Supporting Multiple Privacy Patterns in Fidelius

范学鹏

设计目标

背景

客户对于隐私计算的需求、场景已经远超最初的“数据不出域”这一简单场景，为了满足这些客户的需求，我们需要对各种数据合作方式进行支持，这给 Fidelius 提出了新的挑战，我们希望能尽可能的在 Fidelius 中支持这些场景，减少系统开发及维护成本。

设计目标

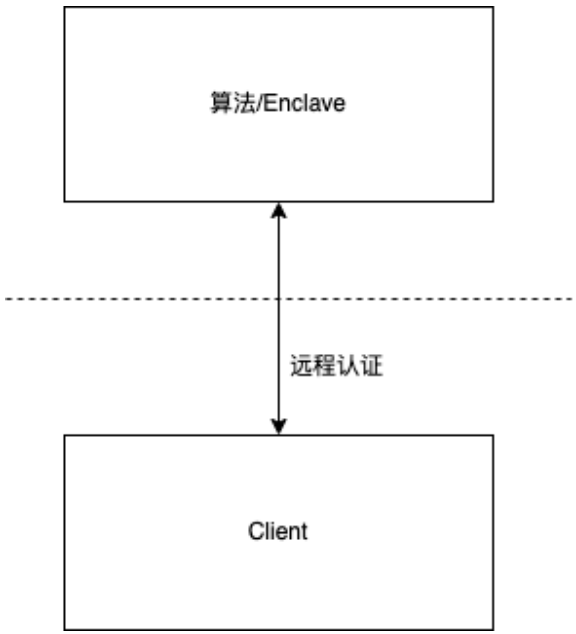
在不同的场景需求之外，我们希望满足：

1. 能够支持不同的 TEE 环境，而不仅仅是 Intel SGX；
2. 对新的场景扩展友好；
3. 尽可能复用已有的模块。

我们不讨论...

Fidelius 是隐私计算中间件，Fidelius 之上的算法、周围的工具、与智能合约的交互、以及智能合约的算法不在本文的讨论范围。

基于 TEE 的隐私计算模型



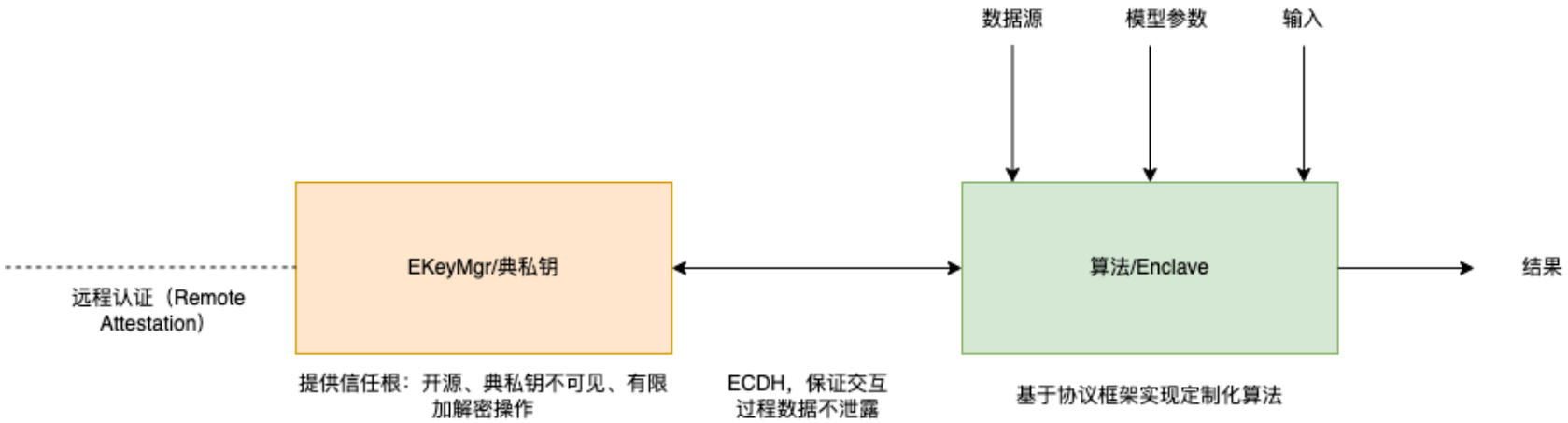
- Client 将算法/Enclave 发送给 Server（云计算环境下，Server 是可以作恶的）；
- Client 对算法/Enclave 进行远程认证，保证算法/Enclave 被正确启动；
- 然后给定输入数据、运行算法、得到结果。

整个计算模型没有定义数据源、模型参数等问题，仅能保证计算过程及输入参数是符合预期的，且计算过程、输入参数、计算结果不会泄露给宿主机。

Fidelius 隐私计算模型

Fidelius 的计算模型“基于但有别于”传统的基于 TEE 的隐私计算模型，也不同于联邦学习或多方安全计算。

1. 在输入参数之外，还需要考虑数据源、模型参数的问题；
2. 需要考虑多种计算方式，即多种隐私计算模式。
3. 考虑到多硬件平台的支持，不应该过多的依赖远程认证。

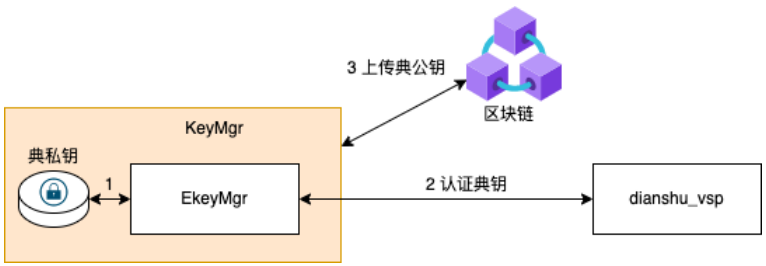


Fidelius 隐私计算模型仅在初始化阶段依赖远程认证，通过远程认证建立信任根后，在计算过程中，不再依赖于远程认证。这一方面简化了部署要求，另一方面降低了交互成本。

典钥

典钥的特点

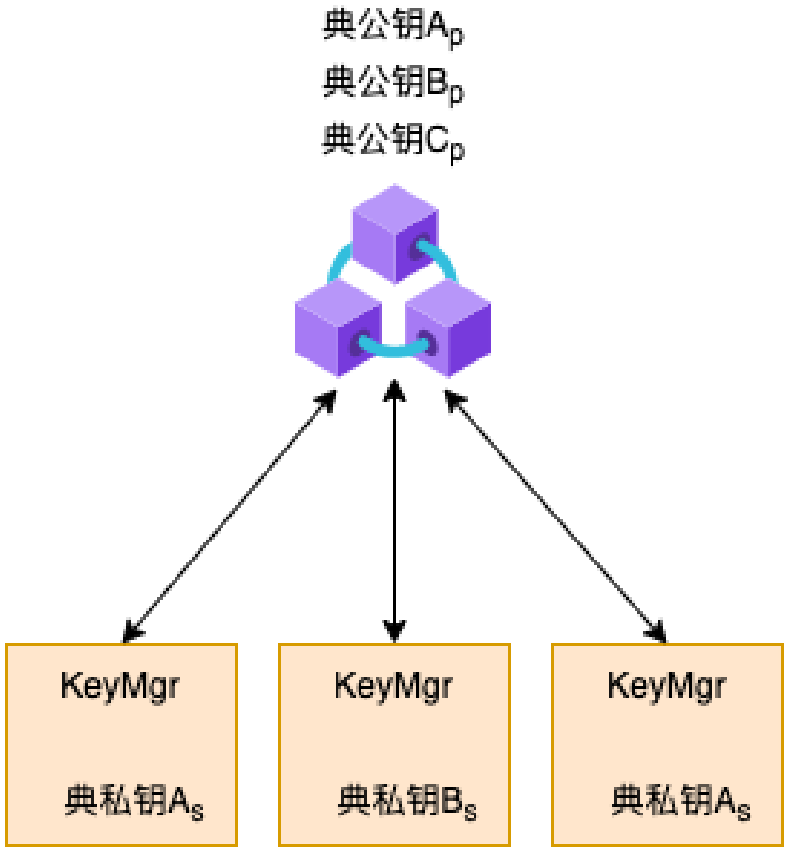
- 仅在当地节点、当前用户的当前 EKeyMgr 中可见；
- 仅能用做非常有限（限定前缀）的解密，不能用做任意目的的解密。



1. 在 Enclave 内生成典钥，并将典私钥密封（seal）到本地；
2. 将典公钥及密码报告（report）发送给典枢的验证服务进行验证，验证成功后，对典公钥进行签名；
3. 上传典公钥及签名到区块链，完成典公钥的发布。

典钥

典钥一般表示，如下图所示



符号表示

我们约定公钥密码体系使用的符号如下：

- 加密：*encrypt*$(P, m, \textit{label})$，表示使用公钥 P，对消息 m 进行加密，返回加密数据 c；*label* 做为认证码，取值范围为 $\{\texttt{crypto\_prefix\_arbitrary}, \texttt{crypto\_prefix\_forward}\}$。
- 解密：*decrypt*$(S, c, \textit{label})$，表示使用私钥 S，对消息 c 进行解密，返回解密数据 m；注意，*label* 的值必须与加密时提供的值一致，否则返回错误。
- 签名：*sign*(S, m)，表示使用私钥 S 对消息 m 进行签名，返回签名 *sig*。
- 验签：*verify*$(P, m, \textit{sig})$，表示使用公钥 P 对消息 m 验证签名是否为 *sig*，返回 `true` 或者 `false`。

对于可信执行环境（节点）i，其生成的典钥对记为 $\{S_d^i, P_d^i\}$，其中 S_d^i 为私钥，P_d^i 为公钥。可信执行环境对外提供了受限的加密、解密功能，即其中 *label* 的取值只能为 `crypto_prefix_arbitrary`。

枢钥

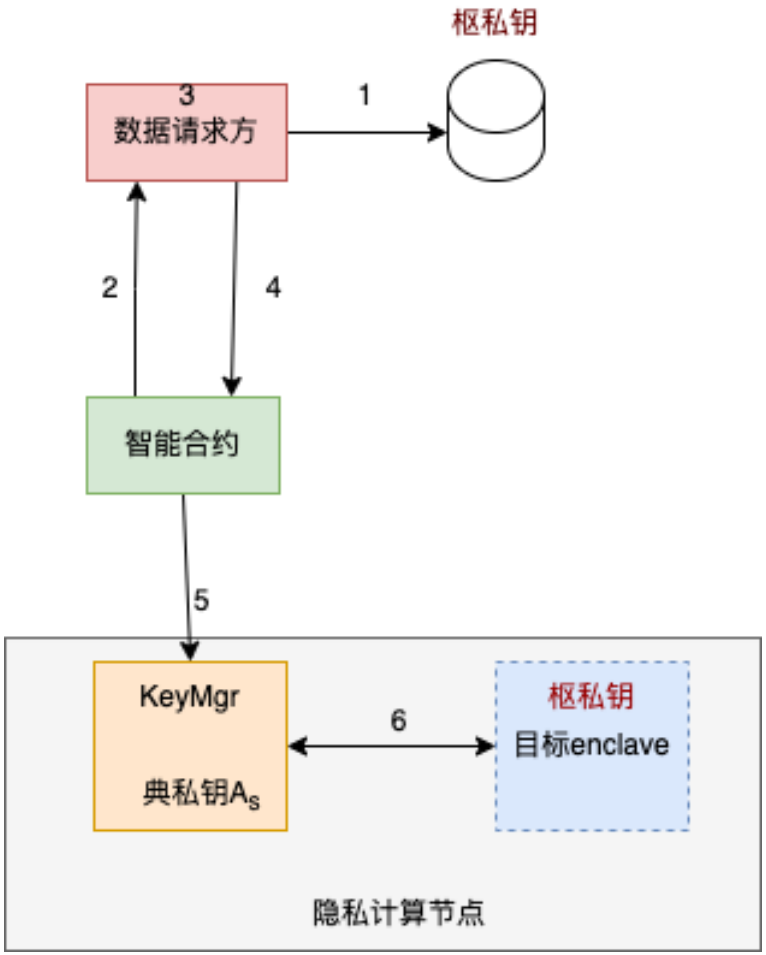
枢钥的特点

- 在本地环境生成，可以是硬件设备、浏览器等环境；
- 在 Fidelius 中，提供了 `yterminus` 命令行工具，也提供了 JavaScript，Python 的封装；
- 枢钥的使用应该是一次性的，不应该重复使用。

我们把枢钥对记为 $\{S_s^i, P_s^i\}$，其中 S_s^i 为私钥，P_s^i 为公钥。枢钥的加密、解密功能不受限。

枢钥转发

目标：隐私计算节点的目标 enclave 内获取枢私钥，并且保证该枢私钥不会泄露。



1. 在本地生成枢钥，注意，枢钥不一定在 TEE 内生成，可以使用 Javascript、Python 等生成；
2. 从智能合约（区块链）中获取典公钥；
3. 使用典公钥加密枢私钥；
4. 将加密之后的枢私钥及签名提交到区块链上；
5. 隐私计算节点从区块链上获取加密的枢私钥及签名；
6. 将枢私钥转发给目标 enclave。

注意：3 中加密时的 *label* 为 `crypto_prefix_forward`，由前文可知，典钥不能解密并暴露枢私钥给不可信环境。

5. EkeyMgr::forward_private_key

首先，在不可信环境下调用 forward_private_key。

```
std::unordered_map<bytes, bytes> private_key_table; //全局变量

uint32_t forward_private_key(const bytes & encrypted_shu_skey,
    const bytes & dian_public_key, const bytes & target_enclave_hash,
    const bytes &sig){

    bytes dian_skey = load_key_pair(dian_public_key); //从磁盘读取并unseal私钥
    shu_skey = decrypt(dian_skey, encrypted_shu_skey, crypto_prefix_forward);
    shu_pkey = gen_pkey_from_skey(shu_skey);
    bytes data = dian_public_key + target_enclave_hash;
    bool succ = verify(shu_pkey, data, sig);
    if(succ){
      private_key_table[shu_pkey + target_enclave_hash] = dian_public_key + shu_skey;
      return success;
    }
    return fail;
}
```

代码来源keymgr/default/enclave/ekeymgr.cpp

6.1 目标
enclave::request_private_key_for_public_key

目标 enclave 主动请求枢公钥对应的枢私钥。

```
uint32_t request_private_key_for_public_key(const bytes & public_key,
        bytes & private_key, bytes & dian_pkey){
    bytes request_msg = make_request_from(public_key);
    bytes response_msg = m_keymgr_session->send_request_and_recv(request_msg);
    dian_pkey = bytes(request_msg.data(), PUBLIC_KEY_SIZE);
    private_key = bytes(request_msg.data() + PUBLIC_KEY_SIZE, PRIVATE_KEY_SIZE);
    bytes check_pkey = gen_pkey_from_skey(private_key);
    if(check_pkey == public_key){
        return success;
    }
    return fail;
}
```

代码来源core/ypc_t/analyzer/interface/keymgr_interface.h

注意，其中 m_keymgr_session->send_request_and_recv(request_msg) 请求会间接调用后续的 handle_pkg 方法，获得请求的内容。

6.2 EKeyMgr::handle_pkg

EKeyMgr 接收请求，并查找枢公钥对应的枢私钥。

```
bytes handle_pkg(const bytes & data,
        dh_session * session){
    //注意，此处source_enclave_hash不能伪造
    bytes source_enclave_hash = session->get_source_enclave_hash();
    bytes request_public_key = data;
    bytes f1 = request_public_key + source_enclave_hash;
    if(private_key_table.find(f1) != private_key_table.end()){
        return private_key_table[f1];
    }
    bytes any_enclave_hash = sha3("any enclave");
    bytes f2 = request_public_key + any_enclave_hash;
    if(private_key_table.find(f2) != private_key_table.end()){
        return private_key_table[f2];
    }
    return "";
}
```

代码来源keymgr/default/enclave/ekeymgr.cpp

查找结果会经过加密传输给目标 enclave，从而使得目标 enclave 获得对应的枢私钥。

许可（Allowance）

许可是指在给定输入参数下，允许给定算法使用给定模型参数或给定数据源进行计算的密码学证明，按照用途分为模型参数许可以及数据源许可。
许可做为输入的一部分，由模型提供方或数据源提供方提供给算法。算法在开始执行实际的计算任务前，会对许可进行验证。当许可验证通过后，才会执行实际的计算任务。
一个许可的结构如下：

$$\{Sig, DHash, PKey\}$$

其中，*Sig* 是一个签名，*DHash* 是模型参数或者数据源的 Hash，*PKey* 是许可提供方的公钥，一般是枢公钥。

许可生成

注意，此处仅提供了输入参数的 hash，并未提供输入参数本身，因此许可生成方并不能得知输入参数的具体值。

```
bytes generate_allowance(const bytes & shu_skey,
        const bytes & param_hash, const bytes & enclave_hash,
        const bytes & dian_pkey, const bytes &dhash){
    bytes data = param_hash + enclave_hash + dian_pkey + dhash;
    bytes sig = sign(shu_skey, data);
    return sig;
}
```

代码来源toolkit/terminus/cmd/cterminus/allowance.cpp

许可验证

注意，此处的 param_hash 及数据源的 dhash 会进行进一步的校验，因此不能伪造。

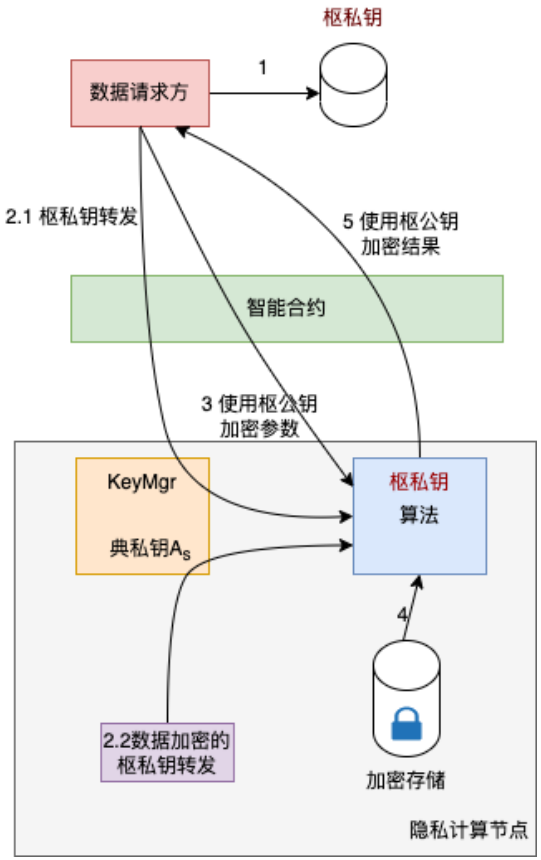
```
bool check_allowance(const bytes & param_hash, const bytes & sig,
        const bytes & shu_pkey, const bytes & dhash){
    bytes dian_pkey, shu_skey;
    request_private_key_for_public_key(shu_pkey, shu_skey, dian_pkey);
    //注意：此处enclave_hash不能伪造
    bytes enclave_hash = get_current_enclave_hash();
    bytes data = param_hash + enclave_hash + dian_pkey + dhash;

    bool succ = verify(shu_pkey, data, sig);
    return succ;
}
```

代码来源core/include/ypc_t/analyzer/interface/allowance_interface.h

经典例子：数据不出域

该模式是最为经典的模式，常见于对数据隐私或数据主权要求很高的场合。

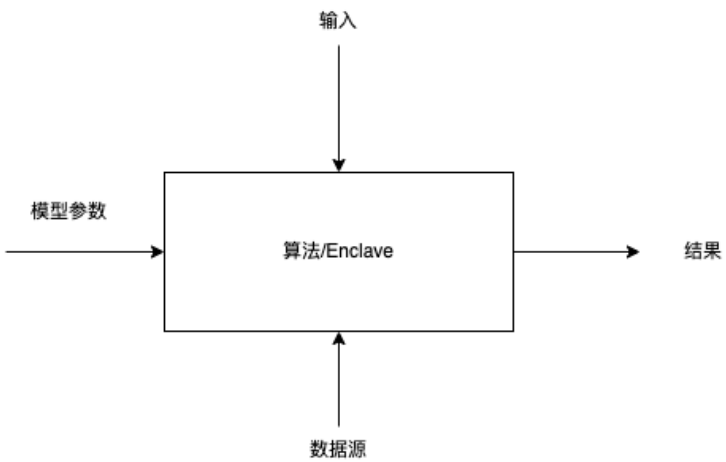


1. 数据请求方生成枢私钥；
2. 枢私钥转发（见前文）；
3. 使用枢公钥加密算法参数，通过区块链转发给算法；
4. 算法从本地读取数据，算法通过 EKeyMgr 获取枢私钥，解密算法输入参数，并进行计算；
5. 算法对结果使用枢公钥加密，并使用枢私钥进行签名，结果通过区块链转发给数据请求方。

注意

在实现中，为了减少交互次数，通常步骤 2 与步骤 3 会合并为一笔交易。

隐私计算模式的设计选择（Design Choices）



- 除了基本的密码协议（例如 secp256k1 或其他椭圆曲线）的选择外，还有诸多待选择的空间。
- 这可能仍然不完备，例如数据源可能需要支持流式数据，因此，我们几乎是在一个不能确定未来需求的情况下设计 Fidelius。

1. 模型参数：
   - 有模型参数、但不需要隐私保护；
   - 有模型参数、需要隐私保护；
   - 无模型参数；
2. 输入：
   - 输入参数需要加密；
   - 输入参数不需要加密；
3. 数据源：
   - 无数据源、单数据源、多数据源；
   - 数据源加密；
   - 数据源在/不在本地
4. 结果：
   - 结果较小、直接上链；
   - 结果较大、不能直接上链；
   - 结果在本地直接查看，不需要加密；
   - 结果加密并转发给另一个 enclave；

Fidelius 中的 algo_wrapper

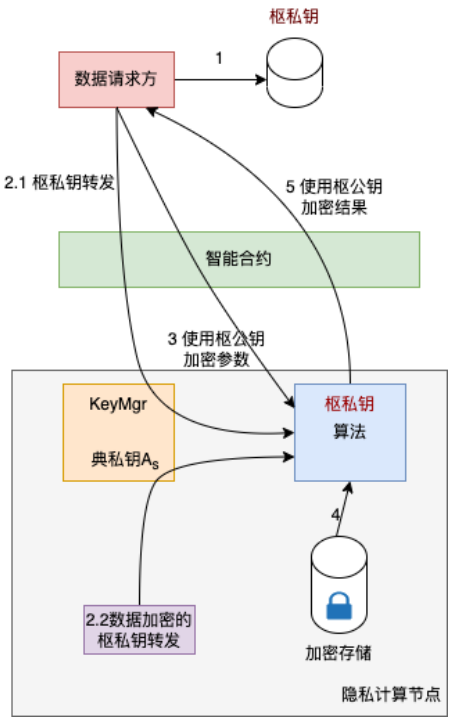
```
template<typename Crypto, typename DataSession, typename ParserT,
 typename Result, typename ModelT = void,
 template <typename> class DataAllowancePolicy = ignore_data_allowance,
 template <typename> class ModelAllowancePolicy = ignore_model_allowance>
class algo_warpper;
```

代码来源core/include/ypc_t/analyzer/algo_wrapper.h

- Crypto：密码协议簇，目前支持 ypc::crypto::eth_sgx_crypto，兼容以太坊。
- DataSession：数据源方式，支持 noinput_data_stream, raw_data_stream, sealed_data_stream, multi_data_stream。
- ParserT：表示自定义的算法类。
- Result：表示结果的类型，支持 local_result, onchain_result, offchain_result, forward_result。
- ModelT：表示模型的类型，是 ff::util::ntobject<...>。
- DataAllowancePolicy，表示数据源的许可验证策略，支持 ignore_data_allowance, check_data_allowance。
- ModelAllowancePolicy，表示模型的许可验证策略，支持 ignore_model_allowance, check_model_allowance。

通过对上述参数的选择和组合，可以支持不同的场景。

经典例子：数据不出域




```
ypc::algo_warpper<ypc::crypto::eth_sgx_crypto,
      ypc::sealed_data_item,
      enclave_iris_means_parser,
      ypc::onchain_result<ypc::crypto::eth_sgx_crypto>
> pw;

YPC_PARSER_IMPL(pw);
```

加密数据托管 1-单一数据托管

数据提供方没有算力或存储，将数据托管到三方或云服务环境中，但是要求：

1. 数据的隐私不能泄露，隐私数据的存储使用加密存储；
2. 数据不能未经许可使用，数据的每次使用都需要经过数据提供方授权。

```
ypc::algo_warpper<ypc::crypto::eth_sgx_crypto,
      ypc::sealed_data_stream,
      enclave_iris_means_parser,
      ypc::onchain_result<ypc::crypto::eth_sgx_crypto>,
      void,
      ypc::check_data_allowance
> pw;

YPC_PARSER_IMPL(pw);
```

该模式用于改进原有的数据托管（SaaS）方式，适用于对数据托管有需求，但是又需要隐私保护的场景。

加密数据托管 2-多数据融合

由于计算任务的复杂性，多方数据需要放在同一个节点上进行计算，数据在加密转移到同一个服务器后，数据的使用需要经过许可。

```
ypc::algo_warpper<ypc::crypto::eth_sgx_crypto,
      ypc::multi_data_stream,
      enclave_iris_means_parser,
      ypc::onchain_result<ypc::crypto::eth_sgx_crypto>,
      void,
      ypc::check_data_allowance
> pw;

YPC_PARSER_IMPL(pw);
```

该模式下，数据的许可需要所有的数据使用方同意。

加密数据托管 3-多数据融合

数据的运营直接托管给三方，数据的使用不再需要经过许可。

```
ypc::algo_warpper<ypc::crypto::eth_sgx_crypto,
      ypc::multi_data_stream,
      enclave_iris_means_parser,
      ypc::onchain_result<ypc::crypto::eth_sgx_crypto>
> pw;

YPC_PARSER_IMPL(pw);
```

注意：虽然此处数据的使用不需要经过数据提供方的许可，但是仍然需要经过数据运营方的许可，且该许可过程在区块链系统中存证，因此并不会滥用数据。

保护模型隐私

该模式适用于对模型参数的隐私保护有需求，希望对外提供模型的同时，不泄露相关的参数。

```
ypc::algo_warpper<ypc::crypto::eth_sgx_crypto,
      ypc::noinput_data_stream,
      enclave_iris_classifier,
      ypc::local_result,
      iris_model,
      ypc::ignore_data_allowance,
      ypc::check_model_allowance
> pw;

YPC_PARSER_IMPL(pw);
```

其他

还有更多的模式需要解决：

1. 联邦学习；
2. 流式数据处理；
3. ...

Question & Answer