Analyse und Implementierung einer Multi-Faktor-Authentifizierung mit Shamir's Secret Sharing

Nicolas Proske
Ostbayerische Technische Hochschule Amberg-Weiden
Moderne Anwendungen der Kryptographie
E-Mail: n.proske@oth-aw.de
Matr.-Nr.: 87672270

***Zusammenfassung*—Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.**

***Schlüsselwörter*—MFA, Secret Sharing**

I. Einleitung

In der heutigen digitalen Welt, in der eine überwältigende Menge an Daten generiert, gespeichert und über verschiedene Plattformen übertragen wird, ist der Bedarf an Datenschutz und Datensicherheit relevanter denn je. Die mit der Digitalisierung einhergehenden Möglichkeiten bergen ein erhebliches Risiko für Datendiebstahl, unbefugten Zugriff und Cyberangriffe. Obwohl laut einer Umfrage [1, S. 23] die Hälfte der Erwachsenen weltweit glauben, dass die von ihnen ergriffenen Maßnahmen ausreichen, um sich gegen Identitätsdiebstahl zu schützen, sind 63 Prozent darüber besorgt, dass ihre Identität gestohlen wird. Weiter fühlen sich knapp sieben von zehn Menschen heute anfälliger für Identitätsdiebstahl als noch vor ein paar Jahren. Ein wesentlicher Grund für die Zunahme von Identitätsdiebstahl liegt neben zu schwachen Passwörtern hauptsächlich daran, wie Menschen damit umgehen. Bei einer Frage bezüglich der mehrmaligen Verwendung derselben Benutzernamen und Passwörter haben 82 Prozent zugegeben, zumindest manchmal dieselben Anmeldedaten für unterschiedliche Konten zu verwenden. Knapp die Hälfte davon, etwa 45 Prozent, verwenden sogar immer oder in den meisten Fällen dieselben Zugangsdaten [2, S. 12].

Im Rahmen der vorliegenden Studienarbeit wird eine Analyse und Implementierung einer Multi-Faktor-Authentifizierung mit Shamir's Secret Sharing durchgeführt. Dazu wird zu Beginn auf unterschiedliche Authentifizierungsarten eingegangen, einschließlich relevanter Fakten sowie der jeweiligen Vor- und Nachteile. In einem weiteren Schritt erfolgt eine kurze Beschreibung der Methode inklusive einer mathematischen Veranschaulichung, um ein gemeinsames Verständnis für die nachfolgende Analyse der Implementierung zu erlangen. Daran anknüpfend führt eine Betrachtung relevanter Sicherheitsaspekte zu einer Zusammenfassung.

II. Authentifizierung mit Faktoren

A. Ein-Faktor-Authentifizierung

Lange Zeit haben Authentifizierungsmethoden auf einem einzelnen Identifikationsfaktor beruht, in der Regel einer Kombination aus Benutzername und Passwort. Wenn diese beiden Parameter über mehrere Dienste hinweg identisch sind, bedeutet dies, dass ein Angreifer, der ein einziges Konto kompromittiert, automatisch Zugriff auf die anderen Konten erhält — Dabei spielt die Stärke des Passworts keine Rolle. Dieser One-Factor-Authentication (OFA) Ansatz hat jahrzehntelang das Rückgrat der Informationssicherheit gebildet. Dennoch hat sich angesichts der zunehmenden Komplexität von Cyberbedrohungen gezeigt, dass die Abhängigkeit von einem einzigen Faktor für die Authentifizierung eine Schwachstelle darstellt, die anfällig für verschiedene Verletzungen wie Brute-Force-Angriffe, Phishing und Keylogging ist. Diese Schwachstellen verdeutlichen, dass OFA für heutige Anwendungsfälle in aller Regel keine ausreichende Sicherheit mehr bietet.

B. Multi-Faktor-Authentifizierung

Multi-Factor-Authentication (MFA) stellt einen signifikanten Fortschritt in der Evolution der digitalen Sicherheitsmaßnahmen dar. Im Gegensatz zur Einzelfaktor-Authentifizierung, die üblicherweise auf einer einzigen Form des Nachweises wie einem Passwort basiert, erhöht MFA die Sicherheit durch zusätzliche Schutzebenen, indem mehrere unabhängige Zugangsdaten für die Authentifizierung erforderlich sind. Diese Zugangsdaten können in drei Hauptkategorien eingeteilt werden:

1) *Wissen*: Informationen, die der Benutzer kennt, wie Passwörter, PINs und Antworten auf geheime Fragen.

2) *Eigenheit*: Biologische Merkmale, die einzigartig für den Benutzer sind, wie Fingerabdrücke, Netzhautmuster oder Gesichtserkennung.
3) *Besitz*: Gegenstände oder Geräte, die der Benutzer besitzt, wie Smartphones, Chipkarten oder physische Schlüssel. Die Bestätigung des Besitzes kann verschiedene Formen annehmen, angefangen von der Entgegennahme und Eingabe eines per SMS an eine registrierte Telefonnummer gesendeten Codes bis hin zum Einsetzen eines physischen Schlüssels in ein Schloss.

Der Hauptvorteil von MFA gegenüber OFA liegt daher im schichtbasierten Ansatz. Selbst wenn ein Angreifer es schafft, einen Authentifizierungsfaktor zu umgehen, bieten die verbleibenden Faktoren weiterhin Schutz. Eine Kompromittierung eines Faktors gefährdet also nicht die Gesamtsicherheit. Trotz der Stärken bringt eine MFA auch eigene Herausforderungen mit sich, wie beispielsweise die potenziell erhöhte Komplexität und die Notwendigkeit für Benutzer, mehrere Authentifizierungsfaktoren zu verwalten. Dennoch überwiegen die Vorteile der MFA oft diese potenziellen Nachteile, insbesondere in Umgebungen, in denen der Schutz sensibler Daten oberste Priorität hat.

III. Shamir's Secret Sharing

Secret Sharing ist ein grundlegender Baustein der modernen Kryptographie. Eines der bekanntesten Verfahren wurde am 1. November 1979 veröffentlicht und ist nach seinem Erfinder Adi Shamir, einem israelischen Kryptographen, benannt: Shamir's Secret Sharing [3].

Es basiert auf der Idee, ein Geheimnis in mehrere Teile, sogenannte Shares, aufzuteilen. Um das Geheimnis wiederherzustellen, müssen eine bestimmte Anzahl dieser Shares zusammengebracht werden. Jeder einzelne Share ist für sich genommen bedeutungslos und gibt keinerlei Informationen preis. Ein Schwellenwert definiert die minimale Anzahl von Shares, die erforderlich sind, um das Geheimnis wiederherstellen zu können. Dies stellt sicher, dass das Geheimnis selbst dann sicher bleibt, wenn ein Teil der Shares verloren gehen oder in die Hände eines Angreifers gelangen. Das dabei verwendete (k, n)-Schwellenwertschema legt fest, wie viele k Shares benötigt werden, um auf das Geheimnis zu kommen, n ist größer k und bezieht sich auf die Gesamtzahl der Shares, in die das Geheimnis aufgeteilt wird.

A. Mathematische Veranschaulichung

Shamir's Secret Sharing basiert auf dem Prinzip der Polynominterpolation in endlichen Körpern, wobei k Punkte ein Polynom vom Grad $k-1$ eindeutig definieren. Um dies anhand eines mathematischen Beispiels zu veranschaulichen, wird im Folgenden ein (2, 3)-Schwellenschema mit $k = 2$ und $n = 3$ betrachtet, bei dem das Geheimnis S der Zahl 42 entspricht. Sei $p = 43$ eine Primzahl mit $p > S$. Alle Berechnungen erfolgen im endlichen Körper $\mathbb{F}_p$.

1) Generierung der Shares: Der erste Schritt besteht darin, ein Polynom vom Grad $k - 1 = 2 - 1 = 1$ aufzustellen:

$$f(x) = mx + b \mod p$$

Die Konstante b entspricht dabei dem Geheimnis S. Aus Gründen der Übersichtlichkeit wird in diesem Beispiel $m = 4$ gewählt:

$$f(x) = 4x + 42 \mod 43$$

Im nächsten Schritt erfolgt die Berechnung von n Punkten in der Ebene. Dazu wird für $x = 1...n$ eingesetzt:

$$\begin{aligned}
&\text{Für } x = 1 : y_1 = f(1) = 4 * 1 + 42 \mod 43 = 3\\
&\text{Für } x = 2 : y_2 = f(2) = 4 * 2 + 42 \mod 43 = 7\\
&\text{Für } x = 3 : y_3 = f(3) = 4 * 3 + 42 \mod 43 = 11
\end{aligned}$$

Jeder der berechneten Punkte $(1, 3), (2, 7), (3, 11)$ repräsentiert dabei einen Share.

2) Rekonstruktion mit linearem Gleichungssystem: Sind nun k Punkte gegeben, kann das ursprüngliche Geheimnis rekonstruiert werden. Im Folgenden werden die Punkte $(1, 3)$ und $(2, 7)$ als Gleichungen in einem linearen Gleichungssystem dargestellt:

$$\begin{aligned}
&1.\ m + b = 3\\
&2.\ 2m + b = 7
\end{aligned}$$

Die Unbekannten werden nun über das Substitutionsverfahren gelöst. Durch Umstellen der ersten Gleichung nach b folgt $b = 3 - m$. Dieser Ausdruck wird in die zweite Gleichung eingesetzt, was zu $2m + (3 - m) = 7$ führt. Daraus folgt $m = 4$. Die erhaltene Lösung für m wird dann in die umgestellte erste Gleichung eingesetzt, um b zu berechnen: $b = 3 - 4 = -1$. Da die Berechnungen im endlichen Körper $\mathbb{F}_{43}$ durchgeführt werden, wird das Ergebnis $\mod 43$ genommen, um das Geheimnis im Wertebereich von 0 bis $p - 1$ zu erhalten: $b = -1 \mod 43 = 42$, was dem Geheimnis $S = 42$ entspricht. Bei größeren Werten von k würde ein Polynom höheren Grades und ein entsprechend größeres lineares Gleichungssystem entstehen.

3) Rekonstruktion mit Lagrange-Interpolations-Formel: Die Lagrange-Interpolation ist das in der Praxis am häufigsten eingesetzte Verfahren zur Bestimmung des Polynoms einer bestimmten Ordnung, das durch eine gegebene Menge von Punkten verläuft. Diese Methode bietet den Vorteil, dass sie direkt eine Formel zur Rekonstruktion des Geheimnisses liefert, ohne dass ein Gleichungssystem explizit gelöst werden muss. Die Koeffizienten $m_0, ..., m_{k-1}$ eines unbekannten Polynoms f vom Grad $k - 1$ aus k Punkten (x_i, y_i) können wie folgt berechnet werden:

$$f(x) = \sum_{i=1}^{k} \left[y_i \cdot \prod_{\substack{1 \le j \le k \\ i \ne j}} \frac{x - x_j}{x_i - x_j} \right] \mod p$$

Unter Verwendung dieser Formel lässt sich $m_0 = f(0)$ und damit das Geheimnis S aus k gegebenen Punkten berechnen [4, S. 65 f.].

IV. Die Idee der Kombination

Das Ziel dieser Arbeit besteht darin, die Vorteile beider Verfahren zu kombinieren, um eine robuste und sichere Authentifizierungsmethode zu entwickeln.

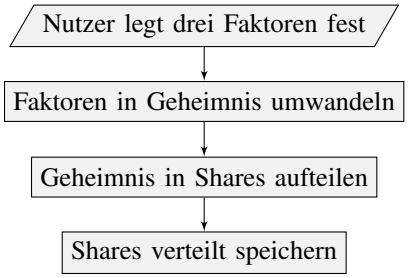

Abbildung 1. Überblick: Generierung der Shares

Abbildung 1 zeigt den groben Ablauf, wie die Shares erzeugt werden. Im Kern wird ein Geheimnis aus mehreren Authentifizierungsfaktoren generiert und mithilfe von Shamir's Secret Sharing aufgeteilt. Die zur Multi-Faktor-Authentifizierung verwendeten Faktoren entsprechen einem Passwort (Wissen), dem Fingerabdruck des Nutzers (Eigenheit) und einem Wiederherstellungsschlüssel in Form eines QR-Codes (Besitz). Die Kombination aller drei Faktoren dient als Grundlage zur Berechnung des Geheimnisses, welches anschließend mittels Shamir's Secret Sharing in drei Shares aufgeteilt wird, wobei nur zwei Stück zur späteren Authentifizierung benötigt werden. Jeder Share wird anschließend eindeutig zu einem der obigen Faktoren zugeordnet und auf unterschiedlichen Wegen sicher abgelegt, zum Beispiel der Passwort-Share in einer Datenbank, der Fingerabdruck-Share im sicheren Bereich eines Smartphones und der zugehörige Share zum Wiederherstellungsschlüssel als QR-Code ausgedruckt an einem sicheren Ort.

Nachdem alle Shares erfolgreich generiert wurden und gespeichert sind, kann sich der Nutzer mit zwei der drei zu Beginn festgelegten Faktoren authentifizieren. Mit Blick auf Abbildung 2 wählt der Nutzer zu Beginnn aus, welche zwei Faktoren er dazu verwenden möchte. Nach Eingabe eines korrekten Faktores gibt das System den verknüpften Share frei, welcher im Anschluss zur Rekonstruktion verwendet wird. Sobald beide Shares zur Verfügung stehen wird das Geheimnis wiederhergestellt und überprüft, ob der Wert dem ursprünglichen Geheimnis entspricht. Falls ja, ist die Authentifizierung erfolgreich, andernfalls erhält der Nutzer eine Fehlermeldung.

A. Vorteile

1) Erhöhte Sicherheit: MFA und SSS ergänzen sich gegenseitig, um eine robuste Sicherheitsarchitektur zu schaffen. Während MFA bereits eine zusätzliche Sicherheitsebene durch die Verwendung mehrerer Faktoren bietet, stellt Shamir's Secret Sharing sicher, dass die zu schützenden Daten

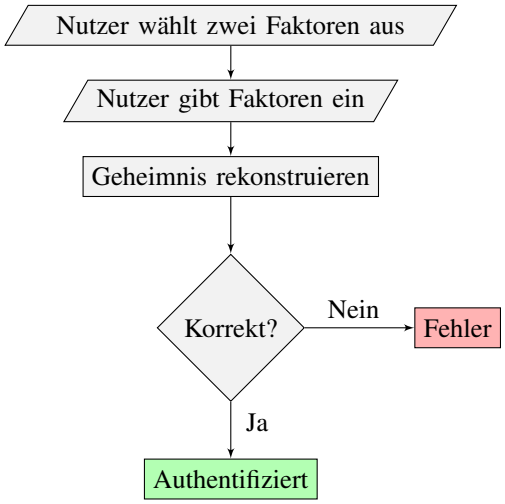

Abbildung 2. Überblick: Authentifizierung mit Shares

unverschlüsselt dezentral gespeichert werden können, da ein einzelner Share keine Rückschlüsse auf das Geheimnis zulässt. Dadurch wird das Risiko eines vollständigen Datenlecks oder unbefugten Zugriffs drastisch minimiert.

2) Flexibilität: Benutzer können aus drei verschiedenen Optionen (Passwort, Fingerabdruck und Wiederherstellungsschlüssel) zur Authentifizierung wählen. Darüber hinaus kann die Aufteilung von Shares an unterschiedliche Geräte oder Personen erfolgen, um sowohl den Bedürfnissen und Anforderungen des Nutzers als auch eines Systems gerecht zu werden.

3) Schutz vor Datenverlust: Wenn beispielsweise ein Benutzer das verknüpfte Smartphone verliert oder es irreparabel beschädigt wurde, ist weiterhin ein Zugriff über die beiden anderen Faktoren gewährleistet. Dies bietet einen zusätzlichen Schutz vor Datenverlust.

Zusammenfassend lässt sich sagen, dass das Zusammenspiel beider Methoden die Sicherheit eines Authentifizierungsprozesses erhöht. Deshalb wird im Folgenden eine mögliche Implementierung vorgestellt, welche die einzelnen Schritte im Detail verdeutlichen soll.

V. Implementierung

Das folgende Kapitel beschreibt schrittweise die praktische Umsetzung einer in Python (Version 3.10.9) programmierten Kombination von Multi-Faktor-Authentifizierung und Shamir's Secret Sharing. Dieser Prozess gliedert sich in drei Phasen: *Konstruktion des Geheimnisses*, *Generierung der Shares* und *Authentifizierung*. In der ersten Phase wird eine natürliche Zahl auf Grundlage von drei verschiedenen Faktoren konstruiert. Diese dient in Phase 2 als Geheimnis für die Anwendung von Shamir's Secret Sharing, um daraus drei Shares zu erzeugen. Für die abschließende Authentifizierung in Phase 3 werden zwei dieser Shares benötigt.

Anmerkung: Die nachfolgend präsentierten Code-Abschnitte dienen hauptsächlich der Veranschaulichung und sind ohne zusätzliche Anpassungen und Ergänzungen nicht zwingend lauffähig.

A. Phase 1: Konstruktion des Geheimnisses

Vor der Durchführung einer Authentifizierung ist es notwendig, die dafür benötigten Shares zu generieren. Als Grundlage dient hierbei ein Geheimnis, das in diesem Fall auf Basis von drei verschiedenen Authentifizierungsfaktoren erzeugt wird.

1) Nutzer legt drei Faktoren fest: Bei den Faktoren handelt es sich um ein Passwort, einen Fingerabdruck und einen Wiederherstellungsschlüssel. Wie in Listing 1 gezeigt, werden die ersten beiden Faktoren vom Nutzer bereitgestellt, während der dritte Faktor zufällig in Form eines 128 Bit langen hexadezimalen Strings generiert wird.

```
password = input() # 1. Faktor
fingerprint = input() # 2. Faktor
recovery_key = os.urandom(16).hex() # 3. Faktor
```

Listing 1. Initialisierung der drei Faktoren

2) Faktoren umwandeln: All diese Faktoren werden im späteren Verlauf für die Authentifizierung benötigt. Daher ist es wichtig, dass diese Informationen umgewandelt werden, um mögliche Rückschlüsse auf die ursprünglichen Eingaben des Nutzers auszuschließen. Aus diesem Grund werden alle Faktoren in einem weiteren Schritt mittels der SHA256-Hashfunktion umgewandelt (siehe Abbildung 3).

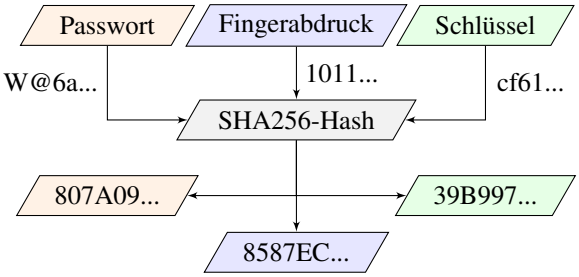

Abbildung 3. Faktoren umwandeln

Zur Realisierung im Quelltext nimmt die in Listing 2 gegebene Funktion `hash_string` einen String `value` entgegen. Dieser Wert wird zunächst mit `.encode()` als Bytes repräsentiert und unter Verwendung der `hashlib`-Bibliothek in einen SHA-256-Hash konvertiert. Nach Anwendung der Funktion auf die vom Nutzer eingegebenen Faktoren wird die `.digest()`-Methode auf den berechneten Hash angewendet, um das Ergebnis als Bytefolge zurückzugeben, um diese im nachfolgenden Schritt in eine ganze Zahl umwandeln zu können.

```
def hash_string(value):
    return hashlib.sha256(value.encode())

password_hash = hash_string(password).digest()
fingerprint_hash = hash_string(fingerprint).digest()
recovery_key_hash =
    hash_string(recovery_key).digest()
```

Listing 2. Hashen der drei Faktoren

3) Interpretation der Hashwerte als Zahlen: Alle drei erhaltenen Hashes müssen nun als Zahlen interpretiert werden, da Shamir's Secret Sharing eine ganze Zahl für das Geheimnis fordert. Die Funktion `hash_to_int` aus Listing 3 nimmt ebenfalls einen Parameter `value` entgegen, der hier die zuvor generierte Bytefolge darstellt. Durch die Verwendung der Methode `int.from_bytes()` mit dem Parameter `value` wandelt die Funktion diese Bytefolge in eine Ganzzahl um. Dabei erfolgt die Interpretation der Bytes in der Reihenfolge „big“, wodurch das Most Significant Bit zuerst und das Least Significant Bit zuletzt berücksichtigt wird. Das Ergebnis der Funktion, eine Ganzzahl, wird zurückgegeben. Anschließend wird diese Funktion auf die Hashwerte von Passwort, Fingerabdruck und Wiederherstellungsschlüssel angewendet und die Zahlen in den entsprechenden Variablen gespeichert.

```
def hash_to_int(value):
    return int.from_bytes(value, byteorder="big")

password_number = hash_to_int(password_hash)
fingerprint_number = hash_to_int(fingerprint_hash)
recovery_key_number = hash_to_int(recovery_key_hash)
```

Listing 3. Hashwerte als Zahlen interpretieren

4) Geheimnis erzeugen: Das Geheimnis ergibt sich nun durch die Aneinanderreihung aller Zahlen. Hierbei werden die Zahlen nicht addiert, sondern in zufälliger Reihenfolge konkateniert. In Listing 4 wird zuerst eine Liste `numbers` erstellt, die die Ganzzahlen aus Listing 3 enthält. Anschließend wird die Liste durch Anwendung der `shuffle`-Methode aus der Bibliothek „random“ zufällig durchmischt. Die so neu geordneten Zahlen werden in einer Schleife durchlaufen, jeder Wert in einen String umgewandelt und an den vorherigen Wert angehangen. Dieser zusammengesetzte String wird schlussendlich in einen Integer umgewandelt und als temporäre Variable `S_temp` zwischengespeichert (siehe Listing 4).

```
numbers = [password_number, fingerprint_number,
    recovery_key_number]
random.shuffle(numbers)
S_temp = int("".join(str(num) for num in numbers))
```

Listing 4. Zahlen zu Geheimnis konkatenieren

Um das Geheimnis während der Authentifizierung bei einer erfolgreichen Rekonstruktion auf Übereinstimmung prüfen zu können, muss es später abrufbar sein. Dazu wird es in Listing 5 mit SHA256 gehasht. Dadurch wird sichergestellt, dass das zu schützende Geheimnis für spätere Zwecke ohne Bedenken in einer Datenbank gespeichert werden kann.

```
S_hash = hash_string(str(S_temp)).hexdigest()
```

Listing 5. Geheimnis hashen

B. Phase 2: Generierung der Shares

Das originale, nicht gehashte Geheimnis `S_temp` wird in der zweiten Phase dazu benötigt, um es mit Hilfe von Shamir's Secret Sharing in einzelne Shares zu zerlegen.

1) Primzahl erzeugen: Alle Berechnungen erfolgen wie auch zu Beginn in der mathematischen Veranschaulichung in einem endlichen Körper. Dieser wird definiert als $\mathbb{F}_p$, wobei p eine Primzahl größer n und S ist. Die Bibliothek „libnum“ stellt die Funktion „generate_prime“ bereit, die unter Eingabe einer Bitlänge die Primzahl in dieser Größenordnung erzeugt. Zur Erzeugung einer solchen Primzahl wird zunächst die Bitlänge von S ermittelt und 8 Bit addiert, um sicherzustellen, dass die generierte Primzahl immer größer als das Geheimnis ist (siehe Listing 6). Diese Primzahl wird für die nachfolgenden Berechnungen verwendet.

```
bit_length = libnum.len_in_bits(S_temp) + 8
p = libnum.generate_prime(bit_length)

assert p > S, "Primzahl kleiner als Geheimnis"
```

Listing 6. Primzahl erzeugen

2) Shares erzeugen: Nachdem alle Vorbereitungen abgeschlossen sind, wird im letzten Schritt das Geheimnis in einzelne Shares zerlegt. Das verwendete (2, 3)-Schwellenwertschema erzeugt insgesamt drei Shares, wovon zwei zur Rekonstruktion benötigt werden. Die Funktion `create_shares(S, p)` in Listing 7 generiert eine Liste von Punkten, die für die Rekonstruktion des Geheimnisses im späteren Verlauf benötigt werden. Zu Beginn wird ein Koeffizient m erzeugt, der als Ganzzahl aus 32 zufälligen Bytes interpretiert wird. Dieser Koeffizient wird im nächsten Schritt dazu verwendet, die y-Koordinaten der Punkte zu berechnen. Dafür wird über alle x-Werte von 1 bis einschließlich 3 iteriert und der dazugehörige y-Wert über $(m*x+S) \mod p$ berechnet, wobei S das Geheimnis und p die errechnete Primzahl ist. Jeder berechnete Punkt (x_i, y_i) wird zur Liste `shares` hinzugefügt. Am Ende wird diese Liste, die die generierten Punkte beziehungsweise Shares enthält, zurückgegeben.

```
def create_shares(S, p):
    m = int.from_bytes(os.urandom(32),
        byteorder="big") # Zufälliger Koeffizient
    shares = []
    for x in range(1, 4): # x aufsteigend iterieren
        y = (m * x + S) % p # y-Wert berechnen
        shares.append((x, y)) # Punkt hinzufügen
    return shares

shares = create_shares(S, p)
```

Listing 7. Shares erzeugen

C. Phase 3: Authentifizierung

Um das Geheimnis wiederherzustellen und die Authentifizierung durchzuführen, müssen zwei Faktoren durch den Benutzer angegeben werden. Nach der vollständigen Eingabe erfolgt die Authentifizierung.

1) Auswahl der Faktoren: In diesem Schritt wird der Benutzer zur Eingabe der gewünschten Faktoren aufgefordert. Der Nutzer gibt zwei Zahlen, getrennt durch ein Leerzeichen, ein. Jede Zahl steht für einen der drei möglichen Faktoren: Passwort (1), Fingerabdruck (2) oder Wiederherstellungsschlüssel (3). Der Benutzer muss dabei zwei unterschiedliche Zahlen auswählen und jede dieser Zahlen muss entweder 1, 2 oder 3 entsprechen.

Listing 7 zeigt die Implementierung. Zuerst wird der Benutzer dazu aufgefordert, seine gewünschten Faktoren über die Tastatur einzugeben. Die Eingabe muss zwei Zeichen enthalten, die durch ein Leerzeichen getrennt sind. Jedes durch ein Leerzeichen getrenntes Zeichen wird als separates Element in einer Liste gespeichert. Als nächstes wird die Funktion `map()` verwendet, um alle Elemente in der Liste in ganze Zahlen umzuwandeln. Um die Zahlen nun in einer Liste zu speichern, wird das `map`-Objekt mit der Funktion `list()` als Liste ausgegeben. Schließlich wird mit `[:2]` der Slicing-Operator angewendet, um sicherzustellen, dass nur die ersten beiden Elemente der Liste, also die zwei vom Benutzer eingegebenen Zahlen, berücksichtigt werden. Selbst wenn der Benutzer mehr als zwei Zahlen eingibt, werden nur die ersten beiden Zahlen für weitere Verarbeitungsschritte verwendet.

Nachdem der Benutzer seine Eingabe getätigt hat, wird überprüft, ob die Eingabe korrekt ist und den erforderlichen Kriterien entspricht. Verschiedene `assert`-Anweisungen werden verwendet, um die Prüfung durchzuführen und im Falle von Fehlern entsprechende Fehlermeldungen auszugeben:

- Die erste `assert`-Anweisung stellt sicher, dass der Benutzer genau zwei Zahlen eingegeben hat. Wenn die Anzahl der eingegebenen Zahlen nicht genau zwei beträgt, wird eine Fehlermeldung mit dem Text „Bitte verwende genau 2 Zahlen“ angezeigt.
- Die zweite `assert`-Anweisung überprüft, ob alle eingegebenen Zeichen entweder 1, 2 oder 3 entsprechen. Falls eine der eingegebenen Zahlen nicht zu den erlaubten Werten gehört, wird eine Fehlermeldung mit dem Text „Bitte verwende nur 1, 2 oder 3“ ausgegeben.
- Die dritte `assert`-Anweisung garantiert, dass der Benutzer zwei unterschiedliche Zahlen eingegeben hat. Wenn die beiden eingegebenen Zahlen identisch sind, wird eine Fehlermeldung mit dem Text „Bitte verwende zwei unterschiedliche Zahlen“ angezeigt.

```
factors = list(map(int, input().split()))[:2]

assert len(factors) == 2
assert all(factor in [1,2,3] for factor in factors)
assert factors[0] != factors[1]
```

Listing 8. Faktoren auswählen

2) Eingabe der Faktoren: In diesem Abschnitt liegt der Fokus auf der Aufforderung an den Benutzer, die ausgewählten Authentifizierungsfaktoren einzugeben. Listing 9 zeigt die Variable `num_factors_map`, ein Wörterbuch, dass den Zahlen 1, 2 und 3 die entsprechenden Werte für Passwort, Fingerabdruck und Wiederherstellungsschlüssel zuweist. Dieses Wörterbuch wird später verwendet, um die vom Benutzer eingegebenen Faktoren mit den gespeicherten Werten abzugleichen.

```
num_factors_map = {
    1: password,
    2: fingerprint,
    3: recovery_key
}
```

Listing 9. Zahlen zu Faktoren zuweisen

Die in Listing 10 definierte for-Schleife ermöglicht es, über die Liste der vom Benutzer ausgewählten Faktoren zu iterieren. Je nachdem, welche Zahlen der Nutzer im letzten Schritt gewählt hat, wird er nun dazu aufgefordet, den dazugehörigen Wert einzugeben. Falls die Faktoren beispielsweise den Werten 1 und 3 entsprechen, muss der Benutzer nacheinander sein zu Beginn festgelegtes Passwort (1) und den Wiederherstellungsschlüssel (3) eingeben. Anschließend erfolgt eine Überprüfung mit `if user_input == num_factors_map[i]`, um festzustellen, ob die Eingabe des Benutzers mit dem gespeicherten Wert des entsprechenden Authentifizierungsfaktors übereinstimmt. Ist dies der Fall, wird der zugehörige Share zur Liste `user_shares` hinzugefügt. Wenn die Eingabe nicht übereinstimmt, wird eine Fehlermeldung angezeigt und der Authentifizierungsprozess abgebrochen.

```
user_shares = []

for i in factors:
    if i == 1:
        print("\nBitte Passwort eingeben:")
    elif i == 2:
        print("\nBitte Fingerabdruck eingeben:")
    elif i == 3:
        print("\nBitte Wiederherstellungsschlüssel
            eingeben:")

    user_input = input()

    # Prüfung auf Übereinstimmung
    if user_input == num_factors_map[i]:
        # Eingabe stimmt mit ursprünglichem Wert
            überein, dazugehöriges Share zu Liste
            hinzufügen
        user_shares.append(shares[i - 1])
```

Listing 10. Eingabe der Faktoren

3) Geheimnis rekonstruieren: Mit den erhaltenen Shares kann das Geheimnis nun rekonstruiert werden.

Die Funktion `lagrange` in Listing 11 berechnet den spezifischen Lagrange-Koeffizienten für den gegebenen Punkt i in der Liste der x-Werte `x_values` im Körper $\mathbb{F}_p$.

Die innere Schleife durchläuft alle x-Werte in `x_values` und multipliziert das bisherige Ergebnis mit dem Kehrwert des Differenzterms $x_i - x_j$. Diese Kehrwerte werden modulo p berechnet, um im endlichen Körper zu bleiben. Hierbei wird `pow((x_values[i] - x_values[j]), p-2, p)` verwendet, um das multiplikative Inverse unter Modulo p zu berechnen (basierend auf dem kleinen Satz von Fermat). Um den Wert des Lagrange-Koeffizienten für $x = 0$ zu berechnen, wird in Listing 11 `0 - x_values[j]` geschrieben.

Wenn alle Multiplikationen durchgeführt sind, steht in `result` der Lagrange-Koeffizient für den Punkt i. Diese Koeffizienten werden dann verwendet, um eine Polynomfunktion zu erstellen, die zur Wiederherstellung des ursprünglichen Geheimnisses verwendet wird.

```
def lagrange(i, x_values):
    result = 1
    for j in range(len(x_values)):
        if j != i:
            result = (result * (0 - x_values[j]) *
                pow((x_values[i] - x_values[j]),
                p-2, p)) % p
    return result
```

Listing 11. Lagrange-Interpolation

Die in Listing 12 definierte Funktion `reconstruct_shares(shares, p)` nimmt als Argumente eine Liste von Shares und eine Primzahl zur Berechnung des Geheimnisses entgegen. Zunächst werden die x-Werte aus den Punkten extrahiert, welche zur Berechnung der Lagrange-Koeffizienten verwendet werden. Die Funktion durchläuft nun alle Shares. Für jeden Share wird das Produkt aus dem y-Wert des jeweiligen Shares und dessen Lagrange-Koeffizienten berechnet und zu dem Geheimnis addiert. Abschließend wird das Geheimnis von der Funktion zurückgegeben. Der Rückgabewert entspricht dem konstanten Term des rekonstruierten Polynoms und somit dem ursprünglich geteilten Geheimnis.

```
def reconstruct_secret(shares, p):
    x_values = [share[0] for share in shares]
    secret = 0

    for i in range(len(shares)):
        secret = (secret + shares[i][1] *
            lagrange(i, x_values)) % p

    return secret

reconstructed_S = reconstruct_secret(user_shares, p)
```

Listing 12. Geheimnis rekonstruieren

4) Prüfung auf Korrektheit: Auf das rekonstruierte Geheimnis wird im letzten Schritt die Hashfunktion SHA256 angewendet, um es mit dem Hashwert des ursprünglichen Geheimnisses vergleichen zu können. In Listing 13 erfolgt im if-else-Block die Prüfung, ob beide ermittelten Hashwerte übereinstimmen. Wenn dies der Fall ist, bedeutet das, dass die Authentifizierung erfolgreich war.

```
reconstructed_S_hash =
    hash_string(str(reconstructed_S)).hexdigest()

if reconstructed_S_hash == S_hash:
    success_print("Authentifizierung erfolgreich.")
else:
    raise Exception("Rekonstruktion nicht möglich.")
```

Listing 13. Hashwerte überprüfen

VI. Relevante Sicherheitsaspekte

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

VII. Benchmarks

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

VIII. Anwendung in der digitalen Welt

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Literatur

[1] Gen Digital Inc., „2023 Norton Cyber Safety Insights Report," Feb. 2023. Adresse: https://filecache.mediaroom.com/mr5mr_nortonlifelock/178041/2023%20NCSIR%20US-Global%20Report_FINAL.pdf (besucht am 05. 06. 2023).

[2] Morning Consult und IBM Security, „Security Side Effects of the Pandamic," Juli 2021. Adresse: https://filecache.mediaroom.com/mr5mr_nortonlifelock/178041/2023%20NCSIR%20US-Global%20Report_FINAL.pdf (besucht am 05. 06. 2023).

[3] A. Shamir, „How to share a secret," *Communications of the ACM*, Jg. 22, Nr. 11, S. 612–613, 1. Nov. 1979, ISSN: 0001-0782. DOI: 10.1145/359168.359176. Adresse: https://dl.acm.org/doi/10.1145/359168.359176 (besucht am 20. 06. 2023).

[4] Bundesamt für Sicherheit in der Informationstechnik, „Kryptographische Verfahren: Empfehlungen und Schlüssellängen," BSI TR-02102-1, 9. Jan. 2023. Adresse: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf (besucht am 25. 06. 2023).