

Erzeugung und Optimierung nichtdeterministischer Zufallszahlen für eine erhöhte Sicherheit in der verschlüsselten Telekommunikation

Robert J. Pietsch

Jugend Forscht (Bundesfinale 2019)

Abstract

In der heutigen Gesellschaft spielt die digitale Datenübertragung eine große Rolle. Egal, ob wir beim Online-Banking Geld überweisen oder im Chat mit anderen Personen private Nachrichten austauschen, ist eine sichere Datenübertragung wünschenswert. Um diese zu gewährleisten, werden die Daten verschlüsselt übertragen. Für die Verschlüsselungen werden Zufallszahlen benötigt, die ein Angreifer nicht erraten darf, da die Daten sonst entschlüsselt werden könnten. Die Zufallszahlen wurden in der Vergangenheit mit komplexen Algorithmen erzeugt, die aber nicht echt zufällig sind. Kennt man alle Parameter, so kann man die Zufallszahlen erraten und mit diesen die Verschlüsselung brechen. Um diesem Problem vorzubeugen, werden regelmäßig komplexere Zufallsalgorithmen entwickelt. Diese Entwicklung ist teuer und wird immer schwieriger.

In meinem Projekt habe ich einen nichtdeterministischen, also physikalisch echten Zufallsgenerator gebaut, der dieses Problem mit echt zufälligen Zufallszahlen lösen soll. Hierfür messe ich verschiedene physikalisch echt zufällige Phänomene wie radioaktive Zerfälle und akustisches Umgebungsrauschen und berechne daraus Zufallswerte. Der Zufallsgenerator ist als Peripheriegerät für Endanwender umgesetzt und kann per USB ausgelesen werden.

Inhaltsverzeichnis

1 Das Projekt	1
1.1 Die Zielsetzung	1
1.2 Der Weg zum Projekt	1
1.3 Physik und Zufall	2
1.3.1 Determinismus	2
1.3.2 Radioaktive Zerfälle	2
1.3.3 Mechanisches Chaospendel	2
1.3.4 Elektrisches Rauschen	2
1.3.5 Quanteneffekte am Halbspiegel	2
2 Konstruktion des Hardwaregeräts	3
2.1 Erste Umsetzungsidee	3
2.2 Testaufbau und Datensammlung	3
2.3 Umsetzungsideen des Generators bei verschiedenen Entropiequellen	4
2.3.1 Radioaktivität	4
2.3.2 Chaospendel	5
2.3.3 Akustisches und elektrisches Rauschen	5
2.3.4 Transmission am Halbspiegel	6
2.4 Prototyp des Gehäuses	6
3 Softwareseitige Implementierung	7
3.1 Messung und Umrechnung zu Zufallszahlen	7
3.1.1 Berechnung aus Zeitpunkten (Radioaktivität)	7
3.1.2 Berechnung aus absoluten Messwerten (Rauschen)	7
3.2 Client-Software	8
3.2.1 Treiber und Rarduino-Service	8
3.2.2 Übertragungsprotokoll	8
4 Bewertung der Zufallsgeneratoren	9
4.1 Aufstellung eines Bewertungsmodells	9
4.1.1 Zufälligkeit und Verteilung der Wertemenge	9
4.1.2 Geschwindigkeit der Zahlengenerierung	9
4.1.3 Sicherheit des Generators	9
4.1.4 Preis des Geräts	9
4.2 Anwendung des Bewertungsmodells	10
4.2.1 Radioaktive Zerfälle	10
4.2.2 Akustisches und elektrisches Rauschen (Generierung durch Hashing)	11
4.2.3 Akustisches und elektrisches Rauschen (Generierung als Bitfolge)	12
5 Zusammenfassung und Fazit	13
5.1 Ergebnis der Arbeit	13
5.2 Weiterer Ausblick	13
5.3 Danksagung	13
6 Anhang	14
6.1 Quellenverweise	14

1 Das Projekt

1.1 Die Zielsetzung

Immer, wenn man eine Internetseite über das verschlüsselte HTTPS-Protokoll aufruft, sollte man davon ausgehen können, dass die übertragenen Daten vor Mithörern zwischen dem besuchten Server und dem eigenen Endgerät (Man-in-the-middle-Angriff) geschützt ist. Dies wird von den gängigen Browsern durch ein grünes Schlosssymbol  in der Adresszeile signalisiert. Das HTTPS-Protokoll basiert genauso wie einige andere Datenübertragungsprotokolle (z.B. POP3S und IMAPS für Emails) auf Transport Layer Security (TLS; früher bekannt als SSL), welches als Sicherheitsebene alle Daten verschlüsselt. TLS arbeitet mit RSA, einer asymmetrischen Verschlüsselung.^[1] Dies bedeutet, dass jede Seite der Kommunikation zwei Schlüssel, nämlich einen privaten und einen öffentlichen Schlüssel generiert. Mithilfe des privaten Schlüssels ist man in der Lage, Daten, welche zuvor mit dem öffentlichen Schlüssel verschlüsselt wurden, wieder zu entschlüsseln. Beim Verbindungsaufbau wird der öffentliche Schlüssel an den Partner überendet, während der private Schlüssel nur im jeweiligen Gerät gespeichert bleibt. Somit ist ein Angreifer zwar in der Lage, zusätzliche Daten mit dem öffentlichen Schlüssel zu verschlüsseln, kann aber keine verschlüsselten Daten entschlüsseln oder verändern.^[2]

Zufallszahlen, und damit der Kern meines Projektes, kommen bei der Generierung der Schlüssel ins Spiel. Die Schlüsselpaare werden aus extrem großen zufälligen Primzahlen generiert. Diese Primzahlen bleiben geheim und werden nicht übertragen. Kennt ein Angreifer trotzdem diese Primzahlen, so ist er in der Lage, die Verschlüsselung zu brechen, indem er sich die Schlüsselpaare errechnet und mit den daraus hervorgehenden privaten Schlüsseln die Kommunikation entschlüsselt. Für eine sichere Datenübertragung sollten die Zufallszahlen also so zufällig und unerratbar wie möglich sein.^[3] Ein Computer arbeitet aber deterministisch, d.h. er nimmt eine Eingabe und wandelt diese in eine bestimmte Ausgabe um, die bei gleichbleibenden Parametern gleich bleibt. Um Zufallswerte zu generieren, sind also sich verändernde Parameter notwendig. Diese sind oft die Systemzeit, zeitliche Abstände zwischen Netzwerkpaketen oder Benutzereingaben mit Maus oder Tastatur (letzteres ist für Server unmöglich). Kennt man diese Parameter und den (Pseudo-)Zufallsalgorithmus, so lassen sich die Primzahlen und daraus folgend die Schlüssel berechnen. Dieses Problem wird aktuell umgangen, indem die Pseudozufallsalgorithmen regelmäßig erweitert und somit komplexer werden. Dies bedeutet eine höhere Prozessorauslastung beim Endnutzer, sowie eine teure und zeitintensive Suche nach neuen Methoden, um die Generatoren sicherer zu machen.^{[4] [5]}

In meinem Projekt möchte ich dieses Problem lösen, indem ich einen physikalisch echten, nichtdeterministischen Zufallsgenerator bauen und diesen schlussendlich in eine TLS-Bibliothek integrieren. Der Generator soll auf zufällig ablaufenden physikalischen Phänomenen basieren. Bei der Konstruktion möchte ich mehrere infrage kommende Messmethoden umsetzen, prüfen und ggf. optimieren, um am Ende ein möglich kostengünstiges und kompaktes Hardwaremodul zu schaffen, welches sowohl für große Serveranlagen (viele Zufallswerte in kurzer Zeit) als auch für Endanwender (beste Kosteneffizienz) geeignet ist und somit eine sicherere Verschlüsselung in beide Richtungen anbietet.

1.2 Der Weg zum Projekt

Da ich mich hobbymäßig viel mit der Programmierung von Internetanwendungen beschäftige, bin ich relativ schnell auf die Absicherung der von diesen Systemen genutzten Daten mit der Hilfe von kryptographischen Methoden gestoßen. Bei meiner Recherche über die Funktionsweise von verschiedenen Verschlüsselungen und verschlüsselten Kommunikationsprotokollen sind mir die unsicher generierten Zufallszahlen, welche eine zentrale Rolle spielen, aufgefallen. Da wir uns im Physikunterricht schon mit einem Einstieg in die Quantenphysik beschäftigt haben, kenne ich eine Reihe von nichtdeterministischen Entropiequellen. Mir ist zudem bewusst, dass es bereits nichtdeterministische Zufallsmodule für Computer gibt (z.B. <https://www.idquantique.com/random-number-generation/>), jedoch haben diese Geräte meist vier- bis fünfstellige Preise und sind deswegen nicht für den einfachen Heimanwender geeignet. Basierend auf diesen Rechercheergebnissen habe ich mir den Bau eines nichtdeterministischen Zufallsgenerators als Ziel gesetzt. Im Laufe der Projektarbeit bin ich auf weitere Fragen gestoßen, die ich im Rahmen dieses Projektes beantworten möchte. Zu diesen gehören unter anderem die Definition von Zufälligkeit und die Frage, welche Entropiequellen am „zufälligsten“ und am besten für Anwender geeignet ist.

1.3 Physik und Zufall

1.3.1 Determinismus

Unter Determinismus versteht man die Idee, dass jeder Prozess aus einer Ursache und einer zugehörigen Folge (*Wirkung*) besteht. Dies bedeutet, dass gleiche Prozesse unter gleichen Umständen immer die selben Ergebnisse produzieren. Nichtdeterministisch ist ein Prozess, bei dem gleiche Ursachen verschiedene Resultate produzieren können. Auf dieser Idee basieren auch große Teile der modernen Quantenphysik, die in vielen Berechnungen und Vorhersagen nur von Wahrscheinlichkeiten ausgeht. [6]

1.3.2 Radioaktive Zerfälle

Radioaktivität beschreibt die Veränderung (den „Zerfall“) von instabilen Atomkernen unter Ausstrahlung von ionisierender Strahlung. Beobachtet man einen radioaktiv strahlenden Körper, so hat man eine große Menge an zerfallenden Atomkernen. Der Zerfallszeitpunkt von diesen ist zufällig, jedoch besagt die sog. Halbwertszeit, wann statistisch die Hälfte der vorhandenen Kerne zerfallen sein sollte. Bestimmte Arten von radioaktiven Zerfällen sind mit einem Geiger-Müller-Zählrohr messbar. Das Zählrohr erkennt die ausgestrahlte ionisierende Strahlung und erzeugt mit dieser eine messbaren Spannungsspitze. [7] [8] [9]

1.3.3 Mechanisches Chaospendel

Das mechanische Doppelpendel, auch Chaospendel genannt, ist eine Konstruktion, bei der an einer Mittelachse ein Pendelarm angebracht ist, an dessen Ende eine weitere Achse mit einem Pendelarm angehängt ist. Wird nun der innere Arm beschleunigt oder gebremst, so beginnt das Pendel scheinbar willkürliche Bewegungen zu vollziehen. Die Bewegung ist chaotisch, was bedeutet, dass eine kleine Änderung am Eingangsparameter eine starke Änderung an der resultierenden Bewegung hervorruft. Diese resultierende Bewegung sind zwar in der Theorie berechenbar, jedoch gibt es in der Praxis so viele Faktoren, die auf die Bewegung einwirken, dass deren Berechnung so gut wie unmöglich ist. Zu den ungenau absehbaren Faktoren zählen zum Beispiel Reibung, Luftdruck und -zug, und Wärmeausdehnung. Da diese Konstruktion aber prinzipiell deterministisch ist, nutze ich dieses Phänomens in meinem Projekt nicht. [10]

1.3.4 Elektrisches Rauschen

Unter Rauschen versteht man in der Physik sich verändernde Abweichungen von zu erwartenden Messwerten, weswegen man Rauschen auch als Störgröße bezeichnet. Elektrisches Rauschen beschreibt die Fluktuation der Spannung und Stromstärke in einem Stromkreis, die auch bei gleichbleibenden Rahmenbedingungen auftreten. Die wichtigsten Ursachen für elektrisches Rauschen sind Schrotrauschen, thermisches Rauschen und die ungleichmäßige Transmission innerhalb von Halbleitern. Schrotrauschen basiert auf der Idee, dass Elektronen ihre Ladung unabhängig voneinander bewegen und die resultierende Stromstärke nur ein statistischer Wert aus diesen einzelnen sich bewegenden Ladungen ist. Thermisches Rauschen basiert auf der Tatsache, dass sich die meisten Widerstände je nach Temperatur in ihrem elektr. Bremsverhalten verändern. Stoßen Elektronen auf die Moleküle im Widerstand, so werden deren Atomrümpfe in Schwingung versetzt und der Widerstand wird warm. Ähnliche Effekte lassen sich auch in Halbleitern finden, die ihre Eigenschaften aufgrund der durchfließenden Ströme leicht verändern. [11] [12] Eine wichtige Eigenschaft des Rauschens ist die nichtdeterministische Zufälligkeit. Die einzelnen Effekte lassen sich nicht vorhersagen, da die Bewegung der Elektronen zufällig ist. [13]

1.3.5 Quanteneffekte am Halbspiegel

Halbspiegel sind ein in der Quantenoptik genutztes Bauteil. Schiebt man einen Lichtstrahl auf diesen, so wird ca. die Hälfte des Lichts reflektiert, während der Rest den Spiegel durchläuft. Diese 50-50-Verteilung ist aber wieder nur ein statistischer Wert, der bei großen Mengen an Photonen (z.B. einem Lichtstrahl) entsteht. Betrachtet man einzelne Photonen, so erkennt man, dass diese zufällig den Spiegel passieren oder von diesem reflektiert werden. Genauso wie beim Fluss von elektrischen Ladungsteilchen verhalten sich auch die Photonen am Halbspiegel voneinander unabhängig, wodurch nichtdeterministischer Zufall möglich ist. [14]

2 Konstruktion des Hardwaregeräts

2.1 Erste Umsetzungsidee

Meine erste Idee zur Umsetzung war die Nutzung verschiedener physikalischer Zufallseffekte. Hierfür habe ich in einer Literaturrecherche alles, was ich an physikalischen Zufallseffekten gefunden habe, aufgeschrieben und mir die vier am vielversprechendsten aussehenden Phänomene weiter angesehen. In den Abschnitten 1.3.2 bis 1.3.5 habe ich bereits die physikalischen Hintergründe der ausgesuchten Phänomene ausgeführt.

Neben der Entropiequelle bin ich auch auf eine Mess- und Kommunikationsvorrichtung angewiesen, die auf Basis eines Mikrocontrollers funktionieren sollte. Da ich zu Beginn der Arbeit nur wenig Erfahrung mit der Programmierung dieser Systeme hatte, habe ich mich für den einsteigerfreundlichen Arduino entschieden. Dieser wird in einem Dialekt der Programmiersprache C++ programmiert und ist sehr gut dokumentiert^[15]. Weitere Vorteile dieses Mikrocontrollers sind der günstige Preis von 10 bis 15 Euro und die hohe Kompatibilität zu den gängigen Betriebssystemen. Diese ist gegeben, da die meisten Arduinoboards per USB mit einem Computer verbunden werden können und dort als serielle Datenadapter funktionieren.

2.2 Testaufbau und Datensammlung

Bei der Messung der in Sektion 1.3 genannten Entropiequellen lassen sich die Messwerte entweder in analog und digital einordnen. Unter die Kategorie „digital“ bzw. „binär“ fallen die zeitlichen Messungen, da diese nur prüfen, ob ein bestimmter Schwellwert der Stromstärke erreicht ist und dies als Signal für den Beginn oder das Ende eines Zeitintervalls nutzen. Mit „analogen“ oder „absoluten“ Messwerten ist die Messung der elektr. Spannung gemeint. Radioaktive Zerfälle lassen sich digital messen, da bei jedem Zerfall kurzzeitig ein Signal vom Geigerzähler ausgeht. Elektrisches Rauschen wird sinnvollerweise analog gemessen.

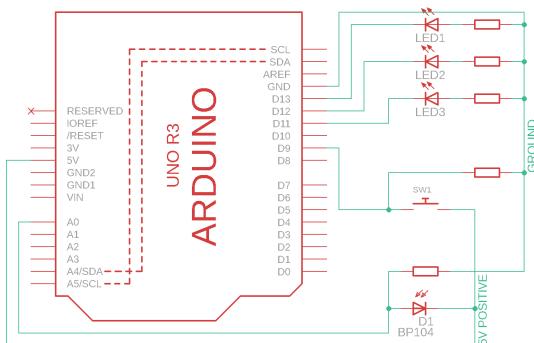


Abbildung 1: Verschaltung (Testaufbau)

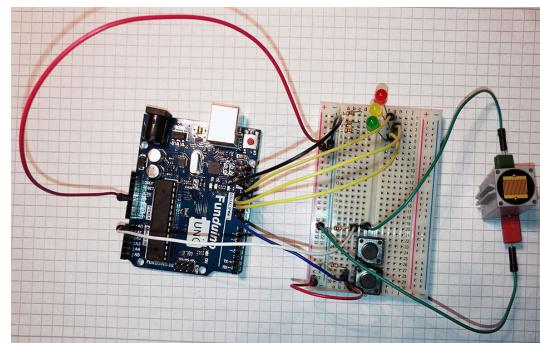


Abbildung 2: Testaufbau

Testaufbau Zur Auswertung dieser Signaltypen habe ich den in Abb. 1 und Abb. 2 gezeigten Versuchsaufbau entwickelt. In diesem stellt der Druckknopf (SW1) den Eingang für digitale Signale und der Photowiderstand (D1 BP104) den Eingang für die analogen Messwerte dar. Die Arduino-Anschlüsse mit der Kennzeichnung A* sind analoge Ein- und Ausgänge und können elektrische Spannungen in 1024 Stufen von 0V bis 5V messen. Dagegen sind die Anschlüsse mit der Bezeichnung D* digital und können nur prüfen, ob ein Schwellwert überschritten wurde. Die LEDs (LED1 - LED3) sind als Statusindikatoren verbaut. Beim weiteren Testen wurden der Schalter und die Photodiode durch die richtigen Messvorrichtungen ersetzt.

Zufallszahlen Die resultierenden Zufallszahlen sollen in einem Bereich von 0 bis 255 liegen. Dies hat den Grund, dass $256 = 2^8$ Möglichkeiten allen möglichen Zuständen eines 8-Bit-basierten Byte in der EDV entspricht.

2.3 Umsetzungsideen des Generators bei verschiedenen Entropiequellen

Für die ersten Tests mit den Messaufbauten habe ich den in Abb. 2 (S. 3) gezeigten Arduino Uno verwendet, welcher durch Steckanschlüsse leicht umbaubar ist. Für die finalen Konstruktionen verwende ich den Arduino Micro, eine kompaktere Fassung zum Verlöten. Die beiden Modelle unterscheiden sich in ihrer Funktionsweise kaum voneinander, der Arduino Micro hat jedoch den Vorteil, dass sein USB-Controller umprogrammierbar ist, wodurch der Arduino als komplett anderes Hardwaregerät erkannt werden kann.^[16]

2.3.1 Radioaktivität

Für diese Messmethode habe ich beim finalen Aufbau einen handelsüblichen Geigerzähler geöffnet und an der Stelle, an der normalerweise ein Summer als Lieferant für akustische Signale für die einzelnen Zerfälle verbaut ist, zwei Kabel verlötet und diese durch eine Bohrung im Gehäuse nach außen geführt. Die Kabel führen zum Steuercircus eines Reed-Relais auf einer an der Rückseite des Geigerzählers befestigten Steckplatine (Abb. 8), wo nun bei jedem Zerfall kurzzeitig ein Stromkreis zu einem der Digitalmesspunkte am Arduino geschlossen wird. Der Arduino ist weiterhin mit zwei Status-LEDs verbunden. Somit ist die Schaltung in der Lage, die Zerfälle beim Testen direkt anzuzeigen. Durch die Umgebungsstrahlung gibt es genügend messbare Zerfälle, um Zufallszahlen erzeugen zu können. Möchte man allerdings eine größere Menge an Zufallswerten in kurzer Zeit generieren, so kann man ein radioaktives Präparat mit niedriger Strahlendosis verwenden. Hierfür verwende ich bisher einen Glühstrumpf, einen Campingartikel, der früher in Gaslampen verwendet wurde. Glühstrümpfe sind mit einem Thoriumsalz versetzt, wodurch diese in unbedenklicher Stärke strahlen.

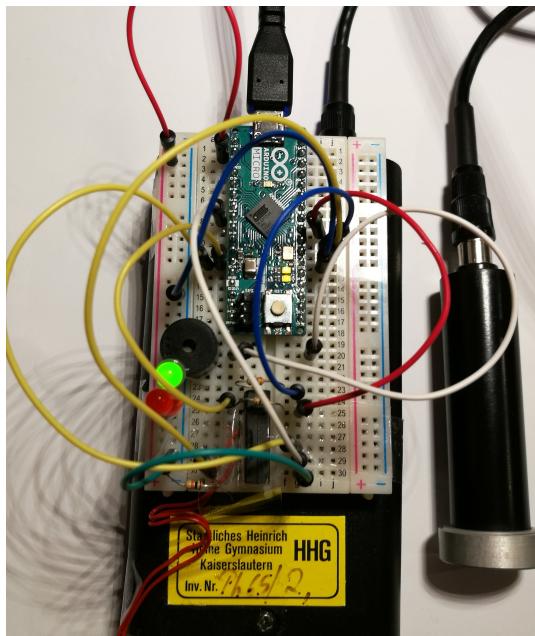


Abbildung 3: Aufbau Geigerzähler

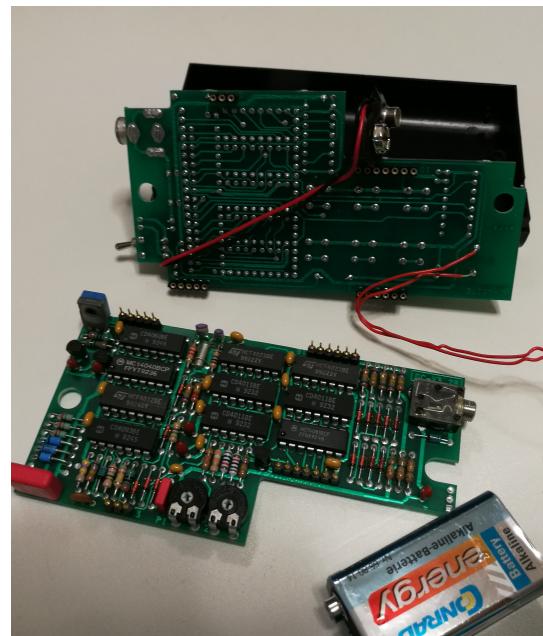


Abbildung 4: Umgebauter Geigerzähler

Für das finale Produkt möchte ich auf den fertig zusammengebauten Geigerzähler verzichten, stattdessen möchte ich ein platzsparendes Erweiterungsmodul mit Geiger-Müller-Zählrohr nutzen, mit dem ich die Zerfälle messe. Zudem möchte ich den Glühstumpf durch einen leichten Industriestrahler (Metallstift mit geringer radioaktiver Aktivität) ersetzen. Aufgrund der niedrigen Dosisleistung und großen Halbwertszeit kommt hierfür das synthetisch erzeugte ²⁴¹Americium infrage, welches bereits in Ionisationsrauchmeldern genutzt wird^[17].

2.3.2 Chaospendel

Meine Idee für die Zufallsgenerierung aus einem Chaospendel ist die Messung der Bewegung des äußeren Pendelarms. Damit sich dieser chaotisch bewegt, muss der innere Pendelarm regelmäßig beschleunigt und abrupt gebremst werden. Diese Rotationsbewegung ist im Schaubild mit dem gelben Pfeil dargestellt. Die Messung der zufälligen Bewegung soll mit der Hilfe von einer oder mehreren Lichtschranken stattfinden, die der äußere Pendelarm in unregelmäßigen Zeitabständen durchläuft. Die Lichtschranken funktionieren wie Photowiderstände, die mit einem Lichtstrahl beschienen werden und damit für den Stromfluss durchlässig sind. Unterbricht ein Gegenstand, hier z.B. der Pendelarm, den Lichtstrahl, so wird der Photowiderstand undurchlässig. Diese Änderung des Stromflusses kann vom Arduino gemessen werden.

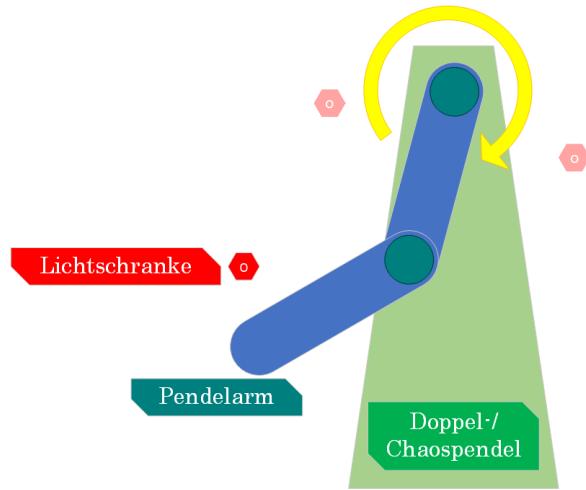


Abbildung 5: Chaospendel mit Lichtschranken

Dieser Ansatz wurde aus zwei Gründen nicht realisiert: Ein Chaospendel ist, wenn auch in der Praxis unberechenbar, deterministisch und widerspricht somit meinem Projektziel. Außerdem hat ein solches Gerät eine unpraktische Größe und ist sehr anfällig für Schäden.

2.3.3 Akustisches und elektrisches Rauschen

Bei diesem Ansatz mache ich mir eine Kombination aus einem zufälligen Umweltfaktor und einem physikalischen Phänomen zu Nutze. Zunächst messe ich mit einem Kapselmikrofon das von einem Summer erzeugte Geräusch zusammen mit anderen Umgebungsgeräuschen. Das hierbei entstandene Rauschen fluktuiert stark und kann exemplarisch im Diagramm gesehen werden:

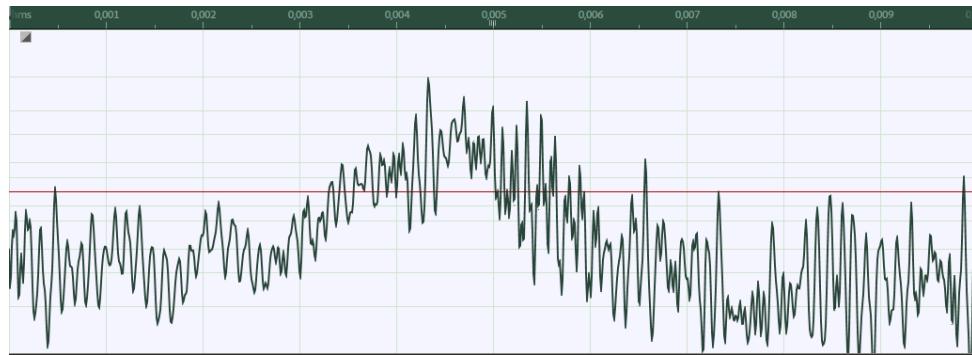


Abbildung 6: Akustisches Rauschen (Messintervall: $\Delta t = 0.01s$)

Nach der Lautstärkenmessung verstärke ich das bereits verrauschte Signal und erhalte somit ein noch stärkeres Rauschen. Das resultierende Signal variiert so stark, dass eine Manipulation durch gesteuerte Hintergrundgeräusche praktisch unmöglich ist. Ein alternativer Aufbau zu diesem Verstärkerrauschen ist die Nutzung einer Z-Diode, einem Bauteil, das sich unter bestimmten Bedingungen wie ein nichtdeterministisch variierender Widerstand verhält. Beide Möglichkeiten werden am Arduino über die Analog-Anschlüsse gemessen und erzeugen hier bereits einen zufälligen Messwert, der meist in einem Rahmen von 150 bis 200 Messstufen schwankt und deswegen weiterverarbeitet werden muss.

2.3.4 Transmission am Halbspiegel

Ein Halbspiegel (bzw. Strahlteiler) ist ein optisches Bauteil, welches einen Lichtstrahl teilt, indem er die Hälfte der Photonen ablenkt und die restlichen durchlässt. Betrachtet man ein einzelnes Photon, so ist es zufällig, welchen Weg dieses wählt. Meine Idee für einen Messaufbau wäre deswegen ein präziser Laserstrahl, von dem nach einem optischen Filter nur noch einzelne Photonen übrigbleiben. Die einzelnen Photonen werden nun am Halbspiegel zufällig abgelenkt oder zu einem Sensor durchgelassen.



Abbildung 7: Zufallswerte durch Photonenablenkung am Halbspiegel

Gängige Photonendetektoren können vom Arduino direkt ausgelesen werden, da diese ein elektrisches Signal aussenden, sobald ein Photon gemessen wurde. Ein großer Vorteil dieser Methode ist der hohe Kompaktheitsgrad, da abgesehen vom Detektor keines der Bauteile größer als eine Fünf-Cent-Münze ist.

Dieser Ansatz konnte aus dem Grund, dass die Erkennung einzelner Photonen sehr komplex und ein fertiger Einzelphotonendetektor entsprechend teuer ist, nicht realisiert werden.

2.4 Prototyp des Gehäuses

Zu einem endanwendergerechten Gerät gehört auch ein Gehäuse. Ich möchte ein solches Gehäuse mit einem 3D-Drucker bauen und habe dafür bereits ein 3D-Modell mit der Software „Inventor“ erstellt:

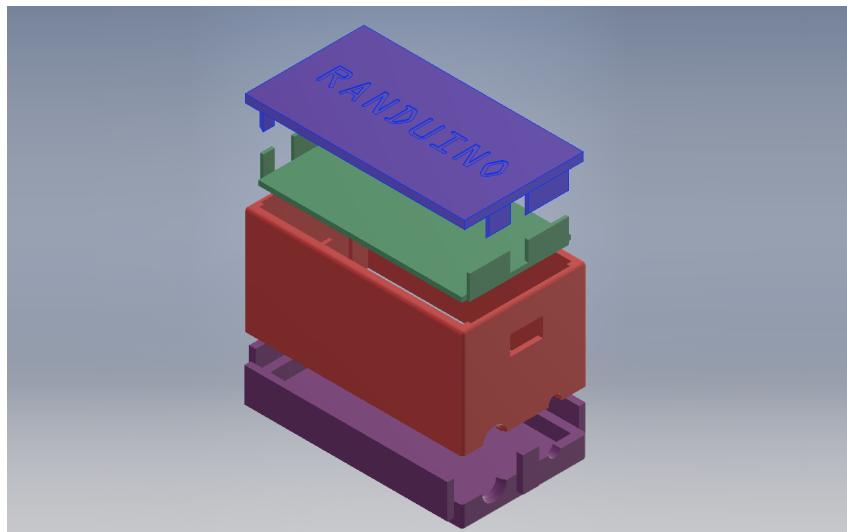


Abbildung 8: CAD-Design des Gehäuses

Das Gehäuse ist aus mehreren Ebenen zusammengesetzt. Es enthält Platz für den Geigerzähler im violetten und roten Teil (s. Abb. 8). Der grün markierte Zwischenboden wird auf halber Höhe in den roten Rumpf eingesetzt. Der Boden enthält Öffnungen für die vom Geigerzähler abgehenden Kabel, sodass die gesamte Elektronik in der oberen Kammer befestigt werden kann. Über dem Zwischenboden soll auch die Rauschmessung verbaut sein, sodass der Prototyp beide Entropiequellen enthält.

3 Softwareseitige Implementierung

3.1 Messung und Umrechnung zu Zufallszahlen

Die radioaktiven Zerfälle und das elektrische Rauschen erschienen mir als Entropiequellen für mein Projekt auch nach intensiverer Recherche und Planung am geeignetsten. Die beiden Ansätze habe ich schlussendlich auch realisiert und die anderen Ansätze aus genannten Gründen verworfen. Dabei messe ich bei den radioaktiven Zerfällen zufällige Zeitpunkte, an denen ein Signal auftritt, und beim Rauschen elektrische Spannungen.

3.1.1 Berechnung aus Zeitpunkten (Radioaktivität)

Bei der Generierung von Zufallszahlen aus Zeitpunkten ist es nicht möglich, die Zufallszahlen erst bei Abruf zu erzeugen, weswegen ein Zwischenspeicher für die Zufallszahlen (Puffer) notwendig ist. Ich habe diesen Puffer als **Queue** realisiert, was bedeutet, dass die Werte in Generierungsreihenfolge gespeichert zurückgegeben werden (**Last-In-Last-Out-Prinzip**).

Meine Lösung für die Generierung aus Zeitpunkten ist ein interner Zähler im Arduino, der dauerhaft von null bis 255 zählt und danach wieder auf null zurückspringt. Trifft nun ein Signal ein, so wird der aktuelle Zählwert im Puffer gespeichert und eine Einstellung wird geändert, sodass der Arduino weiterzählt, ohne bei jedem Zähldurchlauf den Wert erneut zu speichern. Endet das Signal vom Geigerzähler, so wird die Einstellung zurückgesetzt und ein neuer Wert kann generiert werden. Diese Methode ist solange möglich, wie der Arduino schneller zählt als die Kernzerfälle gemessen werden.

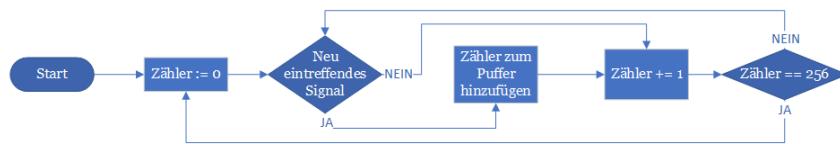


Abbildung 9: Schaubild „Generierung aus Zeitpunkten“

Sobald eine Anfrage nach einer neuen Zufallszahl vom Computer gesendet wird, wird das älteste Element des Puffers übertragen und aus dem Puffer entfernt.

3.1.2 Berechnung aus absoluten Messwerten (Rauschen)

Der Spannungssensor am Arduino gibt nur Messstufen (ganzzahlige Werte von 0 bis 1024) zurück. Da die realistische Wertemenge beim Rauschen nur ca. 100 Werte enthält, und die besonders hohen oder niedrigen Spannungen nur selten gemessen werden, eignen sich die Rohdaten nicht besonders gut als Zufallszahlen. Wird nun vom Computer eine Zufallszahl gefordert, so muss eine Messung durchgeführt und aus dem gemessenen Wert eine Zufallszahl generiert werden. Für die Umwandlung der Messwerte in Zufallszahlen habe ich mir die folgenden beiden Methoden überlegt:

Generierung durch Hashing Die Idee hinter dieser Methode ist die Nutzung einer chaotischen Einwegfunktion, also einer Funktion, die bei einer kleinen Änderung am Eingangsparameter eine große Änderung am Resultat erzeugt. Außerdem kann der Eingangsparameter einer Einwegfunktion nicht aus dem Ergebnis zurückgerechnet werden. Ich habe mich hierbei für die MD5-Methode aus der Prüfsummenberechnung entschieden. Die Funktion gibt einen 32-stelligen Hexadezimalwert, den sog. Hash, zurück. Ich verwende zwei Stellen des Hashes als Zufallswert ($16^2 = 256$ Möglichkeiten für zwei Hexadezimalstellen) und weitere zwei Stellen als Salt (Verfälschungsparameter) für die nächste Berechnung.

Generierung als Bitfolge Bei dieser Methode werden nicht eine, sondern acht Messungen durchgeführt und deren Messwerte jeweils mit mod 2 verrechnet. Ist das Resultat 0, so wird dieses als LOW-Bit an eine Bitfolge angehängt, ist es 1, so wird an die Bitfolge ein HIGH-Bit angefügt. Die Folge aus den acht Bits wird in eine Zufallszahl von 0 bis 255 konvertiert (8 Bits können $2^8 = 256$ Werte darstellen).

3.2 Client-Software

3.2.1 Treiber und Randuino-Service

Treiber Ich verwende den quelloffenen `libusb`-Treiber^[19], welcher eine einfache Kommunikation mit seriellen Verbindungen ermöglicht. Den Treiber habe ich etwas angepasst, sodass dieser auf Geräte mit dem „Randuino“-Hardware-Deskriptor achtet und diesen auch als „Randuino“-Verbindung im System identifiziert. Bei meinen Tests bin auf das Problem der Treiberzertifizierung gestoßen. Windows verlangt, dass Treiber ein Zertifikat besitzen, welches von einer vertrauten Zertifizierungsstelle ausgestellt wurde. Ein solches Zertifikat kostet viel Geld (ca. 400 Euro pro Jahr), weswegen ich eine eigene Zertifizierungsstelle erstellt und im lokalen System installiert habe. Der Randuino-Treiber wird nun mit eigenen Zertifikaten validiert.

Randuino-Service Um auf die Zufallswerte zugreifen zu können, habe ich in der Programmiersprache C++ einen Hintergrunddienst programmiert, welcher sich automatisch zum Randuino verbindet und mit diesem exklusiv kommuniziert. Der Dienst prüft regelmäßig, ob ein serieller Datenadapter verbunden ist. Findet er einen, so wird versucht, eine Verbindung zu diesem aufzubauen und der „connect“-Befehl wird gesendet. Antwortet der Zufallsgenerator an dieser Stelle mit „randuino“ so gilt die Verbindung als aufgebaut. Da theoretisch jedes Gerät diese Antwort senden kann, ist für die Zukunft eine weitere Absicherung an dieser Stelle durch eine Signaturprüfung geplant. Dafür soll der Computer einen sog. Payload (eine kleine Datenmenge) an den Arduino senden, der für diesen eine Signatur generiert und diese zurückübermittelt. Der Computer kann wiederum die Signatur überprüfen und damit sicherstellen, dass er mit einem echten „Randuino“ kommuniziert.

Damit andere Prozesse die Zufallszahlen nutzen können, stellt der Hintergrunddienst einen lokalen Server bereit, auf den sich andere Prozesse verbinden können. Nach einer erfolgreichen Verbindung wird eine Zufallszahl an den verbundenen Prozess gesendet. Ist kein Zufallsgenerator verbunden, oder tritt ein Fehler auf, so wird „device_not_available“ übertagen. In allen Fällen wird die Verbindung nach der Übertagung geschlossen.

Verwendung unter UNIX-artigen Systemen Unter UNIX-artigen Systemen muss der Benutzer der `dialout`-Gruppe angehören (ggf. wird auch die `uucp`-Gruppe benötigt), da sonst die Berechtigungen zum Lesen und Schreiben der seriellen Anschlüsse fehlen.

3.2.2 Übertragungsprotokoll

Softwareseitig funktioniert der Zufallsgenerator nach einem Request-Reply-Schema. Dabei kann der PC bzw. Server bestimmte Zeichenfolgen, gefolgt von einem Zeilenumbruch-Byte („\n“), senden und bekommt eine Antwort, welche ebenfalls den Zeilenumbruch-Character als Stop-Byte nutzt. Der Arduino kann standardmäßig die folgenden Befehle beantworten:

Befehl	Antwort	Beschreibung
<code>connect</code>	<code>randuino</code>	Sendet den Gerätetyp zurück. Wird zur Identifikation beim Verbindungsauflauf benötigt.
<code>single</code>	Zufallszahl oder <code>buffer_empty</code>	Sendet eine Zufallszahl oder eine Meldung bei der Zeitpunktmeßung, dass keine Zufallswerte verfügbar sind.
<code>buffer</code>	<code>B:Puffergröße</code>	Sendet die Zahl der verfügbaren Zufallszahlen. Dieser Befehl ist nur bei der Zeitpunktmeßung verfügbar.
<code>heartbeat</code>	<code>still.alive</code>	Bestätigt eine aktive Verbindung.

Abbildung 10: Befehle und Antworten (jeweils ohne \n)

4 Bewertung der Zufallsgeneratoren

4.1 Aufstellung eines Bewertungsmodells

4.1.1 Zufälligkeit und Verteilung der Wertemenge

Der wichtigste Faktor zur Bewertung eines Zufallsgenerators ist die Zufälligkeit der Werte. Im besten Falle kommen alle Werte ähnlich häufig vor. Um dies zu testen, generiere ich einen Satz von 20000 Zufallswerten pro Generator, da diese Datensatzgröße nach dem FIPS-140-Standard zum Prüfen von Zufallsgeneratoren empfohlen wird,^[20] und lasse diese verschiedene statistische Tests durchlaufen. Hierfür erwarte ich, dass jeder Wert ungefähr $\frac{20000}{256} = 78,18$ Male vorkommt. Ich errechne die durchschnittliche und maximale Abweichung von dieser Erwartung und betrachte diese. Bei der durchschnittlichen Abweichung sehe ich einen Wert unter 10% in Relation zum Erwartungswert als akzeptabel verteilt an, die größte Abweichung sollte einen Wert von 50% relativ zu den 78,18 Vorkommnissen nicht überschreiten. Ein weiterer Test ist die Zählung der geraden Zahlen unter den generierten Zufallswerten, hierbei sollte ungefähr die Hälfte der Zufallszahlen des Datensatzes gerade und ebenso viele ungerade sein. Lässt sich bei diesem Test eine größere Ungleichmäßigkeit ($> 5\%$) erkennen, so gilt der Datensatz als ungleichmäßig verteilt. Außerdem berechne ich den arithmetischen Mittelwert aus allen Zufallszahlen. Auch dieser sollte um nicht mehr als 5% vom erwarteten Mittelwert von 127,5 abweichen. Für die Zukunft möchte ich für die Auswertung der Zufallszahlen statistische Mittel wie die Normalverteilung nutzen und somit die Validierung weiter ausbauen.

Neben einer gleichmäßigen Werteverteilung sollten sich bei einem nichtdeterministischen Zufallsgenerator Folgen bestimmter Werte nicht übermäßig oft wiederholen. Um dies zu überprüfen, habe ich ein Programm geschrieben, welches über die Testmenge iteriert und die Vorkommnisse aller möglicher Ketten mit einer Länge von $2 \leq l \leq 5$ zählt. Für jede Länge l erwarte ich eine durchschnittliche Vorkommniszahl μ_l , die sich aus der Zahl der möglichen Positionen der Ketten (von Startpunkt 0 bis 20000 – l) geteilt durch die Zahl der möglichen Wertekombinationen zusammensetzt. Daraus ergibt sich die Formel $\mu_l = (20000 - l)/(256^l)$. Der Wertebereich von l ist auf maximal 5 eingegrenzt, da es bei Ketten dieser Länge bereits $256^5 \approx 1,0995 \cdot 10^{12}$ mögliche Zusammensetzungen gibt und jede dieser Ketten $\mu_l = 19995/(256^5) \approx 1,82 \cdot 10^{-8}$ mal im Durchschnitt erwartet wird. Somit ist davon auszugehen, dass keine Kette der Länge 5 häufiger als einmal vorkommt^[21].

4.1.2 Geschwindigkeit der Zahlengenerierung

Gerade für Serveranlagen mit hohem Bedarf an Zufallszahlen darf der Generator nicht zum Bottleneck werden. Aus diesem Grund müssen Zufallszahlen schneller generiert als verbraucht werden. Ich gebe die Geschwindigkeit der Generatoren in Zahlen pro Sekunde an, dabei entspricht eine Zahl einem Zufallsbyte. Außerdem sollten die Werte von dem die Werte nutzenden Softwaresystem optimal genutzt werden, sodass z.B. aus einem Zufallsbyte (acht zufällige Bits) acht zufällige true/false-Entscheidungen erzeugt werden und nicht nur das letzte Bit (gerade oder ungerade Zahl) genutzt wird.

4.1.3 Sicherheit des Generators

Die Sicherheit des Generators umfasst zwei verschiedene Faktoren: Zum einen sollte der Generator vor Manipulation geschützt sein, zum anderen soll vom Generator keine Gefahr für die Benutzer ausgehen. Allgemein kann der Manipulationsschutz hardwareseitig durch eine Plombierung gewährleistet werden. Softwareseitig kann eine Signierung eine Möglichkeit sein, sodass der Zufallsgenerator zu einem Code eine Signatur senden muss.

4.1.4 Preis des Geräts

Damit ein solcher Generator in der Praxis verwendet wird, sollte dessen Preis möglichst niedrig sein. Mein Ziel ist es, einen Generator für Endanwender mit einem Bauteilepreis von unter 50 Euro und einen für Server verbesserten Generator für weniger als 300 Euro zu bauen.

4.2 Anwendung des Bewertungsmodells

4.2.1 Radioaktive Zerfälle

Generierung der Zufallszahlen Der Arduino zählt intern bei hoher Frequenz von null bis 255. Trifft ein neues Signal vom Geigerzähler ein, so wird der aktuelle Wert als Zufallszahl im Puffer gespeichert.

Zufälligkeit und Werteverteilung Im folgenden Diagramm sieht man die Werteverteilung einer Stichprobe mit $N = 20'000$ mit radioaktiven Zerfällen errechneten Zufallszahlen:

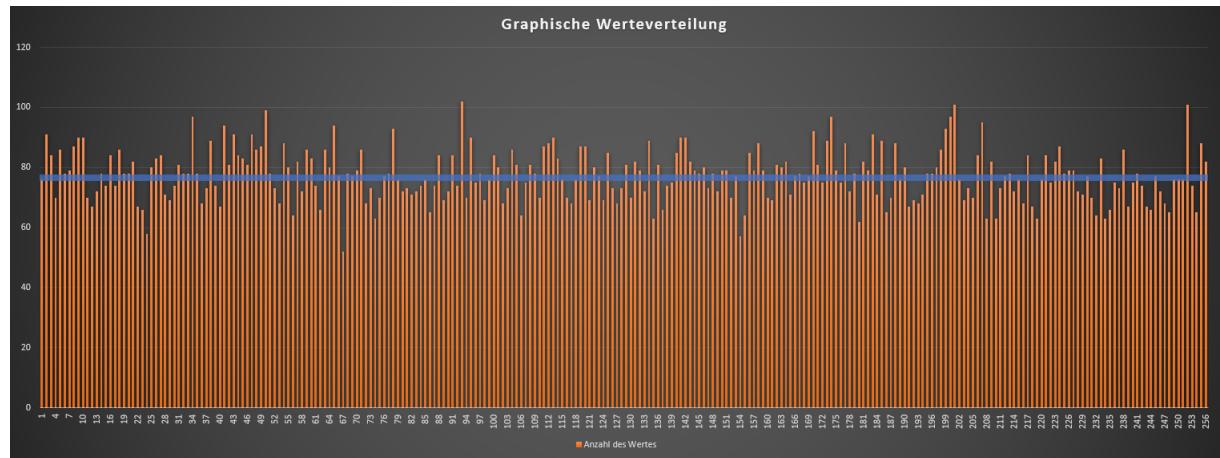


Abbildung 11: Werteverteilung „Radioaktive Zerfälle“
(blau eingezzeichnet die erwartete Häufigkeit)

Bei statistischer Auswertung der 20'000 Zufallswerte erkennt man eine durchschnittliche Abweichung um 6.82 Vorkommnisse von den erwarteten 78,13 Vorkommnissen, was 9% entspricht. Die größte Abweichung beträgt 26,13 Vorkommnisse bei der Zahl 66, was 33% entspricht. Bei der Auswertung eines anderen Zufallszahlensatzes, der mit der gleichen Methodik erzeugt wurde, hatte die Zahl 66 74 Vorkommnisse. Als Durchschnitt aller Zufallswerte wird 127,5 erwartet, der tatsächliche Durchschnitt beträgt 125,46, was einer Abweichung von 1,02% entspricht. Außerdem wird erwartet, dass 10'000 der Zufallszahlen ganzähnlich durch zwei teilbar sind, der tatsächliche Wert weicht mit 10160 nur um 1,60% ab. Alle diese Werte deuten auf einen echten Zufallsgenerator mit etwa gleich verteilten Wahrscheinlichkeiten hin. Der Fakt, dass beim Suchen nach Folgen ebenfalls keine Ketten mit übermäßiger Häufigkeit gefunden wurden, verstärkt diese Vermutung.

Geschwindigkeit In Puncto Geschwindigkeit werden mit der reinen Umgebungsstrahlung ca. 20 bis 40 Zufallszahlen pro Minute generiert. Verwendet man einen handelsüblichen Glühstrumpf als radioaktives Präparat, so erreicht man ca. 3 bis 6 Zahlen in der Sekunde, was 180 bis 360 Zahlen in der Minute entspricht.

Sicherheit Die Strahlung eines verwendeten Glühstrumpfs lässt sich nicht mit einem zweiten Geigerzähler gleich messen, da dieser andere Zerfälle wahrnimmt. Ein zweiter Sicherheitsfaktor ist das radioaktive Isotop, welches zweckentfremdet werden könnte. Die Lösung hierfür sind sog. Industriestrahler, kleine radioaktive Metallstifte, die in unbedenklicher Menge strahlen und im Gerät fest vergossen werden (vgl. 2.3.1 unten).

Preis Der Aufbau kostet zusammen mit dem Geigerzähler ca. 200 Euro, was ihn zwar für Heimanwender ungeeignet macht. Aufgrund der hohen Generierungsgeschwindigkeit ist der Preis für Serveranlagen bezahlbar.

4.2.2 Akustisches und elektrisches Rauschen (Generierung durch Hashing)

Generierung der Werte Bei dieser Methode werden regelmäßig Messwerte der elektrischen Spannung nach Verstärkung des Rauschens eingelesen und jeweils ein MD5-Hash aus dem Messwert und einem Teil des vorherigen Hashes als Salt bestimmt.

Zufälligkeit und Werteverteilung Im folgenden Diagramm sieht man, welche Zufallszahl wie häufig unter 20'000 hierfür mit akustischem Rauschen gezogenen Zufallszahlen vorkam:

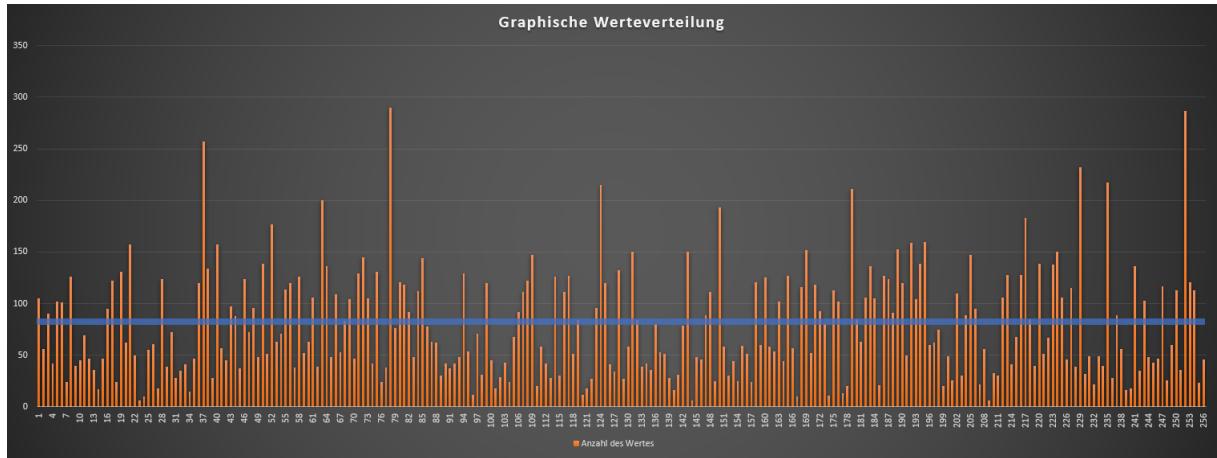


Abbildung 12: Werteverteilung „Akustisches Rauschen (Hashing)“
(blau eingezeichnet die erwartete Häufigkeit)

Bei statistischer Auswertung der 20'000 Zufallswerte erkennt man eine durchschnittliche Abweichung um 42,32 Vorkommnisse von den erwarteten 78,13 Vorkommnissen, was 54,17% entspricht. Die größte Abweichung beträgt 211,87 Vorkommnisse bei der Zahl 77, was 271,18% entspricht. Bei der Auswertung eines anderen Zufallszahlensatzes, der mit der gleichen Methodik erzeugt wurde, hatte die Zahl 77 wieder mit 263 Vorkommnissen einen extrem hohen Anteil. Als Durchschnitt aller Zufallswerte wird 127.5 erwartet, der tatsächliche Durchschnitt beträgt 129.52, was einer Abweichung von 1.58% entspricht. Außerdem wird erwartet, dass 10'000 der Zufallszahlen ganzähnlich durch zwei teilbar sind, der tatsächliche Wert weicht mit 9955 nur um 0.45% ab. Die große Abweichung der Vorkommnisse einiger einzelner Werte, wie sie im Diagramm ersichtlich sind, und die große durchschnittliche Abweichung sprechen gegen eine gleichmäßige Verteilung der Zufallswerte. Zudem konnte ich eine erhöhte Wahrscheinlichkeit bestimmter Folgen nachweisen. Somit muss die Berechnung, nach die die Zufallszahlen aus den Rauschwerten generiert, überarbeitet werden.

Verwendbarkeit Aufgrund der extrem ungleichmäßigen Werteverteilung wurde von der Verwendung dieser Methode abgesehen. Stattdessen wird die in 4.2.3 beschriebene Bitfolgenmethode verwendet.

4.2.3 Akustisches und elektrisches Rauschen (Generierung als Bitfolge)

Generierung der Werte Bei dieser Methode werden acht Messwerte der elektrischen Spannung nach Verstärkung innerhalb von kurzer Zeit eingelesen und jeweils mithilfe der Modulo-Funktion darauf getestet, ob diese gerade oder ungerade sind. Das Zufallsbyte setzt sich dann aus den acht Einzelbits (0 für gerade oder 1 für ungerade Messwerte) zusammen.

Zufälligkeit und Werteverteilung Im folgenden Diagramm sieht man, welche Zufallszahl wie häufig unter 20'000 hierfür mit akustischem Rauschen gezogenen Zufallszahlen vorkam:

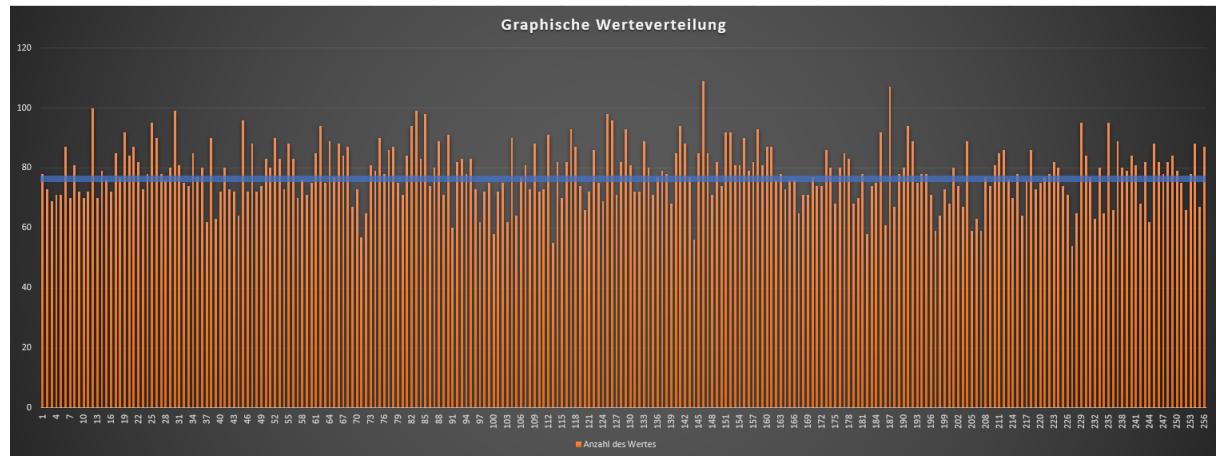


Abbildung 13: Werteverteilung „Akustisches Rauschen (Bitfolge)“
(blau eingezeichnet die erwartete Häufigkeit)

Bei statistischer Auswertung der 20'000 Zufallswerte erkennt man eine durchschnittliche Abweichung um 7,66 Vorkommnisse von den erwarteten 78,13 Vorkommnissen, was 9,8% entspricht. Die größte Abweichung beträgt 30,87 Vorkommnisse bei der Zahl 144, was 39,51% entspricht. Als Durchschnitt aller Zufallswerte wird 127,5 erwartet, der tatsächliche Durchschnitt beträgt 126,74, was einer Abweichung von 0,6% entspricht. Außerdem wird erwartet, dass 10'000 der Zufallszahlen ganzzahlig durch zwei teilbar sind, der tatsächliche Wert weicht mit 9997 nur um 0,03% ab. Auch hier deuten alle statistischen Berechnungen auf einen nichtdeterministischen Generator mit einer Gleichverteilung der Werte. Auch bei dieser Mess- und Rechenmethode konnten keine Auffälligkeiten in Form von sich häufig wiederholenden Ketten gefunden werden.

Geschwindigkeit Wie bereits im Modell beschrieben, können beim akustischen Rauschen nahezu unlimitiert viele Zufallszahlen angefordert werden, jedoch sollte es zwischen den Requests immer einen zeitlichen Abstand für An- und Abklingzeiten geben. Die hieraus resultierende Totzeit kann bei der seriellen Übertragung der Werte an den Zielcomputer verwendet werden, wodurch diese stets eingehalten wird und trotzdem bis zu acht Werte in der Sekunde möglich sind.

Sicherheit Da ein Großteil des Rauschens innerhalb der Verstärkerschaltung erzeugt wird, kann dieses nur durch Anzapfen der inneren Leitungen nach außen geleitet und somit erraten werden. Eine Versiegelung der Hülle würde hierbei zur Absicherung genügen, ein intaktes Siegel indiziert, dass das Gerät nicht geöffnet und manipuliert wurde.

Preis Die Summe der verwendeten Bauteile und des Gehäuses beträgt ungefähr 30 Euro, wodurch dieser Aufbau auch für Heimanwender erschwinglich ist.

5 Zusammenfassung und Fazit

5.1 Ergebnis der Arbeit

Insgesamt habe ich es geschafft, bis zum jetzigen Zeitpunkt zwei der vier geplanten Konstruktionen in funktionsfähige Aufbauten umzusetzen und habe beide Zufallsgeneratoren nach verschiedenen Kriterien geprüft und bewertet. Hierbei habe ich erkannt, dass der auf radioaktiven Zerfällen basierende Zufallsgenerator bereits Zufallszahlen einer höheren Güte produziert, während mein erster auf akustischen Rauschen basierte Ansatz (Generierung durch Hashing) bisweilen sehr ungleich verteilte Zufallswerte von somit eher niedrigerer Güte produziert. Der zweite Ansatz für die Zufallszahlengenerierung aus akustischem Rauschen (Generierung als Bitfolge) eignet sich hingegen sehr gut. Aufgrund seiner hohen Geschwindigkeit ist der auf Radioaktivität basierte Generator für Serveranlagen und der auf elektrischem Rauschen basierte Generator wegen seines niedrigen Preises für Endanwender geeignet.

Im Laufe der Ausarbeitung hat sich mein Blickwinkel auf das Projekt von der Konstruktion eines nicht-deterministischen Zufallsgenerators zur Optimierung und Findung der besten Umrechnungsmethode verschoben. Ferner plane ich die Veröffentlichung der Baupläne und Schaltkreise zusammen mit Erklärungen und möchte es so auch Privatpersonen ermöglichen, sichere Zufallszahlen für kryptographische Vorgänge wie Signaturen zu generieren.

Diese Arbeit wurde außerdem in einer stark abgewandelten Form als Facharbeit im Kurs Physik genutzt.

5.2 Weiterer Ausblick

Im weiteren Verlauf des Projekts möchte ich den Zufallsgenerator zu einem vollendeten Hardwaregerät in anwendergerechter Größe mit einem Gehäuse umbauen. Hierfür überlege ich auch, die Arduino-Plattform zu verlassen und auf Hardwareentwicklung optimierte Mikrocontroller zu setzen. Auch möchte ich das Projekt im zwei voneinander unabhängige Systeme aufteilen, davon eines für Server und das andere auf Endanwender optimiert. Außerdem plane ich eine RSA-Implementierung auf Basis meines Zufallszahlengenerators und teste zudem aktuell auch neue Entropiequellen wie Umpolungseffekte am Magnetfeld. Zur Testung der Wertemenge dieser und der bereits implementierten Generatoren möchte ich mein Prüfungsverfahren mit einer Anwendung der statistischen Normalverteilung erweitern.

Ein weiterer Forschungspunkt ist die Verbesserung der Abhärtung des Geräts, sodass das Abgreifen von Zufallszahlen unmöglich ist. Hierfür möchte ich Kontakt zu Pentesting-Experten aufnehmen und selbst versuchen, Sicherheitslücken zu finden und diese zu schließen.

5.3 Danksagung

Zum Schluss möchte ich mich noch recht herzlich bei allen denen bedanken, die mich bei der Ausarbeitung und bei Versuchen unterstützt und mir mit guten Tipps und Materialien geholfen haben. Hierzu zähle ich auch meine Eltern, Lehrer, Mitschüler und Juroren, die mich zusätzlich immer weiter motiviert haben.

6 Anhang

6.1 Quellenverweise

In Klammern hinter den Quellenverweisen befinden sich die Daten der jeweiligen letzten Zugriffe.

- [1] https://de.wikipedia.org/wiki/Transport_Layer_Security
Abschnitt „TLS in der Praxis“ (31.03.2019)
- [2] <https://curiosity.de/wissenswertes/rsa-verschlüsselung-einfach-erklaert/>
Gesamte Seite (31.03.2019)
- [3] <https://ee.stanford.edu/~hellman/publications/24.pdf>
Abschnitt „Introduction“ (18.03.2019)
- [4] <https://www.random.org/randomness>
Absatz „Pseudo-Random Number Generators (PRNGs)“ (18.03.2019)
- [5] <https://www.elektronik-kompendium.de/sites/net/1910301.htm>
Absatz „Pseudozufallsgenerator“ (19.03.2019)
- [6] Grehn J. & Krause J. (2007) Metzler Physik (4. Auflage). Braunschweig, Deutschland: Schrödel.
Seite 106f.
- [7] <https://leifiphysik.de/kern-teilchenphysik/radioaktivitat-einführung>
Gesamte Seite (23.03.2019)
- [8] <https://www.leifiphysik.de/kern-teilchenphysik/radioaktivitaet-einfuehrung/ionisierung-durch-strahlung> Gesamte Seite (23.03.2019)
- [9] Grehn J. & Krause J. (2007) Metzler Physik (4. Auflage). Braunschweig, Deutschland: Schrödel.
Seiten 482f und 486ff.
- [10] <https://www.leifiphysik.de/waermelehre/deterministisches-chaos/versuche/doppelpendel> Gesamte Seite (31.03.2019)
- [11] https://www.uni-frankfurt.de/68943623/Elektronisches_Rauschen-231020171.pdf
Kapitel I - III (24.03.2019)
- [12] https://www.antennen-env.tu-berlin.de/fileadmin/fg13/Lernmaterialien/analog_10_Elektronisches_Rauschen.pdf - Kapitel 10.1 „Rauschursachen“ (24.03.2019)
- [13] [https://de.wikipedia.org/wiki/Rauschen_\(Physik\)](https://de.wikipedia.org/wiki/Rauschen_(Physik)) - Einleitung (24.03.2019)
- [14] <https://de.wikipedia.org/wiki/Strahlteiler> - Einleitung (31.03.2019)
- [15] <https://www.arduino.cc/en/Guide/HomePage> - (31.03.2019)
- [16] <https://store.arduino.cc/arduino-micro> - Abschnitt „Overview“ (02.04.2019)
- [17] <http://www.chemie.de/lexikon/Americium.html#Verwendung>
Einleitung und Abschnitt „Verwendung“
- [18] <https://qt.io/download> - Spalte „Open Source“ (22.03.2019)
- [19] <http://libusb.sourceforge.net/api-1.0/> - Dokumentation (01.04.2019)
- [20] <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf>
Kapitel 4.7.1 und 4.9 (22.03.2019)
- [21] Bigalke A. & Köhler N. (2011) Mathematik - Analytische Geometrie und Stochastik (2. Auflage).
Berlin, Deutschland: Cornelsen. Seiten 268ff und 306ff.

Dokumentenversion: Fassung vom 04.04.2019