Sichere P2P-Kommunikationsplattform - Technisches Protokoll

Übersicht

Dieses Protokoll beschreibt eine sichere P2P-Kommunikationsplattform mit Fokus auf Integrität, Vertraulichkeit und Verfügbarkeit. Das System kombiniert RSA-Verschlüsselung, AES-Session-Keys und Merkle-Tree-Integritätsverifikation.

1 Komponentenarchitektur

1.1 Client-Komponenten

- **PHONEBOOK**: Hauptanwendung mit Benutzeroberfläche
- **CALL**: Anrufmanagement mit Audio-Streaming
- **ClientRelayManager**: Server-Discovery und Lastverteilung
- **SecureVault**: Sichere Schlüsselspeicherung
- **SecurityMonitor**: Systemhärtung (Linux-only)

1.2 Sicherheitsfeatures

- RSA-4096 für Schlüsselaustausch
- AES-256-CBC für Audio-Streams
- Merkle-Tree für Schlüsselintegrität
- Quantum-Safe SHA3-256 Hashing
- Forward Secrecy durch Session-Keys

2 Protokollabläufe

2.1 Setup-Phase (Initialisierung)

Listing 1: Client-Initialisierung

```
1. Lade/Generiere RSA-Schlüsselpaar (4096 Bit)
2. Lade Client-Name aus client_name.txt
3. Initialisiere Security Monitor (Linux)
4. Starte Relay-Manager für Server-Discovery
5. Baue Benutzeroberfläche auf
```

2.2 Update-Phase (Identität & Telefonbuch)

Listing 2: Registrierung und Update

```
1. Sende REGISTER an Server mit:
   - Öffentlichem Schlüssel
   - Client-Name
   - Version

2. Server antwortet mit:
   - Server Public Key
   - Merkle-Tree aller öffentlichen Schlüssel

3. Client verifiziert Merkle-Root:
   quantum_safe_hash(sorted_keys) == received_hash

4. Bei Erfolg: Identity Challenge
   - Server verschlüsselt Challenge mit Client-Public-Key
   - Client entschlüsselt mit Private-Key
```

```
   - Response mit Server-Public-Key verschlüsselt zurücksenden

5. Bei Verifikation: Telefonbuch-Update
   - Server sendet verschlüsseltes Telefonbuch
   - RSA(Secret) + AES(Phonebook)
   - Client entschlüsselt und aktualisiert UI
```

2.3 Call-Phase (Anrufinitiierung)

Listing 3: Anrufablauf

```
1. on_call_click() - Benutzer wählt Kontakt aus

2. GET_PUBLIC_KEY an Server:
   - Target-Client-ID
   - Caller-Name und ID

3. Server sendet PUBLIC_KEY_RESPONSE:
   - Öffentlicher Schlüssel des Empfängers

4. Caller generiert Session-Key:
   - 16 Byte IV + 32 Byte AES-Key = 48 Byte Secret
   - Call-Daten: {caller_info, aes_iv, aes_key, timestamp}

5. Verschlüsselung mit Empfänger-Public-Key:
   encrypted_data = RSA_encrypt(call_data, recipient_pubkey)

6. CALL_REQUEST an Server:
   - Target-Client-ID
   - Verschlüsselte Call-Daten
   - Caller-Informationen

7. Server leitet an Empfänger weiter:
   - INCOMING_CALL Nachricht

8. Empfänger entschlüsselt mit Private-Key:
   call_data = RSA_decrypt(encrypted_data, recipient_privkey)

9. Empfänger akzeptiert/lehnt ab:
   - CALL_RESPONSE an Server

10. Bei Annahme: Bidirektionale Audio-Streams
    - AES-256-CBC Verschlüsselung
    - UDP-Streaming über Relay oder direkt
```

3 Relay-Management vs. Hole Punching - Sicherheitsanalyse

3.1 Relay-Ansatz: Zentrale Kontrolle

- **Geschützte Clients**: Keine Exposition der Client-IPs nach außen
- **Kontrollierte Infrastruktur**: Feste Seed-Server (sichereleitung.duckdns.org:5060/5061)
- **Integritätskontrolle**: Server überwacht Protokollkonformität
- **Zentrale Gegenmaßnahmen**: Einheitlicher DOS-Schutz für alle Clients
- **Load Balancing**: Automatische Server-Auswahl basierend auf Ping und Last
- **Professionelle Sicherheitsverwaltung**: Zentrale Blacklists, Updates und Reaktionen

3.2 Hole Punching: Dezentrale Risiken

- **IP-Exposition**: Clients geben ihre IP-Adressen preis → Direkte Angriffsfläche
- **Man-in-the-Middle**: Ungeschützte direkte Verbindungen anfällig für MITM
- **Protokollmanipulation**: Jeder Peer kann das Protokoll eigenständig modifizieren
- **Fehlende Integritätskontrolle**: Keine zentrale Instanz zur Überprüfung der Sicherheitskonformität
- **Dezentrale Schwachstellen**: Sicherheitslücken in einzelnen Clients gefährden das gesamte Netzwerk
- **Unkontrollierbare Sicherheit**: Keine Möglichkeit, systemweite Sicherheitsmaßnahmen zu implementieren

3.3 Kritische Sicherheitsvergleiche

Relay-Ansatz	Hole Punching
✓ **Geschützte Clients**: Keine IP-Exposition nach außen	✗ **IP-Exposition**: Clients direkt im Internet sichtbar
✓ **Integrität der Sicherheit**: Zentrale Protokollüberwachung	✗ **Fehlende Integritätskontrolle**: Jeder Peer kann Protokoll manipulieren
✓ **Einheitlicher DOS-Schutz**: Zentrale Abwehrmechanismen	✗ **Individualisierte Angriffe**: Direkte Angriffe auf einzelne Clients
✓ **Kontrollierte Infrastruktur**: Bekannte, gehärtete Server	✗ **Unkontrollierte Peers**: Unbekannte, möglicherweise kompromittierte Clients
✓ **Professionelle Sicherheitsverwaltung**: Zentrale Blacklists, Updates, Reaktionen	✗ **Unkontrollierbare Sicherheit**: Keine systemweiten Sicherheitsmaßnahmen möglich
✓ **Robuste Architektur**: Mehrere Server mit Load-Balancing	✗ **Fragile Verbindungen**: Abhängig von NAT-Kompatibilität und Peer-Verfügbarkeit
● **Praktisch vernachlässigbare Latenz**: 1-5ms bei lokalen Servern, 10-20ms bei geographischer Nähe	✓ **Minimale Latenz**: Direkte Verbindung zwischen Peers

3.4 Angriffsszenarien und Risiken

3.4.1 Hole Punching - Kritische Schwachstellen

- **IP-Spoofing**: Angreifer können gefälschte IP-Pakete injizieren
- **Connection Hijacking**: Übernahme bestehender Verbindungen
- **Denial-of-Service**: Direkte Angriffe auf exponierten Client-IPs
- **Protokollmanipulation**: Böswillige Clients können Sicherheitsfeatures deaktivieren
- **Man-in-the-Middle**: Unverschlüsselte Verbindungsaufbauphase angreifbar
- **NAT-Bypass-Angriffe**: Ausnutzung der NAT-Traversal-Mechanismen
- **Unkontrollierbare Sicherheitslücken**: Keine zentrale Stelle für Patches und Updates

3.4.2 Relay-Ansatz - Kontrollierte Sicherheit

- **Geschützter Verbindungsaufbau**: Alle Nachrichten über gesicherte Server-Kanäle
- **Protokollkonformität**: Server validiert alle Nachrichten auf Format und Sicherheit
- **Zentrale Abwehr**: DOS-Angriffe werden auf Server-Ebene abgewehrt
- **Integritätsmonitoring**: Kontinuierliche Überwachung der Sicherheitskonformität
- **Isolierte Clients**: Keine direkte Angriffsfläche für externe Angreifer
- **Schnelle Reaktion auf Angriffe**: Zentrale Blacklisting und Gegenmaßnahmen
- **Systemweite Updates**: Sofortige Verteilung von Sicherheitspatches

3.5 Latenz- und Verfügbarkeitsanalyse

3.5.1 Latenzoptimierung

- **Lokale Server**: 1-5ms Round-Trip-Time bei Servern im selben Netzwerk
- **Geographische Nähe**: 10-20ms bei regionaler Serverplatzierung
- **Intelligentes Routing**: Automatische Auswahl des nächstgelegenen Servers
- **Audio-Streaming**: 20-40ms Pufferung macht Relay-Latenz praktisch irrelevant
- **Praktischer Nutzen**: Sicherheitsvorteile überwiegen die minimale Latenzerhöhung bei weitem

3.5.2 Verfügbarkeit und Robustheit

- **Multi-Server-Architektur**: Kein Single Point of Failure durch redundante Server
- **Automatisches Failover**: Client wechselt bei Serverausfall transparent
- **Load-Balancing**: Gleichmäßige Verteilung der Last auf mehrere Server
- **Professionelle Wartung**: 24/7 Überwachung und Wartung der Server-Infrastruktur
- **Schnelle Reaktionszeiten**: Sofortige Maßnahmen bei Sicherheitsvorfällen

4 Integritäts- und Sicherheitsfeatures

4.1 Merkle-Tree Schlüsselverifikation

```
def verify_merkle_integrity(all_keys, received_root_hash):
  1. Normalisiere alle öffentlichen Schlüssel
  2. Sortiere Schlüssel konsistent
  3. Merge mit "|||" Separator
  4. Berechne Merkle-Root:
        quantum_safe_hash(merged_keys)
  5. Vergleiche mit Server-Hash
```

4.2 Quantum-Safe Hashing

```
def quantum_safe_hash(data):
  # Fallback: pysha3 für Python 3.5
  if USE_PYSHA3 == False:
      return hashlib.sha3_256(data).hexdigest()
  else:
      return sha3.sha3_256(data).hexdigest()
```

5 Zusammenfassung

Das System implementiert ein umfassendes Sicherheitskonzept mit besonderem Fokus auf **Integrität der Sicherheit**:

- **Ende-zu-Ende-Verschlüsselung**: RSA für Schlüsselaustausch, AES für Daten
- **Forward Secrecy**: Session-Keys für jede Verbindung
- **Schlüsselintegrität**: Merkle-Tree Verifikation aller öffentlicher Schlüssel
- **Quantum-Resistance**: SHA3-256 für Hashing
- **Kontrollierte Infrastruktur**: Relay-Ansatz schützt Clients vor Exposition
- **Integritätskontrolle**: Zentrale Überwachung der Protokollkonformität
- **Professionelle Sicherheitsverwaltung**: Zentrale Blacklists, Updates und Gegenmaßnahmen
- **Robuste Architektur**: Multi-Server-Design ohne Single Point of Failure
- **Praktikable Performance**: Minimale Latenzerhöhung für erhebliche Sicherheitsgewinne

Der **Relay-Ansatz** bietet gegenüber Hole Punching entscheidende Sicherheitsvorteile durch geschützte Clients, kontrollierte Infrastruktur und garantierte Integrität der Sicherheitsimplementierung. Die zentrale Architektur ermöglicht professionelle Sicherheitsverwaltung und schnelle Reaktion auf Bedrohungen, was in dezentralen Peer-to-Peer-Systemen unmöglich ist.