Decentralized Service Architecture for OAuth2.0
draft-hardjono-oauth-decentralized-00

Abstract

This document proposes an alternative service architecture for user-centric control of the sharing of resources, 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. Each OAuth2.0 Nodes is assumed to have the capability to provide AS-services, RS-services and Client-services.

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 http://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 August 5, 2017.
1. Introduction ......................................................... 3
2. The OAuth2.0 Node .................................................. 4
   2.1. Node Definition ............................................. 4
   2.2. OAuth2.0 Services .......................................... 6
   2.3. ON Local Functions ......................................... 7
   2.4. Other OAuth2.0 Terminology ................................ 7
   2.5. Transaction Model .......................................... 8
   2.6. Exclusivity of Services .................................... 9
3. Contracts ............................................................ 9
   3.1. Definition .................................................. 9
   3.2. Types of Contracts ......................................... 10
   3.3. Service acquisition contracts: fields and parameters ... 10
   3.4. Data sharing contracts: fields and parameters .......... 11
4. Contracts and Blockchain Systems ................................. 13
5. Design Issues and Challenges ...................................... 14
   5.1. Support for subset of services ............................ 14
   5.2. Contracts expression language .............................. 14
   5.3. Contracts server ........................................... 15
   5.4. Contracts Access Token (CAT) ............................. 16
   5.5. Blockchain Client .......................................... 16
   5.6. Contracts Access Token (CAT) ............................. 17
   5.7. Public keys and binding to contracts ..................... 17
   5.8. End of Contract Actions .................................... 18
6. IANA Considerations ................................................ 18
7. Security Considerations .......................................... 18
8. Privacy Considerations ........................................... 19
9. Acknowledgements ................................................ 19
10. References .......................................................... 20
   10.1. Normative References ...................................... 20
   10.2. Informative References .................................... 21
Author’s Address ...................................................... 21
1. Introduction

Today the Identity Provider (IdP) role on the Internet has taken-on a centralized role, due to the predominance of the web single sign-on (Web SSO) model as the basis for the user interaction with services on the Internet. The underlying transaction model of Web SSO has elevated the identity provider to be the single source of trust between the user and the destination service or resource provider (e.g. merchant online). The identity provider has become the toll-gate that mediates the interaction between two transacting parties.

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 for the User Managed Access grant of OAuth (referred to as UMA2.0).

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 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 contracts as the legal mechanism to bind service providers:

- **Decentralization of services:** At the infrastructure level, decentralization of service means enabling a user to select the service providers that will provide the user full control over managing access to the user’s data, in particular for user-owned data (e.g. stored by user-controlled personal data stores). Here the service providers are entities that provide AS-services, RS-services and Client-services following the UMA2.0 model for user-centric sharing of data.

- **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. As such, portability of data stores and interoperability of services across providers is crucial to allow the user to retain independence from any specific service provider.

- **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 automated contract agreement model. Such an automated model enables users to switch providers rapidly without degradation in
control, privacy or sharing levels. Although untested technology, we propose a close alignment of our contracts model with that of "smart contracts" proposed by systems such as Corda [Corda] that employ distributed ledger technology (DLT) or blockchain systems.

The P2P 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.

Each node is assumed to have the capability to provide AS-services, RS-services and Client-services following the UMA2.0 context. Additionally, each node is assumed to also have the capability to provide authentication services for other nodes and for users, following the OpenID-Connect 1.0 model.

This document is agnostic as to how OAuth2.0 functions or services are implemented by node operators. A node operator may or may not implemented these services as executable bytecodes (smart contracts).

The decentralized architecture is not dependent on any blockchain system. Any contracts agreement mechanism can be deployed between parties, where both parties digitally sign agreed contracts.

In the following we describe in more details 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 implements the AS-services, RS-services and Client-services following the UMA2.0. Additionally, it implements Authentication Services following OIDC1.0. These services (functions) can be contracted to (leased) by a user to implement resource sharing. The OAuth2.0 Node also has additional infrastructure functions that are used for its own operations and cannot be contracted to by an external entity.

We distinguish between a node operator and a node owner. A node operator is the legal owner of the physical system implementing a given node. The node owner is contextual in nature and represents the contracted owner of a function or service at a node. Contractual ownership of a function or service can be exclusive or multi-
tenanted, although currently for simplicity we propose an exclusive service agreement

Possessing these complete services, a node operator can make available one or more of these services available depending on the context of the transaction, the entity it represents and the entity with whom it is interacting.

A user (Resource Owner or Requesting Party) obtains a given OAuth2.0 service (e.g. AS-service, RS-service, Client-service) from a given node by entering into a service acquisition contract with the node operator. The service acquisition contract makes the user the "owner" of that service at the node for the duration of the contract.

We distinguish between the node operator and the node owner:

- **Node Operator**: The legal owner of the physical system implementing an ON node.
- **Node Owner**: The (logical) owner of service at an ON node acquired for a duration of time under a contract with the node operator.

Contracts and digitally signed by the relevant parties and may be recorded to (or executed by) a blockchain system or distributed ledger system, although such actions must be independent from the legal status of contracts. The current service architecture is agnostic to the specific implementation of the underlying blockchain or distributed ledger system.

Once a resource owner acquires services from a node (e.g. RS-service), it can participate in transaction with a requesting party following the UMA grant of OAuth2.0. The requesting party may also acquire services (e.g. Client-service) from a node, with the purpose of using that service with a compatible service owned by a resource owner.

The public key associated with the service endpoint and the public key associated with the node operator (offering that service) provides a mechanism to detect and prevent conflicts of services. Here, the term ‘conflict’ is seen from the perspective of the user (resource owner or requesting party). It is used to mean a node that simultaneously contracts an RS-service and a Client-service to competing users. The acceptability of this configuration must be decided by the user prior to contracting any service from a node operator.

At the end of the duration of service acquisition contract, both the node and the user (Resource Owner or Requesting Party) must agree on
an asset-transferal mechanism in which the user’s relevant assets (e.g. data; keys; tokens; etc.) must be offloaded from the node to a location designated by the user, such as another node or offline storage. The default behavior of the node is to erase or flush the user’s relevant assets post-transferal and make available the same service to another potential customer (user).

Figure 1: OAuth2.0 Node (ON)

### 2.2. OAuth2.0 Services

The following are services that are implemented by an OAuth2.0 Node (ON) which can be contracted to by an external entity from the ON operator:

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 protected resources.

OpenID Provider  The OpenID Provider (OP) implements authentication of the Requesting Party and the Client. In the case of a Client, it performs authentication via proof of possession, either symmetric keys or asymmetric keys per RFC7800.

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 which cannot be contracted to by an external entity:

Blockchain Client  The Blockchain Client (BC) implements the client role in a blockchain system. This service cannot be contracted to by an external entity.

Contracts Server  The Contracts Server (CS) implements the contracts management and fulfilment for users and with other ON nodes. This service cannot be contracted to by an external entity.

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 (RqP) 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. Transaction Model

The transaction model follows closely that of the UMA grant of OAuth2.0 (also referred to as UMA2.0). Here a Requesting Party (Bob) is seeking access to resources or services offered by the Resource Owner (Alice) through a Resource Server that Alice controls.

The Requesting Party (Bob) selects an ON (e.g. Node #1) to be the Client in the transaction, while the Resource Owner selects 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:

- 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).

- 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 to authenticate himself or herself.
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 method of node selection is outside the scope of this document and will be the subject of future specifications.

2.6. Exclusivity of Services

For simplicity of design, we propose the exclusive ownership of services at all ON nodes for any given time. Furthermore, the node operator must associate a public key with the service endpoint, independent of the current ownership of that service by the user.

When a user (Resource Owner or Requesting Party) contracts a service offered by a node operator, that service is exclusively owned by (and under the full control of) the user throughout the duration of the contract. The node operator must not advertise otherwise, and potential users looking for services must verify (e.g. to a blockchain or contracts ledger) that a service point at a node is currently not under contract.

The long-term binding of a public key with the service endpoint by the node operator across differing owners allow other users to validate the ownership-status of as given service, and also allows for cryptographic level binding for transport security (e.g. TLS).

3. Contracts

Contracts form the basis for services acquisition and for data sharing. Services acquisition occurs between a user (Resource Owner or Requesting Party) and a node on the P2P network. Data sharing contracts occur between a Resource Owner and a Requesting Party, implemented through the services that each user contracts from ON operators.

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

A contract may or may not contain executable code (i.e. smart contract).

3.1. 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 optionally code or
pseudocode that captures the computing functions involved in services acquisition or data sharing.

The intent here is that users (Resource Owner or Requesting Party) would acquire services from nodes using bilateral contracts between the user and the node in an automated and semi-automated fashion, based on standardized templates of contracts.

Similarly, within a data sharing transaction a Requesting Party would acquire access to data under the control of the Resource Owner by using a 3-party contract. In this case, the contract must identify the Client node (under control of the Requesting Party), the Authorization Server node and the Resource Server node (both under the control of the Resource Owner).

3.2. Types of Contracts

We propose distinguishing two (2) types of contracts:

- Service acquisition contract: A Service acquisition contract denotes the acquisition of specific OAuth2.0 services by a user (Resource Owner or Requesting Party) from a given OAuth2.0 Node operator. Service acquisition contracts are typically bilateral between a user and a OAuth2.0 Node operator.

- Data Sharing contract: A data sharing contract denotes the granting of access by a Resource Owner to a data or resource to a Requesting Party using the services of the identified OAuth2.0 Node operator. Data sharing contracts typically involve up to five (5) entities: the Requesting Party, the Resource Owner, the operator of the Client ON, the operator of the Authorization Server ON and the operator of the Resource Server ON.

A data sharing contract presumes that entities involved have acquired relevant services from ON node operators through bilateral service acquisition contracts.

3.3. Service acquisition contracts: fields and parameters

The following information is needed within a service acquisition contracts:

- Type of contract
- Identifier of the user (Requesting Party or Resource Owner)
- Public key of the user
- Identifier of the ON operator
3.4. Data sharing contracts: fields and parameters

The following information is needed within a data sharing contract:

- Type of contract
- Requesting Party:
  - Identifier of the Requesting Party
  - Public key of the Requesting Party
- Client:
  - Identifier of the Client ON operator
  - Client service endpoints
* Public key of the Client ON operator
* Public key of the Client service at the ON

Resource Owner:
* Identifier of the Resource Owner
* Public key of the Resource Owner

Resource Server:
* Identifier of the Resource Server ON operator
* RS service endpoints
* Public key of the Resource Server ON operator
* Public key of the RS service endpoint at the ON

Authorization Server:
* Identifier of the Authorization Server ON operator
* AS service endpoints
* Public key of the Authorization Server ON operator
* Public key of the AS service endpoint at the ON

Duration of data sharing contract
End-of-contract actions
Service fees and payment mechanism (optional)
Dispute resolution method
Legal Prose
  * Data sharing terms and conditions
Code or pseudocode (optional)
Timestamp
Archive location of this contract (optional)
4. Contracts and Blockchain Systems

Blockchain technology offers interesting possibilities with regards to the notarization of contracts (for non-programmatic or "dumb" contracts) and with the regards to execution of code that may be embedded within programmatic contracts (smart contracts). In both cases, public key digital signatures represents a core function to provide source-authenticity of contracts.

The proposed decentralized service architecture based on the peer-to-peer computing paradigm is independent from any given blockchain system, and independent of the permissions of blockchains (permissioned or public).

In this context, we recognize two broad families of blockchains systems that are relevant to our transaction model or pattern:

- Application-specific blockchain system (non-programmatic): The blockchain is designed to perform specific tasks which cannot be extended or modified. Examples include the Bitcoin blockchain system designed for a digital currency application.

- Application-loadable blockchain system (programmatic): The blockchain support the notion of executable code running in one or more of the nodes of the blockchain, where the executable code can be programmed, loaded & offloaded, according to the intended application. The ledger function is integrated into the node, and as such records the results of the execution of the application. Such nodes may use virtual machine (VM) technology to achieve this effect. Examples include the Ethereum platform

For the purposes of contract adherence and fulfilment, both families of blockchains systems can provide assistance to OAuth2.0 Nodes (ON) in fulfilling the transaction model stated above (resource sharing from Alice to Bob). The assistance available to nodes corresponds to the blockchain system type being:

- Contract notarization and archiving: Entities who participate in service acquisition contracts or data sharing contracts can make use of a suitable non-programmatic blockchain system to keep a
(public) log of a signed contract. This can be done by storing a cryptographic hash of the contract within the transaction records of the ledger of the blockchain system. Systems such as the Bitcoin blockchain has some limited capacity to perform this function though the blockchain was not intended for use as a generic log for record keeping.

- Data sharing flow execution and notarization: Nodes that have the capability to receive, load and execute code found in contracts may choose to run parts or all of the service function (Client-service AS-service, or RS-service) within its virtual machine space instead of the general compute space of the node hardware.

- Dispute resolution: Disputes may occur even for completed and archived contracts. Blockchains offer the possibility of providing a common medium to capture evidence regarding a contract.

The proposed decentralized service architecture does not require OAuth2.0 Node (ON) operators to implement the OAuth2.0 services in the form of executable code running in one or more of the nodes of the blockchain.

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 are discussed below.

5.1. Support for subset of services

The proposed decentralized service architecture does not require OAuth2.0 Node (ON) operators to implement and make available all services. An operator is free to choose all or a subset of services.

The method used for an ON to advertise available services is outside the scope of the current document.

5.2. Contracts expression language

The current decentralized service architecture proposes the use of a JSON-based contract, which makes use the JOSE family cryptographic services for JSON (e.g. signing, encryption, key identification). See [JWS] and [JWE].
Such a JSON-based contract will be the subject of a future specification.

5.3. 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 OAuth2.0 Node (ON) regardless of whether the node implements all the OAuth2.0 services or only a subset of these services. The contracts server function cannot be leased-out to users or other entities, and represents an operational infrastructure component for an ON.

For a data sharing contract between a Requesting Party (employing a Client node) and the Resource Owner (employing an AS node), the contracts servers as the Client node and the AS node perform the relevant contracts-related tasks.

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

- Locate and validate templates of standard contracts (optional)
- Interact with peer contracts server at other nodes (data sharing contracts)
- Validate incoming contracts against standard template contracts
- Validate signatures on incoming contracts
- Sign contracts using the relevant private keys
- Record signed contract to blockchain (optional)
- Deliver the executable-code (found in the agreed contract) to the relevant underlying blockchain-related component on the node (optional).

Similar to endpoints in the UMA grant of OAuth2.0, the contracts server exposes a number of endpoints relevant to completing contracts agreement for service acquisition contracts and for data sharing contracts.

The same endpoints should be used by callers for both service acquisition contracts and data sharing contracts. The type of
contract being presented by the caller must be clear from the type-of-contract field.

5.4. Contracts Access Token (CAT)

A data sharing contract may come into existence as part of a Client’s access attempt to a data/resource located at a Resource Server. When the Client is redirected to the AS in order to obtain an access token from the AS, the Client (and Requesting Party) may be required to complete a data sharing contract through a separate flow (sub-flow) with the contracts server. In this case, the default contracts server is that belonging to (underlying) the AS.

The contracts server itself may be a protected resource, access to which requires a special access token from the AS. For ease of understanding, we refer to this token as the Contract Access Token (CAT). Note that the CAT is similar to the PAT token in the UMA context.

The Client must first obtain a CAT from the AS prior to engaging the contracts server. After obtaining a CAT from the AS, the Client (or more correctly the Client’s own contracts server) presents the CAT to the contracts server at the AS together with a proposed contract or contract template identifier.

The contracts server at the AS validates both the CAT and the proposed contract (or template), completes the necessary fields (e.g. the RS location, endpoint and public key), signs it and returns it to the Client. The Client must validate the received contract, counter-signs it and returns it to the contracts server at the AS.

After the contracts server at the AS validate the integrity and signature of the contract, it may optionally record it on the blockchain or distributed ledger.

The AS then issues the Client with the relevant access token (per UMA2.0 flows) to present to the RS. The access token specifies the resources available to the Client and Requesting Party following the agreed contract. It may optionally include pointers to the contract, meaningful only to the AS (i.e. in the case of token validation).

5.5. Blockchain Client

The Blockchain Client (BC) 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 blockchain client interacts with the relevant underlying blockchain systems or distributed ledger system shared between the Requesting Party and the Resource Owner.

The blockchain client is used by the contract server within an OAuth2.0 Node to transmit blockchain-transaction to the target blockchain system. It is also used by the node to validate the existence of a given previous transaction on the blockchain.

The blockchain client may also be used for payments related to service acquisition contracts (e.g. between a user and an ON operator) and to data sharing contracts (e.g. Requesting Party paying Resource Owners).

5.6. Contracts Access Token (CAT)

The current decentralized service architecture proposes the use of a JSON-based contract, which makes use the JOSE family cryptographic services for JSON (e.g. signing, encryption, key identification).

Such a JSON-based contract will be the subject of a future specification.

5.7. Public keys and binding to contracts

Public key cryptography plays an important role in contracts due to its ability to provide technical-trust (through proof-of-possession and strength of the chosen public-key cryptosystem) and to provide legal-trust through the legal acceptance of digital-signatures as legal signatures (i.e. Digital Signature Act).

Node operators must possess the following public key pairs in order to support the proposed decentralized model:

- Operator-level public key: A node operator must possess an operator level public key that it uses to legally sign contracts. This public key must be long-term and be legally associated/owned by the operator (e.g. through a CA-issued certificate).

- Service endpoint public keys: For each OAuth2.0 service supported, the node operator must associate and bind a unique long-term public key to that service

Public key pairs are used in this architecture for the following:

- Contracts signing: the public-key pair is used to digitally signed a contract by entities involved in a service acquisition contract or data sharing contract.
o Service endpoint authentication: the public-key pair is used to authenticate a service end-point and to cryptographically bind the secure channel to the session between the caller and the service end-point (e.g. see RFC5929 and RFC5056)

o Access to blockchain: the public-key pair of a service endpoint can used to submit a transaction to a blockchain or distributed ledger system, where the system accepts the transaction as source-authenticated.

o Contracts execution (smart contracts): the public-key pair is used by the a blockchain or distributed ledger system to execute the code (e.g. bytecode) found within a contract and where the public-key is invoked by the executing code

5.8. End of Contract Actions

At the end of a contract period, an OAuth2.0 Node (ON) must perform post-contract action as specified in the contract (service acquisition contract and data sharing contract). These post-contract action may or may not be recorded in some manner through a blockchain to provide evidence of its own completion.

Examples of end-of-contract actions include transferring data to another repository (e.g. to another RS), flushing all user data cached at the node, claiming payments (e.g. at a payments escrow party), and so on.

The end-of-contract actions will be the subject of a future specification.

6. IANA Considerations

TBD.

7. Security Considerations

This document pertains to a peer-to-peer infrastructure for data sharing based on digital contracts. As such, there are numerous security aspect 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 act 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 service acquisition contracts and data sharing contracts), there is the question of the correct service semantics being captured in the contract, both in the legal prose and within executable code (or pseudocode).

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

The following people made significant contributions to the contents of this document:

- Christian Smith (MIT)
- Dmitri Zagidulin (MIT)
- Greg Linklater (Rhodes University)
10. References

10.1. Normative References


10.2. Informative References


Author’s Address

Thomas Hardjono (editor)
MIT

Email: hardjono@mit.edu