Saranyu: Using Smart Contracts and Blockchain for Cloud Tenant Management

Sambit Nayak, Nanjangud C Narendra, Anshu Shukla
Ericsson Research Bangalore, Bangalore, India
{sambit.nayak,nanjangud.narendra,anshu.shukla}@ericsson.com

James Kempf
Ericsson Research, Santa Clara, USA
james.kempf@ericsson.com

***Abstract*—While blockchains and smart contracts are primarily known for their use as the technologies underlying cryptocurrencies like Bitcoin and Ethereum, these technologies also have applicability in other areas. In this paper, we present an application of smart contracts running on a permissioned distributed ledger (essentially a private blockchain where only credentialed participants are allowed to read and write) to managing tenant and service accounts in a cloud computing data center. The system is called *Saranyu* and it supports four services: identity management, authentication, authorization, and charging. Identity management and authentication are handled using client-generated public/private key pairs in the usual fashion. Authorization is handled as a contract from a grantee of service access rights to a recipient which is a tenant or other service. Charging is supported by *Saranyu* integration with payment gateways for payments related to service resource usage. We describe an implementation of *Saranyu* on top of the Quorum blockchain system. We believe the security, non-repudiation, tamper-resistance, and easy transaction history access brought by blockchain technology will increase transparency and trust in cloud tenant and service management and that the fundamentally distributed nature of the blockchain will make *Saranyu* an excellent match with developing distributed cloud architectures.**

I. Introduction and Motivation

Blockchains and smart contracts are primarily known for their use as the underlying implementation technologies for such high profile services as Bitcoin [2] and Ethereum [3]. Yet these technologies are even more valuable as solutions to problems involved in tracking and accounting for goods and services across multiple parties in business ecosystems, due to their security, non-repudiation and tamper-resistance. Smart contracts enforce precise adherence to pre-specified contract terms and conditions due to their computer-readable nature - essentially automated code execution in a secure cryptographically protected tamper-resistant environment. This makes smart contracts useful for ensuring rigorously specifiable and enforceable business transactions.

The management of tenants and services in a cloud computing data center is an example of an application where the current state of the art involves an in-house solution deployed and managed by the cloud service provider. For example, AWS provides features like IAM [4] and services like Cognito [5] which handle identity management and authentication for access to services deployed in AWS cloud data centers. AWS IAM additionally handles authorization and roles for access to AWS supplied services. AWS MarketPlace [6] allows customers looking for specific software packages to buy access to AMI instances, with Amazon handling the metering, charging, and billing. Yet all of these solutions require the third party service provider to trust the cloud provider's centralized database to do proper accounting and to handle dispute resolution in a transparent fashion. Extending tenant and service management to a distributed cloud platform where the platform may involve multiple parties with competing interests and varying degrees of trust brings additional complexities.

In this paper, we present our system under development, *Saranyu*, which utilizes smart contracts to manage tenant and service accounts and resource usage in a cloud computing data center. *Saranyu* provides four different services: identity management, authentication, authorization for service resource usage, and charging. To the best of our knowledge, we believe that this is the first application of smart contracts to account management for cloud tenants and services.

II. *Saranyu* Architecture

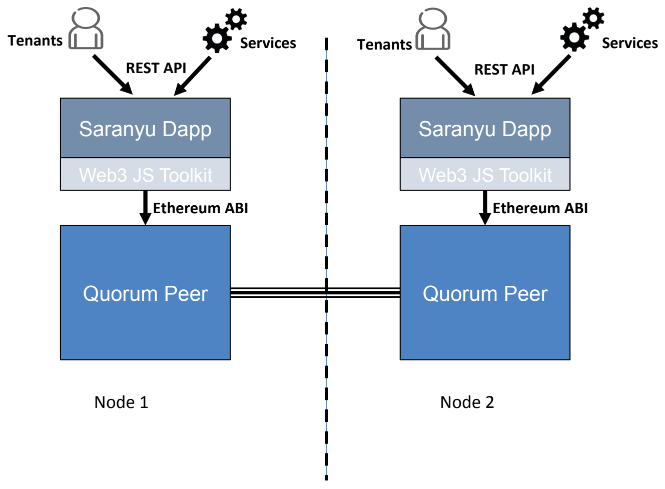

Fig. 1. *Saranyu* Architecture

The architecture of *Saranyu* is depicted in Figure 1. *Saranyu* is built on the open source Quorum [7] distributed ledger and smart contract platform, Quorum being a permissioned implementation of Ethereum. Quorum smart contracts are

The authors wish to thank Joacim Halen, Tomas Mecklin and Wolfgang John for their feedback.

defined in the Solidity smart contract language [9], the same language as used in the Ethereum public blockchain. Just like Ethereum, Quorum has two types of accounts - externally owned accounts and smart contract accounts. Externally owned accounts correspond to an external actor with a public/private key pair. Smart contract accounts hold associated smart contract code in the form of bytecode conforming to the Ethereum ABI [10] that can be executed on an Ethereum Virtual Machine (EVM) implementation [3].

Saranyu functions as a *Dapp* - that is, a blockchain-based, distributed application - on top of Quorum. It accesses the blockchain and interacts with Solidity smart contract objects on it using the Web3 JavaScript (JS) library [11], which in turn uses the Ethereum JSON-RPC interface that a Quorum node exposes, just like an application on the public Ethereum blockchain does. The Web3 JS library translates the return values from Solidity contract functions back into JS, handles exceptions and listens for events from Solidity code. While most existing Dapps like the Ethereum wallet run in a browser on a user's laptop or mobile device, *Saranyu* instead runs on a server in the data center. Conceptually, each server in a particular server cluster (rack, rack row, etc.) has an instance of the *Saranyu* Dapp as well as a Quorum instance running on it, thus forming the blockchain and the tenant/service management platform.

In *Saranyu*, services bundle together resources and other services and offer them as service attributes to tenants through a *rights grant* which is the delegation of the right to use certain service attributes for a specified period of time, charged for at a specific rate, granted by a grantor and received by a recipient. The rights grant is signed by the keys of both parties and is maintained on the blockchain. It constitutes the authorization for the tenant to use the specified service attributes for the period of the grant. Typically, the rights grant will also include a quota, a specification of how much the tenant may use of the service attribute, e.g., number of CPU seconds offered by a compute service.

Tenants establish accounts with *Saranyu*, but services must also register accounts. Accounts for tenants and services are identified by the public key part of a public/private key pair. Services registered with *Saranyu* can also build on other registered services, and the developers can receive payment when those services are used.

III. Tenant and Service Specification and Management

Tenants and services in *Saranyu* are modeled via contract accounts. For tenants, each account specifies the identity of the tenant (the tenant's public key and contact information like email address) and a charging credential for settlement. The account also records and stores the rights delegations that have been granted to the tenant. *Saranyu* performs authentication on each tenant that logs in, via the tenant's private/public key pair. Service contract accounts are similar to tenant accounts, except a service may optionally be assigned to a tenant for charging settlement rather than an outside payment credential.

A. Saranyu Manager

Listing 1 contains the API for tenant enrolment and service registration in the top-level system management contract object, `SaranyuManager`. This is the entry point for enrolling tenants and registering services and maintaining their accounts.

`SaranyuManager` maintains internal data structures for mapping tenant and service identifiers to their respective objects. The key for identifying tenant and service entities is the 256-bit ECC public key [13] (`pck`). This is provided at the time the entity is registered. For tenants, the `enroll()` and `update()` functions take the public key along with other contact information, like the tenant's name (`name`), email address (`email`) and cell phone number (`phone`). These functions also take an address object bound to a Solidity object of type `Credential` for charging to a back end settlement processor, like a credit card company. The `destroy_-tenant()` function calls the `destroy()` function on the Tenant contract object and zeros out the entry in the tenant's mapping, although the tenant record is not removed from the blockchain since records in the blockchain are immutable. The `is_tenant()` function checks if a tenant is enrolled with a given key. The `SaranyuManager` has corresponding functions for registration and unregistration of services.

```
contract SaranyuManager {
  //Tenant Account Management
  function enroll(
    bytes32 pck, string name,
    address credential,
    string email, string phone)
    external { }
  function update(
    bytes32 pck, string name,
    address credential,
    string email, string phone)
    external { }
  function destroy_tenant(bytes32 pck)
    external { }
  function is_tenant(bytes32 pck)
    public constant
    returns (bool) { }
  //Service Account Management  //
  function register_service(
    bytes32 pck, string name,
    address credential, //Credential object
    string owner_email)
    external { }
  function register_service(
    bytes32 pck, string name,
    address tenant) //Tenant object
    external { }
  function unregister_service(bytes32 pck)
    external { }
  function is_service(bytes32 pck)
    public constant
    returns (bool) {}

  // --- continued ---
}
```

Listing 1. *Saranyu* Manager - Tenants and Services

Listing 2 shows the API for managing rights delegations. The `SaranyuManager` contract API contains functions to register and revoke rights delegations as `Delegation` contracts. The `subdel` parameter indicates the number of allowed subdelegations. The issue parameter indicates the date and time of issue, while the expiry parameter indicates when the delegation expires. The revokers array contains the public keys of authorized delegation revokers. Functions for determining if an object is a delegation and for suspending or revoking a delegation are also included.

```
contract SaranyuManager {

  // --- continued ---

  function register_delegation(
    bytes32 grantor,
    string rights_delegation,
    bytes32 recipient,
    uint8 subdel,
    uint256 issue,
    uint256 expiry,
    address[] source_delegations,
    bytes32[] revokers) { }
  function is_delegation(
    address d,
    bytes32 pck) constant returns (bool) { }
  function suspend_delegation(address d) { }
  function revoke_delegation(address d) { }

  event DelegationSuspended(
    bytes32 indexed grantor,
    bytes32 indexed recipient,
    address indexed delegation);
  event DelegationRevoked(
    bytes32 indexed grantor,
    bytes32 indexed recipient,
    address indexed delegation);
}
```

Listing 2. *Saranyu* Manager - Delegations API

B. Tenant and Service

The Tenant contract API shown in Listing 3 defines the basic contract between the tenant and the cloud provider, and manages the collection of rights grants that the tenant has received.

```
contract Tenant {
  function Tenant(
    bytes32 pck, string name,
    address cred, //Credential object
    string email, string phone) { }
  function get_info() constant returns(
    bytes32, string,
    address, //Credential object
    string, string) { }
  function is_delegation(address d) constant
    returns(bool) { }
  function put_delegation(address d) { }
  function update(
    string name, address cred,
    string email, string phone) { }
  function destroy() { }

}
```

Listing 3. Tenant Contract

The get_info function returns the tenant's information, while the update function updates it. The put_delegation function adds a delegation to the tenant's list of delegations. The destructor destroy() calls the internal Solidity function selfdestruct() after destroying all the Delegation objects the tenant has received.

A service is also represented in *Saranyu* using a smart contract Service object. Service accounts are, in principle, little different than tenant accounts, and their API is omitted here due to lack of space. They contain internal data structures representing the resource types they offer, their service attributes and quotas as applicable.

IV. Authorization Delegation

Saranyu represents a tenant's or service's rights delegation as a smart contract of type Delegation, including any quotas specifying the maximum amount of the service attributes that can be consumed by the recipient, charging schedules for the service attributes, and revokers that are allowed to suspend or revoke the delegation. Delegations can also be chained, so a recipient in *Saranyu* like a tenant can further delegate these rights to sub-tenants. An example of this would be a corporate tenant delegating access rights to employees. With the exception of *Saranyu* itself, services cannot sub-delegate access rights that they receive from another service.

A. Delegation Contract

```
contract Delegation {
  function Delegation(
    bytes32 grantor,
    string service_contract,
    bytes32 recipient,
    uint8 subdel,
    uint256 issue,
    uint256 expiry,
    address[] source_delegations,
    bytes32[] revokers) { }
  function isValid() constant returns (
    bool valid) { }
  function isExpired() constant returns (
    bool expired) { }
  function isRevoked() constant returns (
    bool revoked) { }
  function isSuspended() constant returns (
    bool revoked) { }
  function suspend() { }
  function revoke() { }
  function expire() { }
  function check_chain() constant returns (
    bool allowed) { }
}
```

Listing 4. Delegation Contract

The Delegation contract in Listing 4 shows the Solidity API for an authorization delegation. The constructor has parameters grantor and recipient for the registered public keys of the entity granting and receiving the delegated rights. The constructor also has parameters issue and expiry for the time the delegation was issued and when it expires. The grantor can always suspend a delegation and a delegation is implicitly suspended when it expires. Finally, the parameters subdel and revokers in the constructor indicate the number of allowed sub-delegations and the public keys of the authorized delegation revokers.

An example of a service contract is one which specifies a grant from the ComputeService of rights for up to 5 VMs of type *vm_type1* with a maximum per VM memory of 4 GB. The charging of VM CPU time in hours is done every 12 hours at the rate of 0.05 US Dollars per hour.

Figure 2 illustrates how delegation works. Delegations form a chain grounded in the resources provided by the hardware infrastructure. In the figure, a solid arrow indicates a delegation of a service attribute offering, while a dashed arrow indicates a subdelegation. Services S2 and S3 receive delegations of service attributes from S1, which they then utilize to offer as their attributes to other services and tenants. Tenants S1-T1 and S1-T2 receive delegations directly from S1. JoesGarage receives a delegation from S2, which it then subdelegates to its employees, J-T1 through J-T4. S2-T1 and S3-T1 receive delegations directly from the S2 and S3 respectively. Note that subdelegation by tenants directly to services is prohibited.

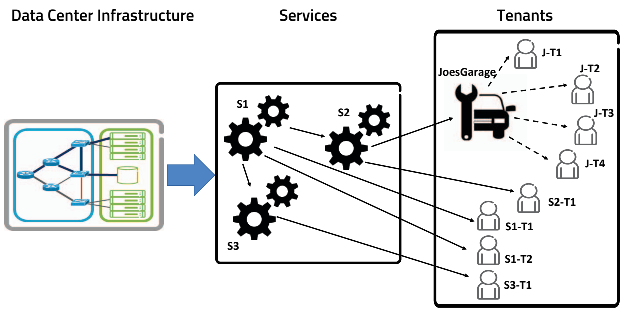

Fig. 2. Tenant and Service Name Space Delegation Lattice Example

B. Authorization Request and Grant

A tenant receives delegations when it subscribes to a service. The subscription can take place at the time the tenant first establishes an account with the cloud provider, as is the case for basic compute, networking and storage services, or at a later date, as with third party services such as streaming media, social networking, etc. They can also update their subscription with a new rights delegaton. *Saranyu* records a collection of rights grant offers from the service signed with the service's private key when the service first registers. The offers are structured by tenant type, so, for example, offers for enterprise customers may be different than for individuals. *Saranyu* records the signed rights grant delegation to the tenant on the blockchain (the rights are signed by both *Saranyu* and the tenant, which constitutes auditable proof), along with revokers and other information. The delegation grant is only allowed if all source `Delegation` contracts in the chain are still valid, which requires validating the entire chain of delegations each time a delegation is granted. *Saranyu* then sends the accepted rights grant delegation to the service, establishing the new subscription. Finally, *Saranyu* reports back to the tenant that the subscription is complete, and the tenant is free to use the service. The service is responsible for monitoring tenant usage of service attributes and limiting usage if the tenant tries to use more attributes than their rights grant allows. This avoids a time-consuming lookup on the blockchain while a tenant is waiting to have something done. The service must also provide the tenant with any required authorization tokens. A similar message sequence holds for services requesting rights grants.

C. Authorization Grant Suspension and Revocation

The revokers of a delegation are those grantors and recipients that are above the recipient in question in the delegation chain. There are two situations in which an authorization delegation can be suspended or revoked:

- Expiration of the validity time period,
- Through intervention of a revoker as indicated in the delegation chain, due to some extraordinary event, such as failure of a charging transaction.

Saranyu will normally suspend an entity's rights grant before revoking it completely. This allows the tenant or service owner an opportunity to pursue a restoration procedure with the service provider prior to losing rights to the grant entirely. A request comes to *Saranyu* from an authorized revoker (i.e. from an entity with a `Revoker` contract in the delegation chain) to suspend the delegation. *Saranyu* passes the request on to the service, which suspends the tenant's access to the service (for example suspending VMs or locking the tenant out of access to their storage volumes) and marks its cached copy of the rights delegation as Suspended. The service returns the results of the operation to *Saranyu* and, if the operation is successful, *Saranyu* marks the tenant's rights grant as suspended on the blockchain, returning the result to the authorized revoker.

Revocation occurs if the tenant or service owner does not pursue a restoration action within a specified period of time or if a tenant or service owner deletes their account with *Saranyu*. Full revocation of a rights grant results in the transfer of the rights back to the original grantor of the delegation. This could be a service or a tenant that has sub-delegated rights.

In generic terms, say the grantor of a delegation D is at level K in the delegation chain (i.e., K levels below the original grantor), and the recipient is at level K+M in the chain (M >0). When the delegation D is suspended, all delegations at level K+M and below in the delegation chain of D will be suspended. If the grant is revoked, grants in D reverts back to the grantor of D at level K+M-1. If that grantor sourced such rights grants of D from any delegations that it itself possesses, then those delegations would have been the source delegations of D in the chain, and thus this effectively amounts to the corresponding rights grants of D getting back to those source delegations.

V. Charging

The fourth service *Saranyu* provides is charging for usage of rights grants. A variety of charging credentials can be provided by tenants or services when they set up their accounts, and settlement proceeds through a backend payment processor. Cryptocurrency can be one of the settlement means but *Saranyu* itself does not support cryptocurrency. The following

Solidity *mixins* provide common functionality involved in charging:

- `Usage`: tracks usage of a service attribute on the blockchain. *Saranyu* obtains usage information from a metrics system, since it is the responsibility of the service owning the service attribute being charged for to report usage,
- `Credential`: the tenant or service charging credentials. Subclasses contain more detailed credential information, and any private information should be encrypted,
- `Chargable`: records a charge to a particular `Credential` for a specified `Usage` on the blockchain.

Further details of charging are omitted due to lack of space, and will be reported in a future paper.

VI. Related Work

Saranyu's rights delegation architecture was inspired by the Berkeley WAVE IoT rights delegation work [14]. WAVE is an IoT identity management, authentication, and authorization service defined on the public Ethereum blockchain for allowing rights delegation of access to IoT devices. *Saranyu's* tenant and service management system is different from WAVE in that a permissioned ledger is required rather than an unpermissioned ledger.

The emerging trend of blockchain based *Dapps* has led to cloud providers such as Microsoft Azure delivering Blockchain as a Service (BaaS) [17]. Similarly the IBM Watson IoT platform [18] provides the open source smart contract system Hyperledger Fabric [19] to create, test, and deploy smart contracts.

Another problem that has motivated our work is how to reduce hardware costs and software licenses/maintenance costs on cloud using multi-tenancy [20]. If services deployed in the cloud can all share the same tenant and service management system as in *Saranyu*, the cost of deploying a service is reduced and the service provider can pass the savings along to their customers.

VII. Discussion and Conclusions

In this paper, we have presented *Saranyu*, a cloud tenant and service management system based on smart contracts recorded on a permissioned, distributed ledger (i.e., a blockchain). Key features of the system are the ability to craft custom smart contracts for particular classes of users and types of payment credentials and to delegate authorization for the use of service attributes to tenants, sub-tenants and other services. In addition, the distributed nature of the underlying blockchain system, in our case Quorum, allows for increased scalability and resilience, and the participation of multiple parties in complex business ecosystems. We have described the design of *Saranyu* and provided some sample Solidity APIs for smart contract specifications. In future work we plan to investigate the performance of the blockchain in larger scale cases with many tenants and services, and to enhance *Saranyu* for more complex cloud service provisioning, such as service composition provisioning in the cloud [22].

References

[1] Diego Ongaro and John K Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pages 305–319, 2014.

[2] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[3] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum Project Yellow Paper*, vol. 151, 2014.

[4] “Identity and Access Management (IAM) - Amazon Web Services (AWS),” https://aws.amazon.com/iam, accessed: 2018-1-8.

[5] “Cognito - Simple and User Sign Up & Sign In,” https://aws.amazon.com/cognito, accessed: 2018-1-8.

[6] “AWS — Amazon Marketplace Management Portal - A Sales Channel for Cloud Software,” https://aws.amazon.com/marketplace/management/tour/, accessed: 2018-1-24.

[7] “Quorum — J.P. Morgan,” https://www.jpmorgan.com/quorum, accessed: 2017-12-28.

[8] “Home jpmorganchase/quorum Wiki GitHub,” https://github.com/jpmorganchase/quorum/wiki, accessed: 2017-12-28.

[9] “Introduction to smart contracts,” http://solidity.readthedocs.io/en/develop/introduction-to-smart-contracts.html, accessed: 2018-1-7.

[10] “Ethereum Contract ABI - ethereum/wiki - Github,” https://github.com/ethereum/wiki/wiki/Ethereum-Contract-ABI, accessed: 2018-1-9.

[11] “GitHub - ethereum/web3.js:Ethereum JavaScript API,” https://github.com/ethereum/web3.js, accessed:2018-1-9.

[12] “JSON Web Token Introduction - jwt.io,” https://jwt.io/introduction/, accessed: 2017-12-28.

[13] H. Mayar, “ECDSA Security in Bitcoin and Ethereum: a Research Survey,” https://blog.coinfabrik.com/wp-content/uploads/2016/06/ECDSA-Security-in-Bitcoin-and-Ethereum-a-Research-Survey.pdf, accessed:2018-1-9.

[14] M.P. Andersen, J. Kolb, K. Chen, D.E. Culler, and R. Katz. “Democratizing Authority in the Built Environment,” in *Proceedings of BuildSys*, November 8-9, 2017, Delft, The Netherlands.

[15] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin, “Formal verification of smart contracts: Short paper,” in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, ser. PLAS ’16. New York, NY, USA: ACM, 2016, pp. 91–96. [Online]. Available: http://doi.acm.org/10.1145/2993600.2993611

[16] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16. New York, NY, USA: ACM, 2016, pp. 254–269. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978309

[17] “Introducing Project ”Bletchley”,” https://github.com/Azure/azure-blockchain-projects/blob/master/bletchley/bletchley-whitepaper.md, accessed: 2018-1-7.

[18] “The basic smart contract sample,” https://github.com/ibm-watson-iot/blockchain-samples/tree/master/contracts/basic/simple_contract, accessed: 2018-1-7.

[19] “Welcome to Hyperledger Fabric,” https://hyperledger-fabric.readthedocs.io/en/release, accessed:2018-1-13.

[20] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle, “Multi-tenant SOA Middleware for Cloud Computing,” in *Cloud computing (cloud), 2010 ieee 3rd international conference on*. IEEE, 2010, pp. 458–465.

[21] “Blockchain Decision Path. Chapter 3: The Promise of Blockchain Technologies,” https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS171x+3T2017/courseware/5f4fa9501e284f4ebebbc00085699e27/6d82cf9ccfe742cbba395953d05ae771/?activate_block_id=block-v1\%3ALinuxFoundationX\%2BLFS171x\%2B3T2017\%2Btype\%40sequential\%2Bblock\%406d82cf9ccfe742cbba395953d05ae771, accessed:2017-12-11.

[22] L. Ramachandran, N. C. Narendra, and K. Ponnalagu, “Dynamic provisioning in multi-tenant service clouds,” *Service Oriented Computing and Applications*, pp. 1–20, 2012.