基于otp算法的双向认证

先举例一个应用场景吧，我们应该都用U盾，或者将军令这种生成动态密钥的工具，其实它内部就是基于OTP算法来实现的。

算法概要

TOTP（基于时间的一次性密码算法）是支持时间作为动态因素基于HMAC一次性密码算法的扩展。

本算法是一个对称算法，也就是说，后台和移动端采用同样的密钥，同时这个算法是依赖于当前的系统时间的，所以可以用于动态验证。

TOTP = HMAC-SHA-1(K, (T - T0) / X)

- K 共享密钥
- T 当前时间戳
- T0 开始的时间戳
- X 时间步长

多唠叨几句:我们整体算法采用处理密钥的方式，是要将密钥转换成byte数组之后进行操作的，还有就是谷歌的这套双向认证算法中严格的采用了Base32字符串的方式，Base32中只有A-Z和2-7这些字符。

代码效果：

```
2017/08/09 20:39:56
983452
2017/08/09 20:39:57
983452
2017/08/09 20:39:58
983452
2017/08/09 20:39:59
983452
2017/08/09 20:40:00
977560
2017/08/09 20:40:01
977560
2017/08/09 20:40:02
977560
2017/08/09 20:40:03
977560
2017/08/09 20:40:04
977560
```

算法的核心类：

```
/**
 * Created by linSir
 * date at 2017/8/8.
 * describe: 算法的核心类
 */

public class PasscodeGenerator {
    private static final int MAX_PASSCODE_LENGTH = 9;

    private static final int[] DIGITS_POWER
```

```
            // 0 1  2   3    4     5      6       7        8         9
            =
{1,10,100,1000,10000,100000,1000000,10000000,100000000,1000000000};

    private final Signer signer;
    private final int codeLength;

    interface Signer {
        byte[] sign(byte[] data) throws GeneralSecurityException;
    }

    public PasscodeGenerator(Signer signer, int passCodeLength) {
        if ((passCodeLength < 0) || (passCodeLength > MAX_PASSCODE_LENGTH)) {
            throw new IllegalArgumentException(
                    "PassCodeLength must be between 1 and " +
MAX_PASSCODE_LENGTH
                            + " digits.");
        }
        this.signer = signer;
        this.codeLength = passCodeLength;
    }

    private String padOutput(int value) {
        String result = Integer.toString(value);
        for (int i = result.length(); i < codeLength; i++) {
            result = "0" + result;
        }
        return result;
    }

    public String generateResponseCode(long state)
            throws GeneralSecurityException {
        byte[] value = ByteBuffer.allocate(8).putLong(state).array();
        return generateResponseCode(value);
    }

    public String generateResponseCode(byte[] challenge)
            throws GeneralSecurityException {
        byte[] hash = signer.sign(challenge);

        int offset = hash[hash.length - 1] & 0xF;
        int truncatedHash = hashToInt(hash, offset) & 0x7FFFFFFF;
        int pinValue = truncatedHash % DIGITS_POWER[codeLength];
        return padOutput(pinValue);
    }

    private int hashToInt(byte[] bytes, int start) {
        DataInput input = new DataInputStream(
                new ByteArrayInputStream(bytes, start, bytes.length - start));
        int val;
        try {
            val = input.readInt();
        } catch (IOException e) {
            throw new IllegalStateException(e);
        }
        return val;
    }
}
```

整体算法的思想和代码实现大概就是这样。