# Blockchain Crypto Service Provider

Elli Androulaki, Angelo De Caro, Volodymyr (V.) Paprotski, Alessandro Sorniotti, Tamas Visegrady

# What is the BCCSP?

BCCSP is the ***Blockchain Cryptographic Service Provider*** that offers the implementation of cryptographic standards and algorithms.

bccsp.go

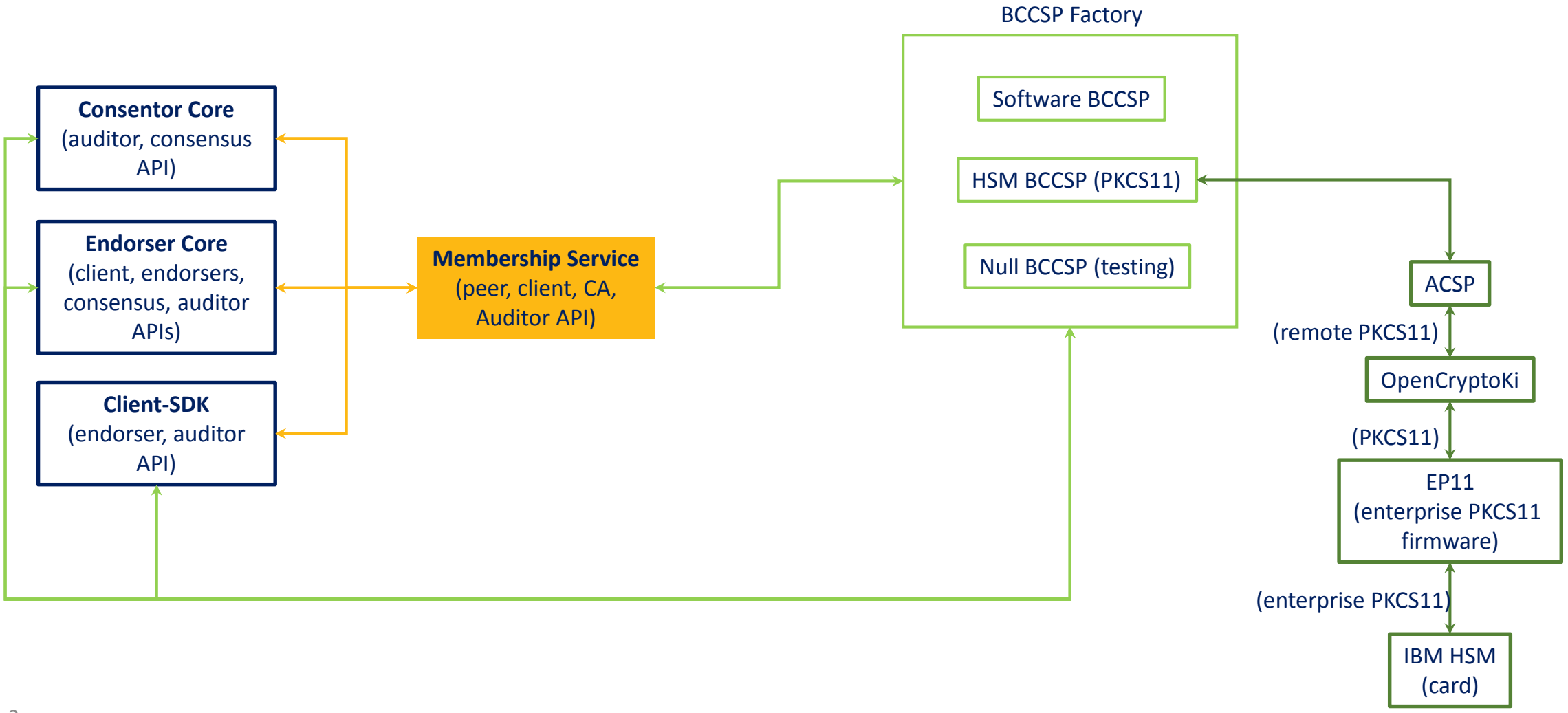
```
type BCCSP interface {

        GenKey(opts GenKeyOpts) (k Key, err error)

        DeriveKey(k Key, opts DeriveKeyOpts) (dk Key, err error)

        GetKey(ski []byte) (k Key, err error)

        ImportKey(raw []byte, opts ImportKeyOpts) (k Key, err error)

        Sign(k Key, digest []byte, opts SignerOpts) (signature []byte, err error)

        Verify(k Key, signature, digest []byte) (valid bool, err error)

        Encrypt(k Key, plaintext []byte, opts EncrypterOpts) (ciphertext []byte, err error)

        Decrypt(k Key, ciphertext []byte, opts DecrypterOpts) (plaintext []byte, err error)
}
```

Key lifecycle

Sign/Verify operations

Encrypt/Decrypt operations

# Integrating BCCSP in Hyperledger Fabric



# BCCSP Design Goals

- **Pluggability**
  - alternate implementations of crypto interface can be used within the HPL/fabric code, **without modifying** the core

- Support for **Multiple CSPs**
  - Easy addition of more types of CSPs, e.g., of different HSM types
  - Enable the use of different CSP on different system components transparently

- International **Standards** Support
  - E.g., via a new/separate CSP
  - Interoperability among standards is not necessarily guaranteed

# BCCSP Key

bccsp.go

```
type Key interface {

        // ToByte converts this key to its byte representation,
        // if this operation is allowed.
        ToByte() ([]byte, error)

        // GetSKI returns the subject key identifier of this key.
        GetSKI() []byte

        // Symmetric returns true if this key is a symmetric key,
        // false is this key is asymmetric
        Symmetric() (bool)

        // Private returns true if this key is an asymmetric private key,
        // false otherwise.
        Private() (bool)

        // PublicKey returns the corresponding public key if this key
        // is an asymmetric private key. If this key is already public,
        // PublicKey returns this key itself.
        PublicKey() (Key, error)
}
```

Key represents a cryptographic key. It can be **symmetric** or **asymmetric**. In the case of an asymmetric key, the key can be **public** or **private**. In the case of a private asymmetric key, the PublicKey() method allows to retrieve the corresponding public-key.

A key can be referenced via the **Subject Key Identifier** (GetSKI)

# Key Lifecycle

In order to perform any cryptographic operation, proper keys need to be generate…

bccsp.go

```
GenKey(opts GenKeyOpts) (k Key, err error)
```

bccsp.go

```
// GenKeyOpts contains options for key-generation with a CSP.
type GenKeyOpts interface {

        // Algorithm returns an identifier for the algorithm to be used
        // to generate a key.
        Algorithm() string

        // Ephemeral returns true if the key to generate has to be ephemeral,
        // false otherwise.
        Ephemeral() bool
}
```

GenKey allows to generate multiple types of keys depending on *opts.*

At very least, the developer has to specify the algorithm to use to generate the key and declare if the key is ephemeral or not. Non-ephemeral keys (long-term keys) are stored an retrievable at any time using their SKIs.

Ephemeral keys are disposed automatically once not referenced anymore.

Notice that, additional parameters can be passed by adding them at GenKeyOpts implementation time.

# Key Lifecycle

Sometimes, it might be necessary to derive a new key from an existing one…

bccsp.go

```
DeriveKey(k Key, opts DeriveKeyOpts) (dk Key, err error)
```

bccsp.go

```
// DeriveKeyOpts contains options for key-derivation with a CSP.
type DeriveKeyOpts interface {

        // Algorithm returns an identifier for the algorithm to be used
        // to generate a key.
        Algorithm() string

        // Ephemeral returns true if the key to generate has to be ephemeral,
        // false otherwise.
        Ephemeral() bool
}
```

DeriveKey allows to derive a new key from an existing one (by HMACing or by re-randomizing, for example). Multiple types of key derivation are possible by specifying appropriate *opts.*

At very least, the developer has to specify the algorithm to use to derive the key and declare if the key is ephemeral or not.

Notice that, additional parameters can be passed by adding them at DeriveKeyOpts implementation time.

# Sign/Verify Capabilities

bccsp.go

```
// Sign signs digest using key k.
// The opts argument should be appropriate for the primitive used.
//
// 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(k Key, digest []byte, opts SignerOpts) (signature []byte, err error)

// Verify verifies signature against key k and digest
Verify(k Key, signature, digest []byte) (valid bool, err error)
```

bccsp.go

```
// SignerOpts contains options for signing with a CSP.
type SignerOpts interface{}
```

Signatures are supported by the BCCSP by exposing the Sign and Verify methods.

The algorithm used to sign is derived by the key. For instance, if key is an ECDSA key then ECDSA is used to sign.

Additional parameters can be passed to the specific signing algorithm by properly implementing the SignerOpts interface.

# Encrypt/Decrypt Capabilities

bccsp.go

```
// Encrypt encrypts plaintext using key k.
// The opts argument should be appropriate for the primitive used.
Encrypt(k Key, plaintext []byte, opts EncrypterOpts) (ciphertext []byte, err error)

// Decrypt decrypts ciphertext using key k.
// The opts argument should be appropriate for the primitive used.
Decrypt(k Key, ciphertext []byte, opts DecrypterOpts) (plaintext []byte, err error)
```

bccsp.go

```
// SignerOpts contains options for signing with a CSP.
type EncrypterOpts interface{}

// SignerOpts contains options for signing with a CSP.
type DecrypterOpts interface{}
```

Encryption is supported by the BCCSP by exposing the Encrypt and Decrypt methods.

The algorithm used to encrypt/decrypt is derived by the key and opts. For instance, if key is an AES key then opts may specify the mode of operation.

Additional parameters can be passed to the specific signing algorithm by properly implementing the EncrypterOpts/ DecrypterOpts interface.