Symmetrischer Verschlüsselungs algorithmus

Ilic Sanja – cs24m018

Überblick über den gesamten Algorithmus

- Algorithmus verwendet eine Kombination aus Permutation und Substitution
- Klartext wird umcodiert → dann in Blöcke aufgeteilt
- jeder Block wird durch mehrere Runden von Permutation und Substitution verschlüsselt
- verschlüsselten Blöcke werden zu einem verschlüsselten Text kombiniert

Die Bedeutung der symmetrischen Verschlüsselung

- symmetrische Verschlüsselung ist eine der ältesten und am weitesten verbreiteten Methoden
- effizient und eignet sich besonders für den sicheren Datentransfer
- häufig verwendet in Anwendungen wie sicheren Kommunikationsprotokollen und Dateiverschlüsselung

Komponente 1 – Substition des Klartexts

- Klartext wird vor der eigentlichen Verschlüsselung mit Ceasar-Verschlüsselung substituiert
- Zeichen werden um festen Wert verschoben (z.B. +3)
- Ziel: Erschweren der Entzifferung des Klartexts, bevor der symmetrische Blockalgorithmus angewendet wird

Komponente 2 - Aufteilung in Blöcke

- umcodierte Text wird in Blöcke fester Größe aufgeteilt (z.B. 8 Zeichen)
- falls nötig, Padding hinzugefügt, um die Blöcke füllen
- Ziel: Strukturierung des Textes für die Blockverschlüsselung

Padding - Sicherstellen einer ordnungsgemäßen Blockstruktur

- Padding sorgt dafür, dass Blöcke die gleiche Größe haben
- notwendig, um Verschlüsselungsprozess zu standardisieren
- Padding-Zeichen werden bei der Entschlüsselung wieder entfernt

Komponente 3 - Permutation und Substitution

- jeder Block in mehreren Runden permutiert und substituiert
- Permutation: Umordnen der Zeichen im Block (Verwendung eines Musters)
- Substitution: Zeichen im Block werden durch XOR mit einem Schlüssel ersetzt
- Ziel: Erhöhen der Sicherheit durch Verschleiern der ursprünglichen Zeichen

Mehrere Runden - Erhöhung der Sicherheit

- mehrere Runden (Permutation und Substitution) verstärken Verschlüsselung
- jede Runde macht es schwieriger, den ursprünglichen Klartext zu rekonstruieren
- erhöht Widerstandsfähigkeit gegen Kryptoanalysen

Komponente 4 - Entschlüsselung und Invertierbarkeit

- Der Algorithmus ist vollständig invertierbar.
- Permutation und Substitution können rückgängig gemacht werden.
- Das Padding wird entfernt und der Klartext wird zurückcodiert.
- Ziel: Wiederherstellung des ursprünglichen Textes ohne Datenverlust.

Vergleich Grafik

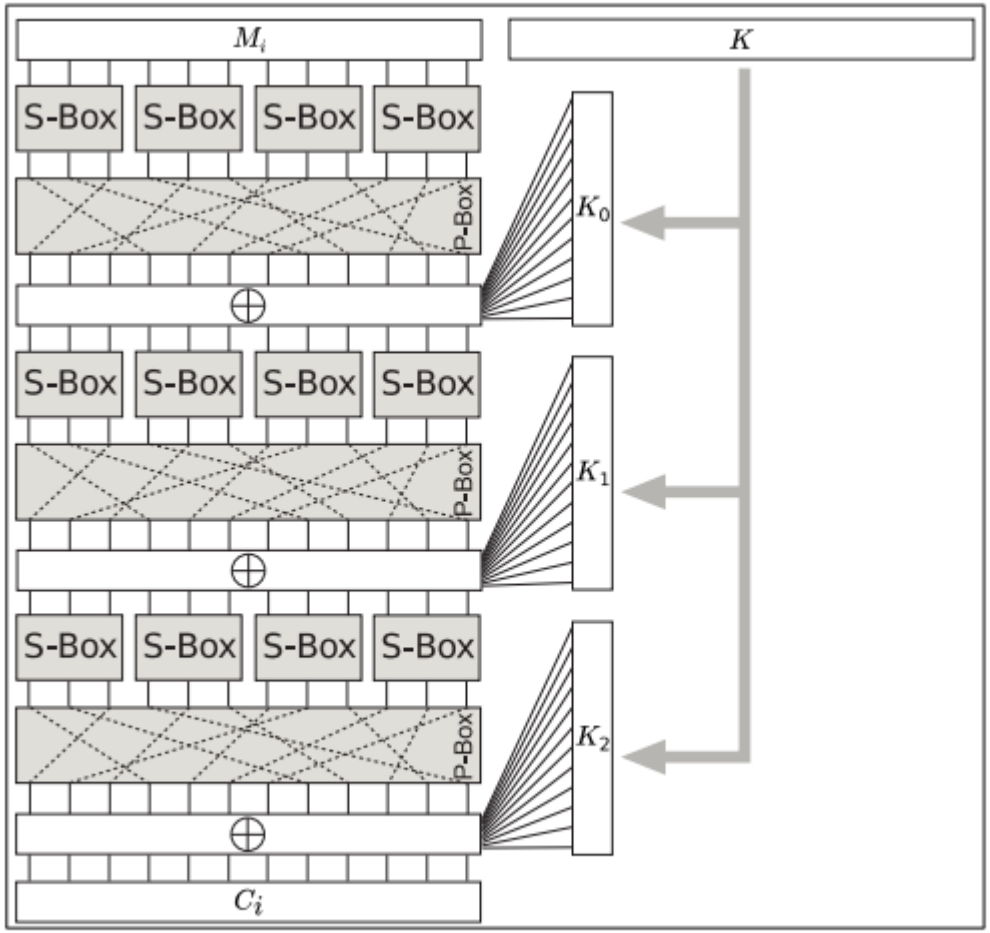

Abbildung 3.4: Blockchiffre: Netzwerk aus S-/P-Boxen und drei Rundendurchläufen [4]

Fazit und Fragen

- Algorithmus bietet eine einfache, aber effektive Verschlüsselung
- jede Komponente spielt eine entscheidende Rolle für die Sicherheit

Vielen Dank für die Aufmerksamkeit!

Code:

```java
public class CustomSymmetricEncryption {

    6 usages
    private static final int BLOCK_SIZE = 8; // Blockgröße in Zeichen
    1 usage
    private static final int ROUNDS = 4; // Anzahl der Runden

    // Umcodieren des Klartexts
    1 usage
    public static String encodePlainText(String plainText) {
        StringBuilder encoded = new StringBuilder();
        for (char c : plainText.toCharArray()) {
            encoded.append((char) (c + 3)); // Verschiebung der Zeichen um 3
        }
        return encoded.toString();
    }

    // Aufteilen des Klartexts in Blöcke mit Padding
    2 usages
    public static String[] splitIntoBlocks(String text) {
        // Padding hinzufügen

        int modulo = text.length() % BLOCK_SIZE; //Berechnet wie viele Zeichen der Text über oder unter der Blockgröße liegt
        int paddingLength = BLOCK_SIZE - modulo;
        if (paddingLength == BLOCK_SIZE) paddingLength = 0; // Kein Padding, wenn der Text exakt passt

        for (int i = 0; i < paddingLength; i++) {
            text += "\n";
        }

        //berechnet wie viele Blöcke es nach dem Padding gibt
        int blockCount = text.length() / BLOCK_SIZE;
        String[] blocks = new String[blockCount];
        for (int i = 0; i < blockCount; i++) {
            int start = i * BLOCK_SIZE;
            blocks[i] = text.substring(start, start + BLOCK_SIZE);
        }
        return blocks;
    }
```

```java
    // Permutation eines Blocks (einfaches Vertauschen der Zeichenpositionen)
    1 usage
    public static String permuteBlock(String block) {
        char[] permutedBlock = new char[block.length()];
        int[] permutationPattern = {2, 0, 3, 1, 6, 4, 7, 5}; // Beispiel für ein Muster
        for (int i = 0; i < block.length(); i++) {
            permutedBlock[i] = block.charAt(permutationPattern[i]);
        }
        return new String(permutedBlock);
    }

    // Verschlüsseln eines Blocks mit Permutation und Substitution
    2 usages
    public static String encryptBlock(String block, String key) {
        StringBuilder encryptedBlock = new StringBuilder();
        for (int round = 0; round < ROUNDS; round++) {
            block = permuteBlock(block); // Permutation des Blocks
            for (int i = 0; i < block.length(); i++) {
                char c = block.charAt(i);
                char k = key.charAt(i % key.length());
                encryptedBlock.append((char) (c ^ k)); // XOR mit Schlüssel als Substitution
            }
            block = encryptedBlock.toString(); // Ergebnis für die nächste Runde verwenden
            encryptedBlock.setLength(0);
        }
        return block;
    }

    // Funktion übernimmt gesamte Verschlüsselung des Klartexts
    1 usage
    public static String encrypt(String plainText, String key) {
        plainText = encodePlainText(plainText); // Umcodieren des Klartexts
        String[] blocks = splitIntoBlocks(plainText); // Aufteilen in Blöcke mit Padding
        StringBuilder encryptedText = new StringBuilder();
        for (String block : blocks) {
            encryptedText.append(encryptBlock(block, key)); // Blockweise Verschlüsselung mit dem Schlüssel
        }
        return Base64.getEncoder().encodeToString(encryptedText.toString().getBytes());
    }
```

```java
    // Entschlüsselungsroutine (ähnlich wie Verschlüsselung, aber umgekehrt)
    1 usage
    public static String decrypt(String encryptedText, String key) {
        byte[] decodedBytes = Base64.getDecoder().decode(encryptedText.getBytes());
        String decodedText = new String(decodedBytes);
        String[] blocks = splitIntoBlocks(decodedText);
        StringBuilder decryptedText = new StringBuilder();
        for (String block : blocks) {
            decryptedText.append(encryptBlock(block, key)); // Entschlüsselung ist hier symmetrisch
        }
        var unpadded :String  = removePadding(decryptedText.toString());

        return decodePlainText(unpadded);
    }

    // Entfernen des Paddings nach der Entschlüsselung
    1 usage
    public static String removePadding(String text) { return  text.trim(); }

    // Dekodieren des Klartexts
    1 usage
    public static String decodePlainText(String encodedText) {
        StringBuilder decoded = new StringBuilder();
        for (char c : encodedText.toCharArray()) {
            decoded.append((char) (c - 3)); // Rückverschiebung der Zeichen um 3
        }
        return decoded.toString();
    }

    public static void main(String[] args) {
        String originalText = "Das ist ein Geheimtext";
        String key = "Schluessel"; // Einfache Schlüsselphrase

        System.out.println("Originaltext: " + originalText);

        // Verschlüsselung des Textes
        String encryptedText = encrypt(originalText, key);
        System.out.println("Verschlüsselter Text: " + encryptedText);

        // Entschlüsselung des Textes
        String decryptedText = decrypt(encryptedText, key);
        System.out.println("Entschlüsselter Text: " + decryptedText);
    }
}
```

Ausgabe:

```
Run:  CustomSymmetricEncryption
C:\Users\sanja\.jdks\azul-17.0.4.1\bin\java.exe "-javaagent:C:\Pr
Originaltext: Das ist ein Geheimtext
Verschlüsselter Text: c1BCF3xmZzNcWEUXWnh7eFhEQ1xrZxoa
Entschlüsselter Text: Das ist ein Geheimtext

Process finished with exit code 0
```