中国网安国密化修改介绍

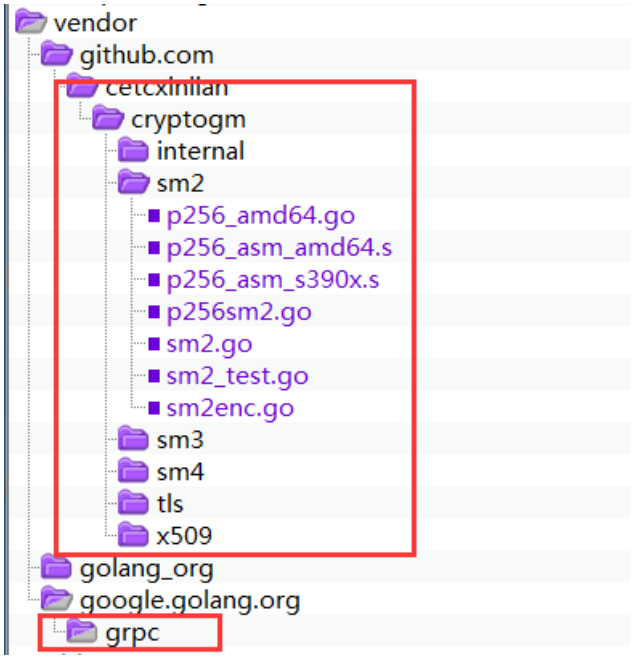
——技术支持（苏云龙，刘地军）

1. 修改思路：

修改主要分为三部分，分别为国密算法支持、国密证书支持、国密 grpc 支持。针对国密算法的支持，我们实现了对应国密算法库，在 fabric 密码调用的部分，添加国密的支持。针对国密证书，我们也提供了对应的库，扩展了原有功能，支持国密算法，这部分修改主要是将 fabric 中的库路径切换为我们提供的库。针对国密 grpc，将原有 grpc 的文件替换为支持国密的 grpc 库。此方案的优势是密码算法作为单独的库提供，如果后期要做密码算法优化或者硬件国密的改造，只需要修改 github.com/cetcxinlian/cryptogm 库中的对应部分即可，兼容性比较好。同时采用 go 语言实现便于调试，经过汇编优化，性能也较好。

2. 算法库介绍：

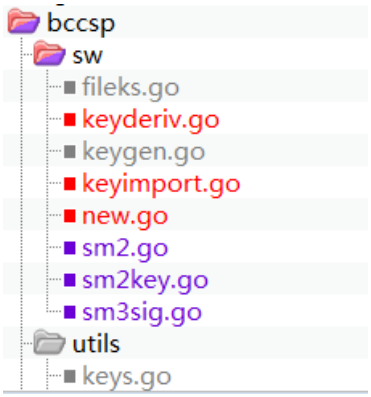
在 vendor 目录中，添加了一个 github.com/cetcxinlian/cryptogm。此库实现了 sm2、sm3、sm4 的算法，以及扩展了 x509、tls 对国密的支持。除此之外，修改了 vendor 中的 grpc 库，扩展功能，支持国密算法。



3. 国密算法支持修改

此部分修改的内容密钥与文件的转换（fileks.go），密钥派生（keyderiv.go），密钥生成（keygen.go），密钥加载（keyimport.go），bccsp 密码处理器初始化

（new.go），pem 与 key 格式互转（keys.go）。以及按照 bccsp 中的调用规范，添了 sm2.go，sm2key.go，sm3sig.go。



以下是每个文件具体的修改：

3.1 fileks.go

3.1.1 GetKey(ski []byte) (bccsp.Key, error)

此函数用于加载密钥，在原有的基础上支持国密国密密钥类型。

```
case "sk":
        // Load the private key
        key, err := ks.loadPrivateKey(hex.EncodeToString(ski))
        if err != nil {
                return nil, fmt.Errorf("Failed loading secret key [%x] [%s]",
        }

        switch key.(type) {
        case *ecdsa.PrivateKey:
                return &ecdsaPrivateKey{key.(*ecdsa.PrivateKey)}, nil
        case *rsa.PrivateKey:
                return &rsaPrivateKey{key.(*rsa.PrivateKey)}, nil
        case *sm2.PrivateKey:
                return &sm2PrivateKey{key.(*sm2.PrivateKey)}, nil
        default:
                return nil, errors.New("Secret key type not recognized")
        }
case "pk":
        // Load the public key
        key, err := ks.loadPublicKey(hex.EncodeToString(ski))
        if err != nil {
                return nil, fmt.Errorf("Failed loading public key [%x] [%s]",
        }

        switch key.(type) {
        case *ecdsa.PublicKey:
                return &ecdsaPublicKey{key.(*ecdsa.PublicKey)}, nil
        case *rsa.PublicKey:
                return &rsaPublicKey{key.(*rsa.PublicKey)}, nil
        case *sm2.PublicKey:
                return &sm2PublicKey{key.(*sm2.PublicKey)}, nil
```

3.1.2 StoreKey(k bccsp.Key) (err error)

此函数用于存储密钥，添加国密密钥支持。

```
case *sm2PrivateKey:
        kk := k.(*sm2PrivateKey)

        err = ks.storePrivateKey(hex.EncodeToString(k.SKI()), kk.privKey)
        if err != nil {
                return fmt.Errorf("Failed storing SM2 private key [%s]", err)
        }

case *sm2PublicKey:
        kk := k.(*sm2PublicKey)

        err = ks.storePublicKey(hex.EncodeToString(k.SKI()), kk.pubKey)
        if err != nil {
                return fmt.Errorf("Failed storing SM2 public key [%s]", err)
        }
```

3.1.3 searchKeystoreForSKI(ski []byte)

通过 ski 加载 key。

```
switch key.(type) {
case *ecdsa.PrivateKey:
        k = &ecdsaPrivateKey{key.(*ecdsa.PrivateKey)}
case *rsa.PrivateKey:
        k = &rsaPrivateKey{key.(*rsa.PrivateKey)}
case *sm2.PrivateKey:
        k = &sm2PrivateKey{key.(*sm2.PrivateKey)}
```

3.2 keyderiv.go

按照密钥派生接口的规范，添加了国密的密钥派生器 sm2PublicKeyKeyDeriver、sm2PrivateKeyKeyDeriver。

```
type sm2PublicKeyKeyDeriver struct{}

func (kd *sm2PublicKeyKeyDeriver) KeyDeriv(k bccsp.Key, opts bccsp.KeyDerivOpts) (bccsp.Key, error) {

type sm2PrivateKeyKeyDeriver struct{}

func (kd *sm2PrivateKeyKeyDeriver) KeyDeriv(k bccsp.Key, opts bccsp.KeyDerivOpts) (bccsp.Key, error)
```

3.3 keygen.go

按照密钥生成器的接口规范，添加国密密钥生成器 sm2KeyGenerator。

```
type sm2KeyGenerator struct {
        curve elliptic.Curve
}

func (kg *sm2KeyGenerator) KeyGen(opts bccsp.KeyGenOpts) (bccsp.Key, error)
```

3.4 keyimport.go

3.4.1 KeyImport(raw interface{},opts bccsp.KeyImportOpts)(bccsp.Key, error)

加载证书中的公钥，添加国密类型公钥的支持。

```
case *ecdsa.PublicKey:
        return ki.bccsp.KeyImporters[reflect.TypeOf(&bccsp.ECDSAGoPublicKeyImportOpts{})].KeyImport(
                pk,
                &bccsp.ECDSAGoPublicKeyImportOpts{Temporary: opts.Ephemeral()})
case *rsa.PublicKey:
        return ki.bccsp.KeyImporters[reflect.TypeOf(&bccsp.RSAGoPublicKeyImportOpts{})].KeyImport(
                pk,
                &bccsp.RSAGoPublicKeyImportOpts{Temporary: opts.Ephemeral()})
case *sm2.PublicKey:
        return ki.bccsp.KeyImporters[reflect.TypeOf(&bccsp.SM2GoPublicKeyImportOpts{})].KeyImport(
                pk.(*sm2.PublicKey), &bccsp.SM2GoPublicKeyImportOpts{Temporary: opts.Ephemeral()})
```

按照密钥加载接口，添加国密类型支持

```
type sm2PKIXPublicKeyImportOptsKeyImporter struct{}

func (*sm2PKIXPublicKeyImportOptsKeyImporter) KeyImport(raw interface{}, opts bccsp.KeyImportOpts) (bccsp.Key, error)

type sm2PrivateKeyImportOptsKeyImporter struct{}

func (*sm2PrivateKeyImportOptsKeyImporter) KeyImport(raw interface{}, opts bccsp.KeyImportOpts) (bccsp.Key, error)
```

3.5 new.go

初始化密码算法管理器，将添加的国密算法加入到 swbccsp 中。

```
// Set the Signers
swbccsp.AddWrapper(reflect.TypeOf(&sm2PrivateKey{}), &sm2Signer{})
swbccsp.AddWrapper(reflect.TypeOf(&ecdsaPrivateKey{}), &ecdsaSigner{})
swbccsp.AddWrapper(reflect.TypeOf(&rsaPrivateKey{}), &rsaSigner{})

// Set the Verifiers
swbccsp.AddWrapper(reflect.TypeOf(&sm2PrivateKey{}), &sm2PrivateKeyVerifier{})
swbccsp.AddWrapper(reflect.TypeOf(&sm2PublicKey{}), &sm2PublicKeyKeyVerifier{})
swbccsp.AddWrapper(reflect.TypeOf(&ecdsaPrivateKey{}), &ecdsaPrivateKeyVerifier{})

// Set the Hashers
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SM3Opts{}), &hasher{hash: sm3.New})
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SHAOpts{}), &hasher{hash: conf.hashFunction})
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SHA256Opts{}), &hasher{hash: sha256.New})
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SHA384Opts{}), &hasher{hash: sha512.New384})
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SHA3_256Opts{}), &hasher{hash: sha3.New256})
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SHA3_384Opts{}), &hasher{hash: sha3.New384})

// Set the key generators
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SM2KeyGenOpts{}), &sm2KeyGenerator{})
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.ECDSAKeyGenOpts{}), &ecdsaKeyGenerator{curve: conf.ellipticCurve})

// Set the key derivers
swbccsp.AddWrapper(reflect.TypeOf(&sm2PrivateKey{}), &sm2PrivateKeyKeyDeriver{})
swbccsp.AddWrapper(reflect.TypeOf(&sm2PublicKey{}), &sm2PublicKeyKeyDeriver{})
swbccsp.AddWrapper(reflect.TypeOf(&ecdsaPrivateKey{}), &ecdsaPrivateKeyKeyDeriver{})
swbccsp.AddWrapper(reflect.TypeOf(&ecdsaPublicKey{}), &ecdsaPublicKeyKeyDeriver{})
swbccsp.AddWrapper(reflect.TypeOf(&aesPrivateKey{}), &aesPrivateKeyKeyDeriver{conf: conf})

// Set the key importers
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SM2PKIXPublicKeyImportOpts{}), &sm2PKIXPublicKeyImportOptsKeyImporter{})
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SM2PrivateKeyImportOpts{}), &sm2PrivateKeyImportOptsKeyImporter{})
swbccsp.AddWrapper(reflect.TypeOf(&bccsp.SM2GoPublicKeyImportOpts{}), &sm2GoPublicKeyImportOptsKeyImporter{})
```

3.6 keys.go

pem 与 key 格式互转，添加国密类型公钥、私钥支持。

3.6.1 PrivateKeyToPEM(privateKey interface{},pwd []byte) ([]byte, error)

私钥格式 key 转为 pem 格式。

```
switch k := privateKey.(type) {
case *ecdsa.PrivateKey:

case *rsa.PrivateKey:

case *sm2.PrivateKey:
```

3.6.2 DERToPrivateKey(der []byte) (key interface{}, err error)

der 格式字节转为私钥的 key。

```
if key, err = x509.ParsePKCS8PrivateKey(der); err == nil {
        switch key.(type) {
        case *rsa.PrivateKey, *ecdsa.PrivateKey, *sm2.PrivateKey:
                return
```

3.6.3 PublicKeyToPEM(publicKey interface{}, pwd []byte) ([]byte, error)

公钥类型 key 转为 pem。

```
switch k := publicKey.(type) {
case *ecdsa.PublicKey:

case *rsa.PublicKey:

case *sm2.PublicKey:
```

3.6.4 PublicKeyToDER(publicKey interface{}) ([]byte, error)

公钥类型 key 转为 der。

```
switch k := publicKey.(type) {
case *ecdsa.PublicKey:
case *rsa.PublicKey:
case *sm2.PublicKey:
```

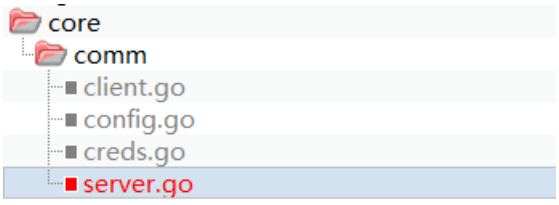
4. 国密证书修改

将 fabric 中 x509 库由"crypto/x509"修改为"github.com/cetcxinlian/cryptogm/x509"。

5 tls 修改

5.1 文件修改

修改通信过程中关于通信协议，和通信密码套件。主要修改文件为以下文件：



以下分别介绍每个文件的修改内容：

5.1.1 client.go

通信客户端，添加国密 tls 通信支持。

1) parseSecureOptions(opts *SecureOptions) error

处理安全选项，根据服务端证书类型，选择 tls 通信过程支持的协议号。

```
if len(opts.ServerRootCAs) > 0 {
        client.tlsConfig.RootCAs = x509.NewCertPool()
        for _, certBytes := range opts.ServerRootCAs {
                err := AddPemToCertPool(certBytes, client.tlsConfig.RootCAs)
                if err != nil {
                        commLogger.Debugf("error adding root certificate: %v", err)
                        return errors.WithMessage(err,
                                "error adding root certificate")
                }
        }
        block, _ := pem.Decode(opts.ServerRootCAs[0])
        if block != nil {
                caCert, err := x509.ParseCertificate(block.Bytes)
                if err != nil {
                        return errors.WithMessage(err, "FMT0024, parse certificate error")
                }
                _, ok := caCert.PublicKey.(*sm2.PublicKey)
                if ok {
                        client.tlsConfig.GMSupport = &tls.GMSupport{}
                        client.tlsConfig.MinVersion = 0
                }
        }
}
```

5.1.2 config.go

通信配置相关定义，添加国密通信套件定义。

```
DefaultTLSCipherSuites = []uint16{
        tls.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
        tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
        tls.TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
        tls.TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,
        tls.TLS_RSA_WITH_AES_128_GCM_SHA256,
        tls.TLS_RSA_WITH_AES_256_GCM_SHA384,
}
// strong GM TLS cipher suites
DefaultGMTLSCipherSuites = []uint16{
        tls.GMTLS_SM2_WITH_SM4_SM3,
}
```

5.1.3 server.go

服务端根据证书私钥类型选择安全配置的密码套件。

```
grpcServer.serverCertificate.Store(cert)

//set up our TLS config
if len(secureConfig.CipherSuites) == 0 {
        // gmtls support
        if _, ok := cert.PrivateKey.(*sm2.PrivateKey); !ok {
                secureConfig.CipherSuites = DefaultTLSCipherSuites
        } else {
                secureConfig.CipherSuites = DefaultGMTLSCipherSuites
        }
}
```

5.1.4 creds.go

根据通信类型，设置通信协议支持的版本。

```
func NewServerTransportCredentials(
        serverConfig *tls.Config,
        logger *flogging.FabricLogger) credentials.TransportCredentials {

        // NOTE: unlike the default grpc/credentials implementation, we do not
        // clone the tls.Config which allows us to update it dynamically
        serverConfig.NextProtos = alpnProtoStr
        // override TLS version and ensure it is 1.2

        // GMTLS support
        if serverConfig.GMSupport != nil {
                serverConfig.MinVersion = tls.VersionGMSSL
        } else {
                serverConfig.MinVersion = tls.VersionTLS12
        }
        serverConfig.MaxVersion = tls.VersionTLS12
        return &serverCreds{
                serverConfig: serverConfig,
                logger:       logger}
}
```

5.2 库替换

将 fabric 中 tls 库由"crypto/tls"修改为"github.com/cetcxinlian/cryptogm/tls"。

6 库文件

6.1 国密密码库

https://github.com/cetcxinlian/cryptogm
国密密码库，实现了 sm2、sm3、sm4、x509、tls。

6.2 fabric 国密支持库

https://gitee.com/suchongming/fabric.git
包含两个分支，分别是 v1.4.6 与 v2.0.0 两个版本的国密化后的代码。