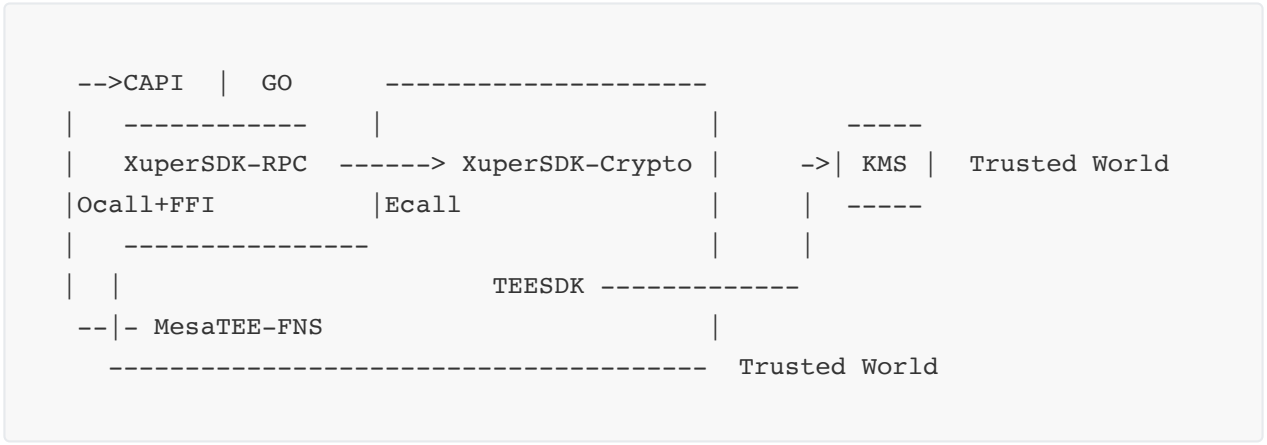
# 拆分设计

从《超级链SDK结构》可以了解当前SDK的架构。

本设计主要解决在TEE内部发起可信交易的问题，打造超级链的**可信数据上链ORACLE**。

SDK未来在3CO项目中会被TEE外面和TEE内部使用到，因此需要保证同一套代码，但是支持签名部分在可以发生在TEE内部， 所以整体SDK分为3个部分 TEESDK, XuperSDK-Crypto 以及XuperSDK-RPC

因此整体的拆分思路是：




功能解释：

- XuperSDK-RPC: 组装交易，发送交易，作为一个c-lib存在，被rust通过FFI调用
- XuperSDK-Crypto： 通过Hash生成TXID，交易签名。作为一个libenclave.so存在，被RPC模块加载。
- TEESDK: 负责跟KMS通信，加密交易的敏感字段；

在解释之前，先增加几个定义：

```
数据标识（Data ID）：  全局以为的代表当前数据的ID，可以是UUID或者编号。
数据存证信息(Data Credential)：(Data ID，加密meta，描述信息，发布时间，发布方信息，签名)
操作存证信息(Ops Credential)：(TaskID，操作人，操作日志hash，操作日志密文，操作时间，前一个操作ID，签名)
```

核心操作流程如下（包括输入数据，输出数据以及计算过程日志存证）

MesaTEE-FNS调用对应的function计算完成之后，获得计算的结果文件，计算结果的存证信息(DC).

具体流程如下

1. FNS 先调用TEESDK，加密敏感数据；
2. [Ocall] InitConfig初始化SDK实例；
3. 开始组装交易

> a. [Ocall] GetAccountFromFile: 从本地文件恢复密码（这里也可以改为从内存中加载）
>
> b. [Ocall] PreExecWithSelecUTXO:
>
> c. [Ocall] GenCompleteTxAndPost
>
> > i. GenComplianceCheckTx
> >
> > > 1. MakeTransactionID
> > > 2. SignECDSA ---> **Ecall**
> >
> > ii. GenRealTx
> >
> > > 1. MakeTransactionID
> > > 2. SignECDSA ---> **Ecall**
> >
> > iii. ComplianceCheck
> >
> > v. MakeTransactionID
> >
> > vi. PostTx

# 详细接口

## XuperSDK-RPC

```
typedef signed char GoInt8;
typedef unsigned char GoUint8;
typedef short GoInt16;
typedef unsigned short GoUint16;
typedef int GoInt32;
typedef unsigned int GoUint32;
typedef long long GoInt64;
typedef unsigned long long GoUint64;
typedef GoInt64 GoInt;
typedef GoUint64 GoUint;
typedef __SIZE_TYPE__ GoUintptr;
typedef float GoFloat32;
typedef double GoFloat64;
typedef float _Complex GoComplex64;
typedef double _Complex GoComplex128;

/*
```

```
  static assertion to make sure the file is being used on architecture
  at least with matching size of GoInt.
*/
typedef char _check_for_64_bit_pointer_matching_GoInt[sizeof(void*)==64/8 ?
1:-1];

#ifndef GO_CGO_GOSTRING_TYPEDEF
typedef _GoString_ GoString;
#endif
typedef void *GoMap;
typedef void *GoChan;
typedef struct { void *t; void *v; } GoInterface;
typedef struct { void *data; GoInt len; GoInt cap; } GoSlice;

#endif

/* End of boilerplate cgo prologue.  */

#ifdef __cplusplus
extern "C" {
#endif


/* Return type for CreateAccount */
struct CreateAccount_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

// RetrieveAccount 通过助记词恢复账户

extern struct RetrieveAccount_return RetrieveAccount(GoString p0, GoInt p1);

/* Return type for GetBinaryEcdsaPrivateKeyFromFile */
struct GetBinaryEcdsaPrivateKeyFromFile_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

// QueryTxidFromTx 查询Txid

extern struct QueryTxidFromTx_return QueryTxidFromTx(GoString p0);

/* Return type for QueryTx */
struct QueryTx_return {
  char* r0;
  GoInt32 r1;
```

```
  GoInt32 r2;
};

// QueryTx 查询交易状态

extern struct QueryTx_return QueryTx(GoString p0, GoString p1, GoString p2);

/* Return type for GetBalanceDetailForSpecificChain */
struct GetBalanceDetailForSpecificChain_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

// GetBalanceDetailForSpecificChain 获取一条链上的被冻结的账户金额

extern struct GetBalanceDetailForSpecificChain_return
GetBalanceDetailForSpecificChain(GoString p0, GoString p1, GoString p2);

/* Return type for GetBalanceForSpecificChain */
struct GetBalanceForSpecificChain_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

// GetBalanceForSpecificChain 获取一条链上的账户余额

extern struct GetBalanceForSpecificChain_return
GetBalanceForSpecificChain(GoString p0, GoString p1, GoString p2);

/* Return type for GetBalance */
struct GetBalance_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

// @todo
// GetBalance 获取多个链上的账户余额
// bcnames bcname逗号分隔

extern struct GetBalance_return GetBalance(GoString p0, GoString p1, GoString
p2);

/* Return type for InitConfig */
struct InitConfig_return {
  char* r0;
  GoInt32 r1;
```

```
  GoInt32 r2;
};

//初始化配置
extern struct InitConfig_return InitConfig(GoString p0, GoString p1, GoString
p2, GoString p3, GoString p4, GoString p5);

/* Return type for GetConfig */
struct GetConfig_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

extern struct GetConfig_return GetConfig();

/* Return type for PreExecCreateAccountUseAddress */
struct PreExecCreateAccountUseAddress_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

// 调用Wasm合约

extern struct InvokeWasmContract_return InvokeWasmContract(GoString p0,
GoString p1, GoString p2, GoString p3, GoString p4, GoString p5, GoString p6,
GoString p7);

/* Return type for PreExecuteInvokeWasmContract */
struct PreExecuteInvokeWasmContract_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

// 预执行Wasm合约

extern struct PreExecuteInvokeWasmContract_return
PreExecuteInvokeWasmContract(GoString p0, GoString p1, GoString p2, GoString
p3, GoString p4, GoString p5, GoString p6, GoString p7);

/* Return type for TransferWasmTxByPreExeResult */
struct TransferWasmTxByPreExeResult_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};
```

```
// 通过使用预执行返回的结果，执行Invoke智能合约

extern struct TransferWasmTxByPreExeResult_return
TransferWasmTxByPreExeResult(GoString p0, GoString p1);

/* Return type for QueryWasmContract */
struct QueryWasmContract_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

// 查询Wasm合约

extern struct QueryWasmContract_return QueryWasmContract(GoString p0, GoString
p1, GoString p2, GoString p3, GoString p4, GoString p5);

/* Return type for CreateChain */
struct CreateChain_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};


//// Transfer 转账

extern struct TransferV2_return TransferV2(GoString p0, GoString p1, GoString
p2, GoString p3, GoString p4, GoString p5, GoString p6, GoString p7);

/* Return type for Transfer */
struct Transfer_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

// Transfer 转账

extern struct Transfer_return Transfer(GoString p0, GoString p1, GoString p2,
GoString p3, GoString p4, GoString p5, GoString p6, GoString p7);

/* Return type for TransferV3 */
struct TransferV3_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};
```

```
// @todo
//// Transfer 转账

extern struct TransferV3_return TransferV3(GoString p0, GoString p1, GoString
p2, GoString p3, GoString p4, GoString p5, GoString p6, GoString p7);

/* Return type for TransferByPlatform */
struct TransferByPlatform_return {
  char* r0;
  GoInt32 r1;
  GoInt32 r2;
};

//
//// Transfer 转账

extern struct TransferByPlatform_return TransferByPlatform(GoString p0,
GoString p1, GoString p2, GoString p3, GoString p4, GoString p5, GoString p6,
GoString p7);
```

## XuperSDK-Crypto

crypto的对RPC提供的接口只有计算签名，输出是私钥和消息，返回签名字符串。 这个接口通过动态链接库提供给cgo使用。

```
fn sign(sk: PrivateKey, msg: &[u8]) -> Result<Vec<u8>>
```

在XuperSDK里面，目前超级链的crypto是通过接口的形式暴露的具体的加密库方法。因此，这里考虑根据编译标签来加载对应的密码学库。

```
                xchain/crypto   <----         ---->   xchain-rust-crypto
                                     |       |
                                base.CryptoClient
```

其中base.CryptoClient定义在github.com/xuperchain/xuperchain/core/crypto/client/base.

# 开发安排

1. XuperSDK-Crypto进入TEE， 5.12-5.13
2. XuperSDK-RPC CAPI封装以及切换签名到XuperSDK-Crypto， 5.14 - 5.18
3. TEESDK进TEE； @zhiyu
4. demo开发； 5.22号
5. 跟mesatee联调 @jiwen @zhiyu