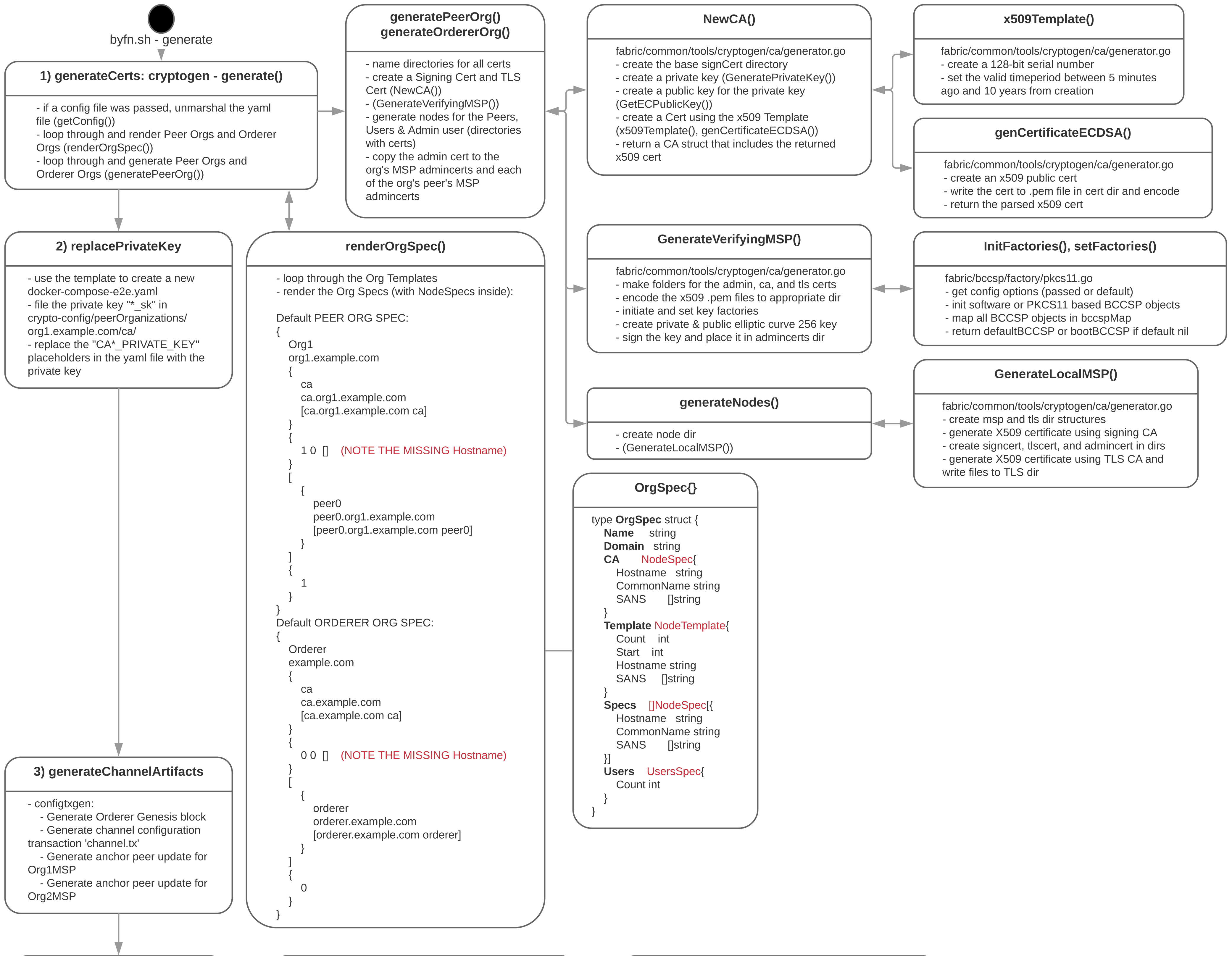


1) generateCerts: cryptogen - generate()

- if a config file was passed, unmarshal the yaml file (getConfig())
- loop through and render Peer Orgs and Orderer Orgs (renderOrgSpec())
- loop through and generate Peer Orgs and Orderer Orgs (generatePeerOrg())

2) replacePrivateKey

- use the template to create a new docker-compose-e2e.yaml
- file the private key "*_sk" in crypto-config/peerOrganizations/org1.example.com/ca/
- replace the "CA*_PRIVATE_KEY" placeholders in the yaml file with the private key

3) generateChannelArtifacts

- configtxgen:
  - Generate Orderer Genesis block
  - Generate channel configuration transaction 'channel.tx'
  - Generate anchor peer update for Org1MSP
  - Generate anchor peer update for Org2MSP

generatePeerOrg()
generateOrdererOrg()

- name directories for all certs
- create a Signing Cert and TLS Cert (NewCA())
- (GenerateVerifyingMSP())
- generate nodes for the Peers, Users & Admin user (directories with certs)
- copy the admin cert to the org's MSP admincerts and each of the org's peer's MSP admincerts

renderOrgSpec()

```
- loop through the Org Templates
- render the Org Specs (with NodeSpecs inside):

Default PEER ORG SPEC:
{
   Org1
   org1.example.com
   {
      ca
      ca.org1.example.com
      [ca.org1.example.com ca]
   }
   {
      1 0  []    (NOTE THE MISSING Hostname)
   }
   [
      {
         peer0
         peer0.org1.example.com
         [peer0.org1.example.com peer0]
      }
   ]
   {
      1
   }
}
Default ORDERER ORG SPEC:
{
   Orderer
   example.com
   {
      ca
      ca.example.com
      [ca.example.com ca]
   }
   {
      0 0  []    (NOTE THE MISSING Hostname)
   }
   [
      {
         orderer
         orderer.example.com
         [orderer.example.com orderer]
      }
   ]
   {
      0
   }
}
```

NewCA()

fabric/common/tools/cryptogen/ca/generator.go
- create the base signCert directory
- create a private key (GeneratePrivateKey())
- create a public key for the private key (GetECPublicKey())
- create a Cert using the x509 Template (x509Template(), genCertificateECDSA())
- return a CA struct that includes the returned x509 cert

x509Template()

fabric/common/tools/cryptogen/ca/generator.go
- create a 128-bit serial number
- set the valid timeperiod between 5 minutes ago and 10 years from creation

genCertificateECDSA()

fabric/common/tools/cryptogen/ca/generator.go
- create an x509 public cert
- write the cert to .pem file in cert dir and encode
- return the parsed x509 cert

GenerateVerifyingMSP()

fabric/common/tools/cryptogen/ca/generator.go
- make folders for the admin, ca, and tls certs
- encode the x509 .pem files to appropriate dir
- initiate and set key factories
- create private & public elliptic curve 256 key
- sign the key and place it in admincerts dir

InitFactories(), setFactories()

fabric/bccsp/factory/pkcs11.go
- get config options (passed or default)
- init software or PKCS11 based BCCSP objects
- map all BCCSP objects in bccspMap
- return defaultBCCSP or bootBCCSP if default nil

generateNodes()

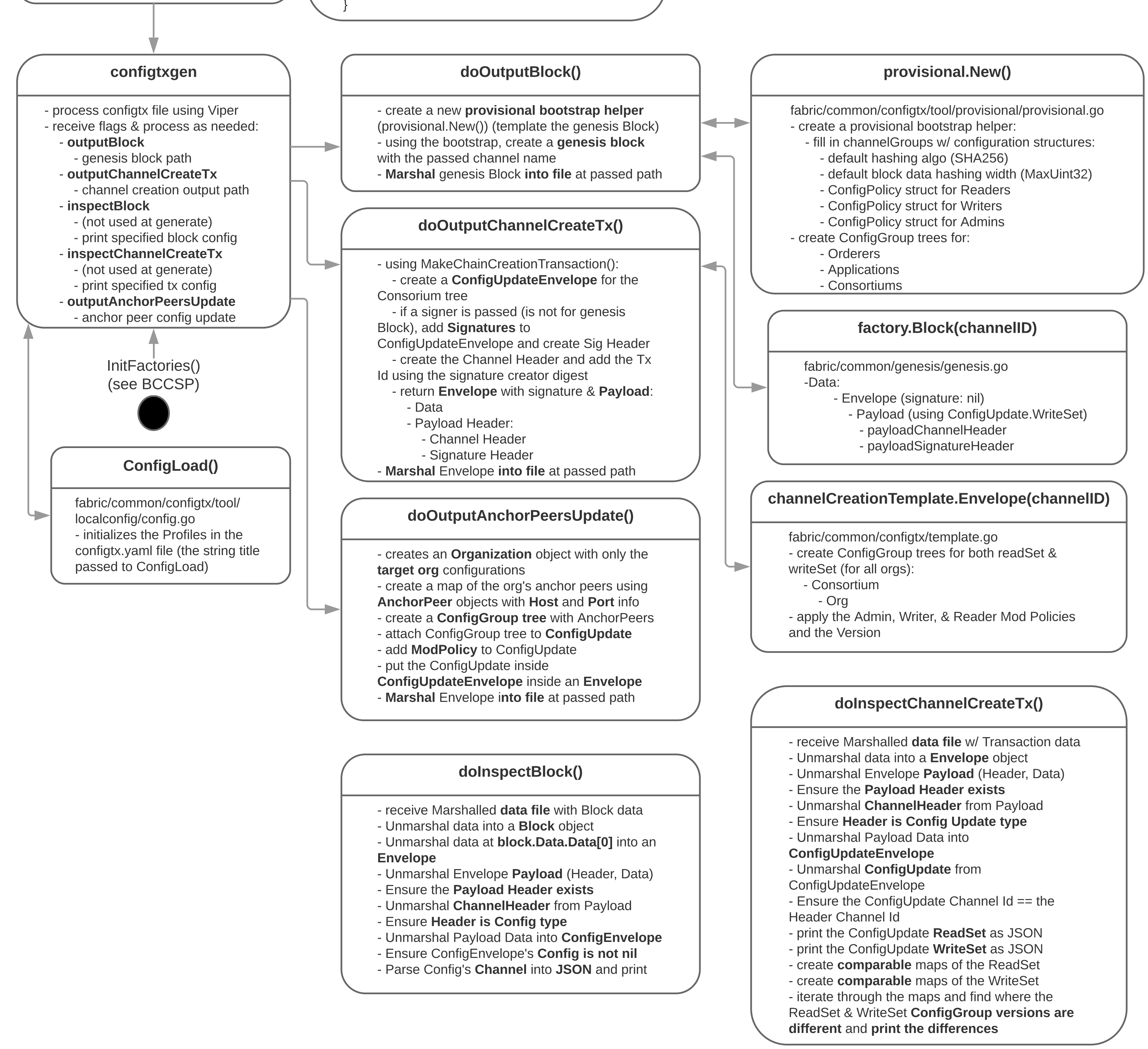
- create node dir
- (GenerateLocalMSP())

GenerateLocalMSP()

fabric/common/tools/cryptogen/ca/generator.go
- create msp and tls dir structures
- generate X509 certificate using signing CA
- create signcert, tlscert, and admincert in dirs
- generate X509 certificate using TLS CA and write files to TLS dir

OrgSpec{}

```
type OrgSpec struct {
   Name     string
   Domain   string
   CA       NodeSpec{
      Hostname   string
      CommonName string
      SANS       []string
   }
   Template NodeTemplate{
      Count    int
      Start    int
      Hostname string
      SANS     []string
   }
   Specs    []NodeSpec[{
      Hostname   string
      CommonName string
      SANS       []string
   }]
   Users    UsersSpec{
      Count int
   }
}
```



configtxgen

- process configtx file using Viper
- receive flags & process as needed:
  - **outputBlock**
    - genesis block path
  - **outputChannelCreateTx**
    - channel creation output path
  - **inspectBlock**
    - (not used at generate)
    - print specified block config
  - **inspectChannelCreateTx**
    - (not used at generate)
    - print specified tx config
  - **outputAnchorPeersUpdate**
    - anchor peer config update

InitFactories()
(see BCCSP)

ConfigLoad()

fabric/common/configtx/tool/localconfig/config.go
- initializes the Profiles in the configtx.yaml file (the string title passed to ConfigLoad)

doOutputBlock()

- create a new **provisional bootstrap helper** (provisional.New()) (template the genesis Block)
- using the bootstrap, create a **genesis block** with the passed channel name
- **Marshal** genesis Block **into file** at passed path

doOutputChannelCreateTx()

- using MakeChainCreationTransaction():
  - create a **ConfigUpdateEnvelope** for the Consorium tree
  - if a signer is passed (is not for genesis Block), add **Signatures** to ConfigUpdateEnvelope and create Sig Header
  - create the Channel Header and add the Tx Id using the signature creator digest
  - return **Envelope** with signature & **Payload**:
    - Data
    - Payload Header:
      - Channel Header
      - Signature Header
- **Marshal** Envelope **into file** at passed path

doOutputAnchorPeersUpdate()

- creates an **Organization** object with only the **target org** configurations
- create a map of the org's anchor peers using **AnchorPeer** objects with **Host** and **Port** info
- create a **ConfigGroup tree** with AnchorPeers
- attach ConfigGroup tree to **ConfigUpdate**
- add **ModPolicy** to ConfigUpdate
- put the ConfigUpdate inside **ConfigUpdateEnvelope** inside an **Envelope**
- **Marshal** Envelope i**nto file** at passed path

doInspectBlock()

- receive Marshalled **data file** with Block data
- Unmarshal data into a **Block** object
- Unmarshal data at **block.Data.Data[0]** into an **Envelope**
- Unmarshal Envelope **Payload** (Header, Data)
- Ensure the **Payload Header exists**
- Unmarshal **ChannelHeader** from Payload
- Ensure **Header is Config type**
- Unmarshal Payload Data into **ConfigEnvelope**
- Ensure ConfigEnvelope's **Config is not nil**
- Parse Config's **Channel** into **JSON** and print

provisional.New()

fabric/common/configtx/tool/provisional/provisional.go
- create a provisional bootstrap helper:
  - fill in channelGroups w/ configuration structures:
    - default hashing algo (SHA256)
    - default block data hashing width (MaxUint32)
    - ConfigPolicy struct for Readers
    - ConfigPolicy struct for Writers
    - ConfigPolicy struct for Admins
- create ConfigGroup trees for:
  - Orderers
  - Applications
  - Consortiums

factory.Block(channelID)

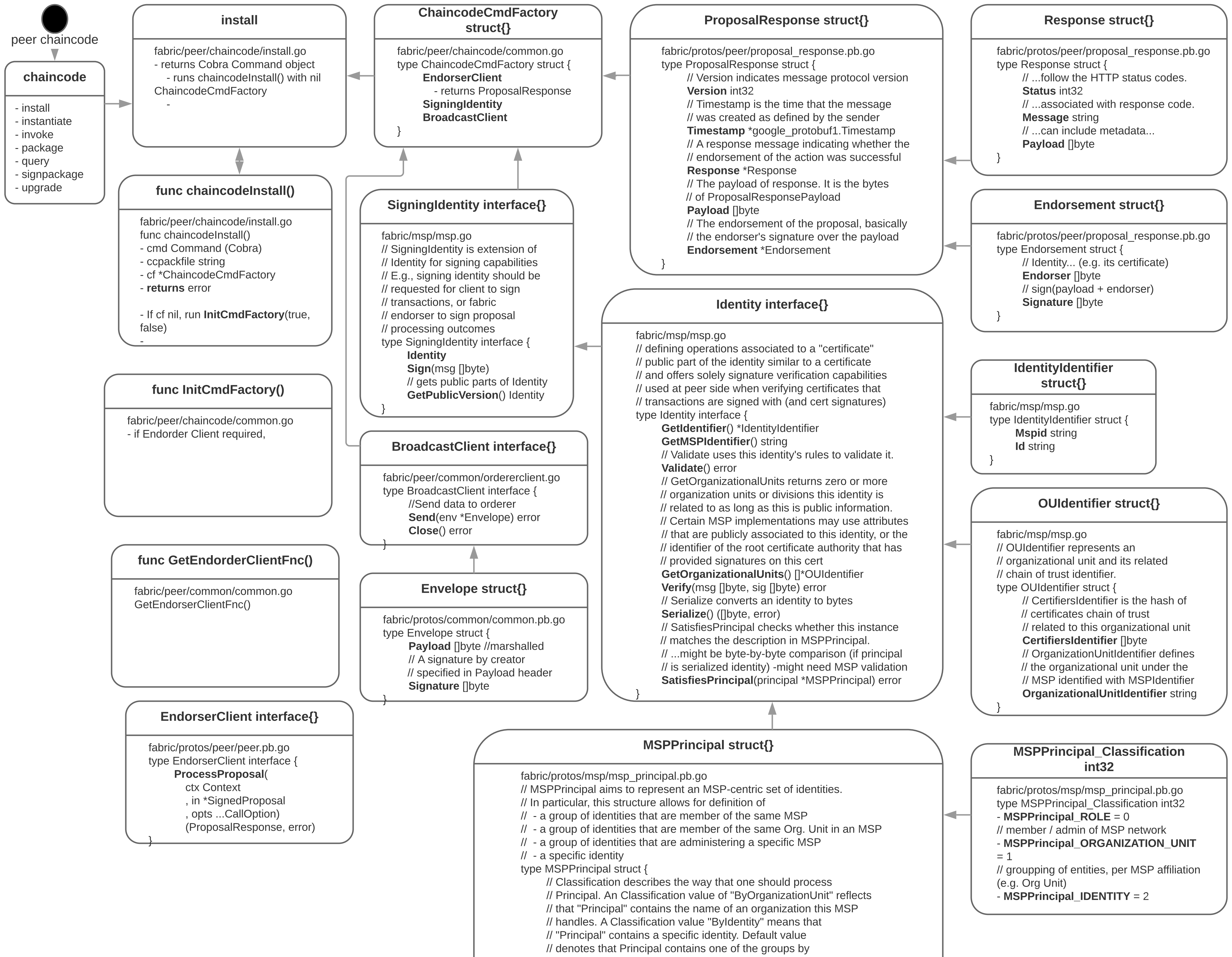
fabric/common/genesis/genesis.go
-Data:
- Envelope (signature: nil)
  - Payload (using ConfigUpdate.WriteSet)
    - payloadChannelHeader
    - payloadSignatureHeader

channelCreationTemplate.Envelope(channelID)

fabric/common/configtx/template.go
- create ConfigGroup trees for both readSet & writeSet (for all orgs):
  - Consortium
    - Org
- apply the Admin, Writer, & Reader Mod Policies and the Version

doInspectChannelCreateTx()

- receive Marshalled **data file** w/ Transaction data
- Unmarshal data into a **Envelope** object
- Unmarshal Envelope **Payload** (Header, Data)
- Ensure the **Payload Header exists**
- Unmarshal **ChannelHeader** from Payload
- Ensure **Header is Config Update type**
- Unmarshal Payload Data into **ConfigUpdateEnvelope**
- Unmarshal **ConfigUpdate** from ConfigUpdateEnvelope
- Ensure the ConfigUpdate Channel Id == the Header Channel Id
- print the ConfigUpdate **ReadSet** as JSON
- print the ConfigUpdate **WriteSet** as JSON
- create **comparable** maps of the ReadSet
- create **comparable** maps of the WriteSet
- iterate through the maps and find where the ReadSet & WriteSet **ConfigGroup versions are different** and **print the differences**



chaincode

- install
- instantiate
- invoke
- package
- query
- signpackage
- upgrade

install

fabric/peer/chaincode/install.go
- returns Cobra Command object
  - runs chaincodeInstall() with nil ChaincodeCmdFactory
  -

func chaincodeInstall()

fabric/peer/chaincode/install.go
func chaincodeInstall()
- cmd Command (Cobra)
- ccpackfile string
- cf *ChaincodeCmdFactory
- **returns** error

- If cf nil, run **InitCmdFactory**(true, false)
-

func InitCmdFactory()

fabric/peer/chaincode/common.go
- if Endorder Client required,

func GetEndorderClientFnc()

fabric/peer/common/common.go
GetEndorserClientFnc()

EndorserClient interface{}

```
fabric/protos/peer/peer.pb.go
type EndorserClient interface {
        ProcessProposal(
            ctx Context
            , in *SignedProposal
            , opts ...CallOption)
            (ProposalResponse, error)
}
```

ChaincodeCmdFactory struct{}

```
fabric/peer/chaincode/common.go
type ChaincodeCmdFactory struct {
        EndorserClient
            - returns ProposalResponse
        SigningIdentity
        BroadcastClient
}
```

SigningIdentity interface{}

```
fabric/msp/msp.go
// SigningIdentity is extension of
// Identity for signing capabilities
// E.g., signing identity should be
// requested for client to sign
// transactions, or fabric
// endorser to sign proposal
// processing outcomes
type SigningIdentity interface {
        Identity
        Sign(msg []byte)
        // gets public parts of Identity
        GetPublicVersion() Identity
}
```

BroadcastClient interface{}

```
fabric/peer/common/ordererclient.go
type BroadcastClient interface {
        //Send data to orderer
        Send(env *Envelope) error
        Close() error
}
```

Envelope struct{}

```
fabric/protos/common/common.pb.go
type Envelope struct {
        Payload []byte //marshalled
        // A signature by creator
        // specified in Payload header
        Signature []byte
}
```

ProposalResponse struct{}

```
fabric/protos/peer/proposal_response.pb.go
type ProposalResponse struct {
        // Version indicates message protocol version
        Version int32
        // Timestamp is the time that the message
        // was created as defined by the sender
        Timestamp *google_protobuf1.Timestamp
        // A response message indicating whether the
        // endorsement of the action was successful
        Response *Response
        // The payload of response. It is the bytes
        // of ProposalResponsePayload
        Payload []byte
        // The endorsement of the proposal, basically
        // the endorser's signature over the payload
        Endorsement *Endorsement
}
```

Response struct{}

```
fabric/protos/peer/proposal_response.pb.go
type Response struct {
        // ...follow the HTTP status codes.
        Status int32
        // ...associated with response code.
        Message string
        // ...can include metadata...
        Payload []byte
}
```

Endorsement struct{}

```
fabric/protos/peer/proposal_response.pb.go
type Endorsement struct {
        // Identity... (e.g. its certificate)
        Endorser []byte
        // sign(payload + endorser)
        Signature []byte
}
```

Identity interface{}

```
fabric/msp/msp.go
// defining operations associated to a "certificate"
// public part of the identity similar to a certificate
// and offers solely signature verification capabilities
// used at peer side when verifying certificates that
// transactions are signed with (and cert signatures)
type Identity interface {
        GetIdentifier() *IdentityIdentifier
        GetMSPIdentifier() string
        // Validate uses this identity's rules to validate it.
        Validate() error
        // GetOrganizationalUnits returns zero or more
        // organization units or divisions this identity is
        // related to as long as this is public information.
        // Certain MSP implementations may use attributes
        // that are publicly associated to this identity, or the
        // identifier of the root certificate authority that has
        // provided signatures on this cert
        GetOrganizationalUnits() []*OUIdentifier
        Verify(msg []byte, sig []byte) error
        // Serialize converts an identity to bytes
        Serialize() ([]byte, error)
        // SatisfiesPrincipal checks whether this instance
        // matches the description in MSPPrincipal.
        // ...might be byte-by-byte comparison (if principal
        // is serialized identity) -might need MSP validation
        SatisfiesPrincipal(principal *MSPPrincipal) error
}
```

IdentityIdentifier struct{}

```
fabric/msp/msp.go
type IdentityIdentifier struct {
        Mspid string
        Id string
}
```

OUIdentifier struct{}

```
fabric/msp/msp.go
// OUIdentifier represents an
// organizational unit and its related
// chain of trust identifier.
type OUIdentifier struct {
        // CertifiersIdentifier is the hash of
        // certificates chain of trust
        // related to this organizational unit
        CertifiersIdentifier []byte
        // OrganizationUnitIdentifier defines
        // the organizational unit under the
        // MSP identified with MSPIdentifier
        OrganizationalUnitIdentifier string
}
```

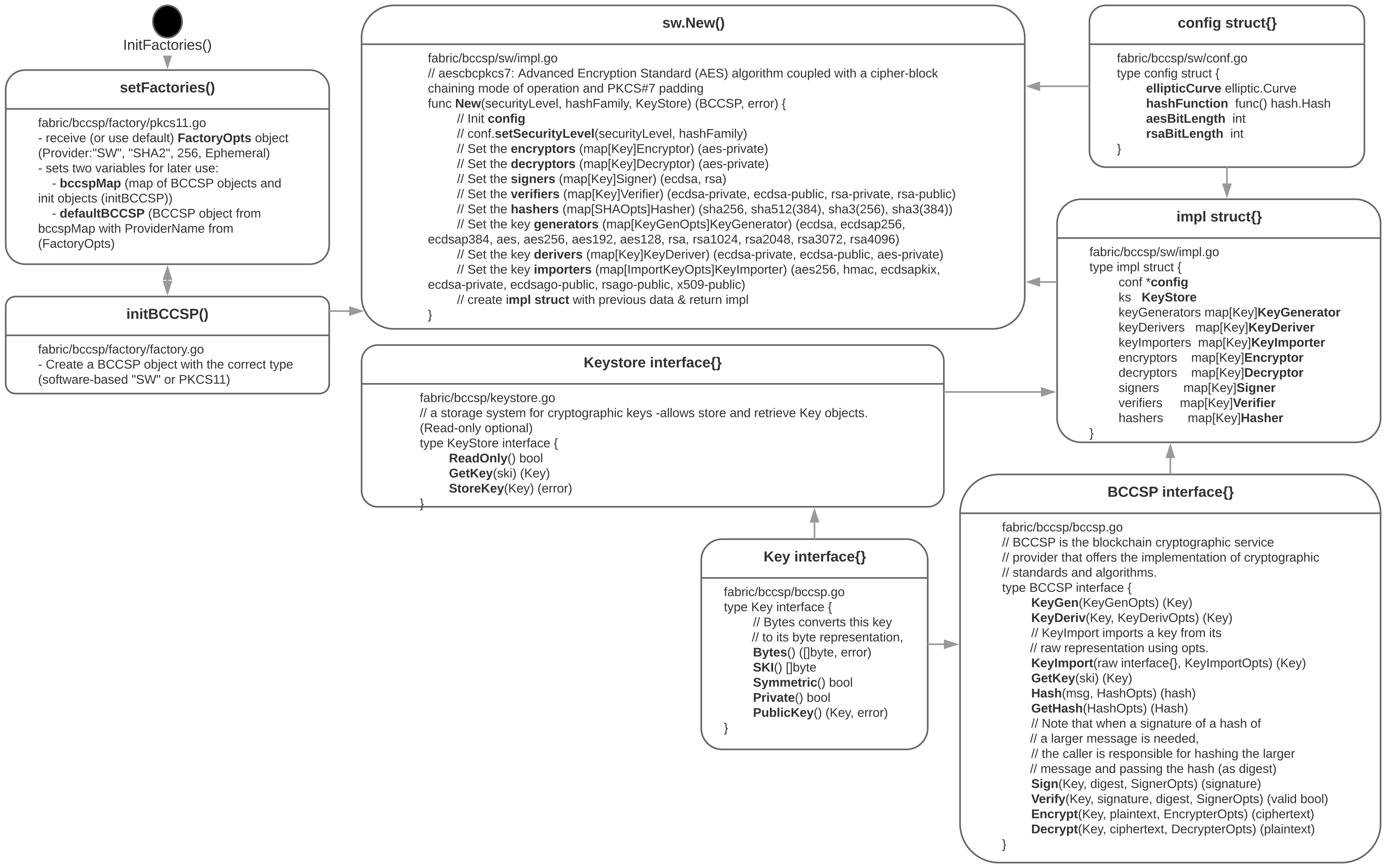
MSPPrincipal struct{}

```
fabric/protos/msp/msp_principal.pb.go
// MSPPrincipal aims to represent an MSP-centric set of identities.
// In particular, this structure allows for definition of
//  - a group of identities that are member of the same MSP
//  - a group of identities that are member of the same Org. Unit in an MSP
//  - a group of identities that are administering a specific MSP
//  - a specific identity
type MSPPrincipal struct {
        // Classification describes the way that one should process
        // Principal. An Classification value of "ByOrganizationUnit" reflects
        // that "Principal" contains the name of an organization this MSP
        // handles. A Classification value "ByIdentity" means that
        // "Principal" contains a specific identity. Default value
        // denotes that Principal contains one of the groups by
```

MSPPrincipal_Classification int32

fabric/protos/msp/msp_principal.pb.go
type MSPPrincipal_Classification int32
- **MSPPrincipal_ROLE** = 0
// member / admin of MSP network
- **MSPPrincipal_ORGANIZATION_UNIT** = 1
// groupping of entities, per MSP affiliation (e.g. Org Unit)
- **MSPPrincipal_IDENTITY** = 2

```
        // that "Principal" contains the name of an organization this MSP
        // handles. A Classification value "ByIdentity" means that
        // "Principal" contains a specific identity. Default value
        // denotes that Principal contains one of the groups by
        // default supported by all MSPs ("admin" or "member").
        PrincipalClassification MSPPrincipal_Classification
        // Principal completes the policy principal definition. For the default
        // principal types, Principal can be either "Admin" or "Member".
        // For the ByOrganizationUnit/ByIdentity values of Classification,
        // PolicyPrincipal acquires its value from an organization unit or
        // identity, respectively.
        Principal []byte
}
```



InitFactories()

setFactories()

fabric/bccsp/factory/pkcs11.go
- receive (or use default) **FactoryOpts** object (Provider:"SW", "SHA2", 256, Ephemeral)
- sets two variables for later use:
  - **bccspMap** (map of BCCSP objects and init objects (initBCCSP))
  - **defaultBCCSP** (BCCSP object from bccspMap with ProviderName from (FactoryOpts)

initBCCSP()

fabric/bccsp/factory/factory.go
- Create a BCCSP object with the correct type (software-based "SW" or PKCS11)

sw.New()

```
fabric/bccsp/sw/impl.go
// aescbcpkcs7: Advanced Encryption Standard (AES) algorithm coupled with a cipher-block
chaining mode of operation and PKCS#7 padding
func New(securityLevel, hashFamily, KeyStore) (BCCSP, error) {
        // Init config
        // conf.setSecurityLevel(securityLevel, hashFamily)
        // Set the encryptors (map[Key]Encryptor) (aes-private)
        // Set the decryptors (map[Key]Decryptor) (aes-private)
        // Set the signers (map[Key]Signer) (ecdsa, rsa)
        // Set the verifiers (map[Key]Verifier) (ecdsa-private, ecdsa-public, rsa-private, rsa-public)
        // Set the hashers (map[SHAOpts]Hasher) (sha256, sha512(384), sha3(256), sha3(384))
        // Set the key generators (map[KeyGenOpts]KeyGenerator) (ecdsa, ecdsap256,
ecdsap384, aes, aes256, aes192, aes128, rsa, rsa1024, rsa2048, rsa3072, rsa4096)
        // Set the key derivers (map[Key]KeyDeriver) (ecdsa-private, ecdsa-public, aes-private)
        // Set the key importers (map[ImportKeyOpts]KeyImporter) (aes256, hmac, ecdsapkix,
ecdsa-private, ecdsago-public, rsago-public, x509-public)
        // create impl struct with previous data & return impl
}
```

config struct{}

```
fabric/bccsp/sw/conf.go
type config struct {
        ellipticCurve elliptic.Curve
        hashFunction  func() hash.Hash
        aesBitLength  int
        rsaBitLength  int
}
```

impl struct{}

```
fabric/bccsp/sw/impl.go
type impl struct {
        conf *config
        ks   KeyStore
        keyGenerators map[Key]KeyGenerator
        keyDerivers   map[Key]KeyDeriver
        keyImporters  map[Key]KeyImporter
        encryptors    map[Key]Encryptor
        decryptors    map[Key]Decryptor
        signers       map[Key]Signer
        verifiers     map[Key]Verifier
        hashers       map[Key]Hasher
}
```

Keystore interface{}

```
fabric/bccsp/keystore.go
// a storage system for cryptographic keys -allows store and retrieve Key objects.
(Read-only optional)
type KeyStore interface {
        ReadOnly() bool
        GetKey(ski) (Key)
        StoreKey(Key) (error)
}
```

Key interface{}

```
fabric/bccsp/bccsp.go
type Key interface {
        // Bytes converts this key
        // to its byte representation,
        Bytes() ([]byte, error)
        SKI() []byte
        Symmetric() bool
        Private() bool
        PublicKey() (Key, error)
}
```

BCCSP interface{}

```
fabric/bccsp/bccsp.go
// BCCSP is the blockchain cryptographic service
// provider that offers the implementation of cryptographic
// standards and algorithms.
type BCCSP interface {
        KeyGen(KeyGenOpts) (Key)
        KeyDeriv(Key, KeyDerivOpts) (Key)
        // KeyImport imports a key from its
        // raw representation using opts.
        KeyImport(raw interface{}, KeyImportOpts) (Key)
        GetKey(ski) (Key)
        Hash(msg, HashOpts) (hash)
        GetHash(HashOpts) (Hash)
        // Note that when a signature of a hash of
        // a larger message is needed,
        // the caller is responsible for hashing the larger
        // message and passing the hash (as digest)
        Sign(Key, digest, SignerOpts) (signature)
        Verify(Key, signature, digest, SignerOpts) (valid bool)
        Encrypt(Key, plaintext, EncrypterOpts) (ciphertext)
        Decrypt(Key, ciphertext, DecrypterOpts) (plaintext)
}
```