Aufgabe 15 “Vigenère Bruteforce Cracker”

Einleitung

Implementierung eines Brute-Force-Angriffs auf eine Vigenère-Verschlüsselung. Die Implementierung verwendet Multiprocessing, um die Entschlüsselungsaufgaben parallel auszuführen und die Berechnungszeit zu verkürzen. Die Implementierung bietet die Option, entweder beim ersten Fund eines passenden Schlüssels zu stoppen oder alle möglichen Kombinationen zu testen.

Funktionsbeschreibung

Funktionen:

1. **brute_force_worker(key, cipher_text, known_plaintext_start)**
   - Diese Funktion wird von den Worker-Prozessen aufgerufen und führt die tatsächliche Entschlüsselung für einen bestimmten Schlüssel durch.
   - **Parameter**:
     - `key`: Der aktuell zu testende Schlüssel als Tuple von Buchstaben.
     - `cipher_text`: Der zu entschlüsselnde Text.
     - `known_plaintext_start`: Der bekannte Klartextanfang (“hello”).
   - **Funktionsweise**: Der Text wird mit dem aktuellen Schlüssel entschlüsselt, und falls der entschlüsselte Text mit dem bekannten Klartextanfang beginnt, wird der Schlüssel zurückgegeben und der entschlüsselte Text ausgegeben. Ansonsten wird zweimal `None` zurückgegeben.
2. **brute_force_vigenere(cipher_text, known_plaintext_start, max_key_length, terminate_on_first_match=True)**
   - Übernimmt die Steuerung des Brute-Force-Angriffs und koordiniert die parallele Entschlüsselung des verschlüsselten Textes.
   - **Parameter**:
     - `cipher_text`: Der verschlüsselte Text.
     - `known_plaintext_start`: Der bekannte Klartextanfang.
     - `max_key_length`: Die maximale Länge des zu testenden Schlüssels.
     - `terminate_on_first_match`: Bestimmt, ob der Prozess beim ersten Fund beendet werden soll.
   - **Funktionsweise**:
     - Die Funktion testet alle Schlüssellängen bis hin zur maximalen Schlüssellänge `max_key_length`.
     - Für jede Schlüssellänge werden alle möglichen Buchstabenkombinationen generiert. Dazu wird die `itertools.product`-Funktion verwendet.
     - Die `partial`-Funktion von `functools` wird verwendet, um die `brute_force_worker`-Funktion partiell zu instanziieren.
     - Ein Pool von Prozessen wird erstellt und die Entschlüsselungsaufgaben werden parallelisiert.
     - Je nach `terminate_on_first_match` wird entweder beim ersten Fund gestoppt oder alle Kombinationen getestet.

Implementierte Optimierungen

- **Verwendung von `functools.partial`**: Optimiert die Parameterübergabe an Worker-Prozesse.
- **Multiprocessing**: Parallele Ausführung der Brute-Force-Operation mit `multiprocessing.Pool`.

- **Effiziente Iteratoren**: Verwendung von `itertools.product` und `itertools.chain` für Speichereffizienz.
- **Optimale Chunk-Größe**: Berechnung einer optimalen Chunk-Größe für gleichmäßige CPU-Auslastung.
- **Optionales frühes Stoppen**: Flexibilität zwischen schneller Erstfund-Suche und vollständiger Analyse.

Beispielaufrufe und Ergebnisse

1. **Validierung der Verschlüsselungsfunktionen**

   ```
   Plain text: test
   Cipher text: diqd
   Decrypted text: test
   ```

2. **Brute-Force mit frühem Stopp (max_key_length=5)**

   ```
   Total number of possible combinations: 12356630
   Key found: xmkey after 11233503 steps
   Decrypted text: helloworld
   Time taken: 8.97 seconds
   ```

3. **Brute-Force mit frühem Stopp (max_key_length=6)**

   ```
   Total number of possible combinations: 321272406
   Key found: xmkey after 11203503 steps
   Decrypted text: helloworld
   Time taken: 8.83 seconds
   ```

4. **Brute-Force ohne frühen Stopp (max_key_length=5)**

   ```
   Total number of possible combinations: 12356630
   Key found: xmkey after 12356630 steps
   Decrypted text: helloworld
   Time taken: 9.74 seconds
   ```

5. **Brute-Force ohne frühen Stopp (max_key_length=6)**

   ```
   Total number of possible combinations: 321272406
   Key found: xmkeyz after 321272406 steps
   Decrypted text: helloudpfx
   Time taken: 257.96 seconds
   ```

Analyse der Ergebnisse

Die Ergebnisse zeigen interessante Muster:

1. **Mit frühem Stopp** (Tests 2 & 3):
   - Beide Tests finden den gleichen Schlüssel “xmkey” nach etwa 11.2 Millionen Schritten
   - Die Ausführungszeiten sind fast identisch (~8.9 Sekunden)
   - Die theoretische Gesamtzahl der Kombinationen spielt keine Rolle, da nach dem Fund gestoppt wird
2. **Ohne frühen Stopp** (Tests 4 & 5):
   - Test 4 bestätigt den Schlüssel “xmkey” nach Durchlauf aller 12.3 Millionen Kombinationen
   - Test 5 findet einen anderen Schlüssel “xmkeyz” nach Durchlauf aller 321.2 Millionen Kombinationen
   - Die Ausführungszeit für max_key_length=6 ist deutlich länger (257.96 Sekunden)
   - Dies demonstriert den exponentiellen Anstieg der Berechnungszeit mit der Schlüssellänge