以开发者角度理解 Flashbots 的 SUAVE 链：除了 MEV、EVM + TEE 还有哪些可能性？

SUAVE 是 Flashbots 开发的去中心化项目，它建立了一个拥有 TEE 环境的网络，解决在 MEV 流程中遇到的诸如密钥保管、多方互信的问题。同时，SUAVE 项目中 TEE 的加入，让 SUAVE 拥有除了解决 MEV 问题以外更多的可能性。

# SUAVE 相关代码库

SUAVE 项目是基于以太坊扩展的，所以它天生是兼容 EVM 的。它目前在 GitHub 上的相关项目有：SUAVE-geth、SUAVE-std、SUAVE-examples 等。

其中，SUAVE-geth 就是在 geth 基础上进行扩展的执行层代码，它主要是在 geth 基础上增加了加密计算环境，以及在加密计算环境下的一些 precompile（预编译）。特别值得一提的是增加了标准 HTTPS 请求的 precompile，这使得开发者可以利用TEE环境给用户提供访问其他网络的功能。此外，它包含了一系列基于 TEE 使用功能的precompile，例如获取加密参数、存储加密信息以及获取加密信息等，这些组成了基于可信环境下的开发基础设施。

SUAVE-std 是为了开发者方便使用而建立的项目，可以理解为开发工具库。例如，它包装了如何使用 HTTP 请求，甚至在此基础上包装了一个使用 ChatGPT 的代码库，这使得开发者无需自己组装 ChatGPT 的请求报文和解析 ChatGPT 的返回报文，只需要在组装 HTTP 请求报文时替换自己的 API key 即可。而 TEE 安全环境保证了 API key 的安全，因为这一切都是在 TEE 环境下进行的。最初，这个 ChatGPT 的标准库默认使用的是 GPT-3.5-turbo 模型，temperature 默认为 0.7。现在增加了灵活的接口，也可以将模型这些作为参数传递进去。

SUAVE-examples 项目主要是为了展示如何进行应用开发的一些案例，或者说是初学者教程更为合适。对于刚接触 SUAVE 应用开发的开发者，可以通过这个项目中的案例进行学习和比对。

# SUAVE 开发实践

由于 SUAVE 是基于以太坊扩展的（它的可执行环境被称为 MEVM，即 Modified Ethereum Virtual Machine），所以智能合约的开发是 EVM 兼容的，官方开发文档都是以 Solidity 进行介绍的。因此，对于开发者而言，Solidity 的开发经验是完全可以用得上的。SUAVE 应用开发中，智能合约的开发可以理解为加上 TEE 环境下加密计算功能的 Solidity 开发。

有几个关键的 SUAVE MEVM precompiles。第一个是 confidentialInputs，这个 precompile 接受来自应用请求时的加密参数，这个参数通常是一些需要加密的私密信息，比如私钥、API key 等，其安全性的保证必然要求其明文只能出现在 TEE 环境里，而在应用开发中，获得这个信息就靠这个接口获得明文。其传输过程是全加密以及安全可靠的，其原理我们后面再说。第二个是 confidentialStore，其作用是存储私密信息，当我们从参数中获取私密信息后，往往当时不需要其参与计算，所以存储起来以备后续使用。第三个是 confidentialRetrieve，这个接口就是后续需要私密信息参与计算时向 TEE 上下文环境请求数据明文时用的。

SUAVE 对私密信息的安全存储，使得开发者可以做到这样的场景：“用户将私钥上传，然后第三方进行业务计算，当满足条件时，第三方能够直接使用用户的私钥进行签名。这样第三方能够在一定规则下使用用户的私钥进行签名，但是第三方绝对无法获取到私钥明文。”

SUAVE 使用 HTTPS 请求进行跨链的操作。其工具集里有一个名为 gateway 的库直接进行跨链信息读取，其本质是用户设定某链的 RPC node，更常见的是用户将诸如 Infura、Etherscan 等的 API key 信息上传，然后在需要调用时，直接使用 HTTP 请求到相应的节点即可。而需要进行跨链写信息的时候，工具集里有 transaction 包，能够帮助开发者对诸如 EIP1559 的报文进行 encode，最后通过 eth\_sendRawTransaction 接口进行交易的广播。

还有一个使用场景值得一提，就是将 Solidity 编译后的 bytecode 作为私密参数上传并进行存储，等到满足条件时进行 deploy 并进行调用，这样就形成了私有的库。这个使用场景可以扩展为：私钥 + 私有 bytecode 库。这样的话，在进行第三方委托调用的时候，可以做到完全隐私的交易。

# SUAVE 特点

SUAVE 最终的状态是一条链，我们称之为 SUAVE 链。SUAVE 链我们可以视之为实现了 MEVM 的一条链。既然是 EVM 兼容的区块链，那么我们同样可以在 SUAVE 上建立诸如 ERC20、ERC721 等资产，其链上操作与 EVM 系列的链没有不同。但是它的独特之处在于加入了链下的操作，诸如向其他链的节点发送交易，链下的操作结果或者是使用条件可以在 SUAVE 链上进行存储，存储的结果是有共识保证的。这样就能做到链下计算与链上状态的一致性。举个例子，开发者可以编写一个智能合约，在链上记录一些条件（也可以修改），当访问某个链网络节点，返回的结果满足要求，就进行预先设定的某 ERC20 资产的转移。

以上都是 SUAVE 的链下可信计算带来的特性。我们知道，SUAVE 是 Flashbots 团队开发的，而且 SUAVE 被 Flashbots 团队视为「The Future of MEV」，所以 bundle 交易的处理肯定是要有的，基于可信环境的 SUAVE 链，MEV 相关原理非常简单：组装 bundle 交易，发送至 Flashbots 的 relay 节点。私钥可以私密存储，甚至代码也可以，这就组成了巨大的使用潜力。比如 builder 能够得到除了在目标链上的 gas 奖励，他还能在 SUAVE 链上获得某些数字资产。对于 MEV 市场而言，能够在私密信息有安全保证的情况下灵活定义业务，这都是目前 MEV 做不到的（目前只能链下传统的基于信任、合同、商誉等保证）。

# SUAVE开发工具以及基础设施

对于开发者而言，一个 dapp 的开发，除了链上智能合约开发，前端开发中诸如 ether.js 等工具集也是重要的一环。SUAVE 应用的开发中，因为 SUAVE 链是基于 EVM 改造的，所以 ether.js，web3.js 等工具也是可以用的，这些工具在与 SUAVE 链上的智能合约交互和与其他 EVM 兼容链没有不同，但是只能调用非 confidential 环境的功能。一个 SUAVE 链的智能合约，分为链上（指 SUAVE 链）和链下（跨链的操作也算这一类）操作，链下操作实际上指的是 confidential 环境计算。对于 confidential 环境计算，Flashbots 团队提供了两种语言的 SDK（Go 和 TypeScript），使用方式在 SUAVE 的文档中有介绍。向 SUAVE 节点发送涉及隐私计算交易（Flashbots 团队称之为 Confidential Compute Request）时，能够带入 confidentialinputs，就是私密参数，这个参数在整个传输过程中，最终明文只会出现在 TEE 环境里。

最后说到智能合约的部署，SUAVE 链的测试网名字叫做 Regil，不过现在已经升级了叫做 Toliman，部署方法在 SUAVE 文档中有详细介绍。其部署的方式、部署后交互方式等等这些与以太坊智能合约部署没有什么不同。

# Kettle

智能合约部署之后，其实际的运行方式与以太坊有所不同。SUAVE 最主要的一个执行单元称之为 Kettle。Kettle 就是 SUAVE 的 TEE运行环境（它包括一个 MEVM 节点和一个 confiditial data store）。当开发者编写智能合约并部署后，用户发送 confidential compute request（以下简称 CCR），智能合约需要用到 confidential compute 的时候，实际上都是 Kettle 来运行的。

Kettle的结构图如下：



我们可以看到，开发者使用 solidity 语言开发、部署应用，最终请求到了 Kettle 之后，都是 MEVM 处理的，MEVM 里面除了有 geth 的功能，还在其上面增加了一些precompile，可以存储，检索私密数据等。此外，它还处理（包括修改、检索）SUAVE链上的状态。

Kettle 主要的工作是接收、处理私密计算，还有处理私密数据存储、检索等。以存储某私密数据为例，整个流程是这样的：用户前端使用 SDK 或者 suave geth 工具向 SUAVE 链上某智能合约发起 CCR 请求，SDK 或者 suave geth 工具会使用数据密钥（对称密钥）对私密数据进行加密，这个数据密钥只会在 Kettle 的环境中才会出现，SUAVE 的 RPC node 也只会看到密文。kettle 与节点是否是一对一的关系，这个从 SUAVE 的文档中没有看到。同理 Kettle 自身、节点以及密钥交换的详细原理，在文档中没有介绍。不过基于已知的加解密流程，开发者有理由相信私密数据从用户前端到 Kettle 内部 TEE 环境之间，数据的保护能够得到保证。

私密数据 Kettle 会保存在 confiditial data store，开发者在开发智能合约时，会指定数据的访问者和修改者，Kettle 会通过其 Transport 网络进行发布，如果是指定本合约访问，那么后续的 CCR 请求也必须发送到这个 Kettle，因为 Kettle 的数据存储并非全局更新的。当开发者将智能合约部署后，用户访问相应的 Kettle（CCR 请求里有一个参数，必须指定 Kettle 地址），其私密数据是能够访问的。当用户发送 CCR，在智能合约里请求私密数据时，使用存储相应数据时建立的 ID 以及 key 进行检索的，也就是说，私密数据访问是通过其键值访问和使用的。

有关 HTTP 请求等，也都是 Kettle 处理的。很明显，这些是属于 SUAVE 链外的工作，也就是说这些工作是单节点运行的，虽然 SUAVE 是一个链，但是其区块链的属性较弱，当 Kettle 运行 CCR 请求时，是不会有很多节点运行，然后验证的。其道理很简单，访问链外的资源，肯定是无法保证一定等幂的。所以这些属于 SUAVE 链外的工作，其结果实际上是依赖节点的。所以，开发者要注意部署时候的 Kettle 地址（这一点看，Kettle可以看作一个特殊的智能合约），后续用户 CCR 请求要带相应的 Kettle 地址。

此外，还有个问题值得开发者注意。当前测试网 Toliman 上，kettle 是不保证运行在 TEE 环境的。所以，在测试网上开发智能合约的时候，要注意保护私密数据，不要把真正的私密数据泄露。

# 总结

SUAVE 链通过引入TEE 环境，给应用开发带来了足够强大的能力，其潜在的应用场景是非常多的。其简洁便利的跨链操作，也给 Dapp 的设计带来了足够的想象空间。

SUAVE 链的 Kettle 设计是能够处理链外资源的，这就带来了验证和共识的问题。不诚实的 Kettle，对网络是有摧毁性的。如何保证 Kettle 不做恶，或者做恶能够得到处罚，或者说保证做恶的成本足够高，这都是需要解决的问题。SUAVE 链的共识采用的 PoA模式，其是否经得住实践的考虑，开发者还在拭目以待。