Лабораторна робота №8.

Time-based One Time Password.

Мета.

Дослідити і реалізувати механізм генерації одноразових паролів TOTP.

Завдання.

Дослідити та реалізувати алгоритм Time-based One Time Password. Створити програму, що демонструє роботу розробленого алгоритму.

Організувати взаємодію з мобільним додатком Google Authenticator.

Виконання.

Двофакторна аутентифікація (або багатофакторна аутентифікація) — це метод ідентифікації користувача в будь-якому сервісі (як правило, в Інтернеті) за допомогою запиту аутентифікаційних даних двох різних типів, що забезпечує двошаровий, а отже, ефективніший захист облікового запису від несанкціонованого проникнення. Це означає, що після ввімкнення двофакторної аутентифікації користувач повинен пройти ще один крок для успішного входу в систему. Стандартні кроки для входу в обліковий запис – це введення логіну та введення пароля.

Включення двофакторної автентифікації додає до порядку входу додатковий крок - отримання одноразового пароля.

При включенні двофакторної аутентифікації з використанням TOTP користувачеві пропонується відсканувати QR-код за допомогою спеціальної програми для смартфона, яка надалі постійно генерує одноразовий пароль для користувача.

Коли користувач включає двофакторну автентифікацію, відбувається наступне.

Внутрішній сервер створює секретний ключ для цього користувача.

Потім сервер передає цей секретний ключ до телефонної програми користувача.

Телефонний додаток ініціалізує лічильник.

Телефонна програма генерує одноразовий пароль, використовуючи цей секретний ключ та лічильник.

Телефонний додаток змінює лічильник через певний інтервал та відновлює одноразовий пароль, роблячи його динамічним.

Для генерації одноразового пароля використовується алгоритм HOTP.

```
// HMAC-based One-Time Password algorithm (RFC 4226)
 public static String calcHotp(
   byte[] secretKey,
   byte[] counter,
   int codeLen,
   String hashFunc,
   int blockSize)
   throws NoSuchAlgorithmException {

  // Check argument, calculate HMAC
  if (!(1 <= codeLen && codeLen <= 9))
   throw new IllegalArgumentException("Invalid number of digits");
  byte[] hash = calcHmac(secretKey, counter, hashFunc, blockSize);

  // Dynamically truncate the hash value
  int offset = hash[hash.length - 1] & 0xF;
  int val = 0;
  for (int i = 0; i < 4; i++)
   val |= (hash[offset + i] & 0xFF) << ((3 - i) * 8);
  val &= 0x7FFFFFFF;
```

```
  // Extract and format base-10 digits
  int tenPow = 1;
  for (int i = 0; i < codeLen; i++)
   tenPow *= 10;
  return String.format("%0" + codeLen + "d", val % tenPow);
 }


 private static byte[] calcHmac(
   byte[] key,
   byte[] message,
   String hashFunc,
   int blockSize)
   throws NoSuchAlgorithmException {

  Objects.requireNonNull(key);
  Objects.requireNonNull(message);
  Objects.requireNonNull(hashFunc);
  if (blockSize < 1)
   throw new IllegalArgumentException("Invalid block size");

  if (key.length > blockSize)
   key = MessageDigest.getInstance(hashFunc).digest(key);
  key = Arrays.copyOf(key, blockSize);

  byte[] innerMsg = new byte[key.length + message.length];
  for (int i = 0; i < key.length; i++)
   innerMsg[i] = (byte)(key[i] ^ 0x36);
  System.arraycopy(message, 0, innerMsg, key.length, message.length);
  byte[] innerHash = MessageDigest.getInstance(hashFunc).digest(innerMsg);

  byte[] outerMsg = new byte[key.length + innerHash.length];
  for (int i = 0; i < key.length; i++)
   outerMsg[i] = (byte)(key[i] ^ 0x5C);
  System.arraycopy(innerHash, 0, outerMsg, key.length, innerHash.length);
  return MessageDigest.getInstance(hashFunc).digest(outerMsg);
 }
```

В алгоритмі TOTP в якості лічильника використовуємо час.

```
// Time-based One-Time Password algorithm (RFC 6238)
 public static String calcTotp(
   byte[] secretKey,
   long epoch,
   int timeStep,
   long timestamp,
   int codeLen,
   String hashFunc,
   int blockSize)
   throws NoSuchAlgorithmException {

  // Calculate counter and HOTP
  long timeCounter = Math.floorDiv(timestamp - epoch, timeStep);
  byte[] counter = new byte[8];
  for (int i = counter.length - 1; i >= 0; i--, timeCounter >>>= 8)
   counter[i] = (byte)timeCounter;
  return calcHotp(secretKey, counter, codeLen, hashFunc, blockSize);
 }
```

На виході отримуємо одноразовий пароль.

682836

Висновки.

Дослідили і реалізували механізм генерації одноразових паролів TOTP.