娃娃机加解密说明

1 加密

1.1 加密方法

加密方法如下：

> 待加密内容：Content
> 密钥(ZEGO 分配)：ServerSecret
> 向量：IV = Random.nextBytes(16)
> 加密后密文为：Base64.encode(IV + AES(Content， ServerSecret,IV))

1.2 加密步骤

加密步骤如下：

1. 获取 serverSecret（加密密钥，由 ZEGO 提供，32 bytes）

2. 生成长度为 16bytes 的随机字符串作为 AES 加密向量

3. AES 加密，使用模式: CBC/PKCS5Padding

4. 将 iv、密文拼接，并对拼接内容 base64 编码，编码结果即为加密密文

2 解密

2.1 解密方法

解密方法如下：

> 待解密密文：EncryptedContent
> 密钥（ZEGO 分配）：ServerSecret
> 解密后明文为： PlainText = AES.decrypt(Content， ServerSecret， IV)
> 其中：

> IV = Base64.decode(EncryptedContent).subString(0, 16)
> Content = Base64.decode(EncryptedContent).subString(16)

2.2 解密步骤

解密步骤如下：

1. 获取 serverSecret（加密密钥，由 ZEGO 提供，32 bytes）
2. 对密文进行 base64 解码
3. 截取 base64 解码内容前 16bytes 数据为 AES 解密向量
4. 截取 base64 解码内容前 16bytes 之后的所有数据为解密密文
5. AES 解密，使用模式 CBC/PKCS5Padding；解密结果即为解密明文

3 加密报文示例

```
[ServerSecret]
12345678901234561234567890123456    // 加密密钥 32bytes

[IV]
1234567890123456                    // AES 加密向量 16bytes

[明文]
helloworld

[密文]
MTIzNDU2Nzg5MDEyMzQ1NlOQ6NWrqwfRO2DdheZia2M=
```

4 go语言示例代码

```
/*
* function:
*     加密娃娃机配置内容
*
* input:
*     serverSecret:   加密密钥，由zego提供，32 bytes
*     PlaintText:     配置文件内容（明文）
```

```
*
* return:
*     ret:            加密结果
*     err:            函数执行错误信息
*/
func EncryptConfig(serverSecret string, PlaintText string) (ret string, err error) {
  //生成长度为16bytes 的AES加解密向量
  iv := MakeRandomIv()

  //加密，AES/CBC/PKCS5Padding
  crypted, err := AesEncrypt([]byte(PlaintText), []byte(serverSecret), iv)
  if err != nil {
    return "", err
  }

  //将iv,密文拼接；最后对拼接结果base64 编码
  buf := string(iv) + string(crypted)
  return base64.StdEncoding.EncodeToString([]byte(buf)), nil
}


/*
* function:
*     解密娃娃机配置内容
*
* input:
*     serverSecret:   加密密钥，由zego提供，32 bytes
*     cryptedText:    密文内容
*
* return:
*     ret:            解密结果
*     err:            函数执行错误信息
*/
func DecryptConfig(serverSecret string, cryptedText string) (string, error) {
  //base64 解码
  buf, err := base64.StdEncoding.DecodeString(cryptedText)
  if err != nil {
    return "", err
  }

  //取前16bytes 数据为aes加解密向量
  iv := buf[:16]

  //取16bytes 之后的所有数据为解密密文
  cryptedData := buf[16:]

  //解密，AES/CBC/PKCS5Padding
  plaint, err1 := AesDecrypt(cryptedData, []byte(serverSecret), iv)
  return string(plaint), err1
}
```

```
/*
* function:
*     生成长度为16bytes 的AES加解密向量
*
* return:
*     ret:            长度为16bytes 的随机字符串
*/
func MakeRandomIv() (ret []byte) {
  str := "0123456789abcdefghijklmnopqrstuvwxyz"
  bytes := []byte(str)
  result := []byte{}
  r := rand.New(rand.NewSource(time.Now().UnixNano()))

  for i := 0; i < 16; i++ {
    result = append(result, bytes[r.Intn(len(bytes))])
  }
  return result
}


/*
* function:
*     AES加密，PKCS5Padding + CBC模式
*
* input:
*     origData:       明文
*     key:            密钥
*     iv:             加解密向量
*
* return:
*     ret:            AES加密结果
*     err:            函数执行错误信息
*/
func AesEncrypt(origData, key, iv []byte) (ret []byte, err error) {
  block, err := aes.NewCipher(key)
  if err != nil {
    return nil, err
  }

  blockSize := block.BlockSize()
  origData = PKCS5Padding(origData, blockSize)
  blockMode := cipher.NewCBCEncrypter(block, iv)
  crypted := make([]byte, len(origData))
  blockMode.CryptBlocks(crypted, origData)
  return crypted, nil
}


/*
* function:
*     AES解密，  PKCS5Padding + CBC模式
*
```

```
* input:
*     crypted:        密文
*     key:            密钥
*     iv:             加解密向量
*
* return:
*     ret:            AES解密结果
*     err:            函数执行错误信息
*/
func AesDecrypt(crypted, key, iv []byte) ([]byte, error) {
  block, err := aes.NewCipher(key)
  if err != nil {
    return nil, err
  }

  blockMode := cipher.NewCBCDecrypter(block, iv)
  origData := make([]byte, len(crypted))
  blockMode.CryptBlocks(origData, crypted)
  origData = PKCS5UnPadding(origData)
  return origData, nil
}
```

5 Java语言示例代码

因为某些国家的进出口限制，需要下载对应版本的Java Cryptography Extension (JCE)，替换JDK安装目录\jre\lib\security下的local_policy.jar和US_export_policy.jar，如果独立JRE的话也是覆盖相同路径的文件后，才提供 AES256 加解密实现。

JDK8 对应的 JCE 文件下载路径

```
BASE64.java

package com.zego.wawaji.utils;

/*
 * Copyright (C) 2010 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
```

```java
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */


import java.io.UnsupportedEncodingException;

/**
 * Utilities for encoding and decoding the Base64 representation of
 * binary data.  See RFCs <a
 * href="http://www.ietf.org/rfc/rfc2045.txt">2045</a> and <a
 * href="http://www.ietf.org/rfc/rfc3548.txt">3548</a>.
 */
public class BASE64 {
    /**
     * Default values for encoder/decoder flags.
     */
    public static final int DEFAULT = 0;

    /**
     * Encoder flag bit to omit the padding '=' characters at the end
     * of the output (if any).
     */
    public static final int NO_PADDING = 1;

    /**
     * Encoder flag bit to omit all line terminators (i.e., the output
     * will be on one long line).
     */
    public static final int NO_WRAP = 2;

    /**
     * Encoder flag bit to indicate lines should be terminated with a
     * CRLF pair instead of just an LF.  Has no effect if {@code
     * NO_WRAP} is specified as well.
     */
    public static final int CRLF = 4;

    /**
     * Encoder/decoder flag bit to indicate using the "URL and
     * filename safe" variant of Base64 (see RFC 3548 section 4) where
     * {@code -} and {@code _} are used in place of {@code +} and
     * {@code /}.
     */
    public static final int URL_SAFE = 8;

    /**
     * Flag to pass to {@link Base64OutputStream} to indicate that it
     * should not close the output stream it is wrapping when it
     * itself is closed.
     */
    public static final int NO_CLOSE = 16;

    //  --------------------------------------------------------
    //  shared code
    //  --------------------------------------------------------

    /* package */ static abstract class Coder {
        public byte[] output;
        public int op;

        /**
         * Encode/decode another block of input data.  this.output is
         * provided by the caller, and must be big enough to hold all
         * the coded data.  On exit, this.opwill be set to the length
         * of the coded data.
         *
         * @param finish true if this is the final call to process for
         *        this object.  Will finalize the coder state and
         *        include any final bytes in the output.
         *
         * @return true if the input so far is good; false if some
         *         error has been detected in the input stream..
         */
        public abstract boolean process(byte[] input, int offset, int len, boolean finish);

        /**
         * @return the maximum number of bytes a call to process()
         * could produce for the given number of input bytes.  This may
         * be an overestimate.
         */
        public abstract int maxOutputSize(int len);
    }

    //  --------------------------------------------------------
    //  decoding
    //  --------------------------------------------------------

    /**
     * Decode the Base64-encoded data in input and return the data in
     * a new byte array.
     *
     * <p>The padding ’=’ characters at the end are considered optional, but
     * if any are present, there must be the correct number of them.
     *
     * @param str    the input String to decode, which is converted to
     *               bytes using the default charset
     * @param flags  controls certain features of the decoded output.
     *               Pass {@code DEFAULT} to decode standard Base64.
     *
     * @throws IllegalArgumentException if the input contains
     * incorrect padding
     */
    public static byte[] decode(String str, int flags) {
        return decode(str.getBytes(), flags);
    }

    /**
     * Decode the Base64-encoded data in input and return the data in
     * a new byte array.
     *
     * <p>The padding '=' characters at the end are considered optional, but
     * if any are present, there must be the correct number of them.
     *
     * @param input the input array to decode
     * @param flags  controls certain features of the decoded output.
     *               Pass {@code DEFAULT} to decode standard Base64.
     *
     * @throws IllegalArgumentException if the input contains
     * incorrect padding
     */
    public static byte[] decode(byte[] input, int flags) {
        return decode(input, 0, input.length, flags);
    }

    /**
     * Decode the Base64-encoded data in input and return the data in
     * a new byte array.
     *
     * <p>The padding '=' characters at the end are considered optional, but
     * if any are present, there must be the correct number of them.
     *
     * @param input  the data to decode
     * @param offset the position within the input array at which to start
     * @param len    the number of bytes of input to decode
     * @param flags  controls certain features of the decoded output.
     *               Pass {@code DEFAULT} to decode standard Base64.
     *
     * @throws IllegalArgumentException if the input contains
     * incorrect padding
     */
    public static byte[] decode(byte[] input, int offset, int len, int flags) {
        // Allocate space for the most data the input could represent.
        // (It could contain less if it contains whitespace, etc.)
        Decoder decoder = new Decoder(flags, new byte[len*3/4]);

        if (!decoder.process(input, offset, len, true)) {
            throw new IllegalArgumentException("bad base-64");
        }

        // Maybe we got lucky and allocated exactly enough output space.
        if (decoder.op == decoder.output.length) {
            return decoder.output;
        }
        // Need to shorten the array, so allocate a new one of the
        // right size and copy.
        byte[] temp = new byte[decoder.op];
        System.arraycopy(decoder.output, 0, temp, 0, decoder.op);
        return temp;
    }

    /* package */ static class Decoder extends Coder {
        /**
         * Lookup table for turning bytes into their position in the
         * Base64 alphabet.
         */
        private static final int DECODE[] = {
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1, -1, 63,
            52, 53, 54, 55, 56, 57, 58, 59, 60, 61, -1, -1, -1, -2, -1, -1,
            -1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
            15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1, -1, -1, -1, -1,
            -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
            41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
        };

        /**
         * Decode lookup table for the "web safe" variant (RFC 3548
         * sec. 4) where - and _ replace + and /.
         */
        private static final int DECODE_WEBSAFE[] = {
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 62, -1, -1,
            52, 53, 54, 55, 56, 57, 58, 59, 60, 61, -1, -1, -1, -2, -1, -1,
            -1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
            15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, -1, -1, -1, -1, 63,
            -1, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
            41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
            -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
        -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
    };

    /** Non-data values in the DECODE arrays. */
    private static final int SKIP = -1;
    private static final int EQUALS = -2;

    /**
     * States 0-3 are reading through the next input tuple.
     * State 4 is having read one '=' and expecting exactly
     * one more.
     * State 5 is expecting no more data or padding characters
     * in the input.
     * State 6 is the error state; an error has been detected
     * in the input and no future input can "fix" it.
     */
    private int state;   // state number (0 to 6)
    private int value;

    final private int[] alphabet;

    public Decoder(int flags, byte[] output) {
        this.output = output;

        alphabet = ((flags & URL_SAFE) == 0) ? DECODE : DECODE_WEBSAFE;
        state = 0;
        value = 0;
    }

    /**
     * @return an overestimate for the number of bytes {@code
     * len} bytes could decode to.
     */
    public int maxOutputSize(int len) {
        return len * 3/4 + 10;
    }

    /**
     * Decode another block of input data.
     *
     * @return true if the state machine is still healthy.  false if
     *         bad base-64 data has been detected in the input stream.
     */
    public boolean process(byte[] input, int offset, int len, boolean finish) {
        if (this.state == 6) return false;

        int p = offset;
        len += offset;

        // Using local variables makes the decoder about 12%
        // faster than if we manipulate the member variables in
        // the loop.  (Even alphabet makes a measurable
        // difference, which is somewhat surprising to me since
        // the member variable is final.)
        int state = this.state;
        int value = this.value;
        int op = 0;
        final byte[] output = this.output;
        final int[] alphabet = this.alphabet;

        while (p < len) {
            // Try the fast path:  we’re starting a new tuple and the
            // next four bytes of the input stream are all data
            // bytes.  This corresponds to going through states
            // 0-1-2-3-0.  We expect to use this method for most of
            // the data.
            //
            // If any of the next four bytes of input are non-data
            // (whitespace, etc.), value will end up negative.  (All
            // the non-data values in decode are small negative
            // numbers, so shifting any of them up and or’ing them
            // together will result in a value with its top bit set.)
            //
            // You can remove this whole block and the output should
            // be the same, just slower.
            if (state == 0) {
                while (p+4 <= len &&
                       (value = ((alphabet[input[p] & 0xff] << 18) |
                                 (alphabet[input[p+1] & 0xff] << 12) |
                                 (alphabet[input[p+2] & 0xff] << 6) |
                                 (alphabet[input[p+3] & 0xff]))) >= 0) {
                    output[op+2] = (byte) value;
                    output[op+1] = (byte) (value >> 8);
                    output[op] = (byte) (value >> 16);
                    op += 3;
                    p += 4;
                }
                if (p >= len) break;
            }

            // The fast path isn’t available -- either we’ve read a
            // partial tuple, or the next four input bytes aren’t all
            // data, or whatever.  Fall back to the slower state
            // machine implementation.

            int d = alphabet[input[p++] & 0xff];

            switch (state) {
            case 0:
                if (d >= 0) {
                    value = d;
                    ++state;
                } else if (d != SKIP) {
                    this.state = 6;
        return false;
    }
    break;

case 1:
    if (d >= 0) {
        value = (value << 6) | d;
        ++state;
    } else if (d != SKIP) {
        this.state = 6;
        return false;
    }
    break;

case 2:
    if (d >= 0) {
        value = (value << 6) | d;
        ++state;
    } else if (d == EQUALS) {
        // Emit the last (partial) output tuple;
        // expect exactly one more padding character.
        output[op++] = (byte) (value >> 4);
        state = 4;
    } else if (d != SKIP) {
        this.state = 6;
        return false;
    }
    break;

case 3:
    if (d >= 0) {
        // Emit the output triple and return to state 0.
        value = (value << 6) | d;
        output[op+2] = (byte) value;
        output[op+1] = (byte) (value >> 8);
        output[op] = (byte) (value >> 16);
        op += 3;
        state = 0;
    } else if (d == EQUALS) {
        // Emit the last (partial) output tuple;
        // expect no further data or padding characters.
        output[op+1] = (byte) (value >> 2);
        output[op] = (byte) (value >> 10);
        op += 2;
        state = 5;
    } else if (d != SKIP) {
        this.state = 6;
        return false;
    }
    break;

case 4:
        if (d == EQUALS) {
            ++state;
        } else if (d != SKIP) {
            this.state = 6;
            return false;
        }
        break;

    case 5:
        if (d != SKIP) {
            this.state = 6;
            return false;
        }
        break;
    }
}

if (!finish) {
    // We're out of input, but a future call could provide
    // more.
    this.state = state;
    this.value = value;
    this.op = op;
    return true;
}

// Done reading input.  Now figure out where we are left in
// the state machine and finish up.

switch (state) {
case 0:
    // Output length is a multiple of three.  Fine.
    break;
case 1:
    // Read one extra input byte, which isn't enough to
    // make another output byte.  Illegal.
    this.state = 6;
    return false;
case 2:
    // Read two extra input bytes, enough to emit 1 more
    // output byte.  Fine.
    output[op++] = (byte) (value >> 4);
    break;
case 3:
    // Read three extra input bytes, enough to emit 2 more
    // output bytes.  Fine.
    output[op++] = (byte) (value >> 10);
    output[op++] = (byte) (value >> 2);
    break;
case 4:
    // Read one padding '=' when we expected 2.  Illegal.
    this.state = 6;
```

```
                return false;
            case 5:
                // Read all the padding '='s we expected and no more.
                // Fine.
                break;
            }

            this.state = state;
            this.op = op;
            return true;
        }
    }

    //  --------------------------------------------------------
    //  encoding
    //  --------------------------------------------------------

    /**
     * Base64-encode the given data and return a newly allocated
     * String with the result.
     *
     * @param input  the data to encode
     * @param flags  controls certain features of the encoded output.
     *               Passing {@code DEFAULT} results in output that
     *               adheres to RFC 2045.
     */
    public static String encodeToString(byte[] input, int flags) {
        try {
            return new String(encode(input, flags), "US-ASCII");
        } catch (UnsupportedEncodingException e) {
            // US-ASCII is guaranteed to be available.
            throw new AssertionError(e);
        }
    }

    /**
     * Base64-encode the given data and return a newly allocated
     * String with the result.
     *
     * @param input  the data to encode
     * @param offset the position within the input array at which to
     *               start
     * @param len    the number of bytes of input to encode
     * @param flags  controls certain features of the encoded output.
     *               Passing {@code DEFAULT} results in output that
     *               adheres to RFC 2045.
     */
    public static String encodeToString(byte[] input, int offset, int len, int flags)
{
        try {
            return new String(encode(input, offset, len, flags), "US-ASCII");
        } catch (UnsupportedEncodingException e) {
            // US-ASCII is guaranteed to be available.
            throw new AssertionError(e);
        }
    }

    /**
     * Base64-encode the given data and return a newly allocated
     * byte[] with the result.
     *
     * @param input  the data to encode
     * @param flags  controls certain features of the encoded output.
     *               Passing {@code DEFAULT} results in output that
     *               adheres to RFC 2045.
     */
    public static byte[] encode(byte[] input, int flags) {
        return encode(input, 0, input.length, flags);
    }

    /**
     * Base64-encode the given data and return a newly allocated
     * byte[] with the result.
     *
     * @param input  the data to encode
     * @param offset the position within the input array at which to
     *               start
     * @param len    the number of bytes of input to encode
     * @param flags  controls certain features of the encoded output.
     *               Passing {@code DEFAULT} results in output that
     *               adheres to RFC 2045.
     */
    public static byte[] encode(byte[] input, int offset, int len, int flags) {
        Encoder encoder = new Encoder(flags, null);

        // Compute the exact length of the array we will produce.
        int output_len = len / 3 * 4;

        // Account for the tail of the data and the padding bytes, if any.
        if (encoder.do_padding) {
            if (len % 3 > 0) {
                output_len += 4;
            }
        } else {
            switch (len % 3) {
                case 0: break;
                case 1: output_len += 2; break;
                case 2: output_len += 3; break;
            }
        }

        // Account for the newlines, if any.
        if (encoder.do_newline && len > 0) {
            output_len += (((len-1) / (3 * Encoder.LINE_GROUPS)) + 1) *
```

```
                (encoder.do_cr ? 2 : 1);
        }

        encoder.output = new byte[output_len];
        encoder.process(input, offset, len, true);

        assert encoder.op == output_len;

        return encoder.output;
    }

    /* package */ static class Encoder extends Coder {
        /**
         * Emit a new line every this many output tuples.  Corresponds to
         * a 76-character line length (the maximum allowable according to
         * <a href="http://www.ietf.org/rfc/rfc2045.txt">RFC 2045</a>).
         */
        public static final int LINE_GROUPS = 19;

        /**
         * Lookup table for turning Base64 alphabet positions (6 bits)
         * into output bytes.
         */
        private static final byte ENCODE[] = {
            'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P',
            'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e',
'f',
            'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
'v',
            'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+',
'/',
        };

        /**
         * Lookup table for turning Base64 alphabet positions (6 bits)
         * into output bytes.
         */
        private static final byte ENCODE_WEBSAFE[] = {
            'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P',
            'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'a', 'b', 'c', 'd', 'e',
'f',
            'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u',
'v',
            'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '-',
'_',
        };

        final private byte[] tail;
        /* package */ int tailLen;
        private int count;
```

```
final public boolean do_padding;
final public boolean do_newline;
final public boolean do_cr;
final private byte[] alphabet;

public Encoder(int flags, byte[] output) {
    this.output = output;

    do_padding = (flags & NO_PADDING) == 0;
    do_newline = (flags & NO_WRAP) == 0;
    do_cr = (flags & CRLF) != 0;
    alphabet = ((flags & URL_SAFE) == 0) ? ENCODE : ENCODE_WEBSAFE;

    tail = new byte[2];
    tailLen = 0;

    count = do_newline ? LINE_GROUPS : -1;
}

/**
 * @return an overestimate for the number of bytes {@code
 * len} bytes could encode to.
 */
public int maxOutputSize(int len) {
    return len * 8/5 + 10;
}

public boolean process(byte[] input, int offset, int len, boolean finish) {
    // Using local variables makes the encoder about 9% faster.
    final byte[] alphabet = this.alphabet;
    final byte[] output = this.output;
    int op = 0;
    int count = this.count;

    int p = offset;
    len += offset;
    int v = -1;

    // First we need to concatenate the tail of the previous call
    // with any input bytes available now and see if we can empty
    // the tail.

    switch (tailLen) {
        case 0:
            // There was no tail.
            break;

        case 1:
            if (p+2 <= len) {
                // A 1-byte tail with at least 2 bytes of
                // input available now.
```

```
                    v = ((tail[0] & 0xff) << 16) |
                        ((input[p++] & 0xff) << 8) |
                        (input[p++] & 0xff);
                    tailLen = 0;
                };
                break;

            case 2:
                if (p+1 <= len) {
                    // A 2-byte tail with at least 1 byte of input.
                    v = ((tail[0] & 0xff) << 16) |
                        ((tail[1] & 0xff) << 8) |
                        (input[p++] & 0xff);
                    tailLen = 0;
                }
                break;
        }

        if (v != -1) {
            output[op++] = alphabet[(v >> 18) & 0x3f];
            output[op++] = alphabet[(v >> 12) & 0x3f];
            output[op++] = alphabet[(v >> 6) & 0x3f];
            output[op++] = alphabet[v & 0x3f];
            if (--count == 0) {
                if (do_cr) output[op++] = '\r';
                output[op++] = '\n';
                count = LINE_GROUPS;
            }
        }

        // At this point either there is no tail, or there are fewer
        // than 3 bytes of input available.

        // The main loop, turning 3 input bytes into 4 output bytes on
        // each iteration.
        while (p+3 <= len) {
            v = ((input[p] & 0xff) << 16) |
                ((input[p+1] & 0xff) << 8) |
                (input[p+2] & 0xff);
            output[op] = alphabet[(v >> 18) & 0x3f];
            output[op+1] = alphabet[(v >> 12) & 0x3f];
            output[op+2] = alphabet[(v >> 6) & 0x3f];
            output[op+3] = alphabet[v & 0x3f];
            p += 3;
            op += 4;
            if (--count == 0) {
                if (do_cr) output[op++] = '\r';
                output[op++] = '\n';
                count = LINE_GROUPS;
            }
        }
```

```
            if (finish) {
                // Finish up the tail of the input.  Note that we need to
                // consume any bytes in tail before any bytes
                // remaining in input; there should be at most two bytes
                // total.

                if (p-tailLen == len-1) {
                    int t = 0;
                    v = ((tailLen > 0 ? tail[t++] : input[p++]) & 0xff) << 4;
                    tailLen -= t;
                    output[op++] = alphabet[(v >> 6) & 0x3f];
                    output[op++] = alphabet[v & 0x3f];
                    if (do_padding) {
                        output[op++] = '=';
                        output[op++] = '=';
                    }
                    if (do_newline) {
                        if (do_cr) output[op++] = '\r';
                        output[op++] = '\n';
                    }
                } else if (p-tailLen == len-2) {
                    int t = 0;
                    v = (((tailLen > 1 ? tail[t++] : input[p++]) & 0xff) << 10) |
                        (((tailLen > 0 ? tail[t++] : input[p++]) & 0xff) << 2);
                    tailLen -= t;
                    output[op++] = alphabet[(v >> 12) & 0x3f];
                    output[op++] = alphabet[(v >> 6) & 0x3f];
                    output[op++] = alphabet[v & 0x3f];
                    if (do_padding) {
                        output[op++] = '=';
                    }
                    if (do_newline) {
                        if (do_cr) output[op++] = '\r';
                        output[op++] = '\n';
                    }
                } else if (do_newline && op > 0 && count != LINE_GROUPS) {
                    if (do_cr) output[op++] = '\r';
                    output[op++] = '\n';
                }

                assert tailLen == 0;
                assert p == len;
            } else {
                // Save the leftovers in tail to be consumed on the next
                // call to encodeInternal.

                if (p == len-1) {
                    tail[tailLen++] = input[p];
                } else if (p == len-2) {
                    tail[tailLen++] = input[p];
                    tail[tailLen++] = input[p+1];
                }
            }

            this.op = op;
            this.count = count;

            return true;
        }
    }

    private BASE64() { }   // don't instantiate
}
```

```
AESUtil.java

package com.zego.wawaji.utils;

import java.security.SecureRandom;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

/**
 * AES 工具类。
 *
 * Copyright © 2017 Zego. All rights reserved.
 *
 * @author mark on 30/12/2017.
 */
public class AESUtil {

    static final private int IV_LENGTH = 16;

    static final private String TRANSFORMATION = "AES/CBC/PKCS5Padding";

    /**
     * 加密.
     *
     * @param content
     *            需要加密的内容
     * @param secretKey
     *            加密key
     * @param logger
     * @return 返回加密的内容
     * @throws Exception
     *             抛出加密异常
     */
    public static byte[] encrypt(String content, byte[] secretKey) throws Exception {
        if (secretKey.length != 32) {
            return null;
        }
            SecretKeySpec key = new SecretKeySpec(secretKey, "AES");
        SecureRandom rnd = new SecureRandom();
        byte[] newSeed = rnd.generateSeed(IV_LENGTH);
        rnd.setSeed(newSeed);
        byte[] ivBytes = new byte[IV_LENGTH];
        rnd.nextBytes(ivBytes);
        IvParameterSpec iv = new IvParameterSpec(ivBytes);
        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        cipher.init(Cipher.ENCRYPT_MODE, key, iv);
        byte[] contentBytes = cipher.doFinal(content.getBytes("UTF-8"));
        byte[] encryptedBytes = new byte[ivBytes.length + contentBytes.length];
        System.arraycopy(ivBytes, 0, encryptedBytes, 0, ivBytes.length);
        System.arraycopy(contentBytes, 0, encryptedBytes, ivBytes.length, contentBytes.length);
        return BASE64.encode(encryptedBytes, BASE64.DEFAULT);
    }

    /**
     * 解密.
     *
     * @param content
     *            需要解密的内容
     * @param secretKey
     *            解密key
     * @return 返回解密的内容
     * @throws Exception
     *             抛出解密异常
     */
    public static byte[] decrypt(String content, byte[] secretKey) throws Exception {

        if (secretKey.length != 32) {
            return null;
        }

        byte[] decryptBytes = BASE64.decode(content.getBytes("UTF-8"), BASE64.DEFAULT);
        byte[] ivBytes = new byte[IV_LENGTH];
        System.arraycopy(decryptBytes, 0, ivBytes, 0, IV_LENGTH);

        byte[] contentBytes = new byte[decryptBytes.length - IV_LENGTH];
        System.arraycopy(decryptBytes, IV_LENGTH, contentBytes, 0, decryptBytes.length - IV_LENGTH);

        SecretKeySpec key = new SecretKeySpec(secretKey, "AES");
        IvParameterSpec iv = new IvParameterSpec(ivBytes);

        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        cipher.init(Cipher.DECRYPT_MODE, key, iv);

        return cipher.doFinal(contentBytes);
        }
    }
```