

OAuth Working Group
Internet-Draft
Intended status: Informational
Expires: September 26, 2018

T. Hardjono
MIT
March 25, 2018

Decentralized Service Architecture for OAuth2.0
draft-hardjono-oauth-decentralized-02

Abstract

This document proposes an alternative service architecture for user-centric control of the sharing of resources following the UMA model, such as personal data, using the decentralized peer-to-peer computing paradigm. The term 'control' is used here to denote the full capacity of the user to freely select (i) the entities with whom to share resources (e.g. data), and (ii) the entities which provide services implementing user-controlled resource sharing. The peer-to-peer service architecture uses a set of computing nodes called OAuth2.0 Nodes (ON) that are part of a peer-to-peer network as the basis for the decentralized service architecture.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 26, 2018.

Internet-Draft

Decentralized OAuth

March 2018

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
2.	The OAuth2.0 Node	5
2.1.	Node Definition	5
2.2.	OAuth2.0 Services	7
2.3.	ON Local Functions	8
2.4.	Other OAuth2.0 Terminology	8
2.5.	ON Public Keys	9
2.6.	Transaction Model	10
2.7.	Exclusivity of UMA-Services	11
2.8.	Identifying UMA-Services	11
3.	Contracts	11
3.1.	Contracts definition	12
3.2.	Smart Contracts	12
3.3.	Types of Contracts	13
3.4.	ON node acquisition contracts: parameters	13
3.5.	Resource sharing contracts: parameters	14
4.	Contracts Server in the UMA Context	15
4.1.	The Contracts server	16
4.2.	Contracts Sub-Flow in the UMA Flow	16
4.3.	Revoking a resource sharing license	17
4.4.	Cascading revocations	18
5.	Design Issues and Challenges	18
5.1.	Instrumentation of nodes	18
5.2.	Protection of private keys	19
5.3.	Throughput of smart contracts agreement	19
5.4.	Moving ON nodes across providers	19

6.	IANA Considerations	19
7.	Security Considerations	19
8.	Privacy Considerations	20
9.	Acknowledgements	20
10.	References	20

10.1.	Normative References	21
10.2.	Informative References	21
	Author's Address	22

[1.](#) Introduction

This document proposes an alternative decentralized service architecture for user-centric control of the sharing of resources, such as personal data, using the decentralized peer-to-peer computing paradigm. More specifically, we propose a decentralized service architecture based on the User Managed Access grant of OAuth (referred to as UMA2.0). As data becomes increasingly crucial for the operations of the data-driven society, the issue of privacy and control over data will become increasingly important for individuals and organizations. The current services paradigm employed today originated from the early years of the development of the Internet ISP model and as such is largely outdated.

One key important contribution of UMA is the recognition that in the real-world deployments of services there are entities (e.g. service provider and operators) which are not specified by the OAuth2.0 framework and which therefore may not visible to the resource owner. For example, UMA recognizes that the client is typically a web application that is owned and operated by a third-party service with whom the resource owner may not have a direct relationship. As such, in the basic OAuth2.0 definition the resources (such as data) flowing from the resource server to the requesting party passes through the client, even though the resource owner may not have authorized the client-operator (a service provider) to have access to this data.

In this document the term 'control' is used to denote the full capacity of the user to freely select (i) the entities with whom to share resources (e.g. data), and (ii) the entities which provide services implementing resource sharing.

We propose the use of a peer-to-peer (P2P) service architecture to

provide decentralization of services and portability of data, using digital smart contracts as the legal mechanism to bind service providers:

- o Decentralization of services: At the infrastructure level, decentralization of service means enabling a user to select the service providers that will provide the user with full control over managing access to the user's resources, in particular for user-owned data.
- o Portability of data and services: At the data level, decentralization of control means the freedom for the user to

switch service providers at any moment in time and with ease. As such, portability of data and interoperability of services across providers is crucial to allow the user to retain independence from any specific service provider.

- o Automated service-provisioning through contracts: Decentralization of service and portability of data can be enabled by automation in the provisioning and de-provisioning of services, based on an automated contracts model. Such an automated model must be legally enforceable and must enable users to switch providers rapidly without degradation in control, privacy or sharing levels.

The recent development of blockchain systems and distributed ledger technologies offers the possibility of parties transacting on a common ledger (public or private), with reduced costs through increased automation and with reduced friction through the increased visibility on the part of the transacting entities. Core to this notion of 'business on the blockchain' is the concept of the smart contract as being the digital equivalent of the paper contract.

The programmability aspect of some blockchain systems offers the possibility of using code on the blockchain to provision, activate and retire services in an automated fashion.

We propose the use of smart contract technology as a means to enable users to legally obtain and control a compute-unit which contain services compliant to well-defined standard specifications. These compute-units could be made available by infrastructure service providers in the form of operating-system-level virtualization images

(i.e. containerization), or other forms of hosted technologies. In this case our compute-unit would contain a set of standard UMA2.0 services. We refer to this compute-unit as the OAuth2.0 Node (ON). We refer to the infrastructure service providers who furnish an ON node as the node-operator (or simply operator).

Each ON node has the capability to provide AS-services, RS-services and Client-services following the UMA2.0 standard. Each node also has the capability to provide authentication services for other nodes and users, following the OpenID-Connect 1.0 model.

The peer-to-peer (P2P) model for the ON nodes enables users who possess ON nodes to transact with each other as equal peers. For example, an individual Bob as the requesting party may create an ON node and use the client in that node to request access to resources belonging to Alice. In her turn, Alice will use the authorization server and resource server in her ON node to respond to Bob's request.

Additionally, and more interestingly, Alice may stand-up a network of ON nodes where each node may be dedicated to managing certain types of resources belonging to Alice. This network of ON nodes may perform inter-federation among themselves, allowing each ON node to serve different requesting parties and clients at scale as suitable to each transaction context and resource type.

A user seeking to acquire and control an ON node should not need to be aware of how the node is implemented and instrumented by node operators. The role of smart contracts here is to ensure that the user obtains from the node operator the ON node as a compute-unit as specified, and that the user has the control over the ON node and its associated resources (user-level resources, and instrumentation or configuration resources for that node).

With regards to smart contracts -- which is currently still an evolving discipline -- one key requirement is the ability for the contractual obligations stated in the smart contract to be legally enforceable. Methods such as combining contract-code with legal-prose within the smart contract offers a possible solution to this question.

In the following we describe in more detail the functions of the node. The reader is assumed to have familiarity with OAuth2.0 [OAuth2.0], OpenID-Connect Core [OIDCCore] and UMA 2.0 [UMA2.0].

2. The OAuth2.0 Node

This document proposes the use a peer-to-peer (P2P) network of nodes as the basis for a decentralized service architecture for user-centric management of data sharing and service provisioning.

2.1. Node Definition

Each node is referred to as an OAuth2.0 Node (ON), and a ON compute-unit implements the following UMA-related services and other services (Figure 1): Authorization Server, Resource Server, Confidential Client, Policy Server, OpenID-Connect Provider, Proxy/Forwarder, Blockchain Client and Contracts Server. These services come with the ON node as a compute-unit regardless whether the owner of the node makes use of (i.e. activates) these services.

An individual or organization obtains an ON unit from the node operator by entering into a ON node acquisition contract with the node operator. The acquisition contract makes the user the "owner" of that ON unit for the duration of the contract.

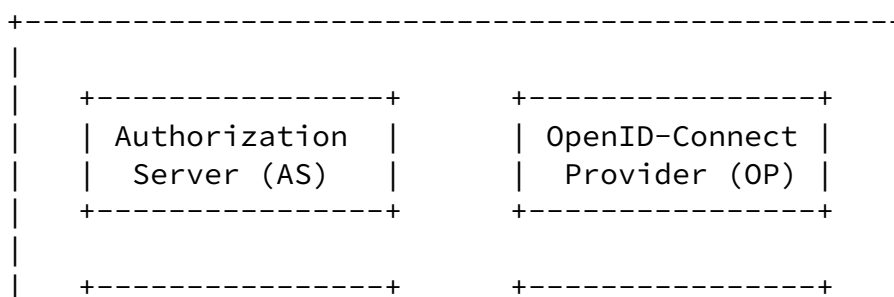
We distinguish between the node operator and the node owner:

- o Node Operator: The legal owner of the infrastructure system implementing and instrumenting an ON node throughout its lifecycle.
- o Node Owner: The legal owner of the ON unit (with its associated UMA services, functions, APIs, and resources) acquired for a duration of time under a contract with the node operator.
- o UMA-services operator: The operator of the services running within an ON node. Given that the node-operator as the infrastructure provider has mechanical control over the ON node as a hosted compute-unit (including processes running inside the node), for simplicity we assume that the node-operator is the legal operator of the UMA-services.

It is crucial to highlight here that as the infrastructure provider

of ON nodes, the node-operator is still considered as the UMA-services operator (at the UMA level), even though the user (owner) has full legal control over the ON unit. This is because the node operator who hosts the ON node still has the technical means to perform unauthorized access to resources flowing between ON nodes and to resident resources (i.e. data at rest) within an ON node. For example, when Bob employs a client within his ON node, the operator who hosts Bob's ON node still has the technical capacity to view resource traffic flowing through that client.

Emerging technologies (e.g. secure enclaves, homomorphic encryption, secure multiparty computation) promise solutions to this possibly 'dishonest' node-operator problem. But until these technologies are well-understood, tested, standardized and widely deployed, it will be difficult if not impossible to prevent (or even detect) dishonest node-operators.



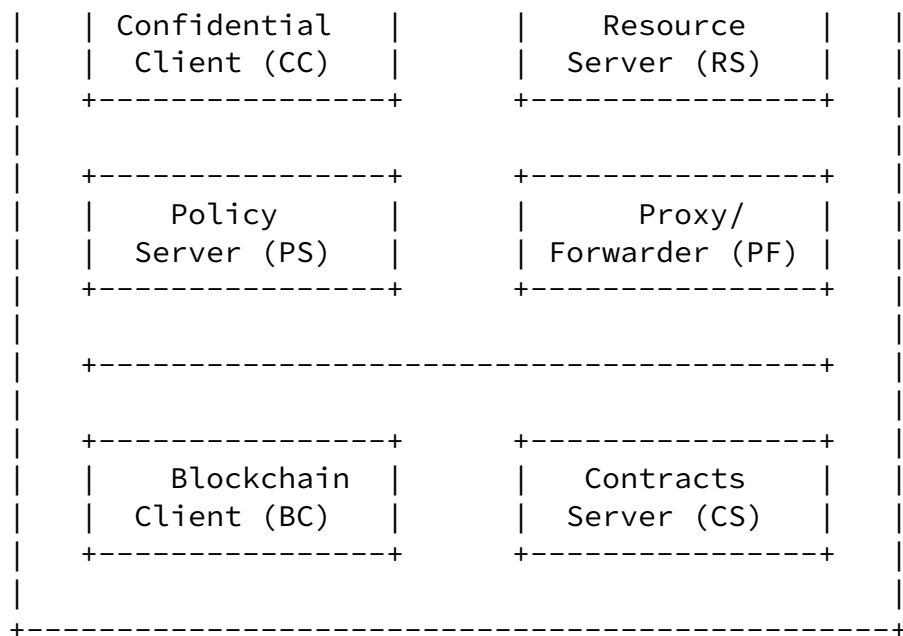


Figure 1: OAuth2.0 Node (ON)

2.2. OAuth2.0 Services

The following are services that are implemented by an OAuth2.0 Node (ON):

Confidential Client: The Confidential Client (CC) is client that possesses capabilities to store secrets, such as cryptographic keys and other confidential parameters.

Authorization Server: The Authorization Server (AS) is a server that protects resources managed at a resource server on a resource owner's behalf.

Resource Server: The Resource Server (RS) is a server that hosts resources on a Resource Owner's behalf, registers resources for protection at an Authorization Server, and is capable of accepting and responding to requests for access to protected resources.

OpenID Provider: The OpenID Provider (OP) implements authentication

of the Requesting Party and the Client. See [[OIDCCore](#)].

Policy Server: The Policy Server (PS) implements the policy administration point (PAP) and policy decision point (PDP) for the Resource Owner, for each resource owned by the Resource Owner.

Proxy/Forwarder: The Proxy/Forwarder (PF) implements proxying to another node, relying on that node's implementation of the same function.

[2.3.](#) ON Local Functions

The following are services that are implemented by an OAuth2.0 Node (ON) for its own operations, and therefore are not available to other entities:

Blockchain Client: The Blockchain Client (BC) implements the client role in a blockchain system.

Contracts Server: The Contracts Server (CS) implements the contracts management and fulfilment for users and with other ON nodes.

[2.4.](#) Other OAuth2.0 Terminology

The following is a set of terminologies used in OAuth2.0 and in UMA2.0:

Requesting Party: The Requesting Party (RpP) is a natural or legal person that uses a client to seek access to a protected resource. The requesting party may or may not be the same party as the resource owner.

Resource Owner: The Resource Owner (RO) is an entity capable of granting access to a protected resource. This is typically an end-user (a natural person) but it can also be non-human entity that is treated as a person for limited legal purposes (a legal person), such as a corporation.

Resource: A digital resource available through an HTTP service.

Protected Resource: A resource to which a resource owner is able to control access grants through an authorization server.

Scope: A bounded extent of access to a protected resource. Scopes are associated with particular resources.

Policy Conditions: Access grant rules configured at an Authorization Server that effect resource protection.

Claim: A statement of the value or values of one or more attributes of an entity.

Permission: Authorized access to a particular resource with one or more scopes. A resource server requests one or more permissions on behalf of a client at an authorization server.

[2.5.](#) ON Public Keys

For each ON node which the node operator establishes under contract with the node-owner, the operator creates a new public-key pair and assigns it to that ON node. We refer to this as the ON unit public-key pair. The operator refers to that ON node (e.g. on the blockchain) using that public key. This public-key pair is unique for each ON node created by the operator, and the operator maintains the private key.

This ON unit public-key pair is different from the operator's public-key pair as a service provider.

Once an ON node is operational and ownership has been transferred to the node-owner (e.g. Alice), the owner must generate and assign a public-key pair to each UMA-service they wish to activate in the ON node. The owner must also generate and assign a public-key pairs for the Blockchain Client (BC) and Contracts Server (CS) in that ON node.

Thus, for example, if Alice is using only the Authorization Server (AS) and Resource Server (RS) within her ON node, she must generate and assign two (2) public-key pairs, for her AS and the RS respectively, plus the public-key pairs for the BC and CS. The node operator must never be able to obtain the private half of the public keys of the services running within an ON node. The mechanics to generate and assign public-key pairs are outside the scope of the current specification.

The following is a list of public-key pairs related to a given running ON node:

- o ON unit public key: This is the public-key pair associated with the ON node as a compute unit operating on the infrastructure of the node operator.
- o UMA-services public keys: This is the public-key pair of the UMA-

service running within an ON node. When invoked or activated by the node-owner, each UMA-service must be associated with a unique

public-key pair. The blockchain client (BC) and the Contracts Server (CS) must always be operational inside an ON node and must each be assigned a public-key pair.

- o Node Operator public key: This is the public-key pair of the node operator as a legal entity. This key pair is used by the node operator for signing smart contracts.
- o Owner public-key pair: This is the public-key pair of the owner (i.e. Alice). This key pair is used by the owner for signing smart contracts

[2.6.](#) Transaction Model

The transaction model follows closely that of the UMA grant of OAuth2.0 (also referred to as UMA2.0). In summary, a Requesting Party (Bob) is seeking access to resources belonging to the Resource Owner (Alice) through a Resource Server that Alice controls.

The Requesting Party (Bob) selects an ON (e.g. Node #1) to be his Client in the transaction, while the Resource Owner has already activated an ON (e.g. Node #3) to be the Authorization Server that protects the target resource located at the Resource Server (e.g. Node #2).

In order for the Requesting Party (Bob) to access the desired resources controlled by the Resource Owner (Alice), two types of exchanges may occur as part of a transaction:

- o Requesting Party and Client authorization: When the Requesting Party uses the Client (Node #1) to request access to a resource at the Resource Server (Node #2) the Client must obtain an access token from Authorization Server (Node #3).
- o Requesting Party authentication: In the process of the Client (Node #1) obtaining an access token from the Authorization Server (Node #4), the Client may be directed to the OpenID-Provider (Node #5) for the Requesting Party and Client to be authenticated. (Note that this is part of the claims-gathering flow of UMA).

The method for Requesting Party to obtain information regarding the available resources at a given Resource Server is outside the scope of this document. The reader is directed to the UMA2.0 specifications.

The method of node selection is outside the scope of this document and will be the subject of future specifications.

[2.7.](#) Exclusivity of UMA-Services

For simplicity of design, we propose the exclusive ownership of all UMA-services within an ON node.

That is, when a user (e.g. Alice) purchases a ON node offered by an node operator, that ON node is exclusively owned by (and under the full control of) Alice throughout the duration of the contract. The node operator must not advertise otherwise, and other users and entities are able to verify the ownership of a given ON node (i.e. as belonging to Alice) by validating the relevant confirmed smart contract transaction (pertaining to the ON node acquisition by Alice from the operator) on the blockchain.

[2.8.](#) Identifying UMA-Services

UMA-Services and their endpoints within a given ON node may be identified based on the combined public keys related to the UMA-service and the ON unit. The cryptographic hash of the combined UMA-service (e.g. AS-service) public key and the ON unit public key produces a hash-value which can be used to look-up the UMA-service uniquely on a directory or on a blockchain-based naming service.

For example, both Alice and Bob may stand-up two distinct ON nodes hosted by the same node operator. Each user may run their own AS-service from within their respective ON nodes. The combined use of the AS service end-point (URI) and these two public keys offer a mechanism to distinguish between Alice's AS from Bob's AS.

[3.](#) Contracts

Contracts form the basis for ON node acquisition (on the

infrastructure level) and for resource/data sharing (at the UMA level). ON node acquisition occurs between a user (individual or organization) and an infrastructure node-operator.

Resource sharing contracts at the UMA-level capture the licensing of access right to resources belonging to the resource owner. In this sense the current proposal follows the UMA licensing model [\[UMA-Legal\]](#).

Contracts must contain legal prose that bind relevant parties and must contain indicators for dispute resolution.

Hardjono

Expires September 26, 2018

[Page 11]

Internet-Draft

Decentralized OAuth

March 2018

[3.1.](#) Contracts definition

Contracts are legally binding agreements expressed in digital format that clearly calls out the actors involved in a transaction, the resources (i.e. data) being shared, legal prose (or pointers to legal prose), methods for dispute resolution, and other legal aspects of the transaction.

At the infrastructure level, the intent here is to support users (individuals or organizations) to acquire ON node compute-units from an infrastructure node operator in an automated or semi-automated fashion, based on standardized templates of contracts.

Similarly, on the UMA level the intent is to support users (individuals or organizations) to acquire licenses for resources in an automated or semi-automated fashion.

For example, when Bob seeks to access resources belonging to Alice, Bob's node operator (e.g. BNO Corp) and Alice's node operator (e.g. ANO Corp) are involved in the transaction.

This is because when resources (e.g. data) are accessed by Bob and flows from Alice's RS to Bob's Client, the node-operator of Bob's Client (namely BNO Corp) may be able to obtain unauthorized access to those resource. The contract must protect against this possible

unauthorized access.

[3.2.](#) Smart Contracts

A smart contract is a set of promises, specified in digital form, including protocols within which the parties perform on these promises. A smart contract should have the following features [[NRF](#)]:

- o Digital form: it is in computer form - code, data and running programs.
- o Embedded: contractual clauses (or equivalent functional outcomes) are embedded as computer code in software.
- o Performance mediated by technological means: the release of payments and other actions are enabled by technology and rules-based operations.
- o Irrevocable: once initiated, the outcomes for which a smart contract is encoded to perform cannot typically be stopped (unless an outcome depends on an unmet condition).

Hardjono

Expires September 26, 2018

[Page 12]

Internet-Draft

Decentralized OAuth

March 2018

Contracts are digitally signed by the relevant parties and may be recorded to a blockchain system or distributed ledger system. A smart contract containing code and legal prose may be used to automate the service agreement process.

For ON nodes, the smart contract defining the ON node acquisition agreement must also specify the post-termination actions to be performed by the node-operator on the user's ON node. This agreement clause must define the transferal mechanism for resources and services within the user's ON node. For example, the objects and relevant assets (e.g. data; keys; tokens; etc.) could be packaged and off-loaded to a location designated by the user in the smart contract, such as another node or offline storage.

[3.3.](#) Types of Contracts

We propose distinguishing two (2) types of contracts:

- o ON node acquisition contract: A ON node acquisition contract denotes the acquisition of specific ON node as a compute-unit by a user (individual or organization) from a given infrastructure node operator. ON node acquisition contracts are typically bilateral between a user and a node-operator.
- o Resource sharing contract: A resource sharing contract denotes the granting of license by a Resource Owner (to data or resources) to a Requesting Party using the services rendered by infrastructure node operator. See [[UMA-Legal](#)].

Parties that transact at the UMA-level through a resource sharing contract must verify that (a) the counterparty possesses valid contracts with their operators, and (b) that the operator contract prohibits the operator from performing unauthorized access to the shared resources at the UMA-level.

3.4. ON node acquisition contracts: parameters

The following are some parameters needed within ON node acquisition contracts:

- o Type of contract ('ON node acquisition')
- o Identifier of the user (individual or organization)
- o Public key of the user
- o Identifier of the ON node operator

- o Public key of the ON node operator
- o Version of ON node (i.e. container/image) and hash
- o ON unit public-key
- o ON unit identifier (e.g. URI)
- o Exclusivity
- o Duration of service

- o End-of-contract actions
- o Service fees and payment mechanism (optional)
- o Dispute resolution method
- o Legal prose
- o Code (for smart contract embodiment)
- o Timestamp
- o Archive location of this contract (optional)
- o Contract template identifier and author (optional)
- o Target blockchain (for smart contract embodiment)
- o Signature of User (individual or organization)
- o Signature of node-operator

[3.5.](#) Resource sharing contracts: parameters

The following are some parameters needed within a resources sharing contract based on the usage of ON nodes. Note that this contract involves only the following UMA-entities: Requesting Party, Client, Client node-operator, Resource Owner, Resource Server, Resource-Server node operator, Authorization Server, Authorization-Server node operator. Other flows and contracts will be addressed in future documents.

- o Type of contract ('resource sharing')
- o UMA-services involved (identifier and public-key): RqP, Client, AS, RS, RO

- o Hash and pointer to the node acquisition contract between the Client-owner and their node-operator.
- o Hash and pointer to the node acquisition contract between the RO-

entity and the node-operator running the ON unit containing their Resource Server.

- o Hash and pointer to the node acquisition contract between the AS-owner and the node-operator running the ON unit containing their Authorization Server.
- o Duration of data sharing contract
- o End-of-contract actions
- o Service fees and payment mechanism (optional)
- o Dispute resolution method
- o Legal Prose
 - * Data sharing license
- o Code (for smart contract embodiment)
- o Timestamp
- o Archive location of this contract (optional)
- o Contract template identifier and author (optional)
- o Target blockchain (for smart contract embodiment)
- o Signature of Requesting Party
- o Signature of Resource Owner data

[4.](#) Contracts Server in the UMA Context

Contracts form the basis for ON node acquisition (on the infrastructure level) and for resource/data sharing (at the UMA level). ON node acquisition occurs between a user (individual or organization) and an infrastructure node-operator.

Resource sharing contracts at the UMA-level capture the licensing of access right to resources belonging to the resource owner. In this sense the current proposal follows the UMA licensing model [[UMA-Legal](#)].

Contracts must contain legal prose that bind relevant parties and must contain indicators for dispute resolution.

[4.1.](#) The Contracts server

The Contract Server (CS) is a new functional capability introduced into this architecture that did not previously exist in the OAuth2.0, OIDC1.0 or UMA2.0 designs.

The contracts server is present at an ON node regardless of which types of UMA-services are activated by the user (node owner). A contracts server in an ON node only serves that node (i.e. it serves the UMA-services active in that ON node). A contracts server cannot be used by services outside its ON node.

Some of the core tasks of the contracts server on a node are as follows:

- o Locate and validate templates of standard contracts
- o Interact with peer contracts server at other nodes (data sharing contracts)
- o Validate incoming contracts against standard template contracts
- o Validate signatures on incoming contracts
- o Sign contracts using the relevant private keys
- o Record signed contract to blockchain (using Blockchain Client)
- o Locate and validate executable-code related to a smart-contract.

Similar to endpoints in the UMA-services within an ON node, the contracts server exposes a number of endpoints relevant to completing contracts agreement (e.g. for resource sharing contracts).

[4.2.](#) Contracts Sub-Flow in the UMA Flow

There are several possible uses of the contracts server in the context of the UMA flows.

For requesting access to protected resources (at the UMA level), the role of the contracts server is to record the signed agreement between the Requesting Party, Client and the Resource Owner prior to the Authorization Server issuing an RPT token (Requesting Party Token) to the Requesting Party.

That is, the contracts server injects a sub-flow within the UMA resource access flow.

More specifically, when the Requesting Party (Bob) uses a Client (in Node #1 owned by Carol) to request access to Alice's resources at the Resource Server (in Node #2 owned by Alice), the contracts server belonging to Alice (in Node 2) must perform the following:

- a. It must prompt the Requesting Party (Bob), namely a human person, to sign the license agreement pertaining to the protected resources belonging to Alice.
- b. It must search for (on the blockchain) the valid ON node acquisition contract between Carol and her node operator. This indicates that Carol is truly the legal owner of the ON unit (Node #1) running the Client.
- c. It must engage the contracts server in Node #1 belonging to Carol to sign the license agreement pertaining to the protected resources belonging to Alice. (Note that Alice's license may already be defined a smart contract sitting on her chosen blockchain system). Both contracts server must record the signed agreement to the same blockchain system, with Alice's contracts server have first choice in the selection of the blockchain system..
- d. If the ON node acquisition contract for Carol's ON unit (Node #1) did not provide legal coverage against her operator eavesdropping (i.e. stealing data or resources), then Alice's contracts server must engage Carol's node-operator to sign a license agreement suitable for the operator.

Only after the respective contracts servers have completed the license agreement and have these recorded on the blockchain (or have it executed on the blockchain), should the RPT token issuance flow continue.

Note that in the UMA context the resource sharing license may have a long duration, which means that the above license signing sub-flow need not be repeated if the license contracts are still valid and

unrevoked.

[4.3.](#) Revoking a resource sharing license

Once of the key value proposition of UMA is the revocation of licenses, which is in-line with a number of requirements in the GDPR.

Hardjono

Expires September 26, 2018

[Page 17]

Internet-Draft

Decentralized OAuth

March 2018

In the case of license revocation, Alice instructs her contracts server to issue a license-revocation transaction (LRT) to the same blockchain where the original license was created or executed.

The license-revocation transaction must point to (i.e. include the hash) of the original license. .

[4.4.](#) Cascading revocations

It is relevant to note that when Alice revokes access to Bob (the Requesting Party), Alice has the option to also revoke both the license contract with Carol (the owner of the Node #1 running the Client used by Bob) and the license contract with the node-operator of Carol's ON node (operator of Node #1).

In this scenario, Alice's contracts server must issue several license-revocation transactions on the blockchain to terminate these license contracts.

However, Alice has the option to retain the (long term) license contract with Carol (owner of the Client in Node #1) and the operator of Node #1. Alice may do this, for example, in anticipation of other future Requesting Parties using the same Client owned by Carol (e.g. the Client is an extremely popular social media client).

[5.](#) Design Issues and Challenges

There are a number of design issues and challenges with the decentralized service architecture, arising from the need to achieve the goals of (i) decentralization of services, (ii) portability of data and services, and (iii) automated service-provisioning through contracts. Some of these issues are discussed below (not an exhaustive list).

[5.1.](#) Instrumentation of nodes

One key question pertains to the instrumentation of ON nodes. A user (potential owner of an ON node) should be able to acquire an ON node with relative ease through the proper instrumentation tools that common today. A node acquisition contract should specify the tools and services available to the user to instrument and control their ON nodes.

Alternate models maybe explored, including the use of a manageability node that represents the user's launch-pad and control point for their entire network of ON nodes.

Hardjono

Expires September 26, 2018

[Page 18]

Internet-Draft

Decentralized OAuth

March 2018

[5.2.](#) Protection of private keys

An important aspect of systems employing public-key cryptography is the protection of the private-keys. Thus, Alice as the owner of a given ON node must protect the keys associated with all the UMA-services (plus the BC and CS) in her ON node.

[5.3.](#) Throughput of smart contracts agreement

Speed of processing smart contracts (e.g. license agreement between to CS servers) is an important aspect because the main UMA flow may be blocking or waiting for the contracts sub-flow to complete.

[5.4.](#) Moving ON nodes across providers

In order to achieve true independence from any given service provider, the owner of a given ON must be able to 'move' his or her ON node (together with all the internal relevant resources and configurations) to a new ON node at a new node-operator.

Several issues remain open, ranging from a standardized packaging format to the need to re-generate some identifiers for the UMA-services (i.e. new ON node is assigned a new ON unit public key by the new node-operator).

Her, there is a role for the Proxy/Forwarder Service in an ON node as

way to redirect requests to the new ON node while the old ON node is being shut-down.

[6.](#) IANA Considerations

TBD.

[7.](#) Security Considerations

This document pertains to a peer-to-peer infrastructure for resource sharing based on the UMA model using digital contracts and licenses. As such, there are numerous security aspects of deployments that need to be considered.

Aside from known traditional security challenges (e.g. channel binding to keys), there are new security questions that arise due to the proposed use of a peer-to-peer network of nodes as the basis for a decentralized service architecture. Some interesting issues include:

Proof of correct execution: One of the security challenges involves obtaining evidence that a node has not deviated from the agreed

behavior (as defined in a contract) and has not created side-effects (intentional or unintentional).

Proof of data erasure: For a node that acts as a resource server holding data belonging to the resource owner, there is a need to provide some mechanism that provides sufficient evidence that data erasure from the node has occurred. Such mechanisms would be useful to parties external to the node, but clearly does not address data copying (data theft) by the node.

Correct service definition: When contracts specify certain agreed services (both in node acquisition contracts and resource sharing contracts), there is the question of the correct service semantics being captured in the specified contract, both in the legal prose and within executable code.

[8.](#) Privacy Considerations

This document follows closely the UMA grant of OAuth2.0. The

authorization server at an ON comes to be in possession of data/resource information that may reveal information about the resource owner, which the authorization server's trust relationship with the resource server is assumed to accommodate.

The Client and Requesting Party are assumed to be a less-trusted. In fact, these entities are considered entirely untrustworthy until the data sharing contract has been established with the Resource Owner.

Following UMA grant of OAuth2.0, the primary privacy duty of the current decentralized design is to the Resource Owner. However, privacy considerations affect the Requesting Party as well. This can be seen in the issuance of an UMA related tokens, which represents the approval of a Requesting Party for a Client (ON) to engage with an Authorization Server to perform tasks needed for obtaining authorization, possibly including pushing claim tokens

9. Acknowledgements

We thank the following for feedback and inputs on technical and legal aspects (alphabetical last name): Justin Anderson (MIT), James Hazard (IACCM), Cameron Kerry (MIT), Eve Maler (ForgeRock), Alex Pentland (MIT), Tim Reiniger (Future Law).

10. References

Hardjono	Expires September 26, 2018	[Page 20]
----------	----------------------------	-----------

Internet-Draft	Decentralized OAuth	March 2018
----------------	---------------------	------------

10.1. Normative References

- [JSON] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", March 2014, <<https://tools.ietf.org/html/rfc7159>>.
- [JWE] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", May 2015, <<http://tools.ietf.org/html/rfc7516>>.
- [JWK] Jones, M., "JSON Web Key (JWK)", May 2015, <<http://tools.ietf.org/html/rfc7517>>.

- [JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", May 2015, <<http://tools.ietf.org/html/rfc7515>>.
- [JWT] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", May 2015, <<http://tools.ietf.org/html/rfc7519>>.
- [OAuth2.0] Hardt, D., "The OAuth 2.0 Authorization Framework", October 2012, <<http://tools.ietf.org/html/rfc6749>>.
- [OIDCCore] Sakimura, N., "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [UMA1.0] Hardjono, T., Maler, E., Machulak, M., and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", December 2015, <<https://docs.kantarainitiative.org/uma/rec-uma-core.html>>.
- [UMA2.0] Maler, E., Machulak, M., and J. Richer, "User-Managed Access (UMA) 2.0", January 2017, <<https://github.com/KantaraInitiative/wg-uma>>.

10.2. Informative References

- [Bitcoin] Nakamoto, S., "Bitcoin: a Peer to Peer Electronic Cash system", 2008, <<https://bitcoin.org/bitcoin.pdf>>.

Hardjono Expires September 26, 2018 [Page 21]

Internet-Draft Decentralized OAuth March 2018

- [BSC-DG] Hardjono, T. and E., Maler, "Kantara Blockchain and Smart Contract Discussion Group Report", January 2018, <<https://kantarainitiative.org/kantara-initiative-releases-first-blockchain-report-addressing-privacy-protection-and-personal-data/>>.

- [NRF] Norton Rose Fulbright, "Can Smart Contracts be Legally Binding Contracts", January 2018, <<http://www.nortonrosefulbright.com/knowledge/publications/144559/can-smart-contracts-be-legally-binding-contracts>>.
- [OIX] "OpenID Exchange", <<http://www.openidentityexchange.org>>.
- [OMS] Hardjono, T., Deegan, P., and J. Clippinger, "On the Design of Trustworthy Compute Frameworks for Self-organizing Digital Institutions (HCI 2014)", June 2014, <http://link.springer.com/chapter/10.1007/978-3-319-07632-4_33>.
- [OpenPDS] de Montjoye, Y., Wang, S., and A. Pentland, "openPDS: On the Trusted Use of Large-Scale Personal Data (IEEE Data Engineering)", December 2012, <<http://sites.computer.org/debull/A12dec/p5.pdf>>.
- [UMA-Legal] Reiniger, T., "A Proposed Licensing Model for User Managed Access", January 2018, <<https://kantarainitiative.org/confluence/display/uma/UMA+Legal>>.

Author's Address

Thomas Hardjono
MIT

Email: hardjono@mit.edu