Analyse der bestehenden Lösung

Post-Quantum-Krypto-Prototyp

Gruppe 3
Berk Meric, Valentin Bäuerle, Dennis Bantel, Florian Vasica

November 2024

Inhalt

Einleitung

- Analyse der bestehenden Lösung
- Ziel: Schwächen identifizieren und Optimierungspotential aufzeigen.
- Fokus: Code, Dokumentation und mögliche Verbesserungen.

Dokumentationsanalyse

Stärken:

- Ausführliche Beschreibung der Algorithmen und Testergebnisse.

Schwächen:

- Unvollständige Beschreibung der Architektur und Libraries.
- Keine Anleitungen zur Reproduzierbarkeit der Tests.

Code-Review

Stärken:

- Klar strukturierter, modularer Code.
- Nutzung aktueller Algorithmen (Kyber, SPHINCS+).

Schwächen:

- Fehlende Fehlerbehandlung bei kritischen Operationen.
- Manuelle Verzögerungen (`usleep`).
- Eingeschränkte Skalierbarkeit für mehrere Clients.
- Redundante Speicheroperationen.

Optimierungspotential

- **Automatisierte Fehlerwiederholung:** Robustere Mechanik bei Verbindungsfehlern.
- **Parallele Verarbeitung:** Multi-Threading zur Performance-Verbesserung.
- **Erweiterte Protokollierung:** Log-Details wie Speicherverbrauch und Latenz hinzufügen.
- **Effizientes Speichermanagement:** Verbesserte Nutzung dynamischer Speicheroperationen.

Fehlerbehandlung und Wiederholungsmechanik

Aktuell:

```
1  if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
2      fprintf(log_file, "Connection Failed (iteration %d)\n", i+1);
3      close(sock);
4      usleep(RETRY_DELAY);
5      continue;
6  }
```

Verbessert:

```
1  for (int retries = 0; retries < MAX_RETRIES; retries++) {
2      if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) == 0) {
3          fprintf(log_file, "Connection Successful "
4              "(iteration %d, retry %d)\n", i+1, retries);
5          break;
6      }
7      fprintf(log_file, "Retrying connection "
8          "(iteration %d, attempt %d)\n", i+1, retries);
9      usleep(RETRY_DELAY);
10     if (retries == MAX_RETRIES - 1) {
11         fprintf(log_file, "Connection Failed after "
12             "%d attempts (iteration %d)\n", MAX_RETRIES, i+1);
13         close(sock);
14         return 1;
15     }
16 }
```

Optimiertes Speichermanagement

Aktuell:

```
1  char *ptr = realloc(mem->memory, mem->size + totalSize + 1);
2  if (ptr == NULL) {
3      printf("Not enough memory (realloc returned NULL)\n");
4      return 0;
5  }
```

Verbessert:

```
1  void *ptr = realloc(mem->memory, mem->size + totalSize + 1);
2  if (!ptr) {
3      fprintf(log_file, "Memory allocation failed "
4          "(iteration %d)\n", i+1);
5      free(mem->memory); // Verhindert Speicherlecks
6      return 0;
7  }
```

Fazit

- Die Analyse zeigt klare Schwächen und Optimierungsmöglichkeiten.
- Fokus auf Automatisierung, Skalierbarkeit und bessere Dokumentation.
- Nächster Schritt: Umsetzung der Optimierungsvorschläge.