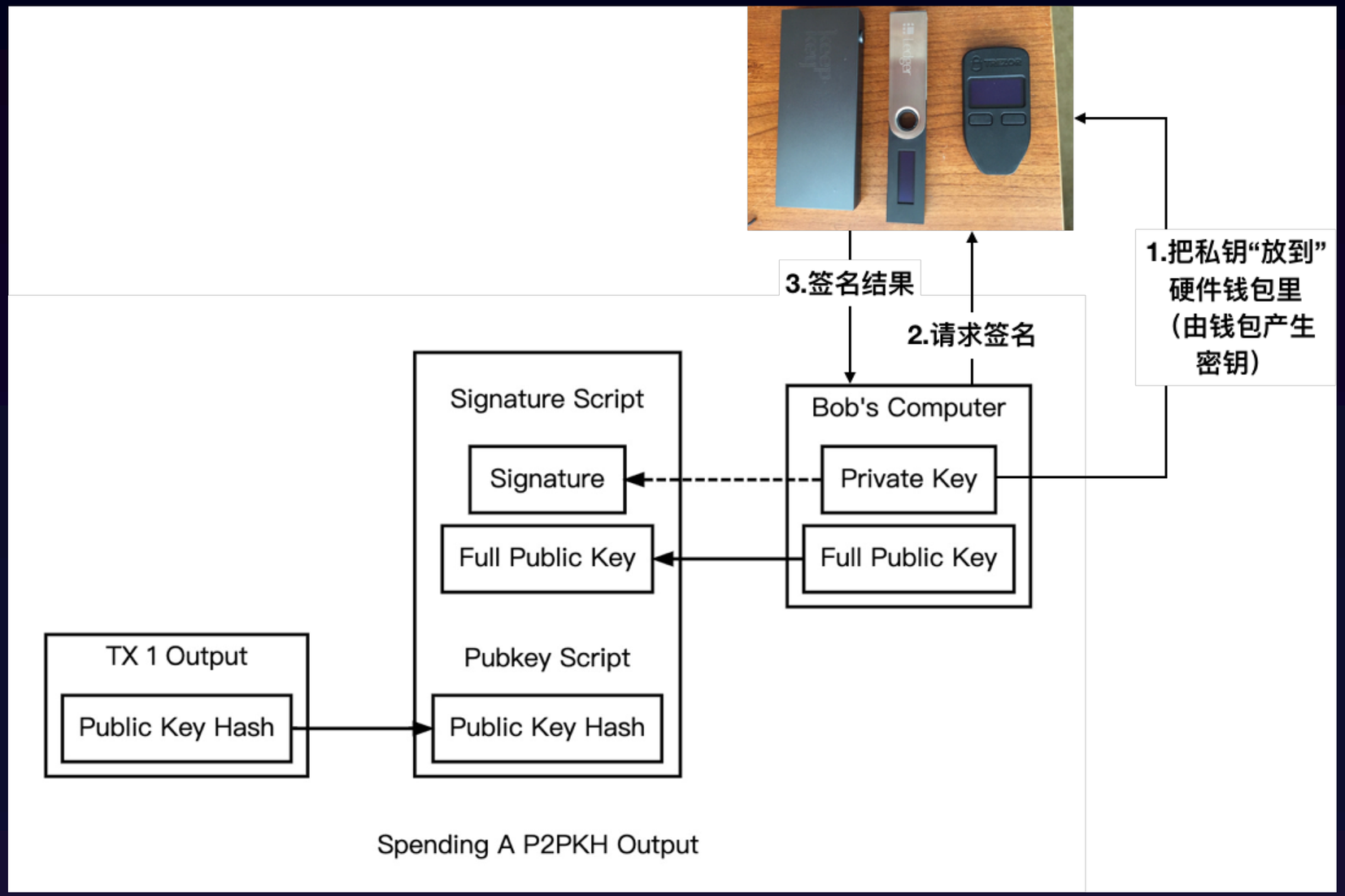
蚂蚁金服
ANT FINANCIAL

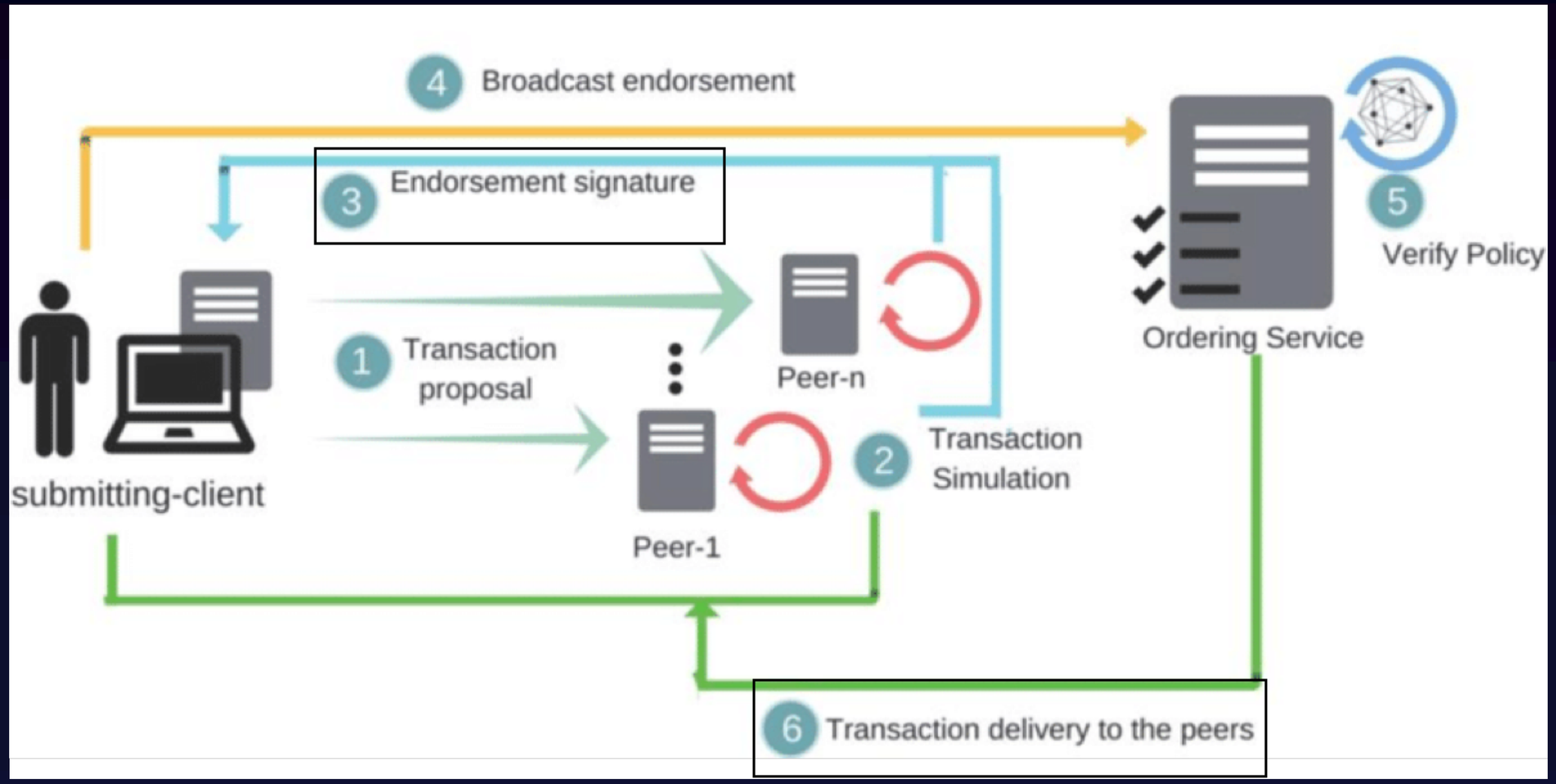
# 为联盟链节点装上“硬件钱包”

使用Intel SGX技术保护Hyperledger Fabric节点私钥

# 比特币与硬件钱包


3.签名结果
2.请求签名
1.把私钥“放到”硬件钱包里（由钱包产生密钥）
Signature Script
Signature
Full Public Key
Bob's Computer
Private Key
Full Public Key
TX 1 Output
Public Key Hash
Pubkey Script
Public Key Hash
Spending A P2PKH Output

# Hyperledger Fabric的节点密钥用途



# Intel SGX技术

蚂蚁金服 ANT FINANCIAL

01

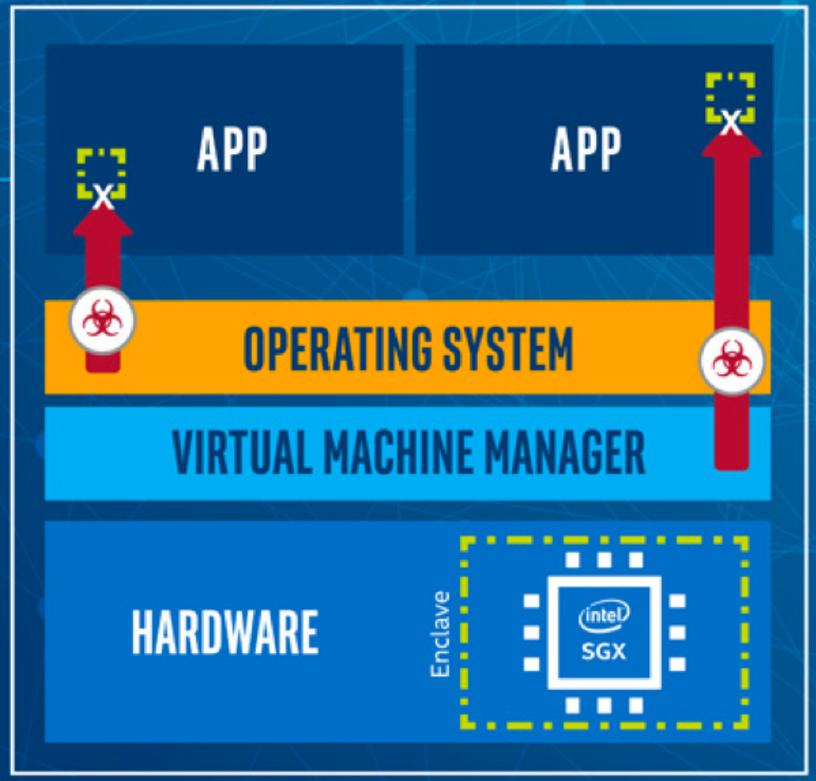
## WHAT IS INTEL® SGX?

Intel® Software Guard Extensions (Intel® SGX)

**What is Intel SGX?**

An Intel architecture extension designed to increase the security of select application code and data, protecting it from disclosure or modification.

Intel processor technologies provide unique capabilities that can improve the privacy, security, and scalability of distributed ledger networks.



# 目标：用Intel SGX保护Fabric节点私钥

蚂蚁金服 ANT FINANCIAL

公共部分：SGX环境准备，Enclave内密码库的实现。

方案1：修改golang crypto lib，劫持ecdsa私钥产生和签名逻辑

方案2：修改fabric bccsp，实现一个基于SGX的plugin bccsp

蚂蚁金服
ANT FINANCIAL

# 准备工作

# 配置SGX软件栈

安装驱动所需内核头文件

例如ubuntu下： apt-get install linux-headers-$(uname -r)

下载对应版本的软件栈安装文件

https://01.org/intel-software-guard-extensions/downloads

依次安装驱动、PSW和SDK

如果是自定义操作系统，则需要下载源文件编译安装

# Linux下编译安装SGX-openssl

确认已经安装SGX SDK

下载sgx-openssl源文件（https://github.com/intel/intel-sgx-ssl）

下载openssl源文件包并放入openssl_source/文件夹

cd Linux

make all install:编译并安装SGX SSL的三个库
libsgx_tsgxssl.a, libsgx_usgxssl.a, libsgx_tsgxssl_crypto.a
默认安装路径是/opt/intel/sgxssl/

# 在容器中运行Enclave

蚂蚁金服 ANT FINANCIAL

/dev/isgx设备的导出

aesm.socket的访问

安装PSW库

蚂蚁金服
ANT FINANCIAL

# 方案1：劫持golang crypto lib

# 基本架构

蚂蚁金服 ANT FINANCIAL

Fabric

Fabric CA

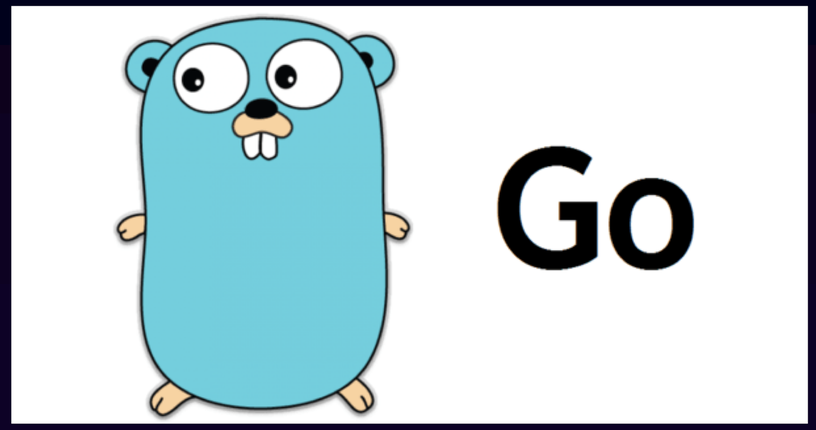
Golang

LibSgxGo

LibSgxWrapper

LibSgxC

Enclave




# 框架的设计

蚂蚁金服 ANT FINANCIAL

LibSgxGo

Cgo文件，在作为Go语言包，调用LibSgxWrapper.so中的C语言代码

LibSgxWrapper.so

通过动态库调用接口(dlopen)动态控制LibSgx.so的代码

保证环境兼容，调试，以及简化语言和语言编译中的问题

LibSgx.so

Enclave代码对应的外部非可信配套代码

enclave.signed.so

Enclave相关代码，在SGX保护中进行密钥的产生和使用，并将私钥通过特定密钥加密导出

# 密钥产生

蚂蚁金服 ANT FINANCIAL

```
import (
    "crypto"
    "crypto/aes"
    "crypto/cipher"
    "crypto/elliptic"
    "crypto/sha512"
    "encoding/asn1"
    "errors"
    "io"
    "math/big"

    "sgx/LibSgxGo"
)
```

```
// GenerateKey generates a public and private key pair.
func GenerateKey(c elliptic.Curve, rand io.Reader) (*PrivateKey, error) {
    k, err := randFieldElement(c, rand)
    if err != nil {
        return nil, err
    }

    priv := new(PrivateKey)
    priv.PublicKey.Curve = c
    priv.D = k
    priv.PublicKey.X, priv.PublicKey.Y = c.ScalarBaseMult(k.Bytes())

        ////// SGX //////
        if c.Params().Name == "P-256" {

                var enclaveId uint64 = 0
                LibSgxGo.InitializeEnclave(&enclaveId)
                var sealedDataSize uint64
                privDenc := make([]byte, 32, 32)
                pubKeyX := make([]byte, 32, 32)
                pubKeyY := make([]byte, 32, 32)
                pubKeyX, pubKeyY, privDenc, _ = LibSgxGo.EcallGenerateKeypairEncprivkey(enclaveId, sealedDataSize)
                LibSgxGo.SgxDestroyEnclave(enclaveId)

                //Replace the key pairs and test print
                priv.PublicKey.X.SetBytes(pubKeyX)
                priv.PublicKey.Y.SetBytes(pubKeyY)
                priv.D.SetBytes(privDenc)
        }
        ////// SGX //////

    return priv, nil
}
```

# 私钥签名

蚂蚁金服 ANT FINANCIAL

```
import (
    "crypto"
    "crypto/aes"
    "crypto/cipher"
    "crypto/elliptic"
    "crypto/sha512"
    "encoding/asn1"
    "errors"
    "io"
    "math/big"

    "sgx/LibSgxGo"
)
```

```
// Sign signs a hash (which should be the result of hashing a larger message)
// using the private key, priv. If the hash is longer than the bit-length of the
// private key's curve order, the hash will be truncated to that length.  It
// returns the signature as a pair of integers. The security of the private key
// depends on the entropy of rand.
func Sign(rand io.Reader, priv *PrivateKey, hash []byte) (r, s *big.Int, err error) {
    // Get min(log2(q) / 2, 256) bits of entropy from rand.
    entropylen := (priv.Curve.Params().BitSize + 7) / 16
    if entropylen > 32 {
        entropylen = 32
    }
```

```
        ////// SGX //////
        //fmt.Println("ecdsa.Sign() c.Params().Name", c.Params().Name)
        if priv.Curve.Params().Name == "P-256" {
        var enclaveId uint64 = 0
                LibSgxGo.InitializeEnclave(&enclaveId)
                x, y := LibSgxGo.EcallSignECDSA(enclaveId, &hash, len(hash), priv.D)
                r.SetBytes(x)
                s.SetBytes(y)
                LibSgxGo.SgxDestroyEnclave(enclaveId)
                return
        }
        ////// SGX //////

    return
}
```

# 根据ECC私钥推导公钥

蚂蚁金服 ANT FINANCIAL

```
import (
    "crypto/ecdsa"
    "crypto/elliptic"
    "encoding/asn1"
    "errors"
    "fmt"
    "math/big"

    "sgx/LibSgxGo"
)
```

```
// parseECPrivateKey parses an ASN.1 Elliptic Curve Private Key Structure.
// The OID for the named curve may be provided from another source (such as
// the PKCS8 container) - if it is provided then use this instead of the OID
// that may exist in the EC private key structure.
func parseECPrivateKey(namedCurveOID *asn1.ObjectIdentifier, der []byte) (key *ecdsa.PrivateKey, err error) {
    var privKey ecPrivateKey
    if _, err := asn1.Unmarshal(der, &privKey); err != nil {
        return nil, errors.New("x509: failed to parse EC private key: " + err.Error())
    }
    if privKey.Version != ecPrivKeyVersion {
        return nil, fmt.Errorf("x509: unknown EC private key version %d", privKey.Version)
    }
```

```
    //priv.X, priv.Y = curve.ScalarBaseMult(privateKey)
    ////// SGX //////
    if curve.Params().Name == "P-256" {
            var enclaveId uint64 = 0
            LibSgxGo.InitializeEnclave(&enclaveId)
            priv.X, priv.Y = LibSgxGo.EcallGetEcPublicKeyFromPrivateKey(enclaveId, privateKey)
            LibSgxGo.SgxDestroyEnclave(enclaveId)
        } else {
            priv.X, priv.Y = curve.ScalarBaseMult(privateKey)
        }
        ////// SGX //////
    return priv, nil
}
```

# EDL文件

蚂蚁金服 ANT FINANCIAL

```
public void ecall_generate_keypair_encprivkey(
    [out] sgx_ec256_public_t* outPublicKey,
    [out, size=in_sealed_private_key_size] uint8_t* out_sealed_private_key,
    size_t in_sealed_private_key_size,
    [out] sgx_ec256_private_t* outEncryptedPrivateKey
);
```

```
//  sign the input digest using ecdsa256, the private key should be decrypted within the enclave
public void ecall_sign_ecdsa(
    [in, size = digest_size] const uint8_t *p_digest,
    uint32_t digest_size,
    [in] sgx_ec256_private_t *in_enc_private,
    [out] sgx_ec256_signature_t *p_signature
);
```

```
// calculate ecc public key X and Y using encrypted private key
// ONLY work for ecc256 key pair
public void ecall_get_public_key_from_private_key(
    [in, size = 32] uint8_t *privateKey,
    [out, size = 32] uint8_t *publicKeyX,
    [out, size = 32] uint8_t *publicKeyY
);
```

# Enclave核心函数 – 密钥产生

蚂蚁金服 ANT FINANCIAL

```
unsigned char entropy_buf[ADD_ENTROPY_SIZE] = { 0 };
EC_KEY * ec = NULL;
int eccgroup;

RAND_add(entropy_buf, sizeof(entropy_buf), ADD_ENTROPY_SIZE);
RAND_seed(entropy_buf, sizeof(entropy_buf));

eccgroup = OBJ_txt2nid("prime256v1"); //P-256, also known as secp256r1 and prime256v1
ec = EC_KEY_new_by_curve_name(eccgroup);
if (ec == NULL) {
    abort();
}

EC_KEY_set_asn1_flag(ec, OPENSSL_EC_NAMED_CURVE);

int ret = EC_KEY_generate_key(ec);
if (!ret) {
    abort();
}
```

```
// encrypt private key
uint8_t bufOutEncryptedPrivateKey[32] = {0};
ret = sgx_aes_ctr_encrypt(&aes_key, bufPlaintextPrivateKey, 32, aes_iv, 1, bufOutEncryptedPrivateKey);

// output encrypted private key
memcpy(outEncryptedPrivateKey, bufOutEncryptedPrivateKey, sizeof(*outEncryptedPrivateKey));
```

# Enclave核心函数 – 私钥签名

蚂蚁金服 ANT FINANCIAL

```
// AES decryption of in_enc_private key
sgx_aes_ctr_decrypt(&aes_key, (const uint8_t *)in_enc_private, 32, aes_iv, 1, (uint8_t *)(&decrypted_private));
```

```
eccgroup = OBJ_txt2nid("prime256v1"); //P-256, also known as secp256r1 and prime256v1
ec = EC_KEY_new_by_curve_name(eccgroup);
if (ec == NULL) {
    abort();
}

// Use input private key from Fabric "in_enc_private" to fill the new key generated above.

BN_bin2bn((const unsigned char *)(&decrypted_private), 32, inPrivKeyBignum);
int ret = EC_KEY_set_private_key(ec, (const BIGNUM*) inPrivKeyBignum);
if (!ret) {
    abort();
}

// sign the digest, the signature is DER encoded
if (ECDSA_sign(0, in_p_digest, 32, signature_der, &sigLen, ec) == 0) {
    abort();
}
```

# Enclave核心函数 –根据ECC私钥推导公钥

蚂蚁金服 ANT FINANCIAL

```
// if the private key is encrypted, then AES decryption of encrypted private key
sgx_aes_ctr_decrypt(&aes_key, (const uint8_t *)privateKey, 32, aes_iv, 1, privateKeyPlainText);
```

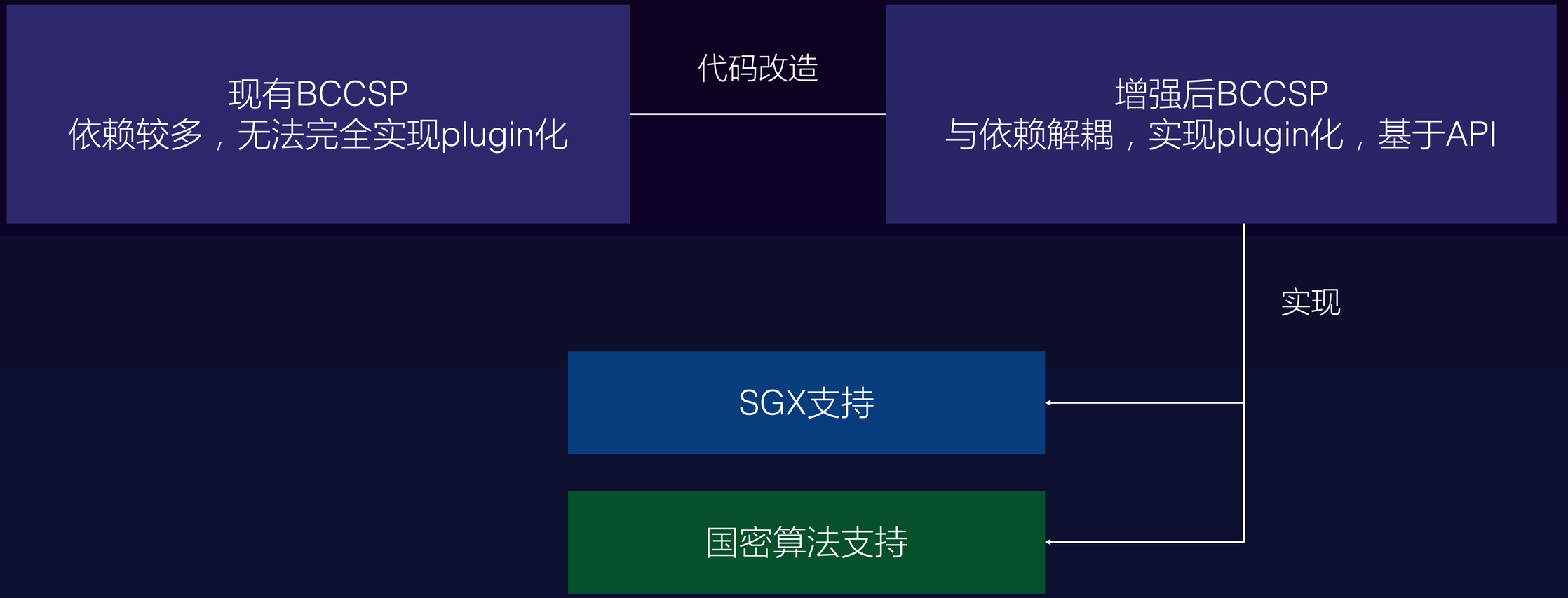
```
eccgroup = OBJ_txt2nid("prime256v1"); //P-256, also known as secp256r1 and prime256v1
ec = EC_KEY_new_by_curve_name(eccgroup);
if (ec == NULL) {
    abort();
}
EC_KEY_set_asn1_flag(ec, OPENSSL_EC_NAMED_CURVE);
const EC_GROUP *ecGroup = EC_KEY_get0_group((const EC_KEY *)ec);
EC_POINT *publickKeyEcPoint = EC_POINT_new(ecGroup);

EC_POINT_mul(ecGroup, publickKeyEcPoint, privateKeyBN, NULL, NULL, NULL);
uint8_t bufPublickKey[65] = {0x04};
EC_POINT_point2oct(ecGroup, (const EC_POINT *)publickKeyEcPoint, POINT_CONVERSION_UNCOMPRESSED, bufPublickKey, 65, NULL);
memcpy(publicKeyX, bufPublickKey + 1, 32);
memcpy(publicKeyY, bufPublickKey + 1 + 32, 32);
```

蚂蚁金服
ANT FINANCIAL

# 方案2：实现一个基于SGX的plugin BCCSP

# 基本架构


现有BCCSP
依赖较多，无法完全实现plugin化
代码改造
增强后BCCSP
与依赖解耦，实现plugin化，基于API
实现
SGX支持
国密算法支持

# BCCSP 接口

蚂蚁金服 ANT FINANCIAL

```go
// Key represents a cryptographic key
type Key interface {

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

    // SKI returns the subject key identifier of this key.
    SKI() []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 a private key,
    // false otherwise.
    Private() bool

    // PublicKey returns the corresponding public key part of an asymmet
    // This method returns an error in symmetric key schemes.
    PublicKey() (Key, error)
}

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

    // Algorithm returns the key generation algorithm identifier (to be
    Algorithm() string

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

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

```go
// BCCSP is the blockchain cryptographic service provider that offers
// the implementation of cryptographic standards and algorithms.
type BCCSP interface {

    // KeyGen generates a key using opts.
    KeyGen(opts KeyGenOpts) (k Key, err error)

    // KeyDeriv derives a key from k using opts.
    // The opts argument should be appropriate for the primitive used.
    KeyDeriv(k Key, opts KeyDerivOpts) (dk Key, err error)

    // KeyImport imports a key from its raw representation using opts.
    // The opts argument should be appropriate for the primitive used.
    KeyImport(raw interface{}, opts KeyImportOpts) (k Key, err error)

    // GetKey returns the key this CSP associates to
    // the Subject Key Identifier ski.
    GetKey(ski []byte) (k Key, err error)

    // Hash hashes messages msg using options opts.
    // If opts is nil, the default hash function will be used.
    Hash(msg []byte, opts HashOpts) (hash []byte, err error)

    // GetHash returns and instance of hash.Hash using options opts.
    // If opts is nil, the default hash function will be returned.
    GetHash(opts HashOpts) (h hash.Hash, err error)

    // Sign signs digest using key k.
    // The opts argument should be appropriate for the algorithm 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
    // The opts argument should be appropriate for the algorithm used.
```

# 修改Plugin BCCSP接口

蚂蚁金服 ANT FINANCIAL

```
166 //In order to decouple plugins and host programs, go plugins should not depend on the host program's
     symbol.
167 //See https://github.com/golang/go/issues/20481  https://github.com/akutz/gpds
168 type PluginKey interface {
169
170 »       // Bytes converts this key to its byte representation,
171 »       // if this operation is allowed.
172 »       Bytes() ([]byte, error)
173
174 »       // SKI returns the subject key identifier of this key.
175 »       SKI() []byte
176
177 »       // Symmetric returns true if this key is a symmetric key,
178 »       // false is this key is asymmetric
179 »       Symmetric() bool
180
181 »       // Private returns true if this key is a private key,
182 »       // false otherwise.
183 »       Private() bool
184
185 »       // PublicKey returns the corresponding public key part of an asymmetric public/private key p
    air.
186 »       // This method returns an error in symmetric key schemes.
187 »       PublicKey() (interface{}, error)
188 }
```

```
190 type PluginBCCSP interface {
191 »       // KeyGen generates a key using opts.
192 »       KeyGen(opts interface{}) (k interface{}, err error)
193
194 »       // KeyDeriv derives a key from k using opts.
195 »       // The opts argument should be appropriate for the primitive used.
196 »       KeyDeriv(k interface{}, opts interface{}) (dk interface{}, err error)
197
198 »       // KeyImport imports a key from its raw representation using opts.
199 »       // The opts argument should be appropriate for the primitive used.
200 »       KeyImport(raw interface{}, opts interface{}) (k interface{}, err error)
201
202 »       // GetKey returns the key this CSP associates to
203 »       // the Subject Key Identifier ski.
204 »       GetKey(ski []byte) (k interface{}, err error)
205
206 »       // Hash hashes messages msg using options opts.
207 »       // If opts is nil, the default hash function will be used.
208 »       Hash(msg []byte, opts interface{}) (hash []byte, err error)
209
210 »       // GetHash returns and instance of hash.Hash using options opts.
211 »       // If opts is nil, the default hash function will be returned.
212 »       GetHash(opts interface{}) (h hash.Hash, err error)
213
214 »       // Sign signs digest using key k.
215 »       // The opts argument should be appropriate for the algorithm used.
216 »       //
217 »       // Note that when a signature of a hash of a larger message is needed,
218 »       // the caller is responsible for hashing the larger message and passing
219 »       // the hash (as digest).
220 »       Sign(k interface{}, digest []byte, opts interface{}) (signature []byte, err error)
221
222 »       // Verify verifies signature against key k and digest
223 »       // The opts argument should be appropriate for the algorithm used.
224 »       Verify(k interface{}, signature, digest []byte, opts interface{}) (valid bool, err error)
```

# 为何修改Plugin BCCSP接口

工程上需要解耦Plugin和Fabric的构建、发布、版本管理等过程。

golang plugin模块的约束。
https://github.com/golang/go/issues/20481

提交给社区的PR
https://gerrit.hyperledger.org/r/c/fabric/+/31024

# 实现一个基于SGX的plugin BCCSP

EDL部分，类似方案1：

密钥产生、密钥Seal后导出、通过私钥获取公钥、私钥签名

涉及BCCSP的以下接口：

BCCSP.KeyGen
BCCSP.KeyImport
BCCSP.Sign
Key.PublicKey

蚂蚁金服
ANT FINANCIAL

# 为何选择方案2？

# 工程需要

蚂蚁金服 ANT FINANCIAL

我们希望SGX增强方案是一种“可选配置”，整体上是一套代码、一套镜像。

在代码版本管理、编译、测试、容器镜像管理、发布、线上运维等各个方面代价最小。

# 可扩展性


最终，我们希望BaaS平台提供一种能力，用户可以提供自定义的Enclave实现。即：用户可以自己打造“硬件钱包”，完全拥有核心密钥的控制权；同时又能把计算和存储放在云上，一样不差的使用BaaS的其他功能。

蚂蚁金服
ANT FINANCIAL

# Q&A