Computer Science and Application 计算机科学与应用, 2019, 9(12), 2394-2402
Published Online December 2019 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2019.912266

Hans汉斯

OpenStack Authentication Protocol Based on User Identity

Yan Li, Hanghang Zhang, Yingying Shui

School of Software, Zhengzhou University, Zhengzhou Henan
Email: ly79@zzu.edu.cn

Received: Dec. 4th, 2019; accepted: Dec. 17th, 2019; published: Dec. 24th, 2019

Abstract

OpenStack is an open source cloud platform management project that relies on tokens generated by Keystone components for identity authentication. In order to take into account the security of OpenStack and improve the authentication efficiency, based on the existing UUID token, PKI token, PKIZ token, and fernet token authentication, this article proposes a brand new identity authentication based on user identity. The combination of permissions uses the two-factor CPK technology to generate a unique key, which ensures the security of the token while taking into account the authentication efficiency, eliminating the steps that the service node needs to assist with the Keystone component when verifying the validity of the token, supporting local verification of the token, and simplifying the authentication process has been improved, and the efficiency of identity authentication has been improved.

Keywords

OpenStack, CPK, Authentication

基于用户标识的OpenStack身份认证协议

李　妍，张航航，水莹莹

郑州大学软件学院，河南　郑州
Email: ly79@zzu.edu.cn

收稿日期：2019年12月4日；录用日期：2019年12月17日；发布日期：2019年12月24日

摘　要

OpenStack是开源云平台管理项目，依靠Keystone组件产生的Token进行身份认证。为了兼顾到OpenStack的安全性和提升认证效率，在已有的UUID token、PKI token、PKIZ token、fernet token认证

文章引用: 李妍, 张航航, 水莹莹. 基于用户标识的 OpenStack 身份认证协议[J]. 计算机科学与应用, 2019, 9(12): 2394-2402. DOI: 10.12677/csa.2019.912266

的基础上本文提出全新的基于用户标识的身份认证，以Keystone为中心将用户身份与权限相结合，利用双因子CPK技术生成唯一密钥，在保证了Token的安全性同时兼顾到认证效率，省去服务节点验证Token的有效性时需要Keystone组件协助的步骤，支持本地验证token，简化了认证流程，提高了身份认证的效率。

关键词

OpenStack，CPK，身份认证协议

1. 引言

OpenStack 项目成立于 2010 年，Keystone 作为身份服务组件是部署 OpenStack 云平台的唯一强制服务[1]。通过创建租户、用户、角色并持久化到 mysql 数据库中，在客户端输入用户名密码，Keystone 验证通过后返回 Token 作为用户身份令牌，用户获取服务时，Keystone 根据令牌验证的结果来确定用户权限。

目前 Keystone 采用的 Token 有：UUID token、PKI token、PKIZ token、Fernet token。在 UUID token 中，用户获取 Token 时其余服务节点依靠 Keystone 确认 Token 的有效性，认证简单，但是若存在大量认证会造成 Keystone 负荷过重。在 PKI token 中支持其余组件在本地认证，却需要 CA 颁发证书，但证书过大会造成请求失败。PKIZ token 虽然对证书进行了压缩，但是也存在证书过大的缺点。Fernet token 为保证安全性则需要定期更换对称密钥。基于以上认证的不足本文提出一种基于用户标识的 Openstack 身份认证协议，利用双因子 CPK 技术和 hash 函数，支持本地认证，减轻 Keystone 负荷，同时利用 hash 函数将数据量减小解决 PKI token 过大的缺点。同时利用双因子 CPK 的安全性，解决 Fernet token 中需要定期更换对称密钥的不足的问题。

2. 相关工作总结

2.1. OpenStack 架构

OpenStack 由美国国家航天航空局(NASA)和 Rackspace 合作开发，Apache 许可证开源，是一个旨在提供基础设施即服务的虚拟化管理平台，能够提供可靠的云部署方案[2]。OpenStack 覆盖了网络、虚拟化、操作系统、服务器等各个方面。OpenStack 包含多个组件，各个组件之间既可以相互配合工作也可以独立部署完成各自的功能，在 Essex 版本之前，OpenStack 的各个子项目如 Nova、Swift 均使用各自独立的认证系统，从 Essex 版本之后开始全面支持 Keystone 提供统一的身份认证服务[3]。主要模块及功能如下。

2.1.1. 计算(Compute)：Nova

控制器，用于为单个用户或使用群组管理虚拟机实例的整个生命周期，根据用户需求来提供虚拟服务。负责虚拟机创建、开机、关机、挂起、暂停、调整、迁移、重启、销毁等操作，配置 CPU、内存等信息规格。

2.1.2. 对象存储(Object Storage)：Swift

用于在大规模可扩展系统中通过内置冗余及高容错机制实现对象存储的系统，允许进行存储或者检索文件。可为 Glance 提供镜像存储，为 Cinder 提供卷备份服务。

2.1.3. 镜像服务(Image Service)：Glance

虚拟机镜像查找及检索系统，支持多种虚拟机镜像格式(AKI、AMI、ARI、ISO、QCOW2、Raw、VDI、VHD、VMDK)，有创建上传镜像、删除镜像、编辑镜像基本信息的功能。

2.1.4. 身份服务(Identity Service)：Keystone

为 OpenStack 其他服务提供身份验证、服务规则和服务令牌的功能，管理 Domains、Projects、Users、Groups、Roles。

2.1.5. 网络&地址管理(Network)：Neutron

提供云计算的网络虚拟化技术，为 OpenStack 其他服务提供网络连接服务。为用户提供接口，可以定义 Network、Subnet、Router，配置 DHCP、DNS、负载均衡、L3 服务，网络支持 GRE、VLAN。插件架构支持许多主流的网络厂家和技术，如 OpenvSwitch。

2.1.6. 块存储(Block Storage)：Cinder

为运行实例提供稳定的数据块存储服务，它的插件驱动架构有利于块设备的创建和管理，如创建卷、删除卷，在实例上挂载和卸载卷。

2.1.7. UI 界面(Dashboard)：Horizon

OpenStack 中各种服务的 Web 管理门户，用于简化用户对服务的操作，例如：启动实例、分配 IP 地址、配置访问控制等。

2.2. OpenStack 的认证机制

Keystone 中的 User 指代任何使用 OpenStack 的实体，可以是真正的用户，其他系统或者服务。当用户请求访问 OpenStack 时，Keystone 会对其进行验证。主要有 admin、demo 两种用户。admin 为 OpenStack 平台的超级管理员，负责 OpenStack 服务的管理和访问权限。demo 为常规用户。除了 admin 和 demo OpenStack 也为 nova，cinder，glance，nutron 等服务也创建了相应的用户，admin 可以对这些用户进行统一的管理。

Keystone 中的 Credentials 是 User 用来证明自己身份的信息，可以是：用户名/密码、Token、API Key 等其他高级方式。

Keystone 中的令牌(Token)是一个字符串，每个令牌都拥有一个具体的权限范围，令牌可以确定用户能访问的资源。如果用户每次都采用用户名/密码访问 OpenStack API，容易泄露用户信息，带来安全隐患，所以 OpenStack 要求用户访问其 API 前，必须先获取 Token，然后 Token 作为用户凭据再进行访问。用户向 Keystone 发送验证请求，并提交证书，Keystone 验证证书响应请求为用户颁发一个身份认证令牌 Token。

Keystone 中的 Project 用于将 OpenStack 的资源(计算、存储和网络)进行分组和隔离。根据 OpenStack 服务对象不同，Project 可以是一个租户、部门或者项目云。在 Openstack 中资源的所有权属于 Project 而不是用户，每个用户必须挂在 Project 里才能访问该 Project 的资源。一个用户可以属于多个 Project。

keystone 中的 Endpoint 是一个网络上可以访问的地址，通常是一个 URL。各个服务通过 Endpoint 暴露自己的 API。Keystone 负责管理和维护每个服务的 Endpoint。

keystone 中的 Role 表示用户角色，代表特定的租户中的用户操作权限。

Keystone 用于为 OpenStack 中的其它组件成员提供统一的认证服务，包括身份验证、令牌的发放和校验、服务列表、用户权限的定义等等。OpenStack 中所有的服务之间的授权和认证都需要经过 Keystone。

因此 Keystone 是云平台中第一个立即需要安装的服务。作为 OpenStack 的基础支持服务，Keystone 主要用于管理用户及其权限、维护 OpenStack services 的 endpoint (服务地址)、Authentication (认证)和 Authorization (鉴权)。

OpenStack 的请求服务流程如图 1 所示：

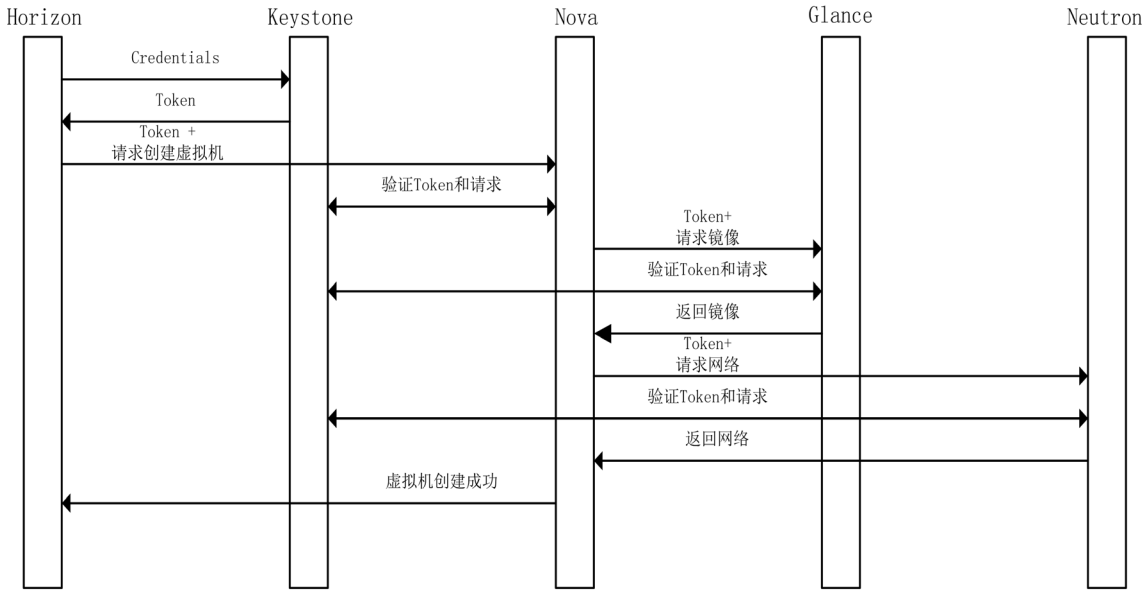

Figure 1. Keystone's authentication and request service process
图 1. Keystone 的认证及请求服务流程

用户通过 Horizon 可视化界面提供 Credentials 给 Keystone，其中 Credentials 为用户用来证明自己身份的信息，可以是用户名/密码、Token、API Key 或者其他高级方式。Keystone 验证通过后返还给用户令牌(Token)，用户使用分配的 Token 进行对其他服务的访问，提供 Token 和访问请求到目标地址，其他服务请求 Keystone 验证 Token 并返回结果来判断用户是否有该权限，从而选择是否给用户提供服务。

2.3. Keystone 现有的认证方式及缺陷

2.3.1. UUID 认证方式

用户在登陆时通过 HorizonWEB 前端的管理界面服务输入自己的用户名/密码或者用户名密码环境变量。Keystone 验证用户名密码，并且生成 Token。

客户端存储 UUID token 然后发送具体的执行请求给服务节点，服务节点发送 Token 给 keystone 请求 keystone 辅助验证，Keystone 从 http 请求中获取 Token，并检查 Token 有效性并返回结果给服务节点。若 token 有效则处理结果并返回客户端结果，若 Token 失败则拒绝客户但请求，并返回 401。

UUID token 简单易用，且长度仅为 32Byte，此外，Token 的生命周期很短，默认是 24 h，这意味着用户在 Token 过期后需要重新进行身份认证[4]。同时这种认证方式也容易给 Keystone 带来性能问题，因为 UUID token 不携带其它信息，OpenStack API 收到该 Token 后，既不能判断该 Token 是否有效，更无法得知该 Token 携带的用户信息。每当 OpenStack API 收到用户请求，都需要向 Keystone 验证该 Token 是否有效。随着集群规模的扩大，Keystone 需处理大量验证 Token 的请求，在高并发下容易出现性能问题。UUID 认证方式如图 2 所示：

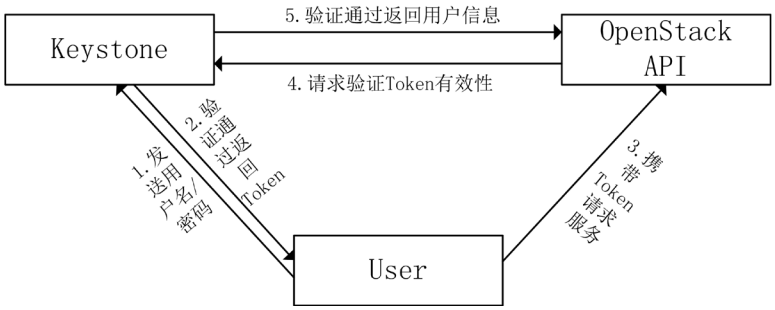

Figure 2. UUID authentication method
图 2. UUID 认证方式

2.3.2. PKI 认证

Keystone 相当于一个权威的认证中心，他用自己和的私钥对用户的 Token 进行签名。OpenStack 服务中的每一个 API 节点都有一份 Keystone 签发的证书，失效列表和根证书。API 不用在直接去 Keystone 认证 Token 是否合法，只需要根据 Keystone 的证书和失效列表就可以确定 Token 是否合法。但是这里还是会有每次都需要请求 Keystone 去获取失效列表的操作，不可避免。与 UUID 相比，PKI token 携带更多用户信息的同时还附上了数字签名，以支持本地认证。但是因为 PKI token 携带了更多的信息，这些信息就包括 service catalog，随着 OpenStack 的 Region 数增多，service catalog 携带的 endpoint 数量越多，很容易超出 HTTP 协议允许的最大 HTTP 请求头大小(默认为 8 KB)，导致 HTTP 请求失败。PKI 认证方式如图 3 所示：

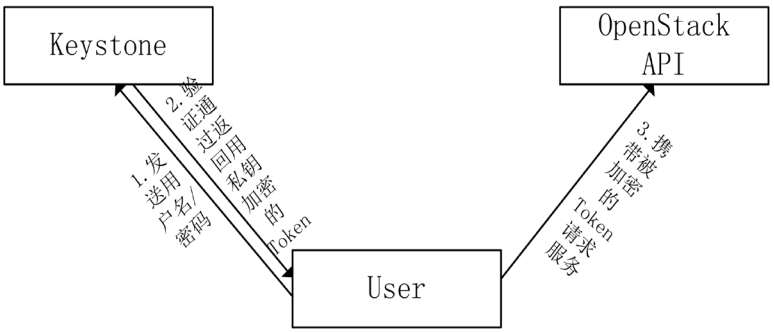

Figure 3. PKI authentication method
图 3. PKI 认证方式

2.3.3. PKIZ 认证

PLIZ 与 PKI 基本一致，但是 PKIZ 在 PKI 的基础上做了压缩处理，但是压缩的效果极其有限，一般情况下，压缩后的大小为 PKI token 的 90%左右，所以 PKIZ 也不能友好的解决 Token 太大问题。

2.3.4. Fernet 认证

Fernet 是专为 API token 设计的一种轻量级安全消息格式，它采用 cryptography 对称加密库加密 Token，具体由 AES-CBC 加密和散列函数 SHA256 签名，Token 只能通过对称密钥读取和更改，不需要持久化 Token，且 fernet token 只加密必要的信息，长度一般不超过 255 字节，避免了 PKI token 过大的问题。同时 fernet token 不需要存储于数据库，减少了磁盘的 IO，解决了 Keystone 数据库由于存储了大量的 Token 导致性能变差，需要经常清理 Token 的问题。但为了提高安全性，需要采用需要定时更换对称密钥。Fernet 认证方式如图 4 所示：

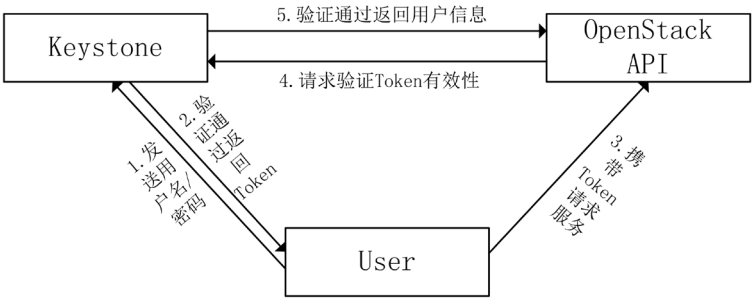

Figure 4. Fernet authentication method
图 4. Fernet 认证方式

Token 类型的选择涉及多个因素，包括 Keystone 服务器的负载、安全因素、维护成本以及 Token 本身的成熟度。从安全的角度上看，UUID 方式中每个请求都带着一个 Token，Token 不管到哪个服务，该服务都会先去验证 Token 的合法性，然后再去执行用户请求的操作[5]，但无需维护密钥，PKI token 需要妥善保管 Keystone 服务器上的私钥，Fernet token 需要周期性的更换密钥，均有不同的优势和不足，因此从安全、维护成本和成熟度上看，UUID token > PKI token/PKIZ token > Fernet token，所以综合比较 Keystone 选择的默认认证方式为 UUID。因为原有的认证方式均无法同时兼顾效率和安全性，这里我们利用组合公钥(CPK)技术提出一种全新的基于用户标识的身份认证协议，来改善认证系统。

3. 基于用户标识的身份认证协议

3.1. 认证流程

在组合公钥(CPK)体制中，组合密钥由标识密钥、系统密钥、更新密钥组成。标识密钥(Identity-Key)由实体的标识通过组合矩阵生成。系统密钥(System-Key)是由中心为各个体定义的随机序列，与标识密钥复合产生一阶组合密钥。更新密钥由个人自行定义，与一阶组合密钥再次复合，形成二阶组合密钥。

以此为基础我们引出了基于用户标识的身份认证协议。在本协议中用户私钥由两部分组成，第一部分为 Keystone 生成的标识私钥，第二部分私钥为用户自己生的伴随密钥，两者结合成为用户私钥。同理，公钥也由两部分组成，一部分为通过公钥矩阵生成为的标识公钥，另一部分为用户生的伴随私钥，两者结合为用户公钥。组合矩阵分为私钥矩阵和公钥矩阵。私钥矩阵仅存在于 Keyetone 中，公钥矩阵存在于用户客户端和各个服务节点。公钥矩阵和私钥矩阵均为 32 行 32 列，标识到矩阵坐标的映射是通过标识的 HASH 变换实现的，将 HASH 输出结果调整成 190 比特的映射序列，每 5 比特决定一个列坐标或行坐标，具体生成过程参考文献[6]。

本协议在生成过程中默认用户和 Keystone 已经协商有共享密钥，协议中用到的符号及描述如表 1 所示：

Table 1. Notation of authentication protocol
表 1. 认证协议符号记法

符号	描述
KS	keystone 认证服务器
US	用户
S	其余服务节点
iskA	用户的标识公钥

Continued

IPKA	用户的伴随私钥
askA	用户的伴随私钥
APKA	用户的伴随公钥
SIG (·)	签名操作
E (·)	加密操作
H (·)	散列操作
SK	用户与 keystone 共享密钥
CPKA	用户组合公钥
cskA	用户组合私钥
SKS	keystone 私钥
m	消息
AU	用户权限标识
ID	用户身份标识
‖	字符串连接符

1) 用户在生成请求 Token 时，首先在本地生成伴随密钥(用户自定义产生的伴随私钥 askA 和伴随公钥 APKA)，并且将伴随公钥和请求信息一同发送给 Keystone。

$$US \rightarrow KS: E_{SK}(APKA \| m)$$

2) Keystone 在返还给用户消息时，首先使用用户 ID、时间戳 T 和权限计算散列函数值，然后再根据散列函数值(H(ID||T||AU))查找标识私钥矩阵得到标识私钥(iskA)，同时成标识公钥(IPKA)，然后对用户生成的伴随公钥和标识公钥进行点加运算，生成组合公钥 CPKA，其中 CPKA = APKA+ IPKA mod n，Keystone 对伴随公钥进行签名，表示对用户该部分公钥的认可。最终 Keystone 将标识私钥、签名后的伴随公钥和组合公钥发送给用户。

$$KS \rightarrow US: E_{SK}(iskA \| SIG_{SKS}(APKA) \| CPKA)$$

3) 用户在请求其余节点服务时，用组合私钥 cskA (其中 cskA = iskA + askA mod n)对自身 ID、服务截止时间戳 T (与第一次法送给 Keystone 服务的时间一致)和权限进行签名作为 Token，并将 Token、Keystone 签名过的标识公钥，未经签名的标识公钥一同发送给服务节点。

$$US \rightarrow S: SIG_{cskA}(ID \| T \| \|AU) \| ID \| T \| AU \| SIG_{SKS}(APKA) \| APKA$$

4) 服务节点分理出未经加密的原信息和 T，先根据 T 验证用户申请是否过期，若失效则不提供服务。同时验证 Keystone 签名若通过则认可用户发送的标识公钥(APKA)，若不通过则拒绝服务，若以上两步验证均通过则进行以下验证。服务节点将原信息经过散列函数处理后根据本地标识公钥矩阵查找用户标识公钥(IPKA)，再将标识公钥与伴随公钥点加得到用户的组合公钥 CPKA，利用公钥对签名信息进行验签。比较验签后的签名和与原信息经过散列函数是否一致，若一致则用户请求正确，若不一致则表示用户请求为伪造，拒绝提供服务。

本协议的认证流程如图 5 所示：

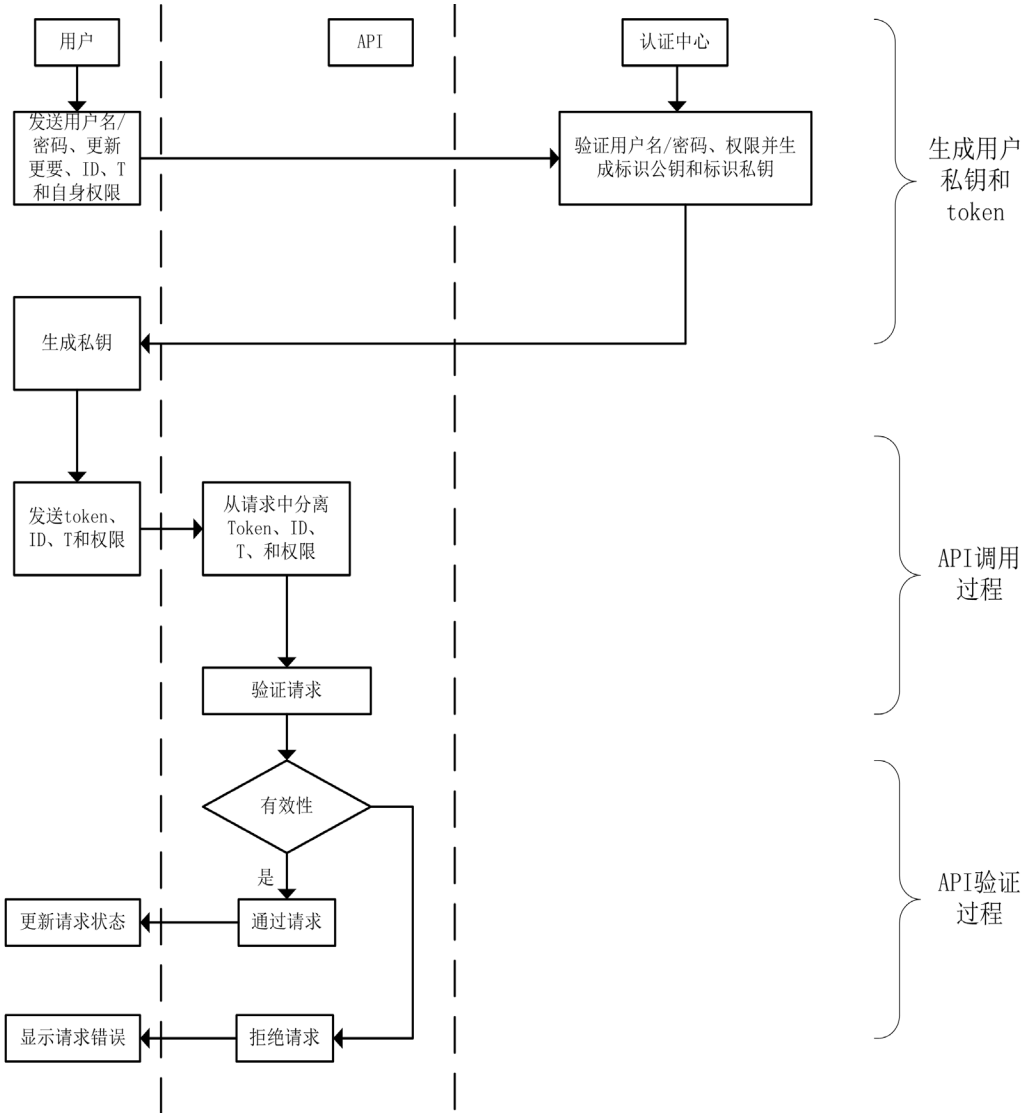

Figure 5. Certification flowchart
图 5. 认证流程图

3.2. 分析总结

- 服务器无需保存用户公私钥，Keystone 只需要保存私钥矩阵，各个其余服务节点只需要保存公钥矩阵即可验证用户身份和权限，由于私钥矩阵在中心，攻击者无法通过其余服务组件得到用户私钥，用户仅需保存好本次申请的私钥就能保证密钥安全性。
- 由于用户的一部分私钥由中心生成，且此部分私钥的生成与用户 ID、截止时间和权限相绑定，用户无法做到伪造此部分私钥，故用户无法更改 Token 的有效时间和自身权限。保证了用户的权限。
- 此协议与 UUID、PKI 认证相比较，在验证 Token 有效性时无需通过 Keystone 辅助减少了对 Keystone 的压力，支持服务节点进行本地验证，验证过程简单快速。与 Fernet 认证相比，无需定期更换对称密钥，当用户 Token 过期后只需要重新申请即可得重新得到新的密钥。且此验证方法依靠组合公钥密码体制具有破密难度大、超大规模密钥管理能力、资源消耗小、管理简便、密钥本身直接证明标识的真伪而无需第三方证明的显著优点[7]。

- 引文[8]采取了一种基于数字证书的身份认证方式，使用 Ukey 设备与用户名密码相结合，采用双层认证，用户在请求资源时，首先需要使用 Ukey 设备完成第一步认证，在得到证书后，还需向密码服务器发送认证，共需 17 次对话才能完成认证，虽然安全性大大提升，但是认证过程过于复杂。引文[9]采用了一种基于 OpnenID 的身份认证，本文与其相比，私钥中的伴随私钥为用户自己生成，并不在服务器中保存，增加了用户密钥的安全性。在引文[10]中身份认证协议依证书颁发机构 CA 颁发证书，在本文中依靠 CPK 组合公钥，给予用户主动设置部分私钥权力，私钥只有用户完全拥有，提升了安全性。

4. 结束语

本文重点对 OpenStack 云平台中的基于 Keystone 的身份认证机制进行了研究，介绍了 OpenStack 中的各个组件的作用，着重分析了 Keystone 身份认证方式，深入探究原有的 UUID，PKI，PKIZ 认证方式优缺点。分析了云平台的安全策略和威胁并在原有的 UUID，PKI，PKIZ 和 Fornet 认证的基础上结合 CPK 双因子进行改进，减少了在验证用户身份时对 Keystone 的依赖，同时也降低了对于用户 Token 的存储，提高了用户身份认证时的效率。

基金项目

郑州大学大学生创新创业训练计划资助项目。

参考文献

[1] Woo, S.W., Joh, H.C., Alhazmi, O.H., *et al*. (2011) Modeling Vulnerability Discovery Process in Apache and IIS HTTP Servers. *Computers & Security*, **30**, 50-62. https://doi.org/10.1016/j.cose.2010.10.007

[2] Wen, X., Gu, G., Li, Q., *et al*. (2012) Comparison of Open-Source Cloud Management Platforms: OpenStack and OpenNebula. 9*th International Conference on Fuzzy Systems and Knowledge Discovery*, Sichuan, 29-31 May 2012, 2457-2461. https://doi.org/10.1109/FSKD.2012.6234218

[3] Khan, R.H., Ylitalo, J. and Ahmed, A.S. (2011) OpenID Authentication as a Service in OpenStack. *The* 7*th International Conference on Information Assurance and Security*, Melaka, 5-8 December 2011, 372-377. https://doi.org/10.1109/ISIAS.2011.6122782

[4] 熊微, 房秉毅, 张云勇, 吴俊, 李素粉. OpenStack 认证安全问题研究[J]. 邮电设计技术, 2014(7): 21-25.

[5] 吴玉宁, 王欢, 苏伟, 等. OpenStack 身份认证安全性分析与改进[J]. 长春理工大学学报(自然科学版), 2015, 38(5): 112-116.

[6] 南相浩, 陈华平. 组合公钥(CPK)体制标准(Ver2.1) [J]. 金融电子化, 2009(2): 61-62.

[7] 赵小伟, 王绍斌. 基于标识算法的密钥管理体系和 CPK 认证[J]. 信息安全与通信保密, 2007(6): 200-202.

[8] 朱智强, 林韧昊, 胡翠云. 基于数字证书的 OpenStack 身份认证协议[J]. 通信学报, 2019, 40(2): 188-196.

[9] 周长春, 田晓丽, 张宁, 杨宇君, 李铎. 云计算中身份认证技术研究[J]. 计算机科学, 2016, 43(S1): 339-341.

[10] Cui, B.J. and Xi, T. (2015) Security Analysis of OpenStack Keystone. International Conference on Innovative Mobile & Internet Services in Ubiquitous Computing, Blumenau, 8-10 July 2015, 283-288. https://doi.org/10.1109/IMIS.2015.44