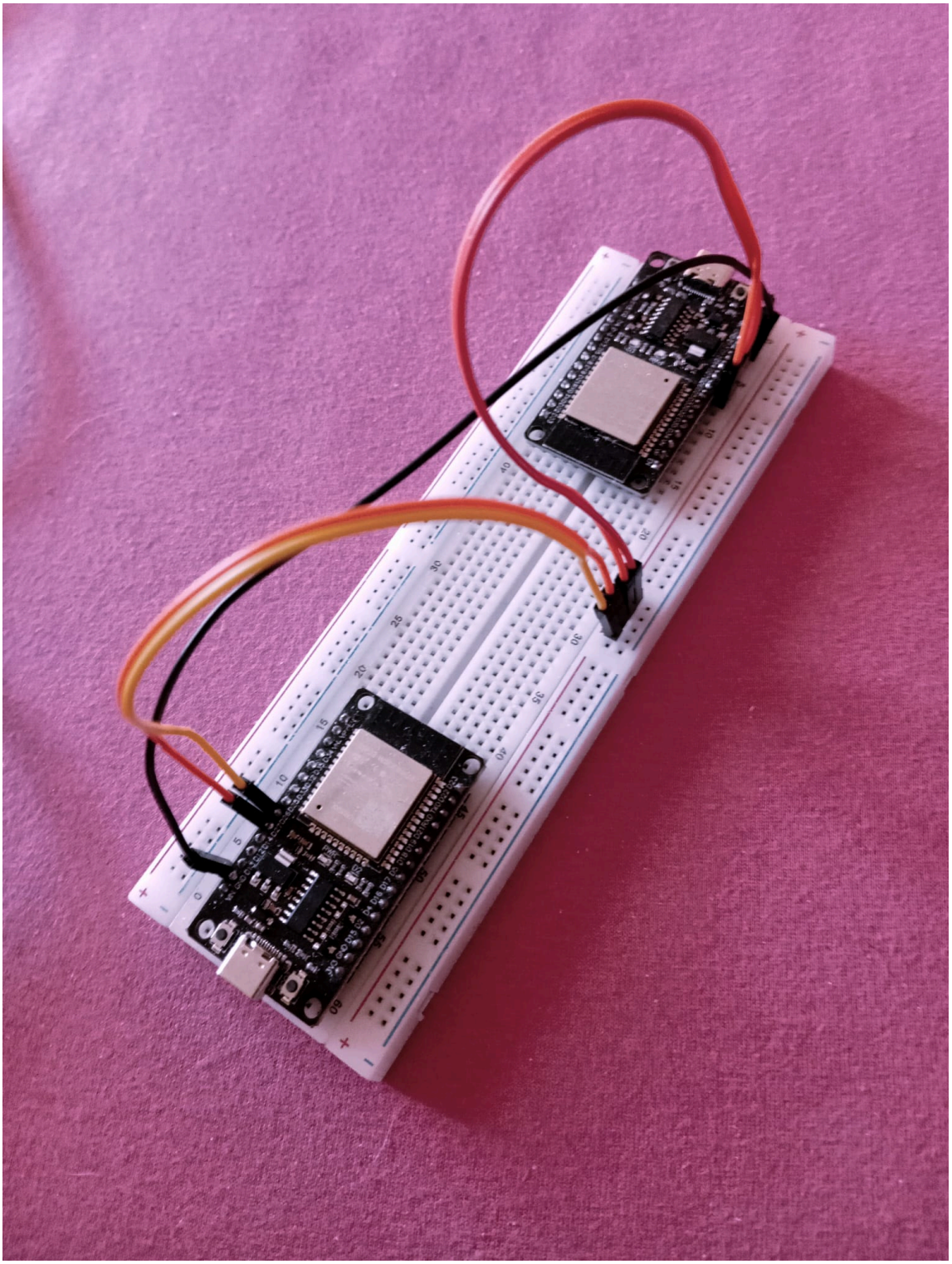
MW-Dezentrales Netzwerk- und Kommunikationsprotokoll



Inhalsverzeichnis

1 Einleitung

1.1 Fachliche Kurzfassung

Dies ist ein kryptographisches, verschlüsseltes und dezentrales **Netzwerkprotokoll** für die Kommunikation zwischen Computern, das mehrere Untergruppen unterstützt. Die Technologie kann sowohl für Kabelverbindungen als auch für 433 Mhz Radiofunkmodule verwendet werden. Generell funktioniert das Protokoll auf jedem geteilten Medium (auch z.B. Licht, Laser usw.), das zwei verschiedene Zustände hat.

1.2 Motivation und Fragestellung

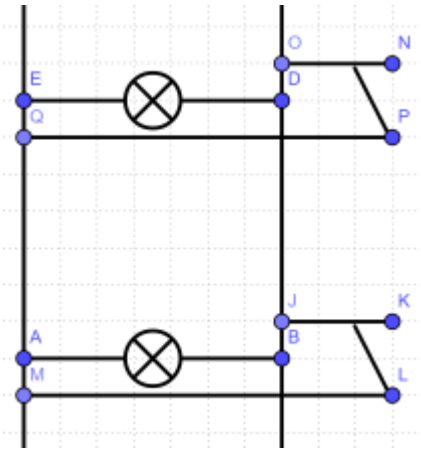
Die zunehmende Nachfrage nach Kommunikationsprotokollen mit quantenresistenter Verschlüsselung für drahtlose und kabelgebundene Netzwerke verlangt nach effizienten und skalierbaren Lösungen. Wie kann ein robustes, kryptografisch sicheres Protokoll für den Einsatz in einfachen, aber vielseitigen Hardwareplattformen realisiert werden? Diese Arbeit bietet eine Antwort durch die Entwicklung eines Protokolls, das sowohl Datenintegrität als auch Vertraulichkeit sicherstellt.

2 Hauptteil

2.1 Vorgehensweise, Materialien und Methoden

- Hardware:
  - Ein Mikrocontroller (ESP-32) für die Signalverarbeitung.
  - Ein Kabel wird zur Datenübertragung benötigt (plus Erdungskabel).
- Software:
  - Implementierung der Protokollregeln in C++ (Platform IO mit dem Arduino-Framework).

Schaltkreis skizze: (Lampe = GPIO Pin (input); Schalter = GPIO Pin (output))



2.2 Hintergrund und theoretische Grundlagen

- **Netzwerk-Hierarchie**: Einteilung in virtuelle Gruppen zur besseren Skalierbarkeit und Sicherheit.
- **Hashing**: Ein Hash ist eine Einwegfunktion, die bei demselben Input immer denselben Output ergibt. Von dem Output kann aber kein Input errechnet werden. Außerdem verändert sich der Output selbst bei kleinen Veränderungen stark. Alle Pakete enthalten einen oder mehere Hashs, um Fehler bei der Datenübertragung des Pakets zu finden.
- **Signierung**: Alle Pakete in einer Gruppe enthalten eine Signierung, um zu prüfen, ob es wirklich von dem richtigen Benutzer gesendet wurde.
- **Salt**: Eine Zusatzdatenmenge zu dem Verschlüsselungsschlüssel, der Mengenanalysen von verschlüsselten Daten erschwert bis hin zu unmöglich macht.
- **Verschlüsselung**: Sensible Datenfelder werden mit einer Kombination aus Passwort und Salt verschlüsselt.
- **Binäre Zahlen**: Ein Zahlensystem das nur mit den Ziffern 1 und 0 arbeitet. In diesem Fall Strom an (`HIGH`) als 1 und Strom aus als 0 (`LOW`).

2.3 Signalzustände

- Die Verbindung kann entweder auf `HIGH` (aktiv/Strom fließt) oder `LOW` (inaktiv/Strom fließt nicht) gesetzt werden.
- Die Zustände können auch Binärziffern darstellen, die zu Binärzahlen zusammengesetzt werden.
- Der Zustand wird in einem Intervall (delayTime (z.B. ***50 Mikrosekunden***) Zeit pro Bit (BAUD-Rate)) geändert.

2.4 Paketformat

Das Netzwerkprotokoll teilt die Daten, die über die Leitung gesendet werden, in Byte-Pakete (8 Bits/8 "Einsen und Nullen"), Chunks und Pakete ein. Dies sind virtuelle Einteilungen, die zum Verständnis des Protokolls wichtig sind.

Pakete: Jedes Paket folgt einem strukturierten Format:

Pakete setzen sich aus Chunks (kleinere Einheiten des Pakets mit Namen, Wert und Länge) zusammen. Diese werden in eckigen Klammern visualisiert.

- `[HIGH]` : Markiert den Beginn eines Pakets. Es ist immer ein 1 Bit Chunk.
- `[FUNCTION=x|1B]` : Ein fester 1-Byte-Chunk, der den Zweck des Pakets definiert.
- Weitere Felder sind im Format `[NAME=VALUE|LENGTH]` angegeben:
  - **NAME**: Name des Chunks.
  - **VALUE**: Wert des Chunks, dessen Standardwert oder Nichts.
  - **LENGTH**: Feldgröße, angegeben als `xB` (Bytes) oder `xBit` (Bits). Hier können auch vorherige Chunk-Variablen verwendet werden.
  - Verschlüsselte Werte werden als `VALUE x (PASSWORD + SALT)` angegeben.
  - Felder können aus verketteten Chunks bestehen.
- `[LOW]` : Markiert das Ende des Pakets. Es ist immer ein 1 Bit Chunk, der die Leitung insgesamt auf `LOW` setzt.

Beispiel:

`[HIG] [FN=100|1B] [CHUNK1|2B] [DATA_LENGTH|1B] [DATA|DATA_LENGTH*1B] [LOW]`

`Start, Funktion 100, 2Bytes wert mit nicht definierten Nutzen, Dantenlänge (1Byte, 0-255), Daten (so lang wie der wert von DATA_LENGTH mal 1Byte), Ende`

2.5 Grundlegende Datenübertragung

Byte-Paket: Ermöglicht die Übertragung eines Bytes (8 Bit) zusammen mit der isFollowing-Flag (1 Bit). Die isFollowing-Flag ist auf 0 gesetzt, wenn das Byte-Paket den Anfang einer Übertragung markiert, und auf 1, wenn das Byte-Paket einem vorherigen Paket folgt.

Sender

In den eckigen Klammern ist der Wert. Dieser Wert zeigt den Zustand (`HIGH` / `LOW`). `HIGH` steht für "ja" oder "1", `LOW` steht für "nein" oder 0.

- Am Anfang wird die "Leitung" auf `HIGH` gesetzt, was den Start des Bytepakets kennzeichnet.
- Am Ende wird die "Leitung" auf `LOW` gesetzt, was dafür sorgt, dass die Leitung, bei dem nächsten Paket, am Anfang, wieder auf `HIGH` gesetzt werden kann (Zustandsänderung).

Einfache Darstellung: `[NAME]` dauert ein Zeitintervall und speichert ein Bit (delayTime/50Microsekunden)

`[HIGH] [IS_FOLLOWING] [BIT_8] [BIT_7] [BIT_6] [BIT_5] [BIT_4] [BIT_3] [BIT_2] [BIT_1] [LOW]`

Beispiel: die Zahl 129 (binär: 10000001) und isFollowing mit dem Wert ja übertragen:

`[HIGH] 110000001 [LOW]`

`[HIGH] [HIGH] [HIGH] [LOW] [LOW] [LOW] [LOW] [LOW] [LOW] [HIGH] [LOW]`

```
//WF=With isFollowingFlag
void rawSendByteWF(uint8_t value, int pin, int delayTime, bool isFollowing)
{
    // Beginn des Pakets
    digitalWrite(pin, HIGH);
    delayMicroseconds(delayTime); // Pause

    // das erste Bit zeigt, ob das Paket auf ein anders Paket folgt oder der Start
eines neuen Pakets ist
    digitalWrite(pin, isFollowing ? HIGH : LOW);
    delayMicroseconds(delayTime); // Pause

    // Jedes Daten-Bit eines Bytes (8 Bits) senden (LSB-first)
    for (uint8_t i = 0; i < 8; i++)
    {
        // Jedes Bit extrahieren und dann senden
        bool bit = (value & (1 << i)) != 0;
        digitalWrite(pin, bit ? HIGH : LOW);

        // Pause
        delayMicroseconds(delayTime);
    }

    // Ende des Pakets
    digitalWrite(pin, LOW);
```

```
    delayMicroseconds(delayTime); // Pause
}
```

Empfänger

```
// Datentyp zur Vereinfachung
struct RawPacket
{
    bool isFollowing;
    uint8_t data;
};

//WF=With isFollowingFlag
RawPacket rawReadByteWF(uint8_t pin, int delayTime)
{
    RawPacket packet;
    packet.data = 0;

    // Auf das Startsignal warten
    while (digitalRead(pin) != HIGH)
        ;

    // 1.5 * delayTime (50 Microsekunden) warten (damit es bei der Hälfte des
nächsten Bits anfängt den Wert auszulesen)
    delayMicroseconds(delayTime * 1.5);

    // isFollowingFlag auslesen
    packet.isFollowing = (digitalRead(pin) == HIGH);
    delayMicroseconds(delayTime);

    // Jedes Bit auslesen und zu einem Byte zusammensätzen
    for (uint8_t i = 0; i < 8; i++)
    {
        if (digitalRead(pin) == HIGH)
        {
            packet.data |= (1 << i); // LSB-first
        }
        delayMicroseconds(delayTime);
    }

    return packet;
}
```

2.6 Paketübertragungsregeln

1. **Startbedingungen**:
    - Um ein Paket zu senden, stellt der Sender sicher, dass die Verbindung für eine festgelegte "maximale Sendezeit" (13 `*` delayTime) auf `LOW` bleibt. Da pro Byte-Paket mindestens einmal der Zustand `HIGH` übertragen werden muss (am Start) und das Senden eines Byte-Pakets ca. 12 `*` delayTime + 1 `*` delayTime (Puffer) dauert, kann man sagen, dass, wenn die Leitung 13 `*` delayTime (50 Microsekunden) lang auf `LOW` bleibt, nichts gesendet wurde und der Nutzer mit dem sicheren "Paket senden" anfangen kann.
    - Wenn die Verbindung während dieses Zeitraums auf `HIGH` wechselt, muss der Sender den Versuch wiederholen.

Code Beispiel

```
void waitForBytePacketEnd()
{
    // den Zeitpunkt, an dem (connection.sendDelay * 13) lange nichts gesendet
wurde, was heißt, dass das letzte Paket zu Ende ist.
    auto timeToWait = micros() + (connection.sendDelay * 13);
    while (true)
    {
        auto now = micros();
        //wenn lange genug gewartet wurde, returnt die Funktion und der code
dahinter kann reibungslos ausgeführt werden.
        if (now > timeToWait)
        {
            return;
        }
        // wenn doch etwas gesendet wird, wird der timer zurückgesetzt
        else if (digitalRead(connection.inpPin) == HIGH)
        {
            auto timeToWait = now + (connection.sendDelay * 13);
        }
    }
}
```

2. **Kollisionsvermeidung**: Geräte verwenden die Zeit seit der letzten Paketübertragung, um für eine zufällige Zeitspanne zwischen (sendDelay `*` 13) und (500 `*` sendDelay) Mikrosekunden zu warten, bevor sie versuchen, die Verbindung auf `HIGH` zu setzen.
    - Wenn die Leitug auf `HIGH` wechselt, muss der Benutzer am Ende des von einem anderen gesendeten Paket, den Versuch wiederholen. Wenn der Sender mehrmals versucht ein Paket zu senden, wird die maximale Zufallszeit verkürzt, so dass es wahrscheinlicher wird, das Paket als nächstes zu senden.
    - Bleibt die Leitung, bis die zufällige Wartezeit vorbei ist, auf `LOW`, kann der Sender die Leitung auf `HIGH` setzen und mit der Übertragung des Pakets fortfahren.

Code Beispiel

```
bool waitForBytePacketToSend()
{
    // den Zeitpunkt, an dem (connection.sendDelay * 13) lange nichts gesendet
wurde + ein zufallsintervall zwischen 0 und ((500 - (sucessFactor * 5)) *
connection.sendDelay)
    auto sucessFactor = messagesNotSend.size() + trysFailed;
    auto maxRandomSendDelay = (500 - (sucessFactor * 5)) * connection.sendDelay;
    auto timeToWait = micros() + (connection.sendDelay * 13) +
random(maxRandomSendDelay > 0 ? maxRandomSendDelay* : 0);
    while (true)
    {
        auto now = micros();
        // wenn lange genug gewartet wurde, wird das Paket gesendet, die Leitung
auf "HIGH" gesetzt und die Funktion returnt true (= geglückt).
        // durch das Setzen auf "HIGH" wird den anderen Benutzern mitgeteilt, dass
sie ihre Pakete nicht mehr senden Können.
        if (now > timeToWait)
        {
            digitalWrite(connection.outPin, HIGH);
            trysFailed = 0;
            return true;
        }
        // wenn doch etwas gesendet (die Leitung auf "HIGH" gesetzt wird) wird,
false = nicht geglückt returnt.
        else if (digitalRead(connection.inpPin) == HIGH)
        {
            trysFailed++;
            return false;
        }
    }
}

// paket senden
void send(const RawPacket &packet) {
  waitForBytePacketToSend();
  sendPacket(packet);
}
```

2.7 Netzwerk-Hierarchie

Das **NETZWERK** ist die physische Verbindung, die mit einer Kabelverbindung, 433 MHz RF-Modulen oder einer physischen Verbindung jeder Art, die mindestens zwei Zustände aufweisen muss, (z.B. Licht an/aus) hergestellt wird.

Die **GRUPPEN** sind virtuelle Netzwerke, die eine Verschlüsselung für eine sichere Kommunikation implementieren. Es kann bis zu **65.536 GRUPPEN** geben. Benutzer innerhalb einer GRUPPE können den Verbindungsprozess verwalten, Daten senden oder bis zu **65.536 BENUTZER** einladen.
Jeder Benutzer ist Mitglied einer oder mehrerer GRUPPEN und kann gleichzeitig mit mehreren Netzwerken verbunden sein. In jeder der Gruppen ist jeder Benutzer gleichberechtigt.

Benutzer können folgende Aktionen durchführen:

- Verschlüsselte Nachrichten an andere Benutzer innerhalb der GRUPPE senden.
- Nachrichten an alle Mitglieder der GRUPPE broadcasten.
- Nachrichten an das gesamte NETZWERK senden.
- Nachrichten im gesamten NETZWERK an MAC-Adressen senden.
- Auf Pakete antworten.

Übersicht der Hierarchie

```
NETZWERK (Kabelverbindung / 433Mhz Radiofunk / Verbindung)
  ├── GRUPPE 1
  │      ├── BENUTZER 1
  │      ├── BENUTZER 2
  │      └── BENUTZER 3
  │          ...
  ├── GRUPPE 2
  │      ├── BENUTZER 27224
  │      └── BENUTZER 2885
  │          ...
  └── GRUPPE 3
         ├── BENUTZER 6
         ├── BENUTZER 7
         ├── BENUTZER 8
         └── BENUTZER 9
             ...
  ...
```

Diese Hierarchie gewährleistet eine strukturierte Organisation der Benutzer innerhalb sicherer und skalierbarer virtueller GRUPPEN, unterstützt durch ein robustes physisches NETZWERK.

2.8 Signierung

1. **Generierung eines zufälligen Werts**
   Der Benutzer generiert einen zufälligen 4-Byte-Wert, bezeichnet als `SIGN_VALUE`.

2. **Hashen des Werts**
   Der Benutzer hasht den Wert `SIGN_VALUE` und extrahiert die letzten 4 Bytes des Hashs, bezeichnet als `SIGN_VALUE_HASH`.

3. **Teilen des Hashs**
   Der Wert von `SIGN_VALUE_HASH` wird mit allen Mitgliedern der Gruppe geteilt.

4. **Signieren einer Nachricht**
   Um eine Nachricht zu signieren, sendet der Benutzer:

   - den ursprünglichen zufälligen Wert (`SIGN_VALUE`)
   - den nächsten gehashten zufälligen Wert (`NEXT_SIGN_VALUE_HASH`), um die nächste Nachricht auch signieren zu können. (Diese muss im gleichen Paket gesendet werden.)

Benutzer, die überprüfen wollen, ob eine Nachricht von dem richtigen Benutzer gesendet wurde, können den Hash des gesendeten Werts `SIGN_VALUE` mit dem Hash `SIGN_VALUE_HASH` abgleichen. Wenn die beiden Hashes äquivalent sind, ist der Sender sicher der echte Sender.

Da eine Hashfunktion eine Einwegfunktion ist, kann kein übereinstimmender Wert ausgängig von dem Hash generiert werden (außer durch Raten). Dies stellt sicher, dass jede Nachricht eindeutig verifizierbar ist.

Das Format lautet:

1. Erstes Paket: (`... [CURRENT_SIGN_HASH|4B] ...`) (Dies zeigt den Nutzern den aktuellen Hash).
2. Folgende Pakete: (`... [LAST_SIGN_VALUE|4B] [CURRENT_SIGN_HASH|4B] ...`) (Kann ein Paket "unterschreiben" und bringt den Hash für das nächste Paket mit).

2.9 Pakettypen

1. **LF_HASH**: der Hash aus Längenvariablen und der Funktion `[2Bit Funktionshash + 6Bit Längenhash]`
2. **HASH**: der Hash aus dem gesamten Paket

Autorisieren

1. IS HERE

Wird verwendet, um Gruppen zu finden.

```
[HIGH] [FUNCTION=1|1B] [GROUP_NAME_LENGTH=L|1B] [LF_HASH|1B] [PACKET_ID=random()|2B] [ANSWER_ID=random()|2B] [GROUP_NAME_STRING=...|L*1B] [CURRENT_SIGN_HASH|4B] [HASH|4B] [LOW]
```

2. HERE IS

JA

Bestätigt die Existenz der Gruppe.

Alle Benutzer in der Gruppe können auf dieses Paket antworten, aber der erste Benutzer in der Gruppe (bestimmt durch die kürzeste zufällige Verzögerungszeit) sendet die Antwort.

```
[HIGH] [FUNCTION=2|1B] [LF_HASH|1B] [PACKET_ID=random()|2B] [ANSWER_ID|2B] [GROUP_ID|2B] [CONNECT_ID|2B] [VERIFY_BYTES=ranom()|4B] [SALT=random()|2B] [HASH|4B] [LOW]
```

NEIN

Keine Antwort, wenn die Gruppe nicht vorhanden ist (Timeout: 1-2 Sekunden).

3. JOIN

Anfrage zum Beitreten einer Gruppe.

```
[HIGH] [FUNCTION=3|1B] [LF_HASH|1B] [PACKET_ID=random()|2B] [GROUP_ID|2B] [CONNECT_ID|2B] [VERIFY_BYTES x (PASSWORD + SALT)|4B] [LAST_SIGN_VALUE|4B] [CURRENT_SIGN_HASH|4B] [HASH|4B] [LOW]
```

4. ACCEPT

Antwort auf eine Beitrittsanfrage.

Alle Benutzer in der Gruppe können auf dieses Paket antworten, aber der erste Benutzer in der Gruppe (bestimmt durch die kürzeste zufällige Verzögerungszeit) sendet die Antwort.

JA

Bestätigt die Existenz der Gruppe.

```
[HIGH] [FUNCTION=4|1B] [LF_HASH|1B] [PACKET_ID=random()|2B] [GROUP_ID|2B] [CONNECT_ID|2B] /* Verschlüsselte Daten beginnen hier */ [NEW_USER_ID|2B] [GLOBAL_CONSTANT=111] [CURRENT_SALT|4B] [ERROR_IDENTIFYER=random()|2B] [SALT_MODIFIER_PER_PACKET=(MODIFIER + VALUE)|2B] [HASH|4B] [LOW]
```

NEIN

Keine Antwort, wenn die Gruppe nicht vorhanden ist (Timeout: 1-2 Sekunden).

5. JOINED

Bestätigt den erfolgreichen Beitritt zur Gruppe.

```
[HIGH] [FUNCTION=5|1B] [LF_HASH|1B] [PACKET_ID=random()|2B] [GROUP_ID|2B] /*
Verschlüsselte Daten beginnen hier */ [ERROR_IDENTIFYER|2B] [USER_ID|2B]
[CURRENT_SALT|4B] [LAST_SIGN_VALUE|4B] [CURRENT_SIGN_HASH|4B] [HASH|4B] [LOW]
```

Erhalten der Signaturen der Benutzer in der Gruppe

6. WHO IS IN THE GROUP

Fragt, wer in der Gruppe ist.

```
[HIGH] [FUNCTION=6|1B] [LF_HASH|1B] [PACKET_ID=random()|2B] [GROUP_ID|2B] /*
Verschlüsselte Daten beginnen hier */ [USER_ID|2B] [CURRENT_SALT|4B]
[LAST_SIGN_VALUE|4B] [CURRENT_SIGN_HASH|4B] [HASH|4B] [LOW]
```

7. I AM IN THE GROUP

Meldet, dass man in der Gruppe ist.

```
[HIGH] [FUNCTION=7|1B] [LF_HASH|1B] [PACKET_ID=random()|2B] [GROUP_ID|2B] /*
Verschlüsselte Daten beginnen hier */ [USER_ID|2B] [LAST_SIGN_VALUE|4B]
[CURRENT_SIGN_HASH|4B] [HASH|4B] [LOW]
```

8. WRONG SIGN

Weist darauf hin, wenn ein Benutzer den falschen Signatur-Hash sendet.

```
[HIGH] [FUNCTION=8|1B] [LF_HASH|1B] [PACKET_ID=random()|2B] [GROUP_ID|2B] /*
Verschlüsselte Daten beginnen hier */ [USER_ID|2B] [USER_WITH_WRONG_SIGN_ID|2B]
[WRONG_SIGN_PACKET_ID|2B] [LAST_SIGN_VALUE|4B] [CURRENT_SIGN_HASH|4B] [HASH|4B] [LOW]
```

9. WRONG SIGN PACKET IS CORRUPTED

Weist darauf hin, wenn ein Hacker fälschlicherweise behauptet, ein Benutzer habe eine falsche Signatur, die jedoch gültig ist. Diese kann nur durch einen anderen Benutzer gesendet werden (nicht HACKER).

```
[HIGH] [FUNCTION=9|1B] [LF_HASH|1B] [PACKET_ID=random()|2B] [GROUP_ID|2B] /*
Verschlüsselte Daten beginnen hier */ [USER_ID|2B] [HACKER_USER_WITH_WRONG_SIGN_ID|2B]
[WRONG_SIGN_PACKET_ID|2B] [LAST_SIGN_VALUE|4B] [CURRENT_SIGN_HASH|4B] [HASH|4B] [LOW]
```

Senden

10. SEND

Sendet Daten an einen bestimmten Benutzer.

```
[HIGH] [FUNCTION=10|1B] [DATA_LENGTH=L|1B] [LF_HASH|1B] [PACKET_ID=random()|2B]
[GROUP_ID|2B] /* Verschlüsselte Daten beginnen hier */ [USER_ID|2B]
[USER_DESTINATION|2B] [LAST_SIGN_VALUE|4B] [CURRENT_SIGN_HASH|4B] [HASH|6B]
[DATA|L*1B] [LOW]
```

11. SEND TO MULTIPLE USERS

Sendet Daten an mehrere spezifische Benutzer.

```
[HIGH] [FUNCTION=11|1B] [USERS_LENGTH=UL|2B] [DATA_LENGTH=DL|1B] [LF_HASH|1B]
[PACKET_ID=random()|2B] [GROUP_ID|2B] /* Verschlüsselte Daten beginnen hier */
[USER_ID|2B] [LAST_SIGN_VALUE|4B] [CURRENT_SIGN_HASH|4B] [HASH|6B]
[USER_DESTINATIONS|UL*2B] [DATA|DL*1B] [LOW]
```

12. BROADCAST INNER GROUP

Sendet Daten innerhalb einer Gruppe an alle Mitglieder.

```
[HIGH] [FUNCTION=12|1B] [DATA_LENGTH=L|1B] [LF_HASH|1B] [PACKET_ID=random()|2B]
[GROUP_ID|2B] /* Verschlüsselte Daten beginnen hier */ [USER_ID|2B]
[LAST_SIGN_VALUE|4B] [CURRENT_SIGN_HASH|4B] [HASH|6B] [DATA|L*1B] [LOW]
```

20. BROADCAST INNER NETWORK

Sendet Daten an alle Geräte im Netzwerk.

```
[HIGH] [FUNCTION=20|1B] [DATA_LENGTH=L|1B] [LF_HASH|1B] [PACKET_ID=random()|2B]
[HASH|6B] [DATA|L*1B] [LOW]
```

21. SEND TO MAC INNER NETWORK

Sendet eine Nachricht an einen Benutzer im Netzwerk über die MAC-Adresse.

```
[HIGH] [FUNCTION=21|1B] [DATA_LENGTH=L|1B] [LF_HASH|1B] [PACKET_ID=random()|2B]
[MAC_ADRESS|4B] [HASH|6B] [DATA|L*1B] [LOW]
```

30. PACKET DATA ERROR

Wenn der Hash nicht mit den gesendeten Daten übereinstimmt, wird dieses Paket gesendet.

```
[HIGH] [FUNCTION=30|1B] [LF_HASH|1B] [PACKET_ID=random()|2B]
[ERROR_PACKET_ID=random()|2B] [HASH|1B] [LOW]
```

31. PACKET IS CORRUPTED

Als Kennzeichen, wenn ein Paket von einem Hacker gesendet wurde. Dieses Paket kann auf jedes Paket folgen, es macht Pakete rückwirkend ungültig.

```
[HIGH] [FUNCTION=31|1B] [LF_HASH|1B] [PACKET_ID=random()|2B]
[HACKER_PACKET_ID=random()|2B] [HASH|1B] [LOW]
```

32. PACKET RECIEVED

Dieses Paket kann erst gesendet werden, nachdem ein vorheriges Paket fehlerfrei übermittelt und erfolgreich empfangen wurde.

```
[HIGH] [FUNCTION=32|1B] [LF_HASH|1B] [PACKET_ID=random()|2B]
[RCV_PACKET_ID=random()|2B] [HASH|1B] [LOW]
```

3 Schluss und Ergebnisse

3.1 Ergebnisse

1. **Implementierung des Unterprotokolls**: Das Unterprotokoll zur Übertragung von roh-binären Daten konnte erfolgreich umgesetzt werden. Dabei wurden die Zustände HIGH und LOW zur Darstellung der Binärwerte 1 und 0 verwendet. Die Übertragung zeigte in ersten Tests eine stabile Kommunikation zwischen Sender und Empfänger.

2. **Hashing, Verschlüsselung und Signaturprüfung**: Die theoretischen Grundlagen zur Sicherstellung der Integrität und Authentizität von Nachrichten durch Hash-Signaturprüfung wurden niedergeschrieben.

3. **Übertragungsrate und Fehlererkennung**: Erste Tests der Datenübertragung ergaben eine zuverlässige Erkennung von Signalstörungen, insbesondere bei "verrauschtem" Eingangssignal. Hierzu trugen sowohl die minimalen Zustandswechsel als auch die Verwendung eines Protokollrahmens für Fehlerkorrektur bei.

4. **Dezentralisierte Gruppenzuweisung**: Die Datenstruktur der Gruppenkommunikation zeigte sich als skalierbar für mehrere Gruppen. Die theoretische Grenze von bis zu 65.536 Gruppen konnte im Code erfolgreich abgebildet werden.

3.2 Ergebnisdiskussion

Die bisherigen Ergebnisse zeigen vielversprechende Ansätze zur Realisierung eines zuverlässigen, verschlüsselten und dezentralisierten Kommunikationssystems auf Basis einfacher Zustandsübertragung.

1. **Erfolgreiche Aspekte**:

- **Skalierbarkeit**: Die Gruppe-Zuordnung kann für verschiedene Anwendungen flexibel genutzt werden.
- **Grundstruktur**: Das Unterprotokoll bewies sich als stabil und erweiterbar, was eine gute Basis für das Hauptprotokoll schafft.
- **Einfache Signalübertragung**: Die auf zwei Zuständen basierende Codierung erwies sich als robust, selbst bei Signalrauschen.

2. **Verbesserungspotenziale**:

- **Hauptprotokoll**: Die Implementierung der definierten Pakettypen ist essenziell, um alle Vorteile des Protokolls, wie etwa die verschlüsselte Kommunikation und Fehlerkorrektur, zu realisieren.
- **Testumgebung**: Eine ausführlichere Testumgebung mit größerer Hardware-Variation wäre wünschenswert, um die Robustheit unter verschiedenen Bedingungen zu testen.

3. **Praktische Anwendungen**: Potenzielle Anwendungen des Systems reichen von drahtlosen Sensor-Netzwerken bis hin zu der verschlüsselten Kommunikation für smarte Geräte. Die Möglichkeit, das Protokoll sowohl kabelgebunden als auch drahtlos zu betreiben, erhöht seine Anwendungsbreite. Theoretisch kann das Protokoll auch bei allen anderen Objekten, die zwei Zustände annehmen können, eingesetzt werden (z. B. Taschenlampen, Laser).

3.3 Ausblick

Das entwickelte Protokoll bietet eine vielversprechende Grundlage für sichere und skalierbare Kommunikation sowohl über drahtlose als auch kabelgebundene Netzwerke. Die Kombination von Hashing, Verschlüsselung und Signaturprüfung sorgt für eine hohe Integrität und Vertraulichkeit der übertragenen Daten.

Zukünftige Arbeiten

1. **Implementierung des Hauptprotokolls**: Der nächste Meilenstein ist die vollständige Umsetzung der definierten Pakettypen. Dadurch wird das System in der Lage sein, Nachrichten und Metadaten standardisiert zu übermitteln.
2. **Erweiterung auf zusätzliche Medien**: Neben drahtgebundenen und Funkübertragungen könnte das Protokoll auf andere Übertragungsmedien wie Lichtsignale (z. B. LEDs oder Laser) ausgeweitet werden, um den Einsatzbereich zu erweitern.
3. **Optimierung der Latenzzeiten**: Adaptive Mechanismen, wie die dynamische Anpassung der Signaldauer basierend auf der Übertragungsqualität, könnten die Effizienz verbessern und die Latenzzeiten verringern.
4. **Integration in IoT-Plattformen**: Die Integration des Protokolls in IoT-Ökosysteme würde praktische Anwendungen ermöglichen, wie z. B. in der Heimautomation oder bei vernetzten Sensornetzwerken.
5. **Erweiterung der Pakettypen**: Das Protokoll könnte durch neue Pakettypen erweitert werden, um zusätzliche Funktionen wie Priorisierung von Nachrichten oder erweiterte Steuerbefehle zu ermöglichen.
6. **Post-Quantenverschlüsselung**: Aktuell basiert das Protokoll auf post-quantenresistente Verschlüsselungsmethoden. Jedoch ist das Netzwerk aktuell noch auf symmetrischen Systemen basiert, dies bedeutet, dass Qantencomputer die Daten nicht entschlüsseln können, jedoch alle Schlüssel manuell ausgetauscht werden müssen. Dieses Probelm muss noch durch post-quantenresistente asymmetrische Verschlüsselungsmethoden gelöst werden.
7. **Entwicklung eines Messenger-Dienstes**: Ein protokollspezifischer Messenger-Dienst könnte eine benutzerfreundliche Möglichkeit darstellen, das Potenzial der Technologie zu demonstrieren und praktische Anwendungsszenarien zu erforschen.

3.4 Fazit

Mit der erfolgreichen Umsetzung des Unterprotokolls ist ein wesentlicher erster Schritt getan. Die nächsten Schritte eröffnen nicht nur neue technische Möglichkeiten, sondern auch den Weg zu vielfältigen Anwendungen in der realen Welt.

3.5 Quellen- und Literaturverzeichnis

1. GPIO Pins (Leiting auf `HIGH` oder `LOW` setzen) https://docs.espressif.com/projects/arduino-esp32/en/latest/api/gpio.html
2. Hash-Eingwefunktionen https://de.wikipedia.org/wiki/Hashfunktion
3. Verschlüsselung https://en.wikipedia.org/wiki/Encryption
4. Siganturen https://de.wikipedia.org/wiki/Digitale_Signatur

3.6 Umgang mit KI

Die Nutzung der KI ChatGPT ist mit einer Autokorrektur vergleichbar. Sie wurde ausschließlich dazu eingesetzt, geschriebene Texte zu verbessern und Rechtschreibfehler zu korrigieren.

Kontakt: Manuel.Westermeier@gmx.de

Signierung: 2bcf5f3d236860434f803bbac8c85213c6afe1ba9b96530304fc76875381f5a0