.5 Ausschnitt aus der Datei extension.vsixmanifest

4.2.3 Phase 3: Quantenprogramme von GitHub laden

Auf den Servern des Onlinediensts GitHub werden Software-Entwicklungsprojekte mittels Filehosting bereitgestellt. Nach Abschluss der Installation vom QDK wird Visual Studio geöffnet und folgende Zeile bei *Local Git Repositories* eingegeben:

https://github.com/Microsoft/Quantum.git

Dadurch werden Q#-Projekte auf die lokale Festplatte geladen. Das Projekt QsharpLibraries.sln wird in Visual Studio geöffnet und zeigt die nun integrierten unterschiedlichen Q#-Projekte. Bei Dateien mit der Endung .sln handelt es sich um Projektmappendateien von Visual Studio[5]. Die durch den GitHub-Download integrierten Projekte und Dateien sowie die Inhalte der QsharpLibraries sind in **Bild 4.6** zu sehen. Sollte vor der Installation vom QDK das .NET Core SDK nicht installiert worden sein, werden beim ersten Aufruf von QsharpLibraries.sln fehlende Features installiert, wozu in den meisten Fällen F# zählt. Diese Phase ist beendet, wenn alle notwendigen Features zum Starten der Quantenprogramme vorhanden sind.

[5] https://msdn.microsoft.com/de-de/library/xhkhh4zs.aspx

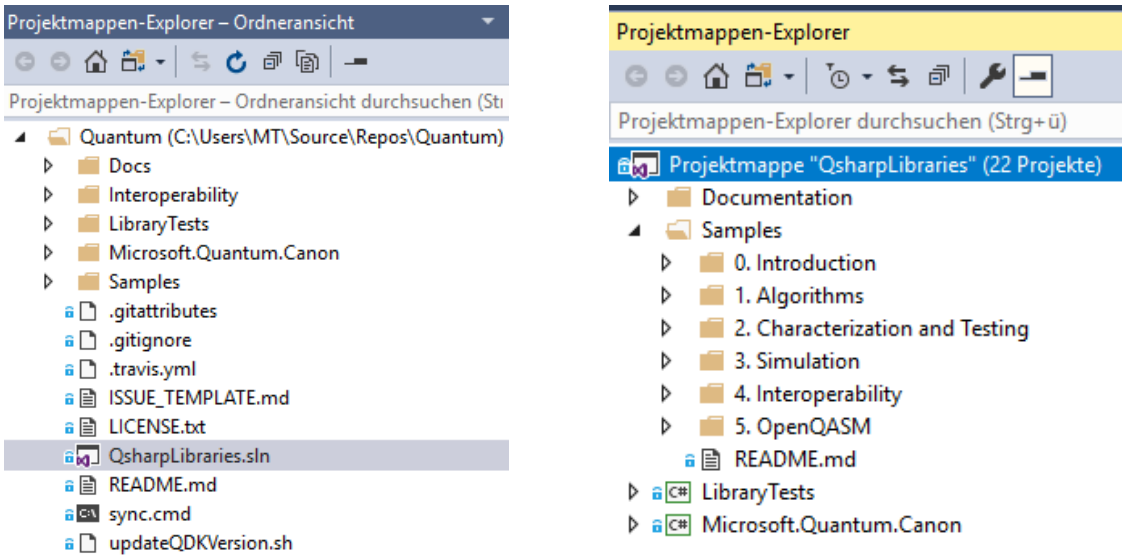


Bild 4.6 GitHub-Inhalte in Visual Studio (links), Inhalt von QsharpLibraries (rechts)

4.2.4 Phase 4: Quantenprogramm starten

Nach Phase 3 wird das Quantenprogramm *TeleportationSample* als Startup-Projekt festgelegt und gestartet. Dieses Programm ist Teil der Projektmappe QsharpLibrairies.sln, welche von GitHub für das Quantum Development Kit heruntergeladen werden kann. Es enthält typische Befehle von Quantenprogrammen, die man nicht in anderen Hochsprachen findet.

Ein Quantenprogramm in Q# besteht typischerweise aus zwei Teilen. Der Steuerteil mit der Endung „.cs“ beinhaltet nur klassische Befehle und ruft Funktionen aus dem zweiten Teil auf. Der zweite Teil mit der Endung „.qs“ enthält Funktionen mit Quantenbefehlen und kann als Quantenteil bezeichnet werden. Die Aufgabe des Programms Teleportation ist die Übertragung eines Quantenzustands von einem Qubit auf ein anderes unter Ausnutzung des Verschränkungsprinzips Das zu untersuchende Programm beinhaltet die Nutzung von quantenmechanischen Effekten. Es wird ein Qubit im Basiszustand |0> zunächst in Superposition gebracht und anschließend verschränkt. Das Ergebnis der Qubits wird ausgelesen und ausgewertet.

Die Phase 4 ist beendet, wenn das Konsolenfenster erscheint, das Programm durchgelaufen ist und der Text „*Teleportation successful*“ in der Konsole ausgegeben wird (siehe **Bild 4.7**).

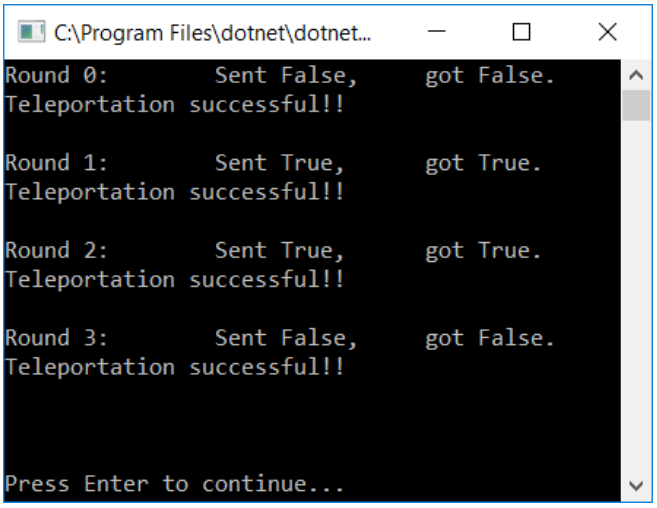


Bild 4.7 Konsolenfenster des Quantensimulator-Programms TeleportationSample

4.2.5 Phase 5: Deinstallation

In dieser Phase werden über das Standard-Windows-Vorgehen *Programme deinstallieren* die Programme Visual Studio 2017, Visual Studio Installer und .NET Core SDK ausgewählt und deinstalliert. Es werden mit der Zustandsmethode, anders als bei den anderen Phasen, mehrere Berichte von idifference2 erzeugt. Ein Bericht wird zwischen den Abbildern 4 und 5 erzeugt, das konkret zeigt, welche Aktionen beim Deinstallieren auf dem Dateisystem durchgeführt werden. Ein weiterer Report wird zwischen den Abbildern 0 und 5 erzeugt. Somit wird der Zustand vor der Installation mit dem Zustand nach der Deinstallation verglichen und dadurch werden Spuren detektiert, die auf eine frühere Installation hindeuten. Ein weiterer Bericht zwischen den Abbildern 1 und 5 zeigt Spuren nach einer Deinstallation, die auf die Nutzung von QDK (ohne Visual Studio und .NET Core SDK) hindeuten.

4.3 Zustandsmethode

Das Programm Q# wird auf einer virtuellen Maschine mit Windows 10 verwendet. Vor und nach den zu untersuchenden Aktionen wird ein Snapshot der Virtuellen Maschine (VM) erstellt. Die Snapshots werden auf Unterschiede untersucht durch das Programm idifference2.py aus dem Projekt DFXML, welches auf der Ubuntu-VM fiwalk verfügbar ist. Idifference2 vergleicht zwei Zustände des Dateisystems und ermöglicht die Spurendetektion von Aktionen durch Vergleich eines Zustands des Dateisystems vor der Aktion mit dem Zustand des Dateisystems nach der Aktion [Gar12].

Die **Tabelle 4** zeigt die sechs Snapshots 0 bis 5 mit den ersten acht Stellen der zugehörigen UUIDs, einer Kurzbeschreibung und der enthaltenen Textdatei auf dem Desktop. Weiterhin ist die Bezeichnung der aus den Snapshots erstellten RAW-Dateien N0 bis N5 aufgeführt.

Tabelle 4 Übersicht der Snapshots und RAW-Dateien

Snapshot	Snapshot-UUID	Beschreibung	Textdatei	RAW-Datei
0	3c2a2802	Ausgangssituation	0_M119_QPC	N0
1	246e2973	Visual Studio 2017 installiert	1_M119_QPC	N1
2	78a51f57	Quantum Development Kit installiert	2_M119_QDK	N2
3	f29c4b88	GitHub geladen und QsharpLibraries.sln ausgewählt	3_M119_QDK	N3
4	cfe4f805	Quantenprogramm Teleportation gestartet	4_M119_QDK	N4
5	96937956	deinstallieren von Visual Studio	5_M119_QDK	N5

Die **Tabelle 5** führt die idiff-Reporte auf und zeigt auf welchen Abbildern die Reporte basieren. Der idiff-Report D basiert bspw. auf einem Vergleich der Snapshots 3 und 4 und soll Spuren des Quantenprogramms anzeigen. Um sicherzustellen, dass es sich um den korrekten Report handelt und dieser fehlerfrei erstellt wurde, werden die enthaltenen Textdateien überprüft. Im konkreten Beispiel wird korrekt die Umbenennung der Textdatei von 3_M119 in 4_M119 angezeigt.

Tabelle 5 Übersicht der idiff-Reporte

Idiff2-Report	basiert auf	Beschreibung	Textdateien enthalten
A	0 / 1	Was mach Installation Visual Studio?	Renamed 0_M119 to 1_M119
B	1 / 2	Was mach Installation QDK?	Renamed 1_M119 to 2_M119
C	2 / 3	Was macht GitHub?	Renamed 2_M119 to 3_M119
D	3 / 4	Was macht Quantenprogramm?	Renamed 3_M119 to 4_M119
E	4 / 5	Was macht Deinstallation?	Renamed 4_M119 to 5_M119
F	0 / 5	Spuren einer Deinstallation von VS, .NETCore, QDK?	Renamed 0_M119 to 5_M119
G	1 / 5	Spuren einer Deinstallation von QDK und Github?	Renamed 1_M119 to 5_M119

4.4 Klassifizierung von Spuren

Unter dem Begriff digitale Spuren versteht man im Allgemeinen Spuren, die auf Daten basieren, welche in Computersystemen gespeichert oder übertragen worden sind. Dabei sind digitale Spuren immer auch physische Spuren, die beispielsweise als Ladezustand von Speicherzellen im Arbeitsspeicher oder als Magnetisierung auf der Oberfläche einer Festplatte vorliegen. Mit dieser Definition sind auch die Qubits bei Quantencomputern als digitale Spuren erfasst mit der Besonderheit, dass diese während der Durchführung von Berechnungen nicht binär sein müssen.

Im Rahmen dieser Arbeit werden charakteristische Spuren als solche definiert, die nur in einer der Spurenmengen der unterschiedlichen Phasen vorkommen und auch nicht bei der Leerlauf-Spurenmenge entstehen. Robuste Spuren werden als solche Spuren definiert, deren Entfernen oder Detektieren Detailkenntnisse über Betriebssysteme voraussetzt und welche somit einem typischen Standardanwender verborgen bleibt.

5 Spurenanalyse von Q# im Dateisystem

Wie in **Kapitel 4** beschrieben, werden die Spuren von Q# mittels der Zustandsmethode durch idifference2.py akquiriert und im Anschluss analysiert. In diesem Kapitel werden die Untersuchungsergebnisse und die technisch vermeidbare sowie technisch unvermeidbare Spurenmenge vorgestellt.

Bei der Quantenprogrammiersprache Q# sind keine anti-forensischen Verfahren implementiert und auch kein Interesse zur Vermeidung oder Verminderung von Spuren seitens des Herstellers gegeben. Aus diesem Grund liefern die eingesetzten Untersuchungsmethoden eine Vielzahl von Spuren. Es wird deshalb jeweils eine Auswahl an Spuren dargestellt.

Das Erstellen des Differenzabbildes zwischen zwei Snapshots mittels idifference2.py erzeugt eine Textdatei mit fünf Kategorien:

- Neu erstellte Dateien (New files)
- Gelöschte Dateien (Deleted files)
- Umbenannte Dateien (Renamed files)
- Dateien mit geänderten Inhalten (Files with modified contents)
- Dateien mit geänderten Eigenschaften (Files with changed properties)

Als Spuren werden insbesondere die neu erstellten und umbenannten Dateien ausgewertet. Anhand der Renamed Files wird auch sichergestellt, dass die korrekten idiff-Reporte ausgewertet werden. Das Verzeichnis C:/*Users/MT* ist das Benutzerverzeichnis, weshalb die Bezeichnung MT durch den Benutzernamen des angemeldeten Users entsprechend ersetzt werden muss.

5.1 Phase 1: Installation Visual Studio und .NET Core SDK

Die Installerdatei[6] mit der Anfangsbezeichnung *vs_community.exe*[7] ist 1,06 MB groß und ruft ein Menü zur Installation auf. Es muss das *.NET Core cross-platform development* Tool mitinstalliert werden, weshalb die Installation insgesamt 3,35 GB groß ist. Durch die Installation mit den in **Abschnitt 4.3** aufgeführten Einstellungen entstehen persistente Spuren im Dateisystem. Nach der Installation befinden sich viele neue Verzeichnisse und Dateien auf dem Dateisystem, die alle für die Anwendung relevanten Daten enthalten, wie das zur Ausführung von Q#-Programmen notwendige dotnet.exe-Programm (siehe **Bild 5.1**).

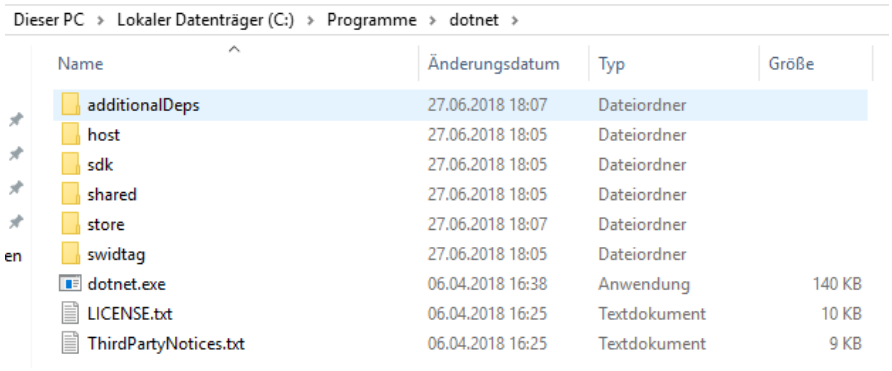


Bild 5.1 Inhalt des dotnet-Verzeichnisses

Neben den Installationsverzeichnissen werden auch Einträge im Startmenü von Windows und die Prefetch-Datei VS_COMMUNITY erzeugt. Die **Tabelle 6** beschreibt wichtige erzeugte Dateien und Verzeichnisse. Unerwartete Dateizugriffe erfolgen seitens des Virtual Studio und .NET Core SDK-Installationsprozesses nicht.

Tabelle 6 Verzeichnisse und Dateien bei der Installation

Nr.	Verzeichnis / Datei	Beschreibung
1	C:/Program Files/dotnet	Microsoft .NET Core SDK 2.1.301 erzeugt den Pfad und installiert hier dotnet.exe
2	C:/Program Files(x86)/Microsoft SDKs	Installationsverzeichnis von .NET Core SDK und F#
3	C:/Program Files(x86)/Microsoft Visual Studio	Installationsverzeichnis von Visual Studio
4	C:/ProgramData/Microsoft/Windows/Start Menu/Programs/Visual Studio 2017.lnk	Eintrag in der Start-Menüleiste von Windows
5	C:/Windows/Prefetch/VS_COMMUNITY__87450679.151941-24178403.pf	Prefetch-Datei des Installers
6	C:\Users\MT\source\repos	Hier werden die GitHub-Repositories abgelegt

[6] Heruntergeladen von https://www.visualstudio.com/de/downloads/?rr=https%3A%2F%2Fdocs.microsoft.com%2Fde- de%2Fquantum%2Fquantum-installconfig%3Fview%3Dqsharp-preview%26tabs%3Dtabid-vs2017

[7] Die exakte Bezeichnung lautet vs_community_1989901343.151673292.exe

Die **Tabelle 7** führt charakteristische Spuren für Phase 1 auf, bei deren Existenz auf die Installation von Visual Studio und .NET SDK Core geschlossen werden kann. Da diese Spuren nach einer Deinstallation nicht mehr existent sind, bedeutet dies, dass eine Deinstallation nicht durchgeführt wurde. Anhand der Bezeichnungen der Spuren lassen sich diese Visual Studio (Spuren Nr. 1-5) und .NET Core SDK (Spuren Nr. 6-10) zuordnen.

Tabelle 7 Charakteristische Spuren durch die Visual Studio sowie .NET Core SDK Installation

Nr.	Charakteristische Spur
1	Program Files (x86)/Common Files/microsoft shared/MSEnv/PublicAssemblies/de/Microsoft.VisualStudio.VSHelp.xml
2	Program Files (x86)/Microsoft Visual Studio/Shared/Packages/WebGrease.1.1.0/tools/WG.exe
3	Program Files (x86)/MSBuild/Microsoft/Portable/VisualStudio/v12.0/Microsoft.VisualStudio.PortableLibrary.Build.Tasks.dll
4	Program Files (x86)/NuGet/Config/Microsoft.VisualStudio.Offline.config
5	ProgramData/Microsoft/VisualStudio/Setup/x64/Microsoft.VisualStudio.Setup.Configuration.Native.dll
6	Program Files/Microsoft SDKs/Azure/.NET SDK/v2.9/packages/LocalPackagesInstallation Instructions.docx
7	Program Files (x86)/Reference Assemblies/Microsoft/Framework/.NETCore/v4.5/System.Runtime.dll
8	Program Files (x86)/Reference Assemblies/Microsoft/FSharp/.NETCore/3.259.3.1/FSharp.Core.dll
9	Program Files/dotnet/dotnet.exe
10	Program Files/Microsoft ASP.NET Core Runtime Package Store/2.0/eula.rtf

5.2 Phase 2: Quantum Development Kit

Die zur Installation notwendige Datei *QsharpVSIX.vsix* wird beim Herunterladen durch den Edge-Browser standardmäßig im Benutzerverzeichnis *Users/MT/Downloads/* abgelegt. Da das QDK eine Erweiterung (Extension) von Visual Studio ist, wird diese im Visual Studio-Verzeichnis eingetragen. Die Installation des QDK erzeugt keine Shortcuts oder Startprogramm-Einträge. Es kann nicht über *Programme deinstallieren* gelöscht werden. Da auch kein Extra-Programm zur Deinstallation vorhanden ist, kann es nicht mehr ohne Aufwand aus Visual Studio entfernt werden. Änderungen im Dateisystem werden hauptsächlich in den zwei Benutzerverzeichnissen

- *Users/MT/AppData/Local/Microsoft/VisualStudio/15.0_2063b37f/Extensions/gsc45wsn.anb/Grammars* und
- *Users/MT/AppData/Local/Microsoft/VisualStudio/15.0_2063b37f/Extensions/gsc45wsn.anb/ItemTemplates/CSharp/1033* (siehe Spur 4)

vorgenommen, da es sich beim QDK um eine Visual Studio-Erweiterung handelt. Charakteristische Spuren für diese Phasen sind die Dateien und Verzeichnisse in **Tabelle 8**. Als einzige dieser Spuren wird *Users/MT/AppData/Local/Microsoft/VisualStudio/15.0_2063b37f/Extensions/gsc45wsn.anb/Grammars/Qsharp/qsharp.tmLanguage.json* (Spur 2) bei einer Deinstallation entfernt. Somit weist diese Spur auf eine einsatzbereite Installation von Visual Studio und zeitgleich Q# hin. Diese Datei enthält neben der Versionsnummer von Q# (hier 0.1.6) auch die C# reservierten Wörter, die nicht in Q# verwendet werden dürfen. Im Prefetch-Verzeichnis wird die VSIXINSTALLER.EXE.pf-Datei mit aufgenommen.

Tabelle 8 Charakteristische Spuren neu erstellter Dateien durch das Quantum Development Kit

Nr.	Spur
1	Users/MT/AppData/Local/Microsoft/CLR_v4.0_32/UsageLogs/VSIXInstaller.exe.log
2	Users/MT/AppData/Local/Microsoft/VisualStudio/15.0_2063b37f/Extensions/gsc45wsn.anb/Grammars/Qsharp/qsharp.tmLanguage.json
3	Users/MT/Downloads/QsharpVSIX.vsix
4	Users/MT/AppData/Local/Microsoft/VisualStudio/15.0_2063b37f/Extensions/gsc45wsn.anb/ItemTemplates/CSharp/1033/QsharpFileTemplate/QsharpFileTemplate.vstemplate
5	Windows/Prefetch/VSIXINSTALLER.EXE-077612EF.pf
6	Windows/Prefetch/VSLAUNCHER.EXE-B0F8B5AA.pf

5.3 Phase 3: GitHub

Die Phase 3 lädt die Bibliotheken und Programmbeispiele von GitHub nach Visual Studio. Idiff2 liefert charakteristische Spuren für Phase 3, wie in **Tabelle 9** zu sehen.

Tabelle 9 Charakteristische Spuren für die Phase 3 GitHub

Nr.	Spur
1	ProgramData/Microsoft/NetFramework/BreadcrumbStore/netcore,FSharp.Core,4.2.3
2	Users/MT/.nuget/packages/microsoft.quantum.development.kit/0.2.1806.1503-preview/tools/qsc/FSharp.Core.dll
3	Users/MT/.nuget/packages/runtime.native.system.net.security/4.0.1/dotnet_library_license.txt
4	Users/MT/Documents/Visual Studio 2017/Code Snippets/Visual Basic/My Code Snippets
5	Users/MT/source/repos/Quantum/Samples/Teleportation/obj
6	Users/MT/source/repos/Quantum/Samples/UnitTesting/obj/qsharp
7	Users/MT/source/repos/Quantum/QsharpLibraries.sln
8	Users/MT/source/repos/Quantum/updateQDKVersion.sh
9	Windows/Prefetch/GIT.EXE-0BB09BF5.pf
10	Windows/Prefetch/GIT-REMOTE-HTTPS.EXE-C383836A.pf

Dabei sind alle Spuren auch nach der Deinstallationsphase vorhanden. Zum Laden der Bibliotheken wird die Datei git.exe und git-remote-https.exe verwendet. Die Quanten-Projekte werden in das Benutzer-Verzeichnis *Users/MT/source/repos/Quantum/Samples/* kopiert. Die

für das QDK benötigte Bibliothek *Users/MT/.nuget/packages/microsoft.quantum.development.kit/0.2.1806.1503-preview/tools/qsc/FSharp.Core.dll* wird ebenfalls geladen.

Die Datei *updateQDKVersion.sh* enthält ein Bash-Skript zur Aktualisierung des QDK. Die Datei *QsharpLibraries.sln* listet alle Visual Studio-Projekte mit Quantenprogrammen auf, wozu auch das TeleportationSample gehört (siehe **Bild 5.2**).

```
Microsoft Visual Studio Solution File, Format Version 12.00
# Visual Studio 15
VisualStudioVersion = 15.0.26730.16
MinimumVisualStudioVersion = 10.0.40219.1
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "Microsoft.Quantum.Canon",
"Microsoft.Quantum.Canon\Microsoft.Quantum.Canon.csproj", "{2A1B0D78-2CE6-44DB-BFA1-3575F3C36321}"
EndProject
Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") = "Documentation", "Documentation", "{561759D2-4D2D-4EE3-9565-
9AAEC4A7D64B}"
        ProjectSection(SolutionItems) = preProject,             README.md = README.md     EndProjectSection
EndProject
Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") = "Samples", "Samples", "{614D2908-AAF9-48CE-B688-C28CC99C1D19}"
        ProjectSection(SolutionItems) = preProject              Samples\README.md = Samples\README.md     EndProjectSection
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "TeleportationSample", "Samples\Teleportation\TeleportationSample.csproj",
"{B88BE160-367F-43F1-BB56-8E21972D43D4}"
EndProject
Project("{2150E333-8FDC-42A3-9474-1A3956D46DE8}") = "Ising Model", "Ising Model", "{6E6063FB-43EB-41D1-B095-
4701A5D966BF}"
EndProject
Project("{9A19103F-16F7-4668-BE54-9A1E7A4F7556}") = "PhaseEstimationSample",
"Samples\PhaseEstimation\PhaseEstimationSample.csproj", "{F36C17E3-68EA-4ECF-828E-3FDEFFD8941D}"
EndProject
```

Bild 5.2 Ausschnitt aus QsharpLibraries.sln

5.4 Phase 4: Quantenprogramm starten

Das Ausführen des Quantenprogramms TeleportationSample führt zu wenigen Änderungen im Dateisystem. Neben den erzeugten Prefetch-Dateien Windows/Prefetch/RUNTIMEBROKER.EXE-D5B1A02A.pf und Windows/Prefetch/RUNTIMEBROKER.EXE-D5B1A02A.pf (siehe auch **Unterabschnitt 5.6**) werden neue Verzeichnisse und Dateien nur im Benutzerverzeichnis erzeugt (siehe **Tabelle 10**). Die Einträge in den SettingsLogs (Spur 1) zeigen nur an, dass Programme mittels Visual Studio ausgeführt wurden, geben aber keine Hinweise darauf, welches Programm ausgeführt wurde. Eindeutig dagegen sind die Einträge im Repository-Verzeichnis (Spuren 3-6), die den Programmnamen integriert haben. Die Spur 6 kann im Editor angezeigt werden und enthält Teile des Quellcodes des Programmes Teleportation sowie die Funktionsaufrufe.

Tabelle 10 Charakteristische Spuren für die Phase 4

Nr.	Spur
1	Users/MT/AppData/Local/Microsoft/VisualStudio/SettingsLogs/2018-08-04-16.48.07.409-devenv-3768.tsv
2	Users/MT/AppData/Local/Microsoft/VisualStudio/15.0_2063b37f/1031/ResourceCache.dll
3	Users/MT/source/repos/Quantum/Microsoft.Quantum.Canon/obj/Debug/netstandard2.0/Microsoft.Quantum.Canon.dll
4	Users/MT/source/repos/Quantum/.vs/QsharpLibraries/v15/.suo
5	Users/MT/source/repos/Quantum/Samples/Teleportation/obj/Debug/netcoreapp2.0/TeleportationSample.csproj.CopyComplete
6	Users/MT/source/repos/Quantum/Samples/Teleportation/obj/qsharp/src/TeleportationSample.g.cs
7	Windows/Prefetch/VBCSCOMPILER.EXE-176EEC9F.pf
8	Windows/Prefetch/RUNTIMEBROKER.EXE-D5B1A02A.pf

5.5 Phase 5: Deinstallation

Das Betriebssystem deinstalliert in dieser Phase die Anwendungen vom System, weshalb hauptsächlich Löschoperationen durchgeführt werden. Die Auswertung des Differenzbildes der Snapshots 4 und 5 zeigt, dass keine neuen Dateien und Verzeichnisse und somit keine charakteristischen Spuren erzeugt werden.

Tabelle 11 Charakteristische Spuren für die Phase Deinstallation

Nr.	Spur
1	Program Files (x86)/IIS Express/Microsoft.VisualStudio.Utilities.Internal.Net35.dll
2	Program Files (x86)/Microsoft SDKs/F#/10.1/Framework/v4.0/de/FSharp.Build.resources.dll
3	Program Files (x86)/Microsoft Visual Studio/Shared/Packages/Microsoft.Data.Edm.5.6.0/lib/sl4/Microsoft.Data.Edm.SL.dll
4	Program Files (x86)/ReferenceAssemblies/Microsoft/Framework/.NETPortable/v5.0/SupportedFrameworks/ASP.NET Core 1.0.xml
5	Program Files/dotnet/swidtag/Microsoft .NET Core SDK - 2.1.201 (x64).swidtag
6	Windows/Prefetch/DOTNET-SDK-2.1.201-WIN-X64.EX-15AA0757.pf
7	Windows/Prefetch/VS_COMMUNITY__1527698340.1530-49018498.pf
8	Windows/Prefetch/VS_INSTALLER.WINDOWS.EXE-B0A22389.pf
9	Windows/Prefetch/VSINITIALIZER.EXE-97E7C497.pf

Aus forensischer Sicht von Interesse bei der Deinstallation sind Dateien, die beim Installieren und Anwenden der Programme entstehen, aber nicht durch die Deinstallation entfernt werden. Solche persistenten Spuren können als Nachweis dienen, dass die Programme auf dem Betriebssystem des zu untersuchenden Rechners installiert waren. Diese Spuren werden durch

das Differenzbild der Snapshots 0 und 5 erzeugt. Die in **Tabelle 6** des **Unterabschnitts 5.1** genannten Dateien werden bei der Deinstallation ausnahmslos entfernt. Robuste und technisch unvermeidbare Spuren wie in **Abschnitt 4.4** beschrieben sind die Prefetch-Dateien. Die **Tabelle 11** führt Spuren auf, die eine Installation von Visual Studio und .NET Core SDK auch nach der Deinstallation hinterlassen.

Die **Tabelle 12** listet Spuren auf, die auch nach der Deinstallation vorhanden sind und vom QDK erzeugt wurden. Diese Spurenmenge stammt aus dem Differenzbild der Snapshots 1 und 5. Es bleiben der Ordner *Users/MT/source/repos*/*Quantum* ebenso wie das Verzeichnis *Users/MT/.nuget/packages/microsoft.quantum.development.kit* erhalten. Der Ordner *Users/MT/source/repos/* wird bei Deinstallation nicht entfernt, da dieser Ordner vom Benutzer gespeicherte Projekte enthalten könnte. Aufgrund der einfachen Löschung des Ordners handelt es sich um eine technisch vermeidbare und nicht-robuste Spur. Die Download-Datei sowie die Prefetch-Datei bleiben ebenfalls vorhanden.

Tabelle 12 Charakteristische Spuren für die frühere Verwendung vom QDK

Nr.	Spur
1	Users/MT/.nuget/packages/libuv/1.9.0/runtimes/win7-x64/native/libuv.dll
2	Users/MT/.nuget/packages/microsoft.quantum.development.kit/0.2.1806.1503-preview/build/ QSharp.xaml
3	Users/MT/.nuget/packages/microsoft.quantum.development.kit/0.2.1806.1503-preview/ tools/qsc/ FSharp.Core.dll
4	Users/MT/AppData/Local/Microsoft/VisualStudio/15.0_2063b37f/TextMateCache/Qsharp.language.cache
5	Users/MT/Downloads/QsharpVSIX.vsix
6	Users/MT/source/repos/Quantum/Interoperability/python/qsharp
7	Users/MT/source/repos/Quantum/QsharpLibraries.sln
8	Users/MT/source/repos/Quantum/Samples/Teleportation/Program.cs

5.6 Spuren im Prefetch-Verzeichnis

Das Betriebssystem Windows nutzt die Prefetch-Funktion und erzeugt Systemdateien, die sogenannten Prefetch-Dateien, zur Startoptimierung von Anwendungen. Dabei werden häufig genutzte Dateien und Bibliotheken bereits beim Systemstart in den Arbeitsspeicher[8] geladen und ermöglichen somit ein schnelleres Starten. Gleichzeitig können diese Systemdateien forensisch verwendet werden, denn sie stellen einen Indikator für die jetzige oder ehemalige Existenz von ausführbaren Dateien auf einem System dar. Diese Prefetch-Dateien sind als robuste Spuren zu klassifizieren, da Standard-Anwender selten von solchen Dateien Kenntnis

[8] Internetquelle: http://www.pctipp.ch/tipps-tricks/kummerkasten/windows-xp/artikel/wozu-ist-der-prefetch-ordner-da-33023/, abgerufen am 20.12.2017

haben[9] und das Verzeichnis standardmäßig im Explorer ausgeblendet ist. Beim Durchlaufen der verschiedenen Phasen werden aufgrund des Aufrufs von ausführbaren Dateien mehrere Prefetch-Dateien erzeugt und im Verzeichnis *c:\Windows\Prefetch* abgelegt. Die Dateien haben nach dem letzten Bindestrich einen achtstelligen Hashwert, der aus dem Pfad, von dem aus das Programm gestartet wurde, gebildet wird (siehe **Bild 5.3**). Die Dateiendung von Prefetch-Dateien ist *„.pf“*.

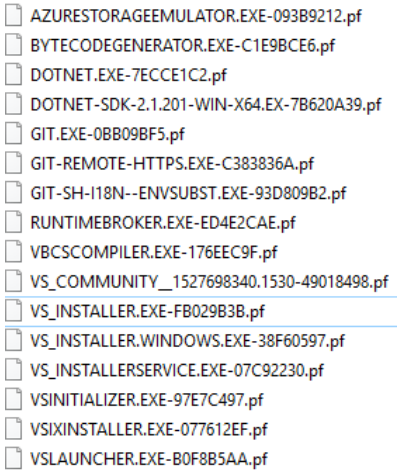
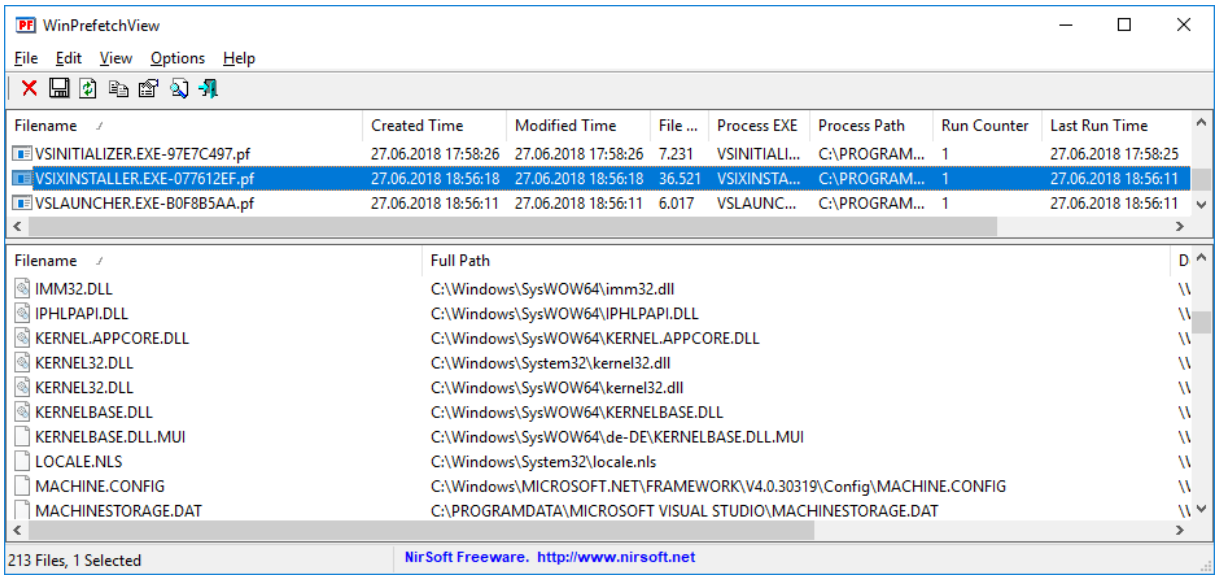


Bild 5.3 Inhalt des Prefetch-Verzeichnisses

Die Prefetch-Dateien können nicht mit dem Editor Notepad angezeigt werden. Ein forensisch sinnvolles Programm zur Auswertung dieser Dateien ist WinPrefetchView. Es zeigt die Anzahl der Programmaufrufe (Run Counter) und die Zeitstempel der Aufrufe (Last Run Time) an (siehe **Bild 5.4**). Damit kann ausgesagt werden, wann das Programm installiert (Created Time) und wann und wie häufig es ausgeführt wurde.



[9] Simon Jansen: Analyse der Spurenmenge der Anwendung OpenVPN Client Version 2.3.9 in Microsoft Windows. Technischer Bericht Nr. 1 vom 15.2.2016.

Bild 5.4 Prefetch-Datei VSIXINSTALLER.EXE.pf im WinPrefetchView

Die **Tabelle 13** führt die neu erstellten Prefetch-Dateien für jede Phase auf. Dabei werden die Prefetch-Dateien jedes Snapshots auf Änderungen untersucht. Maßgebend ist dabei das Ausgangs-Snapshot 0 mit insgesamt 124 Prefetch-Dateien. Keine Spuren stellen Prefetch-Dateien dar, die durch Hintergrundprozesse des Betriebssystems erzeugt und somit nicht durch manuell ausgelöste Aktionen verursacht wurden.

Tabelle 13 Erzeugte Prefetch-Dateien ohne Angabe der Kennziffern

Phase	**Neu erzeugte Prefetch-Datei**
1 Visual Studio und .NET Core SDK installieren	AZURESTORAGEEMULATOR.EXE.pf DOTNET.EXE.pf DOTNET-SDK-2.1.201-WIN-X64.EX.pf MICROSOFTAZURECOMPUTEEMULATOR.pf VS_COMMUNITY.pf VS_INSTALLER.WINDOWS.EXE.pf VS_INSTALLERSHELL.EXE.pf
2 QDK installieren	VSHIVESTUB.EXE.pf VSINIXINSTALLER.EXE.pf VSLAUNCHER.EXE.pf
3 GitHub	DATAEXCHANGEHOST.EXE.pf GIT.EXE.pf GIT-REMOTE-HTTPS-EXE.pf GIT-SH-I18N.pf
4 Quantenprogramm starten	DOTNET.EXE.pf MSBUILD.EXE.pf MSVSMON.EXE.pf VBSCOMPILER.EXE.pf
5 Deinstallation	DOTNET-SDK-2.1.201-WIN-X64.EX.pf VS_INSTALLER.EXE.pf VS_INSTALLER.WINDOWS.EXE.pf VS_INSTALLERSERVICE.EXE.pf

Die Phase 1 erzeugt zusätzliche 74 Prefetch-Dateien, wobei der Großteil während der lang andauernden Installationsphase vom Betriebssystem erzeugt wird und keinen Bezug zu den manuell ausgelösten Aktionen hat. Spuren für die Installation von Visual Studio sind die Dateien, die mit VS beginnen, wie VS_COMMUNITY.EXE.pf und VS_INSTALLER_WINDOWS.EXE.pf. Die Dateien DOTNET.EXE und DOTNET-SDK-2.1.201-WIN-X76.EX.pf werden durch die Installation von .NET Core SDK erzeugt. Anhand des letzten MAC-Zeitpunktes kann auf den wahrscheinlichen Zeitpunkt der Installation geschlossen werden.

Die Phase 2 erzeugt nur vier zusätzliche Prefetch-Dateien, wobei drei durch das QDK erzeugt wurden. Diese sind VSHIVESTUB.EXE.pf, VSIXINSTALLER.EXE.pf und VSLAUNCHER.EXE.pf. Die Phase 3 erzeugt vier Spuren im Prefetch-Verzeichnis: DATAEXCHANGE.EXE.pf, GIT.EXE.pf, GIT-REMOTE-HTTPS-EXE.pf, GIT-SH-I18N.pf. Das erstmalige Ausführen eines Quantenprogrammes in der Phase 4 erzeugt u.a. die Datei VBSCOMPILER.EXE.pf. Ein wiederholtes Ausführen desselben Programmes führt zu keinen

weiteren Prefetch-Dateien, aber einer Erhöhung des Run Counters von DOTNET.EXE.pf, MSBUILD.EXE.pf und MSVSMON.EXE.pf. Das Vorhandensein dieser Dateien ist somit eine Spur für die Verwendung von Q# und der letzte MAC-Zeitpunkt zeigt die letztmalige Verwendung des Programmes. Die Deinstallations-Phase 5 entfernt keine Dateien aus dem Prefetch-Ordner, womit diese Prefetch-Dateien robuste Spuren auf eine Verwendung von Q# sind. Die Deinstallationsroutine erzeugt die Prefetch-Dateien VS_INSTALLER.EXE und inkrementiert den Run Counter bei den Prefetch-Dateien DOTNET-SDK-2.1.201-WIN-X64.EX.pf, VS_INSTALLER.WINDOWS.EXE und VS_INSTALLERSERVICE.EXE.pf.

5.7 Übersicht der Spuren im Dateisystem

Die Übersicht in **Tabelle 14** zeigt, ob zuordenbare Spuren in den jeweiligen Spurenmengen der Phasen von idifference2 enthalten sind. Dabei wurden für ein strukturiertes Vorgehen die links gelisteten Strings in den Spurenmengen gesucht.

Tabelle 14 Übersicht von gefundenen Spuren im Dateisystem

Phase	**1 VS und .NET**	**2 QDK**	**3 GitHub**	**4 Q#-Programm**	**5 Deinstall**
Visual Studio, VisualStudio	Ja	Ja	Ja	Ja	Ja
.Net Core SDK, dotnet, .NETCore	Ja	Nein	Ja	Nein	Ja
F#, Fsharp	Ja	Nein	Ja	Nein	Ja
Quantum, Q#, Qsharp, VSIX	Nein	Ja	Ja	Ja	Ja
Github, GIT.exe	Nein	Nein	Ja	Nein	Ja
Teleportation	Nein	Nein	Ja	Ja	Ja
Prefetch	Ja	Ja	Ja	Ja	Ja

6 Fazit und Ausblick

Ein Quantencomputer ist ein Computer, dessen Funktionsweise auf den Gesetzen der Quantenmechanik beruht. Quantencomputer beinhaltet einen klassischen Rechner, der das quantenmechanische Experiment an den Qubits steuert und das Quantenprogramm beinhaltet. Somit hinterlässt auch ein Quantencomputer digitale Spuren in den Speichern des Rechners.

Diese Arbeit hat gezeigt, dass jeder Quantencomputer auch digitale Spuren erzeugt, da Teile eines Quantencomputers immer auch herkömmliche Speichersysteme sind. Die Quantenprogrammierung erfolgt unter Zuhilfenahme von Entwicklungsplattformen und –umgebungen, die digitale Spuren erzeugen.

Die untersuchte Quantenprogrammiersprache Q# ist als .NET Sprache umgesetzt und erzeugt persistente Spuren im Dateisystem, in dem Windows-Prefetch-Ordner und in der Registry. Somit können im Rahmen einer Datenträgerforensik-Analyse Rückschlüsse auf die Verwendung von Q# und eine frühere Installation gezogen werden.

Ein Großteil der persistenten Spuren der Installationen der notwendigen Programme im Dateisystem wird durch eine Deinstallation entfernt. Das Vorhandensein bestimmter Verzeichnisse und Dateien, die auch nach der Deinstallation bestehen bleiben, liefert Hinweise auf eine frühere Installation. Dadurch kann auch eine frühere Installation des Quantum Development Kits und das Laden einer speziellen Quanten-Bibliothek von GitHub nachgewiesen werden.

Die vom Betriebssystem Windows 10 erzeugten Prefetch-Dateien mit Bezug zu Q# sind robuste Spuren, die Installationen und auch die Ausführung nachweisen. Durch das Programm WinPrefetchView lassen sich somit die wahrscheinlichen Installationszeitpunkte und die Häufigkeit der Ausführungen ermitteln.

Diese Arbeit zeigt ein forensisches Vorgehen bei der Spurenakquise und Analyse der Quantenprogrammiersprache Q#. Es bleibt abzuwarten, welche Realisierungen von Quantencomputern Erfolg haben wird. Ein wichtiger Teil davon wird eine effiziente und benutzerfreundliche Quantenprogrammiersprache sein. Kristallisieren sich bestimmte Quantensprachen heraus, sollten diese forensisch untersucht werden, um im Fall von Straftaten unter Zuhilfenahme von Quantencomputern aktiv werden zu können.

.

7 Literaturverzeichnis

[Bsi16] Bundesamt für Sicherheit in der Informationstechnik: Die Lage der IT-Sicherheit in Deutschland 2016, Verlagshaus Zarbock Frankfurt am Main, Art.-Nr. BSI-LB16/505, Oktober 2016

[Btc16] Quantencomputer eine Gefahr für Bitcoin, veröffentlich am 29.03.2016, Internetquelle: https://www.btcgermany.de/quantencomputer-eine-gefahr-fuer-bitcoin/ (Abruf am 21.06.2018)

[Gar12] Garfinkel, S.: Digital forensics XML and the DFXML toolset, Digital Investigation 8, Seiten 161-174, 2012; Internetquelle: https://simson.net/clips/academic/2012.DI.dfxml.pdf (Abruf am 06.08.2018)

[Gol17] Wunderlich-Pfeiffer, Frank: Quantencomputer werden heute schon programmiert, veröffentlicht am 07.08.2017 Internetquelle: https://www.golem.de/news/simulatoren-quantencomputer-werden-heute-schon-programmiert-1708- 129205.html (Abruf am 05.05.2018)

[Jod14] Jodoin, Eric: Straddling the Next Frontier Part 2: How Quantum Computing has already begun impacting the Cyber Security landscape, SANS Institute, veröffentlicht 09.08.2014

[Mic17] Microsoft: Welcome to the Microsoft Quantum Development Kit Preview, veröffentlicht am 09.10.2017 Internetquelle: https://docs.microsoft.com/en-us/quantum/?view=qsharp-preview (Abruf am 05.05.2018)

[Mic17a] Microsoft: What is Quantum Computing, veröffentlicht am 11.12.2017 Internetquelle: https://docs.microsoft.com/de-de/quantum/quantum-concepts-1-intro?view=qsharp-preview (Abruf am 05.06.2018)

[Mic17b] Microsoft: The Q# Programming Language, veröffentlicht am 11.12.2017 Internetquelle: https://docs.microsoft.com/de-de/quantum/quantum-qr-intro?view=qsharp-preview (Abruf am 05.08.2018)

[Ove11] Overill, Richard: Digital Quantum Forensics: Challenges and Responses, veröffentlich in Future Information Technology, 6th International Conference FutureTech 2011, ISBN 978-3-642-22308-2, Springer-Verlag, 2011

[Par17] Parbel, Matthias: Q#: Microsofts Development Kit für Quantencomputing mit eigener Programmiersprache und Simulator, veröffentlicht am 12.12.2017 Internetquelle: https://www.heise.de/developer/meldung/Q-Microsofts-Development-Kit-fuer-Quantencomputing-mit-eigener-Programmiersprache-und-Simulator-3915895.html (Abruf am 08.05.2018)

[Pho17] Photonik Fachzeitschrift für Optische Technologien, ISSN1432-9778, Seite 12, 05/2017

[Seb17] Sebayang, A.: Microsoft veröffentlicht Entwicklungskit für Quantenrechner, 2017 Internetquelle: https://www.golem.de/news/q-und-qdk-microsoft-veroeffentlicht-entwicklungskit-fuer-quantenrechner-1712-131613.html (Abruf am 31.07.2018)

[Sof08] Sofge, D.: A Survey of Quantum Programming Languages: History, Methods, and Tools, Proceedings of the Second International Conference an Quantum, Nano, and Micro Technologies (ICQNM 2008), D01: 10.1109/ICQNM.2008.15, 2008, Seite 66-71m IEEE Computer Society, pp. 66-71, 2008, Internetquelle: https://pdfs.semanticscholar.org/ad58/9535382ac00d91e553d356618c312584 9603.pdf (Abruf am 14.06.2018)

[Spe12] Spektrum der Wissenschaft: Quanteninformation -Teleportation - Kryptografie - Quantencomputer (Spektrum Highlights) Broschiert, ISSN: 0947-7934, 1/2012, 28. Februar 2012