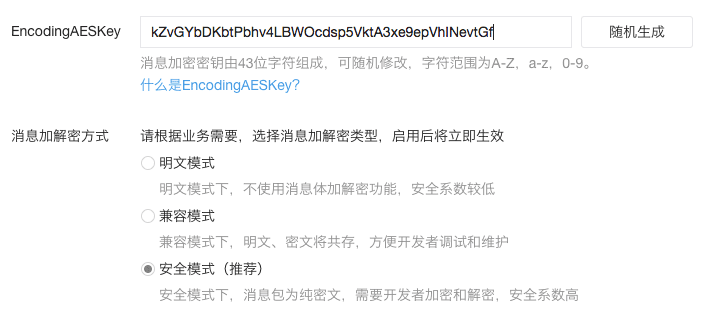
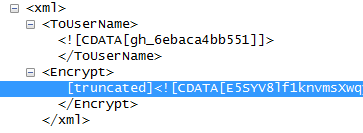
在上一篇“[接收文本消息](http://tonybai.com/2014/12/20/receive-text-for-wechat-public-platform-dev-in-golang/)”一文中，我们了解到：公众服务与微信服务器间的消息是“裸奔”的（即明文传输，通过抓包可以看到）。显然这对于一些对安 全性要求较高的大企业服务号来说，比如银行、证券、电信运营商或航空客服等是不能完全满足要求的。于是乎就有了微信服务器与公众服务间的数据加密 通信流程。

公众号管理员可以在公众号“开发者中心”选择是否采用"安全模式"(区别于明文模式)：



一旦选择了“安全模式”，微信服务器在向公众号服务转发消息时会对XML数据包部分内容进行加密处理。这类加密后的请求Body中的XML数据变 成了下面这样：



xml数据基本结构变成了:

<xml>  
    <ToUserName>xx</ToUserName>  
    <Encrypt>xx</Encrypt>  
</xml>

另外在“安全模式”下，Http Post Request line中也增加了两个字段：encrypt\_type和msg\_signuature，用于消息类型判断以及加密消息内容有效性校验：

POST /?signature=891789ec400309a6be74ac278030e472f90782a5&timestamp=1419214101&nonce=788148964&encrypt\_type=aes&msg\_signature=87d7b127fab3771b452bc6a592f530cd8edba950 HTTP/1.1\r\n

其中：

encrypt\_type = "aes"，说明是加密消息，否则为"raw”，即未加密消息。  
msg\_signature=sha1(sort(Token, timestamp, nonce, msg\_encrypt))

对于测试号，测试号配置页面没有加密相关配置，因此只能通过“[微信公众平台接口调试工具](https://mp.weixin.qq.com/debug)”来进行相关加密接口调试。

**一、消息签名验证**

对于“安全模式”下的消息交互，首先要做的就是消息签名验证，只有通过验证的消息才会进行下一步解密、解析和处理。

消息签名验证的原理是比较微信平台HTTP Post Line中携带的msg\_signature与通过Token、timestamp、nonce和msg\_encrypt等四个字段值计算出的 msg\_signture是否一致，一致则通过消息签名验证。

我们依旧在procRequest中完成对“安全模式”下消息的签名验证。

//recvencryptedtextmsg.go  
type EncryptRequestBody struct {  
        XMLName    xml.Name `xml:"xml"`  
        ToUserName string  
        Encrypt    string  
}

func makeMsgSignature(timestamp, nonce, msg\_encrypt string) string {  
        sl := []string{token, timestamp, nonce, msg\_encrypt}  
        sort.Strings(sl)  
        s := sha1.New()  
        io.WriteString(s, strings.Join(sl, ""))  
        return fmt.Sprintf("%x", s.Sum(nil))  
}

func validateMsg(timestamp, nonce, msgEncrypt, msgSignatureIn string) bool {  
        msgSignatureGen := makeMsgSignature(timestamp, nonce, msgEncrypt)  
        if msgSignatureGen != msgSignatureIn {  
                return false  
        }  
        return true  
}

func parseEncryptRequestBody(r \*http.Request) \*EncryptRequestBody {  
        body, err := ioutil.ReadAll(r.Body)  
        if err != nil {  
                log.Fatal(err)  
                return nil  
        }  
        requestBody := &EncryptRequestBody{}  
        xml.Unmarshal(body, requestBody)  
        return requestBody  
}

func procRequest(w http.ResponseWriter, r \*http.Request) {  
        r.ParseForm()

        timestamp := strings.Join(r.Form["timestamp"], "")  
        nonce := strings.Join(r.Form["nonce"], "")  
        signature := strings.Join(r.Form["signature"], "")  
        encryptType := strings.Join(r.Form["encrypt\_type"], "")  
        msgSignature := strings.Join(r.Form["msg\_signature"], "")

        … …

        f r.Method == "POST" {  
                if encryptType == "aes" {  
                        log.Println("Wechat Service: in safe mode")  
                        encryptRequestBody := parseEncryptRequestBody(r)  
                         
                        //Validate msg signature  
                        if !validateMsg(timestamp, nonce, encryptRequestBody.Encrypt, msgSignature) {  
                                log.Println("Wechat Service: msg\_signature is invalid")  
                                return  
                        }  
                        log.Println("Wechat Service: msg\_signature validation is ok!")  
                … …  
        }  
        … …  
}

程序编译执行结果如下：  
$sudo ./recvencryptedtextmsg  
2014/12/22 13:15:56 Wechat Service: Start!

用手机微信发送一条消息给公众号，程序输出如下结果：

2014/12/22 13:17:35 Wechat Service: in safe mode  
2014/12/22 13:17:35 Wechat Service: msg\_signature validation is ok!

**二、数据包解密**

到目前为止，我们已经得到了经过消息验证ok的加密数据包EncryptRequestBody 的Encrypt。要想得到真正的消息内容，我们需要对Encrypt字段的值进行解密处理。微信采用的是[AES](http://zh.wikipedia.org/wiki/AES)加解密方案， 下面我们就来看看如何做AES解密。

在开发者中心选择转换为“安全模式”时，有一个字段EncodingAESKey需要填写，这个字段固定为43个字符，它就是我们在运用AES算 法时需要的那个Key。不过这个EncodingAESKey是被编了码的，真正用来加解密的AESKey需要我们自己通过解码得到。解码方法 为：

AESKey=Base64\_Decode(EncodingAESKey + “=”)

Base64 decode后，我们就得到了一个32个字节的AESKey，可以看出微信加密解密用的是AES-256算法(256=32x8bit)。

在Golang中，我们可以通过下面代码得到真正的AESKey：

const (  
        token = "wechat4go"  
        appID = "wx5b5c2614d269ddb2"  
        encodingAESKey = "kZvGYbDKbtPbhv4LBWOcdsp5VktA3xe9epVhINevtGg"  
)

var aesKey []byte

func encodingAESKey2AESKey(encodingKey string) []byte {  
        data, \_ := base64.StdEncoding.DecodeString(encodingKey + "=")  
        return data  
}

func init() {  
        aesKey = encodingAESKey2AESKey(encodingAESKey)  
}

有了AESKey，我们再来解密数据包。微信公众平台开发文档给出了加密数据包的解析步骤：

1. aes\_msg=Base64\_Decode(msg\_encrypt)  
2. rand\_msg=AES\_Decrypt(aes\_msg)  
3. 验证尾部$AppId是否是自己的AppId，相同则表示消息没有被篡改，这里进一步加强了消息签名验证  
4. 去掉rand\_msg头部的16个随机字节，4个字节的msg\_len和尾部的$AppId即为最终的xml消息体

微信Wiki中如果能用一个简单的图来说明Base64\_Decode后的数据格式就更好了。这里进一步说明一下，解密后的数据，我们称之 plainData，它由四部分组成，按先后顺序排列分别是：

1、随机值       16字节  
2、xml包长度    4字节 （注意以BIG\_ENDIAN方式读取）  
3、xml包  （\*这部分数据的长度由上一个字段标识，这个包等价于一个完整的文本接收消息体数据，从ToUsername到MsgID都 有）  
4、appID

其中第三段xml包是一个完整的接收文本数据包，与“接收消息”一文中的标准文本数据包格式一致，这就方便我们解析了。好了，下面用代码阐述解 密、解析过程以及appid验证：

在procRequest中，增加如下代码：

**// Decode base64**  
cipherData, err := base64.StdEncoding.DecodeString(encryptRequestBody.Encrypt)  
if err != nil {  
        log.Println("Wechat Service: Decode base64 error:", err)  
        return  
}

**// AES Decrypt**  
plainData, err := aesDecrypt(cipherData, aesKey)  
if err != nil {  
        fmt.Println(err)  
        return  
}

**//Xml decoding**  
textRequestBody, \_ := parseEncryptTextRequestBody(plainData)  
fmt.Printf("Wechat Service: Recv text msg [%s] from user [%s]!",  
            textRequestBody.Content,  
            textRequestBody.FromUserName)

根据解密方法，我们先对encryptRequestBody.Encrypt进行base64 decode操作得到cipherData，再用aesDecrypt对cipherData进行解密得到上面提到的由四部分组成的plainData。plainData经过xml decoding后就得到我们的TextRequestBody struct。

这里难点显然在aesDecrypt的实现上了。微信的加密包采用aes-256算法，秘钥长度32B，采用PKCS#7 Padding方式。[Golang](http://tonybai.com/tag/golang)提供了强大的AES加密解密方法，我们利用这些方法实现微信包的解密：

func aesDecrypt(cipherData []byte, aesKey []byte) ([]byte, error) {  
        k := len(aesKey) //**PKCS#7**  
        if len(cipherData)%k != 0 {  
                return nil, errors.New("crypto/cipher: ciphertext size is not multiple of aes key length")  
        }

        block, err := aes.NewCipher(aesKey)  
        if err != nil {  
                return nil, err  
        }

        iv := make([]byte, aes.BlockSize)  
        if \_, err := io.ReadFull(rand.Reader, iv); err != nil {  
                return nil, err  
        }

        blockMode := cipher.NewCBCDecrypter(block, iv)  
        plainData := make([]byte, len(cipherData))  
        blockMode.CryptBlocks(plainData, cipherData)  
        return plainData, nil  
}

对于解密后的plainData做appID校验以及xml Decoding处理如下：

func parseEncryptTextRequestBody(plainText []byte) (\*TextRequestBody, error) {  
        fmt.Println(string(plainText))

**// Read length**  
        buf := bytes.NewBuffer(plainText[16:20])  
        var length int32  
        binary.Read(buf, binary.BigEndian, &length)  
        fmt.Println(string(plainText[20 : 20+length]))

**// appID validation**  
        appIDstart := 20 + length  
        id := plainText[appIDstart : int(appIDstart)+len(appID)]  
        if !validateAppId(id) {  
                log.Println("Wechat Service: appid is invalid!")  
                return nil, errors.New("Appid is invalid")  
        }  
        log.Println("Wechat Service: appid validation is ok!")

**// xml Decoding**  
        textRequestBody := &TextRequestBody{}  
        xml.Unmarshal(plainText[20:20+length], textRequestBody)  
        return textRequestBody, nil  
}

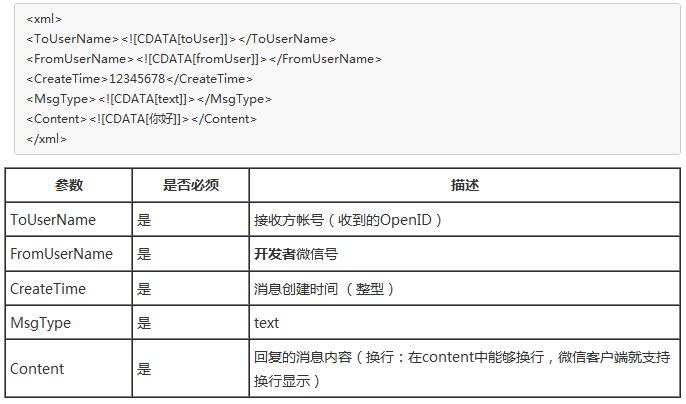
编译执行输出textRequestBody：

&{{ xml} gh\_6ebaca4bb551 on95ht9uPITsmZmq\_mvuz4h6f6CI 1.419239875s text **Hello, Wechat** 6095588848508047134}

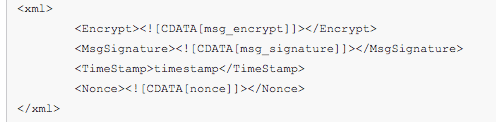
**三、响应消息的数据包加密**

微信公众平台开发文档要求：公众账号对密文消息的回复也要求加密。

对比一下普通的响应消息格式和加密后的响应消息格式：



加密后：



我们定义一个结构体映射响应消息数据包：

type EncryptResponseBody struct {  
        XMLName      xml.Name `xml:"xml"`  
        Encrypt      CDATAText  
        MsgSignature CDATAText  
        TimeStamp    string  
        Nonce        CDATAText  
}

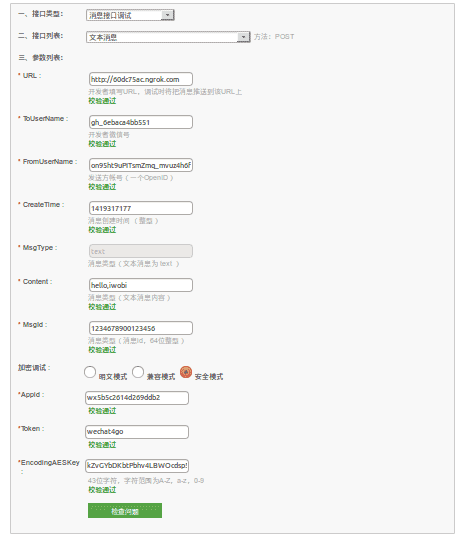
type CDATAText struct {  
        Text string `xml:",innerxml"`  
}

我们要做的就是给EncryptResponseBody的实例逐一赋值，然后通过xml.MarshalIndent转成xml数据流即可，各字 段值生成规则如下：

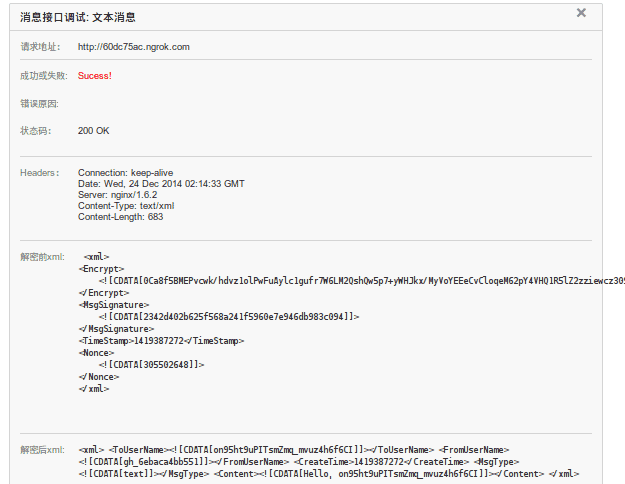
Encrypt = Base64\_Encode(AES\_Encrypt [random(16B)+ msg\_len(4B) + msg + $AppId])  
MsgSignature=sha1(sort(Token, timestamp, nonce, msg\_encrypt))  
TimeStamp = 用请求中的值或新生成  
Nonce = 用请求中的值或新生成

微信公众接口的加密复杂度要比解密高一些，关键问题在于加密结果的判定和加密逻辑的调试，AES加密出的结果每次都不同，我们要么通过微信平台真实操作验证，要么通过微信提供的在线调试工具验证加密是否正确。这里强烈建议使用在线调试工具(测试号只能选择这一种)。

在线调试工具的配置参考如下，ToUserName和FromUserName建议填写真实的（通过解密Post包打印输出得到）：



如果在线调试工具收到你的应答，并解密成功，会给出如下反馈：



在procRequest中，我们在接收解析完Http Request后，通过下面几行代码构造一个加密的Response返回给微信平台或调试工具：

responseEncryptTextBody, \_ := **makeEncryptResponseBody**(textRequestBody.ToUserName,  
                                textRequestBody.FromUserName,  
                                "Hello, "+textRequestBody.FromUserName,  
                                nonce,  
                                timestamp)  
w.Header().Set("Content-Type", "text/xml")  
fmt.Fprintf(w, string(responseEncryptTextBody))

func makeEncryptResponseBody(fromUserName, toUserName, content, nonce, timestamp string) ([]byte, error) {  
        encryptBody := &EncryptResponseBody{}

        encryptXmlData, \_ := **makeEncryptXmlData**(fromUserName, toUserName, timestamp, content)  
        encryptBody.Encrypt = value2CDATA(encryptXmlData)  
        encryptBody.MsgSignature = value2CDATA(makeMsgSignature(timestamp, nonce, encryptXmlData))  
        encryptBody.TimeStamp = timestamp  
        encryptBody.Nonce = value2CDATA(nonce)

        return xml.MarshalIndent(encryptBody, " ", "  ")  
}

应答Xml包中只有Encrypt字段是加密的，该字段的生成方式如下：

func makeEncryptXmlData(fromUserName, toUserName, timestamp, content string) (string, error) {  
        // **Encrypt part3: Xml Encoding**  
        textResponseBody := &TextResponseBody{}  
        textResponseBody.FromUserName = value2CDATA(fromUserName)  
        textResponseBody.ToUserName = value2CDATA(toUserName)  
        textResponseBody.MsgType = value2CDATA("text")  
        textResponseBody.Content = value2CDATA(content)  
        textResponseBody.CreateTime = timestamp  
        body, err := xml.MarshalIndent(textResponseBody, " ", "  ")  
        if err != nil {  
                return "", errors.New("xml marshal error")  
        }

        // **Encrypt part2: Length bytes**  
        buf := new(bytes.Buffer)  
        err = binary.Write(buf, binary.BigEndian, int32(len(body)))  
        if err != nil {  
                fmt.Println("Binary write err:", err)  
        }  
        bodyLength := buf.Bytes()

        // **Encrypt part1: Random bytes**  
        randomBytes := []byte("abcdefghijklmnop")

**// Encrypt Part, with part4 - appID**  
        plainData := bytes.Join([][]byte{randomBytes, bodyLength, body, []byte(appID)}, nil)  
        cipherData, err := **aesEncrypt**(plainData, aesKey)  
        if err != nil {  
                return "", errors.New("aesEncrypt error")  
        }

        return base64.StdEncoding.EncodeToString(cipherData), nil  
}

func aesEncrypt(plainData []byte, aesKey []byte) ([]byte, error) {  
        k := len(aesKey)  
        if len(plainData)%k != 0 {  
                plainData = PKCS7Pad(plainData, k)  
        }  
         
        block, err := aes.NewCipher(aesKey)  
        if err != nil {  
                return nil, err  
        }

        iv := make([]byte, aes.BlockSize)  
        if \_, err := io.ReadFull(rand.Reader, iv); err != nil {  
                return nil, err  
        }

        cipherData := make([]byte, len(plainData))  
        blockMode := cipher.NewCBCEncrypter(block, iv)  
        blockMode.CryptBlocks(cipherData, plainData)

        return cipherData, nil  
}

根据官方文档： 微信所用的AES采用的时CBC模式，秘钥长度为32个字节（aesKey），数据采用PKCS#7填充；PKCS#7：K为秘钥字节数（采用32），buf为待加密的内容，N为其字节数。****Buf需要被填充为K的整数倍****。因此我们pad要加密的数据时，**务必pad为k(=32)的整数倍**，而不是aes.BlockSize(=16)的整数倍。

采用安全模式后的公众号消息交互性能似乎下降了，发送"hello, wechat"给公众号后好长时间才收到响应。

微信公众号接收加密消息的代码在[这里](https://github.com/bigwhite/experiments/tree/master/wechat_examples/public/3-encryptedtextmsg)可以下载。这些代码只是演示代码，结构上绝不算优化，大家可以将这些代码封装成通用的接口为后续微信公众平台接口开发奠定基础。

文章地址：

https://tonybai.com/2014/12/24/recv-encrypted-text-msg-for-wechat-public-platform-dev-in-golang/