Durch eine Blockchain und ein Peer-to-Peer-System gesicherte Online-Ausleihe

Gruppe:

Alexander Reimer und Matteo Friedrich

Gymnasium Eversten Oldenburg

Betreuer: Herr Dr. Glade & Herr Husemeyer

Inhaltsverzeichnis

1 Kurzfassung

Wir haben versucht, eine Blockchain und ein Peer-to-Peer-Netzwerk zu programmieren, um damit eine sichere Online-Ausleihe für unsere Schulbibliothek zu erschaffen. Das sind neuartige Techniken, mit denen man Daten sicher zwischen verschiedenen Computern synchronisieren kann. Mit sicher ist gemeint, dass diese Daten „unhackbar“, also nachträglich unveränderbar sind. Das wird dadurch gewährleistet, dass das Netzwerk der Computer dezentral ist, es also keinen kontrollierenden Zentralcomputer gibt. Dadurch entscheidet immer die Mehrheit über die Richtigkeit der Daten. Dann kann ein Hacker nämlich nicht einfach über den zentralen Server alle Dateien verändern, sondern muss jeden einzelnen, oder zumindest die Mehrheit der Computer hacken. Ab einer gewissen Menge an teilnehmenden Computern wird dadurch ein extrem sicheres System erschaffen. Dieses wird Peer-to-Peer-Netzwerk genannt.

Um nicht alle Daten synchronisieren zu müssen, wird eine Prüfsumme aus den Daten berechnet, auch *Hash* genannt. Dieser Hash wird jedoch nicht einfach aus einem Datenblock berechnet, sondern aus allen. Dabei werden diese aufeinander aufbauend berechnet. D.h., für jeden Hash eines Datenblockes wird der Hash des vorherigen Datenblocks zur Berechnung genutzt. Wenn sich nun ein Eintrag im irgendeinem Datenblock ändert, ändert sich dadurch auch der dazugehörige Hash und durch die Abhängigkeit der Hashes voneinander auch alle darauffolgenden Hashes bis zum letzten. Deshalb muss man nur den letzten Hash mit den anderen Computern im Peer-to-Peer-System teilen - er ist der einzige nötige Wert zur Überprüfung der Korrektheit aller Datenblöcke.
Um die oben genannten Ziele zu erfüllen, haben wir ein Programm geschrieben, welches eine vorgegebene Datenliste (Blockchain) in einen finalen Hash umwandelt und ihn dann über ein Peer-to-Peer-Netzwerk mit allen anderen Computern vergleicht. Für den Austausch von Daten nutzen wir im momentanen Programm OneDrive, einen Cloud-Dienst von Microsoft. Dadurch ist das Programm nur so sicher wie OneDrive und der genutzte Microsoft Account. Um dies zu verbessern, wollen wir in Zukunft das schulinterne IServ-Netzwerk benutzen, dass alle Schulrechner miteinander verbindet. Für das Projekt haben wir einen Computer und einige Programmiersprachen (zuletzt „Julia“) benutzt.
Leider haben wir es bis zum Abgabezeitpunkt nicht geschafft, eine Funktion zum Hinzufügen neuer Blöcke fertig zu stellen. Hierbei liegt die Schwierigkeit darin, den neuen Block immer bei allen Computern gleichzeitig einzufügen, da sich durch einen neuen Block der Hash verändert. Bei einer Verzögerung haben dann nicht alle Computer den gleichen Hash und so werden einige fälschlicherweise gebannt. Allerdings haben wir uns bereits überlegt, wie wir dieses Problem lösen können über eine zusätzliche Datei, in der Benutzer Vorschläge für neue Blöcke eintragen können.

2 Einleitung

Das Ziel dieses Projektes war es, ein sicheres Online-Ausleihe-System für unsere Schulbücherei zu programmieren. Dazu haben wir eine Blockchain und das Peer-to-Peer verwendet, damit die Ausleihe sicher und dezentral organisiert ist.
Eine Blockchain und das Peer-to-Peer-System sind neuartige Technologien, durch welche es möglich ist, Informationen in einer öffentlich einsehbaren Datenbank zu speichern, zu verarbeiten, zu teilen und zu verwalten, wobei sie trotzdem „unhackbar“, also nachträglich unveränderlich bleiben [3, 6].

2.1 Das Peer-to-Peer-System

Die Sicherheit wird dadurch gewährleistet, dass das gesamte Synchronisationsnetzwerk dezentral ist, es also keinen alles kontrollierenden Zentralcomputer gibt. Das bedeutet, dass jeder Computer im Netzwerk eine Stimme hat und beim Datenüberprüfen für „Ja“ stimmt, wenn er die gleichen Daten wie der zu überprüfende Computer hat und für „Nein“, wenn nicht [6].
Daraufhin werden die Stimmen ausgezählt und dementsprechend alle gebannt, die mit der Mehrheit nicht übereinstimmen. Also entscheidet immer die Mehrheit über die Richtigkeit der Daten. Dadurch kann ein Hacker sich nämlich nicht einfach in den zentralen Server hacken und von dort aus alle Dateien verändern, sondern muss jeden einzelnen, oder zumindest die Mehrheit der Computer/Server hacken. Ab einer gewissen Menge an teilnehmenden Computern wird dadurch ein extrem sicheres System erschaffen. Dieses wird Peer-to-Peer-Netzwerk genannt. Doch vor allem kleine Systeme sind anfällig für sogenannte 51%-Angriffe. Bei diesen Hackangriffen hackt ein Hacker 51%, also die Mehrheit der Computer und umgeht so den Schutz des Peer-to-Peers. Bitcoin, das auch mit dieser Technik arbeitet, hätte so einen Angriff 2014 erleben können [7], bei Bitcoin Gold ist es 2018 tatsächlich passiert und hat bereits Millionen von Dollar an Schaden angerichtet [8].

2.2 Die Blockchain

Jeder einzelne Datensatz (Ausleiher, Ausleihdatum, Buchnummer, …) wird *Block* genannt. In einer kontinuierlichen Liste von Blöcken werden diese Blöcke durch eine Umwandlung zu Hashes verkettet. Einen Hash (ein Kettenglied einer Blockchain) erstellt man mit einer nicht zurück verfolgbaren Rechenvorschrift, auch Falltürformel genannt, wobei wichtig ist, dass jeder Block sich auf seinen Vorgänger bezieht. Denn sobald sich irgendwo in der Blockchain ein Block etwas ändert, meist verursacht durch Hacker, die die Daten zu ihren Gunsten beeinflussen wollen, verändert sich der dazugehörige Hash. Durch die Verbindung jedes Hashes zu allen Vorgängern wird so eine Kettenreaktion ausgelöst, die alle folgenden Hashes bis zum letzten verändert. Das sorgt dafür, dass wir nur den letzten Hash und nicht eine ganze Kette vergleichen müssen, da sich alle Veränderungen in ihm widerspiegeln. Dieses gesamte System wird *Blockchain* genannt [3].

Natürlich gibt es dabei auch eine Schwäche: Wenn man z.B. 1000 mögliche Werte in nur 100 möglichen Werten, also z.B. statt in 4 in 3 Stellen darstellen möchte, gibt es für jeden Hash 9 andere Zustände, in die der Block verändert werden kann, ohne das es durch den Hash bemerkt wird. Aber ein Hacker müsste diese natürlich erst herausfinden und wäre dadurch in seinen Möglichkeiten auch extrem eingeschränkt.

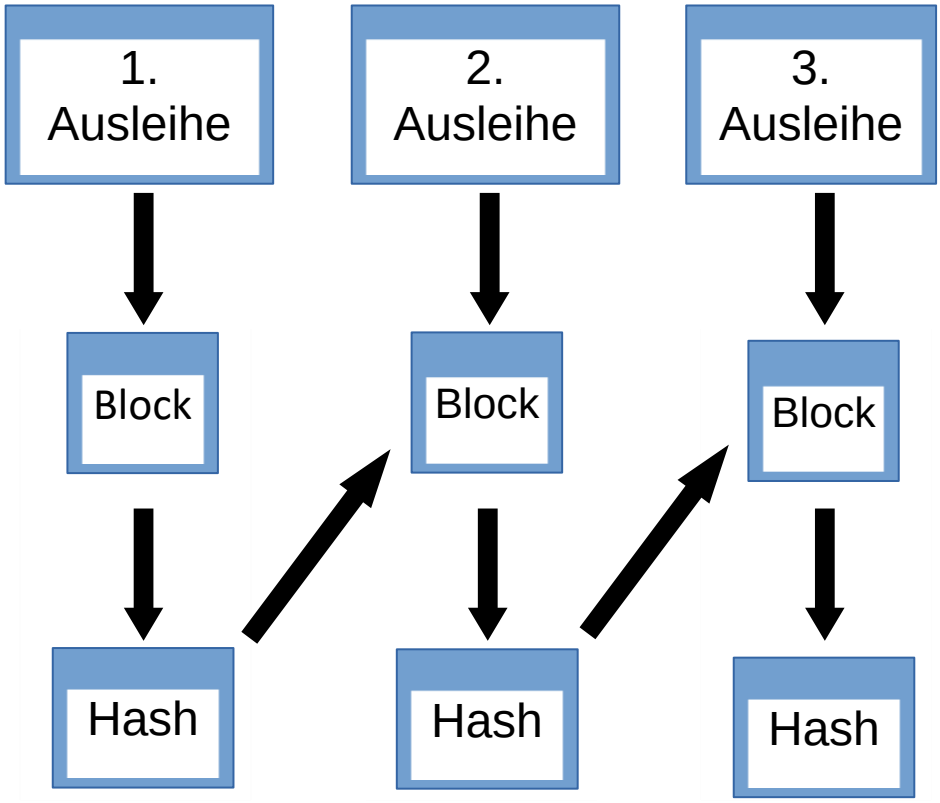

Abbildung 1: Schematische Darstellung einer Blockchain

2.3 Das Ziel

Unser erstes Ziel war es, eine kaum hackbare Blockchain zu programmieren. Dabei stießen wir auf einige Probleme und Schwierigkeiten, wie z.B. bei der Vernetzung von mehreren einzelnen Rechnern. Anschließend haben wir unsere bisher nur theoretisch funktionierende Blockchain auf das Online-Ausleihe System der Bibliothek zugeschnitten und hoffen natürlich, dass unsere Schulbücherei dieses auch nutzen wird, sobald es beendet ist.

3 Methode und Vorgehensweise

3.1 Materialien

- Software
  - Julia (eine Programmiersprache, die vor allem für numerisches und wissenschaftliches Rechnen entwickelt wurde) [2]
  - Atom (Editor für verschiedene Programmiersprachen - in unserem Fall Julia)
  - Firefox(Webbrowser)
- Hardware
  - Laptop DESKTOP-3VIJI2B von Lenovo

3.2 Vorgehensweise

Für dieses Projekt haben wir die Programmiersprache „Julia“ verwendet. Um dem folgenden Bericht besser zu folgen, ist es hilfreich, sich eine Erklärung der Grundbefehle durchzulesen (Dokumentation der Befehle) [1]. Trotzdem werden wir den Code auch für Leute, die keine Programmiererfahrung haben, erklären.

3.2.1 Programmierung eines Blocks

Zuerst haben wir überlegt, wie wir einen einzelnen Block erstellen könnten. Die Lösung hierfür ist ein „*mutable Struct*“. Das ist ein einzelner Datentyp, der mehrere vorgegebene Daten auf einmal speichern kann.

Mutable Struct Beispiel	
mutable struct Block	*#definiert einen „test“ mit den Werten (Buch, Benutzer, etc.)*
FirstVar :: String	*#hier werden den einzelnen Werten Datentypen zugeordnet*
date::Int	
names::Array	
end	
b = Block(„MyVar“, "31122007", Array([„Peter“, Fritz“, „Malte“]))	*#hier wird ein „test“ mit den eingegebenen Werten erzeugt*
println(b)	*#die Variable b, die jetzt vom Datentyp „test“ ist, wird ausgegeben*
julia> Block(6, "Matteo", 21112019, 69)	*#hier ist die Ausgabe von Julia*

3.2.2 Programmierung einer Kette von Blöcken

Eine Blockchain ist eine Kette aus Blöcken. Eine Kette aus vielen verschiedenen Daten wird in Julia als 1D Array beschrieben. Mit dem folgenden Code wird ein Array erzeugt, in den man einen *mutable Struct* hinzufügen kann:

```
Array = Array{Block}(undef, 0)
```

3.2.3 Programmierung des Hashcodes

Im Anschluss haben wir nach einem Hashcode gesucht. Zur Lösung dieses Problems fanden wir den *Fletcher Algorithmus* [4]. Dieser Algorithmus ist ein Kompromiss aus einem langsamen und komplizierten, dafür aber sicherem Algorithmus und einem schnellerem dafür aber fehleranfälligen Verfahren wie einer einfachen Prüfsumme. Bei dem Fletcher Algorithmus werden die Dezimalwerte der einzelnen Dateneinträge zusammenaddiert und je nach ihrer Position gewichtet. Dazu müssen natürlich zuerst alle Daten in ihren Dezimalwert umgewandelt werden. Sobald ein Abschnitt verändert wird, ändert sich auch das Ergebnis.

Der Fletcher Algorithmus funktioniert mit Hilfe eines Prüfwertes. Er besteht aus zwei einzelnen Summen (Sum1 und Sum2). Zuerst bekommt die Funktion einen Eingabewert, der in einzelne Abschnitte unterteilt wird. Diese werden dann, wenn nötig, in eine Dezimalzahl umgewandelt. Nun summiert sich Sum1 schrittweise alle Abschnitte durch. Sum2 ist die Summe aller Sum1 Werte. Mit der Modulofunktion kann man Sum1 und Sum2 auf einen bestimmten Wert abgrenzen.

Beispiel:

In diesem Beispiel ist der Eingabewert „Hallo“ und jeder Buchstabe ein einzelner Abschnitt. Der maximale Wert ist 255. Der Hash ist 124.

Tabelle 1: Hashberechnung

Nachricht	H	a	l	l	O	**Ergebnis**
dezimaler Wert	72	97	108	108	111	
sum1	72	169	277->22	130	241	241
sum2	72	241	263->8	138	379->124	**124**

3.2.4 Programmierung der Online Verknüpfung

Als nächstes mussten wir eine Verknüpfung für mehrere Endgeräte finden. Auch dieses Problem konnten wir lösen: Die Lösung war OneDrive. Auf OneDrive kann man nämlich Dateien Online in einer Cloud speichern und mit Julia einfach bearbeiten und auslesen.

Beispiel Dateiauslesung	
io = open(„Dateiname", read = true, write = true)	*#hier wird die Datei geöffnet (zuerst Name der Datei, dann mögliche Operatoren)*
write(io, „BlaBlaBla“)	*#jetzt wurde der Text „BlaBlaBla“ in die Datei geschrieben*
myVar = readline(io)	*#nun wurde die Datei gelesen und myVar auf den Inhalt gesetzt(„BlaBlaBla“)*
close(io)	*#die Datei wurde geschlossen*
println(myVar)	*#myVar wird ausgegeben*

4 Ergebnisse

Als nächstes haben wir die bis jetzt erklärten Funktionen auf eine Schulbücherei zugeschnitten.

4.1 Das System

Bei unserem System bekommt jeder User (Benutzer der Blockchain) eine eigene Datei, die nur er ändern, aber jeder lesen kann. Diese Datei beinhaltet den vom Computer des Users errechneten Hashcode. Wenn dieser Hashcode nicht mit dem von der Mehrheit übereinstimmt, wird die Datei mit dem falschen Hashcode gebannt und von allen anderen beim zukünftigen Vergleichen ignoriert.

4.2 Ein einzelner Block

Als erstes haben wir überlegt, welche Werte ein Block braucht und kamen zu dem Ergebnis, dass ein Block die Buchnummer des auszuleihenden Buchs, den Benutzer, das Ausleihedatum und den Hash des letzten Blocks braucht.

Block	
mutable struct Block	*#erzeugt den Block mit den Infos(Buch, Benutzer, etc.)*
buchnummer::Int	*#hier werden den einzelnen Werten Datentypen zugeordnet*
benutzer::String	
ausleiheDatum::Int	
vorHash::Int	
end	
b = Block(„00034“, "Peter", 31122019, 100)	*#hier wird ein Block mit den eingegeben Werten erzeugt*
println(b)	*#die Variable b, die jetzt vom Datentyp „Block“ ist, wird ausgegeben*
julia> Block(„13504“, "Peter", 31122019, 100)	

4.3 Der Hashcode

Nun mussten wir uns überlegen, wie wir den Fletcher Algorithmus für einen Block anwenden könnten. Wir kamen zu dem Entschluss, dass jeder Wert (Buchnummer, Benutzer, Ausleihedatum, vorHash) einen Abschnitt darstellt. Der maximale Wert haben wir auf 99 festgelegt.

Hash erstellen	
function mach_hash(b_vecAll, hashes)	*#macht einen Hash für jeden Block der „Liste“ Blockchain(b_vecAll)*
sum1 = 0	*#setzt beide Prüfsummen auf 0 zurück*
sum2 = 0	
for i = 1:length(b_vecAll)	*#geht in einer Schleife alle Blöcke der „Liste“ b_vecAll durch*
sum1 = mod(sum1 + b_vecAll[i].buch, 99)	*#führt den Fletcher Algorithmus bei jedem Block durch*
sum2 = mod(sum1 + sum2, 99)	
sum1 = mod(sum1 + str2int(b_vecAll[i].benutzer), 99)	
sum2 = mod(sum1 + sum2, 99)	
sum1 = mod(sum1 + b_vecAll[i].ausleiheDatum, 99)	
sum2 = mod(sum1 + sum2, 99)	
sum1 = mod(sum1 + b_vecAll[i].vorHash, 99)	
sum2 = mod(sum1 + sum2, 99)	
end	*#beendet die Schleife*
return sum2	*#speichert den letzten Hash(sum2)*
end	*#beendet die Funktion*

4.4 Das Überschreiben des alten Hashes

In dieser Funktion wird der alte Hash mit dem neuen, vorher bei „create_hash“ erstellten Hash, überschrieben, falls der Computer nicht schon vorher gebannt wurde.

Hash online stellen		
function writeHash(sum2)		
io = open("1.txt", read = true)	*#öffnet die eigene Datei mit den Daten*	
ß = readline(io)	*#liest die Daten aus der Datei aus*	
close(io)	*#schließt die eigene Datei mit den Daten*	
if ß == "banned"	*#wird ausgeführt, wenn die Datei gebannt wurde*	*#...wird der User darauf hingewiesen*
println("Die IP " * SysString * " wurde gebannt")		*#wahr wird ausgegeben*
return true		
else	*#wenn die Datei nicht gebannt wurde dann...*	
io = open("1.txt", write = true)	*#...öffnet die eigene Datei*	
write(io, sum2)	*#...schreibet den Hash(sum2) in die Datei*	
close(io)	*#...die Datei wird geschlossen*	
return false	*#...wird falsch ausgegeben*	
end	*#*	
end	*#beendet die Funktion*	

4.5 Beschaffung der anderen Hashes

In dieser Funktion werden die Hashes der anderen Computer ausgelesen und in „akt_hashes“ gespeichert.

Andere Hashes auslesen	
function getAkt_hashes(NichtÜberein)	
NichtÜberein = 0	*# Variable “NichtÜberein“ wird auf 0 gesetzt*
akt_hashes = zeros(Int, 0)	*#Erzeugt eine leere Liste aus Nullen namens „akt_hashes“(„aktuelle Hashes“)*
for i = 1:4	*#eine Schleife, die 4 mal, also für jeden Computer einmal, wiederholt wird*
Computer = string(i)	*#“Computer“ wird auf die Anzahl der Wiederholungen der Schleife gesetzt*
io = open(Computer* ".txt", read = true)	*#öffnet die Datei des Computer mit der Nummer „Computer“*
Inhalt = readline(io)	*#setzt „Inhalt“ auf den Inhalt (Hash) der Datei*
close(io)	*#schließt die Datei*
if Inhalt == "banned"	*#Wenn der Computer gebannt wurde, wird er ignoriert*
push!(akt_hashes, 0)	
NichtÜberein -= 1	
else	*#sonst wird der Inhalt, also der Hash des Computers, als neues Element in der Liste „akt_hashes“ gespeichert*
ss = 0	
s = Inhalt	
for i2 = 1:length(s)	
c = s[i2]	
ss = ss * 10	
ss += Int(c)	
end	
push!(akt_hashes, ss)	
end	
end	
return akt_hashes	*#akt_hashes wird gespeichert*

4.6 Der „Hacktest“

Diese Funktion testet, ob sum2, also der letzte Hash, der Mehrheit entspricht und gibt das Ergebnis als Boolean (Wahrheitswert) aus (true/false).

Der Hacktest		
function TestIfHacked(akt_hashes, Sys)		
for i = 1:length(akt_hashes)	*#für jedes Element von akt_hashes*	
if akt_hashes[Sys] == akt_hashes[i]	*#wird abgefragt, ob das Element von Akt_hashes dem eigenen Hash entspricht*	
Überein += 1	*#wenn ja, wird 1 zu „Überein“ addiert*	
else	*#sonst*	
NichtÜberein += 1	*#wird NichtÜberein um 1 aufsummiert*	
end	*#hier wird die Abfrage beendet*	
end		
if Überein > NichtÜberein	*#wird ausgeführt wenn Überein größer als Nichtüberein ist*	*#“false“ wird ausgegeben*
return false		
else	*#wird ausgeführt wenn Überein nicht größer als Nichtüberein ist*	*#“true“ wird ausgegeben*
return true		
end	*#hier wird die Abfrage beendet*	
end	*#hier wird die Funktion beendet*	

4.7 Neue Blöcke

Leider hatten wir keine Möglichkeit, das Einfügen von neuen Blöcken bis zum Abgabetag des Berichts fertig zu stellen. Das Problem dabei liegt daran, dass der neue Block nicht bei allen Rechnern gleichzeitig hinzugefügt werden würde (weil das Netzwerk ja dezentral ist und die Rechner nicht alle gleich schnell gehen oder Netzwerkzugang haben). Dadurch hätte der Rechner, an dem der neue Block hinzugefügt werden würde, einen neuen Hash und würde sofort gebannt werden, da sich der Hash aller anderen noch nicht geändert hätte.

Trotzdem möchten wir hier das System dieser Funktion, welches wir uns schon ausgedacht haben, vorstellen: Jeder User erhält eine zusätzliche Datei für das Hinzufügen neuer Blöcke. In diese Datei schreibt er Vorschläge für neue Blöcke, die er der Blockchain anhängen möchte. Alle anderen User suchen diese Dateien ab und übernehmen den Vorschlag eines Users, wenn sie mit diesem einverstanden sind. Sobald die Mehrheit der User (mehr als 50%) einen Block als Vorschlag in ihre Datei geschrieben hat muss er von allen Usern hinzugefügt werden. Rechner, die

sich kurzzeitig um diesen einen Block unterscheiden, werden nicht gebannt. Zudem darf man nur einen neuen Vorschlag machen, wenn alle Dateien leer sind, um Kollisionen zu vermeiden.

4.8 ID Register

Jeder User besitzt eine Liste, in der er allen anderen Usern des Peer-to-Peer-Netzwerks Werte zuordnet.

Eine 0 bedeutet, dass der User nicht gesondert behandelt wird.

Eine 1 bedeutet, dass der User zu langsam ist.

Eine 2 bedeutet, dass der User keinen Hash veröffentlicht hat.

Und eine 3 bedeutet, dass der User gebannt wurde.

Wenn der Wert, der für einen User abgespeichert wurde, zwischen 1 und 3 liegt, wird der User im weiteren Verlauf des Programms, also in der Funktion TestIfHacked ignoriert. Sobald ein User irgendwann wieder den „richtigen“ Hash veröffentlicht, wird sein Wert auf 0 gesetzt. Dies ist die Grundlage für das Hinzufügen neuer Blöcke, da ein User, der länger braucht, um einen Block hinzuzufügen, nicht gleich gebannt, sondern erst als „zu langsam“ eingestuft werden würde und so, nach dem Hinzufügen des neuen Blocks und der Aktualisierung des Hashes, einfach weiter machen könnte. Dasselbe gilt, wenn man als 2 eingestuft wurde. So können User, die offline gehen, ihren Hash löschen und leer lassen, ohne gebannt zu werden.

Diese Methode ersetzt das vorherige false/true System, bei dem ein User bei einem ungleichen Hash sofort und für immer gebannt wurde.

Dieses ID Register haben wir zwar bereits umgesetzt, jedoch wurde die Länge des Programms damit von ~200 auf ~300 Zeilen verlängert. Da wir zusätzlich Zeitmangel und außerdem ein 15 Seiten Limit hatten, wurde der Rest des Berichts nicht daran angepasst und die Umsetzung im Programm nicht erklärt.

4.9 Der Hauptteil

Dieser Teil des Programms wird beim Ausführen der Blockchain gestartet und verwendet alle oben beschriebenen Funktionen. Er wird wiederholt, bis der eigene Hash dem der Mehrheit widerspricht. In diesem Fall wird das Programm beendet.

Hauptteil		
const Sys = 1	*#die Konstante Sys wird auf die Systemnummer gesetzt*	
b_vecAll = Array{Block}(undef, 0)	#b_vecAll wird auf einen Array des Typs Block gesetzt und geleert	
sum2 = 100	#Sum2 wird am Anfang auf den Genesis Wert gesetzt (100)	
newBlock(Block, 01, "Tom", 051119, sum2, b_vecAll)	#neue Blöcke werden erstellt und in b_vecAll gespeichert	
newBlock(Block, 02, "Mark", 081119, sum2, b_vecAll)	#noch ein neuer Block	
newBlock(Block, 03, "Ute“, 081119, sum2, b_vecAll)	#noch ein neuer Block	
newBlock(Block, 04, "Ulf", 081119, sum2, b_vecAll)	#noch ein neuer Block	
newBlock(Block, 13, "Olaf", 101119, sum2, b_vecAll)	#noch ein neuer Block	
while true	#eine unendliche Schleife	
sum2 = mach_hash(b_vecAll, hashes)	#es wird ein neuer Hash von b_vecAll berechnet und als sum2 gespeichert	
writeHash(sum2, Sys)	#führt writeHash aus (s. o.)	
akt_hashes = getAkt_hashes()	#führt getAkt_hashes aus (s. o.)	
Gehackt = TestIfHacked(akt_hashes, Sys)	#führt TestIfHacked aus (s. o.) und setzt Gehackt auf den ausgegebenen Boolean (s.o.)	
if Gehackt == true	#wird ausgeführt, falls Gehackt true ist, also der Computer gehackt wurde	
SysString = string(Sys)		#schreibt banned in die Datei. Nun wird sie von den anderen ignoriert
io = open(SysString * ".txt", write=true)		
write(io, „banned“)		
close(io)		
break		#bricht das Programm ab
end		
end		

5 Diskussion

Wir haben unser Forscherziel nicht ganz erreicht, denn obwohl die Blockchain im Prinzip funktioniert, ist sie für Benutzer ohne Programmierkenntnisse noch nicht einfach anwendbar. Um die Blockchain zu starten, muss man nämlich erst auf jedem Rechner die Sprache Julia mit den nötigen Paketen und einen Editor (z.B. Atom) installieren. Um dies zu verbessern, arbeiten wir daran, eine Julia-Datei in ein einfach ausführbares Programm umzuwandeln. Dazu haben wir einen vielversprechenden Artikel gefunden. Außerdem wäre es sinnvoll, den Betrieb der Blockchain über eine graphische Oberfläche zu steuern. Auch dazu haben wir Julia Pakete gefunden, mit denen man das programmieren kann.

Um eine Verbindung zu den anderen Computern aufzubauen, müsste man sich außerdem überall mit einem Microsoft Konto bei OneDrive anmelden. Doch dafür gibt es zumindest eine halbwegs zufriedenstellende Antwort: Wir könnten das Schulinterne IServ-Netzwerk benutzen, um die Computer miteinander zu vernetzen. Dadurch wäre die Blockchain aber nur in der Schule funktionsfähig, also könnte man sich nichts von zuhause ausleihen.

Das nächste Problem ist die fehlende Möglichkeit, neue Blöcke in die Blockchain einzufügen, also ein neues Buch abzugeben/zu verlängern. Wir haben zwar eine mögliche Lösung für dieses Problem entwickelt (s. 4.7 - 4.8), sind jedoch noch nicht zur Umsetzung gekommen

6 Quellen

[1] Julia Dokumentation (https://docs.julialang.org/en/v1/)
[2] Julia als executable (https://github.com/dhoegh/BuildExecutable.jl)
[3] Blockchain (https://www.youtube.com/watch?v=4FU3tc-foaI)
[4] Hashcode (1) (https://de.wikipedia.org/wiki/Fletcher’s_Checksum)
[5] Hashcode (2) (https://www.youtube.com/watch?v=2zPoJHwRV30)
[6] Peer-to-Peer (https://de.wikipedia.org/wiki/Peer-to-Peer)
[7] 51% Gefahr für Bitcoin (https://www.handelsblatt.com/finanzen/maerkte/devisen-rohstoffe/virtuelle-waehrung-in-der-krise-bitcoin-leidet-an-hausgemachten-problemen/10057444.html)
[8] Erfolgreicher Angriff auf Bitcoin Gold (https://www.heise.de/select/ct/2018/14/1530921921642329)

Alle Quellen wurden zuletzt im Januar 2020 eingesehen.