海葵鱼安全组件

使用说明书

岸思科技



目录

[1. 概述 1](#_Toc519955006)

[1.1. 背景 1](#_Toc519955007)

[1.1.1. 安全性问题 1](#_Toc519955008)

[1.1.2. 易用性问题 1](#_Toc519955009)

[1.1.3. IBC体系 1](#_Toc519955010)

[1.1.4. 功能特点 1](#_Toc519955011)

[1.2. 方案与架构 1](#_Toc519955012)

[1.2.1. 系统架构 1](#_Toc519955013)

[1.2.2. 操作流程 1](#_Toc519955014)

[1.2.3. Token 1](#_Toc519955015)

[1.2.4. Password 1](#_Toc519955016)

[1.2.5. 私钥 1](#_Toc519955017)

[1.3. 安全性分析 1](#_Toc519955018)

[1.4. 易用性分析 1](#_Toc519955019)

[2. 安全注意事项 1](#_Toc519955020)

[2.1. 安全连接 1](#_Toc519955021)

[2.2. 验签 1](#_Toc519955022)

[2.3. Password的处理 1](#_Toc519955023)

[2.4. 设备信息的处理 1](#_Toc519955024)

[2.5. 不同安全级别的应用场景 1](#_Toc519955025)

[2.6. 第二鉴权方式 1](#_Toc519955026)

[2.7. 挂失与遗忘Password 1](#_Toc519955027)

[3. 配置与接口 1](#_Toc519955028)

[3.1. 注册与生成AppID 1](#_Toc519955029)

[3.2. Neo Service 1](#_Toc519955030)

[3.2.1. 安装和配置 1](#_Toc519955031)

[3.2.2. 接口描述和使用示例 1](#_Toc519955032)

[3.3. Nemo Kit(ios android) 1](#_Toc519955033)

[3.3.1. 使用说明 1](#_Toc519955034)

[3.3.2. 接口描述和使用示例 1](#_Toc519955035)

[4. 结语 1](#_Toc519955036)

# 概述

海葵鱼安全组件是一套面向开发者的用于鉴权的安全组件，可以在绝大部分服务端和iOS及Android平台运行。

与传统的鉴权方式相比，海葵鱼安全组件在安全性和易用性上都有极大提升。

## 背景

服务端对客户端的鉴权，当前绝大部分产品使用Password模式，在较高安全需求的应用场景中，也有使用基于PKI体系的鉴权模式。但在多年的实践中，这两种方式在安全性和易用性上都显露出了诸多弊端。

### 安全性问题

#### Password的存储

基于Password的鉴权方式，其最大的难点在于Password的在服务端的存储，这需要依赖于数据库。

但与数据库相关的开发，往往很难避免遗留下SQL注入漏洞，使得攻击者有了批量盗取的可乘之机。

直接在数据库中存储Password的明文显然极为危险，但就算存储Password的Hash值也并不安全，因为存在彩虹表攻击。以至于Password大规模泄露屡见不鲜。

#### 私钥的存储

对应的，在某些高安全应用场景下使用PKI体系来进行鉴权，但私钥在客户端的存储更是危机四伏。尤其是在Android这种开放平台上，即使对私钥使用了再多的保护手段，一个“应用克隆”漏洞就能让这些保护形同虚设。

2018年初，攻击者利用“应用克隆”漏洞，攻破了大量的主流Android App，直接将沙盒里的所有东西，包括私钥，一并复制过去。在这种攻击下，即使最先进的白盒密码技术也无能为力。

传统上解决私钥安全问题，依靠的是如智能卡、U盾等安全硬件，但这将极大降低用户体验。

#### 登录管理

传统上的登录管理，其中的权限部分需要依托于数据库来实现。但如果系统存在SQL注入漏洞，那攻击者就可以通过在SQL语句中注入恒真或恒假语句的方式跳过权限判断逻辑。

#### 设备绑定

在一些高安全应用场景中使用设备绑定的做法，但目前的所有设备绑定皆依靠数据库实现。同样，如果系统存在SQL注入漏洞，那攻击者就可以通过SQL注入攻击绕过设备绑定的判断逻辑。

### 易用性问题

#### PKI体系的实现

在开发PKI体系时，需要做CA相关的开发，还需要涉及密钥生成、数字证书颁发、签名、证书链验证、签名验证等相关开发，流程繁琐、编程复杂。而且数字证书在使用上非常不灵活。

因此，很多产品在使用时无法做到双向认证，只能做到客户端认证服务端，或者只能做到每个客户端使用同一个证书的“伪双向认证”。

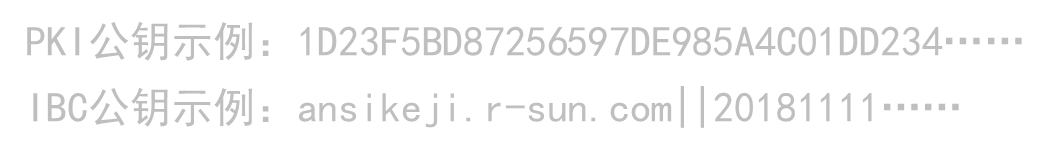
#### 登录管理与设备绑定

当这两项都依托于数据库来实现时，就涉及到较为复杂的判断逻辑，这在开发上并不容易，而且容易产生Bug。

### IBC体系

为了解决上述安全性和易用性问题，海葵鱼安全组件使用了IBC(Identity-Based Cryptography)体系。

IBC体系与PKI体系最大的不同在于：PKI体系中的公钥为一段近乎随机的信息，无密码学以外的含义，因此需要通过数字证书来与用户的身份做绑定，因此体系复杂，验证繁琐；在IBC体系中的公钥是一段有现实意义的信息，无需额外的措施来保证这一点，由此大大简化了流程，并产生了更灵活的新应用模式。



### 功能特点

通过对IBC体系的创新性应用，海葵鱼安全组件做到了：

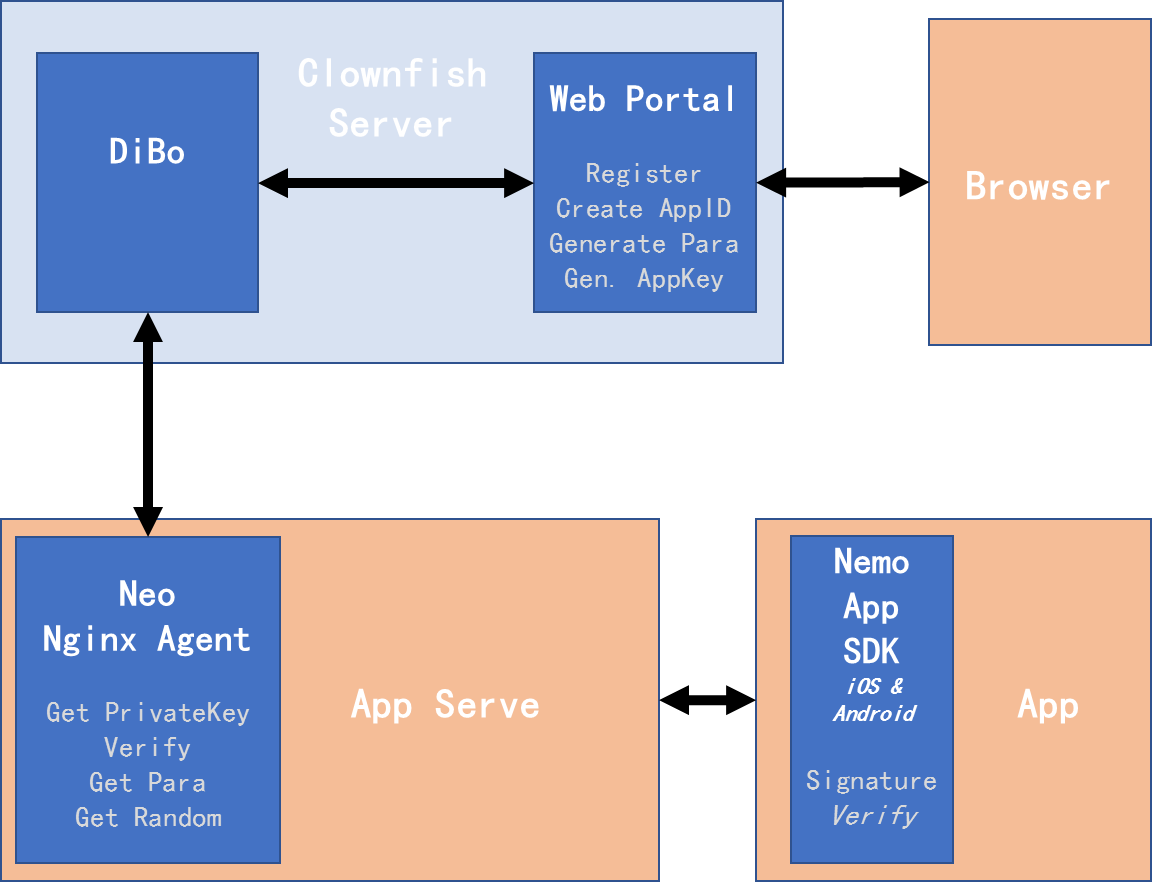
* Password零存储：不需要存储任何Password或其相关信息。
* 私钥抗泄露：单纯的私钥泄露不会导致危险的后果。
* 高效验签：只需要验签一次，而非递归验证证书链。
* 安全登录管理：登录验证时不需要访问数据库，简单且安全。
* 安全设备绑定：不基于数据库实现的设备绑定，快速且稳固。
* 高扩展性：赋予开发者极大的扩展空间，可以灵活地适应业务需求。
* 快速部署：开发完备的组件可以非常方便且快速地部署到服务端和各个平台的客户端。
* 安全等级管理：简洁实现安全等级管理，平衡安全性与易用性。

## 方案与架构

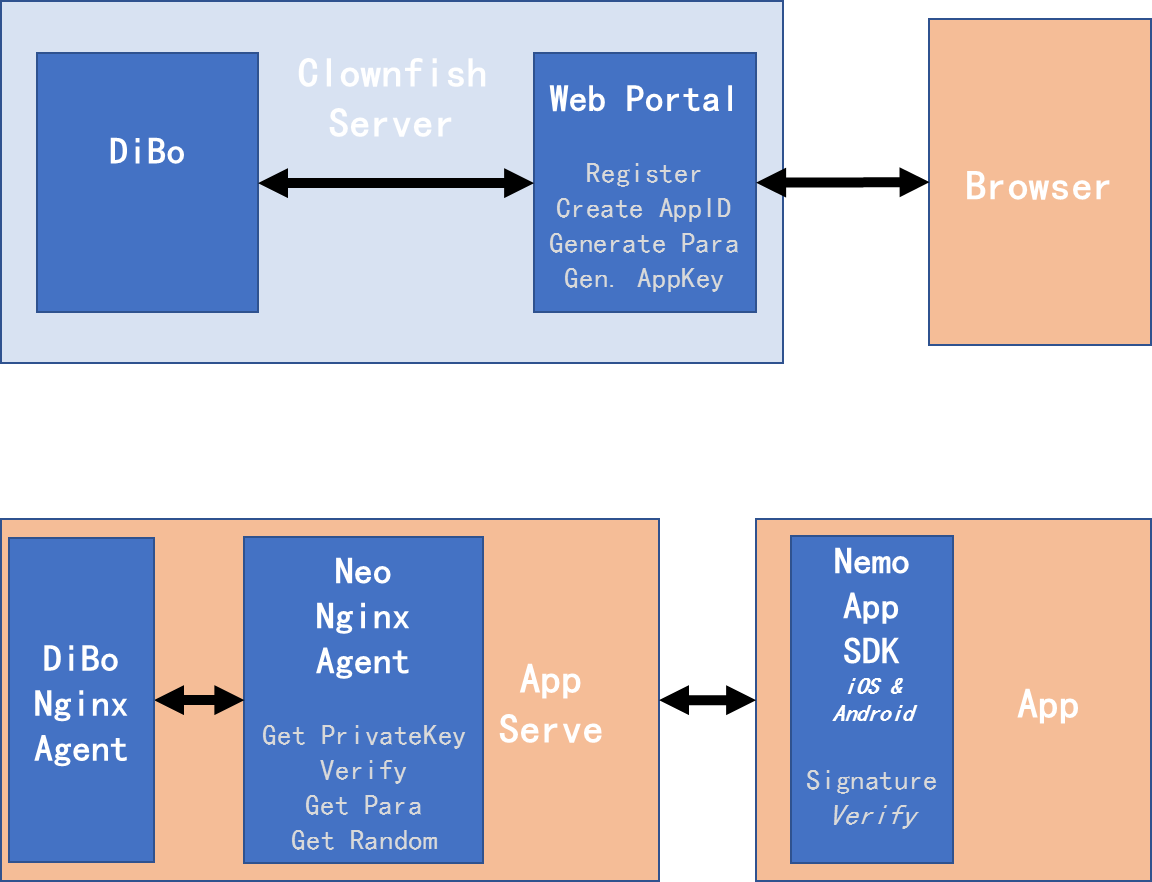
### 系统架构

在使用上，海葵鱼有两种架构形式。

架构1：



架构2：



其中，**DiBo**为用于私钥生成的核心安全模块，**Neo**为放置于应用服务端的代理，**Nemo**为放置于应用客户端的SDK。

出于对DiBo模块的极高安全防护要求，我们强烈建议使用架构1。如果开发者坚持使用架构2，则需自行负责对DiBo的安全防护。

### 操作流程

#### 注册与AppID生成

开发者登录海葵鱼安全组件的官方网站（www.r-sun.top），在上面完成注册流程。

注册完毕后，开发者可以为自己开发的每一个应用生成一个AppID，当然也可以几个应用共用一个AppID，但在从海葵鱼的角度看，每一个AppID就意味着一套独立的体系，不同体系之间不能交叉使用。

生成完AppID后，海葵鱼服务器会为每一个AppID生成一个AppKey和一套AppParameter，开发者可以在登录后从海葵鱼官网获取。AppKey可以在AppID管理界面修改。

注：上述描述仅限于架构1，架构2请联系岸思公司洽谈。在测试阶段，上述功能可能并不开放在网站上，请联系岸思公司完成上述操作。联系电话:18611150177;邮箱:1276121866@qq.com

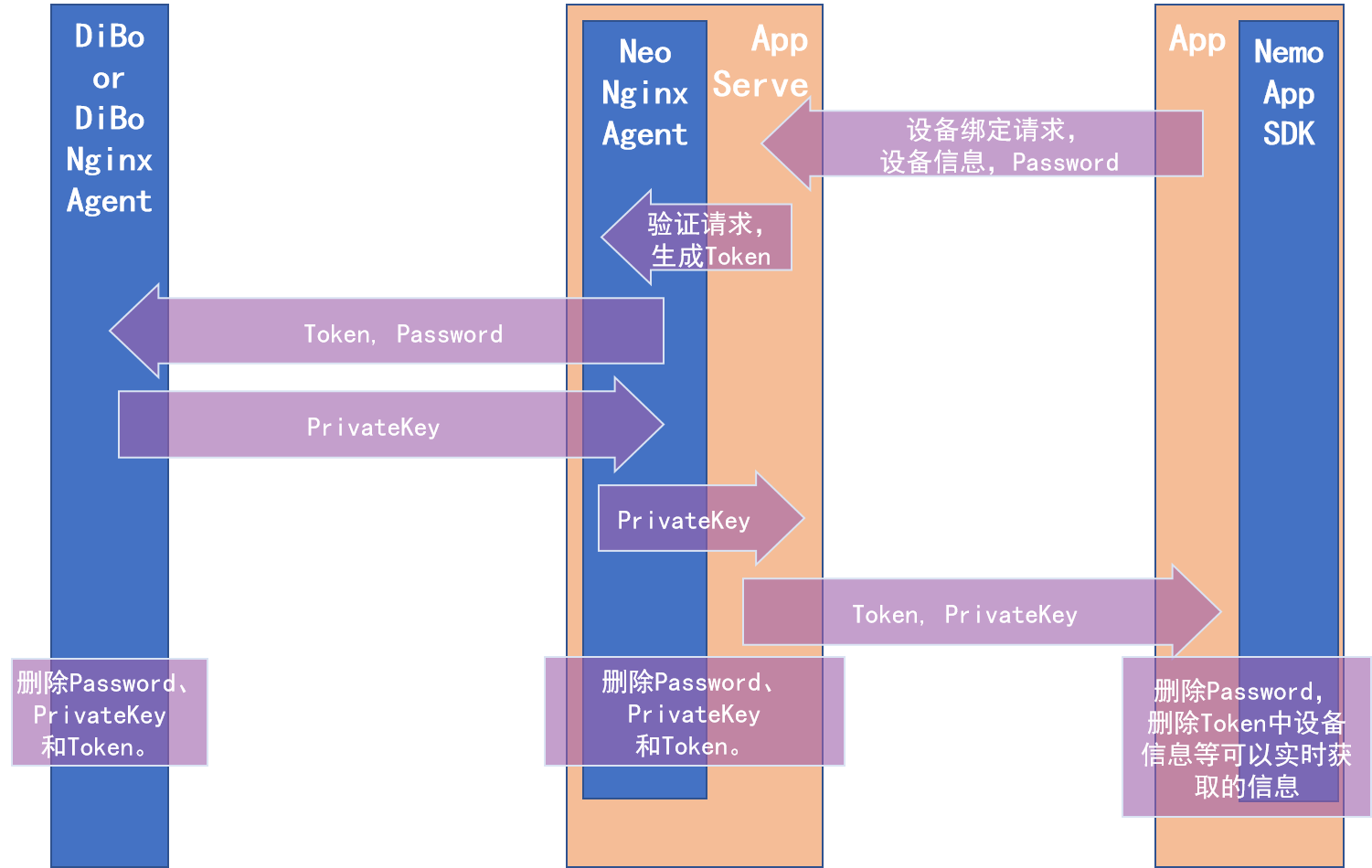
#### 设备绑定

开发者首先需要去海葵鱼官方网站（www.r-sun.top）下载Neo和Nemo，并以此为基础进行开发。注：在测试阶段，上述组件可能不开放在网站上，请联系岸思公司获取。联系电话:18611150177;邮箱:1276121866@qq.com

配置阶段：开发者将Neo加载到自己的服务端，并将AppID、AppParameter等参数配置到Neo。开发者还需要将AppParameter配置到客户端。详情请参阅下文的示例。

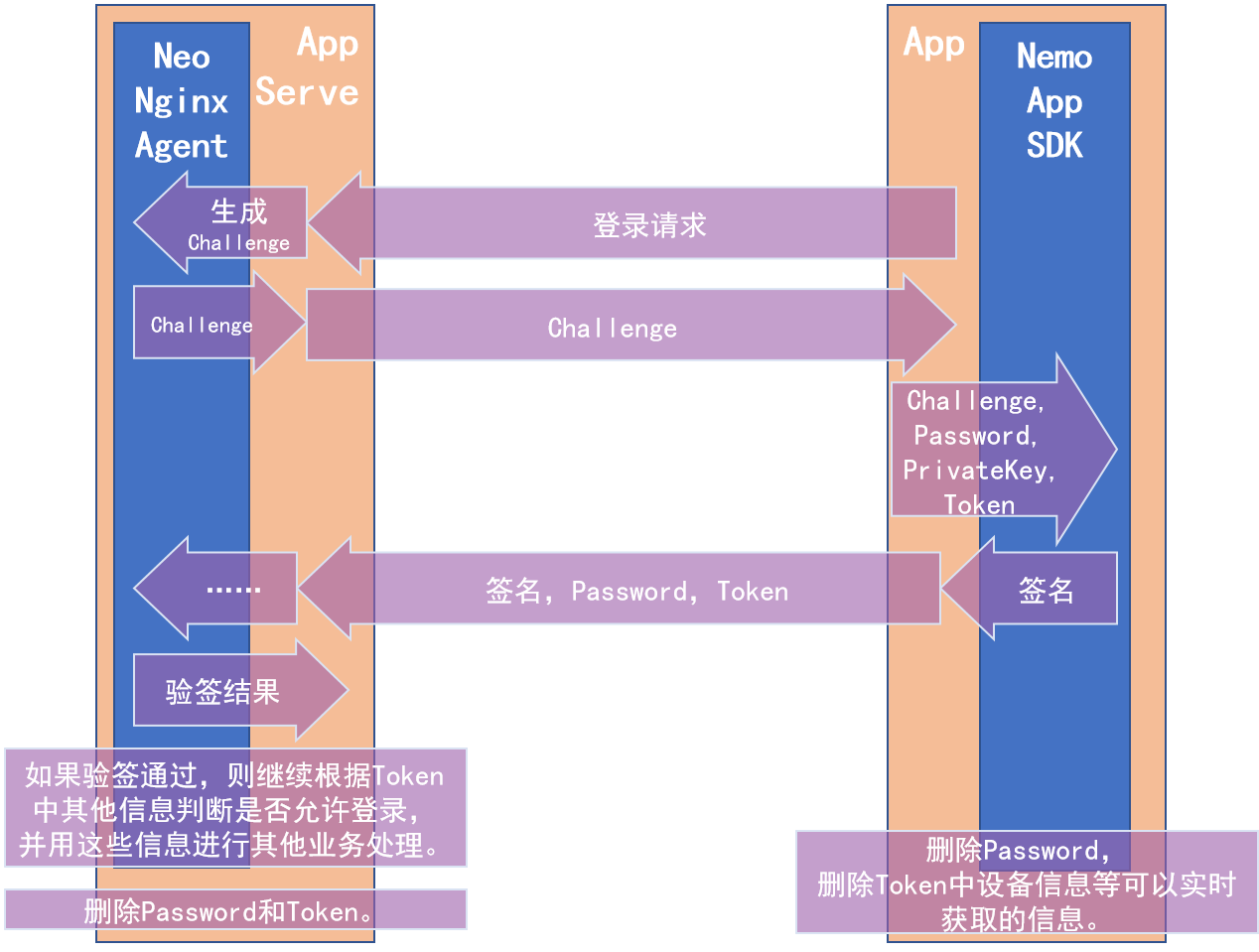
设备绑定阶段：

1. 客户端向服务端发起设备绑定请求，同时将该唯一标识该设备的设备信息和由用户输入Password一并发送过来。
2. 服务端验证该请求并通过后，基于这些信息生成一个Token，并调用Neo的Get\_Private\_Key接口，输入Token和Password为该Token生成一个私钥。
3. Neo将会通过DiBo生成一个私钥(PrivateKey)并返回给服务端。
4. 服务端将PrivateKey发送给客户端。客户端保存PrivateKey，并彻底删除Password以及Token中诸如设备信息等可以实时获取的信息。
5. 操作完成后，DiBo和Neo将彻底删除Token、PrivateKey和Password。开发者必须让服务端彻底删除Password和PrivateKey，并建议开发者在服务端删除整个Token，或者删除Token中诸如设备信息等关键信息。



#### 登录

1. 客户端向服务端发起登录请求。
2. 服务端调用Neo的Get\_Challenge接口，Neo将为服务端返回一个Challenge。
3. 服务端将Challenge发给客户端。
4. 客户端获取用户输入的Password，并实时获取设备信息等信息组成完成的Token。调用Nemo的Signature接口，输入Challenge、Password、Token和PrivateKey。Nemo将返回一个数字签名。
5. 客户端将数字签名、Token和Password发送给服务端。客户端随即立刻彻底删除Password以及Token中诸如设备信息等可以实时获取的信息。
6. 服务端调用Neo的Verify接口，并输入这些信息和Challenge。Neo将返回对数字签名的验证结果。
7. 如果数字签名有效，则继续根据Token中其他信息判断是否允许登录，并用这些信息进行其他业务处理。如果签名无效，则拒绝登录。
8. 操作完毕后，Neo将彻底删除Password和Token。开发者必须让服务端彻底删除Password，并建议开发者在服务端删除整个Token，或者删除Token中诸如设备信息等关键信息。



### Token

Token是整套安全体系关键所在。Token是一个由开发者定义的信息串，在功能上近似相当于IBC体系的公钥。

虽然Token的内容完全可以由开发者自由定义，但为了达到安全目的，我们强烈建议Token中应该至少包含如下内容：

* **账户ID或账户名**：登录的对象
* **设备信息**：用于设备绑定
* **账户权限信息**：用于登录后的权限管理
* **Token编号**：便于Token管理，防止生成重复的Token
* **Token失效日期**：便于Token管理

Token示例：

**Ansi Tech.|| iPhone7 F17RH5U12345678||Read Only||00000017||20220718**

Token在设备绑定时由服务端生成，服务端根据客户端上传的信息，根据开发者定义的逻辑生成Token。

开发者还可以往Token中自由地添加更多的信息，以满足各种业务需求。Token中的各种信息，不但可以作为登录时验证的依据，还可以帮助服务端减少数据库的压力。

### Password

Password由客户端的使用者输入，这里的Password为设备Password，只针对当前设备。

Password也可以使用手势密码等确定性信息。

在设备绑定结束以及登录结束后，服务端与客户端都应该将Password彻底删除，以实现Password零存储的特性。

海葵鱼安全组件的任何部分都严格遵循Password零存储原则，采取用后即焚策略。

### 私钥

私钥存储且仅存储于客户端，单纯的私钥泄露不会导致严重的安全后果。如果服务端生成的Token中存在一个不断递增的Token编号，则每一个私钥都将只生成一次而永不重复。

## 安全性分析

在安全性上，海葵鱼安全组件可以抵御如下攻击：

* 盗取Password：由于架构中服务端与客户端都不存储Password及其相关信息，使得Password被盗取的可能性大大降低。即使Password泄露，但攻击者无法获取私钥的话，依然无法攻击成功。
* 盗取私钥：在开放平台的客户端中，确实存在私钥被盗取的可能，但是因为攻击者没有获得Password，所以无法计算出有效的数字签名并通过验签。
* 非绑定设备登录：攻击者无法获取正确的设备信息组成有效的Token，因此无法登录成功。

Password、私钥和设备信息，构成三位一体的防护盾，只要三者不同时被攻击者获取，安全性就可以保障。即使存在“应用克隆”等严重漏洞，相关攻击也难以达成。

* 对访问权限的SQL攻击：由于访问权限可以直接写在Token中，因此可以不依赖数据库直接进行权限管理，从根本上杜绝了SQL攻击的可能。即使攻击者企图通过修改Token来完成SQL攻击，但因为验签是所有验证的第一步，此企图会随着验签失败而被挫败。
* 对设备绑定的SQL攻击：同样，由于设备信息是Token的一部分，而不需要由数据库存储，所以此项攻击不可行。

SQL攻击一直是服务端的重大威胁，经常防不胜防。通过使用海葵鱼安全组件，即使攻击者找到了可以进行SQL攻击的漏洞，但因为对Password、访问权限和设备绑定的攻击皆无法达成，损失将大大减少。

## 易用性分析

海葵鱼安全组件通过全新的架构和密码学技术，将易用性提升上新一个台阶。

* 减少数据库依赖：Password及其相关信息、权限信息、设备绑定信息皆不需要服务端存储，大大减少了对数据库的依赖，提高了开发和维护的效率。
* 轻松实现真双向认证：开发者在实现真双向认证时，不需要再苦恼于数字证书的颁发、证书链的验证等操作，调用海葵鱼安全组件高度集成的接口就能轻松实现真双向认证。
* 使用灵活：开发者可以在Token中加入更多的信息以完成更多的业务功能。

# 安全注意事项

虽然海葵鱼安全组件已经高度集成，但仍有一些安全事项需要开发者注意。

## 安全连接

为了防止窃听、钓鱼攻击或中间人攻击，开发者必须做到客户端可以对服务端进行认证（即单向认证），并且二者的通信处于加密状态。当前广泛使用的SSL协议即可达到此目的。

而Neo与DiBo之间的通信将是双向认证的，为了达到此效果，开发者需要对Neo进行证书配置，详情请参阅下文的示例。。

## 验签

在登录时，为了从根本上杜绝SQL注入攻击，服务端一系列验证操作的第一步，必须是调用Neo的Verify接口进行验签。

只有验签通过后，才可以进行后续的验证操作（如验证Token是否过期等）及业务操作。这么做的目的，是为了防止攻击者通过修改Token来发起SQL注入攻击。

## Password的处理

出于Password零存储的安全目标，Password只应由用户记忆。因此，海葵鱼安全组件中的任何部分都对Password严格采取用后即焚策略。开发者应该采取同样的策略，在设备绑定结束以及登录结束后，服务端与客户端都应该将Password彻底删除，尤其要注意清理缓存中的Password。

## 设备信息的处理

为了完成设备绑定的效果，客户端保存Token时应该删除其中的设备信息，在登录时，再实时获取设备信息，组装出完整的Token。这样即使存在“应用克隆”漏洞，攻击者也无法获得完整的Token。

## 不同安全级别的应用场景

为了增加便利性，开发者可以在设备绑定时为客户端生成两组Token，一组Token为上述标准Token，另一组低安全Token对应于一个默认的Password。

这样，在低安全需求的操作时，使用低安全Token进行登录，不需要用户输入Password，优化用户体验。只有在进行高安全需求的操作时才使用标准Token，这时才需要让用户输入Password。

## 第二鉴权方式

开发者可引入第二鉴权方式（如短信验证码等）为设备绑定操作进行额外验证。第二鉴权方式还可用于下述的挂失与遗忘Password处理。

## 挂失与遗忘Password

开发者可以设计一套挂失机制，当设备丢失、私钥泄露、Password泄露等情形下，用户可以进入挂失流程，此时可以采取第二鉴权方式进行验证。

挂失的实现，可以采取Token黑名单的形式。将查询Token黑名单作为登录时的验证操作的一部分，但流程顺序上依然应该排在验签之后。Token中的Token编号和Token失效日期将有助于建立一个高效紧凑的Token黑名单。

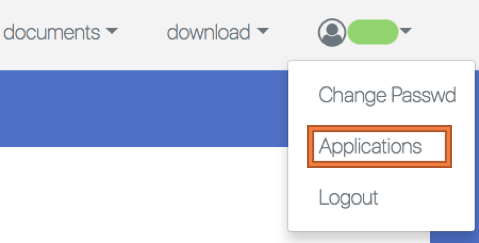
用户挂失后，可以重新为用户进行设备绑定操作，使用新的Password，生成新的Token和私钥。

用户遗忘Password时，也可采用类似的操作，重新进行设备绑定。

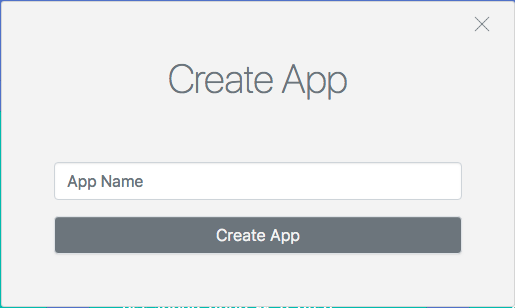
# 配置与接口

## 注册与生成AppID

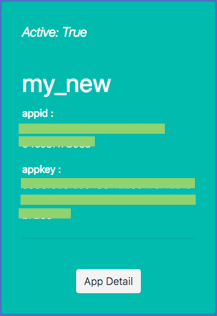
访问岸思的官方网站https://www.r-sun.top, 注册用户之后, 可以在用户菜单下找到”应用(Applications)”选项:



进入后点击”添加应用(add App)”, 输入用名称后点击”创建应用(Create app)”:



之后用户会得到如下界面并得到app\_id和app\_key:



点击”App Detail”进入详细页面, 用户可以重新生成新的appkey和parameter



详细页内的appid, appkey, parameter将被用于配置Neo Service.

## Neo Service

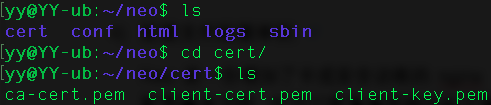
### 安装和配置

Neo Service是运行在linux ubuntu上的应用, 以下安装和配置均以ubuntu 16.04 LTS为例.

1. 下载Neo Service程序包, 复制到ubuntu服务器并解压(解压前请确认程序包的所有者):

tar xzvf neo.tar

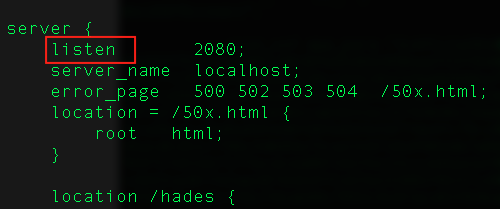
1. 进入证书配置文件夹(cert), 将从岸思获取到的证书文件放置于此(默认自带了演示版证书, 您需要向岸思索取证书:1276121866@qq.com), 如下图所示:

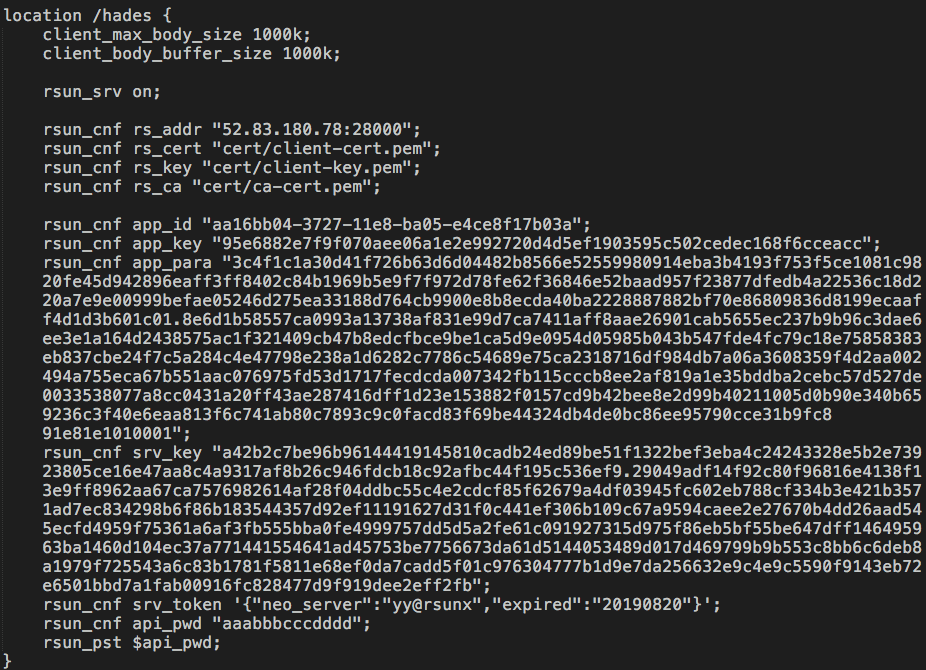


1. 进入配置文件夹(conf), 编辑配置文件, 以下以服务器自带的配置文件(neo.conf)举例:

首先需要说明的是, Neo是一台增加了海葵鱼安全功能的nginx 服务器, 你可以参考nginx的配置命令来配置基本功能, 也可以参考岸思提供的示例配置文件(neo.conf)配置服务器.

以下说明Neo Service几条关键配置:





1. Listen 关键字提示后面为服务的端口号
2. location关键字提示后面为服务baseUrl地址, 示例中是/hades, 说明了服务请求地址的baseUrl. 举例来说, 比如调用”获取挑战”接口则: url是http://ServerIP:2080/hades?cmd=challenge注意后面?cmd=之后的关键字为岸思Neo服务对应的命令, 分别为challenge, rskey, param, verify, signature, new\_pairs, encrypt.
3. rsun\_srv 关键字为服务开关, 必须为on才能服务. off关闭
4. rsun\_cnf 关键字为与岸思Neo服务相关的配置其中包括:
5. rs\_addr 为岸思dibo服务器的地址
6. rs\_cert 为岸思提供的为与dibo做双向认证的证书
7. rs\_key 为岸思提供的为与dibo做双向认证的私钥
8. rs\_ca 为岸思提供的为与dibo做双向认证的ca证书
9. app\_id 为在岸思网站上注册后提供的app\_id
10. app\_key 为在岸思网站上注册后提供的app\_key
11. srv\_token 为用户自定义的Neo服务器token.
12. srv\_key 为根据Neo服务器token生成的对应私钥(此私钥获得需要调用rskey接口).
13. api\_pwd 调用接口口令
14. rsun\_pst $api\_pwd 需要提交调用接口口令声明

### 接口描述和使用示例

#### Get\_Private\_Key

接口地址: http:// ServerIP:2080/hades?cmd=rskey

或者: http:// ServerIP:2080/hades?cmd=r

请求方式: POST, x-www-form-urlencoded

传入参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| token | string | 各个鉴权因子（如UID、设备信息、失效日期等）组成的token |
| passwd | string | 对应私钥的口令 |
| api\_pwd | string | 调用接口口令 |

返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| private | string | 对应token的私钥 |

调用示例(javascript AJAX):

var settings = {

"async": true,

"crossDomain": true,

"url": "http://10.0.2.10:2080/hades?cmd=rskey",

"method": "POST",

"headers": {

"Content-Type": "application/x-www-form-urlencoded",

"Cache-Control": "no-cache"

},

"data": {

"token": "uid\_iphone6s\_myapp101\_20180925",

"passwd": "pwd123",

"api\_pwd": "xxxxxx"

}

}

$.ajax(settings).done(function (response) {

console.log(response);

});

返回示例:

{

"private":"xxxxxx",

"status":0

}

#### Vetify

接口地址: http:// ServerIP:2080/hades?cmd=verify

或者http:// ServerIP:2080/hades?cmd=v

请求方式: POST, x-www-form-urlencoded

传入参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| token | string | 各个鉴权因子（如UID、设备信息、失效日期等）组成的token |
| signature | string | 对Challenge的数字签名 |
| challenge | string | 即Challenge |
| passwd | string | 对应私钥的口令 |
| api\_pwd | string | 调用接口口令 |

返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| verify | bool | True表示验证通过, false为不通过 |

调用示例(javascript AJAX):

var settings = {

"async": true,

"crossDomain": true,

"url": "http://10.0.2.10:2080/hades?cmd=verify",

"method": "POST",

"headers": {

"Content-Type": "application/x-www-form-urlencoded",

"Cache-Control": "no-cache"

},

"data": {

"token": "uid\_iphone6s\_myapp101\_20180925",

"signature": "",

"challenge": "6c478e2dc0a520635c0f89c1c0fa6d44.1529576213",

"passwd": "pwd123",

"api\_pwd": "xxxxxx"

}

}

$.ajax(settings).done(function (response) {

console.log(response);

});

$.ajax(settings).done(function (response) {

console.log(response);

});

返回示例:

{

"verify": true,

"status": 0

}

#### Get\_Parameter

接口地址: http://ServerIP:2080/hades?cmd=param

请求方式: GET

返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| comm\_param | string | 对应app\_id的共有参数 |
| srv\_token | string | 用户自定义的Neo服务器token |

返回示例:

{

"comm\_param": "xxxx",

"srv\_token": "xxx",

"status": 0

}

#### Get\_Random

接口地址: http:// ServerIP:2080/hades?cmd=challenge

请求方式: GET

返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| challenge | string | 获取的挑战字符串，即获取Challenge |

返回示例:

{

"rand": "challenge.9810b928486c96e582cf4efc0547125f.1534868864",

"status": 0

}

注意, 获取的Challenge字符串格式为”报头.随机数.当前时间戳”

#### 获取签名

接口地址: http:// ServerIP:2080/hades?cmd=signature

或者:http:// ServerIP:2080/hades?cmd=s

请求方式: POST, x-www-form-urlencoded

提交参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| challenge | string | 需要签名的消息(格式:报头.消息.时间戳) |
| level | string | 选取算法0为高安全级算法,1为快速算法 |
| c\_header | string | 确认签名消息的报头 |
| key | string | 可选,当为空值或不提交时,默认以服务器私钥(srv\_key)签名 |
| api\_pwd | string | 调用接口口令 |

注意: 无法对以exkey\_pub和exkey\_pri为报头的消息签名

提交示例:(JavaScript ajax)

var settings = {

"async": true,

"crossDomain": true,

"url": "http://ServerIP:2080/hades?cmd=s",

"method": "POST",

"headers": {

"Content-Type": "application/x-www-form-urlencoded",

"Cache-Control": "no-cache",

"Postman-Token": "4c1f6b16-e2e7-49c7-88cb-0ae5c948fe1a"

},

"data": {

"challenge": "challenge.43680e25e888e3142a857784c64a81a4.1534682897",

"level": "1",

"c\_header": "challenge",

"key": ""

"api\_pwd": "xxxxxx" }

}

$.ajax(settings).done(function (response) {

console.log(response);

});

返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| signature | string | 签名 |
| status | int | 状态值, 0为正常值 |

返回示例:

{

"signature": "xxx",

"status": 0

}

#### 获取交互秘钥

接口地址: http:// ServerIP:2080/hades?cmd=new\_pairs

或者:http:// ServerIP:2080/hades?cmd=n

请求方式: POST, x-www-form-urlencoded

提交参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| level | string | 选取算法0为高安全级算法,1为快速算法 |
| api\_pwd | string | 调用接口口令 |

提交示例:(JavaScript ajax)

var settings = {

"async": true,

"crossDomain": true,

"url": "http:// ServerIP:2080/hades?cmd=n",

"method": "POST",

"headers": {

"Content-Type": "application/x-www-form-urlencoded",

"Cache-Control": "no-cache",

"Postman-Token": "68683ff1-117a-4ffe-8552-9c66621dc3cd"

},

"data": {

"level": "1"

"api\_pwd": "xxxxxx" }

}

$.ajax(settings).done(function (response) {

console.log(response);

});

返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| private\_key | string | 用于交互秘钥的私钥 |
| public\_key | string | 用于交互秘钥的公钥 |
| public\_key\_signature | string | Neo服务器私钥对交换秘钥公钥的签名 |
| status | int | 状态值, 0为正常值 |

返回示例:

{

"private\_key": "exkey\_pri.49d610574f7192b94b29ca7ee8004b4509182c84fcced1f2f664be01b344c34b62218c63e2bd3e0123e80d32330c5dfdeac2c73640f2cfee3acfa6fdbc3eb6d7.1534867939",

"public\_key": " exkey\_pub.122551ace63ba22c8daae9dd105487b55ad483c11ad43152ca80d3ab53e6f6360a19476fdea58fd78dd6452f59eeee9b8dcf1fec3eb140b40edd6a8ed3077bec.1534867939",

"public\_key\_signature": "xxx",

"status": 0

}

#### 交换秘钥加解密

接口地址: http:// ServerIP:2080/hades?cmd=encrypt

或者:http:// ServerIP:2080/hades?cmd=e

请求方式: POST, x-www-form-urlencoded

提交参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| mode | string | 加密或解密, 1为加密, 0为解密 |
| text | string | 加解密文本, 加密时提交明文, 解密时提交密文 |
| key | string | 秘钥, 加密时提交公钥, 解密时提交私钥 |
| api\_pwd | string | 调用接口口令 |

提交示例:(JavaScript ajax)

var settings = {

"async": true,

"crossDomain": true,

"url": "http:// ServerIP:2080/hades?cmd=e",

"method": "POST",

"headers": {

"Content-Type": "application/x-www-form-urlencoded",

"Cache-Control": "no-cache",

"Postman-Token": "d2748075-026c-4d9e-a282-42761715ddbd"

},

"data": {

"mode": "1",

"text": "this is plain text",

"key": "xxxxxx"

"api\_pwd": "xxxxxx" }

}

$.ajax(settings).done(function (response) {

console.log(response);

}); });

返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| cipher\_str | string | 使用公钥加密后的密文 |
| plain\_str | string | 使用私钥解密后的明文 |
| status | int | 状态值, 0为正常值 |

返回示例:

{

"cipher\_str":"ea9fa6aed8dcac5a8f31252b6b3fb0df84ed118f3d4e6aae52826996d98e85efcfa8f11a0a077655ef8491bc6983f0aab84a8f61b84b20451936cbbe043aab86133c0bc2e0f5c32dd113261b0ae945eb22cd1e7fbe5a128e5bd1a05ff30cb74ff6ae5a1a8b637e7f950b4cc3f2013715aaca",

"status": 0

}

## Nemo Kit(ios android)

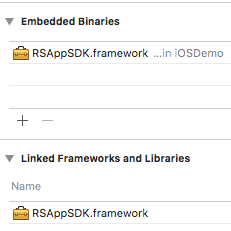
### 使用说明

**iOS端:**

将RSAppSDK.framework拖入工程中:

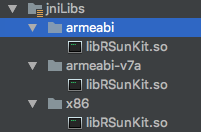


配置工程:



**Android端:**

将so文件放入jniLibs文件夹



将jar文件放入libs文件夹



### 接口描述和使用示例

#### Signature

**iOS端:**

接口:



接口说明:对获取的挑战信息签名

接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| level | int | 安全级别选择。0为最高安全级别，但速度较慢；1为中等安全级别，但速度较快。开发者可根据具体的安全需求及实际情况自行选择安全级别。 |
| privateKey | NSString | 对应token的私钥 |
| parameter | NSString | 系统的公共参数 |
| challenge | NSString | 需要签名的消息 |
| header | NSString | 消息的报头 |

调用示例:



返回说明: 对Challenge的数字签名。

返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| status | int | 0为正确 |
| signature | NSString | 对challenge的签名 |

**Android端:**

接口:



接口说明:对获取的挑战信息签名

接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| level | int | 安全级别选择。0为最高安全级别，但速度较慢；1为中等安全级别，但速度较快。开发者可根据具体的安全需求及实际情况自行选择安全级别。 |
| privateKey | String | 对应token的私钥 |
| parameter | String | 系统的公共参数 |
| challenge | String | 获取的Challenge |

返回说明: 对Challenge的数字签名。

调用示例:



返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| status | int | 0为正确 |
| signature | string | 对challenge的签名 |

#### Verify

**iOS端:**

接口:



接口说明: 验证签名信息

接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| token | NSString | 各个鉴权因子（如UID、设备信息、失效日期等）组成的token |
| passwd | NSString | token对应私钥的口令 |
| param | NSString | 系统的公共参数 |
| challenge | NSString | 获取的挑战信息，即Challenge |
| sign | NSString | 对Challenge的数字签名 |

返回说明:类型: BOOL; 值为YES时验证通过, NO为未通过.

调用示例:



**Android端:**

接口:



接口说明: 验证签名信息

接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| token | NSString | 各个鉴权因子（如UID、设备信息、失效日期等）组成的token |
| passwd | NSString | token对应私钥的口令 |
| param | NSString | 系统的公共参数 |
| challenge | NSString | 获取的挑战信息，即Challenge |
| sign | NSString | 对Challenge的数字签名 |

返回说明:类型: int; 值为1时验证通过, 0为未通过.

调用示例:



#### GetChallenge

**iOS端:**

接口:



接口说明: 获取挑战信息，即Challenge。

返回说明: 获取挑战信息, 格式为”随机数.时间戳”

调用示例:



**Android端:**

接口:



接口说明: 获取挑战信息，即Challenge。

返回说明: 获取挑战信息, 格式为”随机数.时间戳”

调用示例:



#### Hash\_password

**iOS端:**

接口:



接口说明: 获取加盐并哈希后的密码。为提高安全, 用户在输入密码后应对密码做加盐操作, 建议salt使用系统获取的uuid字符串。

返回说明: 获取加盐并哈希后的密码

调用示例:



**Android端:**

接口:



调用示例:



#### generateKeyPair

**iOS端:**

接口:



接口说明:生成交换秘钥对

接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| level | int | 安全级别选择。0为最高安全级别，但速度较慢；1为中等安全级别，但速度较快。开发者可根据具体的安全需求及实际情况自行选择安全级别。 |
| privateKey | NSString | 对应token的私钥 |
| parameter | NSString | 系统的公共参数 |

返回说明: 交换公私钥及使用token私钥对交换公钥的签名

调用示例:



返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| status | int | 0为正确 |
| private\_key | NSString | 交换秘钥私钥 |
| public\_key | NSString | 交换秘钥公钥 |
| public\_key\_signature | NSString | 公钥签名 |

**Android端:**

接口:



接口说明: 生成交换秘钥对

接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| level | int | 安全级别选择。0为最高安全级别，但速度较慢；1为中等安全级别，但速度较快。开发者可根据具体的安全需求及实际情况自行选择安全级别。 |
| privateKey | String | 对应token的私钥 |
| parameter | String | 系统的公共参数 |

返回说明: 交换公私钥及使用token私钥对交换公钥的签名。

调用示例:



返回参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| status | int | 0为正确 |
| private\_key | string | 交换秘钥私钥 |
| public\_key | string | 交换秘钥公钥 |
| public\_key\_signature | string | 公钥签名 |

#### encrypt

**iOS端:**

接口:



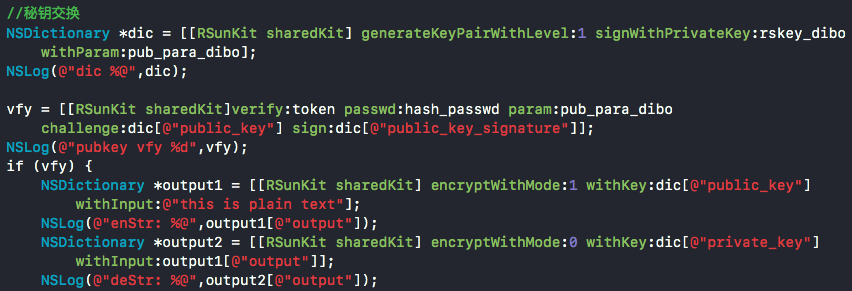
接口说明:使用交换秘钥对加解密

接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| mode | int | 加解密模式,1为加密,0为解密 |
| input | NSString | 加解密文本, 加密时提交明文, 解密时提交密文 |
| key | NSString | 秘钥, 加密时提交公钥, 解密时提交私钥 |

返回说明: 加解密文本

调用示例:



接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| status | int | 0为正确 |
| output | NSString | 加解密文本, 加密时返回密文, 解密时返回明文 |

**Android端:**

接口:



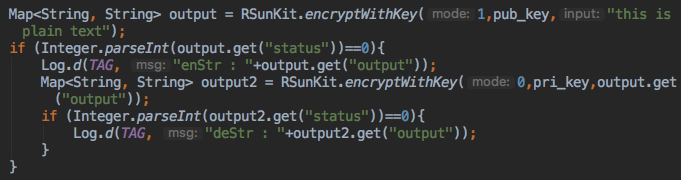
接口说明: 使用交换秘钥对加解密

接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| mode | int | 加解密模式,1为加密,0为解密 |
| input | string | 加解密文本, 加密时提交明文, 解密时提交密文 |
| key | string | 秘钥, 加密时提交公钥, 解密时提交私钥 |

返回说明: 加解密文本。

调用示例:



接口参数:

|  |  |  |
| --- | --- | --- |
| 参数名 | 类型 | 说明 |
| status | int | 0为正确 |
| output | String | 加解密文本, 加密时返回密文, 解密时返回明文 |

# 结语

海葵鱼不仅提出了一种全新的鉴权理念, 还对过去基于数据库的开发模式提出了新的思路, 开发人员可以基于开放式token开发出很多不同类型的应用. 简介而高效, 但是因为是全新的理念, 开发思路上还需要开发者不停的探索.